Universidad ORT Uruguay

Facultad de Ingeniería

Whizdy

Sistema de Scouting para adquisición de talentos impulsada por Inteligencia Artificial

Entregado como requisito para la obtención del título de Ingeniero en Sistemas

Martín Caffarena - 230351

Franco Daneri - 260284

Nicolás González - 231843

Tutor: Bruno Ferrari

2025

Declaración de autoría

Nosotros, Martín Caffarena, Franco Daneri y Nicolás González, declaramos que el trabajo que se presenta en esta obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizamos el Proyecto Final de carrera de Ingeniería en Sistemas;

- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;

- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;

- En la obra, hemos acusado recibo de las ayudas recibidas;

- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente que fue construido por otros, y que fue construido por nosotros;

- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Martín Caffarena	Franco Daneri	Nicolás González
16-10-2024	16-10-2024	16-10-2024

Agradecimientos

Queremos dedicar este espacio para poder agradecer a todos aquellos, los cuales han sido parte de este proyecto con cualquier tipo de apoyo.

En primer lugar, agradecer a nuestros familiares, amigos y parejas, los cuales creemos que su apoyo y soporte emocional ha sido fundamental tanto durante el transcurso del proyecto como de la carrera en sí misma.

Por otro lado agradecemos a Bruno Ferrari, nuestro tutor, por habernos apoyado en todo momento y poder guiarnos a lo largo del proyecto, siempre estando a disposición, motivándonos para lograr mejores resultados y compartiendo su conocimiento a pleno. Su apoyo ha sido uno de los pilares de este proyecto.

Agradecer también a GoGrow permitiéndonos ser parte de Whizdy, compartiendo toda información sobre este y estando a disposición de compartir su conocimiento y recursos en todo momento.

Por último, gracias a todos los profesores con los que nos hemos cruzado durante el proyecto y a lo largo de la carrera. Pudiendo aportar todo tipo de aspecto al proyecto en sí y a nuestra formación académica, que esperemos dé sus frutos en un futuro como profesionales.

Muchas gracias a todos ellos.

Franco, Martín y Nicolás

Abstract

Los procesos actuales de selección de personal de las empresas enfrentan desafíos críticos: alta dependencia de tareas manuales, evaluaciones subjetivas, falta de trazabilidad en las decisiones y escasa retroalimentación hacia los candidatos.

Para atacar estos problemas es que surge Whizdy, una plataforma *SaaS* de *scouting* y selección de talento impulsada por inteligencia artificial. El sistema transforma el proceso tradicional en uno explicable, medible e informativo, integrando un motor de *matching* semántico basado en *embeddings*, modelos de lenguaje y bases de datos vectoriales.

La solución se implementó mediante una arquitectura híbrida que combina componentes internos desarrollados específicamente para el proyecto con servicios externos especializados en inteligencia artificial y almacenamiento de datos. Esta arquitectura modular permite integrar capacidades avanzadas de análisis semántico y generación de lenguaje natural sin requerir desarrollo propio de modelos de IA.

El proyecto fue gestionado bajo una metodología ágil adaptada, con iteraciones quincenales, validaciones continuas con el cliente y documentación reflexiva.

Se entregó una primera versión funcional validada con usuarios reales, que incluye flujos completos para candidatos y empresas para un proceso robusto de selección de personal, sistema de notificaciones, autenticación segura, *dashboard* analítico y generación de explicaciones de *matching* mediante IA.

Palabras clave

Whizdy; Inteligencia artificial; Selección de personal; *Scouting*; *Embeddings*; *Prompting*; Metodologías ágiles; *Qdrant*; Candidatos; *AWS*; *PoC*; Vectores; *Next.js*; *Python*

Glosario

API: Interfaz que define cómo se comunican aplicaciones mediante solicitudes y respuestas estandarizadas.

App Router: Sistema de enrutado de *Next.js* basado en la carpeta app/ con rutas, *layouts* y *loaders* por segmento. [6]

Autenticación: Proceso de verificar la identidad de un usuario o servicio.

Autorización: Permisos que determinan qué acciones o recursos puede usar un usuario autenticado.

AWS: Plataforma de servicios en la nube de Amazon para cómputo, almacenamiento, redes y más.

AWS Amplify: Herramientas y *hosting* de *AWS* para crear, conectar y desplegar apps web y móviles rápidamente. [27]

AWS Secrets Manager: Servicio para almacenar y rotar credenciales y secretos de forma segura. [27]

Backlog: Lista priorizada de todo el trabajo pendiente del producto. [3]

Backlog grooming: Proceso de refinar, aclarar y priorizar ítems del *backlog* (también llamado "*refinement*").

Backend: Lógica interna de una aplicación o sistema.

Base vectorial: Base de datos que almacena y organiza datos como vectores. [14]

Bug: Defecto o error en el *software* que provoca un comportamiento no deseado.

CI/CD: Integración y entrega/despliegue continuos para automatizar pruebas y *releases*.

Clockify: Herramienta para registrar y reportar horas de trabajo por tarea/proyecto.

Commit: Registro de cambios en un repositorio con un identificador y mensaje asociado. [34]

Connection pooling: Reutilización de conexiones a base de datos en un pool para reducir latencia y costos.

Demo: Presentación del incremento de producto para mostrar valor y recibir *feedback*.

Docker: Plataforma de contenedores para empaquetar y ejecutar aplicaciones de forma aislada y portable.

EC2: Servidores virtuales de *AWS* para ejecutar aplicaciones con control del sistema operativo y recursos. [27]

Embedding: Representación numérica en forma de vector que captura el significado de texto o imágenes. [7]

Endpoint: URL específica de una API que expone una operación o recurso.

Entregabilidad: La capacidad de un *software* de ser entregado exitosamente a sus usuarios.

Épica: Gran objetivo o iniciativa que se descompone en múltiples historias de usuario.

Figma: Herramienta colaborativa para diseño de interfaces y prototipos.

Fine tuning: Ajuste de un modelo pre entrenado con datos específicos para una tarea. [11]

Frameworks: Marco de trabajo que proporciona una estructura, herramientas y directrices estandarizadas para simplificar y acelerar el desarrollo de *software* u otros proyectos

Free tier: Cuota gratuita con límites de uso que ofrecen algunos servicios en la nube.

Frontend: Parte de una aplicación web con la que los usuarios interactúan directamente, abarcando la interfaz visual y la interactividad de un sitio o aplicación.

Fullstack: Desarrollo que abarca *frontend* y *backend* de una aplicación.

Gemini: Asistente de inteligencia artificial de *Google*.

Git: Sistema distribuido de control de versiones para gestionar cambios en el código. [34]

Git-flow: Estrategia de ramas con main, *develop*, *features*, *releases* y *hotfixes* para utilizar mediante *git*. [28]

GitHub: Plataforma para alojar repositorios Git con colaboración, issues y PRs. [34]

GitHub Actions: *CI/CD* nativo de *GitHub* para automatizar *builds*, *tests* y despliegues.

Hard Skills: Habilidades técnicas y medibles (programación, *SQL*, estadística, etc.).

Higher-Order Functions: Funciones que reciben otras funciones como argumento o las retornan.

HTTP: Protocolo de comunicación de la web para solicitudes y respuestas entre cliente y servidor.

IAM: Servicio de *AWS* para gestionar usuarios, roles y permisos de acceso a recursos. [27]

IntelliSense: Autocompletado y ayuda contextual en editores de código como Visual Studio Code.

JSON: Formato ligero de intercambio de datos basado en pares clave–valor. [30]

JSON Web Token: estándar abierto que permite transmitir información de forma segura entre dos partes, como un cliente y un servidor, mediante un objeto *JSON*. [30]

JWT: Sigla de *JSON Web Token*. [30]

Lambda: Servicio serverless de *AWS* para ejecutar código bajo demanda sin gestionar servidores.

Layouts: Plantillas persistentes por sección/ruta que comparten UI en el *App Router* de Next.js. [6]

Matching: Cálculo de compatibilidad entre dos entidades.

Merge: Integración de cambios de una rama a otra dentro de un repositorio Git. [34]

Microservicios: Arquitectura que divide el sistema en servicios pequeños, independientes y desplegables por separado.

Middleware: Capa intermedia que intercepta y transforma requests/responses en una app.

MVP: Producto mínimo viable para validar hipótesis con el menor esfuerzo.

Next.js: *Framework* de *React* para apps web con *routing*, *SSR/SSG* y *APIs* integradas. [6]

Node.js: Entorno de ejecución de *JavaScript* del lado del servidor basado en V8.

Observabilidad: Capacidad de comprender el estado interno de un sistema mediante métricas, logs y trazas.

Onboarding: Proceso de incorporación de nuevos empleados a una empresa, que abarca la integración cultural, la formación en el puesto y la socialización con el equipo.

OpenAI: Organización de IA creadora de modelos como GPT y herramientas asociadas.

ORM: Técnica de programación que actúa como un puente entre los lenguajes de programación orientados a objetos y las bases de datos relacionales. [13]

PoC: Prueba de concepto para validar viabilidad técnica rápidamente y con alcance acotado.

Portabilidad: Facilidad para ejecutar un *software* en múltiples entornos sin cambios significativos.

PostgreSQL: Base de datos relacional open-source avanzada y extensible. [22]

Pricing: Estrategia y política de fijación de precios de un producto o servicio.

Prisma: *ORM* para *Node.js/TS* con esquema declarativo y cliente tipado. [13]

Product Backlog: *Backlog* del producto con todas las necesidades priorizadas por valor.

Product Owner: Rol de *Scrum* responsable de maximizar valor y gestionar el *Product Backlog*.

Prompt: Instrucción o contexto que se le da a un modelo generativo para guiar su salida. [10]

Prompting: Técnicas y prácticas para diseñar *prompts* efectivos y reproducibles. [10]

Pull Request: Propuesta de cambios para revisión antes de integrarlos en la rama principal.

Python: Lenguaje de programación código abierto, interpretado y de alto nivel, con gran ecosistema científico.

Qdrant: Base de datos vectorial *open-source* con búsquedas por similitud y filtrado. [14]

RDS: Servicio de bases de datos relacionales administradas de *AWS* (*Postgres*, *MySQL*, etc.). [27]

React: Biblioteca de *JavaScript* para construir interfaces de usuario con componentes.

Release: Versión publicada del *software* lista para distribución o producción.

Rendimiento: Desempeño del sistema (medible en latencia, *throughput*, uso de recursos) bajo carga real.

Repository Pattern: Patrón que abstrae el acceso a datos agrupando operaciones por agregado/entidad. [33]

Repositorio: Lugar donde se almacena y organiza un conjunto de información, ya sea digital o física. [34]

RF: Requerimientos Funcionales; describen lo que el sistema debe hacer.

RNF: Requerimientos No Funcionales; definen cómo debe comportarse de acuerdo a ciertos aspectos (seguridad, rendimiento, etc.).

S3: Servicio de almacenamiento de objetos de *AWS* para datos estáticos, *backups* y *assets*. [27]

SaaS: Modelo donde el *software* se ofrece como servicio accesible por internet.

Scope creep: Expansión no controlada del alcance sin ajustar tiempo o recursos.

Scrum: Marco ágil basado en iteraciones, roles definidos y eventos para entregar valor continuo. [1]

Score: Puntuación numérica que resume una métrica de interés (p. ej., nivel de *matching*).

Scouting: Búsqueda proactiva de candidatos potenciales para futuros roles.

Scikit-learn: Librería de *machine learning* en *Python* con algoritmos clásicos y utilidades.

Selección de personal: Proceso para evaluar a los candidatos y elegir al más adecuado para un puesto vacante.

shadcn/ui: Colección de componentes de *React* estilizables con *Tailwind* y *Radix* como base.

Soft skills: Habilidades blandas, son competencias personales y sociales no técnicas que influyen en cómo trabajamos y nos relacionamos con otros.

Story Point: Unidad relativa para estimar esfuerzo/complejidad de historias de usuario en un marco ágil.

Terraform: Herramienta para definir y aprovisionar infraestructura con código declarativo. [29]

Transformers: Arquitectura de *deep learning* basada en atención, base de los *LLM* modernos.

TypeScript: Superconjunto de *JavaScript*, que esencialmente añade tipos estáticos y objetos basados en clases.

UX/UI: Diseño de experiencia (*UX*) e interfaz (*UI*) que optimiza uso y apariencia del producto.

Vacante: Posición laboral abierta publicada por una empresa para buscar y contratar candidatos que cumplan con requisitos específicos.

Vector: Arreglo numérico con magnitud/dirección; en *Machine Learning*, representación de características. [14]

VPC: Red virtual aislada en la nube que segmenta y protege recursos. [27]

Zod: Librería de *TypeScript* para definir esquemas y validar datos con tipos seguros.

Índice

1. Introducción

1.1. Presentación del equipo

El equipo de trabajo del proyecto está integrado por 3 estudiantes de la carrera Ingeniería en Sistemas de la Universidad ORT Uruguay.

- Martín Caffarena (230351)
- Franco Daneri (260284)
- Nicolas Gonzalez (231843)

Los tres integrantes del equipo se conocían previamente al inicio de la realización del proyecto y habían coincidido en distintas instancias de trabajo, por lo que se contaba con una ventaja significativa al conocer formas de trabajo, fortalezas y debilidades. Esto logró favorecer la comunicación y la colaboración debido a la confianza mutua y el conocimiento previo de los diferentes estilos de trabajo.

Además es oportuno mencionar, que los integrantes mantienen intereses similares, tanto académica como profesionalmente, alineándose con los objetivos de este proyecto.

1.2. Alcance y límites del documento

Este documento se organiza como un relato completo del proyecto a lo largo de todo un año de trabajo. Inicia presentando el contexto y la definición del problema a resolver, incluyendo a los usuarios involucrados y las condiciones que dieron origen a la necesidad del producto. Luego, expone la solución propuesta y los fundamentos que la respaldan, para después detallar la investigación técnica previa que orientó las decisiones de diseño e implementación. A continuación, se documenta la ingeniería de requerimientos que formalizó el alcance del sistema, y se describe la arquitectura del sistema, destacando las decisiones clave y las tecnologías seleccionadas. Posteriormente, se abordan los aspectos de proceso y metodología que habilitaron las entregas iterativas, para luego pasar a la implementación, la infraestructura y la estrategia de despliegue. También se incluye el enfoque de aseguramiento de la calidad, las métricas y evidencias recolectadas, así como los resultados de validación con el cliente y usuarios. Finalmente, se presentan los riesgos gestionados, las lecciones aprendidas, las conclusiones y el trabajo a futuro, dejando planteada la evolución del producto más allá del *MVP* entregado.

El alcance de esta documentación se centra en el producto entregado y en las decisiones del proyecto. En síntesis, el documento busca mostrar de manera clara y completa el proceso recorrido, las decisiones adoptadas, el trabajo realizado y los logros alcanzados, dejando una base consistente para futuras iteraciones del producto, plasmando claramente todo lo realizado por el equipo de trabajo.

1.3. Objetivos

Desde el inicio del proyecto se definieron objetivos claros y medibles que permitieran evaluar su éxito. Estos objetivos funcionaron como brújula, manteniendo al equipo enfocado y alineado en cada etapa, garantizando que todos los esfuerzos llevasen hacia metas compartidas.

Se dividieron los objetivos en tres categorías principales, objetivos académicos, objetivos del proceso y objetivos del producto.

1.3.1. Objetivos académicos

OA1 - Aplicar los conocimientos adquiridos durante la carrera

La intención es consolidar el aprendizaje en condiciones profesionales, poner en práctica, en un escenario real y más complejo que los abordados a lo largo de la carrera, los conceptos de ingeniería de *software* adquiridos, como, definición de requerimientos, desarrollo, gestión de proyectos, formas de trabajo, entre otros, integrándose frente a restricciones de alcance, tiempos y un cliente real.

OA2 - Ampliar conocimientos y dominio en nuevas tecnologías

Explorar y dominar tecnologías y prácticas no utilizadas previamente, así como expandir la experiencia en conocimientos previos, para fortalecer el perfil académico-profesional de los integrantes del equipo; la intención es cerrar brechas y ampliar la empleabilidad del equipo de los mismos.

OA3 - Adquirir experiencia en realización de producto

Ejecutar de punta a punta el ciclo de vida de un producto, desde la definición del problema y los requerimientos hasta el despliegue y *handover*, desarrollando criterio sobre *trade-offs* de alcance, tiempo, costo y calidad; la intención es consolidar experiencia práctica en construcción de producto, más allá de la mera entrega de un proyecto.

OA4 - Aprobar el proyecto de fin de carrera con nota de excelencia

Una prioridad fundamental para el equipo fue ejecutar la realización de un proyecto que tenga muy buenos resultados en términos académicos. Por lo que se fijó como objetivo académico obtener una calificación ≥ 95/100 de proyecto.

1.3.2. Objetivos del proceso

OPC1 - Mantener una comunicación clara y continua con el cliente

Sostener canales y continuidad de comunicación con el cliente que brinden visibilidad permanente sobre avances, cambios y decisiones, facilitando *feedback* temprano y trazable; la intención es alinear expectativas de forma continua, reducir retrabajos y estando en alineación continua con el cliente y sus necesidades reales.

OPC2 - Mantener una relación sólida con el cliente durante todo el proceso

Construir una relación de confianza y colaboración bidireccional con el cliente, con disponibilidad y respuesta oportuna a consultas de ambas partes; la intención es favorecer decisiones informadas en un clima de trabajo saludable, donde ambas partes se beneficien y exista seguridad en el cumplimiento de compromisos.

OPC3 - Aplicar de forma efectiva un marco de gestión ágil para la gestión del proyecto

Adoptar y adaptar prácticas ágiles al contexto del proyecto y del equipo, manteniendo foco en resultados; la intención es maximizar la entrega temprana de valor y la capacidad de respuesta ante el cambio sin perder control de calidad ni trazabilidad.

OPC4 - Reflexionar y mejorar continuamente luego de cada iteración

Realizar instancias sistemáticas de reflexión para identificar aciertos, problemas y oportunidades de mejora, definiendo acciones concretas para el futuro; la intención es asegurar un proceso de mejora continua que eleve la calidad del producto y la eficiencia del equipo en cada ciclo.

1.3.3. Objetivos del producto

OPO1 - Mejorar los procesos de adquisición de personal del cliente

Optimizar tiempos y calidad del proceso de selección interno del cliente, facilitando la identificación de candidatos con mejor ajuste a cada posición y reduciendo tareas manuales y retrabajos; la intención es aumentar eficiencia y efectividad en la decisión de *hiring*, entregándole valor tangible al cliente desde el primer uso.

OPO2 - Proveer al cliente un *MVP* cerrado con la proyección de un producto comercial

Entregar un *MVP* listo para uso interno, de alcance cerrado y sin huecos funcionales críticos, con criterios de aceptación cumplidos y defectos severos minimizados; adicionalmente, disponer de arquitectura y documentación que permitan, con un esfuerzo incremental acotado, su adaptación a una oferta comercial. La intención es garantizar funcionamiento inmediato que aporte valor al cliente, con fundaciones técnicas sólidas que permitan su crecimiento gradual hacia un producto comercial robusto.

OPO3 - Aportar de forma continua valor al cliente

Planificar y priorizar el trabajo según su impacto en el cliente final, asegurando que cada iteración entregue funcionalidades útiles, valide hipótesis de negocio y/o contribuya al crecimiento sostenible del producto.

OPO4 – Entregar un producto integral y validado

Asegurar que el sistema se presente completo, pulido y libre de carencias perceptibles; que cada componente opere con robustez y coherencia bajo criterios de aceptación claros y exigentes. Habiendo atravesado diversas rondas sistemáticas de pruebas funcionales y no funcionales, seguidas de validación con usuarios clave, a fin de mitigar previamente los defectos de mayor impacto. De esta forma, entregando un producto cerrado, confiable y listo para operar.

OPO5 - Obtener una satisfacción completa de parte del cliente

Alcanzar una valoración positiva del cliente respecto a comunicación, dedicación, calidad y alineación con lo solicitado, reflejada en su percepción de valor y confianza en el equipo; la intención es evidenciar el ajuste del problema y la solución, consolidando la relación del producto y habilitando la continuidad y expansión del proyecto.

1.4. Cliente

GoGrow es una empresa especializada en desarrollo de *software* a medida y servicios de *staff augmentation*, orientada a ofrecer soluciones tecnológicas adaptadas a las necesidades específicas de cada proyecto y a ampliar equipos técnicos con talento especializado de forma flexible y escalable. Esta última línea de negocio se vincula directamente con la necesidad abordada en este trabajo: optimizar sus procesos internos de adquisición de talento.

Durante el proyecto, los representantes por parte de GoGrow fueron Franco Pariani (CEO), Bruno Pariani (*UX/UI*), Nicolás Erlichman (*Tech Advisor*) y Martina Vega (*Recruiter*), quienes participaron en instancias de definición, validación y demostraciones.

1.5. Motivación y elección del proyecto

El proyecto fue seleccionado a través de la feria de proyectos de la Universidad ORT. El equipo analizó tres alternativas y optó por esta propuesta por su alcance *end-to-end* (desde descubrimiento y definición hasta despliegue), la claridad y utilidad del problema planteado por el cliente y el contexto de desarrollo *web* moderno con integración de IA (Inteligencia Artificial), un área de alto interés para los integrantes.

Adicionalmente, se ofrecía espacio para investigar tecnologías y proponer mejoras, presentando desafíos a nivel teórico, técnico y metodológico que el equipo consideró motivantes. La temática de adquisición y gestión de talento resultaba especialmente atractiva por su potencial comercial, combinando así aprendizaje significativo para el equipo, aporte real al cliente y proyección futura.

2. Problema y Validación del Problema

2.1. Contexto

Como ya se mencionó, el cliente genera valor a través de dos líneas de servicio, el desarrollo de *software* a medida y el *staff augmentation*, la cual está fuertemente ligada con el contexto del proyecto. Este último consiste en integrar profesionales tecnológicos, desarrolladores, diseñadores, etc, dentro de equipos de clientes externos por tiempo determinado o indefinido, asumiendo la selección, la contratación y la administración de recursos humanos mientras el cliente se concentra en los resultados del negocio. En ese esquema, la capacidad de encontrar, evaluar y disponer de perfiles ajustados y tiempos acotados deja de ser un proceso administrativo más para convertirse en uno de los pilares de la oferta de servicios. Se puede entender entonces como cualquier optimización o mejora en la calidad de dicha búsqueda y obtención de recursos humanos impacta de forma directa en el valor y crecimiento de la empresa.

2.1. Descripción del problema

Uno de los métodos más comunes hoy en día para la selección de personal es la búsqueda reactiva: cuando surge una vacante se publica el aviso y se espera la llegada de postulaciones. Aún con múltiples canales y tácticas de búsqueda, persisten limitaciones recurrentes: alto consumo de tiempo, subjetividad en la evaluación, seguimiento manual y *feedback* escaso hacia los candidatos. Estos factores dificultan decisiones oportunas y consistentes. De esta manera, el problema se transforma y ya no se trata de “encontrar perfiles” sino de priorizar, comparar y justificar decisiones con criterios trazables, objetivos, explicativos y sin grandes demoras.

2.2. Dolores del cliente y situación actual

De acuerdo a lo conversado con el equipo de GoGrow al comienzo del proyecto y sumando las entrevistas que se tuvo con la *recruiter* de la empresa, se identificaron algunos dolores sobre su proceso de selección de personal actual.

En primer lugar, se observó que los procesos manuales son ineficientes ya que la búsqueda manual de candidatos consume mucho tiempo, hay una desorganización en la gestión manual de perfiles y vacantes y muchos procesos llegan a ser subjetivos, no siempre contratando al candidato más adecuado.

Estas limitaciones generan procesos largos y tediosos para ambas partes, teniendo que coordinar entrevistas, hacer un seguimiento manual sobre los estados de los candidatos y entregar *feedback* a los mismos.

Por último, el cliente indica que la información sobre los candidatos suele estar desactualizada y dispersa, sobre todo en los perfiles de los candidatos que fueron evaluados en procesos anteriores. Al no contar con una forma centralizada de acceder a esta información, puede darse el caso de perder la trazabilidad del proceso.

2.3. Investigación con usuarios

A partir de la entrevista realizada a la reclutadora de GoGrow y de las encuestas aplicadas a candidatos, se intentó validar que los problemas que se habían identificado de forma teórica son reales y se reproducen en los procesos de selección de personal actuales. Se buscó validar el problema con personas que participan activamente en los dos lados del mostrador: quienes buscan talento y quienes buscan oportunidad. El objetivo era comprobar si la lentitud, la subjetividad y la falta de retroalimentación que habíamos detectado en la fase de investigación del proyecto, la cual fue una fase para reforzar la investigación ya realizada por parte del cliente, eran percibidos de la misma manera por quienes viven el proceso.

2.3.1. Entrevista a *recruiter*

En la fase temprana del proyecto se realizó una entrevista con Martina Vega, *recruiter* de GoGrow, donde se hicieron una gran variedad de preguntas para poder entender mejor la problemática y, por tanto, validar que este es un problema real actualmente en los procesos de selección de personal (Ver anexo 16.2 "Notas meet validación problema con *recruiter*"). Martina al ser la responsable de los procesos de selección de personal dentro de la empresa podía dar un punto de vista valioso como un posible usuario de empresa.

La entrevista reveló que, en la práctica, la mayor parte del peso recae todavía en el criterio del reclutador y en la cantidad de entrevistas que logra hacer (normalmente dos o tres: un filtro inicial, una técnica y una de "*fit*"). Las habilidades duras se validan con años de experiencia, certificaciones o pruebas técnicas; sin embargo, las blandas y el ajuste cultural se evalúan "a ojo" mediante preguntas abiertas que giran en torno al modelo *STAR* (Situación, Tarea, Acción, Resultado). Esta forma de trabajo genera varios efectos negativos: consume mucho tiempo, obliga al candidato a repetir la misma historia en cada empresa y deja fuera del registro valiosos matices que solo quedan en la memoria del entrevistador. Además, la dispersión de la información (CVs en carpetas locales, notas en mails o en el aire) impide construir una trazabilidad útil para decisiones futuras.

Por otro lado, la entrevistadora establece que cada vacante exige un orden de prioridades distinto: a veces pesa más el inglés C1, otras la experiencia en cloud o la disponibilidad inmediata. La entrevistada quiere poder asignar un porcentaje visible a cada criterio (por ejemplo, 40% idioma, 30% seniority, 20% *fit* cultural, 10% certificaciones) y que el sistema ordene automáticamente los candidatos según ese peso, sin tener que revisar todos los CV como hasta ahora.

Martina indica que utiliza inteligencia artificial (por ejemplo *Chat GPT*) para poder no partir de "en blanco" y tener una idea de descripciones o ciertos aspectos que debe redactar sobre una vacante a la hora de publicarla. Ella plantea que si se pudiese tener una forma de obtener una descripción o redactar ciertos aspectos de la vacante generados con inteligencia artificial sería de gran ayuda y agilizaría la postulación de vacantes en gran medida.

Otro aspecto a tener en cuenta según Martina son los “descartes rápidos”, ya que normalmente hay ciertos aspectos que son obligatorios sobre un candidato a la hora de considerarlo. Antes de abrir la convocatoria suele tener una lista mental de requisitos excluyentes: no vive en la Argentina, pretende más del 30% sobre el tope presupuestario o no puede trabajar en modalidad híbrida, por dar algunos ejemplos. Actualmente debe abrir cada CV, buscar la información y anotar el “no” en una planilla aparte. Un formulario de auto-filtrado obligatorio, con campos como país de residencia, expectativa salarial, disponibilidad y nivel de inglés, que impida siquiera postularse a quienes no cumplen los mínimos le ahorraría entre 20 y 30 minutos por candidato y evitaría la fricción de decir “no” a postulantes que nunca tuvieron chance.

A su vez, Martina confesó que solo da *feedback* técnico y sólo si la persona llegó a la entrevista final; el resto recibe el genérico “seguiremos con otro perfil”. Le genera “malestar” porque sabe que es información que el candidato podría aprovechar. Un sistema que, al descartar, envíe automáticamente un breve informe (“Tu experiencia en *Node* es sólida, pero el cliente requiere nivel C2 de inglés y tu puntaje fue B2”) aumentaría la percepción de marca empleadora y reduciría los *mails* de reclamo que también le quitan tiempo.

En resumen, la entrevista con Martina validó que el proceso actual recae casi por completo en el criterio manual del recruiter: ponderaciones “en la cabeza”, filtros eliminatorios revisados uno por uno, descripciones redactadas con ayuda externa y *feedback* casi inexistente para el 90% de los candidatos. Además, ella necesita poder asignar pesos visibles a cada requisito, auto-filtrar quienes no cumplen mínimos, generar textos de vacantes con IA y, al descartar, entregar al postulante un breve informe automático que hoy no da por falta de tiempo.

2.3.2. Encuestas a candidatos

Por otro lado, se aplicaron casi 90 encuestas a personas mayores de edad, con edades variables entre 18 y 60 años de edad, con una mayor concentración de encuestados entre los 20 y 30 años, con experiencias que van desde “sin experiencia” hasta más de 10 años en diversos sectores laborales (Ver preguntas y resultados en anexo 16.3 “Encuesta validación problema con usuarios candidatos”). Los datos obtenidos son un fuerte apoyo sobre que el problema no es solo interno del reclutador: el candidato vive el proceso como una “caja negra”, donde se le da poca información sobre la evolución del proceso.

Se ve como más de la mitad calificó la comunicación como “regular” o “mala” y el 62% recibe *feedback* “nunca” o “solo a veces”. La frase más repetida fue “rechazo sin explicación”; 7 de cada 10 participantes afirmaron haber quedado descartados sin saber qué criterio falló. Al preguntarle a los encuestados “¿Qué tipo de información te gustaría recibir al finalizar un proceso de selección?” las respuestas fueron en su gran mayoría mostró interés en devoluciones sobre el rechazo o aceptación, indiferentemente del caso. Con esto se puede ver como la poca información brindada hacia los candidatos durante o finalizado el proceso es muy poca, poniendo nerviosismo e incertidumbre sobre la persona que participa del proceso. A su vez, al no brindarle una devolución en ningún momento no se le permite a los candidatos mejorar sobre los aspectos que pueden llegar a tener debilidades.

Además, casi la mitad cree que las empresas valoran sólo formación y años de experiencia; dos tercios sienten que las plataformas actuales no permiten cargar habilidades blandas, estilo de trabajo o motivaciones, indicando que les gustaría poder brindar más información que solo el CV en su perfil (habilidades blandas, conocimientos, personalidad, estilo de trabajo, entre otros). En palabras de los encuestados: “No está claro qué información es valiosa”, “terminas subiendo el mismo CV a todos lados”. Pudiendo quedar fuera el candidato que más se adapta a una vacante solamente por no tener un buen CV o por no poder ver más allá del mismo.

El 68 % cree que ha perdido chances por “llegar tarde a la vacante” o porque “hay demasiadas postulaciones”, lo que implica que muchos perfiles nunca llegan a ser evaluados en profundidad. Es así como no se llega a evaluar todos los candidatos posibles, lo que hace que en muchos casos se tome una decisión sobre una cantidad limitada y no siempre la mejor

sobre a qué candidato seleccionar. Por lo que es una necesidad poder llegar a evaluar a todos para tomar la mejor decisión sobre a quién contratar.

Por otro lado, el 92 % ve “muy” o “bastante” acertado un sistema que analice intereses, valores personales y estilo de comunicación para recomendar empleos. También 9 de cada 10 están de acuerdo con recibir, al finalizar un proceso, un breve informe que incluya: Motivo concreto de la decisión, fortalezas detectadas y áreas de mejora para futuras aplicaciones. Viendo así que la mayoría de encuestados están a fín con una plataforma que tenga un perfil donde se pueda establecer más información sobre el candidato y tener mayor comunicación y devoluciones durante los procesos de selección de personal.

2.3.3. Conclusión de las investigaciones con usuarios

La entrevista con Martina y las respuestas de más de ochenta posibles candidatos componen el mismo escenario: un proceso que consume mucho tiempo para unos y genera demasiadas dudas para otros. Del lado de la empresa se confirmó el costoso trabajo que implica revisar manualmente cada perfil, recordar criterios cambiantes y redactar desde cero cada convocatoria. Del lado del postulante quedó en evidencia la sensación de entrar a una “caja negra” donde no se sabe qué pesa más, en qué momento se decide ni por qué se descarta.

Ambas investigaciones muestran que la desproporción entre las partes perjudica la calidad de la decisión: quien evalúa no dispone de datos estandarizados sobre motivaciones, valores o estilo de trabajo, mientras que quien es evaluado desconoce los puntos de corte y repite la misma historia en cada nueva instancia. La comunicación se rompe en el camino: el reclutador prioriza avanzar con los cree que “primero encajan” y el candidato recibe, a lo sumo, un correo genérico sin información relevante sobre la decisión tomada.

En conjunto, se valida que la lentitud, la subjetividad y la falta de retroalimentación no son simples molestias: constituyen un problema compartido que afecta la eficiencia del proceso, la percepción de marca empleadora y, en última instancia, la posibilidad de encontrar el ajuste correcto entre persona y puesto. Resolverlo implica migrar de un sistema basado en criterios implícitos y memoria individual a uno que articule datos, ponderaciones transparentes y cierres informativos para ambas partes.

2.4. Hallazgos y formalización del problema

A partir del análisis de los dolores del cliente y de la situación actual, se identificaron tres categorías principales de ineficiencias en los procesos de selección.

En primer lugar, se encontraron ineficiencias operativas, caracterizadas por una alta dependencia de procesos manuales que consumen tiempo excesivo. Además, existe una gestión dispersa de la información de candidatos y vacantes, lo que genera pérdida de trazabilidad en las evaluaciones previas.

En segundo lugar, se observaron limitaciones en la toma de decisiones, principalmente debido a evaluaciones subjetivas que no garantizan la selección del candidato más adecuado. Esta situación se ve agravada por la falta de criterios estandarizados y comparables que permitan una evaluación justa y consistente.

Por último, se detectaron deficiencias comunicacionales, como el seguimiento manual inadecuado del estado de los candidatos y la retroalimentación insuficiente hacia los postulantes. Esto se suma a la coordinación compleja de entrevistas y procesos, lo que afecta negativamente la experiencia tanto de los candidatos como de los equipos de reclutamiento.

Teniendo en cuenta estos descubrimientos, se plantea el problema de la siguiente forma: ¿Cómo se pueden sistematizar y optimizar los procesos de selección para reducir la carga manual, mejorar la trazabilidad de decisiones y potenciar la comunicación con los candidatos?

3. Solución Propuesta

En este capítulo, se presenta la solución propuesta para abordar los problemas identificados en los procesos de selección de nuestro cliente y personas que han participado en procesos de selección.

3.1. Surgimiento de la solución

La iniciativa surge de parte del cliente, GoGrow, a partir de su propia experiencia en *staff augmentation* y de los procesos de selección que aplica internamente, identificando ineficiencias, subjetividad y pérdida de trazabilidad en cada búsqueda. Ante ese panorama, la empresa vio la oportunidad de construir una herramienta que optimiza y agiliza sus procesos de búsqueda y adquisición de talento, y que, con el tiempo, pudiera convertirse en un producto comercial. De esta reflexión nace Whizdy.

3.2. Visión de Whizdy

Whizdy es una plataforma *SaaS* de *scouting* y selección asistida por Inteligencia Artificial (IA). Integra un sistema de *matching* multidimensional que transforma un proceso desorganizado, subjetivo y poco trazable en uno explicable, medible e informativo, mejorando y optimizando la experiencia tanto para candidatos como para empresas.

A diferencia del reclutamiento tradicional donde las empresas esperan que los candidatos apliquen para evaluar postulantes, Whizdy identifica y permite contactar directamente a profesionales que coinciden con el perfil buscado.

3.3. Perfiles de usuario y casos de uso

Se pueden agrupar los usuarios de la solución propuesta en dos perfiles claros, candidatos y empresas, cada uno con funcionalidades y objetivos particulares dentro del mismo sistema.

3.3.1. Perfil de candidato

Los candidatos son profesionales que buscan oportunidades laborales y desean ser encontrados por empresas alineadas con sus habilidades, experiencia y preferencias profesionales. Sus principales funcionalidades incluyen:

- **Configurar perfil profesional completo**: *Onboarding* estructurado estableciendo información básica, habilidades, experiencia profesional, preferencias laborales y salariales.
- ***Dashboard* con información centralizada:** Acceso a un *dashboard* personalizado donde poder ver de forma resumida todas las posiciones abiertas en las que se está participando con su valoración de *matching* y estado, visualización de calendario con entrevistas agendadas, resumen gráfica de valoraciones de *matching* de su perfil y una serie de *KPIs* detallando elementos claves sobre sus procesos de selección y rendimiento.
- **Participar en procesos de selección**: Interacción de manera estructurada con las empresas a través de notificaciones en tiempo real, participa en entrevistas presenciales o virtuales coordinadas por la plataforma, proporciona y recibe *feedback* sobre el proceso, y evalúa ofertas de trabajo para tomar decisiones informadas.
- **Explorar vacantes de trabajo**: Se tiene acceso a todas las vacantes de trabajo que se encuentran activas en el momento, pudiendo guardar, mostrar interés y acceder al detalle completo de la misma. Así como también acceder a los perfiles corporativos detallados, viendo la cultura empresarial, posiciones activas y beneficios ofrecidos, y así poder tomar decisiones informadas.

3.3.2. Perfil de empresa

El perfil de empresa corresponde al acceso corporativo para responsables de identificar, evaluar y contratar talento. Sus principales funcionalidades incluyen:

- **Configurar perfil corporativo**: Establecer su perfil en la plataforma definiendo información básica, configurando elementos de *branding*, detallando beneficios, compensaciones y oportunidades de desarrollo, y agregando certificaciones y reconocimientos para posicionarse como empleador atractivo.
- **Crear vacantes de trabajo**: Crear ofertas laborales especificando habilidades técnicas requeridas, estableciendo atributos relevantes de la misma, y gestionando el ciclo de vida desde borrador hasta activación.
- **Gestionar procesos de candidatos**: Administrar el flujo de candidatos a través de las diferentes etapas del proceso de selección y mantener trazabilidad completa del progreso de cada candidato en tiempo real.
- **Visualizar candidatos**: Tener acceso centralizado a los diferentes candidatos disponibles, visualizando *scoring* de compatibilidad detallado para cada candidato. Pudiendo siempre tomar una decisión fundamentada sobre cómo seguir el proceso de un candidato.
- **Evaluar y dar seguimiento a candidatos**: Coordinar entrevistas presenciales o virtuales, registrando calificaciones cuantitativas y *feedback* cualitativo de manera estructurada, gestionando el avance de candidatos entre etapas del proceso y dando *feedback* del mismo, dejando comentarios entre miembros de la empresa.
- **Analizar métricas de reclutamiento**: Las empresas acceden a datos analíticos incluyendo tiempo promedio de contratación y tasas de conversión, analizan la efectividad del *matching* por posición y departamento, realizan comparativas temporales para identificar tendencias, y obtienen *insights* estratégicos para optimizar procesos y tomar decisiones basadas en datos.

3.4. Alcance del proyecto

Al comenzar el proyecto se acordó la entrega de un *MVP* funcional de uso interno exclusivo: diseño fiel al prototipo de *Figma*, *backend* centralizado y una primera versión experimental del sistema de *matching*. El cliente dejó en claro que, en esta iteración, el algoritmo no necesitaba estar sumamente refinado y que un despliegue productivo no era obligatorio; bastaba con demostrar viabilidad.

El equipo cumplió ese mandato y lo amplió. El *backlog* final superó los 50 requerimientos funcionales, se fijaron umbrales medibles para cada atributo de calidad y el *matching* atravesó múltiples ciclos de ajuste y validación con usuarios. Se realizaron múltiples sesiones de *testing* que depuraron el sistema de defectos críticos, se automatizó el *pipeline* de *CI/CD*, se ejecutaron sucesivos despliegues y se documentó la arquitectura pensando en las fases siguientes del producto, llegando a la entrega de un sistema funcional, cerrado y listo para el uso interno por parte del cliente.

Ante la extensión del entregable, la operatividad del sistema y el valor que ya aporta, surge la pregunta: **¿Estamos realmente ante un *MVP*?** Se retomará esta reflexión más adelante.

4. Ingeniería de Requerimientos

El siguiente capítulo cubre cómo se obtuvo y se planteó el alcance funcional y no funcional del producto, dejando como cierre los listados consolidados de requerimientos funcionales (RF) y requerimientos no funcionales (RNF). La intención es narrar el recorrido completo desde la identificación de necesidades hasta su formalización.

4.1. Relevamiento

El punto de partida fue una investigación realizada por GoGrow, donde se analizaron otros proyectos, estudiaron la competencia y extrajeron ideas de aplicaciones similares en el mercado. Con base en este análisis, su equipo de *UX/UI* elaboró un diseño inicial en *Figma* de aproximadamente 15-20 pantallas, incluyendo ventanas principales, flujos básicos y títulos iniciales de historias de usuario.

A partir de este material, el equipo expandió los títulos iniciales de las historias de usuario a historias más completas, basándose en el diseño inicial de *Figma* así como también en las reuniones de trabajo que se llevaron a cabo con el equipo de GoGrow para comprender el proceso real de *scouting* y selección de personal. En estas instancias se identificaron dolores operativos adicionales, como tiempos elevados, seguimiento manual, comunicación irregular y decisiones poco trazables, bosquejando un primer marco de solución.

Este material, brindado por el cliente, funcionó desde el comienzo como referencia visual y de comportamiento para discutir alcance y expectativas iniciales, convirtiéndose en la fuente central de entendimiento de las expectativas y visualización del cliente sobre el producto. Durante todo este proceso se realizaron consultas puntuales para aclarar supuestos y completar huecos funcionales identificados.

Como ya se mencionó, el relevamiento se profundizó con una entrevista específica a la *recruiter* de GoGrow, quien aportó la mirada cotidiana del proceso: cómo se originan las vacantes, qué información falta al comparar perfiles, dónde se demoran decisiones y en qué puntos se pierde trazabilidad. Con esa perspectiva, se ajustó lo recibido, se señalaron otras debilidades y se propusieron mejoras puntuales orientadas a que los flujos fueran operables en la práctica; varias de esas sugerencias se incorporaron como comportamientos y microinteracciones coherentes con la visión planteada por el cliente.

Durante el período posterior a la primera entrega al equipo, se continuó profundizando en el entendimiento del problema y en la definición de requerimientos, mientras el equipo de *UX/UI* del cliente avanzaba en la evolución del material de diseño.

En determinado momento, se presentó una nueva versión del diseño, que había crecido de forma significativa hasta superar ampliamente el centenar de páginas. Este incremento incorporó funcionalidades, variantes de flujo y elementos no contemplados inicialmente, pero necesarios para materializar la visión completa del producto.

Esta expansión generó algunas diferencias entre la interpretación original del equipo, basada en la primera especificación visual y las conversaciones iniciales y la nueva versión entregada por el cliente. Para resolverlas, fue necesario un trabajo adicional de alineación: se revisaron historias, se refinaron datos y alcances, y se ajustaron los requerimientos a la especificación visual final y a las necesidades operativas reflejadas en el nuevo material de diseño.

De este modo, la especificación visual se consolidó como un instrumento dinámico de comunicación de la visión del cliente y como guía continua para determinar y profundizar el alcance definitivo del proyecto.

En paralelo al trabajo sobre los requerimientos funcionales se abordó la definición de requerimientos no funcionales. El cliente no tenía una idea concreta sobre los mismos inicialmente a diferencia de los requerimientos funcionales, sin embargo al ser consultado, enfatizó la necesidad de resguardar la seguridad y el aislamiento de datos, es decir que cada empresa y cada candidato pudiera acceder únicamente a su propia información. Si bien el equipo estuvo de acuerdo que ese punto era necesario e indispensable para la realización del proyecto, considero este como no suficiente para un proyecto de este alcance, por lo que el equipo propuso un conjunto acotado y razonable de RNF, calibrado al uso esperado del producto, a las restricciones, supuestos vigentes y a la viabilidad técnica del *stack* seleccionado.

La propuesta se planteó con un criterio claro para asegurar un nivel de calidad verificable sin convertir los RNF en un freno para la entrega. Así, se priorizaron categorías con impacto directo en la operación, fijando objetivos claros y medibles, acordes a un lanzamiento interno y controlado. El cliente aprobó esta línea por considerarla razonable y alineada con sus expectativas, aclarando la búsqueda de un sistema robusto en lo esencial,

construido con buenas prácticas y estándares conocidos, sin imponer exigencias desproporcionadas para esta primera versión.

En síntesis, el trabajo de relevamiento avanzó en dos etapas claras, primero partiendo del *Figma* inicial de 15–20 pantallas, con talleres y una entrevista a la *recruiter* para aterrizar dolores operativos y convertir expectativas en historias de usuario verificables, y luego, sobre una versión del diseño que superó el centenar de páginas, cuya evolución exigió alinear criterios, refinar datos y ajustar alcances. El resultado fue un *backlog* priorizado y trazable a necesidades explícitas, con la especificación actuando como instrumento dinámico de comunicación y gestión del alcance. En paralelo, se definieron los requerimientos no funcionales como condiciones transversales y se consolidó un conjunto acotado y medible, calibrado al contexto y al proyecto. (Se puede visualizar el crecimiento de los diseños de *Figma* en el anexo 16.4. “Evidencia - Crecimiento de diseño en *Figma*”)

4.2. Especificación y consolidación

Con las necesidades claras, se formalizó cada capacidad como historia de usuario con el formato “Como [actor], quiero [acción] para [objetivo]”, agregando criterios de aceptación y, cuando era requerido, notas de implementación. Para ordenar visualmente el flujo de valor desde la perspectiva de cada actor, se aplicó la técnica de *User Story Mapping* [35], disponiendo las historias en un mapa bidimensional que separó y contrastó los recorridos de candidatos y empresas según sus objetivos y prioridades operativas. Este ejercicio permitió detectar *gaps*, evitar duplicaciones y validar la cohesión general del *backlog* antes de agrupar las historias por perfil (candidato/empresa) y por áreas de uso (vacantes, candidatos, *matching*, notificaciones, suscripciones, entre otras), consolidando finalmente los tickets que describirían esencialmente lo mismo.

El *Figma* sirvió como contrato visual: cada discusión sobre flujos, datos y funcionalidades terminaba descritas allí. La especificación textual y la visual evolucionaron juntas; cuando surgían nuevas ideas, se evaluaban contra los dolores operativos y contra el alcance del proyecto antes de incorporarlas, siempre considerando como foco su valor aportado al cliente. De este modo, el *backlog* buscaba mantenerse priorizado y trazable a fuentes claras de necesidad.

En resumen, la especificación convirtió expectativas y pantallas en historias “tangibles”, ordenadas por rol y por dominio funcional, listas para ser validadas. (Ver anexo 16.5. “Evidencia - Primeras historias de usuario” para ver algunas historias de usuario pertenecientes a la primer versión, sobre las cuales se continuó trabajando)

4.3. Criterios de aceptación generales

A continuación se describen los criterios que se adoptaron para validar cada requerimiento funcional. El foco se pone exclusivamente en la validez de la funcionalidad entregada. Por lo tanto, no se abordan aspectos propios del proceso de gestión de cambios, como puede ser revisiones de código, integración continua, *definition of done*, entre otros elementos, los cuales se tratan con detalle en el siguiente capítulo.

Para evitar repetir bajo cada requerimiento funcional una lista idéntica de validaciones, se estableció un conjunto mínimo y transversal de criterios que se aplicó a todos los RF. Estos criterios se limitaron a verificar que la implementación respetara fielmente los diseños y patrones fijados en *Figma*, garantizando continuidad visual y comportamiento uniforme con el resto de la plataforma. Asimismo, se exigió que la nueva capacidad no altere negativamente funcionalidades ya existentes y que el código producido siguiera los estándares de calidad definidos para el proyecto.

Cuando el requerimiento incorporaba lógica de negocio, se acompañó de *tests* unitarios que validaran dicha lógica; si se modificaban reglas previas, se actualiza la *suite* correspondiente. En los casos en que la funcionalidad implicaba decisiones de diseño o de negocio no evidentes, se dejó registro escrito que documentara dichas decisiones, según correspondiera. Por último, se verificó que los flujos expuestos al usuario mostrarán mensajes, estados y retroalimentación coherentes con el sistema de diseño adoptado.

4.4. Requerimientos funcionales

A continuación se consolidan las capacidades funcionales de la plataforma. Para facilitar la lectura de las mismas se agrupan y describen de forma integrada las funcionalidades que el producto debe ofrecer.

4.4.1. Capacidades comunes a los dos perfiles

RF1) Registro tradicional. La plataforma permite crear nuevas cuentas mediante *email* y contraseña, exigiendo que el usuario complete manualmente la información de perfil inicial requerida. Se exige verificación del correo electrónico a través de un código temporal antes de habilitar el *setup* completo de la cuenta, asegurando la validez del usuario y reduciendo intentos de registro indebidos.

RF2) Registro con Google. Alternativamente, los usuarios pueden registrarse mediante *OAuth* 2.0 con Google, autocompletando automáticamente información básica del perfil desde la cuenta de Google (nombre, email, foto de perfil) [23].

RF3) Inicio de sesión tradicional. Los usuarios luego del registro tradicional pueden acceder ingresando el email y la contraseña registrados previamente.

RF4) Inicio de sesión con Google. Los usuarios registrados con Google pueden acceder directamente mediante *OAuth* 2.0 sin necesidad de credenciales adicionales, unificando la experiencia de acceso y manteniendo los mismos controles de sesión y permisos que el resto del sistema.

RF5) Canal de *feedback* hacia la plataforma. Tanto empresas como candidatos pueden enviar comentarios sobre su experiencia tras cada proceso de selección realizado; los envíos quedan registrados para análisis interno del equipo de la plataforma.

RF6) Sesión protegida. La navegación y las *APIs* quedan protegidas por autenticación basada en *tokens* firmados (*JWT*); las rutas exigen sesión válida y se rechazan accesos no autenticados.

RF7) Expiración de sesión. Las sesiones tienen un tiempo de vida limitado y expiran automáticamente tras un periodo de inactividad definido. Al expirar, el sistema requiere una nueva autenticación para continuar con el acceso, protegiendo la cuenta ante sesiones abandonadas o dispositivos desatendidos.

RF8) Recuperación de credenciales. Ante la pérdida de credenciales, el usuario puede recuperar el acceso mediante un enlace seguro con tiempo de expiración limitado.

4.4.2. Perfil de empresa

4.4.2.1. Gestión de perfil

RF9) *Onboarding* de empresa. La compañía completa un alta guiada indicando rubro, localización, tamaño de la empresa y modalidad de trabajo, valores y certificaciones, enlaces de interés, logo y datos del responsable de la cuenta entre otros. Esta información construye el perfil público y habilita el resto de las operaciones. (Ver Imágenes en anexo 16.1 en “RF9 - Onboarding de empresa”)

RF10) Edición de perfil y configuración. La empresa puede actualizar en todo momento su información institucional, manteniendo consistencia entre lo visible para candidatos y lo que alimenta los procesos internos.

RF11) Visualización de perfil. Desde su propia sesión, la compañía puede acceder a su perfil de empresa para verificar cómo se presenta ante los candidatos. (Ver Imágenes en anexo 16.1 en “RF11 - Visualización de perfil”)

RF12) Gestión de miembros y permisos (multi-cuenta). La empresa puede crear usuarios dentro de su organización, asignar roles y limitar acciones.

RF13) Perfil cultural de la empresa. La empresa puede completar un cuestionario para modelar cultura y valores organizacionales de la misma.

4.4.2.2. Gestión de vacantes

RF14) Crear y publicar vacantes. La empresa crea y publica posiciones definiendo título, tipo, modalidad, *seniority*, ubicación, descripción, responsabilidades, habilidades técnicas y blandas con nivel, compensación, lenguajes requeridos, requisitos de experiencia, entre otros. (Ver Imágenes en anexo 16.1 en "RF14 - Crear y publicar vacantes")

RF15) Guardar como borrador. Durante la creación, la compañía puede guardar el avance como borrador y reanudar luego desde el mismo punto, sin perder datos parciales.

RF16) Duplicar vacantes existentes. Para agilizar publicaciones similares, es posible duplicar una vacante previa como plantilla, ajustar los campos necesarios y continuar el flujo de creación normal.

RF17) Ciclo de vida de la vacante. Cada vacante transita estados (activa, archivada, finalizada, entrevistando, etc) con reglas de transición claras y efectos sobre postulaciones y *matching*. El cambio de estado queda registrado y se refleja en las vistas operativas.

RF18) Archivado y reactivación de vacantes. Más allá del ciclo general, la empresa puede archivar una vacante para pausar su operación y reactivarla luego, conservando historial y contexto de candidatos.

RF19) Detalle de vacante con estadísticas. La vista detallada concentra todos los campos de la posición y muestra métricas operativas: métricas de cantidad de visitas de candidatos, interesados, entrevistas realizadas, resumen de *performance* de la vacante y un resumen de las métricas de candidatos por estado y su evolución en el proceso. (Ver Imágenes en anexo 16.1 en "RF19 - Detalle de vacante con estadísticas")

RF20) Visualización y gestión de vacantes. La compañía visualiza todas sus vacantes en una vista de tipo tabla o *board*, con indicadores por etapa y resumen de candidatos por estado. Desde estas vistas puede cambiar el estado de cada vacante de forma segura y ver el efecto en tiempo real. (Ver Imágenes en anexo 16.1 en "RF20 - Visualización y gestión de vacantes")

RF21) Control de *intake* de vacantes. Durante el proceso, la empresa decide si una vacante sigue recibiendo candidatos desde el *matching* o queda cerrada al ingreso, sin afectar el seguimiento de candidatos ya en proceso.

4.4.2.3. Sistema de *matching*

RF22) Recomendación automática de candidatos. Por cada vacante publicada, el sistema obtiene y mantiene un conjunto de candidatos recomendados a partir de toda la información ingresada en la vacante.

El *matching* se calcula por categorías, *Education*, *Technical*, *Soft Skills* y *General Information*, aplicando criterios de compatibilidad y umbrales definidos para cada vacante. Solo se consideran candidatos elegibles según prefiltrado (ubicación/modalidad, experiencia mínima, expectativas salariales, estado/interés).

El resultado del *matching* se actualiza de forma periódica y ante modificaciones en la vacante.

RF23) Actualización automática de *matching*. Más allá de acciones manuales, el sistema mantiene actualizadas las coincidencias de candidatos con vacantes de la empresa, refrescando resultados de manera periódica.

RF24) Recálculo de *matching* manual. Ante cambios en una vacante o ante dudas puntuales, la empresa puede forzar una reevaluación del *matching* para obtener un resultado actualizado inmediatamente.

RF25) Explicación de *matching* con IA (a demanda). Bajo pedido, el sistema genera y persiste una explicación legible de por qué el candidato resulta relevante (o no) para la vacante, ayudando a fundamentar decisiones y facilitar el entendimiento de los resultados obtenidos

RF26) Compatibilidad cultural en el *matching*. Al realizarse una recomendación o *matching* entre candidato y vacante, el sistema calcula un *sub-score* de *cultural fit* y lo incorpora al resultado de *matching* del candidato a la vacante.

4.4.2.4. Gestión de candidatos

RF27) Búsqueda y pool general de candidatos. Más allá de las vacantes activas, la empresa explora un *pool* global (con foco en "*Open to Work*") aplicando filtros por palabras clave, *skills*, experiencia y ubicación para descubrir talento relevante.

RF28) Visualización de candidatos por vacante. Para cada vacante, la empresa ve el conjunto de candidatos con columnas ordenables (*match*, *skills* requeridas cubiertas, promedio de entrevistas, estado dentro de la vacante, etc.) y puede alternar entre tabla y *board* para gestionar por etapas. (Ver Imágenes en anexo 16.1 en "RF28 - Visualización de candidatos por vacante")

RF29) Vista general de candidatos. La organización puede ver todos los candidatos de todas sus vacantes a la vez para tener una visualización más general del estado de sus candidatos evaluados

RF30) Visualización básica de perfiles. Desde listados y *boards*, la empresa abre rápidamente el perfil del candidato para revisar su información clave relacionada sin abandonar el contexto operativo. (Ver Imágenes en anexo 16.1 en "RF30 - Visualización básica de perfiles")

RF31) Visualización completa de perfiles. La vista completa permite analizar el perfil completo del candidato, visualizando educación, experiencia, certificaciones, habilidades duras y blandas, principios/valores, expectativas salariales, *links* de interés del candidato y sí demostró interés por la vacante o no, incluyendo comparativas ágiles de *skills*. (Ver Imágenes en anexo 16.1 en "RF31 - Visualización completa de perfiles")

RF32) Descargar CV del candidato. La empresa puede descargar el CV de los candidatos en caso requiera de más información del mismo fuera del perfil del sistema.

RF33) Resumen automatizado de perfiles. La empresa ve un resumen generado por IA que sintetiza experiencia, habilidades y formación, pensado para lectura rápida durante el *screening*.

RF34) Historial del candidato en la empresa. Para un candidato dado, la compañía consulta todas las vacantes de la misma empresa en las que participó, con su evolución, resultado y estados, preservando trazabilidad.

RF35) Resultados de *matching*. La empresa accede a un desglose del *matching* por secciones (educación, información general, técnicas y blandas), además de compatibilidad por localidad, expectativas salariales y comparativas visuales de *soft skills*. (Ver Imágenes en anexo 16.1 en "RF35 - Resultados de *matching*")

RF36) Visualización del *match* cultural. En conjunto con la información detallada del *matching* para un candidato dentro de una vacante, la empresa puede ver el resultado y los factores del *cultural fit* con su empresa.

RF37) Resultados psicométricos del candidato. En conjunto con la información del perfil de candidato, la empresa puede visualizar los resultados del candidato en los *test* de psicometría realizados

RF38) Cambiar estado de candidatos con *feedback* y notificaciones. La compañía mueve candidatos entre etapas (*recommended*, *interviewing*, *in review*, *offered*, *discarded*, etc.), dejando comentarios internos, enviando *feedback* al candidato por correo cuando corresponde y registrando la acción para trazabilidad. (Ver Imágenes en anexo 16.1 en "RF38 - Cambiar estado de candidatos con *feedback* y notificaciones")

RF39) Gestión de candidatos guardados. La organización marca candidatos para su futuro seguimiento y accede a una vista centralizada de guardados, con filtros para retomar evaluaciones.

4.4.2.5. Proceso de selección

RF40) Agendar entrevistas. Desde la ficha del candidato, la empresa agenda entrevistas para una vacante concreta, definiendo estado, motivo, fecha, hora y enlace; todo queda visible en el flujo y preparado para integraciones de calendario. (Ver Imágenes en anexo 16.1 en "RF40 - Agendar entrevistas")

RF41) Reseñar entrevistas. Tras cada instancia, la empresa puede registrar calificación y comentarios de la entrevista; el promedio de entrevistas se refleja en listados para acelerar la comparación entre candidatos. (Ver Imágenes en anexo 16.1 en "RF41 - Reseñar entrevistas")

RF42) Comentarios por vacante. La empresa deja notas internas asociadas a una vacante específica y puede ver comentarios históricos de otras vacantes del mismo candidato, con diferenciación visual y autor/fecha. (Ver Imágenes en anexo 16.1 en "RF42 - Comentarios por vacante")

RF43) Contacto directo. Desde la ficha, la empresa inicia contacto por los canales soportados utilizando plantillas preconfiguradas que inyectan el contexto de la vacante.

4.4.2.6. Evaluaciones

RF44) Crear *assessments* propios. Posibilidad de crear y editar cuestionarios (*assestments*), definiendo preguntas, su tipo, su valoración, respuestas y lógica de corrección.

RF45) Asignar *assessments* a vacantes. Asociar uno o varios *assessments* a una vacante, estableciendo un *score* mínimo y obligatoriedad de la misma.

RF46) Notificaciones de *completion*. Poder enviar a candidatos de forma individual o a todos los considerados recordatorios para la realización de los *assessments* asignados a la vacante.

RF47) Resultados por candidato/vacante. Visualizar para cada candidato el detalle de sus resultados, pudiendo observar puntuación total, por pregunta, fecha de realización, tiempo de realización y estado del *assessment.*

RF48) Estadísticas agregadas de *assessments*. Ves estadísticas y resumen de resultados de los *assessments* de una vacante, visualizando tasas de realización, puntajes promedio y tendencias.

4.4.2.7. Herramientas de productividad

RF49) *Dashboard* de empresa. Un tablero resume candidatos por vacante, vacantes abiertas, *KPIs* del proceso, calendario de entrevistas y accesos rápidos para gestión diaria, incluyendo visualizaciones globales por estado y opciones de filtrado. (Ver Imágenes en anexo 16.1 en "RF49 - *Dashboard* de empresa")

RF50) Asistencias de IA para completar vacantes. Desde la creación/edición, la empresa puede autocompletar descripción, responsabilidades y educación requerida a partir de los datos ya ingresados, acelerando la redacción sin perder control editorial.

RF51) Notificaciones con top 5 por vacante. La empresa recibe alertas con los cinco candidatos más fuertes para cada vacante, facilitando foco y acción temprana sobre perfiles prioritarios.

RF52) Integración con *Calendly* para entrevistas. Tras la creación de una entrevista el sistema agenda la entrevista en el calendario de la empresa y el candidato automáticamente, validando disponibilidad de tiempo.

4.4.2.8. Suscripciones y facturación

RF53) Modelo *freemium* para empresas. Modo gratuito con límites de uso (p. ej., vacantes activas, candidatos recomendados por mes/vacante, asistencias de IA). Al alcanzar el cupo, se restringen sólo las operaciones sujetas a límite sin bloquear procesos en curso. Planes *Premium* elevan o eliminan topes y habilitan capacidades extra.

RF54) Gestión de suscripción. Desde los detalles del perfil de la empresa, acceso a planes y límites, monitorear consumo, iniciar/cambiar plan (*upgrade*/*downgrade*), cancelar/renovar. Se muestra estado (activa, prueba, pago pendiente), próximo ciclo y efectos del cambio antes de confirmar.

RF55) Pasarela de pagos y facturación. *Checkout* y portal de cliente con *Stripe*; la plataforma no almacena datos de tarjeta. Sincronización por *webhooks* para reflejar cobros/estados. Descarga de facturas en PDF, actualización de datos de facturación y avisos ante fallos de pago.

4.4.3. Perfil de candidato

4.4.3.1. Gestión de perfil

RF56) *Onboarding* y perfil profesional. El candidato completa un *onboarding* guiado donde registra datos personales y de contacto, una breve descripción profesional, educación y credenciales, idiomas, habilidades blandas y duras, expectativas salariales, intereses y experiencia laboral; incorpora imagen de perfil, enlaces de interés (por ejemplo, *LinkedIn*) y su CV en PDF. Toda esta información conforma su perfil público y alimenta los procesos de evaluación. (Ver Imágenes en anexo 16.1 en "RF56 - Onboarding y perfil profesional")

RF57) Edición continua y configuración. El candidato puede modificar su perfil en cualquier momento para reflejar cambios y mejorar su presentación. En la sección de configuración activa o desactiva "*Open to Work*", ajusta preferencias de notificaciones y administra sus credenciales de inicio de sesión, manteniendo control sobre visibilidad y privacidad.

RF58) Visualización de perfil. Desde su propia sesión, el candidato puede acceder a su perfil para verificar cómo se presenta ante las empresas. (Ver Imágenes en anexo 16.1 en "RF58 - Visualización de perfil")

4.4.3.2. *Dashboard* y visualización

RF59) Tablero del candidato. Al ingresar dispone de un *dashboard* que resume sus procesos de participación y muestra indicadores útiles: evolución y *KPIs* de desempeño, calendario de entrevistas y un resumen visual de su valoración de *matching* general para las vacantes que fue considerado, permitiendo observar áreas fuertes y oportunidades de mejora de su perfil. (Ver Imágenes en anexo 16.1 en "RF59 - Tablero del candidato")

RF60) Oportunidades y procesos activos. Desde la vista de oportunidades navega entre las vacantes disponibles y consulta sus procesos activos. En todo momento visualiza el estado de cada vacante y su posición dentro del flujo, con actualizaciones claras para sostener la trazabilidad. (Ver Imágenes en anexo 16.1 en "RF60 - Oportunidades y procesos activos")

RF61) Centro de *feedback* en la plataforma. El candidato puede ver, dentro de la aplicación, el *feedback* que le dejó la empresa en rechazos y ofertas sobre sus procesos de selección, más allá de los correos recibidos con esta información.

4.4.3.3. Sistema de *matching*

RF62) Recomendación automática de mi perfil a vacantes compatibles. El perfil del candidato se evalúa de manera continua contra vacantes activas, generando recomendaciones cuando existe compatibilidad suficiente. El cálculo se basa en las mismas categorías *Education, Technical, Soft Skills y General Information*, ponderadas según los pesos que la empresa definió para cada vacante, y sólo se consideran procesos donde el prefiltrado lo habilite (por ejemplo, modalidad/ubicación, experiencia mínima, *skills* obligatorias y expectativas salariales). El candidato puede recibir notificaciones cuando su perfil aparece entre los más relevantes y visualizar, cuando corresponda, los motivos de la compatibilidad.

4.4.3.4. Exploración de oportunidades

RF63) Detalle de vacante. Desde cualquier sección de la consola donde el candidato vea listados de vacantes o sus procesos puede acceder a los detalles de la vacante, pudiendo observar en detalles la misma, visualizando descripción, responsabilidades, requisitos, beneficios, habilidades duras y blandas comparadas contra las del candidato, compensación y demás atributos relevantes. (Ver Imágenes en anexo 16.1 en “RF63 - Detalle de vacante”)

RF64) Guardar vacantes para más tarde. El candidato puede guardar vacantes activas para leer con calma y retomarlas luego desde una sección dedicada.

RF65) Demostrar interés en una vacante. Cuando una oportunidad le resulta atractiva, el candidato puede marcar su interés explícito desde el detalle de la vacante o desde listados. La marca queda registrada y visible para la empresa en sus vistas operativas, de modo que el equipo de selección puede priorizar perfiles con señal positiva. El candidato puede desmarcar su interés en cualquier momento.

RF66) Perfil de la empresa. El candidato puede acceder a la información en detalle de la empresa autora de una vacante y revisa toda la información disponible, lo que aporta contexto sobre cultura y propuesta de valor. (Ver Imágenes en anexo 16.1 en “RF66 - Perfil de la empresa”)

RF67) *Feedback* y notificaciones. El candidato recibe *feedback* constante de sus procesos: cambios de estado, comunicaciones relevantes, por ejemplo, cuando corresponde, mensajes al ser destacado como uno de los perfiles más fuertes para una vacante o al ser rechazado, con comentarios que aportan aprendizaje y evitan silencios prolongados.

4.4.3.5. Evaluaciones

RF68) Completar y consultar *assessments*. El candidato dispone de una sección dedicada donde accede a todos los *assestments* asignados, con su estado visible y, cuando aplique, fecha límite y obligatoriedad. Puede iniciar, retomar y entregar cada evaluación, así como revisar resultados.

RF69) *Test* psicométrico del candidato. El candidato completa un *test* psicométrico que genera un informe y enriquece su perfil. El resultado puede ser consultado por la empresa cuando el candidato es recomendado para una vacante.

RF70) *Test* cultural del candidato. El candidato responde un cuestionario que modela sus valores y preferencias culturales; este se utiliza para formar un *fit* empresarial del candidato. El candidato puede consultar su perfil cultural y entender áreas de mayor y menor afinidad.

4.5. Requerimientos no funcionales

Esta sección establece las condiciones transversales que el producto debe cumplir para operar.

4.5.1. Seguridad y privacidad de datos

El acceso a cualquier recurso de la plataforma exige autenticación robusta y autorización por rol y pertenencia. Todas las APIs deben requerir un token firmado (*JWT*) con tiempo de vida acotado y renovación controlada; cuando aplique, se admite *OAuth* 2.0 como mecanismo de identidad. El control de acceso debe aplicar aislamiento lógico por organización: las empresas solo pueden acceder a datos de su propia compañía y de sus candidatos vinculados; los candidatos solo pueden acceder a su perfil y a sus procesos. Las contraseñas se almacenan mediante *hashing* seguro; la comunicación cliente–servidor se realiza sobre *TLS* y las políticas *CORS* (*Cross-Origin Resource Sharing*) son restrictivas. Los CV y datos personales se tratan bajo minimización de datos y necesidad de conocer; los logs deben omitir o anonimizar PII (información que identifica o hace identificable a una persona física; ej.: nombre, cédula, domicilio, email, teléfono, etc). El sistema expone trazabilidad de acceso (auditoría) y define políticas de retención y eliminación de datos acordes al uso interno del cliente. Asimismo, el tratamiento se ajustará a la Ley uruguaya N.º 18.331 y su Decreto 414/009 (URCDP): se informarán finalidades, base legal y responsable; se habilitarán los derechos de acceso, rectificación, actualización e inclusión/supresión. Se aplicarán salvaguardas reforzadas ante datos sensibles [2].

4.5.2. Disponibilidad

El sistema debe sostener alta continuidad, minimizando o mitigando por completo la pérdida de datos ante interrupciones. En consecuencia, la plataforma debe exponer *endpoints* de salud por servicio y, ante anomalías, retirar y reiniciar instancias no saludables. Además, el sistema debe permitir configurar políticas de respaldo y restauración, de modo que existan copias y procedimientos claros para recuperar información. Por último, las dependencias externas deben estar desacopladas: si una cae, el impacto se limita a funciones no críticas sin bloquear la operación principal; en paralelo, los procesos no críticos pueden encolarse o diferirse hasta que las condiciones se normalicen.

4.5.3. Rendimiento

En la interacción con datos y servicios, el sistema debe de mantener lecturas y escrituras ágiles: el p95 de APIs de consulta directa será ≤ 1 segundos y el de *endpoints* agregados o más demandantes será ≤ 2 segundos; en operaciones CRUD, el p95 desde el envío hasta la confirmación será ≤ 800 milisegundos. Para sostener estos objetivos, el sistema debe de aplicar caché en datos de baja rotación con invalidación clara ante cambios. Bajo la carga esperada del producto (hasta 25 usuarios concurrentes en situaciones de mucho estrés), estos tiempos deben de sostenerse sin degradaciones apreciables; por encima de ese umbral, la degradación debe de ser gradual y sin bloquear funcionalidades clave.

A su vez, el proceso de *matching* debe de ejecutarse fuera de la ruta crítica: las actualizaciones periódicas se realizarán en segundo plano. En ningún caso el *matching* debe de bloquear la navegación ni el resto de APIs de lectura y CRUD.

Finalmente, en la interfaz, el sistema debe de presentar cargas ágiles: en condiciones nominales, el p95 de las vistas estándar será ≤ 2 segundos y el tiempo hasta interacción útil en el primer ingreso será ≤ 2,5 segundos. El renderizado debe de permanecer estable durante la carga de datos para preservar la continuidad percibida y evitar bloqueos de interacción.

4.5.4. Observabilidad

Todos los servicios deben emitir logs estructurados con identificador de trazado para seguir una petición extremo a extremo y permitir correlación entre *backend*, *matching* y cualquier servicio auxiliar. Debe existir *tracing* distribuido entre componentes y exposición de métricas clave, como latencia p50/p95/p99, tasa de error y *throughput*, junto con paneles consolidados para su visualización. El sistema debe permitir configurar alertas sobre umbrales de error, latencia y volumen de llamadas, y habilitar la atribución de fallos a elementos o componentes concretos (incluyendo autoría y contexto cuando aplique), manteniendo siempre la privacidad de los datos.

4.5.5. Portabilidad

La aplicación y sus dependencias deben contenerizarse y la infraestructura definirse como código para reproducir entornos (desarrollo, *staging*, producción) de manera consistente y auditable. El sistema debe permitir levantar un entorno de desarrollo local con esfuerzo mínimo mediante contenedores, así como operar en modo híbrido (por ejemplo un *backend* de Whizdy App *on premise* contra un sistema de *matching* en la nube) a través de configuraciones declarativas. La configuración específica de entorno debe externalizarse, evitando acoplamientos a máquinas o rutas locales y *hardcodeadas*.

4.5.6. Entregabilidad (*CI/CD* y despliegue)

El proyecto debe disponer de *pipelines* automáticos por entorno con etapas de *linting*, pruebas y *build*, generando artefactos versionados y promoviendo entre entornos mediante aprobaciones controladas. El despliegue debe ejecutarse como código y proveer mecanismos de *rollback* y validaciones *post-deploy* (*health*) antes de marcar una versión como estable. El manejo de secretos no debe exponerse en repositorios ni artefactos y quedará a cargo de un sistema de gestión de credenciales integrado con los *pipelines*.

4.6. Atributos de calidad del sistema

Los requerimientos no funcionales establecen qué tan bien debe comportarse el sistema y se traducen en umbrales medibles; los atributos de calidad, en cambio, capturan la esencia de la experiencia que queremos ofrecer y por qué esa percepción es clave para que Whizdy cumpla con su verdadero propósito. Son cualidades emergentes que guían cada decisión de diseño y que, aunque no se negocian con cifras, se viven en cada interacción: la tranquilidad de saber que los datos están protegidos, la confianza de que la plataforma no fallará en el momento crítico, la satisfacción de usar una interfaz que no necesita un manual. A continuación presentamos los atributos que consideramos centrales y la razón de su peso en nuestras elecciones.

4.6.1. Escalabilidad

La escalabilidad es la capacidad del sistema para crecer, en usuarios, en carga de trabajo, en volumen de datos, etc, sin que su esencia se resquebraje y sin que el equipo deba detenerlo para reconstruirlo [36]. El equipo consideró este atributo de calidad clave para el desarrollo del proyecto porque, aunque la primera fase está pensada para el uso interno de una sola empresa, la visión del producto incluye ofrecerlo a múltiples organizaciones con cantidades de usuarios más importantes.

4.6.2. Mantenibilidad

La mantenibilidad es la capacidad del sistema para recibir nuevos requisitos, funcionalidades o ajustes, así como a nuevos desarrolladores que los implementen, sin que cada cambio reabra heridas que ya estaban cicatrizadas [36]. Es la cualidad que permite que el código se lea, entienda y modifique con confianza, preservando lo que ya funciona mientras se amplía lo que vendrá. El equipo la consideró indispensable para Whizdy porque el proyecto no termina con la entrega académica: debe seguir evolucionando para acomodar reglas de negocio futuras, integraciones no previstas y mejoras de experiencia; si el sistema no puede adaptarse sin generar regresiones, el producto muere el día de la defensa.

4.6.3. Observabilidad

Por otro lado, la observabilidad es la capacidad de mirar dentro del sistema mientras corre y comprender, sin adivinar, qué está sucediendo en cada rincón [36]: qué servicio habló con cuál, cuánto tardó, qué decisión tomó y por qué. Es la cualidad que convierte la "caja negra" en una superficie transparente, permitiendo detectar cuellos de botella, anticipar fallas y explicar comportamientos antes de que el usuario perciba algo extraño. El equipo la consideró esencial para Whizdy considerando su complejidad de requerimientos y sistema, cualquier anomalía inexplicable se vuelve incertidumbre y tiempo perdido; con la observabilidad, cada incidente deja de ser un misterio para convertirse en una oportunidad medible de mejora continua.

4.6.4. Portabilidad

La portabilidad es la capacidad de llevar el producto de un entorno a otro [36], como un entorno *cloud* distinto, *data-center* propio o laptop de desarrollo, sin reescribir líneas ni romper funcionalidad. El equipo la consideró necesaria para Whizdy teniendo en cuenta la hoja de ruta del cliente puede incluir a futuro migraciones por costó, cumplimiento normativo o simple voluntad, haciendo este proceso más sencillo y rápido.

4.6.5. Usabilidad

La usabilidad es la propiedad que reduce la fricción cognitiva entre la intención del usuario y la respuesta del sistema [36], minimizando el esfuerzo mental, la memoria intermedia y la curva de aprendizaje necesarios para completar una tarea. El equipo la consideró crítica para Whizdy porque el producto será operado por reclutadores y candidatos no técnicos que necesitan realizar flujos extensos, como completar perfiles, publicar vacantes, evaluar postulantes, sin frustración; al convertir conceptos complejos en pasos claros, prever errores y ofrecer retroalimentación inmediata, buscamos conseguir una experiencia coherente y de baja carga cognitiva que invita al uso continuo y no requiere capacitación externa.

4.6.6. Seguridad y privacidad

La seguridad y privacidad es el conjunto de propiedades que garantizan la confidencialidad, integridad y disponibilidad selectiva de la información frente a accesos no autorizados, alteraciones o filtraciones [36]. Incluye aislamiento lógico entre actores, control de acceso fino, cifrado en tránsito y en reposo, así como la minimización de superficies de ataque. El equipo la consideró el atributo fundacional de Whizdy porque el dominio maneja datos que cada usuario puede no querer compartir, como CV, expectativas salariales, evaluaciones internas, etc. Solo cuando la protección es un hecho invisible puede surgir la confianza necesaria para que candidatos y empresas compartan información crítica.

4.6.7. Disponibilidad

La disponibilidad es la propiedad que asegura que los servicios y datos están accesibles cuando el usuario los demanda [36], medida como la continuidad de operación frente a fallos parciales o totales de infraestructura. El equipo la consideró central para Whizdy porque los procesos de selección operan en ventanas de tiempo fijas, una interrupción en el servicio, aunque sea breve, desplaza al usuario hacia herramientas alternativas rompiendo la trazabilidad y la cadena de valor; la percepción de confiabilidad se construye cuando el sistema permanece funcional aunque un componente, un servicio o una zona de disponibilidad deje de responder.

4.6.8. Rendimiento

El rendimiento es la propiedad que describe la capacidad del sistema de completar operaciones dentro de límites de latencia y *throughput* que no degradan la experiencia humana, manteniendo la capacidad de respuesta bajo carga creciente [36]. Se traduce en tiempos de respuesta predecibles, procesamiento asíncrono de tareas pesadas y uso eficiente de recursos computacionales. El equipo lo consideró vital para Whizdy porque la interacción constante entre listados, filtros, formularios y resultados de *matching* exige que la interfaz se actualice más rápido que el ciclo de atención del usuario; cuando la latencia supera ese umbral el sistema es percibido como lento y la motivación de uso decae, llevando al abandono antes de que el valor del producto pueda evidenciarse.

4.7. Supuestos y restricciones

Esta sección delimita el marco en el que se desarrolló el sistema y se supone la operación del producto desarrollado. Aclarando restricciones y límites prácticos, y aclarando supuestos tomados para el diseño y validación de las funcionalidades.

Primero que nada debemos destacar una de las principales restricciones del mismo, y es que el proyecto opera en un espacio temporal acotado, ya que se dispone de un año calendario para el desarrollo, con sus inicios de la primera fase en octubre de 2024 hasta mediados de Octubre de 2025, determinado por los plazos académicos de la universidad. A su vez, el tamaño del equipo es fijo y el tiempo disponible por integrante es limitado, dado que todos los miembros del equipo realizaron el desarrollo del proyecto con actividades laborales, así como otras actividades académicas en paralelo. Estas condiciones exigen priorización estricta, iteraciones cortas y un alcance coherente con un producto viable.

Por otro lado, el equipo adoptó una política de costos mínimos, donde se privilegiaron planes gratuitos y niveles *Free Tier* en los servicios de soporte, como plataformas de despliegue, *logging*, correo, entre otros. La decisión de la adopción de esta política se basó en la evaluación de que, dado el alcance y los requisitos del proyecto, era una alternativa viable y un desafío razonable. Esta restricción condiciono cuotas, tasas y capacidades disponibles, llevando a un desarrollo de estos aspectos del sistema con esta restricción en mente. Un ejemplo claro de esta restricción se dio en el despliegue del sistema en *AWS*, donde los límites y cuotas del *Free Tier* condicionaron algunos aspectos de este. Llegando incluso a dificultar el plan original de despliegue, llevando al equipo a presentar la barrera encontrada al cliente discutiendo opciones y acordando una solución que permitiese al equipo continuar respetando dicha limitante. Las implicancias técnicas y operativas de este caso se desarrollan más adelante en el documento; aquí alcanza con señalar que la política de costos no fue abstracta, sino que impactó decisiones concretas de arquitectura y operación.

El desarrollo del sistema se realizó con el supuesto de que sería utilizado como una aplicación web orientada a escritorio. No se contemplaron aplicaciones móviles nativas ni una experiencia móvil completa; por lo tanto, la compatibilidad se limitó a navegadores modernos de *desktop*.

Otro supuesto que guió el trabajo fue el uso esperado del sistema por parte de GoGrow: como ya se mencionó, la primera fase es interna a la organización del cliente, con un volumen controlado de usuarios concurrentes y despliegues en sus propias cuentas *cloud*. Bajo estas condiciones, el dimensionamiento y la operación se mantuvieron acotados y acordes a los objetivos del producto.

Asimismo, se tomó como supuesto un modelo de ingesta por autogestión: la empresa comparte el acceso a la plataforma y cada posible candidato crea su cuenta, completa su perfil y queda disponible para los procesos de selección. En este esquema, el crecimiento de la base de candidatos es progresivo y no depende de integraciones masivas ni de campañas externas.

Por otro lado, se asumió que los usuarios cuentan con ciertos conocimientos de base: manejo básico de inglés para comprender una interfaz redactada en ese idioma; familiaridad elemental con herramientas web de uso cotidiano; y nociones mínimas del dominio por parte de ambos perfiles (candidatos y roles de RR.HH.), incluyendo conceptos como *hard skills* y *soft skills*. Por último, se exige además que todos los textos ingresados por el usuario como perfiles de candidatos y vacantes, estén redactados en inglés, dado que los modelos de *embeddings* están optimizados para su uso en inglés y esto asegura un mayor nivel de uso.

5. Proceso y Metodología

5.1. Enfoque ágil y adaptación de *Scrum*

El proyecto se desarrolló bajo una metodología ágil [4] basada en *Scrum* [1], adaptada al contexto del equipo y del proyecto, mediante trabajo iterativo, incremental y guiado por el valor aportado al cliente, e integrada con elementos de *Kanban* [3], para la visualización y gestión del flujo de trabajo. Aunque no se implementaron *sprints* formales ni reuniones diarias estándar, se mantuvieron las ceremonias clave que aportaban valor: iteraciones cada dos semanas, micro reuniones entre miembros del equipo, una reunión semanal junto al tutor, retrospectivas, demos y planificaciones al inicio de cada ciclo. Esta estructura permitió avanzar de forma incremental, validar funcionalidades en curso y adaptarse rápidamente a los cambios.

La metodología adoptada combinó la planificación y revisión típica de *Scrum* con la visualización *Kanban* del flujo de trabajo, utilizando un tablero con columnas que reflejaban el estado real de cada tarea. Este enfoque híbrido permitió mantener la flexibilidad necesaria para un equipo pequeño y un entorno de alta incertidumbre, sin perder la orientación al valor ni la trazabilidad del trabajo realizado.

El trabajo se dividió en tres etapas claramente diferenciadas. La primera, de investigación y preparación, se extendió durante los primeros meses del proyecto y fue fundamental para establecer bases sólidas. En este período el equipo relevó requerimientos, evaluó tecnologías y herramientas, investigó en profundidad inteligencia artificial y sus aplicaciones, así como el uso de *embedding* y arquitectura de *software*, y definió el alcance inicial. Además, se realizó un análisis comparativo de tecnologías y costos (licenciamiento e infraestructura), incluyendo modelos de lenguaje (*LLM*) y servicios en la nube, junto con pruebas de concepto (*PoC*) para contrastar desempeño y viabilidad antes de consolidar el *stack* definitivo. También se identificaron riesgos, se establecieron procesos de trabajo y se definieron las primeras versiones del *backlog*. Esta etapa permitió reducir significativamente la incertidumbre técnica y de negocio antes de comenzar el desarrollo propiamente dicho.

La segunda etapa, de desarrollo, inició en enero y se extendió durante aproximadamente siete meses. Fue en esta fase donde se aplicó de forma plena el enfoque ágil, con iteraciones de dos semanas, entregas parciales, validaciones con el cliente y ajustes

continuos. Durante este período se construyó la solución, se refinó el algoritmo de *matching*, se integraron nuevas funcionalidades y se estabilizó el producto.

La tercera y última etapa, de documentación y *handoff*, tuvo una duración aproximada de un mes y estuvo dedicada exclusivamente a la sistematización del conocimiento generado, la preparación de entregables finales y la transferencia del proyecto al cliente. Esta etapa fue clave para garantizar la sostenibilidad del producto una vez finalizada la participación del equipo de desarrollo.

Junto a la realización de iteraciones de dos semanas, al cierre de cada una se realizaban retrospectivas internas centradas en identificar qué funcionó, qué debía ajustarse y qué acciones concretas se implementarían en el ciclo siguiente. Por otro lado, también se llevaron a cabo demos con el cliente para mostrar avances, validar funcionalidades y recoger *feedback* temprano. Estas instancias no solo sirvieron para alinear expectativas, sino también para ajustar el rumbo del producto en función de lo aprendido.

La definición de roles fue adaptativa y colaborativa. No se establecieron roles estrictos ni se designaron figuras como *Scrum Master*. Los tres integrantes del equipo asumieron tareas de desarrollo, pruebas, revisión y gestión durante todo el proyecto. Sin embargo, con el tiempo y en función de los intereses y fortalezas de cada uno, se fueron consolidando ciertos focos de responsabilidad:

- Martín actuó como *Product Manager* y referente de *backend*, además de centrarse en realizar la evaluación de usabilidad mediante pruebas funcionales.
- Franco se encargó de la infraestructura del sistema y de la comunicación entre las dos partes del sistema, además de su rol de especialista en Inteligencia Artificial.
- Nicolás se especializó en desarrollo *frontend*, sostuvo la calidad del código y de las pruebas unitarias y se ocupó de la documentación de las *APIs*.

Estas especializaciones operaron como focos de referencia y no como límites, por lo que la ejecución se mantuvo transversal y colaborativa dentro del equipo en todo momento.

Para contar con visibilidad y medición del trabajo, y así gestionar, organizar y estimar mejor, se midieron capacidad y avance por iteración. Durante todo el proceso se utilizó Jira, que permitió estimar historias en puntos, observar la velocidad del equipo y detectar desvíos

tempranos. A través de sus tableros, se pudo analizar la distribución de tareas por estado, identificar cuellos de botella y evaluar el crecimiento del *backlog*.

Otro aspecto clave de la metodología ágil implementada se puede visualizar en la gráfica que se presenta a continuación. Esta gráfica, que abarca todas las iteraciones del proyecto, no solo refleja la cantidad de *story points* planificados contra los completados por iteración, sino que también ilustra la evolución del rendimiento del equipo a lo largo del tiempo.

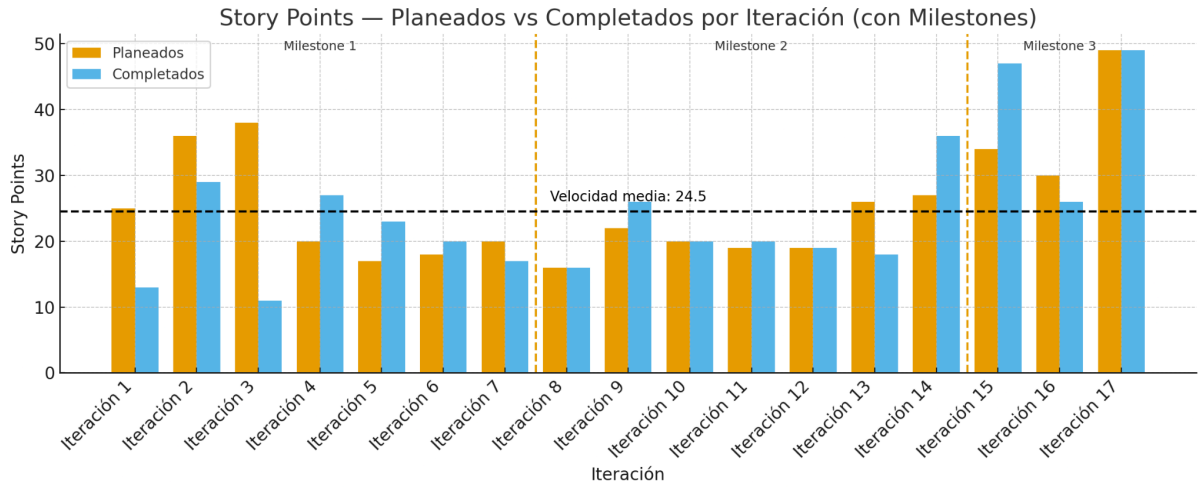


Figura 1: Gráfica de *Story Points* por Iteración

Se observan elementos clave de la etapa de desarrollo del proyecto, como al principio, durante el primer *Milestone*, el equipo experimentó dificultades para estimar correctamente la cantidad de trabajo que podía realizar por iteración. Esto se puede atribuir a varios factores, como ser un equipo nuevo, trabajar con tecnologías no antes empleadas y barreras iniciales para la colaboración, sumado a otros factores los cuales se profundizará más adelante en este capítulo, lo cual también contribuyó a la variabilidad en las estimaciones.

La gráfica también muestra cómo el equipo, utilizando procesos de metodología ágiles, fue capaz de adaptarse y mejorar su rendimiento a medida que el proyecto avanzaba. A medida que el proyecto avanzaba el equipo logró encontrar su velocidad estable, donde podía estimar con bastante precisión la cantidad de trabajo planeado contra el que se terminaba realizando. Esto refleja una mejora en el proceso ágil y en la capacidad del equipo para adaptarse a los desafíos que se presentaron

La velocidad promedio del equipo fue de 24.5 *Story Points* por Iteración, alcanzando velocidades más altas a finales de la etapa de desarrollo. Se aprecia un crecimiento considerable en la cantidad de trabajo realizado por el equipo por iteración una vez que los tres miembros del equipo dejaron de tener clase y pudieron dedicar más horas al proyecto.

La gráfica también muestra algunas caídas, como la observada al final de la penúltima iteración, debida a la última revisión del proyecto, la cual el equipo concentró su dedicación en ella. De la misma forma, se puede identificar las otras dos revisiones visualizando la gráfica, ya que permiten ver cómo la velocidad o el *throughput* del equipo disminuye en esos casos puntuales.

Esta visualización no solo muestra la evolución del trabajo realizado por el equipo, sino que también refleja cómo el uso de metodologías ágiles y las buenas prácticas implementadas ayudaron a la estabilización y progreso continuo del equipo, tanto a nivel de desarrollo como a nivel de cohesión y organización del equipo.

En conjunto con esto se registró la dedicación por tarea y a nivel general del proyecto mediante *Clockify* para mejorar la trazabilidad y ajustar la planificación a la capacidad real. (ver anexo 16.9. “Registro de horas y métricas de dedicación”)

Esta base de datos permitió desagregar el esfuerzo total de 1.845 horas en las tres fases clave: Investigación (148 horas, 8 %), Desarrollo (1.478 horas, 80 %) y Documentación (219 horas, 12 %). Dentro de la fase de Desarrollo, los 17 ciclos completaron un promedio de 87 horas por iteración; el *Milestone* 3, de sólo tres iteraciones, alcanzó 131 horas por ciclo, evidenciando el salto de disponibilidad horaria una vez que el equipo dejó de tener clase. Además, entre un 15% y un 20% de ese tiempo se destinó a actividades de gestión (planificación, priorización, manejo del *backlog*, etc.), mientras que las reuniones, como con el tutor (aproximadamente 27 horas) e internas (aproximadamente 215 horas en total), representaron alrededor del 12% del esfuerzo global. Aunque el registro fue exhaustivo, se entiende que toda esta información constituye una estimación y es posible que el esfuerzo en horas hombre sea mayor.

La combinación de *story points*, revisión de velocidad y seguimiento de bloqueos estabilizó el ritmo y habilitó decisiones informadas en cada iteración. El uso de *Jira* y su integración con *Clockify* favoreció estas prácticas ágiles al concentrar estimación, priorización y seguimiento en un mismo entorno.

En síntesis, la combinación de iteraciones ágiles, roles flexibles, uso intensivo de *Jira* y *Clockify*, junto a la integración de *feedback* continuo permitió al equipo mantener un ritmo estable, detectar desvíos tempranos y ajustar el rumbo sin perder de vista el valor entregado al cliente. Este enfoque híbrido, construido sobre la base de la investigación inicial y refinado en cada ciclo, fue clave para alcanzar un producto funcional, alineado con las expectativas y dentro de los tiempos acordados.

5.2. Planificación por *Milestones* e iteraciones

Tras definir el enfoque de trabajo ágil, el siguiente paso fue traducirlo en una planificación concreta y compartida con el cliente. Para ello, se estructuró el desarrollo en tres *Milestones* encadenados, cada uno subdividido en iteraciones de dos semanas. Esta segmentación respondía a dos objetivos: mantener un ritmo sostenible de entrega y generar puntos de validación formales con el cliente antes de avanzar a la siguiente etapa.

Cada *Milestone* contó con dos instancias formales. Por un lado, un *checkpoint* intermedio donde se presentó al cliente un informe de avance con lo realizado y lo pendiente, con foco en transparencia, *feedback* temprano y resolución de consultas. Por el otro, un cierre, donde se consolidaron resultados en un informe final y se realizó una demo para validar entregables y recoger comentarios que alimentaron el siguiente *Milestone* (Para ver en mayor detalle la información contenida en dichos informes ver anexo 16.11. “Evidencia - Informes por *Milestone*”). Durante las iteraciones se registraron notas con avances, bloqueos, acciones y aprendizajes para mantener la trazabilidad. (Para ver ejemplos concretos de las notas tomadas por iteración y por *Milestone* ver anexo 16.12. “Evidencia - Notas por Iteración” y anexo 16.10. “Evidencia - Notas por Milestone”)

Previo al inicio de los *Milestones* y la etapa de desarrollo como ya se mencionó, durante la primera etapa de investigación y preparación, el equipo realizó un importante esfuerzo en la traducción de requerimientos funcionales a historias de usuarios. Mapeando estas en un tablero de *Miro*. Primero, se identificaron las historias de usuario y se diagramaron en una gráfica de esfuerzo contra impacto (ver anexo 16.6. “Gráfica inicial esfuerzo vs impacto”), separando los elementos en cuatro sectores claros: poco esfuerzo y poco impacto, mucho esfuerzo y poco impacto, poco esfuerzo y mucho impacto y por último mucho esfuerzo e impacto. Este ejercicio permitió al equipo priorizar visualmente qué elementos eran más críticos para el éxito del proyecto.

Además, se creó un diagrama de priorización que asignaba una prioridad del 1 al 5 a cada historia de usuario, lo que se tradujo más adelante en las prioridades de *Jira (Lowest, Low, Medium, High, Highest)* (ver anexo 16.7. "Primera priorización de historias de usuario"). Finalmente, se desarrolló un diagrama que asignaba las historias de usuario en los diferentes *Milestones*, teniendo en cuenta su prioridad y el alcance de cada hito (ver anexo 16.8. "*Roadmap* inicial").

Estas herramientas visuales no solo ayudaron al equipo a organizar y priorizar su trabajo, sino que también fueron compartidas con el cliente para su validación y aprobación. Esto permitió al equipo y al cliente estar alineados en cuanto a las expectativas y al ritmo de desarrollo. Cabe destacar que, a lo largo de los *Milestones*, el equipo mantuvo la coherencia con la planificación inicial, a pesar de las agregaciones y cambios en el *backlog*. Cada carga que se ve en los diagramas son los requerimientos funcionales traducidos en historias de usuario. Aunque estas fueron las primeras, posteriormente se evolucionaron por diferentes razones, como nuevos requerimientos, cambios, reescritura de *tickets*, descubrimiento de nuevas funcionalidades, mejoras, etc. Esto refleja simplemente lo que se hizo inicialmente y fue lo que se validó por el cliente.

En conjunto, cada *Milestone* se trató fiel a lo que fue el proyecto, fuera de las agregaciones, cambios y expansiones del *backlog* que se hayan realizado, dado que en el arranque de cada *Milestone* se realizó una re-priorización del contenido y una re-estimación en equipo de los ítems a abordar.

Fue entonces que la fase de desarrollo comenzó en enero, luego del trabajo previo de definición de requerimientos, alcance, tecnologías y arquitectura.

El *Milestone* 1 tuvo una duración aproximada de tres meses, siendo una primera instancia de desarrollo y de adaptación y se enfocó en establecer la base del sistema, adaptar el *stack* y construir una primera versión del algoritmo de *matching*.

El *Milestone* 2, también de tres meses, profundizó el *matching* y la gestión de vacantes, refinando funcionalidades existentes e incorporando nuevas capacidades priorizadas.

El *Milestone* 3, de un mes y medio, se centró en validaciones del sistema, despliegue y cierre del producto.

5.3. Documentación continua y aprendizaje reflexivo

Una vez establecidos los hitos de entrega, se requiere un mecanismo que garantizara la memoria del proceso y que alimentara la mejora continua. Para ello se implementó una práctica de documentación reflexiva en paralelo al desarrollo.

Una práctica que acompañó todo el ciclo de vida del proyecto fue la documentación continua del proceso. Esta no se limitó a la entrega final, sino que se fue construyendo en paralelo al avance del trabajo. No solo se registraron notas al cierre de cada iteración o *Milestone*, sino que también se fueron documentando en tiempo real las decisiones técnicas y de negocio que se tomaban durante la etapa de investigación inicial, los cambios de alcance que se acordaban con el cliente, y los aprendizajes surgidos de cada reunión con la *recruiter* o con el tutor. (Para ver ejemplos y evidencia de esta documentación continua ver anexo 16.13. "Evidencia - Documentación continua" y anexo 16.15 "Evidencia - Evidencia generación y documentación de procesos del proyecto").

De esta forma, cualquier ajuste en los requerimientos, nueva funcionalidad, cambios en alcance o decisión arquitectónica quedaba asentada en documentos internos, lo que permitió mantener una trazabilidad clara de por qué se había tomado cada camino, y facilitó más adelante la recopilación de información para la documentación final sin tener que reconstruir el recorrido.

Al cierre de cada iteración y de cada *Milestone*, estas notas se complementaban con métricas concretas: cantidad de *story points* completados, número de *tickets* cerrados, desafíos encontrados, aprendizajes obtenidos y acciones a implementar en el siguiente ciclo.

Esta documentación interna sirvió como herramienta de reflexión y mejora continua. Permitió al equipo detectar patrones de retraso, identificar cuellos de botella, ajustar dinámicas de trabajo y mantener una memoria institucional del proyecto. Además, fue clave para justificar ante el cliente ciertos cambios de ritmo o priorización, ya que se podía respaldar con registros históricos concretos.

Esta práctica también facilitó la transición hacia la etapa final de documentación y *handoff*, ya que gran parte del contenido necesario para los entregables ya estaba sistematizado. De esta forma, lo que podría haber sido una carga adicional al final del proyecto se convirtió en una construcción gradual y sostenida de conocimiento.

5.4. Gestión del *backlog*

El *backlog* se gestionó en *Jira*, que brindó visibilidad del trabajo en curso y del planificado. Se utilizaron historias de usuario para representar valor de producto, a su vez, se utilizaron épicas para agrupar conjuntos de historias bajo objetivos funcionales mayores, como por ejemplo “Gestión de Vacantes” y “Gestión de Candidatos”, mejorando la trazabilidad entre el *roadmap* y el trabajo diario, también se manejaron tareas para actividades internas y de investigación, *bugs* para correcciones y mejoras para optimizaciones detectadas durante el desarrollo. La estimación se realizó en *story points* con la secuencia de Fibonacci, mediante *planning poker* integrado a *Jira*. Cada integrante votaba en simultáneo y de forma oculta, y luego se discutían las diferencias para llegar a un consenso. Este mecanismo permitió obtener una medida unificada de esfuerzo por *ticket*, anticipar riesgos y planificar la capacidad de cada iteración con mayor precisión.

En el acumulado del proyecto se administraron más de 200 *tickets* entre historias de usuario, tareas, mejoras, *bugs* y épicas, lo que brindó una base objetiva para la planificación y el control del avance.

Cada *ticket* mostraba de forma explícita su prioridad, la estimación en *story points*, la persona asignada y el autor, y transitaba por un flujo claramente definido:

El flujo fue el siguiente:

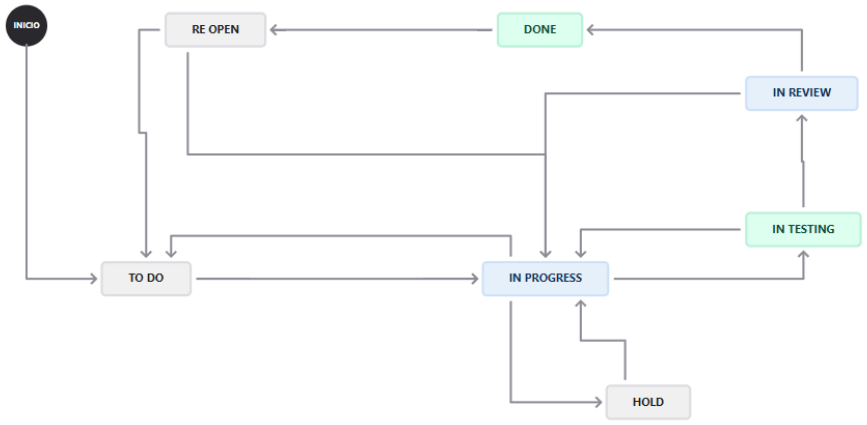


Figura 2: Diagrama flujo Tickets Jira

- *To Do*: ítem pendiente planificado para la iteración.
- *In Progress*: ítem en ejecución.

- *In Review*: con *PR* abierto y en proceso de revisión, con verificación de *pipelines* de integración continua.
- *Testing*: en verificación funcional y de aceptación.
- *Done*: completado, con evidencias y, cuando correspondía, preparado para demo o despliegue.
- *Hold*: estado no lineal utilizado cuando el ítem quedó en espera por definición, dependencia o validación externa. Desde *Hold* el ticket retornaba al estado que correspondiera una vez resuelto el impedimento.

El trabajo se visualizó en un tablero *Kanban* cuyas columnas reflejaron el flujo descrito. Se utilizó el esquema de prioridades de *Jira* (*Lowest*, *Low*, *Medium*, *High*, *Highest*). La combinación de prioridad, estimación y dependencias definió el orden de toma en cada iteración, manteniendo visible qué abordar primero y por qué.

Como práctica de control de versiones se aplicó un flujo tipo *GitFlow*: ramas estables *main* y *develop*, y ramas de trabajo por *ticket* con el patrón WZY-123-titulo-resumido-en-inglés [28]. Cada cambio se desarrolló en su rama y se integró mediante *Pull Request* hacia *develop*, con revisión entre pares y verificación de *pipelines*. En los cierres de *Milestone* se consolidó *develop* en main, manteniendo la trazabilidad. Con cada despliegue se registró un *release* en main que significó la liberación estable de código.

Profundizando en los pasos realizados para llevar cambios a *develop* desde una rama de desarrollo, cada *Pull Request* requirió al menos una aprobación antes del *merge*; para cambios de mayor impacto (módulos críticos, seguridad o decisiones de arquitectura) se solicitó la aprobación de dos revisores. Los revisores dejaban comentarios y sugerencias directamente en el *PR* y el autor iteraba hasta resolverlos.

El *pipeline* de *GitHub Actions* estaba compuesto por tres elementos clave: la ejecución completa de todos los *tests* del sistema, el build de la aplicación y la validación del cumplimiento de las reglas de *linting*. Este conjunto de verificaciones se ejecutaba automáticamente con cada *Pull Request* abierto, lo que garantiza que cualquier cambio propuesto fuera evaluado en tiempo real antes de ser integrado a la rama principal. Además, con cada *commit* realizado sobre una rama en desarrollo, se ejecutaban también los *tests*, lo que permitió detectar errores o fallos de forma temprana, sin esperar a la apertura del *PR*.

Esta configuración reforzó la cultura de calidad dentro del equipo, ya que cualquier cambio que rompiera el *pipeline* era inmediatamente visible y debía ser corregido antes de continuar.

Solo cuando el CI se encontraba en estado verde, es decir, cuando todas las verificaciones pasaban sin errores y se contaba con las aprobaciones requeridas, el *PR* era mergeado. Esta automatización no solo evitó la introducción de regresiones, sino que también aceleró la integración entre ramas, al reducir la fricción manual y aumentar la confianza en cada cambio que se incorporaba al proyecto.

Además, los *commits* siguieron un prefijo normalizado que indicaba la naturaleza del cambio: *ADD*, *UPDATE*, *DELETE* o *REFACTOR*, seguido de una breve descripción. Esta convención facilita la lectura del historial y el enlace con los *tickets* de Jira.

Como práctica central se aplicó el *backlog grooming* o refinamiento del *backlog*. En estas sesiones se reescribieron *tickets* para que quedaran claros, independientes y de tamaño adecuado; se separaban ítems demasiado amplios; se ajustaban descripciones que no reflejaban el trabajo real; y se ordenaban según dependencias y prioridad.

Durante el *Milestone* 1 el *grooming* se realizó al finalizar cada iteración, debido al volumen de definiciones en movimiento propio del arranque. Al cierre de este *Milestone* se efectuó un *grooming* amplio para ordenar y poner al día el *backlog*, con adición de *tickets* y propuestas de ajuste de alcance y prioridades presentadas al cliente en las instancias de cierre. A partir del *Milestone* 2 se consolidó un sistema de *grooming* más completo al final de cada *Milestone*, donde se proponían cambios de prioridad o alcance, se validaban con el cliente y quedaban registrados para su incorporación al *backlog* y al *roadmap*.

La *Definition of Ready* ordenó la entrada de trabajo: cada historia debía estar descrita con claridad, incluir criterios de aceptación, tener dependencias identificadas y una estimación en story points, y, si correspondía, detalles de implementación.

La *Definition of Done* ordenó la salida: *Pull Request* con la funcionalidad aprobada por al menos un revisor, integración continua en verde, reglas de calidad estática cumplidas, pruebas unitarias actualizadas y documentación adjunta si era requerida.

El proceso de ingesta de nuevos *tickets* se formalizó así: a medida que surgían *bugs*, mejoras, nuevas funcionalidades solicitadas por el cliente o ajustes derivados de

descripciones incompletas, los ítems se redactaron de forma breve primero en un archivo compartido para que el equipo revisara su validez y estandarizara su escritura. Luego se documentaron en *Jira*, se priorizaron y se ubicaron en el *roadmap* y en la iteración correspondiente. Cuando el cambio afectaba alcance o prioridades, se presentaba y discutía con el cliente en las instancias establecidas (*demos* o *checkpoints*). De esta forma, la gestión del *backlog* integró un refinamiento continuo, criterios de calidad claros y una ingesta controlada, manteniendo alineada la ejecución con las necesidades del producto.

5.5. Gestión de cambios y comunicación

La comunicación con el cliente, como ya se mencionó, se dio en *checkpoints* y cierres de cada *Milestone*, así como en demos de validación. Entre esas instancias se realizaron intercambios puntuales por *Slack* para consultas de ambas partes y para seguimiento. Durante el *Milestone* 3 se efectuaron despliegues al cierre de cada iteración hacia un ambiente de *QA*, al cual se compartió acceso con el cliente para validaciones tempranas; además, se proveyó un documento de *QA* para canalizar reportes de errores, mejoras y consultas. En la fase final solo se registraron consultas, sin cambios, mejoras ni errores a incorporar.

La relación de trabajo fue cercana y colaborativa: aunque el cliente tenía una visión clara del producto, mantuvo apertura al intercambio y a incorporar propuestas del equipo, lo que permitió co-construir la solución y alinear cada decisión con el valor de negocio esperado.

Los cambios de alcance o prioridad se proponían y acordaban en esas mismas instancias con el cliente (*checkpoint*, cierre o *demo*), ya fuera a partir del refinamiento del *backlog* o por solicitudes del propio cliente. Se presentaban con una síntesis de cambio que incluía impacto, prioridad y próximas acciones, y se incorporaban al plan de trabajo, manteniendo la trazabilidad entre solicitud, análisis, acuerdo y ejecución.

Dentro del equipo, la coordinación diaria se mantuvo por *WhatsApp* y *Discord*, con micro reuniones a lo largo de la semana para resolver dependencias, cerrar definiciones y atender necesidades puntuales de planificación, gestión o estimación, sin esperar al cierre de la iteración. Estas instancias resultaron especialmente útiles cuando algún integrante se enfrentaba a una dificultad inesperada y requería apoyo de otro miembro.

La coordinación con el tutor se sostuvo mediante una reunión semanal orientada a revisar entregables, despejar dudas y, cuando correspondía, orientar ajustes de proceso o de alcance para mantener el enfoque del proyecto.

5.6 Aplicación de la Metodología

A continuación parece oportuno el ejemplificar una de las situaciones en las cuales se encontró el equipo durante la realización del proyecto la cual se tuvo que abordar y resolver mediante y a través del uso de la metodología y parece un ejemplo claro del uso y surgimiento de la misma en un punto clave y decisivo del proyecto.

Se ampliará el tema *scope creep* al finalizar el *Milestone* 1. Partiendo de la definición de *scope creep* como la expansión no controlada del alcance del proyecto, más allá de lo originalmente pactado, sin que esto esté acompañado de un ajuste equivalente en tiempo, recursos o priorización.

Como ya mencionamos se dividió el proyecto en fases, siendo una de ellas el desarrollo, y esta misma en *Milestones* e iteraciones. Fue entonces durante las primeras iteraciones del *Milestone* 1, que el equipo se empezó a encontrar con múltiples dificultades, algo que por supuesto es esperable considerando que eran las primeras iteraciones de desarrollo de un nuevo proyecto y equipo.

Sin embargo el equipo se estaba encontrando con grandes dificultades a la hora de estimar y realizar tareas o *tickets*, ya que estos, a pesar del trabajo realizado inicialmente, parecían presentar problemas persistentes: se identificó que muchas historias de usuario estaban mal definidas, no eran independientes, eran demasiado amplias o demasiado genéricas y en varios casos ocultaban trabajo no visible a simple vista. Esto dificultaba tanto la estimación como la distribución de tareas dentro del equipo.

Por otro lado, como ya se mencionó, había surgido esta nueva versión expandida de diseño de *Figma* del cliente, y el equipo se empezó a encontrar con gran cantidad de nuevos escenarios, casos, cambios e incluso nuevas funcionalidades. Lo que estaba llevando a un crecimiento y cambio acelerado y continuo del *backlog*.

Fue entonces que, en las retrospectivas de cada iteración, comenzó a surgir una sensación creciente: el trabajo pendiente avanzaba más rápido que la capacidad del equipo para absorberlo, desalineando el progreso real con lo planificado inicialmente.

En ese momento la metodología con la que se trabajó durante el proyecto empezó a tomar forma: se consolidaron sesiones de *backlog grooming* tras cada ciclo, la toma sistemática de notas y la búsqueda activa de aprendizajes, dando lugar, entre otras cosas, al proceso de ingesta de cambios.

Sin embargo, a pesar del esfuerzo extra del equipo para poder de cierta forma sincronizar lo que eran los requerimientos, casos e ideas plasmadas por el cliente en el *Figma* con nuestro *backlog*, sumado a las problemáticas encontradas en algunos de los *tickets* generados inicialmente, más la suma del trabajo que ya de por sí era necesario para continuar con el desarrollo del sistema y poder seguir con el *roadmap*, el *backlog* parecía estar creciendo cada vez más, así como el alcance del proyecto, superando la velocidad de realización y finalización de los *tickets*.

Y esto se respalda con datos claros, porque lo que el equipo empezaba a sentir en la retrospectiva, lo podía ver en todo momento en *Jira* en el siguiente diagrama de flujo acumulado.

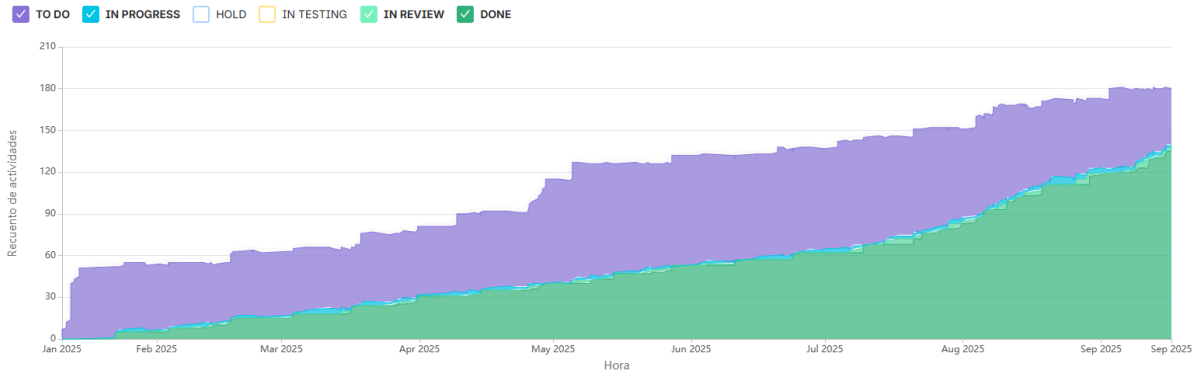


Figura 3: Gráfica crecimiento *Backlog*

Donde el *backlog*, durante el primer *Milestone* 1, crecía más rápido que la curva de velocidad. Mostrando una línea que no sólo ascendía, sino que se aceleraba. Cada iteración dejaba un salto visible, producto de las sesiones de *backlog grooming* y la ingesta constante de nuevos *tickets*: *bugs*, mejoras, reescrituras, funcionalidades emergentes del nuevo *Figma*.

El equipo veía esa línea en tiempo real, la tendencia seguía siendo clara: el *backlog* crecía más rápido de lo que se podía cerrar.

Esto se convirtió en la evidencia objetiva de algo que las retrospectivas ya habían empezado a intuir: no alcanzaba simplemente con trabajar más, se necesitaba redefinir en qué trabajar.

Fue así que se decidió, al cierre del *Milestone* 1, realizar un *backlog grooming* general: no solo una revisión de *tickets*, sino una revisión completa del alcance del proyecto.

Se reescribieron historias, se depuraron *tickets*, se agregaron criterios de aceptación claros y se estableció un orden estructurado que respetara dependencias entre tareas. Se añadieron comentarios de implementación, se introdujeron tareas de investigación donde faltaba información y se aseguró de que cada requerimiento, funcionalidad y caso de uso presente en el *Figma* del cliente quedara reflejado y detallado en el *backlog*.

Pero, como se mencionó al inicio, un *scope creep* solo puede ser absorbido si se acompaña de un ajuste equivalente en tiempo, recursos o priorización.

En este caso, ni el tiempo ni los recursos eran negociables: el proyecto tenía una duración fija y un equipo acotado. La única variable fue revisar el alcance.

Fue así que el equipo, en conjunto con el cliente, llegó a la conclusión de que el proyecto debía enfrentar una re-priorización crítica y una evaluación del alcance para los *milestones* restantes.

Con esta información, el equipo elaboró un informe formal de repriorización, que se presentó al cliente durante el cierre del *Milestone* 1 en conjunto con el informe final del *Milestone* (ver anexo 16.14. “Documento de repriorización del *backlog* - Cierre Milestone 1”). En este documento se detalló el estado actual del *backlog*, se justificó técnica- y estratégicamente la necesidad de ajustar el alcance, y se propuso despriorizar ciertos requerimientos funcionales que, aunque eran valiosos, no eran críticos para el proyecto. Indicando razones y alternativas para estos. Tomando siempre como guía para la elección de estos elementos el valor que se le aportaba al cliente con cada funcionalidad.

Las funcionalidades afectadas fueron:

- **Multicuentas para empresas**: Se consideró que una cuenta administradora compartida cubría por completo las necesidades de la fase interna; el multiusuario aportaba valor futuro, pero no diferencial para el momento.
- **Pasarela de pagos y versión *premium***: Dado que el producto se orientaba a uso interno, la monetización quedaba fuera del horizonte inmediato. Integrar una pasarela implicaba un esfuerzo de seguridad, cumplimiento y mantenimiento que no se iba a reflejar en valor visible hasta una segunda fase.
- **Psicometría y evaluaciones técnicas**: El sistema no buscaba ser un *all-in-one* de reclutamiento, sino especializarse en *matching*. Desarrollar estos módulos desde cero requeriría investigación y diseño de los *tests*, validaciones y desarrollo extendido de funcionalidades.

Todas estas funcionalidades contaban con alternativas viables que no comprometían la experiencia *core* del producto. Por ejemplo, las evaluaciones técnicas podían realizarse durante las entrevistas y sus resultados cargarse manualmente en el sistema.

Esta propuesta fue presentada al cliente no como una reducción de alcance, sino como una estrategia de enfoque: concentrar los esfuerzos en el los elementos diferenciales del sistema, el algoritmo de *matching* inteligente, la gestión de vacantes y la experiencia de usuario y dejar para una segunda versión todo aquello que en el momento no era crítico pero podía integrarse más adelante. El cliente validó la propuesta tras una breve sesión de preguntas, aceptando que aquellas funcionalidades quedarán en el *backlog* para una segunda versión del producto.

Como resultados, llegamos a un producto que sí se podía entregar, donde el equipo logró:

- Enfocar el esfuerzo en un producto funcional, probado y desplegado, sin dispersar recursos en funcionalidades no críticas.
- Mejorar la calidad del *backlog*: *tickets* más pequeños, claros, estimables y con criterios de aceptación definidos.
- Consolidar una relación de confianza con el cliente, basada en transparencia, comunicación temprana y toma de decisiones conjunta.

Por otro lado, también se observan otros dos grandes resultados visibles en elementos ya mencionados: la gráfica *Story Points* Planeados contra Completados y el diagrama de *Backlog*.

Se logró estabilizar la estimación: a partir del primer *Milestone*, la curva de velocidad por iteración deja de ser una línea errática y empieza a mostrar una capacidad predecible del equipo, reflejo de que se había aprendido a estimar con mayor precisión lo que realmente se podía cerrar en dos semanas.

Y, en paralelo, se consiguió que el tamaño del *backlog* dejara de crecer de forma abrupta; tras el *grooming* general, se ve en el diagrama que, si bien el *backlog* siguió creciendo por *bugs* o ajustes menores, ya no era una avalancha, sino un flujo controlado y manejable.

A nivel de alcance, tras la repriorización, el proyecto se acotó a los requerimientos funcionales ya descritos, con la única excepción de los requerimientos funcionales; 5, 12, 13, 36, 37, 44 a 48, 52 a 55, 59, 61, 67, 68 y 70. Estos RF, ya documentados y especificados en historias de usuario, quedaron fuera del entregable final. Mencionar también que algunos elementos extra tampoco llegaron a ser implementados, como por ejemplo, la integración con *Calendly* y la plataforma de *feedback* interno, los cuales son elementos fueron repriorizados en una instancia posterior y mucho más acotada, luego del *Milestone* 2, dado que contaban con alternativas simples y no bloqueaba la experiencia central del producto ni restaba valor al mismo. Igualmente, destacar como incluso tras estas omisiones, el sistema consolidó más de 50 requerimientos funcionales en la versión entregada.

Este episodio demostró, en la práctica, cómo la metodología no fue un conjunto de rituales, sino una respuesta a una tensión real. *Backlog grooming*, retrospectivas, documentación continua, visualización de datos y validación con el cliente dejaron de ser "ceremonias" para convertirse en herramientas que permitieron detectar un *scope creep* antes de que se volviera crítico, ajustar el rumbo y entregar un producto robusto, completo y alineado con las expectativas.

6. Gestión de Riesgos

En esta sección se hablará sobre algunos riesgos que el equipo identificó tanto al inicio del proyecto en esta primera fase de investigación sobre las herramientas, procesos y distintos aspectos sobre el desarrollo del producto, como durante la fase de desarrollo del proyecto. Estos riesgos están basados en supuestos técnicos y operativos y no solo orientaron decisiones tempranas, sino que también servirán como insumo para la continuidad del desarrollo, ya sea como prevención o como base para planes de contingencia.

La identificación de riesgos se realizó de forma colaborativa durante la etapa inicial del proyecto. El equipo reunió información técnica y operativa a partir del alcance definido, las tecnologías seleccionadas y el cronograma estimado. Mediante un proceso de lluvia de ideas estructurado, cada integrante aportó posibles amenazas desde su área de conocimiento, las cuales fueron agrupadas y analizadas en conjunto.

Para la estimación de probabilidad e impacto, se utilizó una escala cualitativa (baja, media, alta), consensuada a partir de la revisión de referencias externas y de experiencias previas en proyectos académicos y laborales. La clasificación final se validó de manera colectiva, garantizando que las prioridades reflejaran un criterio compartido y no individual. Además, para cada riesgo se documento estrategias de mitigación específicas para cada caso.

6.1. Riesgos iniciales

R1) Dependencia de servicios externos
Probabilidad: Baja
Impacto: Alto

El uso de servicios externos como *Gemini*, *Google Credentials*, *Qdrant*, *Stripe*, *OpenAI*, *Google Gmail*, entre otros pueden generar problemas si estos servicios presentan indisponibilidad, cambios de políticas/precios, límites de uso (*rate limits*) o modificaciones de APIs/SDKs. Además, una integración incorrecta o inestable puede interrumpir funcionalidades críticas de nuestra plataforma.

Respuestas al riesgo:

- Diseñar un sistema desacoplado que permita reemplazar servicios externos rápidamente en caso de fallos o cambios en nuestra plataforma o servicio.
- Implementar cada integración de modo que, si el servicio falla, solo se vea afectada la funcionalidad concreta que lo usa y el resto del sistema siga operando normalmente.
- Mantener actualizadas las dependencias y realizar pruebas regulares para verificar la compatibilidad.
- Estudiar las políticas y condiciones de uso de estos servicios, prestando atención a posibles cambios.

Evaluación: Durante el desarrollo no se enfrentaron grandes problemas frente a la implementación de servicio externos, a pesar de tener gran preocupación sobre ellos y de la dificultad que tendría implementarlos.

R2) Complejidad en el algoritmo de *matching*
Probabilidad: Media
Impacto: Alto

La construcción del algoritmo de recomendación/*matching* puede resultar más compleja de lo previsto y derivar en resultados imprecisos o irrelevantes si no se elige adecuadamente la combinación de datos, *features*, *embeddings*/*LLMs* y reglas de negocio; esto se agrava por brechas de conocimiento específico y por la dificultad de validar objetivamente la calidad del *matching* en contextos reales (datos ruidosos, sesgos, vacantes

heterogéneas y señales débiles), lo que podría afectar la satisfacción de usuarios y la adopción del producto.

Respuestas al riesgo:

- Investigar posibles implementaciones, tecnológicas y técnicas a usar.
- Diseñar y probar prototipos tempranos para evaluar la viabilidad técnica.
- Consultar con expertos en *LLM* y utilización de IA para obtener mejores enfoques.
- Trabajo iterativo de análisis y mejora sobre el algoritmo de *matching* para favorecer una mejora continua
- Realizar etapa de validación utilizando datos reales.

Evaluación: El algoritmo de *matching* tuvo un proceso largo, pasando por una fase de investigación, una *PoC* y un extenso proceso de desarrollo, con múltiples iteraciones, y evaluación constante. Gracias a definir este riesgo y poner énfasis en la investigación previa necesaria se pudo llegar a una versión estable y precisa del algoritmo de *matching*.

R3) Gestión de datos
Probabilidad: Alta
Impacto: Alto

La plataforma procesa y almacena datos personales de candidatos y datos sensibles de empresas; cualquier falla de seguridad, error de configuración, uso indebido o tratamiento inadecuado (incluido el intercambio con terceros como proveedores de IA) pudiendo provocar exposición, pérdida o alteración de información sensible.

Respuestas al riesgo:

- Adoptar estándares de seguridad y reglas/leyes de manejo de datos.
- Implementar encriptación de datos.
- Es importante considerar el manejo adecuado de los datos compartidos con servicios externos, como *OpenAI*, que utilizaremos para procesar información sensible de candidatos y empresas. Esto incluye garantizar la seguridad, privacidad y cumplimiento normativo en el tratamiento de estos datos.
- Limitar el acceso a datos sensibles únicamente a sistemas y personas autorizadas.

Evaluación: Este es un riesgo que continúa vigente, si bien se llevaron acciones para evitar exponer información sensible como por ejemplo utilizar *hash* para el manejo de contraseñas, manejo de *CORS* y aislamiento lógico de la información. Sin embargo, este riesgo será evaluado con mayor precisión cuando la aplicación esté productiva.

R4) Gestión de pagos (pasarelas de pago y suscripciones)
Probabilidad: Media
Impacto: Medio

La integración con pasarelas de pago y el manejo de suscripciones puede fallar por indisponibilidad del proveedor, errores en *webhooks*, rechazos bancarios, cambios de *API* o configuración incorrecta, generando pérdidas financieras (cobros no registrados o duplicados), interrupciones de suscripciones o mala experiencia del usuario.

Respuestas al riesgo:

- Evaluar uso e implementación de pasarelas confiables y ampliamente usadas como por ejemplo *Stripe*.
- Desarrollar y probar en entornos de prueba con cuentas de desarrollo, asegurando el correcto funcionamiento especialmente en estos requerimientos.
- Tener reacciones planificadas ante escenarios donde se pudiese perder dinero para cualquiera de las dos partes.

Evaluación: No se llegó a implementar la pasarela de pagos por cambios en los requerimientos del sistema, podría verse como un riesgo para cuando se vaya a implementar la misma en un siguiente etapa del producto.

R5) Cambios inesperados en los requisitos del proyecto
Probabilidad: Alto
Impacto: Alto

Los requisitos pueden variar por decisiones del cliente o por factores externos fuera de nuestro control como por ejemplo, aparición de nuevos requerimientos funcionales o cambios de requerimientos existentes . Estos cambios, especialmente cuando ocurren en una etapa avanzada del proyecto, impactan en la planificación, en la calidad y en el alcance, pudiendo comprometer plazos y calidad del producto.

Respuestas al riesgo:

- Aplicar metodologías ágiles para manejar cambios de manera iterativa y ajustable.
- Establecer un proceso claro de evaluación y priorización de cambios junto con los clientes.
- Documentar los cambios de alcance y comunicarnos con el cliente.
- Realizar trabajo constante sobre la metodología de trabajo utilizada para poder responder a estos cambios de forma rápida y organizada, adaptando los procesos del equipo y la forma de trabajo de ser necesario.
- Reservar tiempo en la planificación para manejar posibles ajustes en los requisitos.

Evaluación: A lo largo del desarrollo del proyecto tanto los requerimientos como el alcance fueron variando, algo que ya se discutió en profundidad previamente. Sin embargo el equipo logró manejar de forma correcta esta situación y gracias a una respuesta temprana y conocimiento de la posibilidad del surgimiento de este riesgo, se logró sobrepasar, alcanzando un flujo fluido, bien estructurado y explicativo.

R6) Conocimiento de las tecnologías que trabajamos

Probabilidad: Media

Impacto: Alto

La falta de experiencia profunda en tecnologías clave a utilizar, por ejemplo, IA, *Next.js* y pasarelas de pago podría retrasar el desarrollo o comprometer la calidad del producto.

Respuestas al riesgo:

- Procurar la capacitación del equipo en estas tecnologías clave durante las primeras etapas del proyecto.
- Utilizar herramientas y *frameworks* bien documentados y con comunidad activa.

Evaluación: Se ve reflejado en la velocidad del equipo durante el desarrollo como al comienzo del proyecto la experiencia en las tecnologías era una limitante del mismo, pero apoyándose entre los miembros del equipo y con documentación disponible sobre la misma se pudo tener una curva de aprendizaje exitosa.

R7) Conocimiento de *scouting*

Probabilidad: Media

Impacto: Medio

La falta de experiencia específica en procesos de reclutamiento y selección puede traducirse en un diseño inadecuado de funcionalidades clave (p. ej., *match* cultural, validación de habilidades, señales de *seniority*), lo que deriva en recomendaciones poco útiles, sesgos no detectados, flujos confusos para *recruiters* y métricas de éxito que no reflejan la realidad del proceso (*pipeline*, entrevistas, ofertas).

Respuestas al riesgo:

- Colaborar con especialistas en *recruiting* y en recursos humanos para definir funcionalidades relevantes.
- Implementar un sistema de retroalimentación para mejorar continuamente la plataforma según el uso real.

Evaluación: Esto era una preocupación grande del equipo de desarrollo, porque para poder llevar a cabo el desarrollo eficientemente es necesario entender la lógica de la misma. Se realizaron entrevistas con *recruiter* y reuniones con el cliente que tenían un contexto amplio sobre *scouting* y además se realizaron validaciones con la *recruiter* y posibles usuarios de la plataforma. Con esto se pudo entender correctamente la lógica del proceso.

6.2. Riesgos futuros

A continuación se incluye el conjunto de riesgos que fueron identificados fuera del alcance del presente desarrollo. Estos escenarios no fueron considerados en el plan de mitigación inmediato, ya que su materialización está asociada a etapas posteriores del ciclo de vida del producto, como su escalamiento, operación sostenida o inserción en un contexto productivo real. Su documentación busca servir como insumo para futuras iteraciones o equipos de mantenimiento, facilitando la anticipación de desafíos técnicos y de negocio una vez que el sistema trascienda el estado de validación académica.

R8) Escalabilidad y performance del *matching* en producción

Probabilidad: Media
Impacto: Alto

Al crecer el volumen de vacantes, candidatos y consultas simultáneas, el *pipeline* (*embeddings*, búsqueda vectorial, valores del *matching*) puede volverse lento o inestable, afectando tiempos de respuesta y experiencia de usuario.

Respuestas al riesgo:

- Ver la posibilidad de escalar el componente de *matching* de manera dinámica dependiendo del tráfico.
- Evaluar constantemente el rendimiento y capacidad de la plataforma.
- Continuar probando el sistema de *matching* para optimizarlo y mejorarlo.

R9) Seguridad sobre entradas y comunicación en *matching*

Probabilidad: Media
Impacto: Alto

Vulnerabilidades en la validación de entradas y canales de comunicación del sistema de *matching* pueden ser puntos de ataque que comprometan la integridad de las recomendaciones y expongan información confidencial. Los atacantes pueden explotar campos de entrada no validados en CVs, descripciones de trabajo y comunicaciones entre

usuarios para inyectar contenido malicioso, manipular algoritmos de *matching*, o interceptar datos sensibles durante el proceso de emparejamiento candidato-empresa.

Respuestas al riesgo:

- Aplicar sanitización robusta de *inputs* en campos de texto libre o carga de archivos y detección de anomalías en contenido de CVs y perfiles.
- Cifrar todas las comunicaciones entre usuarios utilizando protocolos seguros de extremo a extremo y aplicar *rate limiting* en comunicaciones para prevenir *spam*.
- Establecer detección automática de patrones anómalos en el comportamiento de *matching* y comunicaciones y alertas en tiempo real para actividades sospechosas como intentos de acceso no autorizado a perfiles.

R10) Costos de operación de IA e infraestructura

Probabilidad: Alta
Impacto: Alto

El crecimiento acelerado en el uso de servicios de IA y almacenamiento vectorial puede generar costos operativos inesperados. Los costos pueden dispararse debido al uso intensivo de *APIs* de modelos de lenguaje para procesamiento de CVs y generación de contenido, almacenamiento de *embeddings* a gran escala, y procesamiento de *matching* en tiempo real.

Respuestas al riesgo:

- Implementar alertas automáticas cuando el gasto en servicios de IA supere umbrales predefinidos por período.
- Aplicar monitoreo granular de costos por *feature* y por usuario.
- Optimizar el almacenamiento de *embeddings* mediante técnicas de compresión y cuantización sin pérdida significativa de precisión.

R11) Problemas de adopción/*engagement* (*recruiters* y candidatos)

Probabilidad: Media
Impacto: Medio

Si los *recruiters* y candidatos no obtienen valor rápido, cae la adopción y el *engagement*, activando un efecto de red negativo al haber menos oferta cae la demanda y viceversa. Un *onboarding* engorroso, formularios largos o baja relevancia disparan el abandono temprano.

Respuestas al riesgo:

- Aplicar autocompletado inteligente desde CVs y perfiles de *LinkedIn* para reducir fricción en la creación de perfiles.
- Implementar funcionalidades que generen valor inmediato como *insights* de mercado laboral, análisis de CV, o recomendaciones de mejora de perfil.
- Aplicar notificaciones inteligentes que alerten sobre oportunidades sin generar *spam* o fatiga de notificaciones.

R12) Deuda de pruebas y propagación de errores a producción

Probabilidad: Media
Impacto: Alto

La ausencia de *tests* robustos incrementa el riesgo de introducir *bugs* durante refactorizaciones o nuevas implementaciones, especialmente en flujos complejos que involucran múltiples servicios y estados. Esta deuda técnica puede manifestarse como fallos silenciosos en producción, inconsistencias en datos de usuarios, o degradación gradual de la calidad del *matching* sin detección temprana.

Respuestas al riesgo:

- Implementar umbrales mínimos de cobertura de código del 85% para módulos críticos con gates automáticos en *CI/CD*. Además, establecer escenarios bordes y de error para todas las funcionalidades del sistema.

- Establecer *environments* de *testing* con seeds de datos controlados que repliquen escenarios de producción.
- Implementar *smoke tests* automáticos en producción que validen funcionalidades críticas después de cada despliegue.

R13) Migraciones de esquema y compatibilidad de datos

Probabilidad: Media
Impacto: Medio

Cambios en la estructura de base de datos y esquemas de *API* pueden generar incompatibilidades que resulten en fallos durante despliegues, corrupción de datos existentes, o inconsistencias en reportes y métricas después de actualizaciones. Las migraciones mal planificadas pueden causar *downtime* prolongado, pérdida de información crítica de usuarios, o estados inconsistentes donde diferentes versiones del sistema interpretan los mismos datos de manera diferente.

Respuestas al riesgo:

- Establecer versionado de esquemas con políticas claras de deprecación gradual y períodos de gracia para actualizaciones.
- Crear entornos de *staging* que repliquen exactamente la estructura y volumen de datos de producción para *testing* de migraciones y establecer *scripts* de migración idempotentes que puedan ejecutarse múltiples veces sin efectos secundarios.
- Organizar sesiones de pasaje de versiones entre ambientes y poder ver posibles errores en migraciones y solucionarlos en tiempo real durante el pasaje.

R14) Dependencia de personas clave / rotación del equipo

Probabilidad: Media
Impacto: Medio

La concentración de conocimiento crítico en miembros específicos del equipo puede generar vulnerabilidades operacionales significativas cuando estos individuos abandonan la organización o no están disponibles. Este riesgo se manifiesta en cuellos de botella para *releases*, decisiones técnicas que requieren aprobación de personas específicas, y conocimiento implícito sobre arquitectura, procesos de negocio o integraciones que no está documentado formalmente.

Respuestas al riesgo:

- Establecer sesiones de *pair programming* obligatorias para transferencia de conocimiento durante desarrollo de funcionalidades complejas.
- Tener una documentación actualizada y completa sobre la plataforma y sus funcionalidades para poder capacitar al resto de miembros del equipo.

R15) Dependencia de modelos de los que no somos propietarios

Probabilidad: Baja
Impacto: Alto

La utilización de modelos de *embeddings* alojados en plataformas externas, como *Hugging Face*, de los cuales no somos propietarios, puede generar vulnerabilidades operativas y estratégicas. Cambios en las políticas de uso, eliminación de los modelos, modificaciones en sus pesos o arquitecturas, o la introducción de planes pagos podrían afectar la disponibilidad del servicio o degradar la calidad del servicio *matching*. Además, la falta de control sobre las versiones puede dificultar la reproducibilidad de resultados y generar inconsistencias entre entornos.

Respuestas al riesgo:

- Mantener copias locales y controladas de los modelos críticos para evitar interrupciones ante cambios externos.

- Documentar la versión exacta y las dependencias de cada modelo utilizado, asegurando trazabilidad y reproducibilidad.
- Evaluar modelos alternativos bajo licencias abiertas y planificar un posible cambio de modelo.

7. Investigación y Justificación de IA

7.1. Contexto y planteamiento del problema

Como ya se mencionó anteriormente, uno de los objetivos de Whizdy es permitir a las empresas encontrar a los candidatos más adecuados para sus vacantes, mejorando la precisión y relevancia del proceso de reclutamiento. Para lograrlo, se requirió desarrollar un **sistema de *matching* inteligente** capaz de emparejar candidatos y vacantes.

El desafío principal radica en que el emparejamiento basado en palabras clave o filtros simples no captura la **similitud semántica** entre las descripciones de vacantes y los perfiles de candidatos. Por ejemplo, un candidato con experiencia en "desarrollo *backend*" podría ser altamente relevante para una vacante que menciona “ingeniería en sistemas” o “programación”, aunque los términos exactos no coincidan.

Dada la información estructurada y no estructurada de los perfiles de candidatos y de las vacantes, el sistema requería:

- Comparar semánticamente más allá de la coincidencia exacta de palabras.
- Ponderar múltiples dimensiones del *matching* según las prioridades de cada vacante.
- Generar *rankings* precisos y justificados de candidatos para cada posición.

Para alcanzar este objetivo, se realizaron varias etapas de “investigación” con el objetivo de explorar y evaluar diferentes técnicas de IA aplicables a este sistema y también llevar a la práctica ejemplos mediante una *PoC*.

7.2. Opciones evaluadas

Durante la primera fase de investigación, realizada en la primera fase del proyecto, se evaluaron múltiples técnicas y tecnologías de IA para abordar tanto el problema central de *matching* como funcionalidades complementarias.

7.2.1. *Embeddings*

Los ***embeddings*** son representaciones numéricas de texto en vectores de espacios multidimensionales que capturan el significado semántico y permiten medir la similitud semántica entre palabras, frases o documentos completos [7]. Textos con significados similares son transformados a vectores que se ubican cerca entre sí en el espacio vectorial, facilitando los cálculos de similitud semántica entre sí.

Por ejemplo, "desarrollador *Python*" e "ingeniero en sistemas con experiencia en *Python*" tendrían vectores muy cercanos semánticamente, mientras que el vector "contador en el área de impuestos" estaría distante a los demás. Esta técnica permite comparar semánticamente textos de manera eficiente y replicable.

Se consideraron varias aplicaciones con esta técnica:

1. Comparar perfiles de candidatos con descripciones de vacantes.
2. Generar *embeddings* separados para habilidades técnicas, blandas, experiencia laboral y *fit* cultural.
3. Ajustar la importancia de cada dimensión según el tipo de vacante.
4. Uso de bases de datos vectoriales para el almacenamiento de *embeddings* y consultas de similitud eficientes.

7.2.2. *RAG (Retrieval-Augmented Generation)*

RAG es una técnica que combina la recuperación de información relevante desde una base de conocimiento con la generación de texto mediante un *LLM* [8]. Primero busca documentos o información pertinente (ej: vacantes similares previas) y luego los utiliza como contexto para que el *LLM* genere una respuesta informada y específica. Esto permite que el modelo acceda a información actualizada o específica del dominio sin haberla visto durante su entrenamiento.

Se consideraron varias aplicaciones con esta técnica:

1. El autocompletado de vacantes sugiriendo descripciones, responsabilidades y requisitos basados en el título y datos básicos
2. Recomendaciones personalizadas y explicaciones de por qué ciertos candidatos son ideales para una vacante
3. Sugerencias para los candidatos sobre cómo optimizar perfiles según requisitos de roles buscados

7.2.3. *Prompting* y *LLM* (*Large Language Models*)

Los ***LLMs*** son modelos de IA entrenados con grandes cantidades de texto que pueden entender y generar contenido en lenguaje natural [9]. El ***prompting*** es la técnica de dar instrucciones específicas en lenguaje natural a estos modelos para realizar tareas concretas [10]. Por ejemplo: "Dado el perfil de este candidato y esta vacante, genera un resumen explicando por qué es un buen *match*". Los *LLMs* son flexibles y pueden adaptarse a diferentes contextos sin necesidad de entrenamiento adicional.

Se consideraron varias aplicaciones para esta técnica:

1. La generación de texto mediante un *prompt* específico para generar un resumen de determinado candidato.
2. Generar una explicación entendible del resultado de un *matching* específico basando la comparación por categorías, por ejemplo explicando la categoría de habilidades técnicas específicamente.

7.2.4. *Fine-tuning*

El ***fine-tuning*** consiste en tomar un modelo de IA pre-entrenado y **reentrenarlo** con un *dataset* específico del dominio de aplicación para especializarlo en una tarea particular [11].

Por ejemplo, se consideró tomar un modelo de *embeddings* genérico y reentrenarlo con ejemplos de matches exitosos entre candidatos y vacantes para que aprenda las particularidades del dominio de recursos humanos. Esta es una técnica que requiere datos etiquetados de calidad, recursos computacionales significativos y expertise en *machine learning*.

7.3. Criterios de Selección y Decisiones Técnicas

En esta primera etapa de investigación, y con bases en los conceptos teóricos de cada una de las técnicas evaluadas, se realizó un análisis preliminar para determinar cuáles eran viables para el alcance y recursos del proyecto.

7.3.1. Técnicas Descartadas: *RAG y Fine-Tuning*

RAG fue descartado debido a que esta técnica requiere una base de datos histórica extensa de vacantes y candidatos para funcionar efectivamente. Además, esta información debe estar categorizada y estructurada de manera específica para que el sistema pueda recuperar el contexto relevante. En el momento de iniciar el proyecto no se contaba con este historial, y construir una base de datos que fuera suficientemente robusta para aplicar *RAG* de manera útil escapaba del alcance y los tiempos disponibles. Si bien esta técnica podría ser considerada en iteraciones futuras una vez que se acumule información histórica de matches y contrataciones, no era viable para esta primera etapa del producto.

Por motivos similares, se decidió descartar la aplicación de ***fine-tuning***. Esta técnica requiere un *dataset* etiquetado de alta calidad que incluya ejemplos de vacantes, candidatos y sus matches ideales, información que tampoco estaba disponible al inicio del proyecto. Adicionalmente, el *fine-tuning* implica un alto costo computacional y requiere tiempo significativo de desarrollo, recursos que excedían las capacidades del proyecto.

Más allá de los recursos técnicos, esta técnica demanda conocimiento avanzado en *machine learning* y experiencia especializada en el diseño de arquitecturas de entrenamiento, expertise que hubiera requerido una curva de aprendizaje considerable y hubiera retrasado el desarrollo del proyecto.

Además, durante la fase de investigación y Prueba de Concepto (detallada más adelante en este capítulo), se comprobó que los modelos pre-entrenados disponibles ofrecían un nivel de efectividad suficiente para validar el concepto y las necesidades iniciales del proyecto.

Posteriormente, en la implementación del sistema, se adoptó un modelo de *embeddings* desarrollado por terceros que ya contaba con *fine-tuning* específico para el dominio de *matching* entre descripciones de trabajo y perfiles profesionales. Esta decisión permitió aprovechar los beneficios de la especialización de dominio sin incurrir en los costos y la complejidad de realizar *fine-tuning* propio. Los detalles de este modelo final y su evaluación se documentan en la sección 8.3.2.

7.3.2. Técnicas Seleccionadas: *Embeddings* y *Prompting*

Habiendo descartado las dos técnicas previas, se decidió enfocar la investigación en profundidad en ***embeddings*** y ***prompting*** **con** ***LLMs*** ya que estas dos técnicas presentaban ventajas clave para el contexto de nuestro proyecto.

Los *embeddings* no requieren datos históricos previos, ya que utilizan modelos pre-entrenados capaces de capturar similitud semántica de manera generalizada. Además, son altamente escalables y eficientes una vez generados, lo que permite realizar comparaciones entre muchos perfiles y vacantes.

El *prompting* con *LLMs*, por su parte, ofrece flexibilidad para generar contenido personalizado sin necesidad de entrenamiento adicional. Simplemente proporcionando instrucciones claras en lenguaje natural, estos modelos pueden producir resúmenes, explicaciones y recomendaciones contextualizadas. Esto lo hacía perfecto para las funcionalidades auxiliares que mejoran la experiencia de usuario.

La combinación de ambas técnicas permiten cubrir las necesidades principales del sistema: *embeddings* como núcleo de nuestro sistema de *matching* para lograr precisión y

escalabilidad, y *prompting* para aportar la flexibilidad y el contexto dinámico necesarios en la generación de contenido personalizado.

Con estas técnicas ya seleccionadas, la siguiente fase de la investigación se enfocó en evaluar herramientas y modelos específicos, desarrollar una prueba de concepto incorporando *embeddings* y validar la efectividad de este enfoque para el problema de *matching* en reclutamiento.

7.4. Prueba de Concepto (PoC)

Una prueba de concepto es un prototipo experimental que permite verificar la viabilidad técnica de una solución antes de su implementación completa. En este proyecto, la PoC sirvió para validar que los *embeddings* eran efectivos para *matching* semántico, comparar diferentes modelos, e identificar la arquitectura óptima del sistema.

La PoC realizada por el equipo se desarrolló en dos iteraciones principales, y tuvo lugar durante la primera fase de investigación del proyecto, cada una con objetivos específicos de validación técnica.

7.4.1. Primera versión - implementación y enfoque

Objetivos de la primera iteración

Esta primera iteración tuvo tres objetivos principales: validar la viabilidad técnica de usar *embeddings* para *matching* semántico, familiarizarse con la utilización de bases de datos vectoriales y la generación de *embeddings*, y establecer una línea base de rendimiento utilizando datos simples que permitieran una evaluación clara de los resultados.

Configuración técnica

Para esta prueba inicial se utilizó el modelo de *embeddings* *all-MiniLM-L6-v2*, que es el modelo por defecto de *ChromaDB*. Este modelo resultó ideal para la experimentación inicial por ser ligero en términos de recursos computacionales, contando con 384 dimensiones y 22.7 millones de parámetros, lo que permitió ejecutarlo localmente sin necesitar una gran cantidad de recursos de *hardware*. Como base de datos vectorial se utilizó *ChromaDB*, y para calcular la similitud semántica entre los vectores se empleó el método de distancia coseno, que es el estándar en este tipo de aplicaciones.

El enfoque metodológico para esta primera prueba consistió en trabajar con un formato simple donde toda la información de cada candidato y vacante se presentaba en un solo párrafo sin estructura explícita. De esta manera, se generaba un único *embedding* por candidato y un único *embedding* por vacante, representando la totalidad de su información en un solo vector.

Dataset de prueba

El *dataset* de prueba se diseñó estratégicamente para validar diferentes niveles de discriminación del modelo. Se trabajó con 4 vacantes y 5 candidatos distribuidos de la siguiente manera: tres de las cuatro vacantes correspondían a desarrollo de *software*, específicamente dos posiciones de *Backend Developer* diferenciadas por años de experiencia y tecnologías requeridas, y una posición de *Frontend Developer*. La cuarta vacante pertenecía a una industria completamente diferente, siendo una posición de *Marketing Specialist*.

Del lado de los candidatos, se contaba con dos perfiles de *backend* con diferentes experiencias y *stacks* tecnológicos, dos perfiles de *frontend* con consideraciones similares, y un candidato del área de *marketing*. La elección de estos candidatos y vacantes fue crucial para probar realmente si el modelo podía distinguir en múltiples niveles: primero, entre industrias completamente diferentes (desarrollo de *software* contra *marketing*); segundo, dentro de la misma industria distinguir entre especialidades (*backend* contra *frontend*); y finalmente, dentro de cada especialidad, si podía diferenciar tecnologías específicas y años de experiencia.

Resultados observados

Los resultados de esta primera iteración fueron mixtos, mostrando tanto aciertos como limitaciones del enfoque. En términos de *matching* correcto, el sistema logró emparejar adecuadamente a los candidatos de *Frontend Developer* con la vacante correspondiente, así como al candidato de *Marketing Specialist* con su respectiva vacante. Además, se observaron distancias semánticas claramente diferenciadas entre roles incompatibles, lo cual validaba que el modelo podía distinguir entre industrias y especialidades diferentes.

Sin embargo, se identificó un problema significativo en uno de los casos de prueba. Para una vacante de *Backend Developer* que requería específicamente experiencia en *JavaScript* y *Node.js* como tecnologías obligatorias, *MongoDB* como base de datos, y un

mínimo de 4 años de experiencia, el sistema asignó la menor distancia (indicando el mejor *match*) a un candidato que presentaba un perfil problemático. Este candidato contaba con 6 años de experiencia en desarrollo *backend*, tenía experiencia con *MongoDB*, pero crucialmente no tenía experiencia en *JavaScript* ni *Node.js*, siendo su *stack* tecnológico *Python* y *Go*.

El análisis de este resultado problemático reveló un patrón incorrecto en el comportamiento del modelo. Parecía estar priorizando la experiencia general en desarrollo *backend* como un concepto amplio, el conocimiento de una tecnología coincidente (*MongoDB*), y los años de experiencia que superaban el requisito mínimo, por encima de la falta de experiencia específica en las tecnologías clave que eran explícitamente requeridas (*JavaScript* y *Node.js*). Este comportamiento sugería que el *embedding* estaba capturando el concepto general de "*backend developer* con experiencia", pero no lograba distinguir suficientemente bien entre *stacks* tecnológicos específicos cuando existían otras similitudes fuertes en el perfil.

Aprendizajes clave

Esta primera iteración proporcionó muchos aprendizajes valiosos que sirvieron de base para la siguiente iteración de la *POC*. En primer lugar, se validó que el concepto fundamental de usar *embeddings* para *matching* semántico era viable y funcionaba para casos básicos de discriminación entre perfiles claramente diferentes. Sin embargo, también se identificó una limitación importante del modelo *all-MiniLM-L6-v2*: su dificultad para distinguir tecnologías específicas cuando existían similitudes generales fuertes en otros aspectos del perfil.

Estos hallazgos llevaron a identificar tres áreas clave de mejora para la siguiente iteración: probar con modelos de *embeddings* más grandes que pudieran capturar mejor los matices técnicos, mejorar la estructura de las descripciones para hacer más explícitas las tecnologías y requisitos clave, y considerar la separación de *embeddings* por dimensiones (habilidades técnicas, blandas, experiencia, etc.) en lugar de un único *embedding* que representara toda la información.

Para consultar los datos específicos de prueba utilizados en esta iteración, incluyendo las descripciones completas de vacantes y candidatos, así como los resultados numéricos detallados del *matching* ver anexo 16.17.1.

7.4.2. Segunda versión - iteración y refinamiento

Objetivos de la segunda iteración

En base a los resultados y aprendizajes de la primera versión, esta segunda iteración se plantearon los siguientes objetivos: resolver los problemas de *matching* identificados anteriormente, evaluar modelos de *embeddings* más sofisticados, mejorar la estructura de datos utilizada para generar los *embeddings*, probar con *datasets* más complejos y realistas, evaluar otras herramientas y finalmente validar la separación de *embeddings* por dimensiones como estrategia para mejorar la precisión y explicabilidad del resultado.

Cambios Implementados

1. Evaluación de modelos de *embedding*

Se realizó una evaluación progresiva de tres modelos de *embeddings*, comenzando desde el baseline establecido en la primera iteración. El primer modelo, *all-MiniLM-L6-v2*, había demostrado las limitaciones ya identificadas con tecnologías específicas. Se procedió entonces a probar *all-mpnet-base-v2*, un modelo significativamente más grande con 768 dimensiones y 109 millones de parámetros. Sin embargo, este modelo produjo resultados muy similares al anterior, manteniendo las mismas limitaciones en la distinción de *stacks* tecnológicos específicos.

Finalmente se evaluó *paraphrase-mpnet-base-v2*, una versión con *fine-tuning* específico para tareas de paráfrasis. Este modelo demostró capacidades superiores que resultaron cruciales para resolver los problemas identificados. Su entrenamiento especializado le permite entender mejor cuando dos textos expresan ideas similares con palabras diferentes, captar matices en descripciones de habilidades técnicas, y crucialmente, distinguir mejor entre tecnologías relacionadas pero diferentes, como *Python* y *Go* contra *JavaScript* y *Node.js*.

2. Mejora en la estructura de datos

El cambio más significativo en esta iteración fue la transformación en cómo se representaban las habilidades técnicas. En la primera versión se utilizaba una única descripción donde las habilidades se mencionaban de forma narrativa, por ejemplo: *"Proficiency in JavaScript (React.js, Node.js), HTML, and CSS. Experience with databases such as MongoDB and PostgreSQL"*, esta estructura que es natural para la lectura humana, no hacía explícita información crítica para el *matching*.

En la segunda versión se adoptó una estructura explícita y estandarizada donde cada habilidad técnica se representaba indicando la habilidad específica, el nivel mínimo requerido, el nivel ideal buscado, y si la habilidad era obligatoria o deseable. Esta estructura permitía que los *embeddings* capturaran de manera más precisa no sólo qué tecnologías se mencionan, sino también la importancia relativa y el nivel de expertise esperado en cada una.

Los resultados demostraron que los *embeddings* generados con esta estructura tenían mejores capacidades de *matching*, probablemente porque la información clave estaba más explícitamente representada y con menos ambigüedad para que el modelo interpretara.

3. Dataset de prueba mejorado

Se incrementó significativamente la complejidad del *dataset* de prueba. Las vacantes se volvieron más detalladas, incluyendo no sólo requisitos técnicos sino también responsabilidades específicas, habilidades blandas, e información sobre la cultura de la empresa. Del lado de los candidatos, se diseñaron perfiles similares pero con diferencias sutiles pero importantes. Por ejemplo, se crearon candidatos que compartían la mayoría de las tecnologías (*React, Node.js, Docker*) pero difieren en aspectos específicos como la base de datos que dominaban (*MongoDB* contra *PostgreSQL*), o en el nivel de *expertise* en herramientas de *DevOps* (*proficient* contra *expert* en *Docker* y *CI/CD*).

El objetivo de esta complejidad aumentada era validar que el sistema podía diferenciar entre candidatos que comparten muchas características pero difieren en aspectos críticos que podrían ser decisivos para una contratación real.

4. Implementación de *embeddings* separados por dimensión

Se implementó una arquitectura de *embeddings* separados donde, para cada candidato y cada vacante, se generaban cuatro *embeddings* independientes: habilidades técnicas, habilidades blandas, experiencia y educación, y valores culturales. Estos cuatro vectores se concatenan posteriormente para formar un *embedding* compuesto que representaba todas las dimensiones del perfil.

Este enfoque ofrecía múltiples ventajas. Permitía calcular *scores* de compatibilidad específicos para cada dimensión, lo cual facilitaba identificar fortalezas y debilidades precisas de cada candidato. También habilitaba la posibilidad de aplicar pesos diferentes según el tipo de vacante, por ejemplo dando más importancia a habilidades técnicas en roles de ingeniería *senior*, o priorizando habilidades blandas y *fit* cultural en posiciones de liderazgo. Finalmente, proporcionaba *feedback* granular que podía ser presentado al empleador de manera más informativa y accionable.

5. Análisis de herramientas

Una vez validada la efectividad de los *embeddings*, en esta segunda fase de investigación también nos enfocamos en analizar qué herramientas de base de datos vectorial podríamos implementar en nuestra solución final. Durante la *PoC* se había utilizado *ChromaDB* por su simplicidad y facilidad de integración para propósitos experimentales, pero era necesario evaluar alternativas que ofrecieran mejor rendimiento, escalabilidad y funcionalidades avanzadas para el sistema en producción.

Se evaluaron tres opciones principales de bases de datos vectoriales: ***Milvus***, ***Superlinked*** y ***Qdrant***. Cada una fue analizada en función de su capacidad para manejar búsquedas híbridas con múltiples vectores, flexibilidad en la ponderación de diferentes dimensiones, y características específicas que beneficiaran al caso de uso de selección de personal.

Milvus es una base de datos vectorial madura diseñada para búsquedas de similitud mediante *Approximate Nearest Neighbor* (ANN). Su funcionalidad de *Hybrid Search* combina resultados de hasta cuatro campos vectoriales diferentes mediante dos estrategias: *WeightedRanker* (asignación de pesos específicos) y *RRFRanker* (equilibrio automático). Ofrece optimización automática de parámetros (*AUTOINDEX*) y particionamiento para

mejorar rendimiento a escala. Sin embargo, presenta limitaciones clave: no genera *embeddings* por sí mismo, la ponderación de campos no es suficientemente flexible para ajustes dinámicos complejos, y carece de explicabilidad integrada del *matching*.

Superlinked se especializa en búsqueda multi-atributo mediante concatenación de vectores con ponderación extremadamente flexible en tiempo de consulta (valores de -1.0 a 1.0). Su característica diferenciadora es la capacidad de usar pesos negativos para buscar diferencias intencionales, y realiza una sola búsqueda eficiente en lugar de múltiples con post-procesamiento. Como solución más reciente, presenta menor madurez que alternativas establecidas, cuenta con menos documentación y comunidad de soporte, y requiere mayor desarrollo personalizado.

Qdrant es una base de datos vectorial *open-source* que ofrece *Named Vectors* (múltiples vectores nombrados por punto) y *Payload-Based Reranking* (reordenamiento usando metadatos) [14]. Soporta búsquedas multi-etapa con filtrado progresivo, consultas híbridas mediante *Prefetch*, y algoritmos de fusión como *RRF* y *DBSF*. Su característica más relevante es la capacidad de implementar funciones de reordenamiento completamente personalizadas y obtener *scores* individuales para cada dimensión, fundamental para la explicabilidad en decisiones de contratación.

Criterios de Evaluación y Decisión

Para tomar la decisión se establecieron criterios específicos relevantes al caso de uso de selección de personal. El sistema debía permitir control total sobre el proceso de *matching*, incluyendo la capacidad de implementar funciones personalizadas de *scoring*. Era fundamental obtener *scores* individuales para cada dimensión del perfil (habilidades técnicas, blandas, experiencia, *fit* cultural) para poder proporcionar explicabilidad a los empleadores sobre por qué un candidato fue seleccionado. La ponderación debía ser flexible y ajustable dinámicamente según el tipo de puesto. Finalmente, el rendimiento debía ser adecuado para manejar búsquedas en bases de datos con potencialmente miles de candidatos.

Característica	**Milvus**	**Superlinked**	***Qdrant***
Hybrid Search integrado	Sí	Sí	Sí
Scores individuales por dimensión	No	No	Sí
Madurez del producto	Alta	Media	Alta
Control de reordenamiento	Medio	Alto	Muy alto
Explicabilidad de resultados	Baja	Media	Alta

Decisión Final: *Qdrant*

Se seleccionó *Qdrant* como la base de datos vectorial para la implementación en producción. Esta decisión se fundamentó en varios factores clave. En primer lugar, *Qdrant* ofrece la mayor flexibilidad para implementar lógica personalizada de reordenamiento, permitiendo control total sobre cómo se combinan diferentes aspectos del perfil de candidatos. La capacidad de obtener *scores* individuales para cada dimensión resultó crítica para la explicabilidad del sistema, un requisito fundamental en recursos humanos donde las decisiones de contratación deben ser justificables y transparentes.

El enfoque de *Named Vectors* de *Qdrant* se alineaba perfectamente con la arquitectura de *embeddings* separados validada en la *PoC*, permitiendo representar claramente diferentes aspectos del perfil (habilidades técnicas, habilidades blandas, experiencia, valores culturales) y realizar búsquedas independientes o combinadas según la necesidad. Adicionalmente, la funcionalidad de búsqueda multi-etapa ofrece un balance óptimo entre velocidad y precisión, crucial para escalar el sistema con bases de datos grandes de candidatos.

Finalmente, *Qdrant* proporciona control total sobre la implementación de pesos personalizados, permitiendo ajustar dinámicamente la importancia de cada dimensión según el tipo de puesto (por ejemplo, priorizando habilidades técnicas en roles de ingeniería o habilidades blandas en posiciones de liderazgo). Esta flexibilidad, combinada con la capacidad de explicar decisiones mostrando *scores* por categoría, posicionó a *Qdrant* como la solución más adecuada para los requerimientos específicos del sistema de selección de personal.

Resultados de la segunda iteración

Comparación de modelos

Se realizaron dos *tests* principales para evaluar el rendimiento de los diferentes modelos. El primer *test* se enfocó específicamente en el caso problemático identificado en la primera iteración: una vacante de *Backend Developer* que requería *JavaScript* y *Node.js*, evaluada contra un candidato con experiencia en *Python* y *Go* pero sin el *stack* requerido. Con *all-MiniLM-L6-v2* y *all-mpnet-base-v2*, el problema persistía y este candidato incorrecto seguía siendo rankeado en primer lugar. Sin embargo, con *paraphrase-mpnet-base-v2*, el candidato fue correctamente posicionado en tercer lugar o inferior, mientras que candidatos con el *stack* adecuado (*JavaScript*/*Node.js*) fueron priorizados. Este resultado confirmó que el cambio de modelo había resuelto efectivamente la limitación identificada.

El segundo *test* consistió en evaluar el *matching* general en todas las vacantes del dataset mejorado. Utilizando *paraphrase-mpnet-base-v2* junto con la estructura de datos mejorada, el sistema produjo *rankings* consistentemente correctos. Para consultar los datos específicos de prueba, perfiles de candidatos y resultados detallados del *matching*, ver anexo 16.17.2.

Un análisis detallado de uno de los matches más complejos demostró la efectividad del sistema. Para una vacante de *Full Stack Developer* que requería *React, Node.js, MongoDB* y *Docker*, el candidato identificado como mejor *match* cumplía con todas las habilidades técnicas requeridas en el nivel ideal (*Expert*), no solo en el nivel mínimo. El segundo mejor candidato, aunque también cumplía con todos los requisitos, los satisfacía en nivel *Proficient* en lugar de *Expert* en algunas áreas. El tercer candidato, especializado en *Frontend*, tenía experiencia limitada en las tecnologías de *backend* requeridas. Esta diferenciación precisa validaba que el sistema estaba capturando correctamente tanto la presencia de las habilidades como el nivel de *expertise* en cada una. Para ver la tabla comparativa detallada de estos candidatos consultar anexo 16.17.2

Validación de *embeddings* separados

Se validó que la separación por dimensiones funcionaba correctamente y aportaba valor real. Para cada *match* se podían obtener *scores* específicos por cada dimensión (habilidades técnicas, blandas, experiencia, *fit* cultural), lo cual permitía visualizaciones detalladas en *interfaces* de usuario, como gráficos de radar que mostraban el perfil de compatibilidad multidimensional. Esta granularidad también facilitaba la identificación de fortalezas y debilidades específicas de cada candidato, y habilitaba el ajuste de pesos según las prioridades particulares de cada empresa o vacante.

7.4.3. Análisis comparativo y resultados de la PoC

Resumen de rendimiento

El análisis comparativo de ambas iteraciones de la prueba de concepto reveló mejoras significativas en múltiples dimensiones. El cambio de modelo de *all-MiniLM-L6-v2* a *paraphrase-mpnet-base-v2* duplicó el número de dimensiones de los vectores (de 384 a 768), incrementando la capacidad representacional del sistema. Especialmente notable fue la mejora en la diferenciación entre tecnologías específicas: el problema identificado en la primera iteración, donde candidatos con *Python/Go* eran rankeados por encima de candidatos con *JavaScript/Node.js* para vacantes que explícitamente requerían este último *stack*, fue completamente resuelto. La estructura de datos evolucionó de una descripción unificada a representación explícita y estructurada, y la arquitectura de *embeddings* pasó de un único vector por entidad a cuatro *embeddings* separados concatenados.

Casos de uso validados

A lo largo de ambas iteraciones de la prueba de concepto se validaron exitosamente múltiples capacidades clave del sistema. Se confirmó el *matching* semántico básico, donde el sistema entiende que términos como "*Backend Developer*" y "*Server-side Engineer*" son conceptualmente similares. Se validó la distinción de tecnologías específicas, diferenciando correctamente entre candidatos con *Python* y *Go* versus *JavaScript y Node.js*. La priorización por nivel de *expertise* funcionó correctamente, rankeando mejor a candidatos con nivel *Expert* cuando ese era el ideal, sobre candidatos con nivel *Proficient*. La comparación multidimensional demostró que el sistema identifica como mejor candidato a quien puntúa bien en todas las dimensiones evaluadas, no solo en una o dos. Finalmente, se validó la

robustez con perfiles complejos, manejando correctamente descripciones extensas con múltiples habilidades y experiencias.

Limitaciones identificadas

A pesar de los resultados positivos, se identificaron limitaciones importantes que deben considerarse. La escalabilidad no fue validada completamente, ya que las pruebas se realizaron con aproximadamente diez candidatos, faltando validación con cientos o miles de perfiles. El *dataset* fue limitado a roles técnicos de *software*, quedando pendiente la validación con roles de otras industrias como ventas, operaciones o finanzas. Los resultados fueron validados manualmente por el equipo de desarrollo, pero no se contó con *feedback* de reclutadores o empleadores reales. Existe incertidumbre sobre si el modelo local *paraphrase-mpnet-base-v2* será suficiente en producción o si será necesario migrar a servicios como *OpenAI*. Finalmente, aunque se diseñó el sistema para ajustar pesos dinámicamente mediante *feedback* de usuarios, esta funcionalidad no pudo ser validada en la PoC por falta de datos históricos de uso real.

Comparación: *embeddings* únicos versus separados

El análisis comparativo entre la arquitectura de *embedding* único utilizada inicialmente y los *embeddings* separados por dimensión reveló *trade-offs* importantes. En términos de precisión general, ambos enfoques funcionaban bien, pero los *embeddings* separados mostraron mejor rendimiento. La granularidad es inexistente en el enfoque de *embedding* único, mientras que los *embeddings* separados ofrecen alta granularidad por dimensión. El ajuste de pesos no es posible con un único *embedding*, mientras que con *embeddings* separados es posible y flexible. La explicabilidad es limitada con un único vector, pero alta cuando se separa por categorías. La complejidad de implementación es baja para *embedding* único y media para *embeddings* separados. El tamaño del vector es de 768 dimensiones para un *embedding* único contra 3,072 dimensiones para cuatro *embeddings* concatenados. El tiempo de procesamiento se incrementa de aproximadamente un segundo a dos segundos.

La decisión final fue clara: los beneficios de granularidad y explicabilidad que ofrecen los *embeddings* separados superan ampliamente el ligero incremento en complejidad de implementación y tiempo de procesamiento. Esta arquitectura permite no solo obtener mejores matches, sino también proporcionar justificaciones más claras y detalladas de por qué un candidato es o no adecuado para una posición.

Decisiones finales de la investigación

Como resultado de la *PoC*, se validaron los conceptos fundamentales que guiarían la implementación del sistema. Se confirmó que los *embeddings* son una técnica viable para *matching* semántico, que la arquitectura de ***embeddings* separados por dimensión** proporciona la granularidad y explicabilidad necesarias, y que ***Qdrant*** es la base de datos vectorial más adecuada por su capacidad de manejar *Named Vectors* e implementar funciones personalizadas de reordenamiento.

Esta *PoC* sirvió como **validación técnica y exploración** de enfoques, pero no fue llevada directamente a producción. Los conceptos y arquitectura validados fueron la base para el diseño del sistema final, cuya implementación con mejoras y optimizaciones se documenta en el Capítulo 8 “Implementación” (sección 8.3 “Whizdy Matching*”*).

7.5. Elección de *LLM*

7.5.1 Contexto y requisitos

Como se mencionó en la sección 7.2.3, el sistema requiere capacidades de generación de texto mediante lenguaje natural para dos funcionalidades específicas como por ejemplo la generación de resúmenes personalizados de candidatos y la creación de explicaciones interpretables sobre los resultados de *matching*. Estas funcionalidades demandan un capaz de procesar información estructurada sobre perfiles profesionales y vacantes, y generar texto coherente y contextualmente relevante en respuesta a *prompts* específicos.

A diferencia del componente de *embeddings*, donde el modelo debe ejecutarse frecuentemente para cada operación de *matching*, las llamadas al *LLM* ocurren bajo demanda cuando un usuario solicita explícitamente dichas funcionalidades.

7.5.2 Criterios de evaluación

Costo operativo: Dada la naturaleza del proyecto y en sincronía con nuestros objetivos del proyecto buscábamos mantenernos en la capa gratuita en la mayor cantidad de servicios posibles, por lo que el costo por *token* de entrada y salida era un factor crítico.

Disponibilidad de *tier* gratuito: La existencia de un *tier* gratuito o de créditos iniciales permitiría validar la funcionalidad y desarrollar el sistema sin costo, posponiendo costos operativos hasta alcanzar volúmenes que justifiquen el gasto.

Capacidades de generación: El modelo debía ser capaz de generar texto coherente y contextualmente apropiado para descripciones profesionales, sin requerir las capacidades más avanzadas de razonamiento complejo que ofrecen los modelos de mayor costo.

Facilidad de integración: La disponibilidad de APIs bien documentadas y librerías en los lenguajes utilizados en el proyecto (*Python*) era esencial para minimizar el tiempo de desarrollo.

7.5.3 Modelos considerados

Se evaluaron los principales proveedores de *LLMs* disponibles en el mercado al momento de la investigación en Octubre de 2024, incluyendo GPT-4 y GPT-3.5 de *OpenAI*, *Claude* de *Anthropic*, y los modelos *Gemini* de *Google*. La comparación se centró principalmente en estructura de precios, límites del *tier* gratuito, y capacidades de generación relevantes para el caso de uso específico del proyecto. La tabla comparativa detallada de precios y características de estos modelos se encuentra documentada en el Anexo 16.23.

7.5.4 Decisión: Google *Gemini*

Se seleccionó *Google Gemini* como el *LLM* del sistema basándose en dos factores principales. En primer lugar, *Gemini* ofrece un *tier* gratuito generoso que permite un volumen significativo de solicitudes mensuales sin costo, facilitando el desarrollo y las etapas iniciales de operación sin inversión monetaria. En segundo lugar, el *pricing* de *Gemini* para uso pagado resultó considerablemente más económico que alternativas como GPT-4, manteniendo capacidades de generación de texto suficientes para los casos de uso de la aplicación.

Las pruebas preliminares de *prompting* confirmaron que *Gemini* producía salidas de calidad adecuada para generar tanto resúmenes concisos de candidatos como explicaciones interpretables de los resultados de *matching*, cumpliendo los requisitos funcionales sin necesidad de recurrir a modelos de mayor costo. Esta decisión permitió minimizar el riesgo financiero en las etapas tempranas del proyecto mientras se validaba la propuesta de valor con usuarios reales.

8. Implementación

En esta sección se profundiza en la solución desarrollada, describiendo su estructura general, las tecnologías utilizadas, los principales detalles de implementación y los aspectos relevantes que caracterizan la plataforma.

8.1. Tecnologías seleccionadas

Para este proyecto se optó por un *stack* que prioriza velocidad de desarrollo, mantenibilidad y simplicidad de despliegue, alineando la experiencia del equipo con las necesidades y preferencias del cliente así como con las necesidades asociadas del proyecto, como la integración con IA.

El cliente estableció el uso de *Next.js* [6] con *shadcn/ui* [17] para el *frontend*, debido a que la especificación visual del sistema había sido diseñada en *Figma* utilizando los componentes y estilos de esta librería. Esta decisión garantiza coherencia directa entre el diseño proporcionado y la implementación final.

Adicionalmente la adopción de *Next.js* representaba una oportunidad técnica valiosa para el equipo, ya que permitía explorar un *framework* moderno y ampliamente adoptado en la industria, alineándose con los objetivos de aprendizaje del proyecto académico.

Por otro lado, GoGrow sugirió *Ruby on Rails* [12] para el *backend*, basándose en su experiencia previa con esta tecnología. Sin embargo, al analizar los requerimientos del proyecto se identificaron riesgos significativos: el equipo no tenía experiencia directa con esta tecnología, mientras que el proyecto ya presentaba desafíos técnicos considerables por la complejidad del sistema, la integración con componentes de inteligencia artificial y el uso obligatorio de *Next.js* en el *frontend*.

Dado que estos requerimientos técnicos eran no negociables y ya implicaban curvas de aprendizaje significativas, se determinó que agregar *Ruby on Rails* como otra tecnología nueva incrementaría innecesariamente los riesgos del proyecto. Por esta razón, se propuso *Node.js* con *TypeScript* para el *backend*, aprovechando la experiencia del equipo y unificando el lenguaje de desarrollo.

A partir de esa propuesta, el cliente sugirió utilizar *Next.js* también en el *backend*, decisión que evaluamos comparativamente con nuestra propuesta inicial. (Véase anexo 16.16 Documento cambio *stack* tecnológico, el cual profundiza la elección de tecnologías y el cambio del *Stack* a partir del documento compartido con el cliente ante el planteamiento de cambio)

La evaluación entre ambas opciones se centró en analizar el equilibrio entre velocidad de desarrollo y escalabilidad futura. La arquitectura con *Node.js backend* independiente ofrecía ventajas claras en términos de escalabilidad, modularidad y flexibilidad tecnológica, permitiendo separar responsabilidades y preparar el terreno para una evolución hacia microservicios. Sin embargo, implicaba una mayor complejidad inicial en configuración, despliegue y coordinación entre servicios, lo que ralentizaría el desarrollo en un contexto de restricciones temporales. Por el contrario, *Next.js fullstack* simplificaba drásticamente la pila tecnológica al unificar *frontend* y *backend* en una sola base de código, eliminando la necesidad de gestionar comunicación entre servicios y permitiendo un despliegue ágil en plataformas como *Vercel* o *Amplify*. Aunque esta opción presenta limitaciones en cuanto a escalabilidad y mayor acoplamiento al *framework*, se priorizó su adopción alineada con los objetivos del proyecto y el carácter académico del proyecto, dejando abierta la posibilidad de evolucionar hacia una arquitectura más distribuida en etapas posteriores.

Considerando el alcance del proyecto, las restricciones temporales del proyecto académico y la necesidad de entregar una primera versión estable, se adoptó la opción *Next.js fullstack*. Esta decisión se basó en que sus ventajas de desarrollo rápido y simplicidad se alineaban mejor con los objetivos del proyecto que las ventajas de escalabilidad a largo plazo que ofrecía *Node*.js *backend* independiente.

Independientemente de las tecnologías elegidas para la aplicación principal, necesitábamos un servicio independiente que se encargará de las operaciones de IA, como el *matching* con *embeddings*, ya que dichos procesamientos requerirían un enfoque tecnológico distinto.

Para este servicio propuso al cliente y se aprobó la elección de *Python*, aprovechando su ecosistema especializado en ciencia de datos, bibliotecas de *machine learning* (como *scikit-learn* y *transformers*) y la integración nativa con *Qdrant* mediante su librería oficial.

Este módulo opera como un servicio independiente de la aplicación, dedicado al procesamiento de *embeddings* y otras funcionalidades relevantes al *matching* e integraciones con IA.

Como resultado se llegó a una arquitectura híbrida con dos módulos claramente separados: Whizdy App que utiliza *Next.js fullstack* con *TypeScript* para la aplicación principal y por otro lado Whizdy Matching que se implementó con *Python* y *Qdrant* para el servicio especializado de IA. Esta separación aísla las cargas computacionales de IA y preserva la simplicidad operativa de Whizdy App, manteniendo a la vez lugar para evolucionar el motor de *matching* sin afectar el ciclo de despliegue de la aplicación principal.

Por otro lado, tanto Whizdy App como Whizdy Matching acceden de forma exclusiva a sus respectivas bases de datos: *PostgreSQL* [22] para la aplicación principal y *Qdrant* para el servicio de *embeddings*. Esta separación garantiza que no existan cruces de acceso ni dependencias directas entre ambas capas de datos, preservando la integridad de cada módulo y facilitando la evolución independiente de sus esquemas y modelos, así como facilitando el futuro escalado independiente de cada uno de los módulos.

La elección de *Next.js fullstack* sobre *Node.js backend* separado redujo significativamente la complejidad operativa y aceleró el desarrollo, permitiendo entregar una primera versión estable en los tiempos requeridos. Paralelamente, la separación del módulo de IA en *Python* garantiza que las operaciones computacionalmente intensivas no comprometan el rendimiento de la aplicación principal y mantiene flexibilidad para evoluciones futuras del algoritmo de *matching*.

Esta arquitectura establece una base sólida que balancea las restricciones temporales del proyecto académico con la necesidad de construir fundamentos técnicos que permitan la evolución hacia un producto comercial. Más adelante se aprovechará el capítulo 9 de Arquitectura del sistema para profundizar sobre este tópico.

8.2. Whizdy App

8.2.1. Frontend

Arquitectura Modular

La implementación del *frontend* sigue una arquitectura basada en componentes modulares utilizando el patrón *App Router* de *Next.js* 13+. La aplicación está estructurada en dos experiencias de usuario diferenciadas y completamente separadas:

- Interfaz para candidatos (/*candidate*): Incluye las funcionalidades de visualización, gestión de perfil, seguimiento de procesos de selección y configuraciones personales.
- Interfaz para empresas (/*company*): Abarca la gestión de las vacantes, evaluación de candidatos, proceso de entrevistas y configuraciones corporativas.

Cada módulo cuenta con su propio conjunto de *layouts*, componentes especializados y flujos de navegación específicos, garantizando separación de responsabilidades y mantenibilidad del código.

Sistema de *layouts* anidados

La aplicación implementa un sistema de *layouts* jerárquicos de *Next.js* que optimiza la reutilización de código y asegura consistencia visual. Teniendo: *layout* raíz (src/app/layout.tsx) que contiene la estructura base y providers globales; *layouts* de usuario (src/app/candidate/layout.tsx, src/app/company/layout.tsx) que implementan *sidebars* específicos y navegación contextual; *layouts* de sección para cada funcionalidad principal (*jobs*, *candidates*, *settings*) teniendo sus elementos comunes dentro de los *layouts* previos, por ejemplo src/app/candidate/settings/layout.tsx

Los componentes heredan automáticamente la estructura y estilos del *layout* padre, asegurando consistencia visual y de comportamiento en toda la aplicación sin duplicación de código.

Biblioteca de Componentes Reutilizables

Al ver que muchas de las vistas del sistema estaban compuestas por componentes compartidos o que compartían similitudes, se trabajó en la aplicación implementando una biblioteca de más de 70 componentes especializados con diferentes propósitos.

- Componentes básicos: *button, input, card, dialog, form* (basados en *shadcn/ui*), etc.
- Componentes de dominio específico: *CandidateCard, JobSummaryCard, MatchingInfoCard, InterviewCalendar,* etc.
- Componentes complejos: *RadarComparisonChart, MatchingBarChart, CandidateBoard, skillsComparison,* etc.

Estos componentes implementan patrones de diseño consistentes, manejan estados internos complejos y facilitan el desarrollo ágil de nuevas funcionalidades mediante composición.

Gestión de estado y almacenamiento local

Se implementó un sistema de almacenamiento local controlado que almacena únicamente información esencial para navegación y experiencia de usuario, implementa *TTL* (*Time To Live*) configurable vía variables de entorno, previene inconsistencias mediante expiración automática de datos obsoletos e incluye manejo de errores robusto para operaciones de *localStorage*.

Esto asegura un uso eficiente del almacenamiento local, contribuyendo al rendimiento y seguridad de la información.

***Middleware* de autenticación**

En adición a tener un sistema de permisos y autenticación, del cual hablaremos más adelante, la aplicación incluye un *middleware* personalizado que se encarga de agregar los datos del mismo como *headers* de autorización a todas las llamadas del *API* y además captura los códigos de sus respuestas, de esta manera maneja los diferentes tipos de errores de autorización (401, 403) dirigiendo al usuario al login automáticamente. Con esto evitamos que un usuario no autenticado o sin los permisos suficientes navegue dentro de la plataforma.

Arquitectura de servicios

Las llamadas al *backend* están centralizadas en módulos especializados agrupados por las interacciones que se tienen con cada entidad o con cada servicio como se tiene análogamente en la lógica de negocios, teniendo un servicio de autorización, otro de candidatos y así para todas las entidades principales del sistema.

Cada servicio encapsula la lógica de comunicación con *endpoints* específicos, la transformación de datos y el manejo de errores consistente, facilitando el mantenimiento y el *testing*.

Aspectos técnicos adicionales

Tailwind CSS [15]: los componentes utilizados provenientes de *shadcn/ui* están fundamentalmente construidos utilizando *Tailwind CSS* para sus estilos y su diseño *responsive*, por lo que es necesario instalar dicho *CSS framework* para poder utilizar esta librería así como también tener la posibilidad de personalizar los estilos de sus componentes.

8.2.2. Backend

El *backend* implementa una arquitectura multicapa robusta que separa las responsabilidades a través de diferentes niveles de abstracción bien definidos, donde cada capa tiene un propósito específico y se comunica de manera controlada con las capas adyacentes. La estructura cuenta con una capa de controladores de *API* (*routes*), una capa de lógica de negocio (*businessLogics*), una capa de datos (*Prisma ORM*), y capas de soporte para *middlewares* y servicios auxiliares.

Capa de presentación (*API Routes*)

La aplicación cuenta con más de 90 *endpoints RESTful* organizados jerárquicamente y aprovechando el sistema de ruteo dinámico para *APIs* que *Next.js* ofrece a través de la carpeta *app/api*. Esta capa actúa como el punto de entrada principal para todas las solicitudes del cliente, proporcionando una interfaz uniforme y consistente para acceder a los recursos del sistema.

Los *endpoints* tienen una separación clara por dominio, como por ejemplo *auth/, candidates/, company/, matching/, common/* y *generative/*, con el objetivo de que cada dominio y *endpoint* tenga sus responsabilidades acotadas y sean fáciles de mantener.

Ejemplos de organización por dominio:

- */api/auth/*: Gestión de autenticación, *login*, registro y validación de *tokens.*
- */api/candidates/*: Operaciones *CRUD* de candidatos, configuraciones y estadísticas.
- */api/company/*: Gestión empresarial, perfiles corporativos y configuraciones.
- */api/matching/*: Algoritmos de emparejamiento y recomendaciones.
- */api/common/*: Recursos compartidos como catálogos de ubicaciones, habilidades y enumeraciones.

Además, contamos con *endpoints* especializados con parámetros dinámicos como pueden ser el id de las entidades, lo que permite acceder a recursos específicos de manera intuitiva y *RESTful*. Por ejemplo, para acceder a los candidatos recomendados de una vacante podemos usar rutas como */api/company/jobs/[id]/candidates/[candidateId]/matching-details*, donde los parámetros dinámicos [id] y [candidateId] permiten una navegación flexible y tipada hacia recursos anidados.

Capa lógica de negocio

La lógica de negocio es el código que actúa de mediador entre la capa de presentación y la capa de datos, interactuando con la información de las solicitudes de la capa de presentación y de la capa de datos para cumplir con la entrega de la información. Esta capa encapsula todas las reglas de negocio, validaciones complejas y operaciones que definen el comportamiento específico del dominio de la aplicación. Cuenta con más de 10 lógicas especializadas implementadas como clases dedicadas, como pueden ser la lógica de candidatos, de empresas, del servicio de *matching*, de autorización y muchas otras. Cada clase de lógica de negocio está diseñada siguiendo el principio de responsabilidad única, garantizando que cada módulo tenga un propósito específico y bien definido.

Esta arquitectura permite que la lógica de negocio sea *testeable* de forma independiente, facilitando las pruebas unitarias y la validación de reglas complejas sin depender de la infraestructura externa.

Gestión de datos y persistencia

Para el manejo de datos y la interacción con la base de datos se utiliza *Prisma* como *ORM* (*Object Relational Mapping*), integrándose directamente dentro de las clases de lógica de negocio en lugar de implementar una capa de acceso a datos separada.

Esta decisión arquitectónica se tomó porque *Prisma* ya provee las funcionalidades necesarias para gestionar modelos, relaciones y validaciones de manera estructurada [13], eliminando la necesidad de una abstracción adicional.

Prisma ofrece un acceso tipado seguro y eficiente a la base de datos relacional *PostgreSQL*, permitiéndonos definir esquemas modularizados así como ejecutar consultas a la base de datos utilizando *TypeScript*. También nos brinda la conversión de tipos automático de las entidades definidas en los schemas de *Prisma* a tipos nativos de *TypeScript*, garantizando type safety en toda la aplicación.

La configuración de *Prisma* utiliza la característica *prismaSchemaFolder* que permite organizar los modelos en archivos separados por dominio funcional y entidad, por ejemplo *candidate.prisma, job.prisma, match.prisma*, etc.

La utilización de *Prisma* proporciona una abstracción robusta que permite trabajar con objetos *TypeScript* sin preocuparse por los detalles de persistencia, mientras mantiene el rendimiento y la integridad de los datos a través de patrones como *Repository Pattern*, *Unit of Work*, y *connection pooling* automático.

***Middlewares* transversales**

El *backend* está compuesto por un sistema robusto de *middlewares* que proporcionan funcionalidades transversales y estandarizadas de seguridad, manejo de errores, autenticación y autorización lo que nos permite manejar estos aspectos de la plataforma de manera automática e integral.

***Middleware* de autenticación**, implementa un patrón de *Higher-Order Functions* que encapsula toda la lógica de autorización y autenticación. Siendo capaz de:

- Validar los *JWT (JSON Web Token)* de la sesión utilizando el *secret* configurable.
- Controla los roles (*candidate*/*company*) a la hora de realizar ciertas acciones y acceder a recursos específicos.
- Manejar los errores de autorización y autenticación (errores 401 *Unauthorized* y 403 *Forbidden*).

***Middleware* de manejo de errores**, implementa un sistema centralizado y tipado de manejo de excepciones implementando una clase base con el código de error y los elementos del mismo, encargándose de errores como 404 *Not Found*, 400 *Validation Error* o 500 errores internos del sistema.

***Middleware* global**, implementa filtrado de rutas y validación temprana definiendo más de 10 rutas públicas y validando *endpoints* que son accedidos por servicios externos a diferencia de la validación de los *endpoints* accedidos desde el *frontend* que ya cuentan con la validación de autenticacion y autorizacion del usuario. A su vez realiza un *tracking* de las request así como el login de los posibles errores.

Gestión de tipos y esquemas

Durante el desarrollo se vio que muchas entidades podían estar presentes en más de una capa e incluso se podían usar en más de una lógica de negocios, por lo que se optó por

centralizar los tipos de datos para el *backend* en una carpeta de tipos pudiendo ser accesibles en todo el sistema. Esta decisión arquitectónica resultó en la implementación de 5 módulos de tipos especializados: el módulo de candidatos con más de 15 interfaces para sus entidades, el módulo de compañía para operaciones corporativas, el módulo de *matching* con definiciones del algoritmo de *matching* y sus resultados, el módulo de vacantes para esquemas de ofertas laborales, y el modelo de generación para integración con servicios de IA. La centralización de tipos permitió lograr reutilización entre capas sin duplicación, tipado fuerte en toda la aplicación, sincronización automática con *Prisma Client*, e *IntelliSense* completo durante el desarrollo.

Complementario a este sistema de tipos, se implementaron esquemas de validación robustos utilizando *Zod*, herramienta que nos permite manejar la validación de la información en procesos complejos como podría ser la creación de la cuenta o de vacantes de trabajo, integrándose con nuestros tipos y esquemas mediante *type inference* automático asegurando consistencia entre la validación y el tipado.

Integración con servicios externos

Además de las capas, una parte importante de nuestro sistema es la implementación de modelos especializados para integrarnos a servicios externos como el envío de emails para notificaciones y validación de cuentas y la integración con *Amazon S3* para el almacenamiento de archivos y documentos.

El sistema de notificaciones utiliza *Nodemailer* junto con *Handlebars template* para el envío de emails dinámicos, así como también *S3* para el almacenamiento de más de 8 templates para diferentes eventos del sistema y un sistema de *queue-based processing* para la escalabilidad.

Al centralizar estas integraciones en una misma carpeta, se logra un diseño más organizado, reutilizable y fácil de mantener, garantizando que las dependencias externas estén aisladas y desacopladas de la lógica central del negocio.

En resumen, la arquitectura de Whizdy *App* implementa patrones como *Repository Pattern* [33], implícito donde *Prisma* actúa como *repository* con métodos tipados, *Service Layer Pattern* [33] con clases especializadas por dominio, *Factory Method* [38] para servicios configurables, y *Dependency Injection* [37] mediante variables de entorno,

garantizando escalabilidad horizontal, mantenibilidad del código, *testing* independiente de componentes, y observabilidad completa a través de *logging* estructurado y monitoreo integrado.

8.2.3. Modelo de datos

El esquema de persistencia de la aplicación es enteramente relacional, donde cada entidad principal del dominio (como usuarios, candidatos, empresas, ofertas de trabajo, etc.) se materializa en una tabla independiente. Esta estructura facilita la reutilización de tablas y relaciones, permitiendo que las entidades compartan atributos comunes y se puedan extender o relacionar fácilmente a través de claves foráneas (*FK*), lo que garantiza la integridad referencial y la consistencia de los datos.

El modelo implementa un patrón de entidades de referencia compartidas donde conceptos como *Hard Skills*, ubicaciones geográficas, industrias y niveles de experiencia se modelan como entidades independientes referenciadas desde múltiples contextos. Por ejemplo, las *Hard Skills* se utilizan simultáneamente en perfiles de candidatos (competencias que poseen), ofertas de trabajo (requisitos técnicos), algoritmos de *matching* (criterios de compatibilidad), y *analytics* (análisis de demanda del mercado). Esta normalización evita duplicación, asegura consistencia semántica y facilita el mantenimiento de catálogos maestros.

Sobre las tablas con mayor cantidad de consultas (candidatos, vacantes, *matches*) se definieron índices simples en los identificadores y estados más consultados y también algunos índices compuestos que cubren las combinaciones de criterios habituales en los procesos de búsqueda y *matching*. Esta estrategia reduce el coste de las operaciones sin incrementar la sobrecarga de mantenimiento.

El sistema de migraciones de *Prisma* permite evolución controlada del esquema de base de datos, manteniendo historial completo de cambios y facilitando *deployment* consistente entre entornos, permitiendo *rollback* inmediato si fuera necesario. Cada migración es versionada y puede incluir tanto cambios de estructura (*DDL*) como modificaciones de datos (*DML*), asegurando que la base de datos pueda evolucionar sin pérdida de información.

La implementación incluye un sistema robusto de *seeds* que son utilizados; en primer lugar, para precargar los datos predefinidos que la plataforma requiere para funcionar (catálogos de habilidades, industrias, ubicaciones, etc.); además, para poblar la base de datos con candidatos ficticios, empresas de prueba, ofertas de trabajo variadas y relaciones de *matching* que permiten probar todas las funcionalidades del sistema en entornos de desarrollo y *testing*.

Complementario a lo que mencionamos anteriormente sobre la posibilidad de poblar mediante *seeds* con casos de pruebas los ambientes bajos, se cuenta con funcionalidades de importación y exportación de datos con los cuales se puede transportar datos necesarios entre entornos o poder facilitar ciertos backups o actualizaciones utilizando estas funcionalidades.

Es posible ver el esquema del modelo de datos en el anexo 16.18 “Modelo de datos”.

8.3. Whizdy Matching

Como ya se mencionó, Whizdy Matching es un servicio independiente de emparejamiento que opera bajo una arquitectura de capas claras: una capa de *API* que expone *endpoints*, una capa de lógica que orquesta el procesamiento de *embeddings* y la ejecución de algoritmos de similaridad, y una capa de datos vectoriales que gestiona el almacenamiento y recuperación de representaciones semánticas en *Qdrant*. Además, cuenta con un consumidor de mensajes asincrónicos para consumir los mensajes almacenados en *RabbitMQ*, pero ahondaremos en eso más tarde.

Utiliza *embeddings* y búsqueda semántica para conectar candidatos y vacantes con precisión, además de generar texto mediante *prompting* en áreas específicas. Su diseño como microservicio desacoplado permite escalar y evolucionar sin afectar la aplicación principal.

Pipeline de procesamiento

El *pipeline* de Whizdy Matching opera como un flujo de datos en cuatro etapas principales. Primero, los datos de candidatos y ofertas laborales ingresan al sistema, donde son transformados y normalizados desde su formato original hacia estructuras internas consistentes. En la segunda etapa, estos datos estructurados se procesan mediante múltiples modelos de *embeddings* especializados que generan representaciones vectoriales semánticamente ricas para diferentes aspectos del perfil profesional: información básica, habilidades técnicas, educación, competencias blandas, entre otros. La tercera etapa almacena estos vectores en *Qdrant*, una base de datos vectorial optimizada que mantiene colecciones separadas para candidatos y trabajos con índices especializados para búsquedas de similitud eficientes. Finalmente, cuando se solicita un *matching*, el sistema ejecuta un algoritmo híbrido que combina filtrado por criterios objetivos con múltiples búsquedas vectoriales paralelas, fusiona los resultados, aplica ponderaciones personalizables y entrega una lista ordenada de candidatos compatibles con *scores* detallados por categoría.

Operaciones principales

El servicio procesa tres tipos de operaciones: generación de *embeddings* para perfiles de candidatos, generación de *embeddings* para ofertas laborales, y cálculo de *matching* entre candidatos y vacantes. Cada operación transforma los datos de entrada mediante modelos

especializados y almacena los resultados vectoriales correspondientes en la base de datos vectorial para su posterior consulta y comparación.

8.3.1. Algoritmo de *matching* detallado

Basándose en los aprendizajes y validaciones de la prueba de concepto, se diseñó el algoritmo de *matching* para el sistema en producción. Los conceptos fundamentales validados durante la *PoC*, *embeddings* separados por dimensión, concatenación de vectores, y búsqueda por similitud semántica, fueron la base de esta implementación. Sin embargo, se realizaron mejoras y ajustes significativos una vez aplicados en el contexto del sistema real de Whizdy Matching.

El algoritmo se ejecuta en cinco pasos principales que transforman la información de candidatos y vacantes en rankings ordenados con justificaciones explicables.

Paso 1: Generación de *embeddings*

El sistema genera dos tipos diferentes de vectores para representar la información: *embeddings* textuales generados mediante modelos de lenguaje, y vectores numéricos construidos directamente a partir de datos estructurados.

Para cada vacante se generan seis vectores que capturan diferentes aspectos del puesto. Cinco de ellos son *embeddings* textuales: el primero representa los detalles generales de la vacante incluyendo descripción, responsabilidades y cultura organizacional; el segundo codifica las habilidades técnicas requeridas con sus niveles de competencia y si son obligatorias o preferidas; el tercero captura los requisitos educativos; el cuarto representa los lenguajes necesarios para el puesto; y el quinto codifica la experiencia esperada en industrias específicas. El sexto vector es de naturaleza distinta: un vector numérico de 24 dimensiones que representa las competencias blandas requeridas, donde cada dimensión corresponde a una habilidad específica (como liderazgo, comunicación o trabajo en equipo) ordenada alfabéticamente, con valores normalizados entre 0 y 1 según la importancia asignada en una escala de 1 a 5.

De manera análoga, para cada candidato se generan seis vectores correspondientes. Los cinco *embeddings* textuales incluyen: información básica del perfil con objetivos profesionales, preferencias de trabajo y ubicación; un resumen de la experiencia profesional

destacando logros y responsabilidades; credenciales educativas y certificaciones obtenidas; lenguajes que domina con sus niveles de competencia; y experiencia acumulada en diferentes industrias. El sexto vector es también numérico, representando las habilidades blandas autoevaluadas por el candidato en una escala de 1 a 10, normalizada entre 0 y 1.

La construcción del texto para cada dimensión sigue una estructura específica que facilita la generación de *embeddings* significativos. Por ejemplo, para habilidades técnicas de una vacante se construye un texto que especifica cada tecnología junto con el nivel de competencia esperado y si es requisito obligatorio o preferible, haciendo explícita información que de otro modo quedaría ambigua.

Paso 1.5: Filtrado preliminar basado en metadata

Antes de realizar las búsquedas vectoriales semánticas, el sistema aplica filtros estrictos basados en metadata para reducir el espacio de búsqueda sólo a candidatos potencialmente compatibles. Estos filtros incluyen verificar que el candidato esté abierto a oportunidades laborales, que el tipo de empleo deseado (tiempo completo, *part-time*, *freelance*) coincida con la vacante, que el ambiente de trabajo preferido (presencial, remoto, híbrido) sea compatible, y que las expectativas salariales del candidato estén dentro del rango ofrecido por la vacante.

Para la ubicación geográfica, el filtro aplica una lógica más flexible: un candidato pasa el filtro si su ubicación coincide con alguna de las ubicaciones de la vacante, o si ha indicado disposición para reubicarse. Esto permite considerar tanto talento local como candidatos dispuestos a mudarse.

Paso 2: Búsquedas vectoriales independientes

A diferencia del enfoque simplificado de la *PoC* donde se concatenaban todos los *embeddings* en uno solo, el sistema en producción realiza seis búsquedas vectoriales independientes en *Qdrant*, una por cada dimensión, comparando el vector correspondiente de la vacante contra el vector equivalente de los candidatos por ejemplo las habilidades técnicas requeridas contra las habilidades técnicas del candidato.

El sistema utiliza la funcionalidad de *Named Vectors* de *Qdrant*, donde múltiples vectores diferentes se almacenan en la misma colección identificados por nombres específicos. Esto permite realizar búsquedas especificando exactamente qué par de vectores comparar en cada dimensión. Todas las búsquedas utilizan similitud coseno como métrica de distancia y retornan los candidatos que superen un umbral mínimo de similitud, optimizando así el rendimiento al descartar matches claramente irrelevantes.

Paso 3: Fusión de *rankings* y agregación de *scores*

Los resultados de las seis búsquedas independientes se combinan para generar un *ranking* final unificado. El sistema calcula primero un *score* mediante *Reciprocal Rank Fusion* (*RRF*), técnica que identifica candidatos que aparecen consistentemente bien posicionados en múltiples dimensiones. Este *score* se almacena como métrica auxiliar para análisis posterior, favoreciendo perfiles balanceados que destacan en varias áreas en lugar de solo una.

Sin embargo, el *ranking* final se determina exclusivamente mediante el *score* ponderado de similitud coseno. Si la vacante tiene pesos personalizados configurados, cada dimensión contribuye al *score* final según su importancia asignada por el empleador. Esto permite, por ejemplo, que una vacante técnica priorice las habilidades técnicas sobre otros aspectos, o que una posición de liderazgo ponga mayor énfasis en habilidades blandas. Si no hay pesos personalizados, el sistema aplica una distribución equilibrada entre las dimensiones principales.

Paso 4: Generación de explicación

Para los candidatos mejor rankeados, el sistema genera explicaciones textuales mediante un *prompt* a *Gemini*. Se envían los *scores* de compatibilidad por cada dimensión junto con información contextual de la vacante, y el modelo de lenguaje genera una explicación en lenguaje natural que destaca las fortalezas del candidato para ese puesto específico y, cuando es relevante, áreas donde podría tener menor alineación. Esto proporciona transparencia y justificación para las decisiones de *matching*.

8.3.2. Mejoras posteriores a la PoC

Cambio del modelo de *embeddings*

Entre las mejoras más importantes se encuentra el cambio del modelo de *embeddings*: mientras que la PoC utilizó modelos pre-entrenados de *Sentence Transformers*, el sistema en producción incorporó un modelo con *fine-tuning* mediante *LoRA* (*Low-Rank Adaptation*), técnica de *Parameter-Efficient fine-tuning* que permitió especializar el modelo para el dominio de recursos humanos. Este cambio fue crítico porque el modelo general *paraphrase-mpnet-base-v2* presentaba una limitación fundamental: confundía habilidades técnicas de dominios diferentes debido a similitudes lingüísticas superficiales.

El problema se manifestaba especialmente en matches entre dominios profesionales distintos. Por ejemplo, al buscar un *Financial Analyst* que requería *Financial Modeling*, Excel avanzado y conocimientos de *GAAP*, desarrolladores *Python* obtenían scores técnicos altos porque el modelo interpretaba "experto en Excel" como semánticamente similar a "experto en *Python*" (ambos procesadores de datos), *"Financial Modeling*" como similar a "*Data Modeling*" (ambos contienen "*modeling*"), y "competente en *GAAP*" como similar a "competente en *Django*" (misma estructura gramatical de nivel de habilidad). El modelo, entrenado para identificar paráfrasis y similitud semántica general, capturaba patrones lingüísticos pero no comprendía que *Python* no es equivalente a Excel, que *Web Development* no es *Finance*, ni que un *Senior Developer* no tiene las mismas competencias que un Senior *Financial Analyst*.

La solución implementada utilizó el modelo *BAAI/bge-large-en-v1.5* con el adaptador *LoRA shashu2325/resume-job-matcher-lora*, específicamente entrenado en descripciones de trabajo reales, CVs profesionales, y pares validados de match/no-match del dominio de recursos humanos. Este entrenamiento especializado permite al modelo comprender que *Financial Modeling* es una habilidad de finanzas no relacionada con programación, que profesionales senior en diferentes dominios no son intercambiables, y que las habilidades técnicas son específicas del dominio y no transferibles basándose únicamente en similitud textual.

Este modelo genera *embeddings* de 1024 dimensiones que capturan con mayor precisión los matices específicos del lenguaje usado en descripciones de vacantes y perfiles profesionales. Las pruebas de validación demostraron mejoras significativas: mientras el modelo original asignaba scores técnicos de 0.71 a desarrolladores *Python* para posiciones de *Financial Analyst*, el modelo especializado reduce estos scores a 0.49, correctamente identificando la baja compatibilidad, mientras asigna scores de 0.95 a profesionales de finanzas reales. Los resultados completos de validación con este modelo final en condiciones de producción se documentan en anexo 16.17.3.

Otras mejoras implementadas

Otras mejoras incluyeron la incorporación de filtros preliminares por metadata que mejoran significativamente el rendimiento al reducir el espacio de búsqueda antes de las comparaciones vectoriales costosas, y la diferenciación entre vectores textuales semánticos y vectores numéricos estructurados según la naturaleza de la información a representar. Los vectores de *soft skills*, por ejemplo, se representan como vectores numéricos de 24 dimensiones con valores normalizados, permitiendo comparación directa mediante distancia euclidiana en lugar de similitud semántica, lo cual resulta más apropiado para competencias evaluadas en escalas numéricas.

8.4. API

A nivel de *API* Whizdy implementa una arquitectura *RESTful* pura que sigue estrictamente los principios fundamentales del protocolo *HTTP* y las convenciones *REST* [16]. La *API* expone recursos a través de *URLs* descriptivas y utiliza los métodos *HTTP* estándar (*GET, POST, PUT, DELETE, PATCH*) para definir las operaciones disponibles sobre cada recurso.

Además se implementan convenciones de nomenclatura consistentes utilizando sustantivos en plural para los recursos principales (***/candidates***, ***/companies***, ***/jobs***) y siguiendo patrones *RESTful* estándar. Los *endpoints* están organizados por contexto funcional, separando claramente las operaciones relacionadas con candidatos, empresas, trabajos, autenticación y servicios auxiliares como *matching* y generación de contenido. Por ejemplo, rutas como ***/api/candidates/stats/process-overview*** y ***/api/candidates/stats/interview-overview*** demuestran una organización clara donde los recursos principales (*candidates*) contienen sub-recursos (*stats*) con operaciones específicas (*process-overview, interview-overview*). Esta estructura facilita la navegabilidad y comprensión de la *API* por parte de los desarrolladores que la consumen.

Por otro lado, implementa uso apropiado de códigos de estado *HTTP* para comunicar el resultado de las operaciones de manera estándar. Las respuestas exitosas utilizan códigos 2xx apropiados, mientras que los errores se comunican mediante códigos 4xx para errores del cliente y 5xx para errores del servidor. Esta implementación permite a las aplicaciones cliente manejar diferentes escenarios de manera robusta sin depender únicamente del contenido de la respuesta. Como se mencionó anteriormente estos códigos de error y mensajes claros son manejados por un *middleware* de error que tiene como función controlar los errores y dar la respuesta más acertada posible.

Para favorecer el rendimiento y una usabilidad más continua, se adoptó un mecanismo de caché a nivel de cliente que conserva temporalmente las respuestas de la *API*. Su tiempo de vida es limitado y configurable mediante variables de entorno. Por defecto, el caché se aplica a todas las consultas, manteniendo en el cliente los resultados para ofrecer respuestas inmediatas en accesos posteriores. No obstante, a nivel de desarrollo cada *endpoint* es configurable de forma independiente para habilitar o deshabilitar el uso de caché cuando la naturaleza del dato lo amerite, evitando mostrar información desactualizada en escenarios

sensibles a la frescura. En contrapartida, para recursos de referencia y catálogos de baja variabilidad, el caché se utiliza de manera explícita para evitar solicitudes repetitivas e innecesarias al *backend*.

En cuanto a la documentación se decidió implementar *Swagger* [21]/*OpenAPI* para tener una documentación completa que describe todos los *endpoints* disponibles, sus parámetros, tipos de respuesta y códigos de estado. Es posible acceder a ella en la ruta /docs, teniendo la posibilidad de explorar la *API* directamente desde el navegador, probar *endpoints* en tiempo real, y comprender rápidamente las capacidades disponibles. (véase anexo 16.24. Evidencia - *Swagger*).

9. Arquitectura del Sistema

Este capítulo presenta la arquitectura del sistema Whizdy, describiendo las decisiones de diseño estructural adoptadas para satisfacer los requerimientos funcionales y no funcionales identificados. Se detalla la organización general del sistema y la comunicación entre los componentes.

9.1. Visión general y decisiones arquitectónicas

9.1.1. Objetivos de la arquitectura

El diseño arquitectónico de Whizdy fue guiado por un conjunto de aspectos específicos derivados de los requerimientos funcionales y no funcionales del sistema:

Rendimiento

Al incluir algoritmos de *matching* donde se requieren cálculos computacionales complejos, podríamos asumir que la plataforma iba a contar con un procesamiento intensivo al realizar estas operaciones: Los algoritmos de *matching* requieren recursos computacionales complejos para la creación de *embeddings* y consultas de similitud vectorial, lo que podría impactar la responsividad de la aplicación principal de estar juntos.

Otro aspecto del rendimiento es la escalabilidad independiente, donde la demanda de procesamiento de las features del módulo de *matching* pueden crecer de manera diferente a las operaciones convencionales de la aplicación.

Por último, la interfaz de usuario debe de mantener tiempos de respuesta aceptables para los usuarios, sin verse afectada a los posibles procesos en background que haga el *matching*.

Mantenibilidad

Del lado de mantenibilidad, el motor de *matching* requería de experimentación con diferentes modelos de IA y librerías especializadas para su desarrollo y funcionamiento. Además, los cambios en algoritmos de *matching* no deben afectar la estabilidad y tampoco las funcionalidades básicas de la aplicación.

Confiabilidad:

La falla del sistema de *matching* no debe comprometer funcionalidades básicas de la aplicación como la autenticación, gestión de perfiles o navegación en general.

Además hay una diferencia de disponibilidad entre la aplicación y el *matching*, mientras que la aplicacion deberia de estar levantada en todo momento para su uso, el *matching* podría estar levantado menos tiempo por ejemplo sólo cuando se tengan que correr ciertas operaciones de generación de *embeddings* o de similitud semántica.

Debido a estos aspectos es que decidimos separar por un lado el Whizdy Matching del Whizdy App. Buscando separar las cargas computacionales del sistema de *matching* mientras se preserva la simplicidad operativa de Whizdy App, permitiendo también mantener a ambos de manera independiente sin que afecten las evoluciones de estos componentes al otro y a su ciclo de despliegue.

¿Cuáles son los beneficios de la separación?

- **Desacoplamiento operacional**: contamos con despliegues independientes que permiten actualizaciones de un componente sin afectar al otro, agregando nuevas features a la aplicación principal sin afectar el procesamiento de *embeddings* y viceversa; mejorar el algoritmo de *matching* sin afectar el flujo de uso normal o de gestión de los usuarios de la aplicación
- **Optimización específica**: Cada servicio puede optimizarse para sus cargas de trabajo particulares, por ejemplo optimizar el *matching* para procesamiento complejos y extensos pero pocas veces por dia, a comparación de la aplicación donde se podrían tener muchas interacciones continuamente pero que son menos intensivas computacionalmente
- **Escalado independiente**: De la mano con el punto anterior, la Posibilidad de escalar recursos computacionales sólo donde son necesarios, por ejemplo aumentar las instancias de *matching*
- **No propagación de fallos**: La falla de un servicio no compromete completamente la funcionalidad del otro servicio ni del sistema en general.

9.1.2. Patrones arquitectónicos

La arquitectura de Whizdy implementa una mezcla de patrones que combina elementos de diferentes estilos arquitectónicos para satisfacer los requerimientos específicos del dominio:

Arquitectura de servicios [18]

Decidimos separar nuestra arquitectura en exactamente dos servicios especializados y muy bien definidos en lugar de una descomposición granular completa por ejemplo como lo podría ser separar la aplicación por recursos o por features.

La idea de esta separación era separar las responsabilidades sin incurrir en la complejidad operacional de múltiples microservicios. Esto nos permite que a la hora de implementarlo tener la posibilidad de manejar las tecnologías específicas y optimizadas y cada servicio con su base de datos específica para el dominio del problema que resuelve.

Event-Driven Architecture [19]

Contamos con comunicación entre servicios mediante cola de mensajes para las operaciones computacionales exigentes del *matching*, permitiendo desacoplar la aplicación de la espera por el procesamiento de los *embeddings* y las requests de *matching*, permitiendo ser tolerantes a los fallos del *matching* asi como tambien permitido escalar el procesamiento.

API-First Design [20]

Además de la comunicación por colas de mensajes también se definen *APIs RESTful* bien definidas en los dos sistemas, permitiendo integrar así el Whizdy Matching y el Whizdy App de otra manera diferente.

Por un lado Whizdy App define *APIs RESTful* bien definidas para el uso del *frontend* y también para la integración con Whizdy Matching para el almacenamiento de los resultados del *matching*. Mientras que Whizdy Matching solo define el *API* para la comunicación con el app.

Al utilizar *APIs* definimos contratos explícitos junto con la documentación automatizada con *OpenAPI* de las mismas.

9.1.3. *Trade-offs*

Complejidad de desarrollo vs. Mantenibilidad y responsabilidad

Aceptamos una mayor complejidad operacional, teniendo dos servicios, con su comunicación y su *deployment* coordinado a cambio de una mantenibilidad y unas responsabilidades acotadas y específicas para cada componente. Tenemos el beneficio de que cada servicio puede evolucionar independientemente con tecnologías especializadas y responsabilidades acotadas a costo de tener una complejidad extra en la comunicación, mayor complejidad a la hora de debuggear y resolver problemas y tener una configuración de estructura más compleja.

Complejidad de despliegue vs. Performance

Implementamos una arquitectura distribuida para optimizar la *performance* de la aplicación mientras paralelizamos en otro servicio la generación de *embeddings* y generación de *matching*, consiguiendo una escalabilidad independiente y sin bloquear la aplicación mientras se llevan a cabo estos procesamientos a el costo de tener una mayor complejidad a la hora de probar y debuggear, tener que hacer *testing* de integración y manejar fallos de red y *timeouts* así como sincronizar datos entre servicios.

Inmediatez y disponibilidad

Priorizamos la disponibilidad continua de la aplicación para un flujo de trabajo normal sobre la inmediatez de los resultados del *matching*. Implementado el procesamiento asincrono de los *embeddings* y *matching* contra el procesamiento real conseguimos el beneficio de que la interfaz permanece responsiva y permite al usuario seguir trabajando mientras el *matching* se procesa en *background*, a costo de que los usuarios experimenten una demora en obtener recomendaciones y en ser recomendados.

9.2. Vistas arquitectónicas

9.2.3. Vista de contexto

En el siguiente diagrama de arquitectura presentamos a Whizdy en un entorno operacional completo, mostrando las interacciones con el usuario, actores externos y sistemas de soporte a nuestra plataforma.

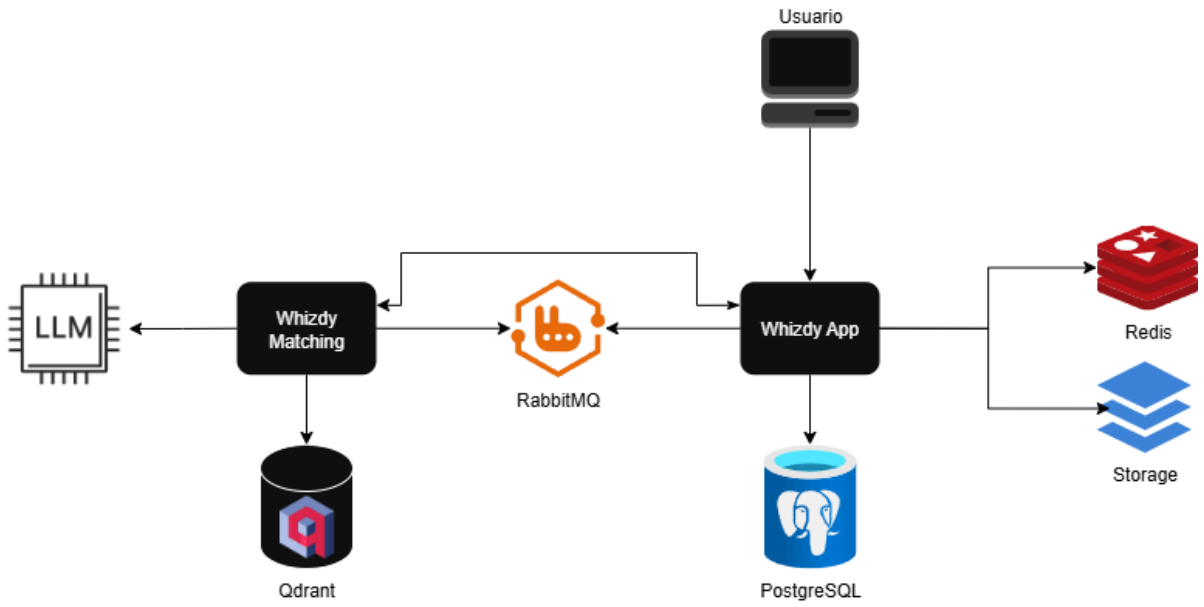


Figura 4: Diagrama Arquitectura

Usuario

Candidatos buscando oportunidades laborales o usuarios que gestionan perfiles de compañía. Interactúan con el sistema únicamente a través de la aplicación realizando el registro, inicio de sesión, gestión de perfiles, interacción con vacantes y las demás funcionalidades de la aplicación.

Sistemas externos

- *RabbitMQ*: facilita la comunicación asíncrona entre Whizdy App y Whizdy Matching, encolando mensajes para desacoplar operaciones críticas de las de procesamiento intensivo.
- *PostgreSQL*: almacena toda la información utilizada en la aplicación junto con sus relaciones, garantizando la consistencia y durabilidad de los datos.
- *Redis*: utilizado como complemento para la implementación del sistema de envío de notificaciones, funcionando como cola de mensajes que permite encolar y desencolar los trabajos de envío de notificaciones.
- *Storage*: utilizado para almacenar documentos, CVs, imágenes de perfil y proporcionando acceso seguro y escalable a estos archivos.
- *Qdrant*: almacena *embeddings* de candidatos y vacantes, y permitiendo la realización de búsquedas de similitud.
- LLM: utilizado para la generación de texto mediante instrucciones específicas.

9.3. Comunicación entre servicios

La arquitectura distribuida de Whizdy requiere mecanismos de comunicación robustos para coordinar las operaciones entre Whizdy App y Whizdy Matching. El diseño implementa dos modalidades de comunicación complementarias que responden a diferentes tipos de requerimientos operacionales: comunicación síncrona para operaciones que requieren respuesta inmediata, y comunicación asíncrona para procesamiento intensivo que puede tolerar latencia adicional.

Esta estrategia dual permite optimizar la experiencia del usuario manteniendo la responsividad de la interfaz para el flujo de trabajo normal, mientras que las tareas computacionalmente intensivas se procesan en segundo plano sin impactar la disponibilidad del sistema principal.

Las decisiones de diseño priorizan la disponibilidad y responsividad de la aplicación principal sobre la inmediatez absoluta de resultados de *matching*, implementando patrones que garantizan que los usuarios puedan continuar utilizando funcionalidades básicas incluso cuando el procesamiento de IA experimenta demoras o interrupciones.

El resto de comunicación entre servicios, como puede ser Whizdy Matching con el LLM o Whizdy App con *S3* se realiza exclusivamente mediante *HTTP* sincronicamente asegurando trazabilidad y compatibilidad entre los servicios.

9.3.1. Comunicación síncrona

La comunicación síncrona entre Whizdy App y Whizdy Matching se implementa mediante APIs RESTful que proporcionan respuestas inmediatas para operaciones que requieren *feedback* instantáneo al usuario.

Patrones implementados

1. *Request-Response* síncrono

El patrón Request-Response se materializa a través de APIs REST donde Whizdy App y Whizdy Matching actúan alternativamente como cliente y servidor según la operación.

Cada petición contiene toda la información necesaria para procesar la operación, y la respuesta incluye el resultado completo o un error explícito.

Este patrón se caracteriza por tener un acoplamiento temporal fuerte ya que el cliente espera la respuesta antes de continuar.

2. *Callback HTTP* (*webhook*)

Whizdy Matching envía resultados de procesamiento ejecutado de forma asíncrona (profundizado en la sección 9.3.2) a Whizdy App mediante *callbacks HTTP*. Aunque el procesamiento de *embeddings* y *matching* es asíncrono y puede tomar varios minutos, la entrega del resultado utiliza *HTTP* para obtener los resultados tan pronto estén disponibles. Esta decisión arquitectónica responde a restricciones de la plataforma de despliegue (detalladas en el siguiente capítulo), que no permite mantener consumidores de colas activos permanentemente en el entorno de Whizdy App.

Protocolo y formato común a ambos patrones

- *HTTP* con métodos *REST* estándar (*GET, POST, PUT, DELETE*).
- *JSON* como formato de intercambio de datos.
- *Headers* de autenticación con *tokens* para todas las operaciones.

Manejo de timeouts y fallbacks

El sistema no implementa mecanismos específicos de reintentos o manejo de timeouts para los callbacks HTTP más allá de los proporcionados por defecto por las bibliotecas utilizadas. Esta decisión se fundamenta en las características de la infraestructura de Whizdy App desplegada en *AWS Amplify* (se profundiza esto en el capítulo 10), que garantiza alta disponibilidad mediante escalado automático y balanceo de carga.

En el contexto del proyecto, se priorizó la simplicidad operativa sobre la resiliencia completa ante fallos, por lo que, para evoluciones futuras el equipo recomienda implementar:

- Sistema de reintentos con *backoff* exponencial
- Monitoreo y alertas de *callbacks* fallidos
- Persistencia temporal de resultados en Whizdy Matching

Casos de uso por patrón

Request-Response síncrono:

- Generación de texto: Cuando un usuario requiere la generación de texto, por ejemplo para la creación de vacantes o la explicación de *scores* en un *matching*, esta operación se realiza en tiempo real. El usuario espera la respuesta antes de continuar con la creación de la vacante. Whizdy App realiza una petición a Whizdy Matching, que genera un prompt al *LLM* con los templates e información brindada.

Callback HTTP:

- Envío de resultados de *matching*: Una vez que el procesamiento de *matching* entre vacantes y candidatos ha finalizado (operación que puede tardar varios minutos), Whizdy Matching envía los resultados a Whizdy App mediante un callback *HTTP*, permitiendo que la aplicación principal actualice inmediatamente el estado sin necesidad de *polling*.

9.3.2. Comunicación asíncrona

La comunicación asíncrona utiliza *RabbitMQ* como intermediario de mensajes (*message broker*) para manejar operaciones computacionalmente intensivas que pueden tolerar latencia adicional sin impactar la experiencia del usuario. Este mecanismo permite el desacoplamiento entre servicios, garantizando que el procesamiento pesado del sistema de *matching* no bloquee las operaciones de la aplicación principal.

Patrón implementado: *Publish-Subscribe* con colas persistentes

RabbitMQ implementa el patrón *Publish-Subscribe* [24] mediante intercambiadores (*exchanges*) que distribuyen eventos a múltiples consumidores utilizando claves de enrutamiento (*routing keys*). Los mensajes se almacenan en colas persistentes que actúan como *buffers* confiables, garantizando la entrega incluso ante fallos temporales de los servicios.

Arquitectura del flujo:

1. **Productores:** Whizdy App publica eventos en el exchange cuando ocurren cambios significativos
2. **Enrutamiento:** El *exchange* distribuye mensajes a las colas apropiadas según las *routing keys*
3. **Colas persistentes:** Almacenan los mensajes de forma duradera hasta su procesamiento
4. **Consumidores:** Whizdy Matching se suscribe a las colas y consume mensajes según su capacidad

Características del patrón:

- **Desacoplamiento:** Los productores no conocen ni dependen de los consumidores.
- **Persistencia de colas:** Las colas son durables y sobreviven a reinicios del *broker*.
- **Persistencia de mensajes:** Los mensajes se escriben en disco antes de considerarse entregados, garantizando durabilidad.
- **Desacoplamiento temporal:** Whizdy App y Whizdy Matching operan a ritmos independientes.
- **Absorción de picos:** Las colas acumulan mensajes durante períodos de alta demanda o cuando Whizdy Matching está temporalmente inactivo.
- **Control de flujo:** Las colas absorben picos de demanda, permitiendo que Whizdy Matching consuma mensajes gradualmente

Eventos implementados (*routing keys*):

- ***embed.candidate***: Generación de *embeddings* ante creación o actualización de perfiles de candidato.
- ***embed.job***: Generación de *embeddings* ante creación o actualización de vacantes.
- ***match.candidates_request***: Ejecución del algoritmo de *matching* ante solicitud de recálculo.

Manejo de errores y garantías

La implementación actual utiliza las configuraciones por defecto de *RabbitMQ* para reconocimiento. Para evoluciones y mejoras futuras se podría implementar:

- **Reconocimientos manuales:** Para garantizar que los mensajes sólo se eliminan tras un procesamiento exitoso.
- ***Prefetch limit*:** Para controlar cuántos mensajes procesa cada consumidor simultáneamente.
- ***Dead Letter Queues* (DLQ):** Para capturar mensajes que fallan repetidamente.
- **Reintentos con *backoff* exponencial:** Para manejar fallos temporales.

Combinación estratégica de patrones

La arquitectura combina estos patrones estratégicamente para optimizar diferentes aspectos del sistema:

- Para operaciones críticas de *UX*: *Request-Response* garantiza *feedback* inmediato
- Para notificaciones del sistema: *Publish-Subscribe* permite distribución eficiente
- Para procesamiento pesado: Colas persistentes con *Publish-Subscribe* proporcionan confiabilidad y escalabilidad

Esta combinación permite que Whizdy mantenga alta disponibilidad para funcionalidades básicas mientras procesa operaciones complejas de manera eficiente en segundo plano, garantizando que ningún tipo de operación comprometa el rendimiento general del sistema.

9.3.3. Diagrama de secuencia

En esta sección se especificarán los diagramas de secuencia para el flujo completo de creación de vacante. Se seleccionó este caso de uso específico porque representa el escenario más exhaustivo del sistema, abarcando todas las formas de comunicación entre servicios: interacciones síncronas HTTP, mensajería asíncrona mediante *RabbitMQ*, procesamiento en background con colas de trabajo en Redis, y comunicación con bases de datos tanto relacionales (*PostgreSQL*) como vectoriales (*Qdrant*). Además, involucra todos los componentes principales de la arquitectura: *Frontend*, *Backend*, servicios de IA, sistemas de notificación y múltiples tipos de usuarios.

Fase 1: creación de la vacante

Esta fase representa la interacción síncrona del usuario con el sistema para crear una vacante y finaliza con la producción del mensaje para luego realizar el procesamiento asíncrono mediante *message broker.*

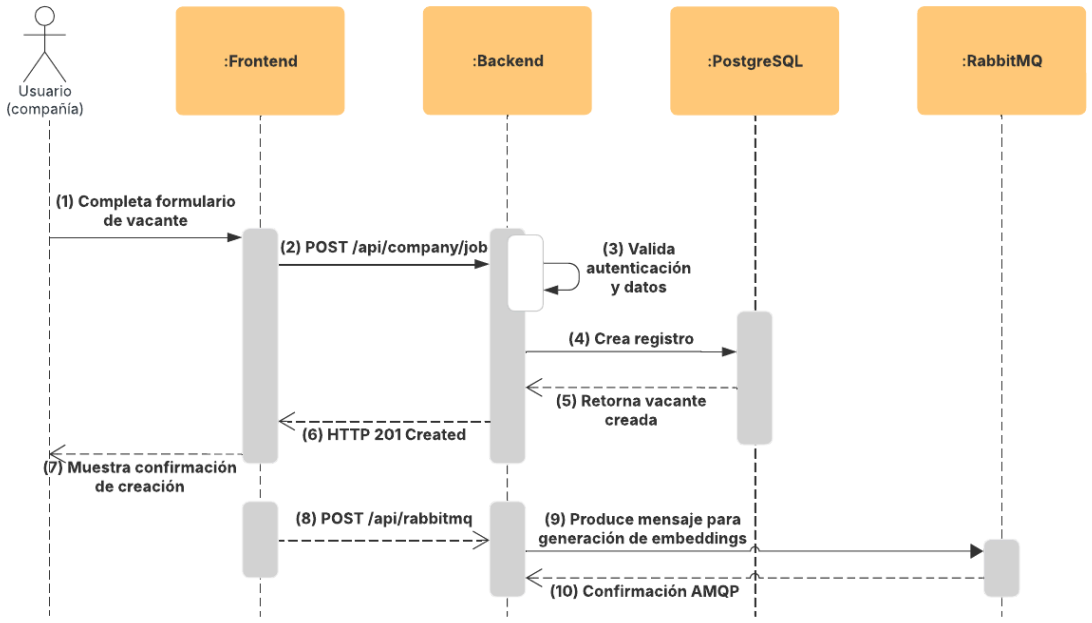


Figura 5: Diagrama secuencia (1/4)

Transición de fase 1 a fase 2

El mensaje número *(9)* de la fase 1 es el mismo mensaje que el número *(1)* de la fase 2, registrado en ambos diagramas para clarificar el punto de transición entre la fase de interacción con usuario y la de procesamiento asíncrono en background. *RabbitMQ* actúa como desacoplador entre la creación síncrona y el *matching* asíncrono.

Fase 2: procesamiento y *matching* asíncrono

Esta fase representa el procesamiento en background iniciado por el mensaje (9) de la Fase 1. El flujo comienza cuando *RabbitMQ* entrega el mensaje al servicio Whizdy Matching (mensaje 1), que procesa la vacante, genera *embeddings*, los almacena en *Qdrant*, realiza la búsqueda de candidatos compatibles y envía los resultados al *backend* mediante *callback*. La fase concluye con el guardado de los matches en *PostgreSQL* (mensaje 11), que actúa como punto de transición hacia la Fase 5 de notificaciones.

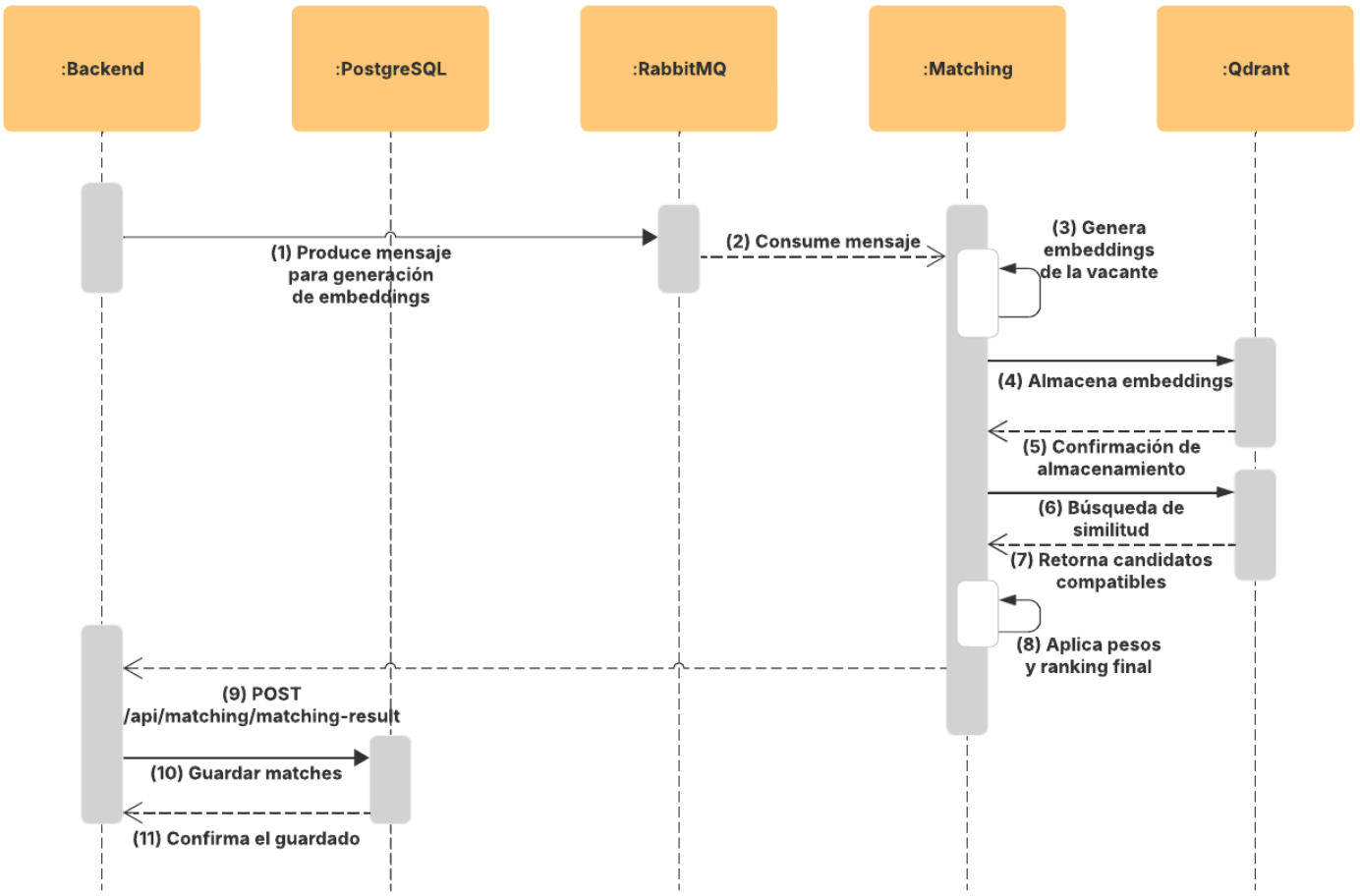


Figura 6: Diagrama secuencia (2/4)

Transición de fase 2 a fase 3

El mensaje número (11) de la fase 2, que confirma el guardado de los *matches* en *PostgreSQL*, actúa como punto de conexión hacia la Fase 3. Esta fase se presenta mediante dos diagramas de secuencia separados debido a que involucra dos flujos con características y *timing* distintos: el primero es el sistema de notificaciones, que se dispara automáticamente e inmediatamente después del guardado de *matches*; el segundo es la visualización de candidatos por parte del usuario, que ocurre de manera independiente cuando la empresa decide consultar los resultados. Aunque ambos flujos forman parte de la misma fase funcional, su separación en diagramas distintos permite una mejor comprensión de la arquitectura asíncrona del sistema.

Fase 3: procesamiento y *matching* asíncrono

Esta fase representa dos interacciones complementarias del sistema: por un lado, el envío automático de notificaciones mediante colas de trabajo en Redis (Diagrama 3A), donde el Whizdy App *backend* encola jobs que son procesados por workers para notificar vía email tanto a la empresa sobre los candidatos destacados como a los candidatos sobre la vacante con la cual son compatibles; por otro lado, la consulta síncrona de resultados iniciada por el usuario (Diagrama 3B), donde la empresa solicita ver la lista de candidatos compatibles y el *Frontend* obtiene los datos desde *PostgreSQL* para presentarlos en la interfaz. Ambos flujos operan de forma desacoplada: las notificaciones se ejecutan en background sin intervención del usuario, mientras que la visualización responde a una acción explícita del usuario que puede ocurrir en cualquier momento posterior al *matching*.

Diagrama 3A

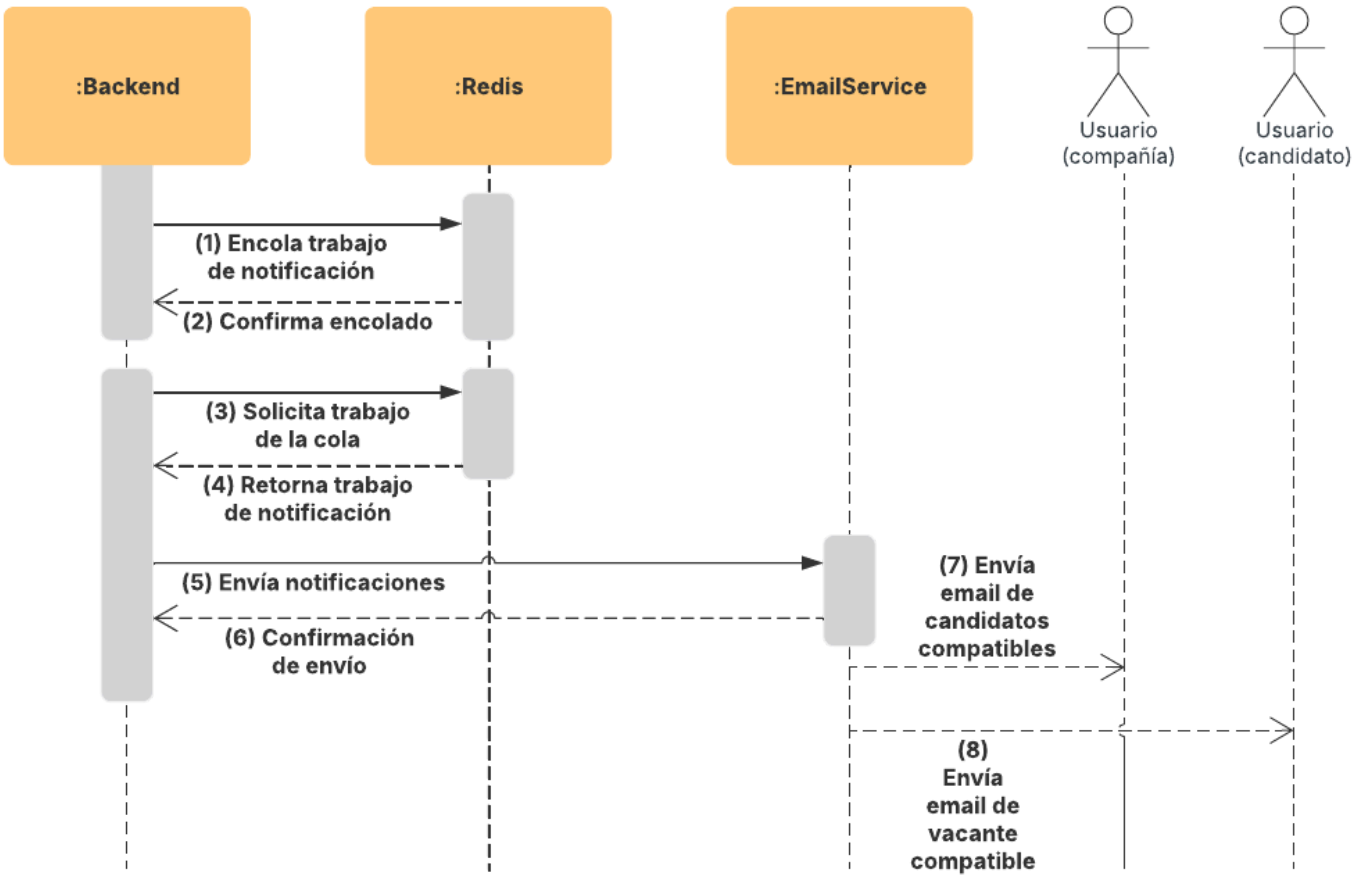


Figura 7: Diagrama secuencia (3/4)

Diagrama 3B

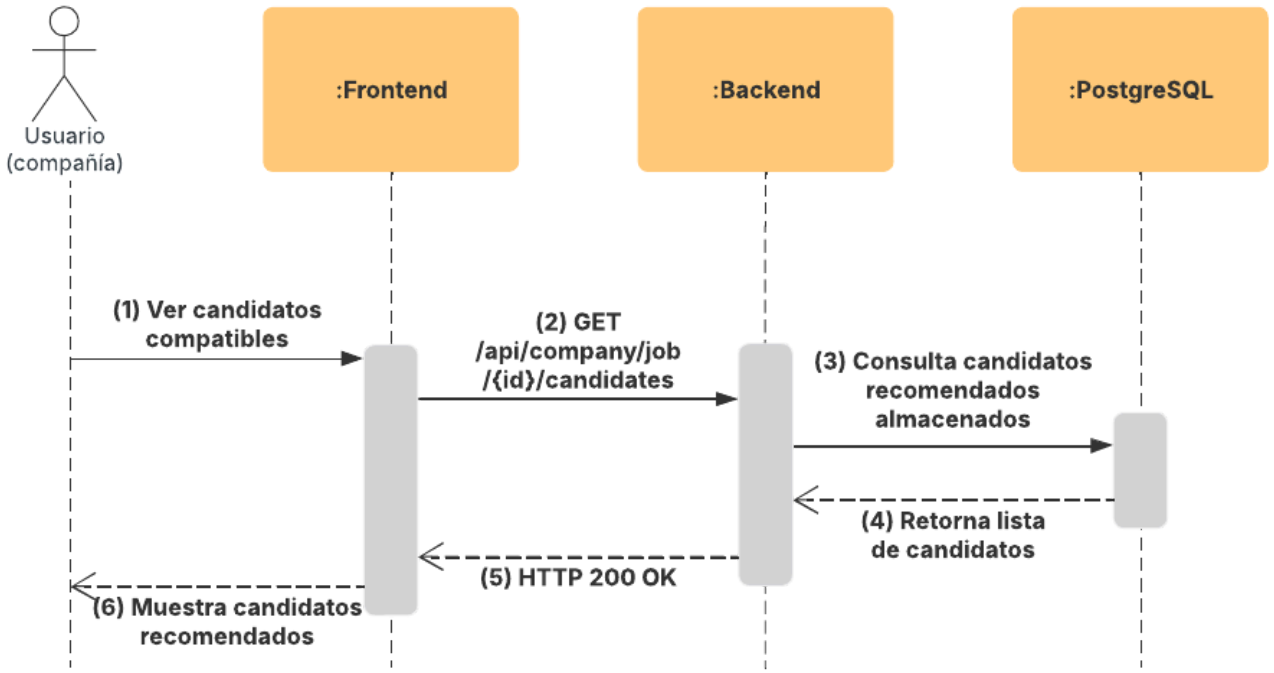


Figura 8: Diagrama secuencia (4/4)

En el anexo 16.19 se puede encontrar una descripción más detallada de cada uno de los mensajes de los diagramas de flujo.

9.4. Aplicación de atributos de calidad en la arquitectura

La arquitectura de Whizdy implementa una separación en microservicios donde Whizdy App y Whizdy Matching operan como servicios independientes comunicados mediante mensajería. Esta decisión permite que cada servicio escale de forma autónoma según su carga específica, ya que el procesamiento de *embeddings* y *matching* semántico requiere más recursos computacionales que la aplicación principal. Además, esta separación proporciona aislamiento de fallos: si el servicio de *matching* experimenta problemas o demoras, la funcionalidad básica de la aplicación permanece disponible para los usuarios.

El sistema utiliza comunicación asíncrona mediante *RabbitMQ* para todo el procesamiento computacionalmente intensivo, específicamente la generación de *embeddings* y el cálculo de *matching*. Esta decisión arquitectónica garantiza que los usuarios reciban confirmación inmediata de sus acciones en menos de dos segundos, mientras el procesamiento pesado ocurre en segundo plano sin bloquear la interfaz. Las colas persistentes de *RabbitMQ* actúan como buffers que absorben picos de demanda, permitiendo que Whizdy Matching consuma mensajes gradualmente según su capacidad sin sobrecargar el sistema.

Para la entrega de resultados del *matching*, se implementó un patrón de callbacks HTTP donde Whizdy Matching notifica activamente a Whizdy App mediante peticiones HTTP una vez completado el procesamiento. Esta decisión elimina la necesidad de *polling* constante desde la aplicación principal, reduciendo significativamente la carga en las APIs y disminuyendo la latencia en la entrega de resultados. Los resultados llegan inmediatamente cuando están disponibles, sin demoras artificiales introducidas por intervalos de *polling*.

Finalmente, la arquitectura emplea un sistema dual de mensajería adaptado a diferentes necesidades: *RabbitMQ* para el procesamiento intensivo del *matching* que requiere garantías de entrega y persistencia, y *Redis* con *BullMQ* para notificaciones ligeras que priorizan velocidad sobre durabilidad extrema. Esta especialización permite optimizar recursos al usar la herramienta más adecuada para cada caso de uso específico, mejorando tanto el rendimiento como la eficiencia operativa del sistema.

10. Infraestructura

Este capítulo describe cómo se materializó la plataforma Whizdy en producción: la arquitectura de red, los servicios cloud que la sostienen, la automatización de su despliegue y elementos importantes de la misma.

10.1. Diseño de infraestructura

Decisión Inicial: AWS Completo

El diseño inicial de infraestructura contemplaba desplegar todos los componentes del sistema en *Amazon Web Services* (*AWS*), aprovechando el programa *AWS Free Tier* para minimizar costos durante el desarrollo del producto. La mayoría de los servicios del sistema se ajustaban bien a las limitaciones del *Free Tier*: *Amazon RDS* para la base de datos *PostgreSQL*, AWS *Amplify* para el *hosting* de la aplicación y *Amazon S3* para almacenamiento de archivos.

El Problema: Whizdy Matching y *EC2*

Sin embargo, el servicio de *matching* presentaba requerimientos que excedían significativamente el *Free Tier* de *Amazon EC2*. Este componente, que ejecuta los modelos de *embeddings* y realiza las búsquedas vectoriales en *Qdrant*, demanda recursos computacionales considerables: *CPU* con capacidad de procesamiento para inferencia de modelos de lenguaje, memoria *RAM* suficiente para mantener el modelo cargado y procesar consultas simultáneas, y almacenamiento para la base de datos vectorial. Las instancias *EC2* Micro disponibles en el *Free Tier* resultaron insuficientes para ejecutar el servicio, mientras que escalar a instancias más grandes implicaba costos mensuales no previstos en el presupuesto del proyecto.

Solución híbrida: migración a Hetzner Cloud

Al comunicar esta limitación al cliente, nos informaron que ellos ya utilizaban *Hetzner Cloud* [25] para parte de su infraestructura y ofrecieron poner a disposición su cuenta para el proyecto. Esta solución presentaba ventajas múltiples: el cliente absorbía directamente los costos del servicio de *matching* eliminando la restricción presupuestaria, *Hetzner Cloud* ofrece mejor relación precio-*performance* en sus instancias *Cloud* con instancias *EC2*

equivalentes, y se mantenía la aplicación principal en *AWS* aprovechando los servicios gratuitos ya configurados. Se decidió entonces una arquitectura híbrida donde únicamente el servicio de *matching* se despliega en *Hetzner Cloud*, mientras que todos los demás componentes permanecen en *AWS*.

Despliegue en AWS

AWS funciona como el entorno principal de despliegue, albergando los servicios críticos del sistema excepto Whizdy Matching.

Aplicación Web y *CI/CD*

AWS Amplify gestiona el despliegue de Whizdy App con integración directa a *GitHub* [34] para *CI/CD* automatizado. Cada *push* a una rama ejecuta automáticamente el *build* y *deployment*, permitiendo mantener la rama *main* como producción y *develop* cómo *staging* para pruebas antes de *releases*.

Almacenamiento y Persistencia de Datos

Amazon *RDS* con *PostgreSQL* funciona como la base de datos relacional centralizada, almacenando información de usuarios, vacantes y postulaciones. La instancia cuenta con respaldos automáticos diarios y políticas de retención para recuperación ante fallos.

Amazon S3 gestiona archivos de usuarios (CVs, imágenes de perfil, banners corporativos), integrado con *CloudFront* para optimizar la distribución mediante *CDN*. El servicio tiene versionado habilitado y políticas *IAM* para control de acceso granular.

Comunicación y Caché

Para la comunicación asincrónica que mencionamos anteriormente, el servicio de *Amazon MQ* nos proporciona un *message broker* (*RabbitMQ*) que facilita comunicación asíncrona entre Whizdy App y Whizdy Matching.

Para la instancia de Redis decidimos utilizar el servicio de *Upstash* ya que nos brindaba una configuración muy sencilla y una cuota de *free tier* extensa.

Monitoreo y Seguridad

Amazon *CloudWatch* centraliza *logs*, métricas y alarmas de todos los servicios *AWS*, permitiendo identificar cuellos de botella y monitorear el *health* del sistema. *AWS IAM* gestiona seguridad mediante roles con mínimo privilegio, donde cada servicio accede únicamente a los recursos necesarios para su función.

Despliegue en Hetzner

Whizdy Matching se despliega en un servidor *Hetzner Cloud* de tipo *CPX21* (3 vCPU, 4GB *RAM*), ejecutando el servicio dentro de contenedores *Docker*. Esta configuración facilita el despliegue, actualización y reinicio automático en caso de fallos.

Dado que *Qdrant* no cuenta con un servicio gestionado específico en *AWS* o *Hetzner* (como sí ocurre con bases de datos relacionales como *RDS*), se desplegó una instancia separada de tipo *CPX11* (2 *vCPU*, 2GB *RAM*) que ejecuta *Qdrant* en un contenedor *Docker*.

Ambas instancias están conectadas a través de una red privada de *Hetzner*. La comunicación entre Whizdy Matching y *Qdrant* se realiza exclusivamente a través de esta red privada, lo que garantiza que *Qdrant* sea completamente privada y solo accesible desde la instancia de Whizdy Matching, sin exposición a internet.

Por su parte, Whizdy Matching sí expone sus servicios públicamente a través de los puertos 80, 443 y 8000 para recibir peticiones *HTTP/HTTPS* externas desde Whizdy App, mientras mantiene su conexión con *Qdrant* de forma segura a través de la red privada interna.

Matching periódico

Además del *CI/CD* para despliegues, el sistema utiliza *GitHub Actions* para automatizar procesos *batch* que requieren ejecución periódica. Específicamente, se implementó un workflow programado mediante cron que se ejecuta cada 24 horas para identificar vacantes activas que requieren actualización de sus *rankings* de candidatos y encolar solicitudes de *matching* de manera automatizada.

El *workflow* ejecuta un *script* que consulta la base de datos para identificar empresas con vacantes activas sin *matches* calculados o que requieren recálculo. Para cada vacante, construye una solicitud con el límite de candidatos y los pesos de *matching* configurados, publicándose como mensaje en la cola de *RabbitMQ*. Whizdy Matching procesa estos mensajes de forma asíncrona ejecutando el algoritmo de *matching* y enviando los resultados a la aplicación, siguiendo el mismo flujo descrito para las solicitudes de *matching* cuando se crea una vacante (el proceso de creación de vacantes fue explicado previamente en 9.3.3).

Esta arquitectura asíncrona permite que el *cronjob* se ejecute rápidamente sin esperar resultados del procesamiento, que se distribuye temporalmente según la capacidad disponible. El sistema incluye *logging* detallado para monitorear el éxito de las solicitudes publicadas y detectar problemas en la automatización.

Esta automatización garantiza que las vacantes activas, incluyendo aquellas creadas días o semanas antes, reciban actualizaciones periódicas de sus *rankings* de candidatos compatibles. Esto permite mantener un flujo continuo de talento: candidatos que se registraron después de la publicación inicial de una vacante, así como aquellos que actualizaron sus perfiles (cambios en disponibilidad laboral, nuevas experiencias, certificaciones recientes, etc.), son evaluados periódicamente y tienen la oportunidad de aparecer como matches compatibles si el algoritmo identifica alineación con los requisitos de la posición. De esta manera, los empleadores no solo reciben candidatos en el momento de publicar una vacante, sino que continúan recibiendo potenciales matches conforme el *pool* de talento disponible evoluciona, mientras que los candidatos obtienen notificaciones de oportunidades compatibles incluso para vacantes publicadas anteriormente a su registro en la plataforma.

10.1.1. Diagrama de infraestructura

La arquitectura general del sistema se representa en la siguiente imagen, donde se observan los componentes desplegados en *AWS* (aplicación principal, bases de datos, servicios de mensajería), el servicio de *matching* en *Hetzner Cloud*, y las integraciones con servicios externos:

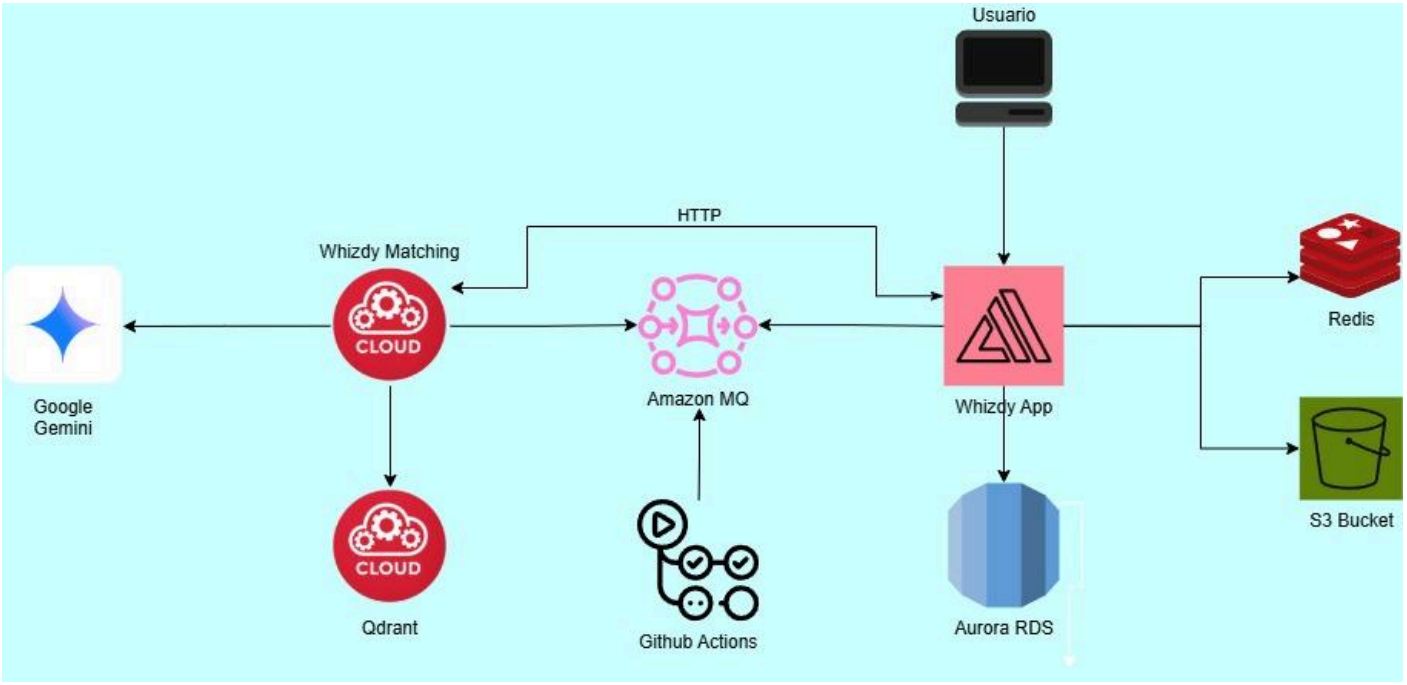


Figura 9: Diagrama de despliegue

10.2. Infraestructura como código

La infraestructura del proyecto se gestiona mediante el enfoque de Infraestructura como Código (*IaC*), utilizando *Terraform* como herramienta principal. Esta práctica permite definir, versionar y desplegar todos los recursos de infraestructura de forma declarativa, reproducible y segura, asegurando coherencia entre entornos y facilitando la modificación del sistema.

Terraform permite describir toda la infraestructura en archivos de configuración escritos en código fuente, donde se especifican desde instancias de cómputo y bases de datos hasta configuraciones de red, permisos y almacenamiento [29]. A través de estos archivos, el equipo automatiza la creación, modificación y eliminación de recursos, reduciendo errores manuales propios de configuraciones a través de interfaces gráficas y manteniendo un control claro sobre el estado del entorno.

Arquitectura Modular y *Multi-Cloud*

La infraestructura se organiza en módulos reutilizables, cada uno encapsulando recursos relacionados con responsabilidades específicas. Se definieron módulos separados para *VPC* (redes y subnets), *RDS* (base de datos *PostgreSQL*), *EC2* (instancias de cómputo), *Hetzner* (servidores cloud para *matching*), *security groups* (reglas de *firewall*), y *IAM* (roles y permisos). Esta arquitectura modular facilita el mantenimiento al aislar cambios en componentes específicos sin afectar al resto del sistema, y permite reutilizar módulos entre diferentes entornos (*staging* contra producción) con configuraciones parametrizadas.

Una ventaja relevante de *Terraform* es su capacidad de gestionar infraestructura *multi-cloud*. El proyecto despliega recursos tanto en *AWS* como en *Hetzner Cloud* desde un único conjunto de archivos de configuración, utilizando múltiples *providers* simultáneamente. Permitiendo la arquitectura híbrida descrita anteriormente, donde los servicios principales residen en *AWS* mientras que el motor de *matching* se ejecuta en *Hetzner*, todo gestionado de manera unificada y consistente.

Gestión de Secretos y Variables Sensibles

Para manejar información sensible como contraseñas de bases de datos, *tokens* de *API* y claves de acceso, se utiliza un archivo *terraform.tfvars* que contiene las variables de entorno específicas del proyecto. Este archivo está excluido del control de versiones mediante *.gitignore*, garantizando que las credenciales nunca se expongan en el historial de *Git* ni en repositorios remotos. Cada desarrollador mantiene su propia copia local con las credenciales necesarias para su entorno.

Flujo de Trabajo con *Terraform*

El flujo de trabajo con *Terraform* se basa en cuatro etapas principales:

Inicialización: Se prepara el directorio de trabajo descargando los *providers* necesarios (*AWS*, *Hetzner*) y configurando el *backend* remoto para el *state file*. Este paso establece el entorno para las operaciones subsecuentes.

Definición: Todos los recursos se declaran en archivos organizados por responsabilidad: la estructura principal del sistema, las variables configurables, la información que se expone de recursos creados, cada módulo con su propia configuración encapsulada.

Planificación: Se genera un plan de ejecución que muestra detalladamente los cambios que se aplicarían en la infraestructura: recursos a crear, modificar o destruir. Este paso permite revisión y validación antes de ejecutar cambios reales, previniendo modificaciones no deseadas en producción.

Ejecución y Versionado: Los cambios se aplican creando o actualizando recursos en *AWS* y *Hetzner* según el plan aprobado. El estado actual de la infraestructura se almacena en un archivo *terraform.tfstate* local, la manera por defecto de *Terraform*, pero una mejora importante sería el guardado remoto para garantizar que múltiples desarrolladores puedan trabajar colaborativamente en la infraestructura sin conflictos.

10.3. Gestión de configuración

El proyecto implementa una clara separación entre código y configuración, siguiendo el principio de externalización de parámetros operativos. Este enfoque permite que la misma base de código se despliegue en diferentes entornos (desarrollo, *staging*, producción) con distintos parámetros sin necesidad de modificaciones en el código fuente.

Configuración de aplicaciones

Las aplicaciones (Whizdy App y Whizdy Matching) utilizan variables de entorno para parametrizar su comportamiento. Estas variables incluyen credenciales de servicios externos, *URLs* de *endpoints* internos, configuraciones de seguridad (como secretos para *JWT*), y parámetros operativos como niveles de *logging* o habilitación de servicios de monitoreo. La externalización de estos valores evita el *hardcodeo* de configuraciones específicas del entorno en el código, facilitando la portabilidad y reduciendo el riesgo de exponer información sensible en repositorios de código.

Para gestionar estas variables de manera segura, cada entorno mantiene su propia configuración independiente que no se versiona en el control de código. Los valores sensibles como credenciales y *tokens* de *API* se almacenan de forma protegida y se inyectan en tiempo de ejecución, garantizando que nunca aparezcan en texto plano en repositorios ni en *logs* de sistema.

Configuración de infraestructura

La infraestructura gestionada con *Terraform* se parametriza mediante variables que definen características como tipos y tamaños de instancias de cómputo, configuraciones de bases de datos (capacidad de almacenamiento, políticas de respaldo, ventanas de mantenimiento), topología de red (subnets, *security groups*, reglas de *firewall*), y recursos de almacenamiento. Esta parametrización permite ajustar recursos según necesidades cambiantes de costo, rendimiento o seguridad sin modificar la lógica subyacente de aprovisionamiento.

Por ejemplo, escalar de un servidor con 2GB de *RAM* a uno con 4GB, o cambiar la frecuencia de respaldos de la base de datos, solo requiere actualizar valores de variables y re-ejecutar el proceso de despliegue, sin tocar la definición estructural de la infraestructura.

Beneficios del enfoque

Esta estrategia de configuración externalizada aporta múltiples ventajas al proyecto. Proporciona flexibilidad operativa al facilitar cambios de configuración sin redespliegues completos. Mejora la seguridad al garantizar que credenciales sensibles no aparezcan en el código fuente ni en el historial de versiones. Facilita la replicabilidad de entornos, permitiendo que cualquier miembro del equipo pueda crear un ambiente funcional completo con la configuración apropiada. Finalmente, reduce el riesgo de inconsistencias entre entornos al centralizar la definición de parámetros y automatizar su aplicación.

10.4. *CI/CD* y estrategia de despliegue

El proyecto implementa un *pipeline* de integración y despliegue continuo que automatiza la entrega de cambios desde el repositorio hasta el entorno productivo, combinando *GitHub Actions* para validación de código y *AWS Amplify* para el despliegue automático de la aplicación principal.

Integración continua con *GitHub* Actions

Cada *push* al repositorio dispara *workflows* de *GitHub Actions* que ejecutan validaciones automáticas previas al *merge* de una rama y a su despliegue. El proceso incluye *linting* para verificar que el código cumple con los estándares de estilo definidos por el equipo, ejecución de la suite de pruebas unitarias y compilación del proyecto para identificar errores tempranamente antes de que lleguen a etapas posteriores. Estas validaciones garantizan que solo código de calidad se integre a las ramas principales, funcionando como una barrera automatizada que previene la introducción de *bugs* en producción.

Despliegue continuo con *AWS Amplify*

AWS Amplify se conecta directamente al repositorio de *GitHub* mediante un *token* de acceso, monitoreando cambios en las ramas configuradas. Cuando detecta un *commit* nuevo en la rama principal (*main*), el servicio automáticamente clona el código actualizado, inyecta las variables de entorno necesarias, ejecuta las migraciones de base de datos mediante *Prisma* para sincronizar el esquema de la base de datos en producción con los cambios del código, compila la aplicación *Next.js* generando los *assets* optimizados, y finalmente despliega la nueva versión en producción reemplazando la versión anterior.

Build spec como infraestructura

Un aspecto distintivo de la configuración es que el *build spec* de *Amplify*, el archivo que define exactamente qué pasos seguir durante el despliegue, está versionado en *Terraform* como parte de la definición de infraestructura, no como un archivo separado en el repositorio de la aplicación. Esta decisión arquitectónica centraliza la configuración del *pipeline* de despliegue junto con el resto de los recursos de infraestructura, facilitando la gestión consistente y el versionado de cambios en el proceso de *build*. Por ejemplo, si se necesita

agregar un nuevo paso en el *pipeline* (cómo ejecutar *seeds* de base de datos o generar assets adicionales), el cambio se hace mediante *Terraform* y se aplica de manera declarativa.

La configuración de *auto-build* en la rama principal asegura que cada rama aprobada y mergeada a *main* se refleje automáticamente en producción sin intervención manual, reduciendo tiempos de entrega desde horas o días a minutos, y eliminando errores humanos propios de procesos manuales de despliegue.

Despliegue de Whizdy Matching en Hetzner

A diferencia de la aplicación principal, el despliegue de Whizdy Matching en *Hetzner* Cloud no cuenta con automatización completa. Cuando se realizan cambios en el código del servicio de *matching*, es necesario relevantar manualmente la instancia para aplicar las actualizaciones desde la rama main del repositorio. Este proceso no afecta a la base de datos vectorial *Qdrant*, ya que reside en una instancia separada y mantiene sus datos persistentes independientemente de los redespliegues del servicio de *matching*.

Esta estrategia de despliegue manual para el *matching* service responde a que las actualizaciones en este componente son menos frecuentes que en la aplicación principal, y que el impacto de un breve *downtime* durante el redespliegue es aceptable dado que las operaciones de *matching* no son síncronas y críticas en tiempo real.

Beneficios del pipeline automatizado

Este enfoque de *CI/CD* permite que el equipo mantenga un flujo de desarrollo ágil donde los cambios validados llegan a producción en minutos desde el *merge* a *main*. La automatización reduce significativamente el riesgo de errores humanos en el proceso de despliegue, libera tiempo del equipo que antes se dedicaba a tareas manuales repetitivas, y proporciona confianza para desplegar con mayor frecuencia, acelerando la entrega de valor a usuarios finales.

10.5. Compromisos y limitaciones de seguridad

El problema de conectividad con VPCs privadas

Durante la implementación de la infraestructura surgió una limitación técnica significativa relacionada con *AWS Amplify Hosting* que forzó compromisos en la arquitectura de seguridad del sistema. *AWS Amplify*, el servicio utilizado para desplegar Whizdy App, opera mediante una arquitectura serverless donde las *API Routes* de *Next.js* se ejecutan en funciones *Lambda* administradas por AWS. Estas funciones *Lambda* corren en la infraestructura propia de *AWS*, no en la *VPC* del usuario, lo cual imposibilita la conexión directa a recursos que residen en subnets privadas.

Esta limitación es inherente al diseño de *Amplify Hosting* y está documentada en múltiples *GitHub issues* del repositorio oficial de *AWS Amplify*, particularmente en los *issues* #3499 y #2782, donde usuarios han solicitado sin éxito la funcionalidad de configurar las *Lambda functions* para ejecutarse dentro de *VPCs* de usuario.

Implicaciones de seguridad

La consecuencia directa de esta limitación fue que servicios como *Amazon RDS* (*PostgreSQL*), *Amazon MQ* (*RabbitMQ*) y potencialmente *ElastiCache*, que idealmente deberían residir en subnets privadas sin acceso desde internet, tuvieron que ser configurados con acceso público para permitir que las *Lambda* functions de *Amplify* pudieran conectarse a ellos. Esto viola principios fundamentales de seguridad en arquitecturas *cloud*, donde las mejores prácticas dictan que bases de datos y servicios de infraestructura críticos deben permanecer en redes privadas, accesibles únicamente desde recursos dentro de la misma *VPC* a través de *security group*s restrictivos.

Exponer estos servicios públicamente, incluso con *security groups* configurados para permitir acceso solo desde rangos de IP específicos de *AWS*, incrementa la superficie de ataque y el riesgo potencial de brechas de seguridad. Aunque las credenciales de conexión permanecen protegidas mediante variables de entorno y los security groups filtran el tráfico entrante, la simple existencia de un *endpoint* público para una base de datos de producción es considerada una mala práctica en entornos de producción.

Alternativas evaluadas

Se evaluaron múltiples alternativas para resolver este problema manteniendo *Amplify* como plataforma de *hosting*. Una opción consistía en implementar una arquitectura de *proxy* mediante *API Gateway* y *Lambda functions* adicionales configuradas dentro de la *VPC* del usuario, que actuarían como intermediarios entre *Amplify* y los recursos privados. Sin embargo, esta solución introducía complejidad significativa, múltiples puntos de falla adicionales, latencia incrementada debido a los cold starts de *Lambda*, y costos operativos considerables sin resolver el problema fundamental de arquitectura.

Otra alternativa evaluada fue migrar servicios como *ElastiCache* a proveedores externos tipo *SaaS* (por ejemplo, *Upstash Redis*) que proveen *endpoints* públicos diseñados para ser accesibles desde plataformas serverless. Si bien esta solución funciona técnicamente y elimina la necesidad de exponer recursos de *AWS* públicamente, introduce vendor *lock-in* adicional, potencialmente mayor latencia por la ubicación geográfica de los servicios externos, y en el caso de bases de datos relacionales como *RDS*, no existe alternativa externa viable que mantenga el nivel de control y seguridad requerido.

Migración necesaria: *EC2* como solución

La solución arquitectónicamente correcta habría sido migrar el despliegue de la aplicación principal desde *AWS Amplify* hacia instancias *EC2* dentro de subnets públicas de la misma *VPC* donde residen los recursos privados. En esta arquitectura, la instancia *EC2* que aloja la aplicación *Next.js* tendría conectividad a *RDS*, *Amazon MQ* y otros servicios a través de la red interna de la *VPC*, sin necesidad de exponer ningún servicio públicamente excepto la aplicación web misma. Los *security groups* podrían configurarse de manera restrictiva: *RDS* aceptaría conexiones únicamente desde el security group de la instancia *EC2*, *Amazon MQ* seguiría el mismo patrón, y todos los servicios permanecerían completamente inaccesibles desde internet.

Esta arquitectura podría implementarse utilizando *Docker* para que la aplicación *Next.js* viviera dentro de un container, facilitando despliegues consistentes y escalabilidad horizontal futura. Opcionalmente, para mayor robustez, podría migrarse a *Amazon ECS* con *Fargate*, que provee orquestación de contenedores gestionada manteniendo la conectividad *VPC* nativa. *ECS* Fargate permitiría además configurar *auto-scaling* basado en métricas como

CPU o cantidad de *requests*, distribuir carga mediante *Application Load Balancers*, y desplegar múltiples instancias en diferentes *availability zones* para alta disponibilidad.

Razones para no implementar la migración

A pesar de que la migración a *EC2* o *ECS Fargate* representaría una mejora sustancial en la arquitectura de seguridad y se alinearía con las mejores prácticas de *AWS*, se decidió mantener el despliegue en *Amplify* con los compromisos de seguridad mencionados. Esta decisión se fundamentó en el contexto y objetivos del proyecto como trabajo académico de desarrollo del producto. El alcance del proyecto priorizaba la validación de funcionalidades del sistema de *matching*, la integración de componentes de IA, y la entrega de una aplicación funcional *end-to-end*, sobre la optimización de aspectos de infraestructura que, si bien importantes en producción, no afectaban la validación técnica del concepto principal.

Adicionalmente, *AWS Amplify* proporcionaba ventajas significativas en términos de velocidad de iteración y simplicidad operativa: el *pipeline* de *CI/CD* completamente integrado y gestionado por *AWS* redujo drásticamente el *overhead* de configuración y mantenimiento de infraestructura, permitiendo al equipo enfocarse en desarrollo de funcionalidades. La integración nativa con *GitHub*, el manejo automático de builds, la distribución mediante *CloudFront*, y el versionado por rama simplificaron el flujo de trabajo en un proyecto con restricciones de tiempo.

Para una eventual transición a producción real con más usuarios, empresas y posiblemente objetivos comerciales, la migración a una arquitectura con *VPC* completamente privada sería imperativa y debería ser una de las primeras tareas en el *roadmap* post-académico del proyecto. El código de infraestructura en *Terraform* está estructurado modularmente de manera que esta migración pueda realizarse sin necesidad de reescribir componentes, simplemente reemplazando el módulo de *Amplify* por un módulo de *EC2* o *ECS* y ajustando las configuraciones de red correspondientes.

11. Aseguramiento de la calidad

Este capítulo detalla las estrategias implementadas para verificar los requerimientos no funcionales y atributos de calidad, así como los mecanismos de *testing* y control continuo que aseguran la correcta operación de cada componente. El objetivo es demostrar que la solución no solo satisface las funcionalidades esperadas, sino que también mantiene un nivel de calidad consistente, medible y sostenible en el tiempo.

11.1. Cumplimiento de requerimientos no funcionales

11.1.1. Seguridad y Privacidad de Datos

Para garantizar la seguridad los usuarios tienen que estar autenticados, no se puede hacer uso de la plataforma sin estar autenticado. Para ello hay dos métodos de autenticación, la primera mediante credenciales tradicionales (email/contraseña con *hash* seguro y *salting*) y la segunda con *OAuth 2.0* con *Google* para simplificar el acceso y aprovechar su infraestructura de seguridad.

Por otro lado, la autorización se basa en roles aplicados por el *middleware* de autorización que intercepta cada solicitud *HTTP* y valida los permisos requeridos antes de permitir el acceso, lo que posibilita aplicar reglas de seguridad de forma granular, incluso a nivel de *endpoint* y operación. Se tiene dos roles que corresponden a los dos perfiles de la plataforma (candidato y empresa), pero se mantiene la posibilidad de a futuro poder agregar nuevos roles fácilmente.

Las sesiones son stateless mediante *JWT* (*JSON Web Tokens*) firmados con algoritmos seguros y expiración [30]. Para operaciones críticas como cambio de contraseña o modificación de datos sensibles, se implementó autenticación de dos factores (2FA) mediante códigos *OTP* enviados por email y generados por algoritmos criptográficos seguros.

Para garantizar la privacidad de información sensible, las contraseñas son almacenadas mediante un *hash* con secreto para su legibilidad en caso de ser expuestas. Por otro lado, los CVs son almacenados en *S3* como privados y solo se puede acceder a estos utilizando un *token* que es validado a la hora de solicitar un CV y descargarlo. Por último, los

datos sensibles tampoco son expuestos en el sistema de *logs*, el cual se hablará más adelante en esta sección, evitando posible exposición de estos.

La comunicación entre el cliente y el servidor está protegida mediante el uso de *CORS* y autenticación, permitiendo únicamente orígenes autorizados y previniendo ataques de tipo *CSRF*.

Por otro lado, *Gemini* se integra para el procesamiento de lenguaje natural en tareas específicas. Para esta integración se aplican mejores prácticas de seguridad *API*: validación y sanitización de entradas antes de enviarlas a la *API*, uso de rate *limiting* para prevenir abusos, manejo de fallos y failover en caso de indisponibilidad, y gestión segura de las claves de acceso mediante variables de entorno, que nunca se exponen en el código fuente.

Además de los mecanismos técnicos, se aplicó seguridad por lógica de negocio para reforzar la segregación de información. A nivel de interfaz, los perfiles de candidato y empresa están completamente separados: rutas, menús y *APIs* son exclusivas para cada rol; un candidato no puede invocar *endpoints* de empresa y viceversa. Dentro del mismo perfil, cada usuario solo accede a sus propios datos, por ejemplo un candidato únicamente tiene acceso a sus elementos y datos, como su CV, sus postulaciones, sus notificaciones, etc, mientras que de la misma forma una empresa solo ve sus vacantes y candidatos. Esto Se refuerza también a nivel de *middleware* de autorización, el cual valida que el *token* del usuario coincida con el id del recurso solicitado, lo que impide listar o modificar registros ajenos.

11.1.2. Disponibilidad

Para asegurar que la plataforma esté accesible en todo momento, se implementaron *health* checks que permiten monitorear el estado de cada servicio. Estos *endpoints* exponen información sobre el estado operativo del servicio, permitiendo detectar irregularidades o fallos. Cuando se identifica un problema, se pueden tomar medidas correctivas como reiniciar el servicio o implementar acciones para mejorar el rendimiento y prevenir caídas, todo basándose en la información que proporcionan estos *health check*s.

El sistema implementa un manejo robusto de errores que previene que fallos individuales comprometan el funcionamiento completo. Cuando ocurre un fallo, en lugar de interrumpir todo el flujo de ejecución, el sistema responde con mensajes apropiados, registra

el incidente para su posterior análisis, y continúa operando con normalidad en las funcionalidades no afectadas.

Finalmente, las integraciones con servicios externos, como el envío de correos electrónicos o el almacenamiento en la nube, se desacoplan de la aplicación principal mediante *circuit breakers* y procesamiento asíncrono. La comunicación con estos servicios se realiza a través de colas de mensajes, lo que evita que demoras o caídas temporales en servicios externos afecten la disponibilidad general de la plataforma.

11.1.3. Rendimiento

Con el objetivo de verificar el cumplimiento de los umbrales definidos anteriormente en la sección del requerimiento no funcional de rendimiento se ejecutó un banco de planes de carga construidos con *K6* contra el entorno desplegado en *AWS*. Se realizaron 14 planes de carga agrupados en *backend* core, servicio de *matching* y *frontend*, aplicando tres escenarios de tráfico distintos; *maintained* (carga normal), *spike* (picos) y *stress* (límites de resistencia). (El análisis detallado de las pruebas de carga y la batería de *tests* funcionales y de integración se incluye en el anexo 16.20. Análisis resultados pruebas de carga del sistema.)

Durante las pruebas de carga el *cache* fue deshabilitado intencionadamente para medir el peor escenario. Tener en cuenta que de lo contrario como ya se mencionó previamente el sistema implementa *cache* a nivel del cliente evitando sobrecargar el sistema con consultas repetitivas y comunes, especialmente útil ante consulta de elementos no comúnmente variables como catálogos.

Los *endpoints* de lectura (*GET*) correspondientes a listados de candidatos, vacantes y *dashboards*, mantuvieron latencias p95 de entre 210 milisegundos y 245 milisegundos en todos los escenarios, con 0 % de errores y un *throughput* que escaló hasta 265 solicitudes por segundo bajo 40 usuarios simultáneos. Esto demuestra que la arquitectura actual satisface holgadamente los objetivos de rendimiento para la navegación habitual de usuarios y empresas.

Las operaciones de escritura no intensivas cumplieron los umbrales en carga normal con p95 aproximadamente entre 380 y 650 milisegundos y aceptaron hasta 100 solicitudes por segundo sin errores. En *stress* se observaron valores p95 cercanos a 2,8 segundos, ligeramente por encima del límite; sin embargo, la tasa de error se mantuvo < 1 % y la

experiencia de usuario no se vio comprometida, dado que estos flujos se completan en *background* y no bloquean la interfaz.

El cálculo sincrónico de *matching* candidato-vacante arrojó p95 alrededor de 2,8 segundos y saturó el servicio ante picos de concurrencia. La generación sincrónica de *embeddings* mostró tiempos p95 entre 27 y 60 segundos con tasas de error del 9-20 %. Estos resultados refuerzan nuestra decisión de tener el procesamiento de *matching* desacoplado de la aplicación principal; en la práctica, el sistema encola la tarea en *RabbitMQ* y responde al usuario en menos de 500 milisegundos, cumpliendo el objetivo de latencia.

En conjunto, el sistema demuestra que soporta la carga prevista para el proyecto (hasta 40 usuarios simultáneos y más de 250 solicitudes por segundo en vistas críticas) sin degradación funcional. Los únicos escenarios que superan los umbrales de latencia son procesos ya diseñados para ejecutarse de forma asíncrona, por lo que el requisito no funcional de rendimiento se ve validado a nivel funcional por las pruebas.

Como se mencionó anteriormente las operaciones mediante servicios externos se realizan de manera asíncrona al igual que los procesos de *matching*. El usuario recibe una respuesta inmediata donde se indica que se encolo correctamente el proceso y luego el mismo es procesado por el servicio correspondiente, evitando problemas de altas demoras y bloqueo del sistema por procesamientos costosos externos a la aplicación principal.

La aplicación, servicio de *matching*, base de datos, entre otros componentes, viven en su propio contenedor. Esto nos permite poder escalar independientemente cada contenedor según demanda y poder darle el mejor rendimiento a la plataforma de la manera más sencilla y eficiente.

Como se mencionó anteriormente, cuando se habló sobre la base de datos relacional de la aplicación, se crearon índices sobre búsquedas más frecuentes y costosas para mejorar el rendimiento de las consultas hacia base de datos. Esto se ve reflejado en los resultados de las pruebas de carga donde se mostró un rendimiento satisfactorio.

11.1.4. Observabilidad

Para garantizar la trazabilidad y el diagnóstico rápido sobre el sistema se incorpora un *stack* completo de observabilidad centrado en *Sentry* [32] y *Amazon CloudWatch*, que permite visualizar, alertar y auditar el comportamiento de la plataforma en tiempo real.

Todos los servicios emiten *logs* estructurados que son enviados automáticamente a *Sentry*, donde se centralizan los errores, excepciones y trazas de ejecución. Esto incluye una descripción, usuario logueado, ubicación de quien emite el *log*, elementos relevantes, error y *status code*, lo que permite seguir una solicitud de extremo a extremo sin perder visibilidad. Además, se ha configurado un *dashboard* en *Sentry* que muestra de forma clara y agrupada los errores por frecuencia e impacto, facilitando la identificación de patrones y la priorización de correcciones.

Las métricas clave de rendimiento, latencia (p50, p95, p99), tasa de error, *throughput* y concurrencia, se capturan desde el código y también desde *Amazon CloudWatch*, donde se almacenan históricamente. Estas métricas se visualizan en paneles configurables que permiten comparar tendencias entre versiones y detectar degradaciones progresivas. Cuando alguna métrica supera los umbrales definidos (por ejemplo, p95 mayor a 1,5 segundos, tasa de error mayor a 1% o caída repentina de *throughput*), se activan alertas automáticas que envían notificaciones por correo electrónico a través de *Sentry*. Estas alertas incluyen el contexto del error, el servicio afectado y un enlace directo al evento para facilitar su análisis.

11.1.5. Portabilidad

Todo el código se encuentra en un repositorio dentro de *GitHub*, así poder mantener un versionado e integrar cambios de manera controlada. Ese mismo código se empaqueta en una imagen *Docker* que puede ejecutarse igual en cualquier lado: desarrollo, *testing* o producción. De esa forma, el entorno se replica con un simple *docker run* y sin haber diferencias entre máquinas.

Como ya se mencionó en el capítulo de infraestructura, esta se encuentra como código mediante *Terraform*. El despliegue se promueve entre entornos aplicando plan/*apply*, eliminando la configuración manual y garantizando que si funciona en un ambiente debe funcionar en los otros. Pudiendo ser sencillo el despliegue en cualquier entorno.

Toda la información específica y dinámica de cada instalación se externaliza en variables de entorno que nunca viajan dentro del contenedor ni del código fuente. Pudiendo poder cambiar ciertas configuraciones o elementos sin tener que hacer un nuevo despliegue. Con esto podemos reproducir nuestro sistema en cualquier lado y personalizarlo sin problemas mediante las variables de entorno.

11.1.6. Entregabilidad

Para garantizar entregas libres de errores se decidió usar *Git Flow* dentro de *Github*. Se estableció que se trabajaría sobre la rama *develop* la cual sería nuestra rama de desarrollo y que únicamente se harían integraciones a la rama *main* en el momento de realizar un despliegue, automatizando el mismo cuando se integraban cambios en la rama.

Dentro de *develop* se estableció un sistema de ramas, para poder realizar cambios primero se debe crear una nueva rama con el número de incidencia (correspondiente a la tarjeta en la cual se estaba trabajando del tablero *Kanban*) y un título descriptivo (Ejemplo: WZY-<Nro>-<Titulo>). Al finalizar el trabajo para poder impactar estos cambios en la rama *develop* se tiene que crear un *Pull Request* (PR) de la rama subyacente hacia *develop*. Al crear el *PR* se ve como se corren validaciones por detrás, se evalúa que se pueda realizar el *build* sin errores, no esté fallando ningún *test* unitario y no falle el *linting*, el cual tiene ciertas reglas sobre la calidad del código que se deben cumplir. Por último, el *PR* tiene que ser aceptado por al menos otro desarrollador que haya testeado los cambios realizados y que vea que se haya corrido el pipeline correctamente. Con todo esto realizado se puede aceptar el *PR* e integrar el código de la nueva funcionalidad o arreglo a *develop*.

11.2. Cumplimiento de atributos de calidad

11.2.1. Escalabilidad

El sistema está compuesto por distintos componentes (servicios, bases de datos, colas de mensajes, etc.), los cuales están separados con la posibilidad de replicarse horizontalmente sin afectar al resto. Esta independencia permite que un aumento de carga en cualquier parte del sistema se resuelva levantando más réplicas del componente afectado, sin rediseñar flujos ni modificar código.

Por otro lado, como ya se mencionó anteriormente aquellos procesos más costosos o que dependen de servicios externos son ejecutados de manera asíncrona. Con esto evitamos que un cuello de botella externo o un pico de cómputo se propague hasta el usuario. Este modelo permite encolar miles de tareas sin bloquear la *API*: si el servicio de notificaciones tarda minutos o el motor de *matching* requiere *CPU* intensiva, la interfaz sigue siendo rápida y el usuario nunca percibe demora.

Además, todo el tráfico externo ingresa por un único punto de acceso: el balanceador de carga administrado por *AWS*. Desde ahí se distribuyen las peticiones entre las instancias sanas y se realiza el *health check* continuo; escalar es tan sencillo como aumentar la capacidad del *auto-scaling group* o cambiar el número deseado de réplicas en la configuración, sin tocar código.

11.2.2. Mantenibilidad

El código está organizado en capas bien diferenciadas: lógica de negocio, controladores, servicios y modelos que coexisten en carpetas separadas y nunca se cruzan responsabilidades. Esta división permite abrir un archivo y saber de inmediato qué hace, dónde impacta y cómo extenderlo sin afectar otras partes del sistema. Además, se siguen prácticas de desarrollo de *Next.js* cómo agrupar el código fuente de las páginas dentro de la carpeta pages, los controladores en la carpeta *API*, los componentes dentro de la carpeta *components*. Dichas prácticas facilitan a los desarrolladores que conocen este tipo de estructura.

El proyecto incorpora herramientas automatizadas que mantienen la calidad del código sin intervención manual constante. *ESLint* asegura consistencia en estilos y detección de problemas potenciales, *Prettier* automatiza la uniformidad del código, y la utilización de sistemas de *CI/CD* previene la propagación de código de baja calidad. Estas herramientas reducen significativamente el esfuerzo de *code review* y mantienen estándares elevados de manera sostenible.

El uso de *TypeScript* a lo largo de toda la aplicación proporciona beneficios sustanciales para la mantenibilidad. La generación automática de tipos desde esquemas de *Prisma* elimina discrepancias entre el modelo de datos y el código de aplicación, mientras que las interfaces bien definidas en archivos como *candidateTypes.ts*, *companyTypes.ts* y

jobTypes.ts crean contratos claros entre diferentes partes del sistema. Esta tipificación permite refactorización segura con detección automática de *breaking changes* en tiempo de compilación.

Complementando estas herramientas de análisis estático, se utiliza *SonarQube* para realizar auditorías periódicas del código fuente. Esta plataforma de análisis de calidad de código identifica code smells, vulnerabilidades de seguridad, duplicación de código, y desviaciones de mejores prácticas de desarrollo. Si bien no se ejecuta en cada *commit* como parte del *pipeline* de *CI/CD*, se realizan análisis periódicos que permiten identificar áreas del código que requieren refactorización o mejora, manteniendo la deuda técnica bajo control y asegurando que el proyecto evolucione manteniendo estándares de calidad elevados a largo plazo.

11.2.3. Observabilidad

La observabilidad no solo cumple el requisito de emitir logs y métricas, sino que se ha convertido en un atributo intrínseco que determina qué tan rápido evoluciona el producto. Gracias a la trazabilidad completa, el equipo puede refactorizar con confianza ante cualquier cambio que degrade latencia o incremente errores detectandose en minutos, reduciendo el tiempo medio de resolución.

A su vez, el *dashboard* unificado (*Sentry* + *CloudWatch*) nos permite transparencia pura sobre las funcionalidades del sistema, pudiendo cualquier desarrollador entender el flujo del mismo sin tener que entrar al código. Esta documentación viva acelera el *onboarding* y disminuye la dependencia de conocimiento tribal.

Además, se permite tener completa visibilidad del funcionamiento de la plataforma utilizando las distintas métricas que nos ofrece el *dashboard*, pudiendo tomar decisiones importantes en base a estas.

11.2.4. Portabilidad

La portabilidad se diseñó como una característica esencial, donde el sistema mantiene su comportamiento y rendimiento en cualquier infraestructura gracias a contenedores, variables externalizadas e infraestructura como código, permitiendo pasar sin fricción entre desarrollo, prueba, producción o distintos proveedores de nube.

El sistema no depende de un entorno particular ni de una infraestructura específica, lo que le otorga independencia tecnológica y reduce la fricción al incorporar nuevas herramientas o proveedores de nube. Asimismo, la capacidad de replicar los entornos con exactitud favorece la trazabilidad de errores y la experimentación controlada, fortaleciendo la robustez del proceso de desarrollo. Esto se materializa en la práctica con *Docker* y *Terraform*, la misma imagen que se levanta localmente con *docker compose up* es la que despliega el *pipeline* en *AWS*, *Azure* o una laptop sin conexión, cambiando únicamente las variables de entorno.

11.2.5. Usabilidad

La definición temprana de los flujos en *Figma* y la participación activa de *UX* previo al proceso de desarrollo permitieron traducir los requerimientos funcionales en experiencias coherentes y centradas en el usuario. Cada componente visual y cada interacción fueron basadas en prototipos previamente validados y con una investigación exhaustiva detrás de ellas.

La interfaz se construyó sobre *Next.js + Shad CDN*, aprovechando componentes accesibles y temas consistentes que facilitan el cumplimiento de estándares de navegación. Toda acción dentro de la plataforma recibe *feedback* visual inmediato: estados de carga, confirmaciones de éxito y mensajes de error que indican cómo resolver el problema. Implementamos *skeletons* y estados optimistas para mantener la sensación de rapidez incluso durante operaciones lentas. La arquitectura de información incluye *breadcrumbs* claros y navegación lateral que reduce la profundidad de clics; filtros persistentes con contadores permiten refinar búsquedas sin perder contexto.

Se realizaron pruebas de usabilidad, las cuales se van a profundizar más adelante en el documento, con el fin de continuar evaluando este atributo de calidad y para poder perfeccionar el mismo. En estas sesiones se llegaron a ver debilidades y fortalezas de la plataforma con respecto a la usabilidad, lo cual es de gran ayuda para mejorar este atributo.

11.3. Estrategia de *testing* como evidencia de calidad

Para mantener la calidad de las funcionalidades, cada lógica de negocio cuenta con su propio conjunto de pruebas unitarias. Se decidió escribir tests unitarios únicamente esta capa, porque es la que aplica las reglas complejas del dominio y donde cualquier cambio puede

afectar el comportamiento esperado. Cada módulo de lógica de negocio tiene un archivo de test que cubre todos los casos de uso: éxitos, errores y límites. Para poder probar esta capa de manera independiente se utilizaron *mocks* que simulan respuestas de bases de datos, colas de mensajes y cualquier otra llamada externa a la capa lógica.

Para ejecutar los *tests* utilizamos *Jest*, el cual no solo nos permite correr la *suite* completa, sino que, mediante el *flag* de *coverage*, genera un informe detallado con el porcentaje de líneas, ramas y funciones cubiertas en cada archivo probado. Establecimos como mínimo un 85 % de cobertura de líneas de código para garantizar que la mayoría de nuestras lógicas de negocio funcionen de la manera esperada. Estos *tests* se integran en nuestro *pipeline* de *CI*: cada vez que se abre un *PR* se ejecutan automáticamente; si alguno falla, el *pipeline* se interrumpe y el cambio no puede ser integrado hasta ser corregido.

Los *tests* fueron realizados mediante el patrón *Arrange-Act-Assert* (AAA) para garantizar claridad y mantenibilidad. En la fase *Arrange* preparamos el entorno: instanciamos y configuramos *mocks*, inicializamos variables y definimos los datos de entrada. En *Act* ejecutamos la función específica bajo prueba. Finalmente, en *Assert* verificamos que el resultado obtenido coincide con el esperado. Esta estructura homogénea facilita la lectura de los *tests* y permite detectar rápidamente qué parte del comportamiento falla cuando un *test* no pasa.

Si bien estos *tests* aseguran en gran parte un correcto funcionamiento a nivel de lógica de negocio, no capturan siempre el comportamiento completo de la solución. Por ello, además de superar los *tests* unitarios, todo *Pull Request* debe ser aprobado por otro desarrollador, quien realiza pruebas manuales en el *frontend* para validar que la funcionalidad cumpla con los flujos esperados punto a punto. De esta forma combinamos validación automatizada con revisión humana, asegurando tanto la calidad del código como la experiencia real del usuario.

12. Resultados y validación de la solución

12.1. Pruebas con usuarios

Para validar nuestra solución se decidieron dos prácticas con distintos enfoques. En primer lugar se decidió hacer una segunda entrevista con la recruiter de GoGrow, en la cual se buscaba poder hacerle una demostración guiada del funcionamiento de la plataforma y obtener *feedback* sobre el flujo de la solución y que aspectos se podrían mejorar. Consideramos muy valiosa la opinión de la recruiter, ya que ella es la que lleva los procesos de selección de personal de la empresa, siendo un usuario tipo de empresa, y además entiende las necesidades de los usuarios candidatos. Por otro lado, se buscaba entender cómo se adaptan usuarios candidatos a la plataforma, ver qué dificultades tenían al utilizarla y obtener opiniones tras su uso, para esto hicimos pruebas funcionales con usuarios de tipo candidatos, donde se les daba libertad para registrarse y utilizar la plataforma.

12.1.1. Entrevista de validación

Realizamos una segunda entrevista a Martina Vega, *Recruiter* de GoGrow (Las notas y conclusiones de la validación con recruiter pueden consultarse en el anexo 16.21. Validación solución *Recruiter*), con experiencia en selección de perfiles IT que actualmente utiliza *TeamTailor* y *LinkedIn Recruiter* en formato *online*, la entrevista tuvo una duración aproximada de 45 minutos brindando distintos aspectos importantes sobre nuestra solución.

En primer lugar, Martina indicó que ver todos los candidatos anteriores en la misma pantalla y que el sistema indique si alguno encaja con la nueva vacante le ahorraría al menos dos horas por proceso. Además, en los procesos de alto volumen (> 200 postulantes), la función de *auto-ranking* le parece muy favorable permitiendo "saltar" directamente al *top* 20 sin leer CVs uno por uno, mientras que en perfiles *senior* prefiere desactivar el filtro automático y usar el *score* solo como "una guía" que puede ser auditada para poder tener cierta referencia pero no sesgar su decisión.

Respecto de la explicabilidad del *matching*, expresó que saber cómo se ponderan las palabras clave, años de experiencia y disponibilidad le daría convicción de que el candidato encabeza la lista por motivos claros, función que se materializa en la tarjeta de trazabilidad desplegable en cada tarjeta de candidato.

Luego, hizo bastante énfasis en poder tener una funcionalidad de carga automática de CV porque reduce la fricción: el candidato sube el PDF y obtiene un perfil pre-completado que puede editar inmediatamente; además, sugirió que el campo "*soft skill*s" sea opcional o convertirse en selector de adjetivos pre-definidos para no intimidar a perfiles *junior*.

Sobre las comunicaciones automatizadas, destacó la utilidad de recibir y poder editar plantillas de emails de estado (por ejemplo "seguís en proceso") para adaptar el tono a la cultura de la empresa, función que incluye el editor de templates con variables dinámicas y programación de recordatorios semanales.

El *feedback* recibido de la recruiter sobre la solución actual establece que ya resuelve puntos críticos del día a día: calificó la creación de vacantes como "clara e intuitiva", destacó que el envío automático de mails al cambiar el estado del candidato "aligera el trabajo" y propuso ir más lejos personalizando las plantillas para adaptar el tono a la cultura de la empresa. Valoró, además, la posibilidad de programar recordatorios periódicos y aseguró que, en procesos masivos, el *ranking* inicial que ofrece la plataforma le ahorra "al menos un par de horas" de revisión manual, confirmando que las funcionalidades actuales ya aportan valor real antes de incorporar nuevas iteraciones.

En resumen, la entrevista confirma que los componentes actuales de la plataforma (creación ágil de vacantes, *matching* explicable, carga automática de CV y comunicación personalizada) resuelven dolores inmediatos del recruiter y generan ahorro de tiempo significativos. Estos resultados refuerzan la línea base sobre la cual iterar nuevas funcionalidades y constituyen evidencia temprana de adopción favorable por parte del usuario empresa.

12.1.2. Pruebas funcionales con usuarios candidatos

Se invitó a tres personas de 23 años en búsqueda activa de empleo a interactuar con la aplicación sin asistencia previa, partiendo desde el *login* y finalizando en la visualización de su pipeline de vacantes (Las pruebas, notas originales y conclusiones de las pruebas con candidatos se incluyen en el anexo 16.22. Validación solución con usuarios candidatos). El objetivo fue observar comportamiento natural, detectar fricciones de usabilidad y validar la propuesta de valor en un estado cercano a la finalización del producto.

Para contextualizar la prueba, a cada candidato se le presentó la misma situación hipotética: un conocido les había recomendado la plataforma y se les compartió el enlace de acceso. Se les pidió entrar, registrarse, completar el perfil de candidato y explorar libremente el sistema. Se les indicó que intentarán recorrer la mayor cantidad de funcionalidades posible.

Una vez finalizada la exploración autónoma, el evaluador, asumiendo el rol de empresa, iniciaba un proceso de reclutamiento ficticio ajustado al perfil del candidato. Causando un evento de *matching*, se movía al candidato por las etapas de la vacante y se le invitaba a revisar sus notificaciones, la ficha de la vacante y su nuevo estado dentro del proceso. Este esquema fue idéntico para los tres participantes, variando únicamente la vacante a la generada para que coincidiera con su formación.

Se estableció como regla intervenir solo si un bloqueo técnico impedía continuar; esto ocurrió una única vez y no alteró la validez de la observación.

Durante los recorridos se observó que la arquitectura general y la navegación eran comprendidas de forma inmediata: todos los participantes localizaron sin ayuda el *dashboard*, el listado de vacantes y el apartado de perfil, y calificaron la interfaz como “bonita y clara”. La idea de un *scouting* automatizado basado en perfil, en lugar del clásico “primero en postularse”, fue vista como “justa” y “diferente a *LinkedIn*”.

No obstante, surgieron tres bloques recurrentes de fricción. El primero fue el *setup* inicial: la cantidad de campos, la obligatoriedad del día exacto en fechas de educación y la opacidad de términos como “*Field of Study*”, “*Certifications*” o ciertas *soft skills* generaron dudas. Dos de los tres usuarios intentaron elegir más de un “objetivo principal” y pidieron autollenado mediante CV o, al menos, ejemplos dentro de cada campo para acelerar la carga.

El segundo bloque coincidió en la falta de *feedback* visual durante tiempos de espera: clics repetidos en botones de login o filtros, y la sensación de que "no pasa nada" hasta que la pantalla se actualiza. Esto se agravó con tooltips que demoran en aparecer y que varios confundieron con botones accionables, lo que aumentó la percepción de inestabilidad.

El tercer aspecto de mejora se trató sobre la claridad de conceptos y estados. Las diferencias entre "*Save Job*", "*Show Interest*" y las secciones "*Opportunities / Saved / In-Process*" no fueron evidentes; todos pidieron tooltips inmediatos o microtextos explicativos. Además, reclamaron visible la compensación en cada vacante, estableciendo que "me molesta no saber cuánto pagan antes de aplicar", y mayor control sobre las notificaciones: les gusta recibir actualizaciones, pero el volumen actual las hace percibir como *spam*; prefieren un centro de notificaciones *in-app* y la posibilidad de desactivar correos por categoría.

A pesar de estos puntos, las funcionalidades principales de la plataforma recibieron evaluación positiva: el porcentaje de match explicable, la trazabilidad de estados ("rechazado", "en revisión", "*top* 5") y la posibilidad de ver estadísticas propias fueron destacadas como "transparencia que otras plataformas no dan". Ningún usuario identificó funcionalidades críticas faltantes ni cuestionó la lógica del *matching*; sus pedidos apuntaron a pulir tiempos de carga, simplificar el formulario inicial y ajustar la redacción de ciertos textos; es decir, mejoras de *UX/UI* antes que cambios de modelo.

12.2. Aprendizajes de las pruebas

Una vez reunida la evidencia de campo (entrevistas, pruebas funcionales y encuestas) se detectan ciertos hallazgos en lugar de simples observaciones. El objetivo es convertir la retroalimentación cruda en lineamientos concretos para la siguiente iteración sin perder de vista la estrategia general.

Aprendizajes clave

1. **Transparencia > Precisión**: poder tener la posibilidad de ver los detalles del *matching* genera mayor confianza que simplemente dar el valor del mismo. Por lo que se trata de dar mayor contexto antes que dar un número aunque este sea más acertado.
2. **Autollenado reduce abandono**: los usuarios suelen abrumarse ante procesos largos como puede ser nuestro *setup* inicial y para evitar que puedan abandonar el proceso en

mitad se podría tener un autocompletado de ciertos campos mediante CVs o perfiles de *LinkedIn*.

3. **Control de notificaciones es parte del valor**: la sensación de *spam* aparece a los 3-4 correos electrónicos; siendo inmediata la necesidad de tener alternativas como *WhatsApp*, notificaciones *in-app* u otra alternativa donde los usuarios puedan sentirse más cómodos con las notificaciones y poder recibir estas con mayor frecuencia.
4. **Tiempo de carga es funcionalidad**: por debajo de 2 segundos se percibe como "rápido"; por encima de 4 segundos se generan clics múltiples y errores de estado. Siendo importante controlar los estados de carga y rendimiento en la plataforma.

Estos hallazgos fueron encontrados por el equipo y plasmados y priorizados en el *backlog*.

12.3. Limitaciones de la solución

A lo largo del desarrollo del proyecto fueron emergiendo distintos techos que definen hoy el perímetro natural del producto. No se trata de funcionalidades ausentes por descuido, sino de restricciones que conviven con la versión actual.

Por un lado, una de las limitaciones más claras que presenta el sistema es su dependencia exclusiva del idioma inglés: *embeddings*, interfaz y textos internos fueron concebidos en ese idioma porque los modelos de similitud semántica que se emplearon, como ya se mencionó, fueron entrenados en inglés y no son multilingües; en consecuencia, cualquier usuario que no maneje inglés deberá traducir o adaptarse para interactuar con la plataforma.

Además, la gestión de entrevistas queda totalmente ajena al producto: aunque la aplicación permite crearlas, asignarlas, cambiar su estado, dejar reseñas y notificar fecha, hora y enlace por correo, no puede verificar si esos horarios coinciden con la agenda de ninguna de las partes; así, el candidato no puede proponer alternativas ni la empresa validar disponibilidad sin salir a un canal externo.

A su vez, la comunicación entre empresa y candidato transcurre fuera del sistema; solo existen dos válvulas, el *feedback* automático que se genera cuando cambia el estado de una postulación o cuando se alcanza un alto nivel de *matching*, y los mails o *WhatsApp* que

las empresas pueden enviar mediante plantillas predefinidas, sin un espacio compartido donde conversar sin abandonar la plataforma.

De igual modo, el *onboarding* de candidatos exige una inversión de tiempo concentrada: cargar estudios, experiencia, habilidades, expectativas y preferencias implica varios pasos que, durante las pruebas con usuarios, se percibieron como tediosos, ya que el sistema no ofrece atajos ni autollenado que acorten ese trayecto inicial.

Finalmente, Whizdy habita únicamente dentro del navegador de escritorio; la interfaz no fue diseñada con criterios para una versión móvil, de modo que usarlo desde un celular implica zoom, scroll horizontal y una experiencia que no fue pensada para pantallas táctiles.

13. Conclusiones y Trabajo Futuro

13.1. Aprendizajes y desafíos

En todo proyecto, por más planificado que esté, o se piense que se está preparado, surgen imprevistos que desafían la planificación original. Whizdy no fue la excepción. A lo largo de su desarrollo, el equipo se enfrentó a una serie de obstáculos que, si bien en su momento representaron retrasos y tensiones, terminaron siendo catalizadores de mejoras profundas en la forma de trabajar. Esta sección busca identificar estos desafíos encontrados, cómo fueron abordados, qué decisiones se tomaron y qué se aprendió en el proceso.

13.1.1. *Scope creep*

Si bien ya se ha mencionado en profundidad este caso, no parece conveniente la omisión del mismo como uno de los principales desafíos encontrados en el proyecto. Aunque no se profundizará nuevamente en el descubrimiento del mismo, si decir que fue uno de los primeros y más complejos desafíos del proyecto. Lo que comenzó como un diseño de 15 pantallas en *Figma*, terminó transformándose en un producto visual de casi 100 páginas, con nuevos flujos, estados y variaciones que no estaban contemplados en la planificación inicial. Este crecimiento no fue arbitrario: reflejaba una evolución natural del cliente al ver su idea tomar forma. Sin embargo, su impacto fue real: el *backlog* creció más rápido que la capacidad de cierre del equipo, y la sensación de "nunca terminar" comenzó a instalarse.

La respuesta no fue simplemente "trabajar más", sino repensar qué trabajar. Se implementó sesiones de *backlog grooming*, donde se reescribieron historias, se dividieron *tickets* demasiado amplios, se definieron criterios de aceptación claros y se priorizó con criterio de valor. Además, se presentó formalmente al cliente un informe de repriorización, donde se proponía despriorizar funcionalidades no críticas sin afectar el valor del producto. Esta decisión, que podría haber sido tomada como una "reducción", fue entendida como una estrategia de enfoque: construir algo cerrado, útil y validable, antes que algo extenso pero inestable.

El resultado fue un *backlog* estable, una velocidad predecible y un equipo que recuperó la confianza en su capacidad de entrega. Más importante aún: se consolidó una relación de confianza con el cliente, basada en transparencia y co-construcción.

13.1.2. Estimación inestable

Al inicio, estimar era un ejercicio de azar. Los *tickets* eran amplios, dependientes entre sí, y en muchos casos ocultaban complejidad técnica o de diseño por detrás. Se sumaba el hecho de que el equipo estaba aprendiendo *Next.js* sobre la marcha, lo que volvía aún más incierto cualquier cálculo. Las primeras iteraciones terminaron con story points incompletos, tareas arrastradas y una sensación de descontrol.

La mejora no llegó con una herramienta nueva, sino con un proceso más rígido en la entrada y más flexible en la salida. Partiendo de una *Definition of Ready* clara: ningún ticket entraba a desarrollo sin criterios de aceptación, sin dependencias identificadas y sin una estimación consensuada en *planning poker*, sumado a proceso de *trackeo* de horas del equipo, lo que permitió, iteración tras iteración, calibrar la capacidad real del equipo.

Con el tiempo, la velocidad se estabilizó. No porque necesariamente se trabajara más, sino porque se trabajaba mejor: *tickets* más pequeños, independientes y con definiciones más claras. La variación de esfuerzo por iteración bajó, y la confianza del equipo en sus propias estimaciones se volvió cuantificable.

13.1.3. Brecha técnica

Next.js, *embeddings* y bases vectoriales eran nuevos para los tres integrantes del equipo. La investigación previa y la prueba de concepto habían servido como punto de partida, pero al llegar al código real la incertidumbre continuaba. Los primeros cambios fueron un ejercicio constante de prueba y error, donde cada *pull request* se convertía en una instancia de aprendizaje compartido.

El avance técnico no se sostuvo por una curva de aprendizaje individual, sino por un proceso colectivo: investigar, fallar, consultar y compartir. Cuando alguno encontraba una solución o un obstáculo, lo documentaba y lo transmitía al resto. Esta dinámica permitió que el conocimiento se distribuyera rápidamente y que las dudas no se estancaran. Los links, ejemplos y aprendizajes se volvían insumos comunes en cada iteración.

Con el tiempo, las herramientas dejaron de ser un freno y pasaron a ser parte del flujo natural de trabajo. La incorporación de utilidades como *ESLint* y la estandarización de buenas prácticas ayudaron a mantener la coherencia del código mientras la productividad aumentaba. Mediante el avance, la brecha técnica se fue cerrando: no porque el equipo dominara todas las tecnologías, sino porque aprendió a avanzar en medio de la incertidumbre, sosteniendo el ritmo sin perder calidad.

Iteración tras iteración la incertidumbre se aplanó: bajaron los comentarios en cada PR, de *spikes* semanales pasamos a cambios directos. La brecha no se cerró de golpe; se fue llenando de puentes que construimos mientras corríamos, hasta que cada integrante sintió que el código que tocaba ya no era territorio desconocido.

13.1.4. Concentración de conocimiento

En equipos pequeños es habitual que el conocimiento se concentre en quien más investiga, dedica más horas o simplemente se siente más cómodo con ciertas partes del producto. Esa asimetría puede parecer inofensiva, pero esconde un riesgo: si una persona se ausenta, parte del proyecto se detiene. Desde el inicio quisimos evitarlo. Nuestro objetivo era que todos los integrantes comprendieran el sistema en su totalidad y pudieran contribuir en cualquier módulo sin depender de un único referente.

La estrategia fue simple, pero sostenida. Cada investigación, prueba o decisión técnica debía compartirse con el resto del equipo, ya fuera en reuniones, a través de documentación interna o en discusiones sobre los pull requests. Cuando alguien profundizaba en un tema, por ejemplo, un nuevo enfoque de test o una integración compleja, lo explicaba al resto y se registraba.

Con el tiempo, esta práctica se volvió parte natural del flujo de trabajo. El conocimiento dejó de ser individual para transformarse en patrimonio colectivo. Al llegar el momento de la documentación final, cualquier integrante del equipo podía explicar con claridad las decisiones tomadas y los componentes del sistema.

13.1.5. Restricciones del *Free Tier*

Un desafío que se sumó de manera transversal vino de la mano de uno de los objetivos del equipo mantener todo el desarrollo y despliegue de forma gratuita, operando dentro dentro de los límites del free tier de *AWS* y de servicios auxiliares como *Sentry*. Lo que en un principio parecía una simple restricción de costos se convirtió rápidamente en un factor de diseño donde debía de hacerse antes la pregunta "¿entra en la franja gratuita?".

Los modelos de *embeddings* más potentes, por ejemplo, superaban el tope de memoria de las instancias gratuitas; eso llevó a buscar alternativas, otros modelos de *embbeding* y como ya se mencionó la adopción de *Hetzner* para el servicio de *matching*, donde el costo por *CPU* y *RAM* es una fracción del de *AWS*.

El resultado fue un despliegue funcional, pero también una lección práctica: la restricción presupuestaria no solo limita recursos, define arquitecturas; aprendimos a negociar con los techos que no podíamos romper y a convertirlos en criterio de decisión más, sin sacrificar el valor entregado.

13.1.6. Conclusión general

Los desafíos no fueron errores del camino: fueron el camino. Cada problema, ya fuera un *backlog* descontrolado, una estimación fallida o una tecnología desconocida, obligó al equipo a construir procesos más sólidos, a comunicarse mejor y a aprender más rápido. Lo que podría haber sido un fracaso iterativo se convirtió en una cultura de mejora continua.

Del *scope creep*, el equipo aprendió a poner límites sin perder ambición: a priorizar con criterio de valor y a construir sobre bases firmes antes de expandir el alcance. De las estimaciones inestables, surgió el entendimiento de que predecir no es acertar, sino medir mejor para decidir mejor, logrando una velocidad sostenida y predecible. La brecha técnica enseñó que no hace falta dominarlo todo para avanzar, sino aprender en conjunto, compartir los errores y transformarlos en conocimiento colectivo. Y frente al riesgo de concentración de conocimiento, se aprendió a documentar, explicar y rotar tareas, garantizando que el producto no dependiera de una sola persona, sino de la cohesión del equipo.

El resultado no es solo un producto funcional, sino un equipo que sabe cómo adaptarse, cómo priorizar, cómo estimar y cómo entregar valor incluso bajo incertidumbre.

Lo que comenzó como un proyecto técnico terminó siendo una lección de colaboración, resiliencia y madurez profesional. Y eso, más que cualquier funcionalidad, es lo que hace a este proyecto un éxito completo.

13.2. Análisis de decisiones tomadas

En este capítulo se busca llevar a cabo una breve reflexión sobre las decisiones principales que el equipo tomó durante el transcurso del proyecto. Sin entrar en detalle, se quiere remarcar algunos elementos ya mencionados en este documento, haciendo alusión al por qué se consideraron importantes para el proyecto y cómo estos influyeron en el resultado final directamente.

13.2.1. División del proyecto en fases

Una de las primeras decisiones tomadas por el equipo fue la estructura principal y el rumbo que seguiría el proyecto. Como ya se mencionó, se separó el proyecto en tres fases claras: Preparación e investigación, Desarrollo y Documentación.

Cada una tuvo un objetivo definido, y consideramos que esta fue una decisión muy acertada, ya que cada fase contribuyó de forma precisa a diferentes necesidades del proyecto.

En primer lugar, la fase inicial permitió al equipo entender el contexto y el problema, así como los dolores del cliente y, por supuesto, su solución. Esta etapa fue la base de la metodología y de las formas de trabajo, además de que el proyecto en sí surgió a partir de las decisiones tomadas en este momento: tecnologías, herramientas e implementación. Esto definió no solo los pasos a seguir, sino también el alcance del proyecto, dando una base sólida para la fase de desarrollo.

También es importante mencionar que esta fase fue clave para la elección de las tecnologías y formas de implementación, como el uso de *embeddings*, cuya investigación permitió al equipo aprender sobre su utilidad para el sistema, algo que por supuesto no podía faltar.

Dando lugar así a la fase de Desarrollo, como ya se mencionó, esta fue la fase principal y comenzó el 20 de enero. Como se puede intuir fácilmente fue la fase en donde se dio el desarrollo del sistema, cada iteración, cada funcionalidad, cada elementos del sistema y

cada corrección se llevó a cabo en esta fase. Viendo en retrospectiva, consideramos que tal vez se podría haber iniciado un poco antes, dándole al equipo un poco más de espacio para el desarrollo del sistema. Sin embargo, esto es algo que, como ya se dejó en claro, no afectó lo logrado por el equipo.

Finalmente, la fase de documentación, que dio lugar a este documento en concreto, permitió al equipo contar con tiempo suficiente para elaborar con detalle y completitud un documento capaz de describir, con el mayor detalle posible, los elementos relacionados al proyecto tanto a nivel de producto como académico.

En conclusión, creemos que la separación de estas fases, así como sus duraciones, tuvieron un fuerte y positivo impacto en el desempeño y finalización del proyecto.

13.2.2. Forma de trabajo

Otra decisión que el equipo considera muy importante fue la forma de trabajo adoptada. Sin repetir lo ya expuesto, basta con recordar que el proyecto se sostuvo sobre una trama de prácticas que se reforzaron mutuamente: relevamiento profundo de requerimientos, gestión activa de los cambios, investigación previa a cada decisión, documentación en paralelo al desarrollo, toma de decisiones colectiva sustentada en datos, ajustes continuos de la propia metodología y comunicación fluida con el cliente, entre otros. Este conjunto de hábitos, operó como un sistema de retroalimentación que permitió al equipo mantener el rumbo y cerrar el proyecto con un producto entregable y validado.

Eso no quiere decir que no haya habido decisiones que fallaron. Por ejemplo, el equipo considera que la validación un poco tardía con usuarios candidatos sobre el problema, si bien justificada bajo múltiples razones, como el hecho de que el equipo quiso enfocarse en los dolores del cliente y la solución que este proponía, además de ya contar con la perspectiva del lado de los candidatos a través las experiencias propias de los miembros del equipo, también entendemos que tal vez una validación del problema más temprana con usuarios candidatos podría haber llevado a mejores resultados.

Sin embargo, es pertinente aclarar que, durante esta validación, no se encontraron grandes descubrimientos que no hubieran sido ya planteados por el equipo. A grandes rasgos, el equipo considera que la forma de trabajo y las decisiones tomadas en referencia a esta fueron muy positivas y enfocadas en aportar valor.

13.2.3. Tecnologías y Organización

Finalmente, llegamos al último conjunto de decisiones que conformaron el proyecto.

Por un lado, la organización del trabajo para la etapa de desarrollo. El equipo consideró muy acertada la separación en hitos e iteraciones, con un planteamiento que permitió desarrollar un sistema complejo y avanzado partiendo de elementos sencillos y básicos, pero clave, introduciendo mejoras sobre el pasar de las iteraciones e hitos. Se trabajó continuamente en mejorar el producto, con sesiones de usabilidad y *testing*, evaluaciones y mejoras del sistema de *matching*, agregando cada vez más valor al producto sin que este trabajo tornara en un proceso confuso o abrumador, incluso considerando el gran alcance del mismo y las limitaciones de recursos.

Por otro lado, la toma de decisión del *stack* tecnológico, que como ya se ha detallado, no fue una decisión de un día, sino más bien un ida y vuelta entre el cliente y el equipo, con análisis y evaluación constante.

La decisión de un sistema *full stack* con *Next.js* monolítico el equipo entiendo fue muy acertada, permitiéndonos desarrollar un sistema complejo con una tecnología nueva sin dificultades excesivas. Además, permitió al equipo aprender y mejorar sus habilidades, uno de los objetivos del proyecto, así como entregar al cliente un producto que cumple con sus requerimientos tecnológicos y deseos.

De la mano de esto, la decisión de no solo implementar un sistema complementario para IA, sino específicamente la elección de *embeddings* como solución para el requerimiento principal del sistema, el *matching* con inteligencia artificial, fue, según el equipo, sumamente acertada. Esta elección permitió alcanzar los objetivos del producto y la intención del mismo. Teniendo en cuenta incluso la evaluación de otras herramientas o tecnologías, esta no solo parecía ser la que más se acercaba a lo requerido y posible, sino que, tras su implementación y comprobación de desempeño, aunque falte evaluar su funcionalidad en “el mundo real”, ha demostrado haber sido un camino correcto a tomar.

13.4. *Roadmap* y mejoras futuras

Ya llegando al final de este documento y el proyecto, resulta pertinente proyectar el futuro posible del producto y de su evolución más allá del producto entregado. Si bien el equipo considera que la versión actual de Whizdy cumple plenamente con los objetivos del proyecto y constituye un producto cerrado y funcional, el aprendizaje obtenido a lo largo del desarrollo permitió identificar múltiples oportunidades de mejora y expansión que, de abordarse en nuevas fases, aportarían un valor sustancial al sistema.

El camino a futuro puede dividirse en dos etapas claras: una fase de consolidación técnica y funcional, orientada a perfeccionar y extender las capacidades del producto actual, y una fase de escalamiento y comercialización, enfocada en la apertura del producto al mercado y en la sostenibilidad de su operación.

Esta primera etapa de consolidación técnica y funcional comprendería todas aquellas mejoras que el equipo identificó durante el desarrollo pero que, por razones de alcance, tiempo o priorización, no fueron incluidas en la versión actual. Su implementación permitiría refinar la experiencia de uso, aumentar la estabilidad del sistema y fortalecer su valor operativo para el cliente.

Entre las principales líneas de trabajo se destacan:

- Integraciones complementarias: conexión directa con *Calendly* para la gestión de entrevistas, integración con *LinkedIn* para inicio de sesión y carga de perfil profesional, y futuras integraciones con plataformas de comunicación internas.
- Optimización del algoritmo de *matching*: revisión de los procesos de normalización de datos de candidatos y vacantes antes de la comparación, evaluación de nuevas estrategias de ponderación y mejora de los modelos de *embeddings* utilizados.
- Sistema de notificaciones ampliado: incorporación de preferencias configurables por usuario, notificaciones internas dentro de la plataforma y panel de control centralizado para su administración.
- Comunicación y *feedback*: creación y edición de templates personalizados por empresa para automatizar mensajes de seguimiento, *feedback* y contacto con candidatos.

- Administración y analítica: desarrollo de un panel de administración global que permita visualizar métricas, indicadores de uso y estadísticas agregadas de empresas, vacantes y candidatos.
- Mejoras de experiencia y configuración: edición dinámica de catálogos de opciones, administración de prompts utilizados por IA para generación de textos o explicaciones de *matching*, y refactorización de componentes clave para mantener escalabilidad y coherencia técnica.
- Infraestructura y rendimiento: configuración del ambiente de producción con reglas de autoescalado y balanceo de carga, actualmente contempladas en la arquitectura pero no activadas en el contexto del proyecto.
- Carga inteligente de perfiles: posibilidad para los candidatos de completar su perfil profesional mediante la subida de su currículum en PDF, permitiendo la extracción y estructuración automática de la información relevante mediante IA.
- Soporte multilenguaje: incorporación de un sistema multilingüe que permita utilizar la plataforma en distintos idiomas, comenzando por versiones en inglés y español, garantizando la expansión del producto a nuevos mercados y una experiencia inclusiva para todos los usuarios.
- Chat o mensajería en tiempo real con el recruiter: incorporar un sistema de comunicación Empresa-Candidato dentro del sistema manteniendo la comunicación más fluida y sencilla.

Superada la etapa de consolidación, el siguiente paso natural es la expansión del producto hacia un modelo comercial que permita su utilización por parte de múltiples organizaciones. Esta fase responde a la visión inicial del cliente: convertir Whizdy en una plataforma *SaaS* de uso abierto, ofreciendo su servicio como herramienta de *scouting* y selección asistida por IA.

Las líneas principales de desarrollo para esta etapa incluyen:

- Pasarela de pagos y modelo de suscripción: integración con servicios como *Stripe* para habilitar un sistema de facturación automatizado y la implementación del modo *freemium*, con planes escalonados que diferencien funcionalidades y límites según el tipo de usuario o empresa.
- Multi-cuentas corporativas: permitir que cada empresa cuente con múltiples usuarios internos con roles y permisos diferenciados (por ejemplo, reclutadores, managers o administradores).

- Versión *mobile*: desarrollo de una versión optimizada para dispositivos móviles, tanto para candidatos como para empresas, manteniendo la coherencia del sistema de diseño actual.
- Evaluaciones integradas: incorporación de assessments directamente dentro de la plataforma, permitiendo a las empresas crear y asignar pruebas específicas por vacante, así como *tests* psicométricos y de *fit* cultural independientes, potenciando la capacidad de evaluación integral.
- Expansión de IA generativa: edición avanzada de prompts y ampliación de las capacidades de generación de texto (resúmenes de candidatos, explicaciones de *matching* y sugerencias contextuales para empresas y candidatos).

Cada uno de estos avances sigue el mismo principio que guió el desarrollo original: evolucionar sin perder foco ni trazabilidad. Las mejoras propuestas no implican un rediseño del producto, sino su maduración progresiva hacia un sistema más completo, escalable y comercialmente viable.

El equipo confía en que, de retomarse el desarrollo, el proyecto podría transitar de un producto interno validado a un producto de mercado sólido, manteniendo los valores que definieron su construcción: claridad técnica, foco en la experiencia de usuario y una integración responsable de la inteligencia artificial como herramienta de apoyo y no de sustitución.

13.5. Entrega final: *handoff* del producto y documentación al cliente

Un aspecto que no puede pasar desapercibido, y que, aunque sea uno de los pasos finales, resulta esencial, es el *handoff* del producto y la documentación al cliente. Este proceso define en gran medida con qué capacidad un futuro equipo del cliente podrá continuar con el desarrollo, mantenimiento y uso del sistema. En otras palabras, el *handoff* no es un acto de cierre, sino el punto de partida para la continuidad del producto.

El proceso se estructura en dos grandes componentes: los elementos entregados y las acciones realizadas junto al cliente.

En primer lugar, se entrega el presente documento como documentación principal del producto, el cual reúne las decisiones técnicas, arquitectónicas y metodológicas tomadas durante el proyecto. Su valor radica en ofrecer una visión integral del sistema y del razonamiento detrás de cada elección, algo difícil de inferir al observar únicamente el código.

En conjunto, se entregan los cuatro repositorios oficiales de código fuente, todos contenidos dentro de la organización de *GitHub* Whizdy y acompañados de su respectiva documentación interna:

- *Whizdy-App*: *frontend* y *backend* principal, desarrollado en *Next.js*.
- *Whizdy-Matching*: aplicación de IA en *Python*/*FastAPI*, responsable del procesamiento y *matching* de candidatos y vacantes.
- *Whizdy-Infra*: infraestructura como código, definida en *Terraform*, que permite reproducir el entorno de despliegue completo en *AWS*.
- *Whizdy-LoadTest*: *scripts* y configuraciones de pruebas de carga (*K6*), utilizados para evaluar el rendimiento y la escalabilidad del sistema.

Cada repositorio cuenta con un *README* detallado, que incluye información sobre estructura de carpetas, comandos útiles, dependencias, configuraciones, variables de entorno y pasos de ejecución. Esto garantiza que un equipo técnico externo pueda comprender, ejecutar y modificar el sistema sin depender del equipo original.

Asimismo, se entrega documentación embebida de las *APIs* mediante la herramienta *Swagger*, la cual describe de forma interactiva el comportamiento de cada endpoint, sus

parámetros, tipos de respuesta y códigos de error. Esta capa de documentación técnica facilita la exploración, validación y futura integración del sistema con otras plataformas.

Como parte del proceso de transferencia, se proporcionan al cliente todas las credenciales asociadas a los servicios del sistema, creadas específicamente para este proyecto. Entre ellas, se incluye la cuenta de correo *whizdy.noreply@gmail.com*, utilizada para el envío automático de notificaciones y validaciones dentro del sistema. Esta cuenta fue empleada de forma unificada en los diferentes servicios asociados, incluyendo *AWS*, *Sentry* y demás integraciones necesarias. Este enfoque busca simplificar la continuidad operativa del cliente, evitando la necesidad de un redespliegue o reconfiguración inmediata, ya que el entorno actual puede seguir funcionando con las credenciales ya establecidas.

Por otro lado también se le comparte al cliente el proyecto en *Jira*, el cual continúe en forma de ticket cada tarea tarea realizada, cada *bug* encontrado y resuelto, todas las mejoras realizadas y por supuesto, todas las historias de usuario. Dando un histórico completo de todo el trabajo realizado sobre el sistema, conteniendo todo el trabajo realizado por el equipo más algunos de los *tickets* como mejoras o requerimientos funcionales, ya mencionados, que no se llegaron a realizar.

Finalmente, el equipo propone la realización de una sesión de resolución de consultas posterior a la entrega aún por definir, destinada a responder dudas específicas que puedan surgir tras la revisión del producto y su documentación. Esta instancia complementaria busca asegurar que el cliente pueda asimilar de forma completa la estructura técnica y funcional del sistema, y resolver posibles incertidumbres que excedan las consultas operativas habituales realizadas por canales como *Slack*.

El proceso de *handoff* marca el cierre formal del proyecto, pero también su continuidad natural. Al transferir un producto documentado, reproducible y plenamente operativo, se garantiza que el conocimiento técnico no quede limitado al equipo desarrollador, sino que pase a formar parte del activo del cliente. La claridad de la documentación, la trazabilidad de las decisiones y la entrega estructurada del código permiten que el sistema pueda seguir evolucionando sin depender de su equipo original, cumpliendo así uno de los principios fundamentales del desarrollo profesional: entregar no solo un producto, sino también su conocimiento.

13.6. Evaluación de cumplimiento de los objetivos del proyecto

En esta sección se analizan los resultados alcanzados en relación con los objetivos definidos al inicio del proyecto. El propósito es revisar, desde una mirada global, el grado de cumplimiento, en medida de lo posible a partir del estado actual del proyecto y producto, de los distintos tipos de objetivos, académicos, del proceso y del producto, y reflexionar sobre cómo se materializaron en la práctica. Más que una evaluación numérica, se trata de una lectura cualitativa de los logros obtenidos y de cómo estos objetivos funcionaron, efectivamente, como guía y brújula del proyecto.

13.6.1. Objetivos académicos

Desde el comienzo, el proyecto se propuso como un espacio de aplicación integral de los conocimientos adquiridos durante la carrera, enfrentando un escenario real, con un cliente, tiempos y restricciones concretas. En ese sentido, Whizdy permitió poner en práctica conceptos de ingeniería de *software*, gestión de proyectos, arquitectura, pruebas, documentación y metodologías ágiles en un entorno que exigió una mirada profesional. La magnitud del trabajo realizado y la coordinación entre tres integrantes con perfiles complementarios representó un desafío que consolidó aprendizajes previos y permitió desarrollar nuevas competencias vinculadas a la gestión y colaboración técnica.

Asimismo, el proyecto implicó un fuerte componente de aprendizaje tecnológico. Se incorporaron herramientas y tecnologías que ninguno de los integrantes había utilizado en profundidad, como *Next.js*, *Sentry*, *Terraform* o el uso de *embeddings* y bases vectoriales aplicadas a sistemas de *matching*. Estas experiencias ampliaron el dominio técnico del equipo y permitieron incorporar metodologías modernas en temas de IA, observabilidad y despliegue en la nube. Por otra parte, la extensión y complejidad del sistema requirió desarrollar capacidades de gestión y pensamiento sistémico propias de la creación de un producto, no solo de un *software* funcional. En conjunto, el equipo no solo aplicó lo aprendido, sino que aprendió mucho más de lo que imaginaba al inicio: de la práctica, de los errores y del producto que fue construyendo.

13.6.2. Objetivos del proceso

En cuanto a los objetivos del proceso, el proyecto también evidencia un cumplimiento sólido y sostenido. Desde el comienzo se priorizó una comunicación clara y continua con el cliente, manteniendo contacto permanente a través de *Slack* y reuniones planificadas, como las demos, revisiones y sesiones de priorización. La interacción fue constante, transparente y bidireccional, generando un flujo de información fluido que permitió ajustar el trabajo sin fricciones y sostener la alineación entre expectativas, tiempos y entregables.

De esta comunicación se desprende el segundo logro: la construcción de una relación sólida y colaborativa con el cliente. A lo largo del proceso, las decisiones se tomaron en conjunto, y tanto el cliente como el equipo mantuvieron una actitud de apertura y disponibilidad. Las revisiones de *backlog*, los informes de avance y los espacios de retroalimentación fueron instancias de mejora genuina, no de control. Este clima de confianza fue determinante para sostener la estabilidad del proyecto incluso ante los cambios de alcance y complejidad que surgieron durante su desarrollo.

Por otra parte, se logró aplicar de forma efectiva un marco de trabajo ágil, adaptado al contexto real del equipo. Las iteraciones, retros y *plannings* se mantuvieron a lo largo de todo el proyecto, evolucionando con cada ciclo. Los ajustes introducidos, como el fortalecimiento de la *Definition of Ready*, la división de *tickets* amplios y la estimación consensuada, consolidaron una dinámica previsible, transparente y orientada a la entrega continua de valor.

Finalmente, el proyecto cumplió con el objetivo de reflexionar y mejorar de forma continua. Cada iteración se convirtió en un espacio para repensar procesos, ajustar estrategias y aprender de lo realizado. Las retrospectivas documentadas, los informes internos y las decisiones de mejora registradas a lo largo de las iteraciones demuestran una madurez creciente en la gestión del equipo. En síntesis, los objetivos del proceso no solo se cumplieron, sino que dejaron un aprendizaje operativo que el equipo podría trasladar a futuros proyectos.

13.6.3. Objetivos del producto

Los objetivos vinculados al producto eran mejorar los procesos de adquisición de personal del cliente, ofreciendo una herramienta que optimizara la identificación, comparación y selección de candidatos. A través de un sistema de *matching* basado en IA y un conjunto integral de funcionalidades, el producto cumple con este propósito: centraliza información, reduce fricciones, agiliza la evaluación de perfiles y otorga trazabilidad al proceso de selección. Si bien su uso interno por parte del cliente aún no ha comenzado formalmente, las validaciones realizadas y la satisfacción expresada en las instancias de revisión confirman que el producto cumple con las necesidades que motivaron su creación.

Del mismo modo, se alcanzó con éxito el objetivo de aportar valor de forma continua. Cada iteración fue guiada por la entrega de funcionalidades útiles y validadas, priorizando siempre aquellas que generaban un impacto tangible en la experiencia del cliente. Esta filosofía de trabajo iterativo permitió que el producto fuera ganando valor progresivamente, consolidándose como una herramienta robusta y coherente con los objetivos de negocio del cliente.

La satisfacción del cliente también es un indicador claro del cumplimiento de los objetivos. Durante las demos y las instancias de presentación, el cliente manifestó una valoración positiva del resultado, destacando la dedicación del equipo, la calidad del producto.

En cuanto al objetivo de entregar un producto integral y validado, los resultados son evidentes. Se completaron todas las funcionalidades críticas previstas, se realizaron pruebas exhaustivas tanto a nivel funcional como no funcional, y se validaron los flujos principales desde las perspectivas de empresa y candidato. El sistema fue entregado desplegado, documentado y listo para operar sin carencias perceptibles ni dependencias externas.

Por último, surge el análisis más profundo: el objetivo de entregar un producto cerrado con potencial comercial. Lo que inicialmente se concebía como un *MVP* evolucionó hacia algo más ambicioso. A medida que el desarrollo avanzó, el producto superó ampliamente las expectativas planteadas tanto en alcance como en madurez. El número y complejidad de los requerimientos implementados, la calidad alcanzada en la experiencia de

usuario, las integraciones desarrolladas y la estabilidad del sistema permiten afirmar que ya no se trata de un producto mínimo, sino de un producto completo y operativo.

Es entonces que vuelve a surgir la pregunta ya mencionada en este documento ¿seguimos ante un *MVP* o ante un producto finalizado? La respuesta del equipo, tras el análisis, es clara: Whizdy ya trasciende la definición de *MVP*. No solo porque cumple con todos los requerimientos funcionales y no funcionales acordados, sino porque su nivel de completitud, validación y despliegue real lo posiciona como una herramienta lista para su uso interno por parte del cliente, e incluso apta para su apertura controlada hacia otras organizaciones.

Al momento de la entrega, el producto ya se encuentra desplegado, probado y documentado; permite gestionar procesos completos de reclutamiento, ofrece trazabilidad, *matching* inteligente, *feedback* automatizado, analítica, autenticación integrada, sistema de notificaciones y administración de perfiles. En definitiva, entrega valor real y autónomo, sin depender de intervenciones adicionales. Esto no solo supera las expectativas iniciales del cliente, sino también el objetivo original del equipo, transformando una meta académica en un producto con impacto tangible y potencial de expansión comercial.

El análisis de cumplimiento de los objetivos evidencia un resultado sobresaliente en los tres niveles planteados. En lo académico, el proyecto sirvió como cierre integrador de la carrera y como espacio de aprendizaje técnico y profesional. En lo procesal, consolidó un modo de trabajo maduro, ágil y colaborativo, con un fuerte vínculo cliente-equipo. Y en lo productivo, derivó en la entrega de una solución sólida, validada y de valor comprobable.

Más allá de las métricas, lo que se desprende es un aprendizaje transversal: la capacidad del equipo para transformar objetivos teóricos en resultados concretos, adaptarse ante la incertidumbre y sostener la calidad técnica y humana del proceso. Whizdy no solo cumple los objetivos que lo originaron, sino que los amplía, demostrando que detrás de cada línea de código hubo una comprensión profunda del problema, una ejecución rigurosa y una clara orientación a entregar valor real.

14. Reflexiones personales

Franco

Fue una experiencia muy enriquecedora en lo académico, profesional y personal.

El proyecto elegido nos permitió cumplir con los objetivos que nos habíamos propuesto para el Proyecto Final de Carrera: aplicar todos los conocimientos adquiridos durante la formación, construir un producto desde cero hasta llegar a una versión completamente funcional, y poner en práctica conceptos de todas las áreas del desarrollo de software.

También tuvimos la oportunidad de gestionar un proyecto real, comunicarnos con un cliente, aprender nuevas tecnologías y aplicar una metodología de investigación iterativa, en la que cada aprendizaje se integraba en el desarrollo de manera continua.

En lo personal, me siento muy satisfecho con el resultado que logramos como equipo y con todo lo que aprendí durante el proceso.

Martin

A mi parecer esta experiencia fue muy valiosa tanto a nivel de proyecto, como a nivel profesional. Poder realizar un proyecto de esta magnitud me pareció sumamente valioso al aportar experiencia en la gestión y desarrollo de un proyecto desde sus inicios. Además, al haber pasado por todas las fases del proyecto, como investigación, desarrollo, despliegue y el ida y vuelta con el cliente considero que es una buena preparación para el día de mañana a nivel laboral.

Otro punto valioso fue el poder aplicar muchos aspectos aprendido a lo largo de la carrera, el ver como todo aquellos que uno fue dando durante todos estos años tiene un motivo y aplicación, considero que es muy satisfactorio.

Estoy muy conforme con el resultado del proyecto, el crecimiento del equipo y con el camino recorrido con el mismo y espero que este documento pueda hacer ver el trabajo arduo y colaborativo de este proyecto.

Nicolás

Por mi parte considero esta experiencia llena de valor. Es una experiencia donde se mezclan la aplicación académica en conjunto con la comunicación con un cliente real y la toma de decisiones en un proyecto grande. Esto considero que nos da una buena experiencia para nuestra carrera profesional.

Por otro lado, la gestión completa y continua del proyecto te da una experiencia que no se obtiene en todos lados. Es una experiencia donde tus decisiones influyen al 100% y te obligan a tomar una postura más directiva, lo cual también considero muy valioso para el día de mañana.

Termino este proyecto sumamente conforme, considero que tanto el resultado como el camino transcurrido dieron resultados muy buenos y aprendizajes valiosos y espero podamos haber transmitido este gran proceso transcurrido.

15. Bibliografía

[1] *Scrum* “La Guía de Scrum”, Noviembre de 2020. [*Online*]. *Available*: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-Latin-South-American.pdf. *Accesed on*: Oct. 13, 2025.

[2] IMPO, “REGLAMENTACIÓN DE LA LEY 18.331, RELATIVO A LA PROTECCIÓN DE DATOS PERSONALES”, 15 de Septiembre 2009. [*Online*]. *Available*: https://www.impo.com.uy/bases/decretos/414-2009. *Accessed on*: Oct. 13, 2025.

[3] Atlassian, “*Kanban - A brief introduction*”, 2025. [*Online*]. *Available*: https://www.atlassian.com/agile/kanban. *Accessed on*: Oct. 13, 2025.

[4] Atlassian, “El consejero ágil”. [*Online*]. *Available*: https://www.atlassian.com/es/agile. *Accessed on*: Oct. 13, 2025.

[5] agilemanifesto.org, “Manifiesto por el Desarrollo Ágil de Software”, 2001. [*Online*]. *Available*: https://agilemanifesto.org/iso/es/manifesto.html. *Accessed on*: Oct. 13, 2025.

[6] Vercel, “Next.js Docs”, 2025. [*Online*]. *Available*: https://nextjs.org/docs. *Accessed on*: Oct. 13, 2025.

[7] Joel Barnard, “*What is embedding*?”. [*Online*]. *Available*: https://www.ibm.com/think/topics/embedding. *Accessed on*: Oct. 13, 2025.

[8] Amazon Web Services, “¿Qué es *RAG* (generación aumentada por recuperación)?”. [*Online*]. *Available*: https://aws.amazon.com/what-is/retrieval-augmented-generation/. *Accessed on*: Oct. 13, 2025.

[9] Amazon Web Services, “¿Qué es un *LLM* (modelo de lenguaje de gran tamaño)?”. [*Online*]. *Available*: https://aws.amazon.com/what-is/large-language-model/. *Accessed on*: Oct. 13, 2025.

[10] OpenAI, “*Prompting*”. [*Online*]. *Available*: https://platform.openai.com/docs/guides/prompting. *Accessed on*: Oct. 13, 2025.

[11] Dave Bergmann, “*What is fine-tuning*?”. [*Online*]. *Available*: https://www.ibm.com/think/topics/fine-tuning. *Accessed on*: Oct. 13, 2025.

[12] Ruby on Rails, "*Ruby on Rails Guides*". [*Online*]. *Available*: https://guides.rubyonrails.org/. *Accessed on*: Oct. 13, 2025.

[13] Prisma Data, Inc., "ORM". [*Online*], 2025. *Available*: https://www.prisma.io/docs/orm. *Accessed on*: Oct. 13, 2025.

[14] Qdrant, "*Qdrant Documentation*", 2025. [*Online*]. *Available*: https://qdrant.tech/documentation/. *Accessed on*: Oct. 13, 2025.

[15] Tailwind Labs Inc., "*Install Tailwind CSS with Next.js*", 2025. [*Online*]. *Available*: https://tailwindcss.com/docs/installation/framework-guides/nextjs. *Accessed on*: Oct. 13, 2025.

[16] Lokesh Gupta, "*What is REST*?", 1 de Abril 2025. [*Online*]. *Available*: https://restfulapi.net/. *Accessed on*: Oct. 13, 2025.

[17] shadcn, "*Introduction - shadcn/ui*". [*Online*]. *Available*: https://ui.shadcn.com/docs. *Accessed on*: Oct. 13, 2025.

[18] Amazon Web Services, "¿Qué es la arquitectura orientada a servicios (*SOA*)?". [*Online*]. *Available*: https://aws.amazon.com/es/what-is/service-oriented-architecture/. *Accessed on*: Oct. 13, 2025.

[19] Confluent, "*What is Event Driven Architecture*?". [*Online*]. *Available*: https://www.confluent.io/learn/event-driven-architecture/. *Accessed on*: Oct. 13, 2025.

[20] Swagger, "*Understanding the API-First Approach to Building Products*". [*Online*]. *Available*: https://swagger.io/resources/articles/adopting-an-api-first-approach/. *Accessed on*: Oct. 13, 2025.

[21] Swagger, "*Swagger Documentation*". [*Online*]. *Available*: https://swagger.io/docs/. *Accessed on*: Oct. 13, 2025.

[22] The PostgreSQL Global Development Group, "*PostgreSQL 18.0 Documentation*". [*Online*]. *Available*: https://www.postgresql.org/docs/current/. *Accessed on*: Oct. 13, 2025.

[23] Google, “Crear credenciales de acceso”, 21 de Agosto 2025. [*Online*]. *Available*: https://developers.google.com/workspace/guides/create-credentials. *Accessed on*: Oct. 13, 2025.

[24] Amazon Web Services, “¿Qué son los mensajes de publicación y suscripción?”. [*Online*]. *Available*: https://aws.amazon.com/what-is/pub-sub-messaging/. *Accessed on*: Oct. 13, 2025.

[25] Hetzner, “*Creating a Server*”. [*Online*]. *Available*: https://docs.hetzner.com/cloud/servers/getting-started/creating-a-server. *Accessed on*: Oct. 13, 2025.

[26] Nico Botha, “*(Step-By-Step) How To Add Font In NextJS*”, 6 de Abril 2023. [*Online*]. *Available*: https://shipsaas.com/blog/how-to-add-font-in-nextjs. *Accessed on*: Oct. 13, 2025.

[27] Amazon Web Services, “Descripción general de Amazon Web Services”, 7 de Agosto 2024. [*Online*]. *Available*: https://docs.aws.amazon.com/es_es/whitepapers/latest/aws-overview/introduction.html. *Accessed on*: Oct. 13, 2025.

[28] Atlassian, “Flujo de trabajo de *Gitflow*”. [*Online*]. *Available*: https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow. *Accessed on*: Oct. 13, 2025.

[29] HashiCrop, “*What is Terraform*?”. [*Online*]. *Available*: https://developer.hashicorp.com/terraform/intro. *Accessed on*: Oct. 13, 2025.

[30] HashiCrop, “*Introduction to JSON Web Tokens*”. [*Online*]. *Available*: https://www.jwt.io/introduction. *Accessed on*: Oct. 13, 2025.

[31] Grafana Labs, “Grafana k6”. [*Online*]. *Available*: https://grafana.com/docs/k6/latest. *Accessed on*: Oct. 13, 2025.

[32] Sentry, “*Sentry for Next.js*”. [*Online*]. *Available*: https://docs.sentry.io/platforms/javascript/guides/nextjs/. *Accessed on*: Oct. 13, 2025.

[33] Martin Fowler, “*Patterns of Enterprise Application Architecture*”. *Addison Wesley*, Noviembre 2002.

[34] GitHub, Inc., “Acerca de GitHub y Git”, 2025. [*Online*]. *Available*: https://docs.github.com/es/get-started/start-your-journey/about-github-and-git. *Accessed on*: Oct. 13, 2025.

[35] Jeff Patton., “*User Story Mapping: Discover the Whole Story, Build the Right Product*”. O’Reilly Media, 2014.

[36] ISO/IEC JTC 1, Subcommittee 7, “*Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Product quality model*”. ISO/IEC, Noviembre 2023.

[37] Thorben, “*Dependency Injection Explained (With Examples)*”, 1 de Mayo 2023. [*Online*]. *Available*: https://stackify.com/dependency-injection/. *Accessed on*: Oct. 13, 2025.

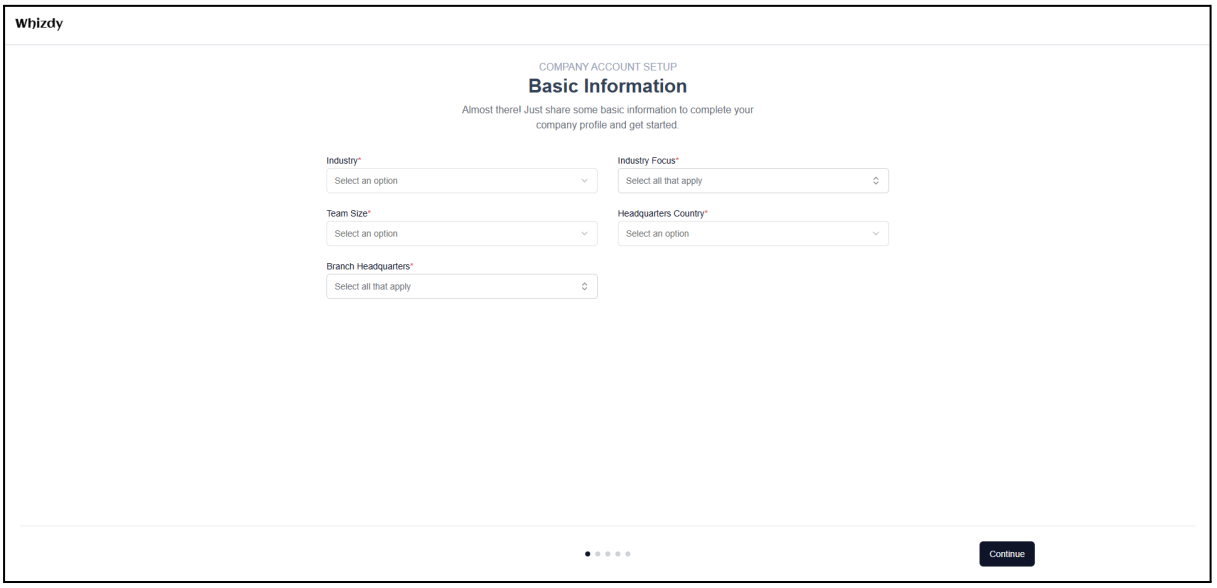
[38] Refactorig Guru, “*Factory Method*”, 2025. [*Online*]. *Available*: https://refactoring.guru/es/design-patterns/factory-method. *Accessed on*: Oct. 13, 2025.

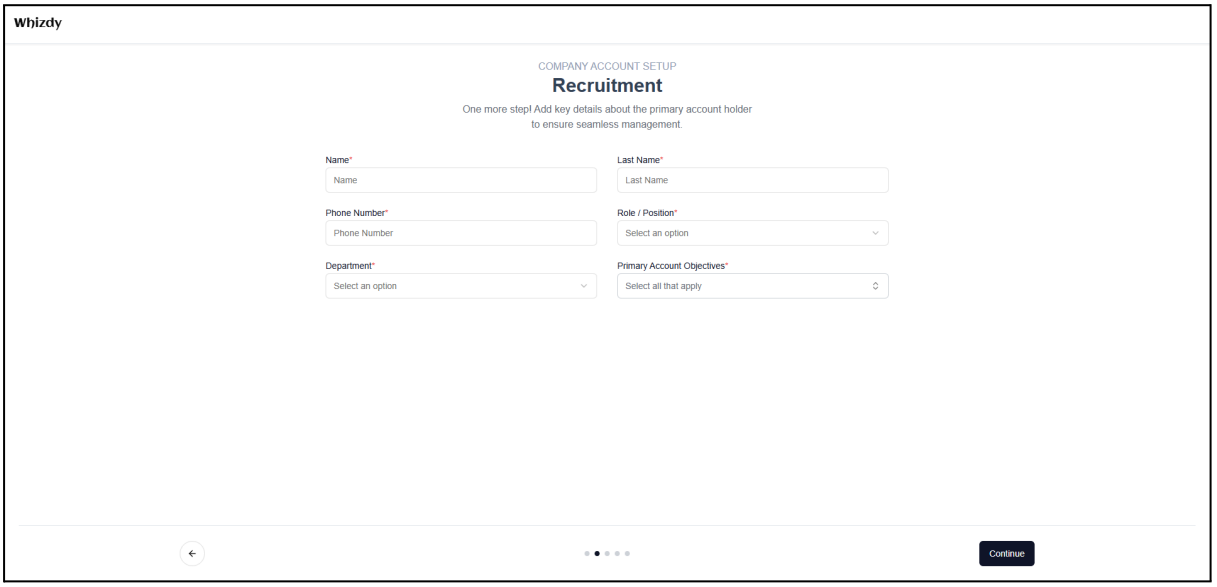
16. Anexo

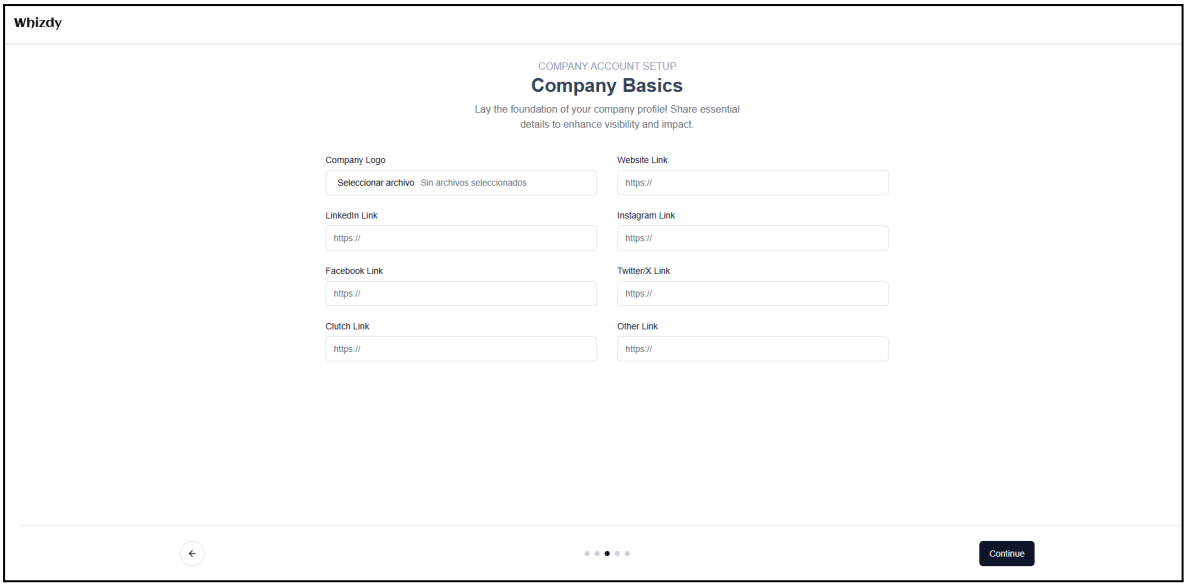
16.1. Imágenes del sistema

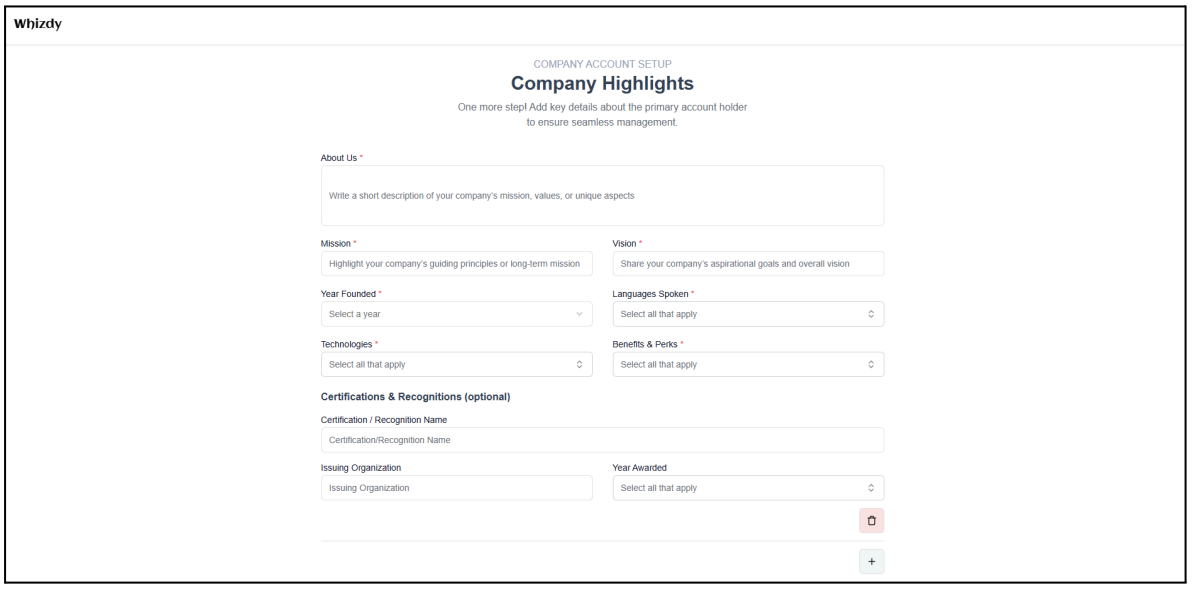
Catalogo de imagenes de principales requerimientos funcionales visuales

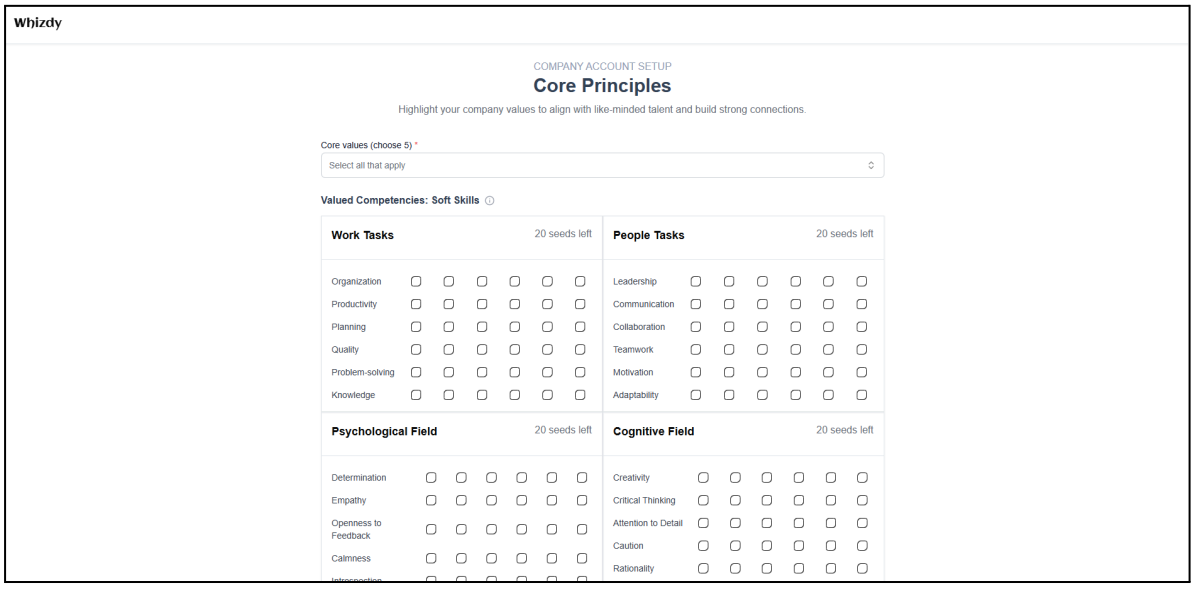
RF9 - Onboarding de empresa:



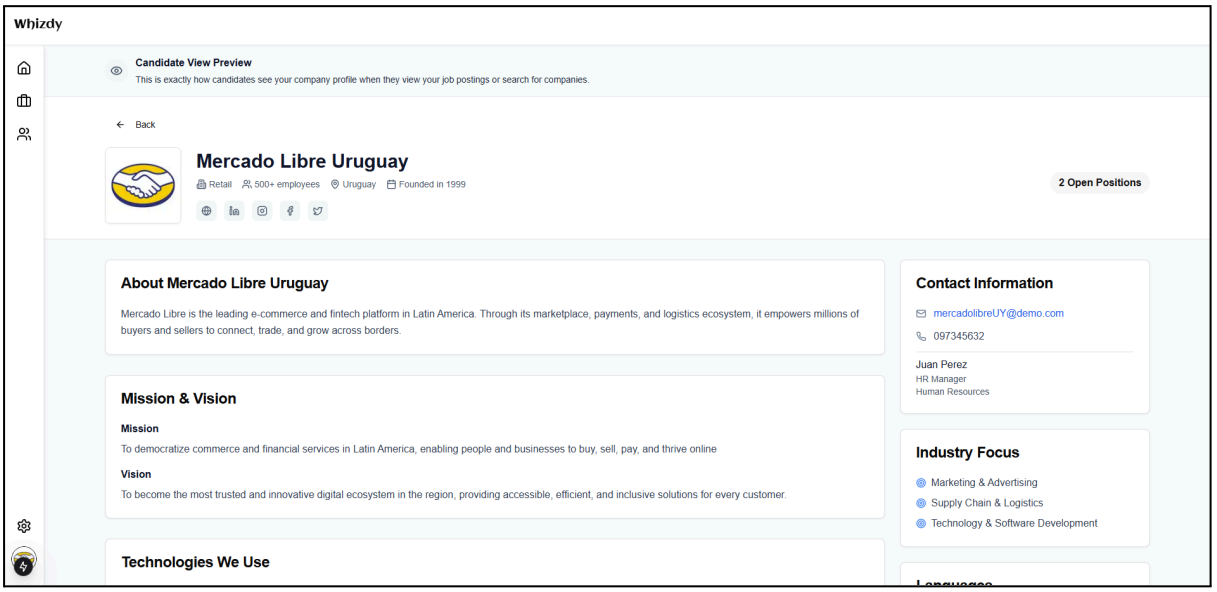


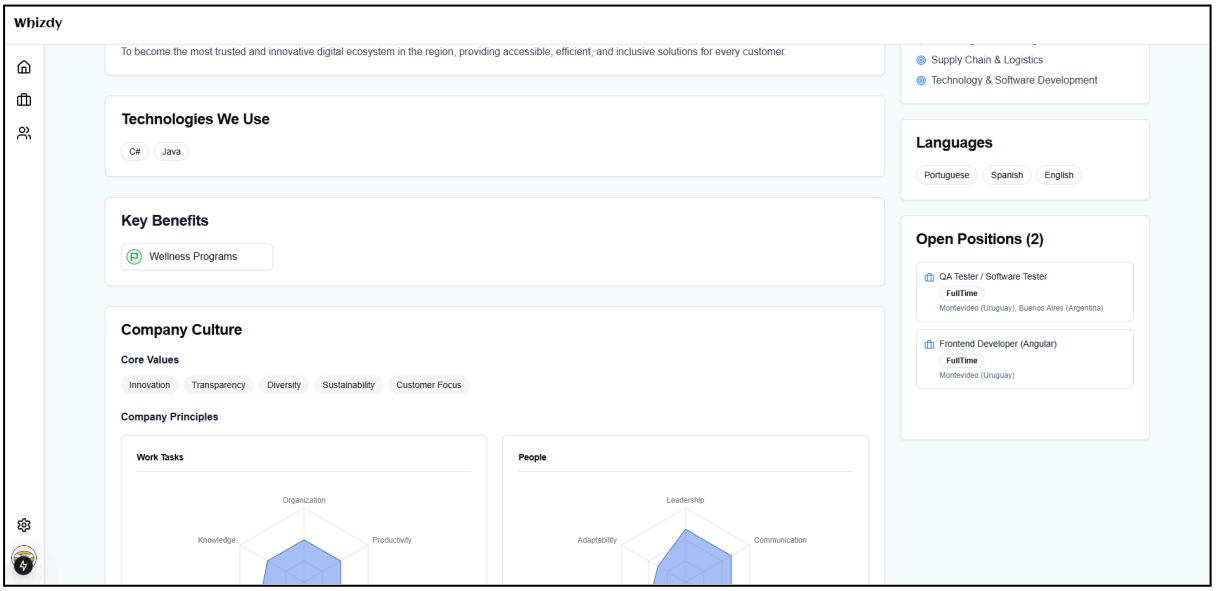

Whizdy
COMPANY ACCOUNT SETUP
Company Basics
Lay the foundation of your company profile! Share essential details to enhance visibility and impact.
Company Logo
Seleccionar archivo
Sin archivos seleccionados
Website Link
https://
LinkedIn Link
https://
Instagram Link
https://
Facebook Link
https://
Twitter/X Link
https://
Clutch Link
https://
Other Link
https://
Continue

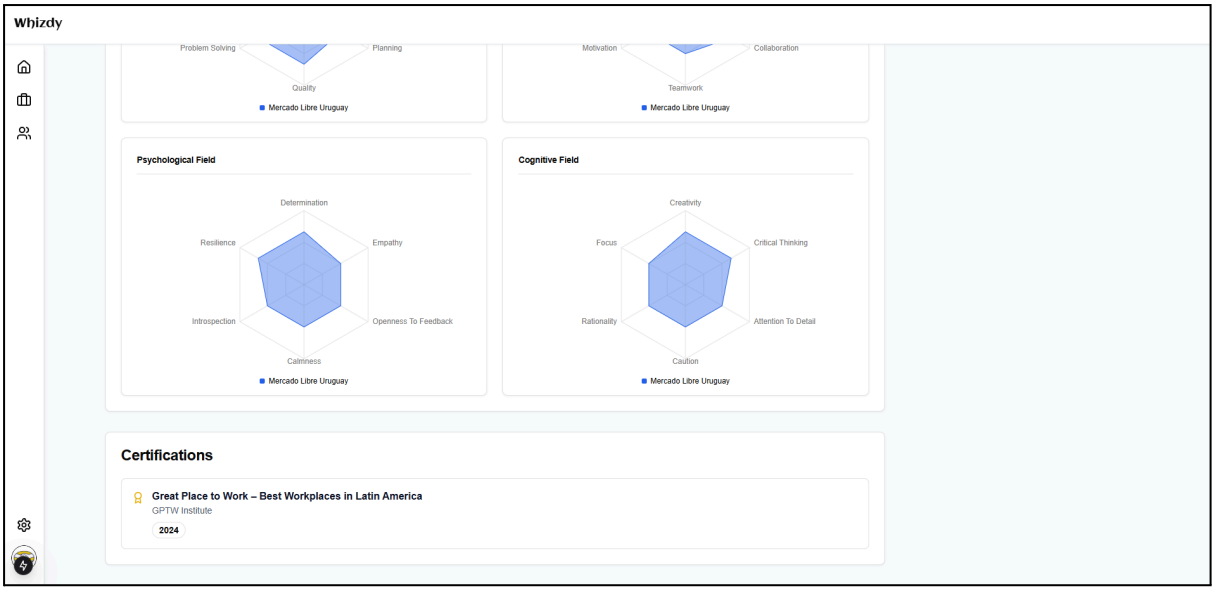

Whizdy
COMPANY ACCOUNT SETUP
Company Highlights
One more step! Add key details about the primary account holder to ensure seamless management.
About Us *
Write a short description of your company's mission, values, or unique aspects
Mission *
Highlight your company's guiding principles or long-term mission
Vision *
Share your company's aspirational goals and overall vision
Year Founded *
Select a year
Languages Spoken *
Select all that apply
Technologies *
Select all that apply
Benefits & Perks *
Select all that apply
Certifications & Recognitions (optional)
Certification / Recognition Name
Certification/Recognition Name
Issuing Organization
Issuing Organization
Year Awarded
Select all that apply


Whizdy
COMPANY ACCOUNT SETUP
Core Principles
Highlight your company values to align with like-minded talent and build strong connections.
Core values (choose 5) *
Select all that apply
Valued Competencies: Soft Skills
Work Tasks
20 seeds left
Organization
Productivity
Planning
Quality
Problem-solving
Knowledge
People Tasks
20 seeds left
Leadership
Communication
Collaboration
Teamwork
Motivation
Adaptability
Psychological Field
20 seeds left
Determination
Empathy
Openness to Feedback
Calmness
Cognitive Field
20 seeds left
Creativity
Critical Thinking
Attention to Detail
Caution
Rationality

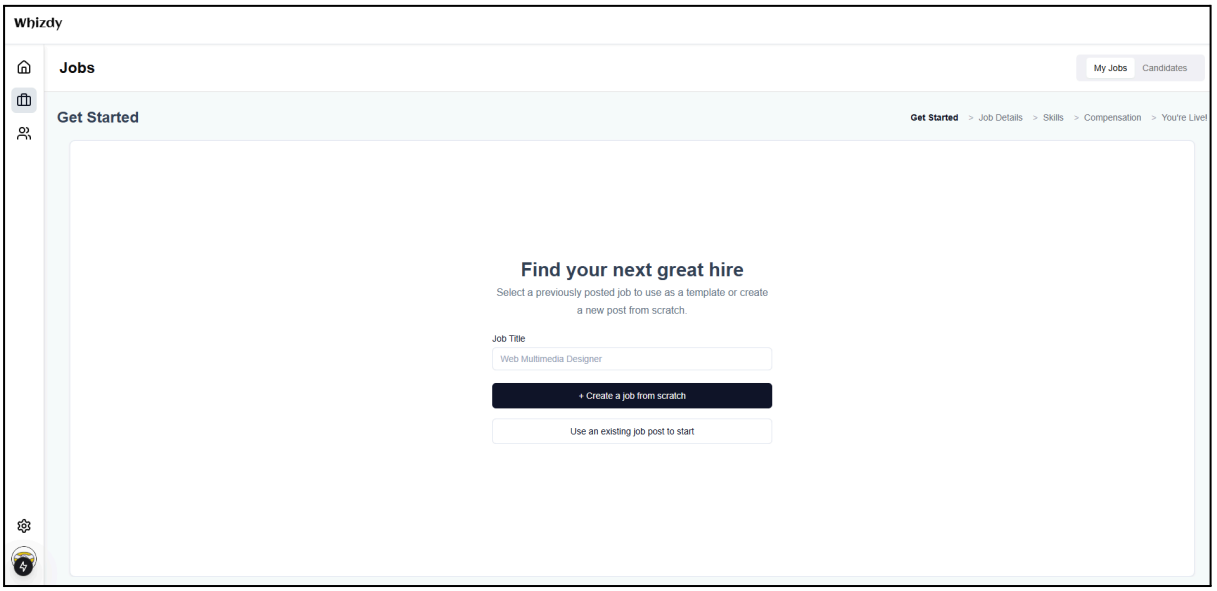
RF11 - Visualización de perfil:

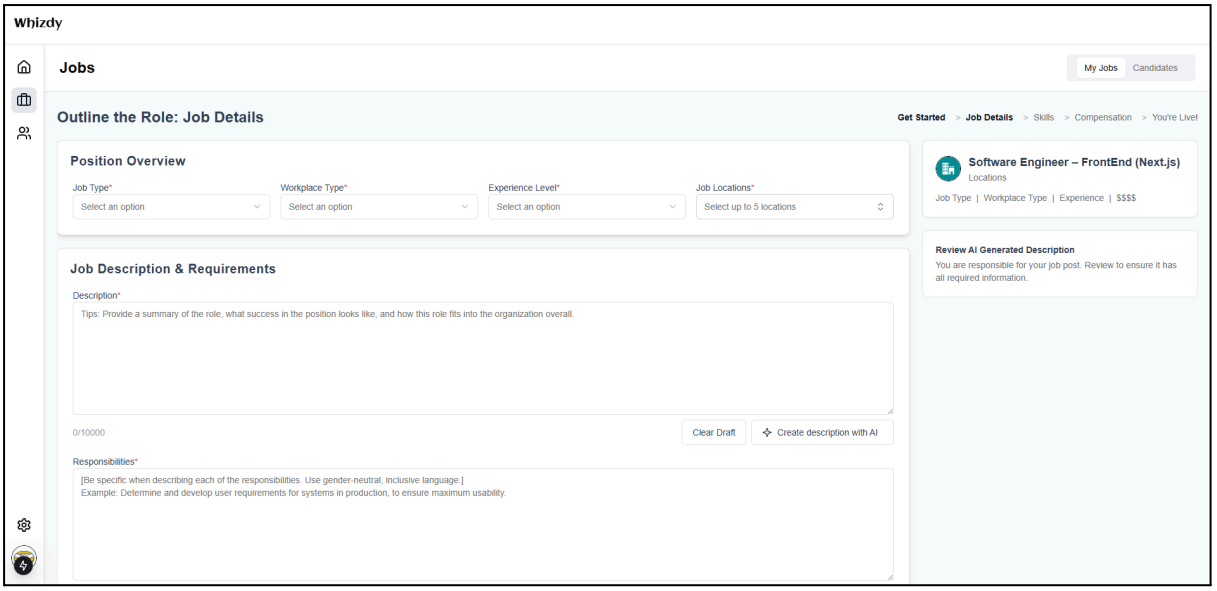


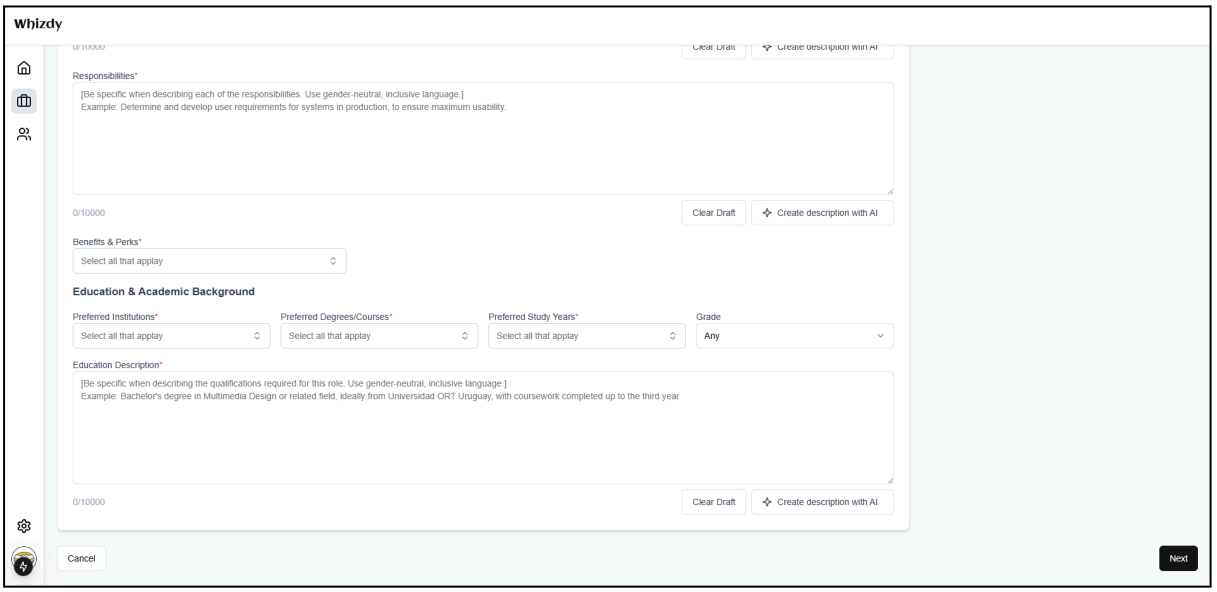


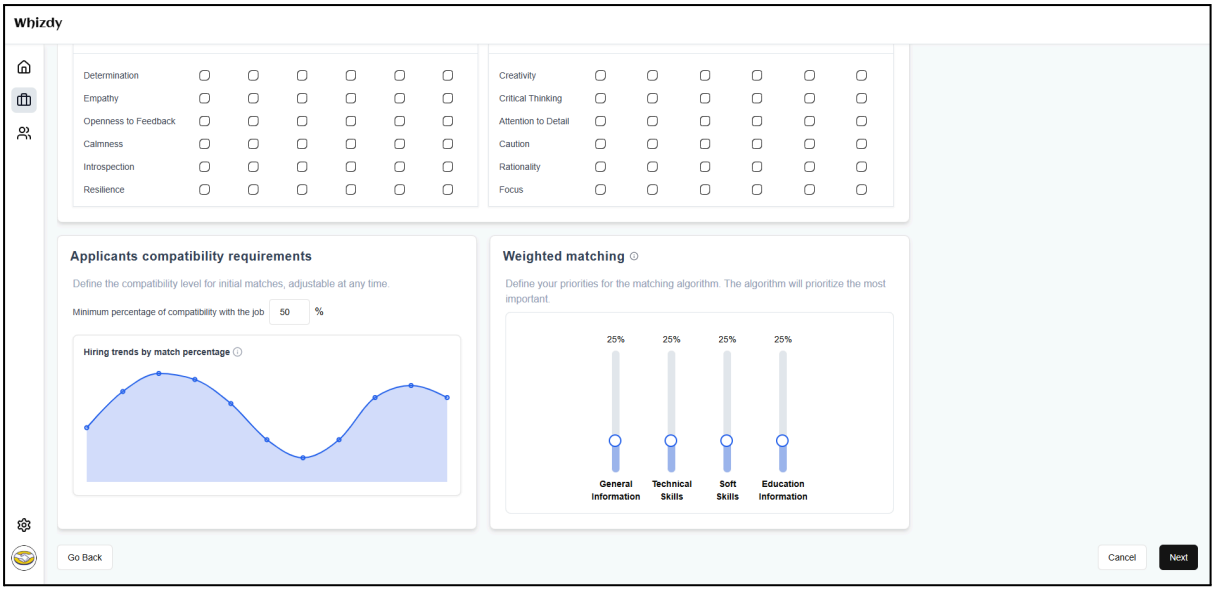


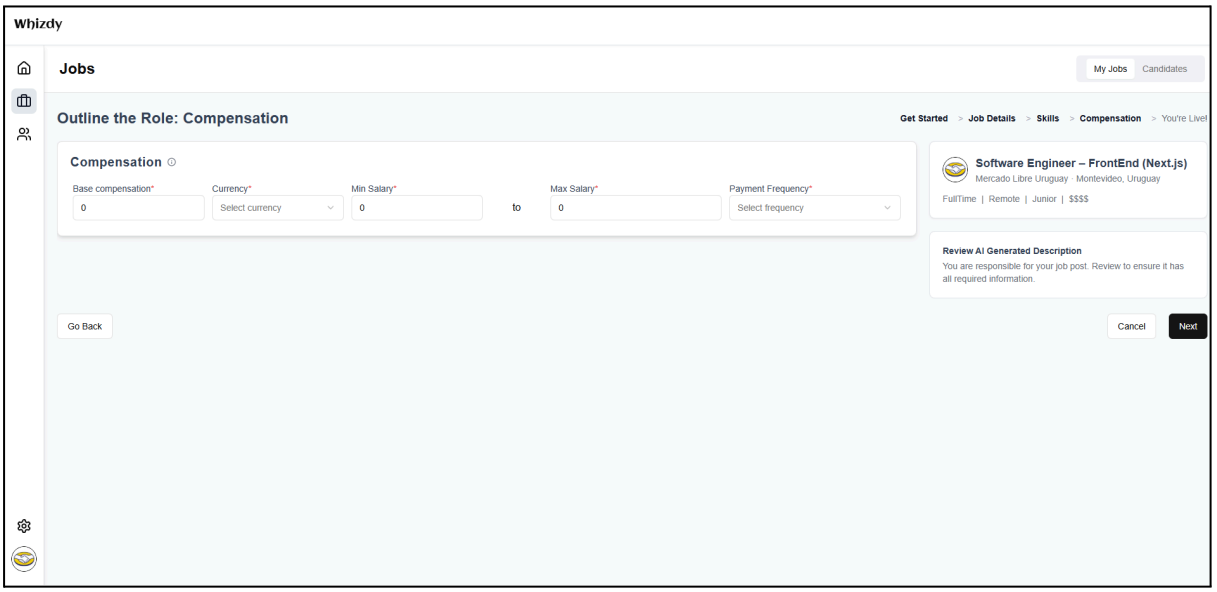
RF14 - Crear y publicar vacantes:



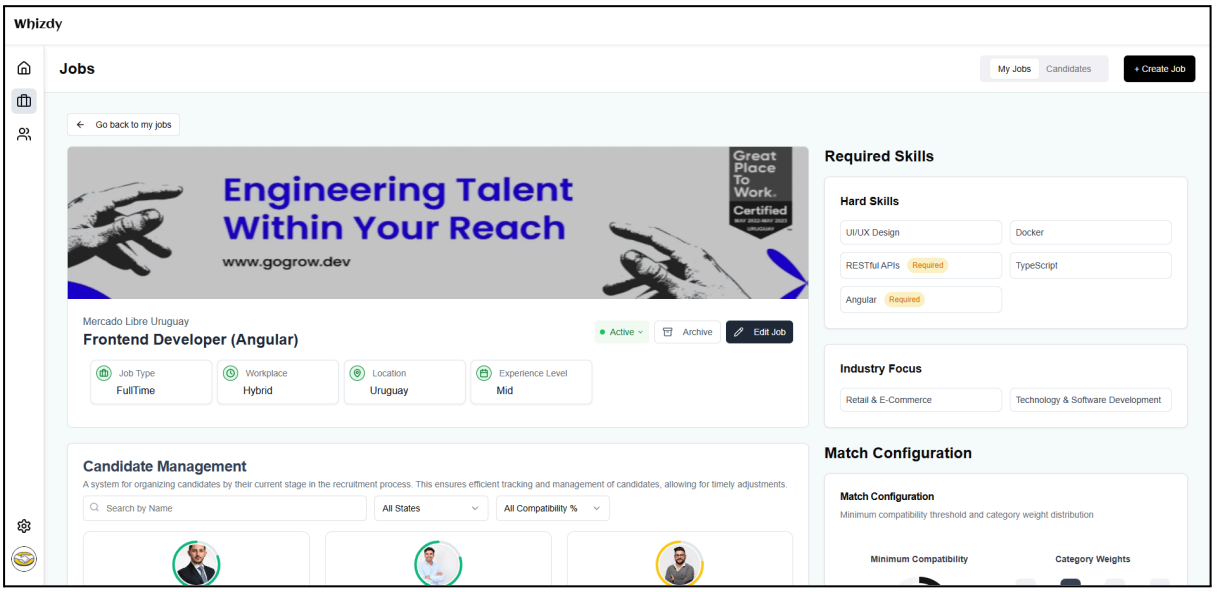

Whizdy
Jobs
My Jobs
Candidates
Outline the Role: Job Details
Get Started > Job Details > Skills > Compensation > You're Live!
Position Overview
Job Type*
Select an option
Workplace Type*
Select an option
Experience Level*
Select an option
Job Locations*
Select up to 5 locations
Software Engineer – FrontEnd (Next.js)
Locations
Job Type | Workplace Type | Experience | $$$$
Review AI Generated Description
You are responsible for your job post. Review to ensure it has all required information.
Job Description & Requirements
Description*
Tips: Provide a summary of the role, what success in the position looks like, and how this role fits into the organization overall.
0/10000
Clear Draft
Create description with AI
Responsibilities*
[Be specific when describing each of the responsibilities. Use gender-neutral, inclusive language.]
Example: Determine and develop user requirements for systems in production, to ensure maximum usability.

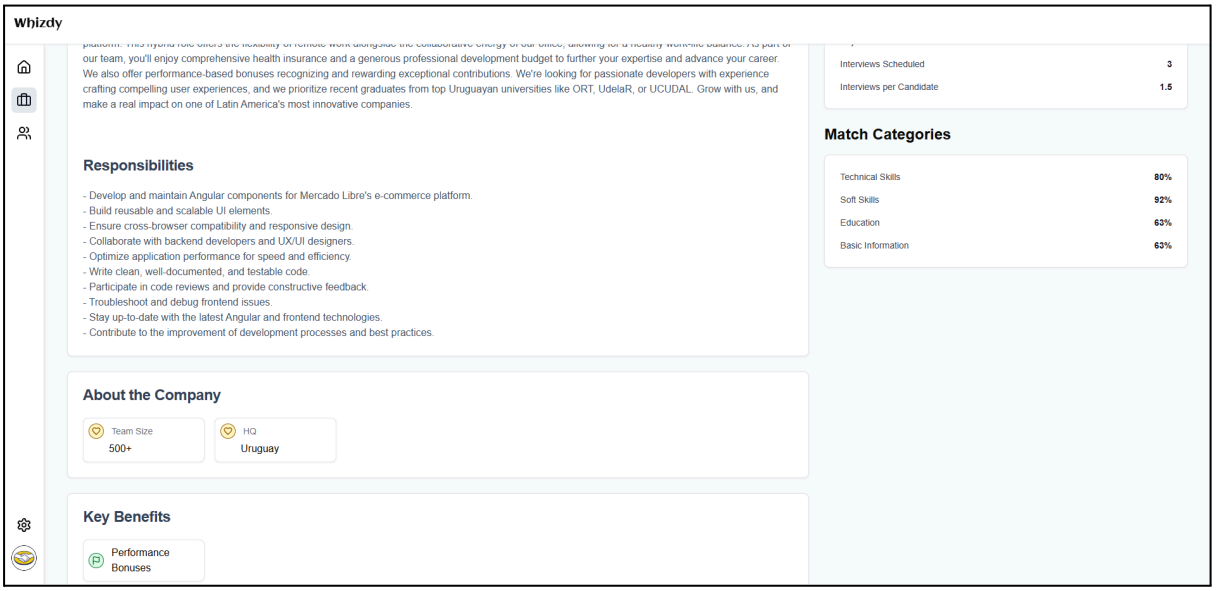

Whizdy
Responsibilities*
[Be specific when describing each of the responsibilities. Use gender-neutral, inclusive language.]
Example: Determine and develop user requirements for systems in production, to ensure maximum usability.
0/10000
Clear Draft
Create description with AI
Benefits & Perks*
Select all that applay
Education & Academic Background
Preferred Institutions*
Select all that applay
Preferred Degrees/Courses*
Select all that applay
Preferred Study Years*
Select all that applay
Grade
Any
Education Description*
[Be specific when describing the qualifications required for this role. Use gender-neutral, inclusive language.]
Example: Bachelor's degree in Multimedia Design or related field, ideally from Universidad ORT Uruguay, with coursework completed up to the third year
0/10000
Clear Draft
Create description with AI
Cancel
Next


Whizdy
Determination
Empathy
Openness to Feedback
Calmness
Introspection
Resilience
Creativity
Critical Thinking
Attention to Detail
Caution
Rationality
Focus
Applicants compatibility requirements
Define the compatibility level for initial matches, adjustable at any time.
Minimum percentage of compatibility with the job
50
%
Hiring trends by match percentage
Weighted matching
Define your priorities for the matching algorithm. The algorithm will prioritize the most important.
25%
25%
25%
25%
General Information
Technical Skills
Soft Skills
Education Information
Go Back
Cancel
Next

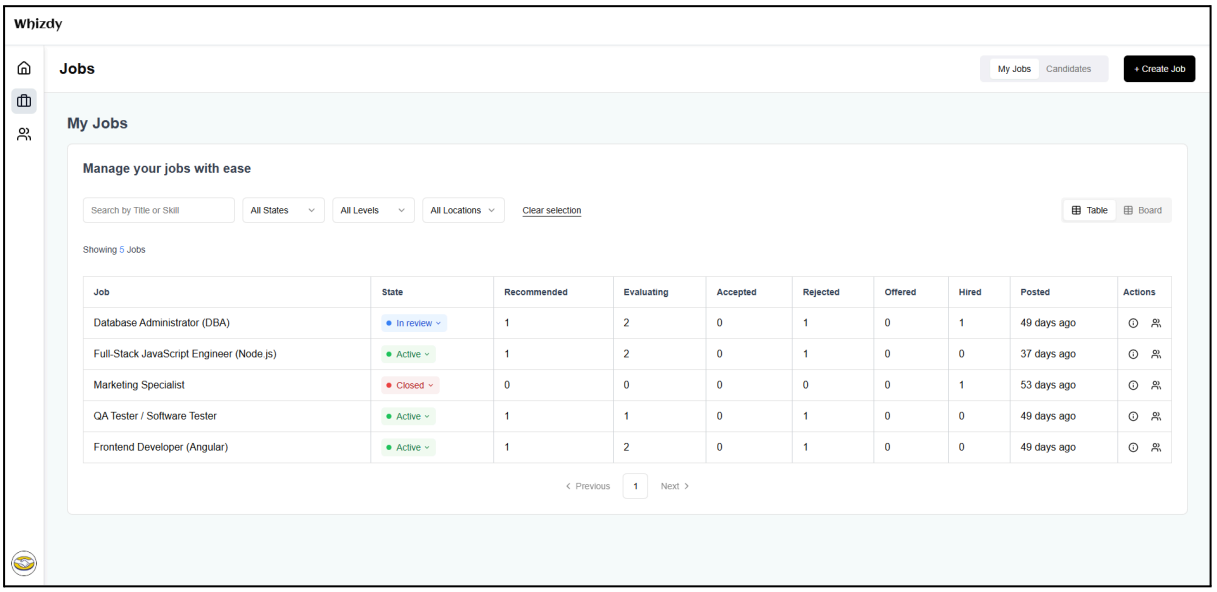

Whizdy
Jobs
My Jobs
Candidates
Outline the Role: Compensation
Get Started > Job Details > Skills > Compensation > You're Live!
Compensation
Base compensation*
0
Currency*
Select currency
Min Salary*
0
to
Max Salary*
0
Payment Frequency*
Select frequency
Software Engineer – FrontEnd (Next.js)
Mercado Libre Uruguay · Montevideo, Uruguay
FullTime | Remote | Junior | $$$$
Review AI Generated Description
You are responsible for your job post. Review to ensure it has all required information.
Go Back
Cancel
Next

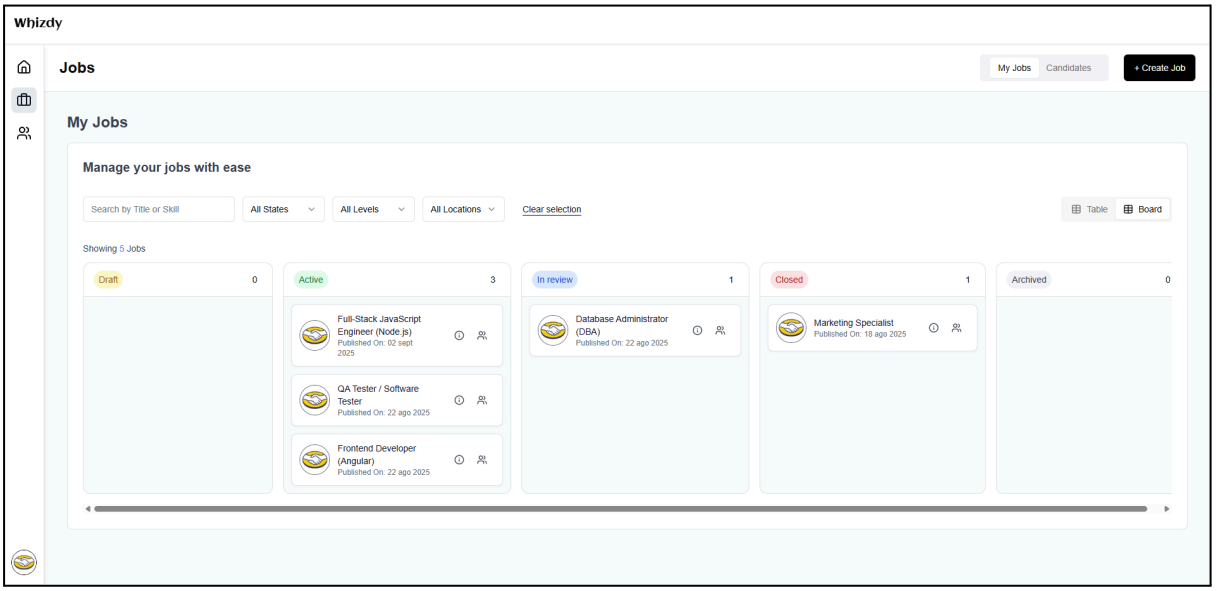
RF19 - Detalle de vacante con estadísticas:



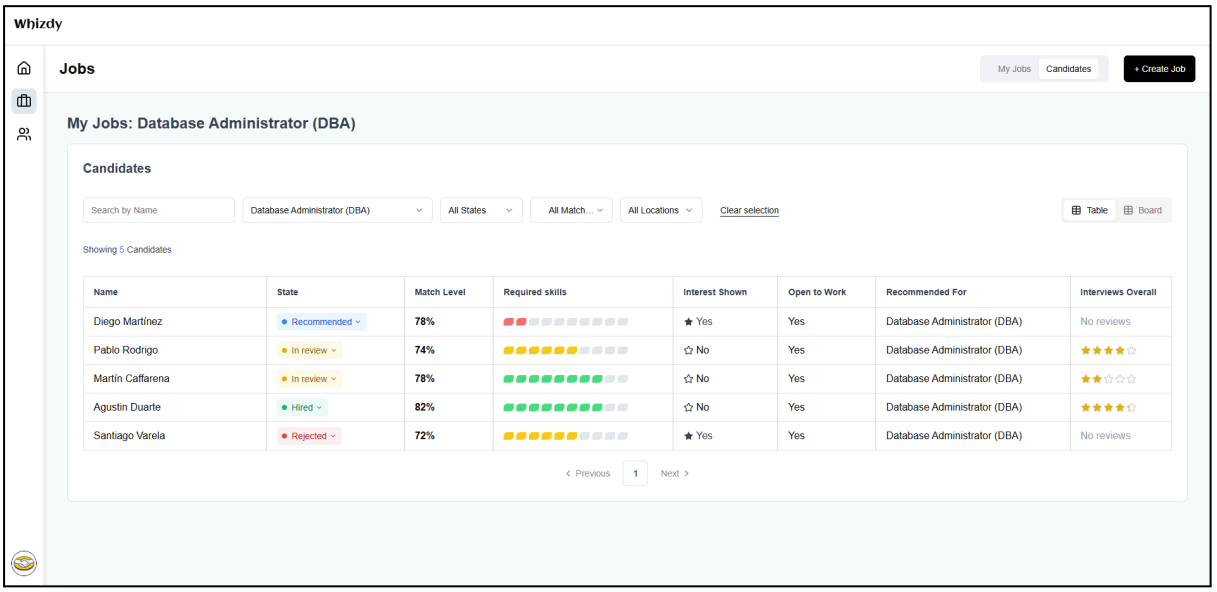


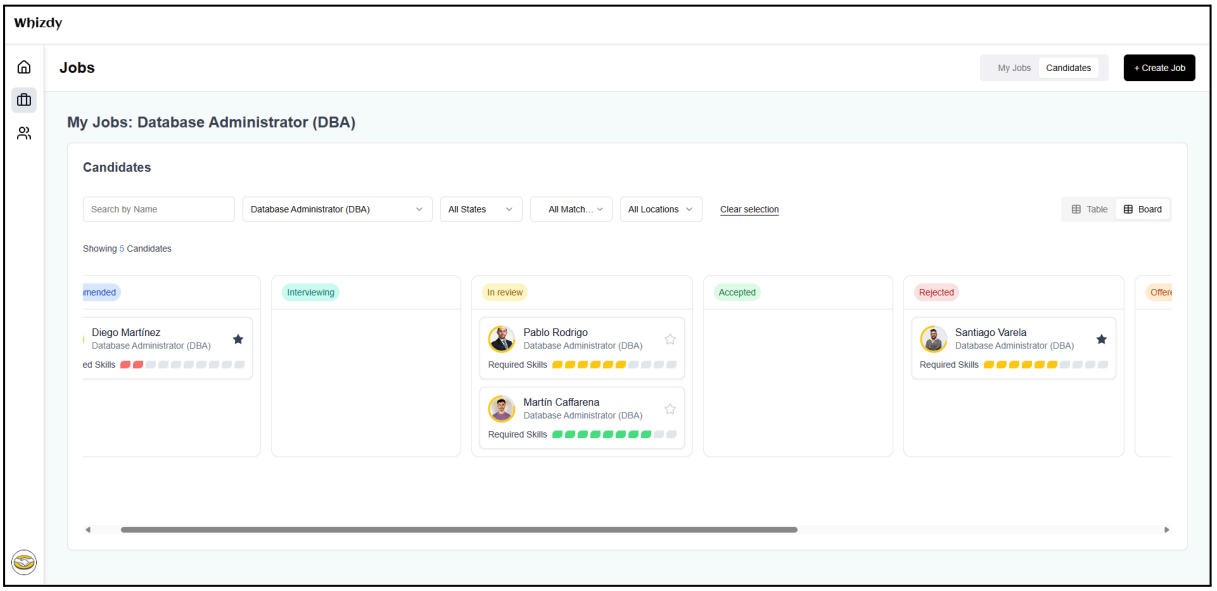
RF20 - Visualización y gestión de vacantes:



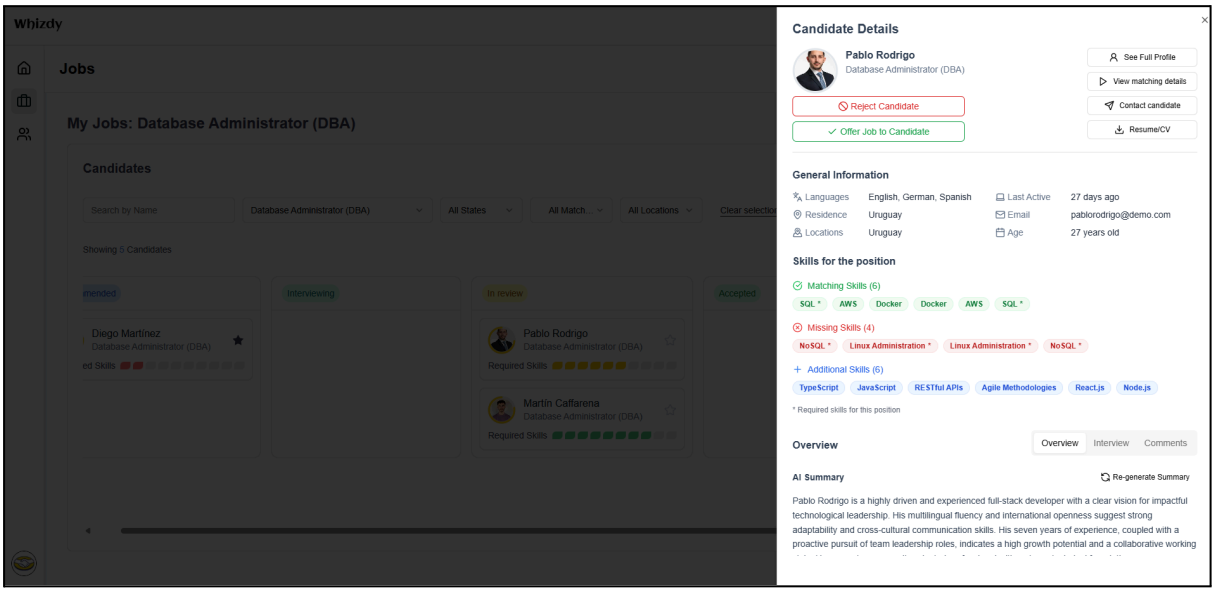


RF28 - Visualización de candidatos por vacante:

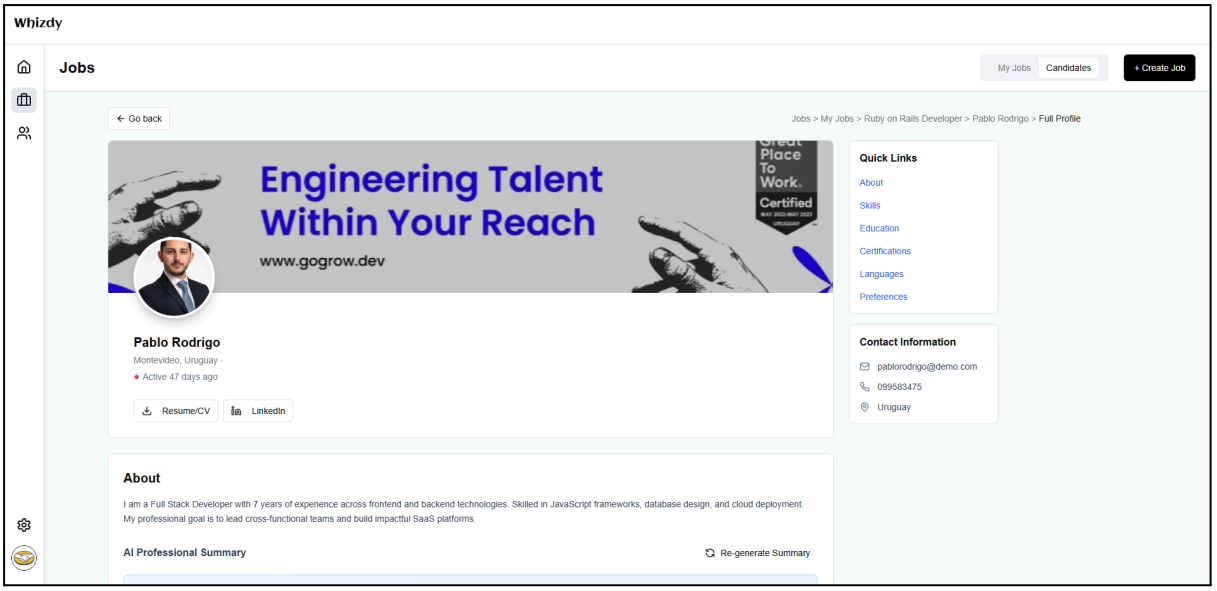


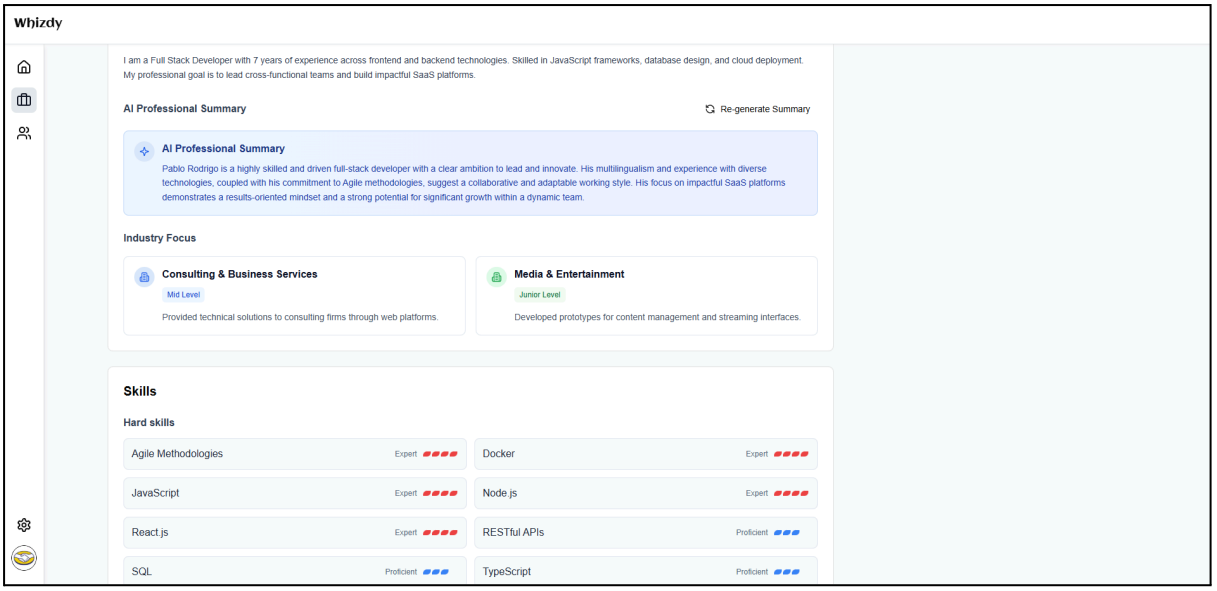


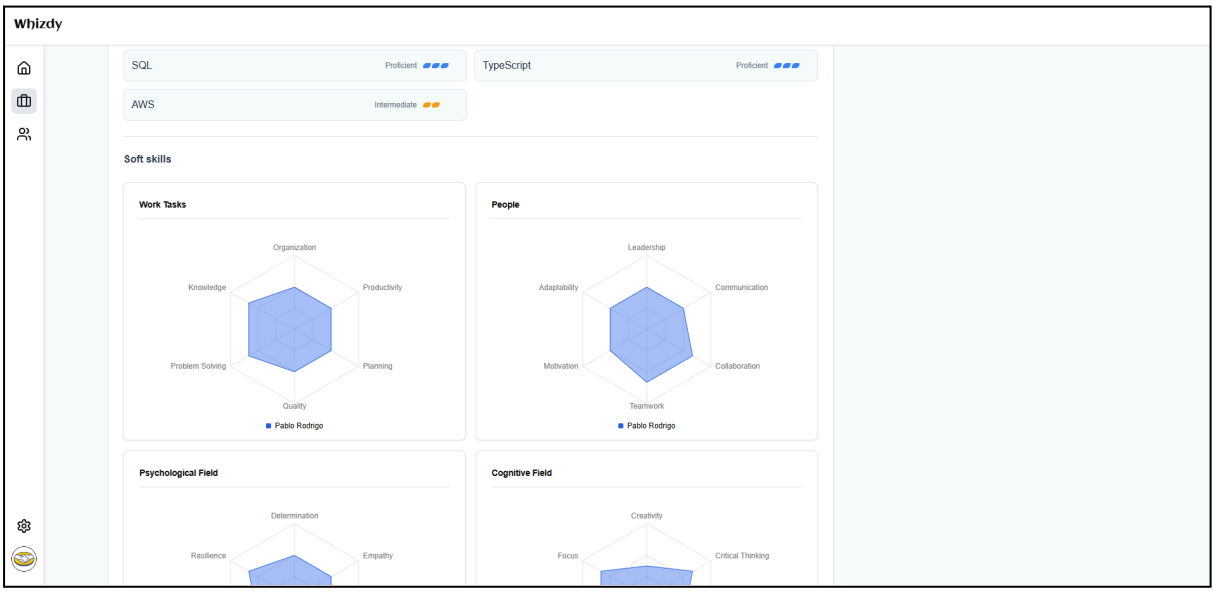
RF30 - Visualización básica de perfiles:

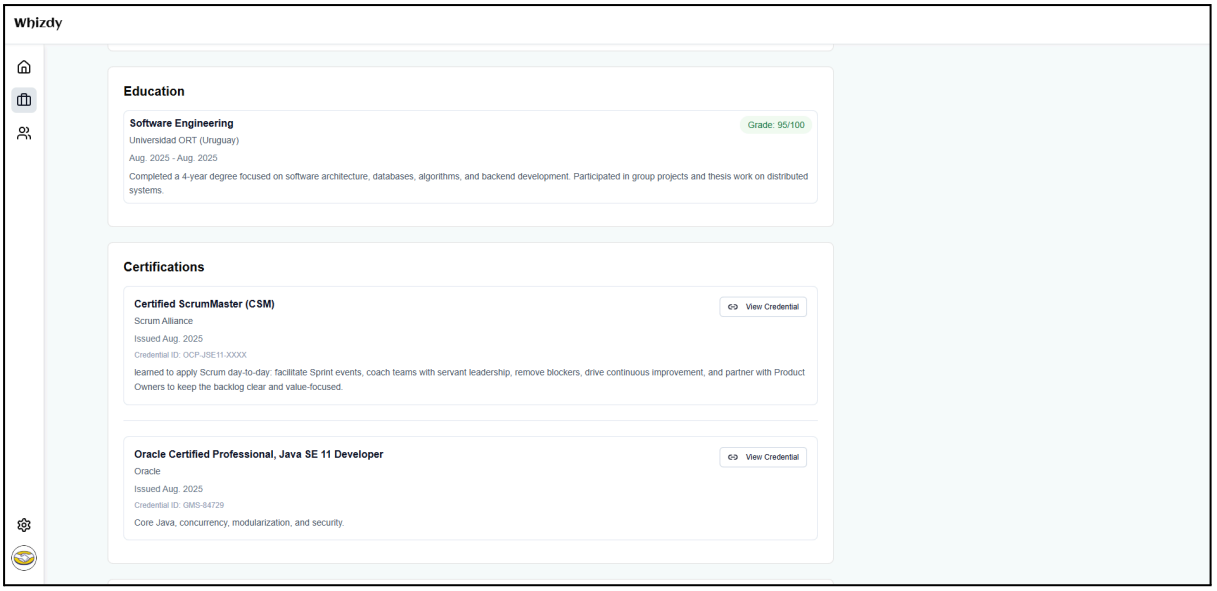


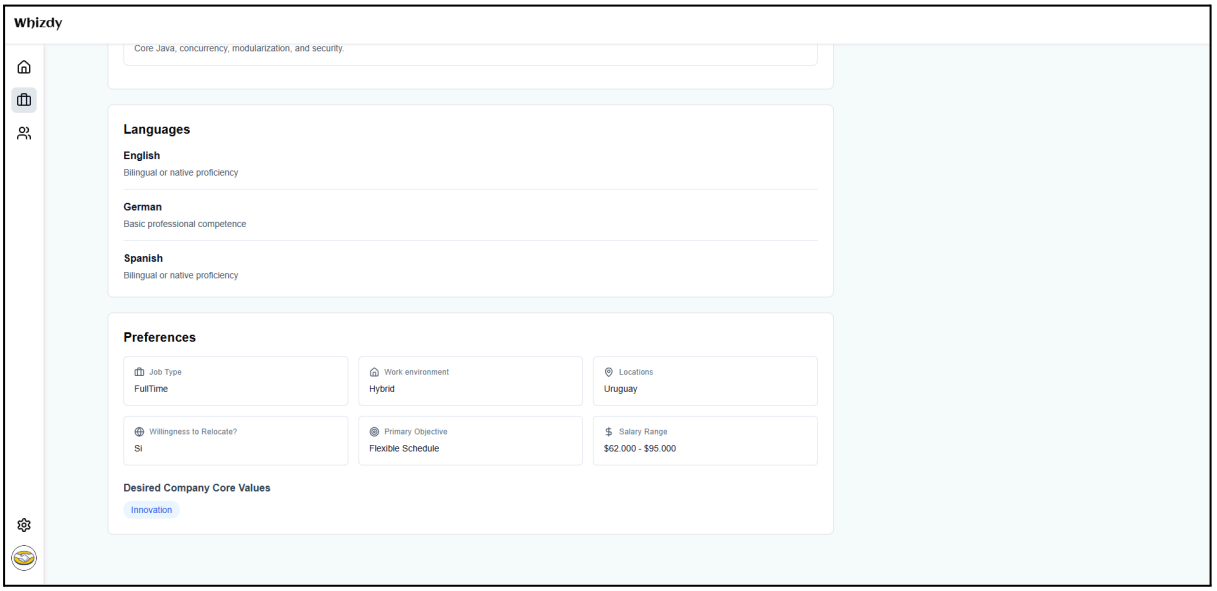
RF31 - Visualización completa de perfiles:



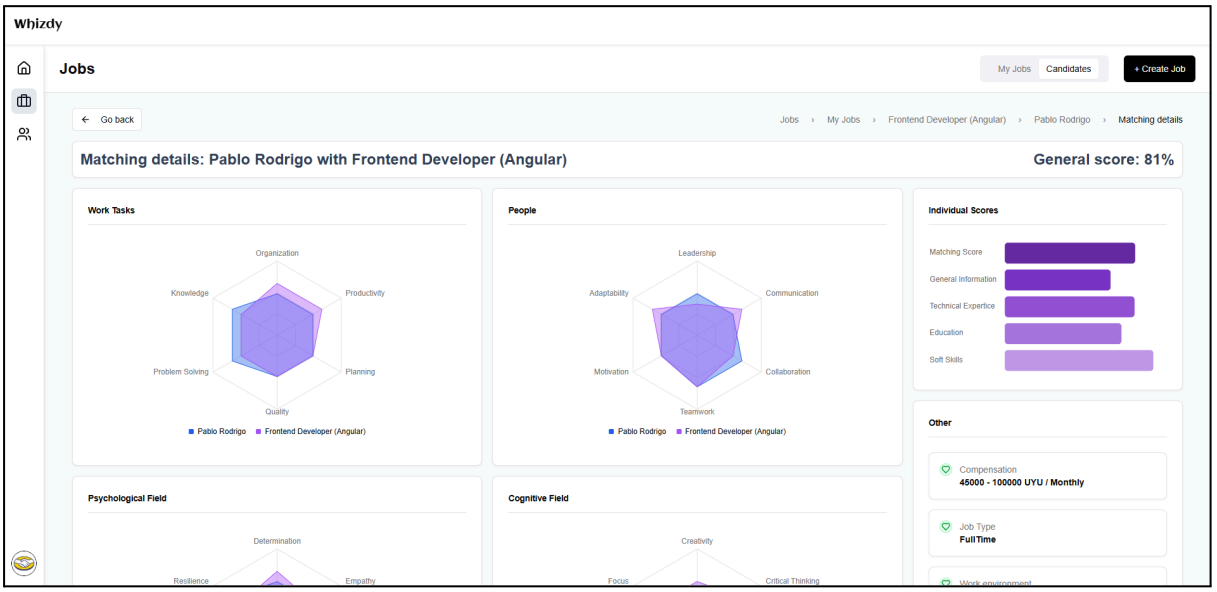

Whizdy
I am a Full Stack Developer with 7 years of experience across frontend and backend technologies. Skilled in JavaScript frameworks, database design, and cloud deployment. My professional goal is to lead cross-functional teams and build impactful SaaS platforms.
AI Professional Summary
Re-generate Summary
AI Professional Summary
Pablo Rodrigo is a highly skilled and driven full-stack developer with a clear ambition to lead and innovate. His multilingualism and experience with diverse technologies, coupled with his commitment to Agile methodologies, suggest a collaborative and adaptable working style. His focus on impactful SaaS platforms demonstrates a results-oriented mindset and a strong potential for significant growth within a dynamic team.
Industry Focus
Consulting & Business Services
Mid Level
Provided technical solutions to consulting firms through web platforms.
Media & Entertainment
Junior Level
Developed prototypes for content management and streaming interfaces.
Skills
Hard skills
Agile Methodologies Expert
Docker Expert
JavaScript Expert
Node.js Expert
React.js Expert
RESTful APIs Proficient
SQL Proficient
TypeScript Proficient


Whizdy
SQL Proficient
TypeScript Proficient
AWS Intermediate
Soft skills
Work Tasks
Organization
Productivity
Planning
Quality
Problem Solving
Knowledge
Pablo Rodrigo
People
Leadership
Communication
Collaboration
Teamwork
Motivation
Adaptability
Pablo Rodrigo
Psychological Field
Determination
Empathy
Resilience
Cognitive Field
Creativity
Critical Thinking
Focus

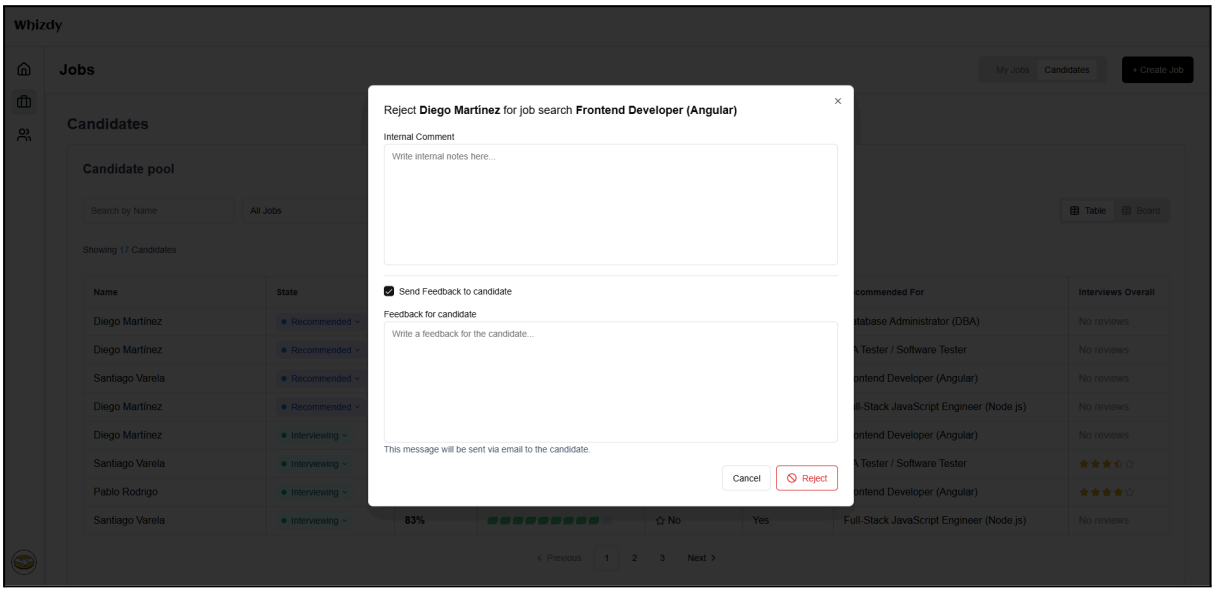

Whizdy
Education
Software Engineering
Grade: 95/100
Universidad ORT (Uruguay)
Aug. 2025 - Aug. 2025
Completed a 4-year degree focused on software architecture, databases, algorithms, and backend development. Participated in group projects and thesis work on distributed systems.
Certifications
Certified ScrumMaster (CSM)
View Credential
Scrum Alliance
Issued Aug. 2025
Credential ID: OCP-JSE11-XXXX
learned to apply Scrum day-to-day: facilitate Sprint events, coach teams with servant leadership, remove blockers, drive continuous improvement, and partner with Product Owners to keep the backlog clear and value-focused.
Oracle Certified Professional, Java SE 11 Developer
View Credential
Oracle
Issued Aug. 2025
Credential ID: GMS-84729
Core Java, concurrency, modularization, and security.

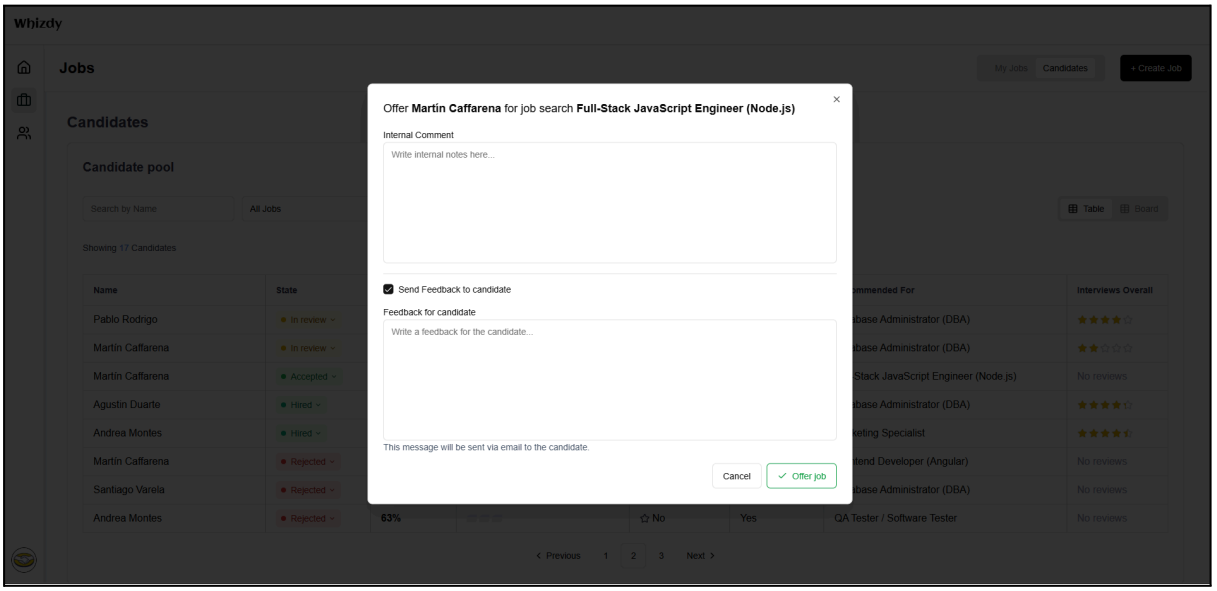

Whizdy
Core Java, concurrency, modularization, and security.
Languages
English
Bilingual or native proficiency
German
Basic professional competence
Spanish
Bilingual or native proficiency
Preferences
Job Type
FullTime
Work environment
Hybrid
Locations
Uruguay
Willingness to Relocate?
Si
Primary Objective
Flexible Schedule
Salary Range
$62.000 - $95.000
Desired Company Core Values
Innovation

RF35 - Resultados de *matching*:



RF38 - Cambiar estado de candidatos con *feedback* y notificaciones:





Update: Application for Fullstack Developer at Genexus was not successful

Recibidos ×

whizdy.noreply@gmail.com
para

Hi Martín,

You were considered for a potential opportunity at **GeneXus Consulting** for the role of **Fullstack Developer at Genexus**. After a detailed review, we regret to inform you that you were not selected to move forward at this time.

Here's some feedback from the company:

> *aaaa*

We encourage you to continue enhancing your profile and skills so we can find the best possible match for your next opportunity.

Stay proactive — you may be scouted for new positions in the future as your profile evolves.

Best regards,
The Whizdy Team

This email was sent to martincaffarena@gmail.com

You're being considered for Frontend Developer (Angular) at Mercado Libre Uruguay

Recibidos ×

whizdy.noreply@gmail.com
para

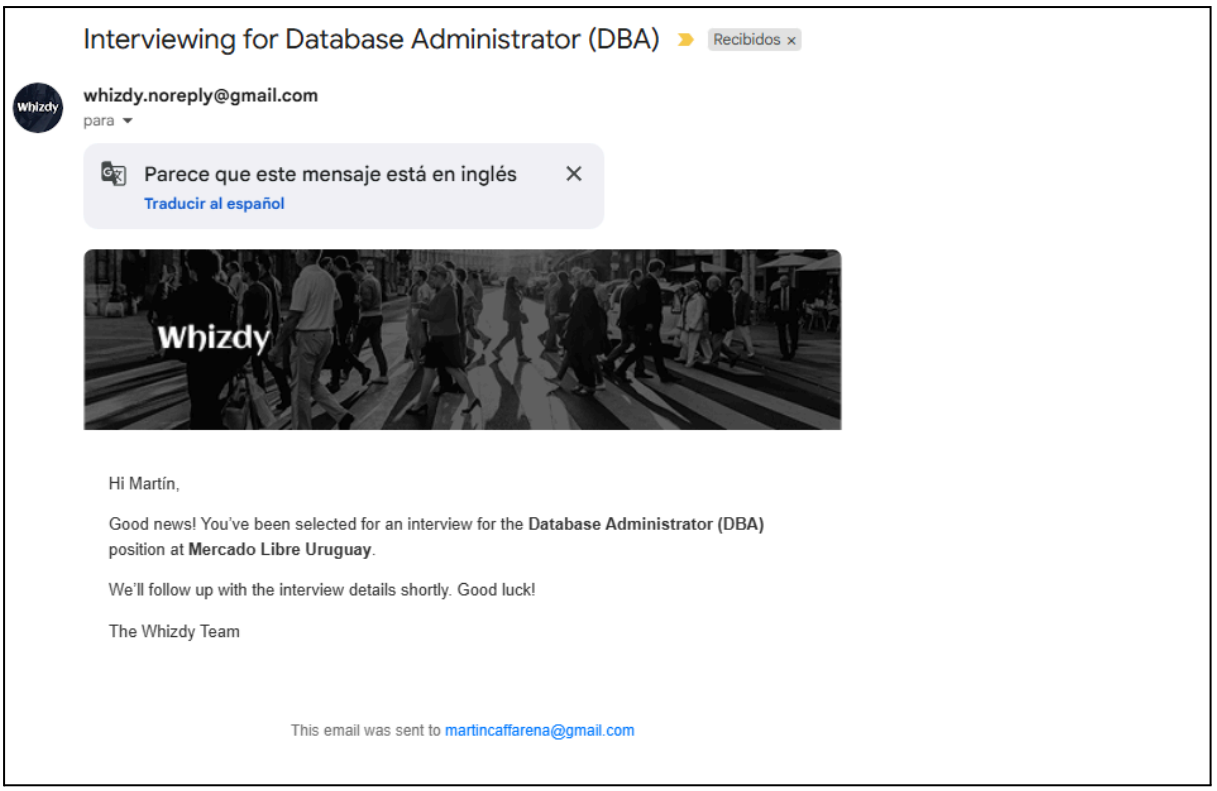
Hi Martín Caffarena,

Great news! Your profile is among the top candidates currently being considered for the position **Frontend Developer (Angular)** at **Mercado Libre Uruguay**.

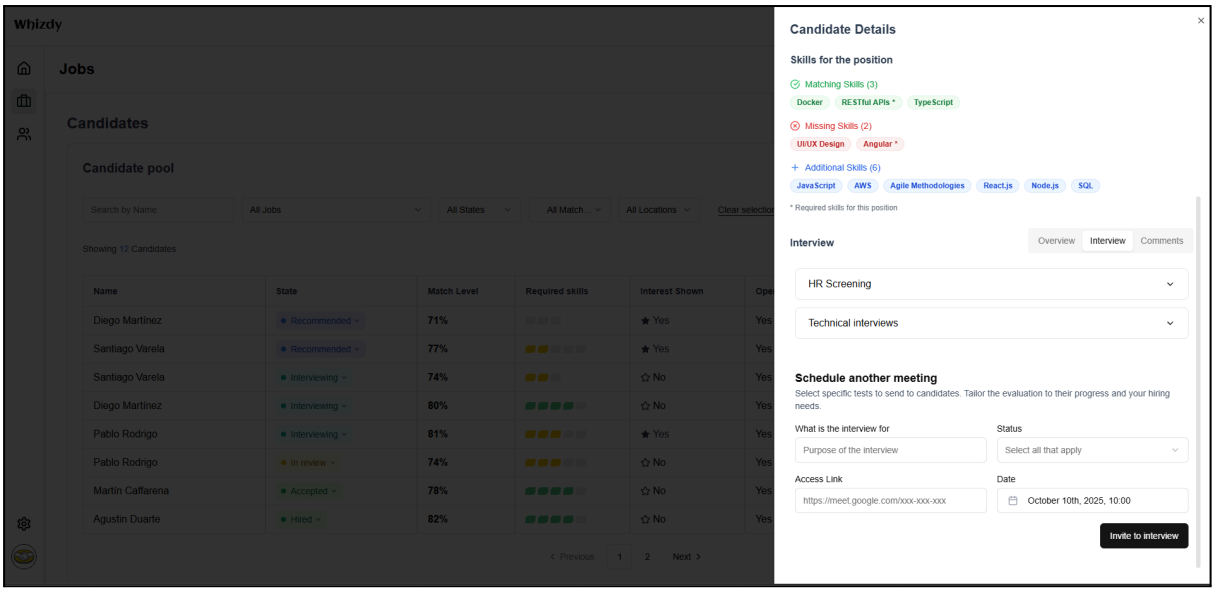
We'll keep you updated on the next steps. Best of luck!

The Whizdy Team

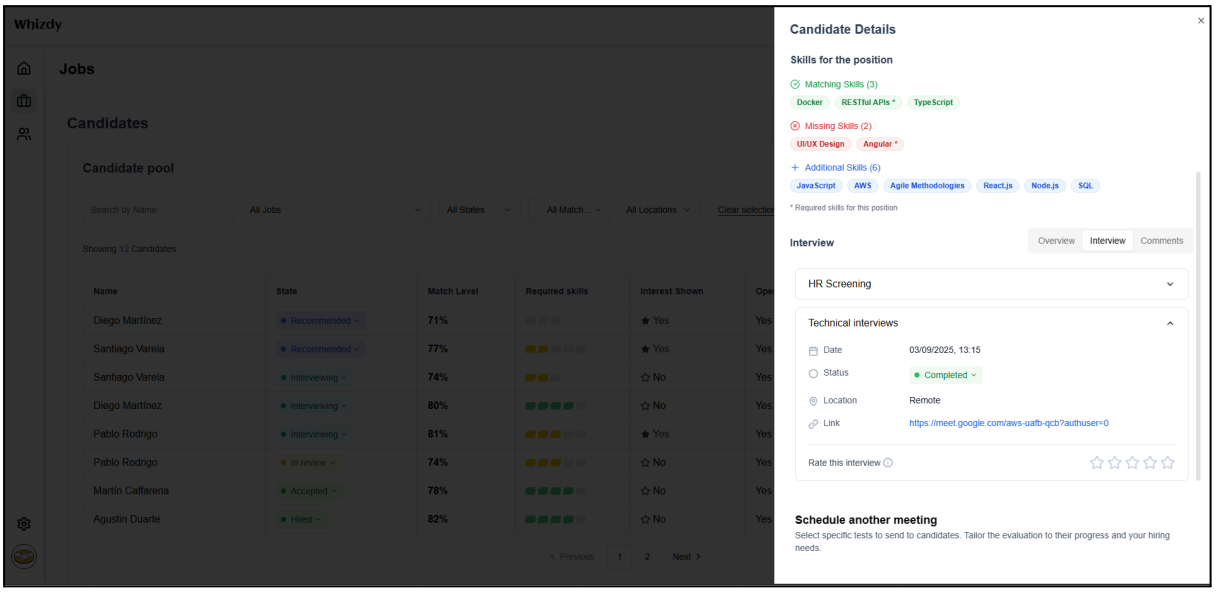
This email was sent to martincaffarena@gmail.com

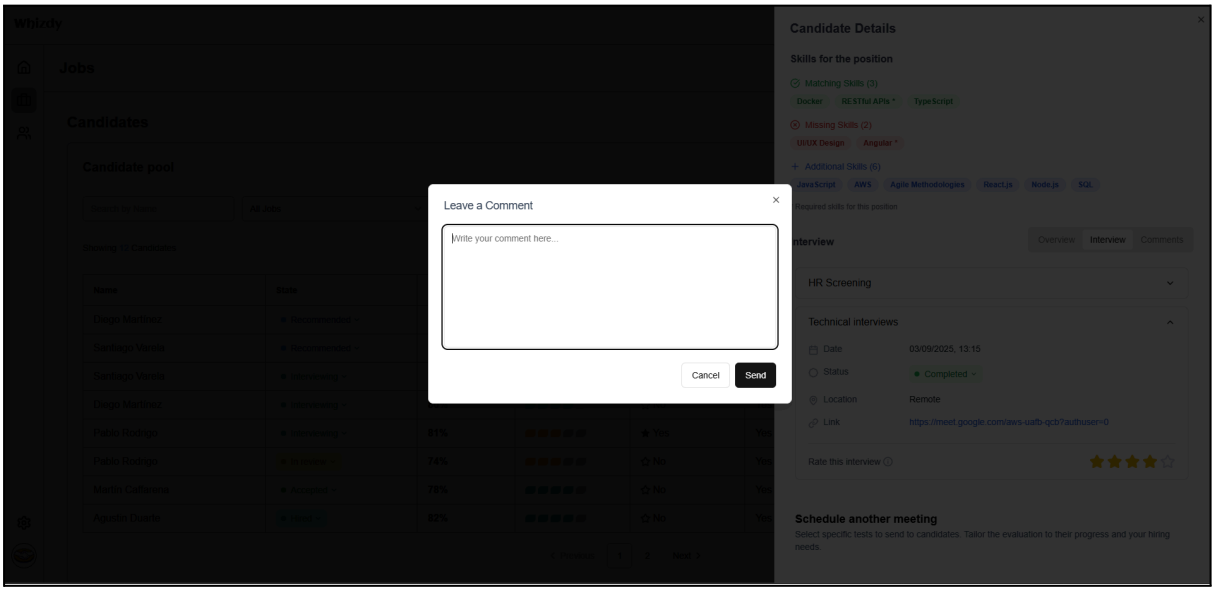


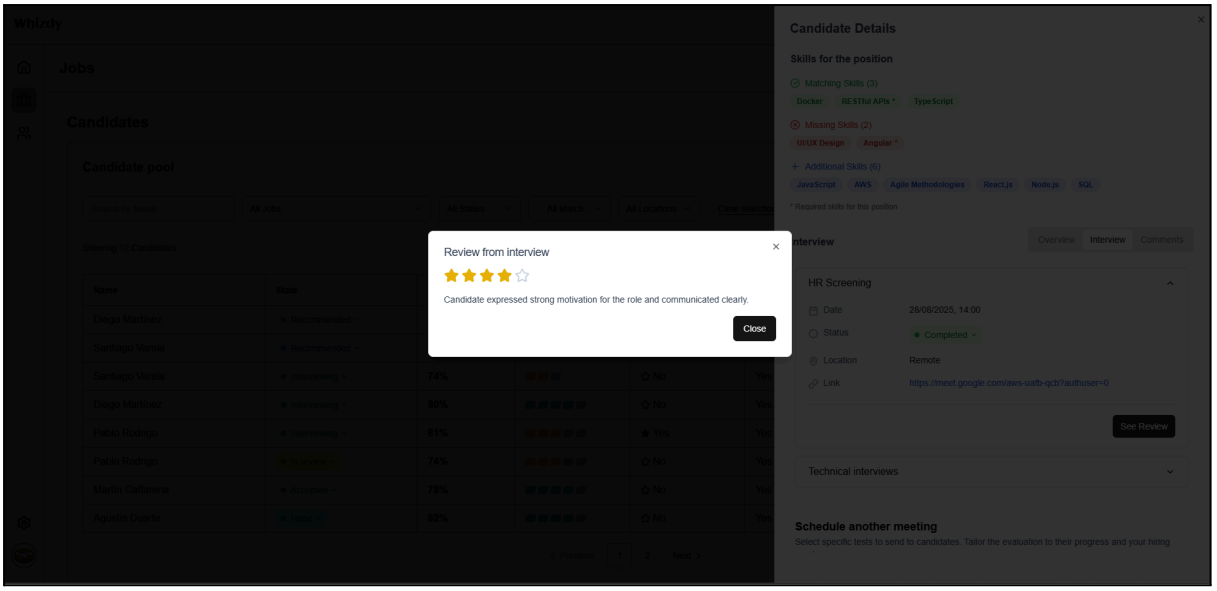
RF40 - Agendar entrevistas:



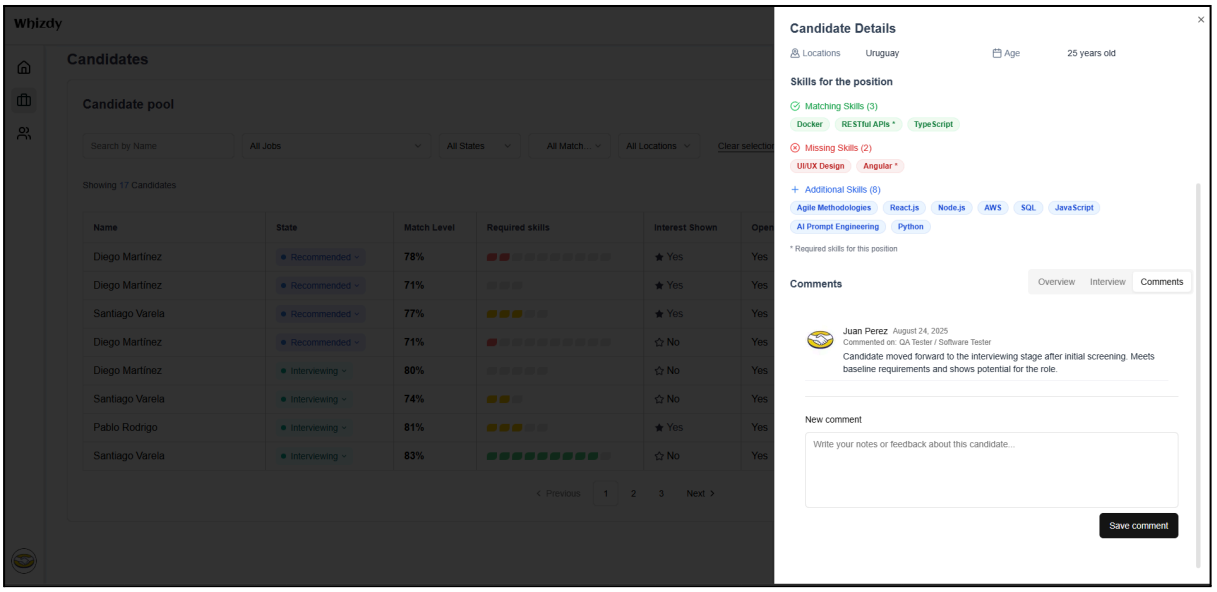
RF41 - Reseñar entrevistas:



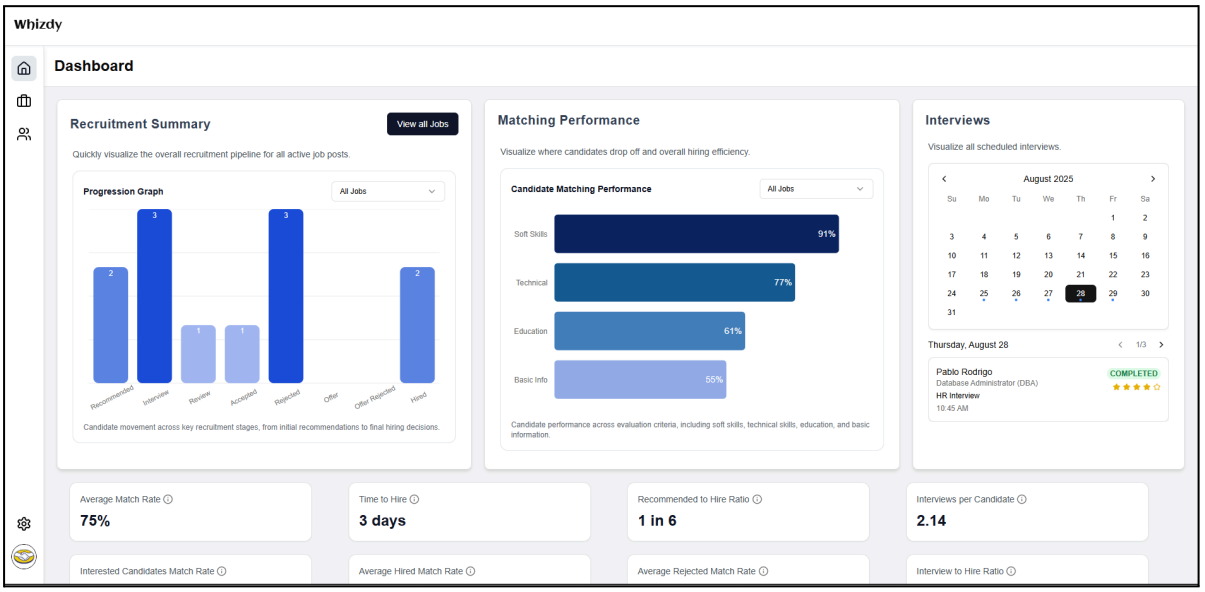


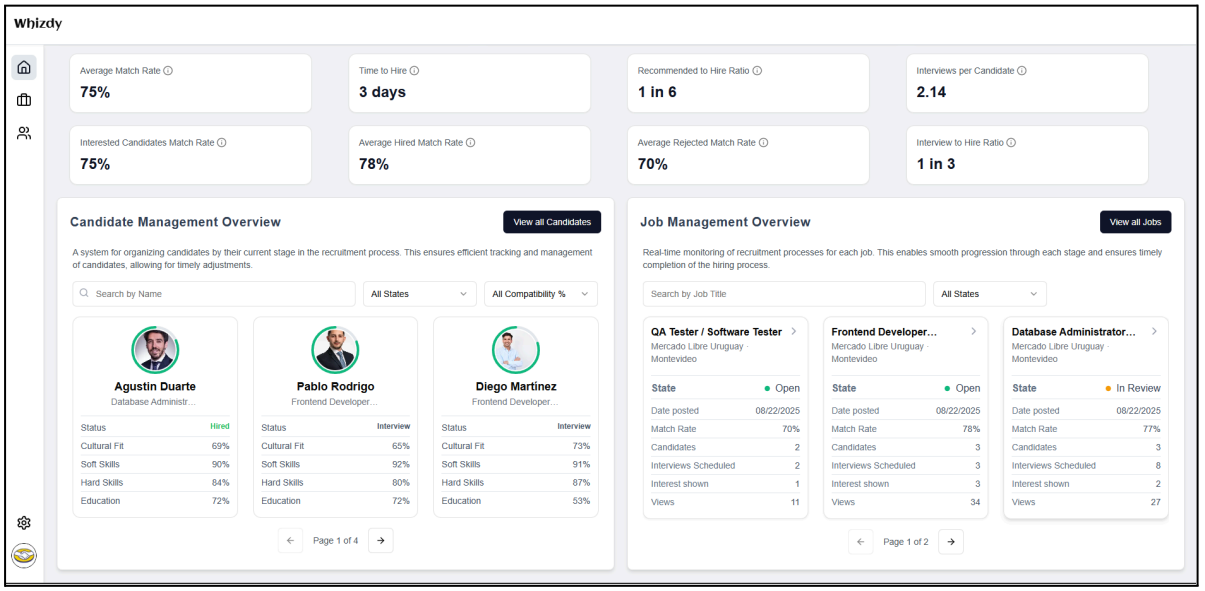


RF42 - Comentarios por vacante:

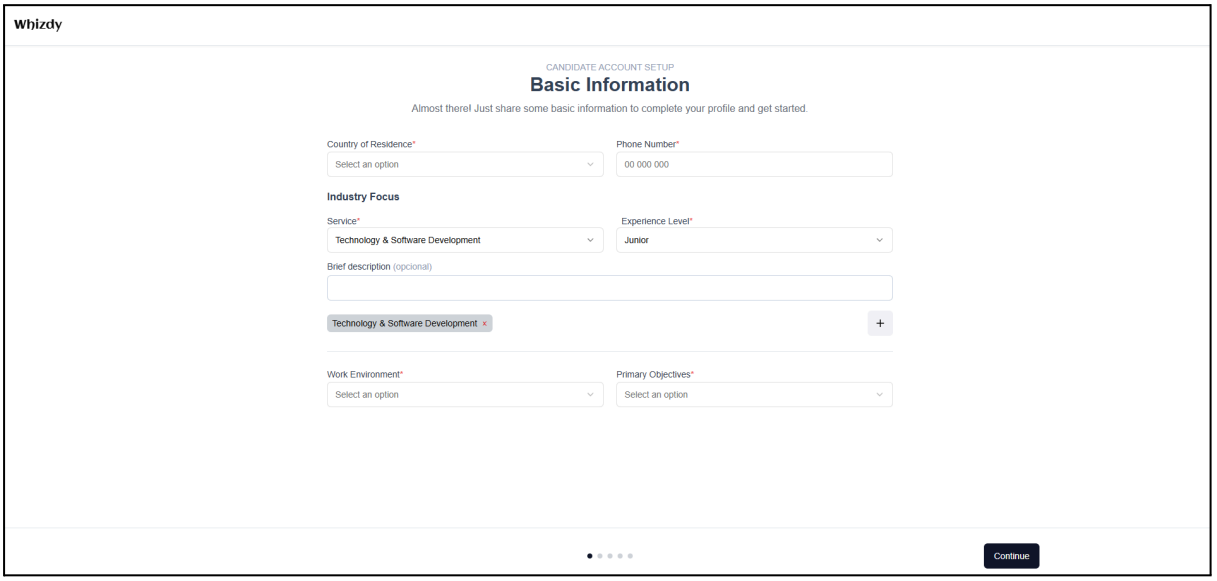


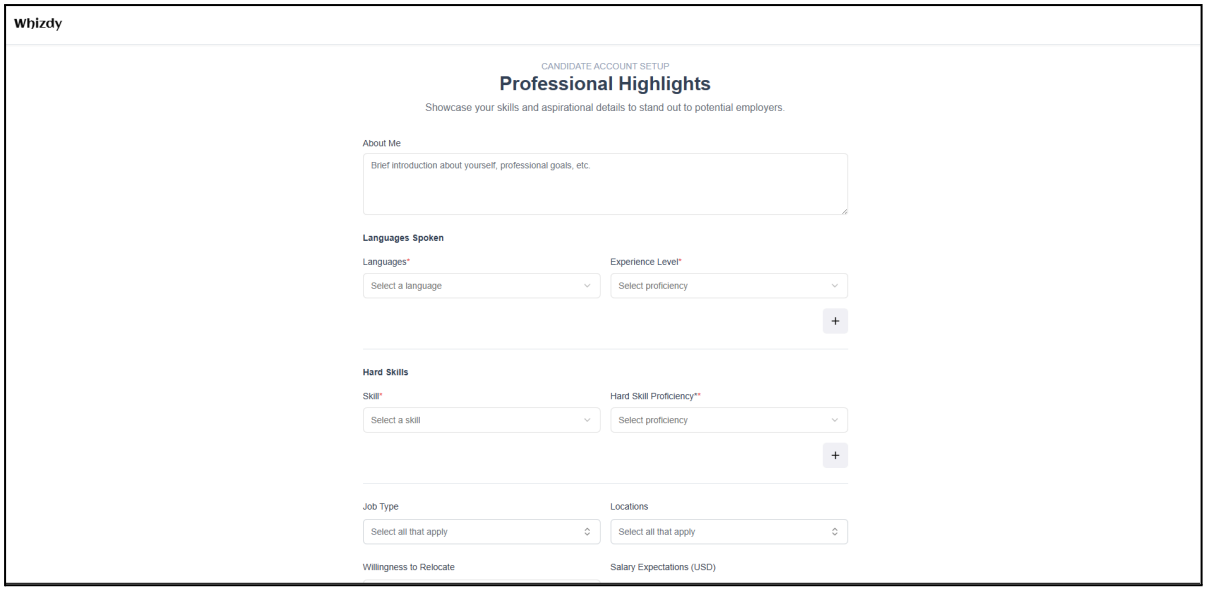
RF49 - Dashboard de empresa:

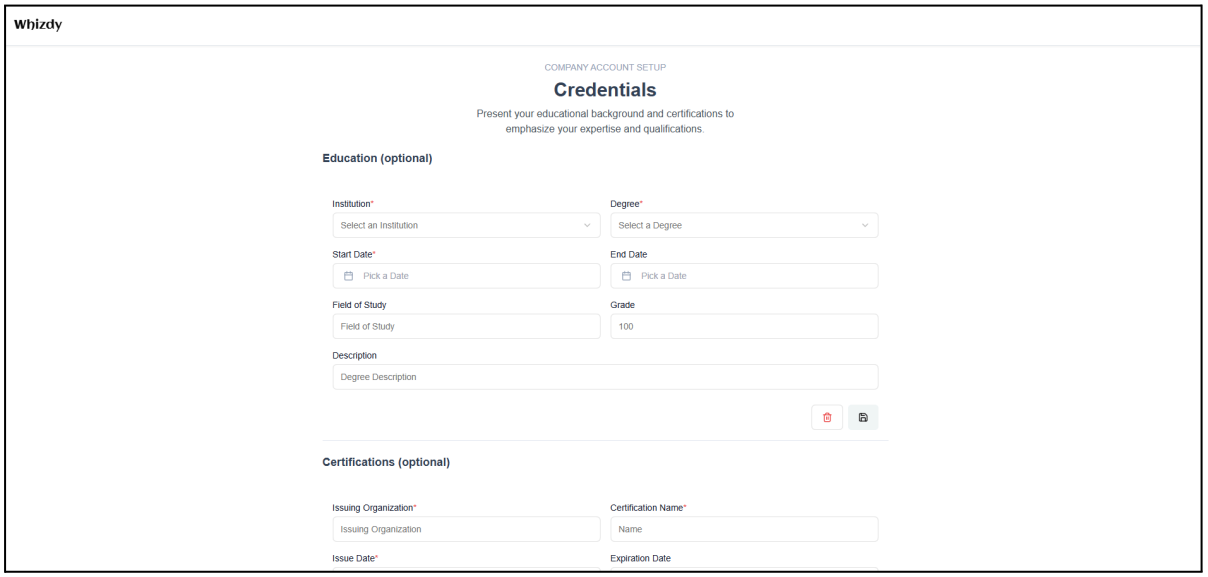


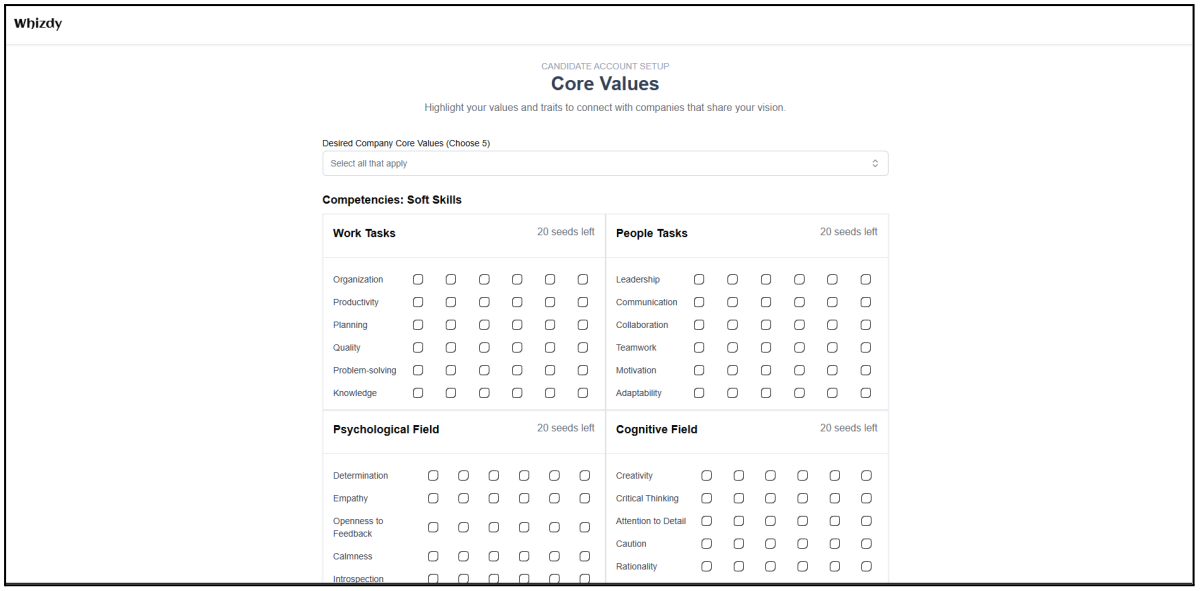


RF56 - Onboarding y perfil profesional:

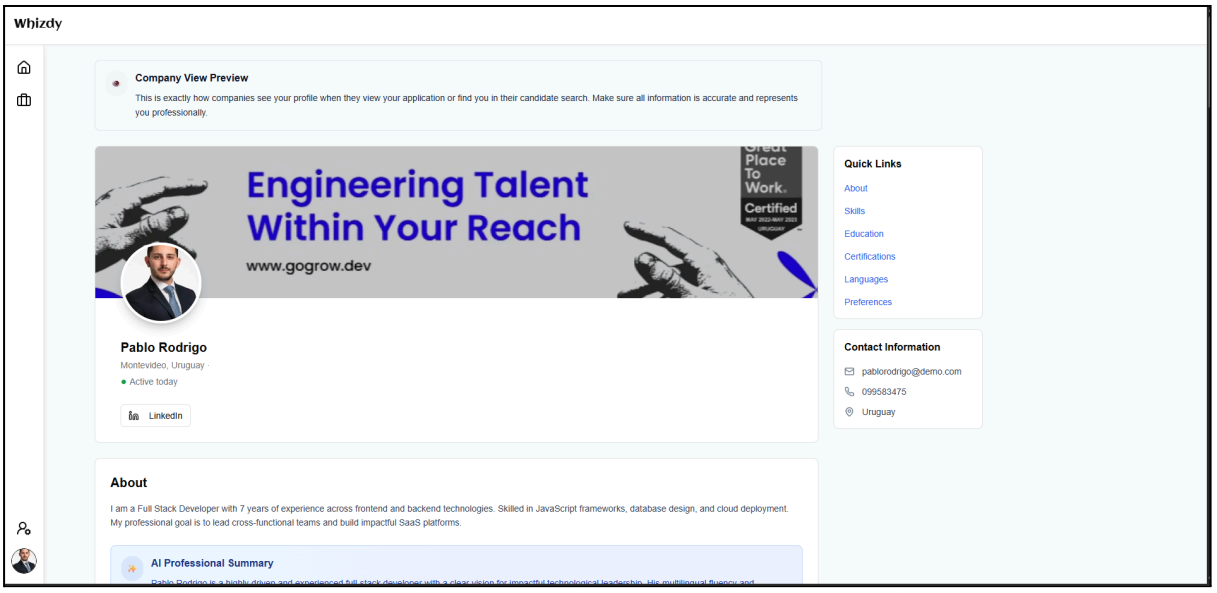


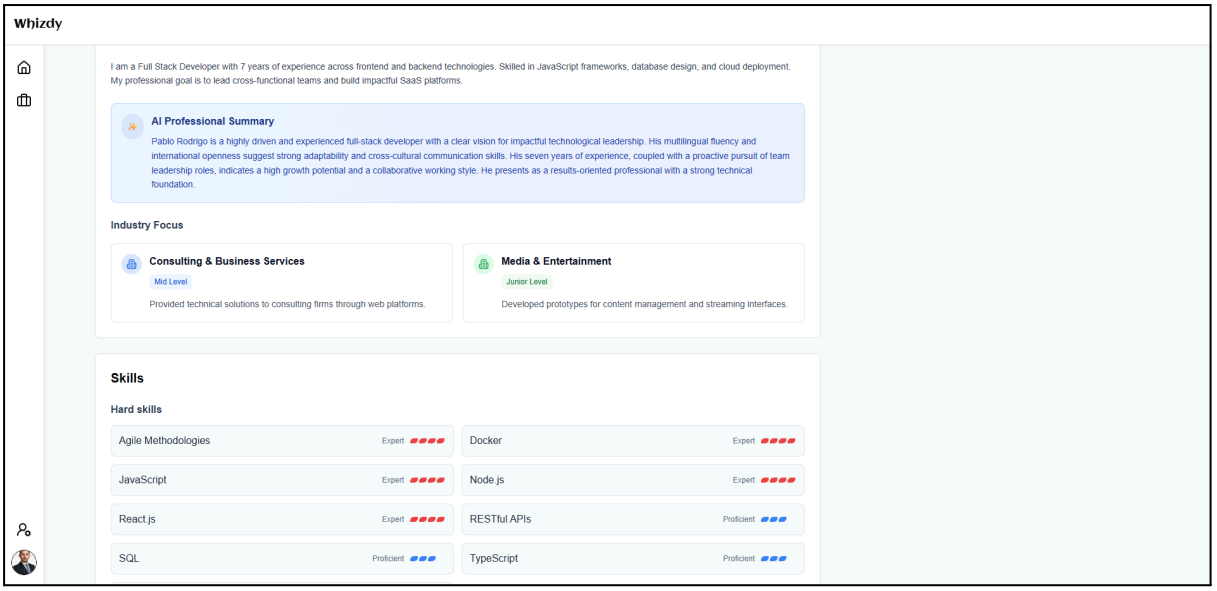

Whizdy
CANDIDATE ACCOUNT SETUP
Professional Highlights
Showcase your skills and aspirational details to stand out to potential employers.
About Me
Brief introduction about yourself, professional goals, etc.
Languages Spoken
Languages*
Select a language
Experience Level*
Select proficiency
Hard Skills
Skill*
Select a skill
Hard Skill Proficiency**
Select proficiency
Job Type
Select all that apply
Locations
Select all that apply
Willingness to Relocate
Salary Expectations (USD)

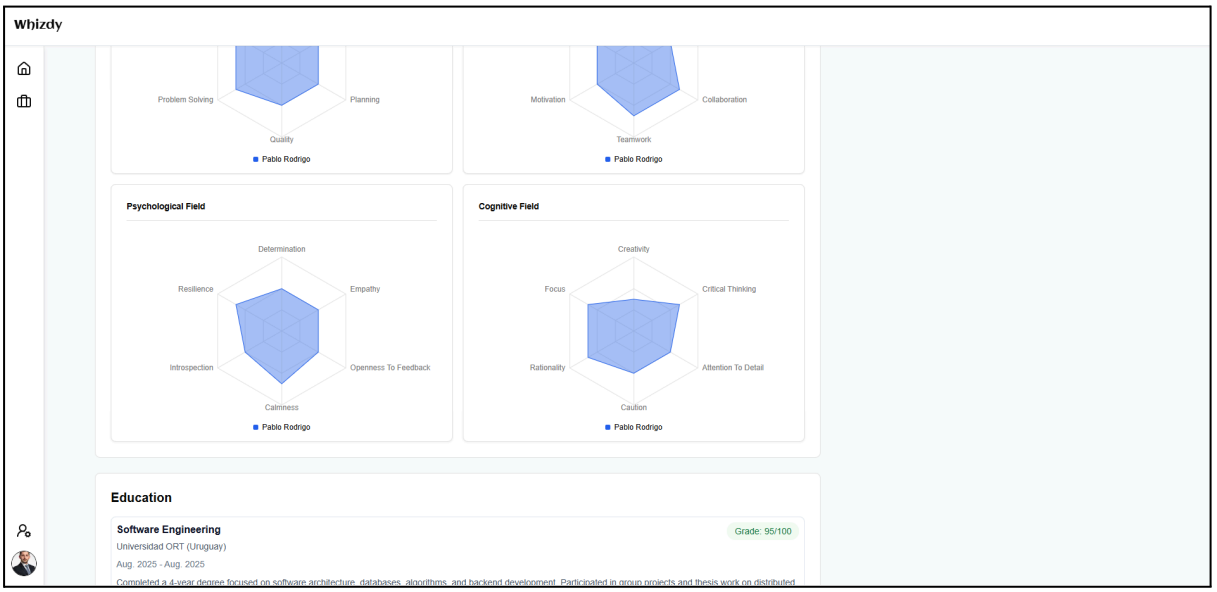

Whizdy
COMPANY ACCOUNT SETUP
Credentials
Present your educational background and certifications to emphasize your expertise and qualifications.
Education (optional)
Institution*
Select an Institution
Degree*
Select a Degree
Start Date*
Pick a Date
End Date
Pick a Date
Field of Study
Field of Study
Grade
100
Description
Degree Description
Certifications (optional)
Issuing Organization*
Issuing Organization
Certification Name*
Name
Issue Date*
Expiration Date

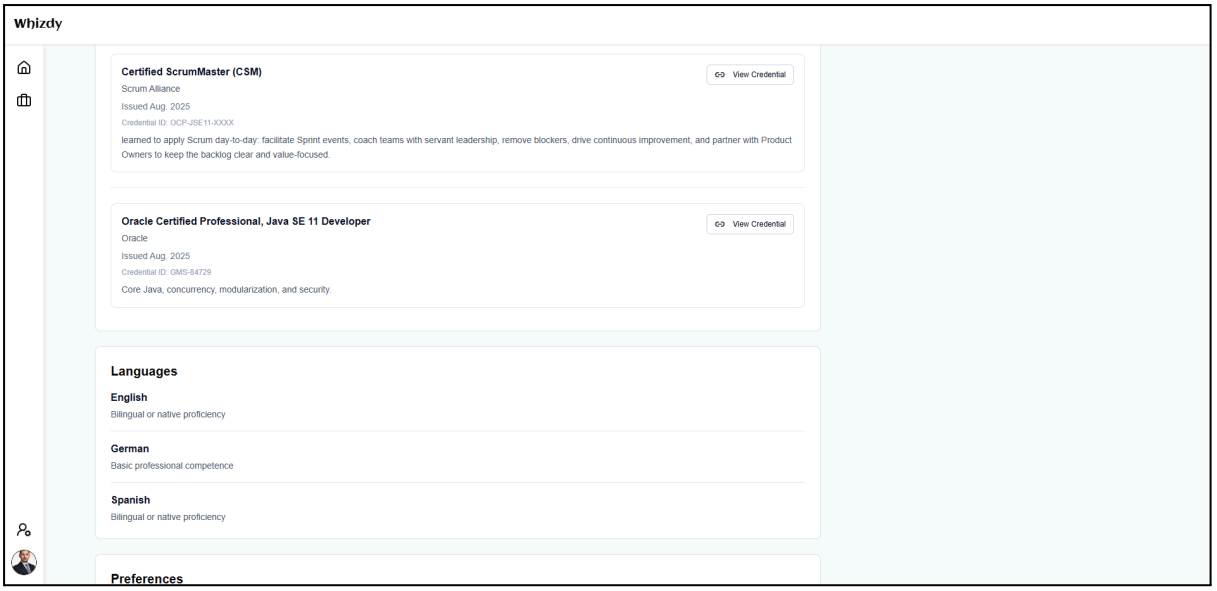


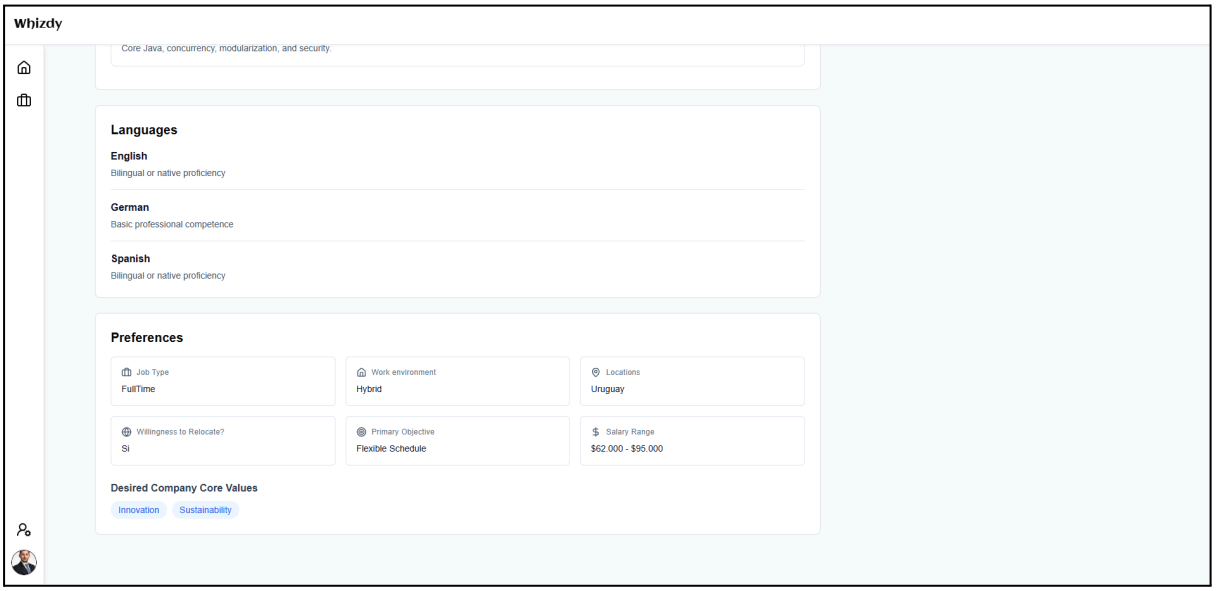
RF58 - Visualización de perfil:



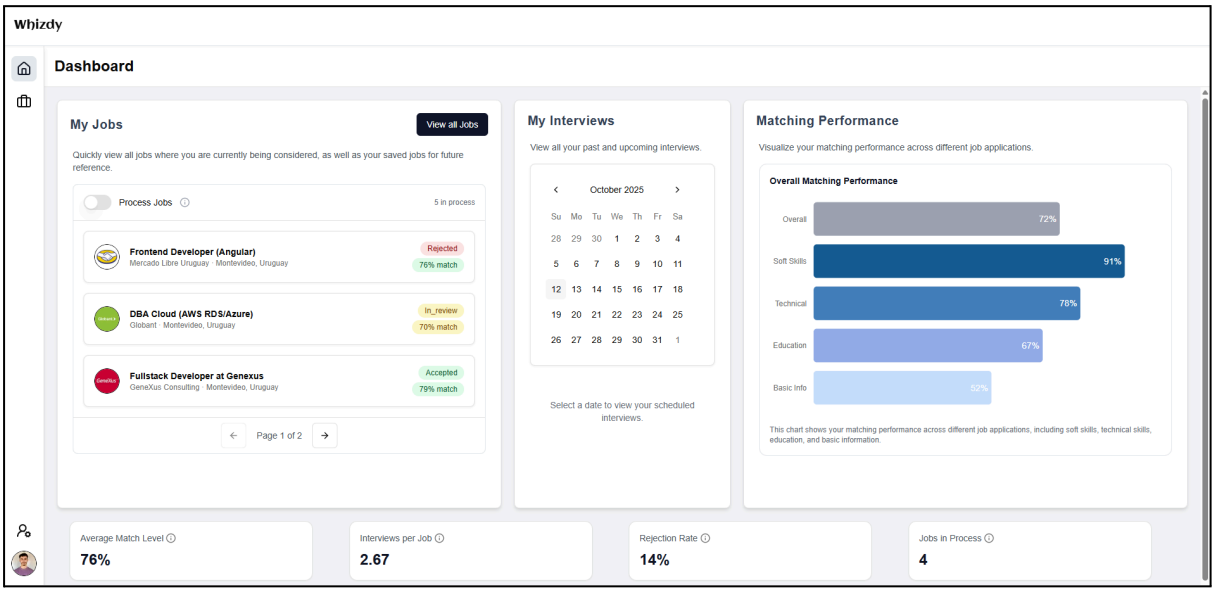

Whizdy
I am a Full Stack Developer with 7 years of experience across frontend and backend technologies. Skilled in JavaScript frameworks, database design, and cloud deployment. My professional goal is to lead cross-functional teams and build impactful SaaS platforms.
AI Professional Summary
Pablo Rodrigo is a highly driven and experienced full-stack developer with a clear vision for impactful technological leadership. His multilingual fluency and international openness suggest strong adaptability and cross-cultural communication skills. His seven years of experience, coupled with a proactive pursuit of team leadership roles, indicates a high growth potential and a collaborative working style. He presents as a results-oriented professional with a strong technical foundation.
Industry Focus
Consulting & Business Services
Mid Level
Provided technical solutions to consulting firms through web platforms.
Media & Entertainment
Junior Level
Developed prototypes for content management and streaming interfaces.
Skills
Hard skills
Agile Methodologies Expert
Docker Expert
JavaScript Expert
Node.js Expert
React.js Expert
RESTful APIs Proficient
SQL Proficient
TypeScript Proficient

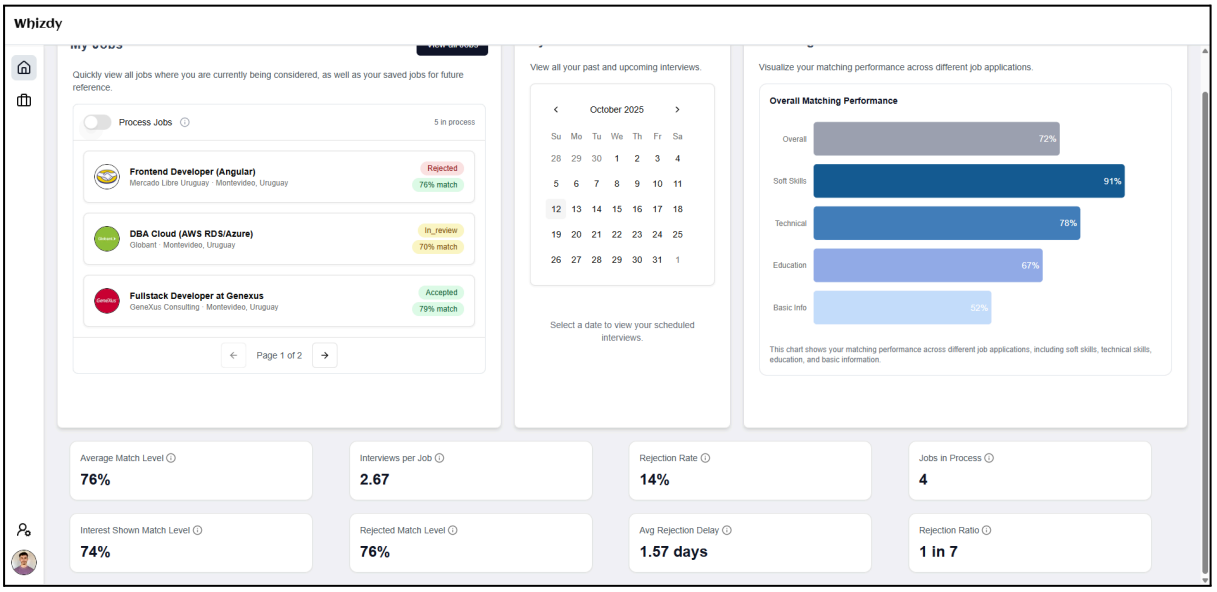

Whizdy
Problem Solving
Planning
Quality
Pablo Rodrigo
Motivation
Collaboration
Teamwork
Pablo Rodrigo
Psychological Field
Determination
Resilience
Empathy
Introspection
Openness To Feedback
Calmness
Pablo Rodrigo
Cognitive Field
Creativity
Focus
Critical Thinking
Rationality
Attention To Detail
Caution
Pablo Rodrigo
Education
Software Engineering
Grade: 95/100
Universidad ORT (Uruguay)
Aug. 2025 - Aug. 2025
Completed a 4-year degree focused on software architecture, databases, algorithms, and backend development. Participated in group projects and thesis work on distributed


Whizdy
Certified ScrumMaster (CSM)
View Credential
Scrum Alliance
Issued Aug. 2025
Credential ID: OCP-JSE11-XXXX
learned to apply Scrum day-to-day: facilitate Sprint events, coach teams with servant leadership, remove blockers, drive continuous improvement, and partner with Product Owners to keep the backlog clear and value-focused.
Oracle Certified Professional, Java SE 11 Developer
View Credential
Oracle
Issued Aug. 2025
Credential ID: GMS-84729
Core Java, concurrency, modularization, and security.
Languages
English
Bilingual or native proficiency
German
Basic professional competence
Spanish
Bilingual or native proficiency
Preferences

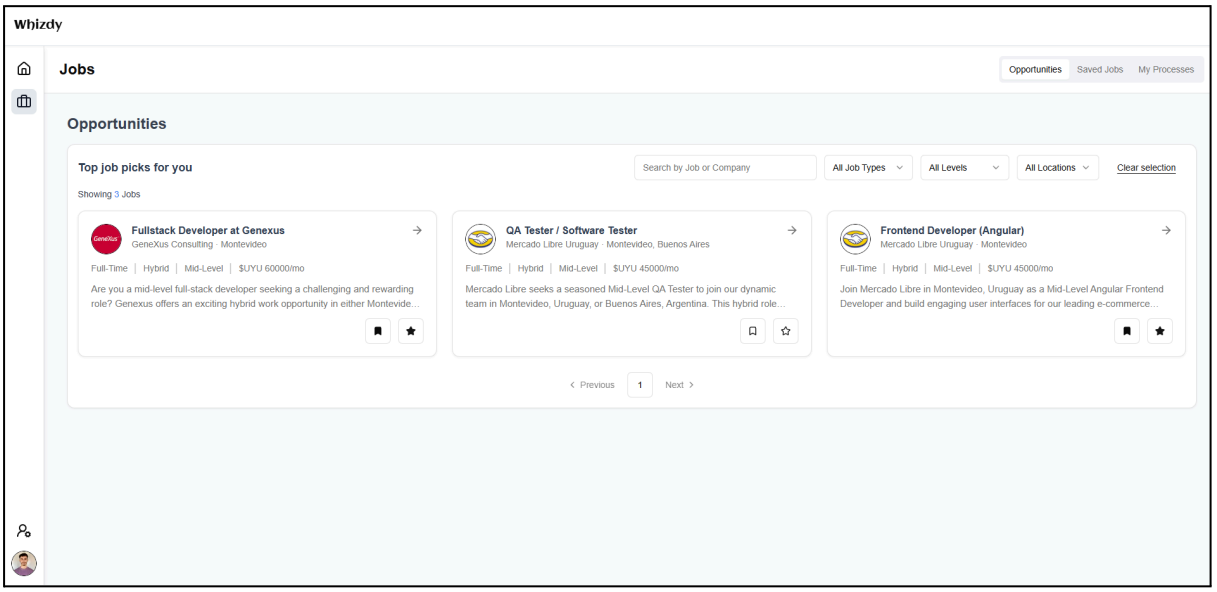

Whizdy
Core Java, concurrency, modularization, and security.
Languages
English
Bilingual or native proficiency
German
Basic professional competence
Spanish
Bilingual or native proficiency
Preferences
Job Type
FullTime
Work environment
Hybrid
Locations
Uruguay
Willingness to Relocate?
Si
Primary Objective
Flexible Schedule
Salary Range
$62.000 - $95.000
Desired Company Core Values
Innovation
Sustainability

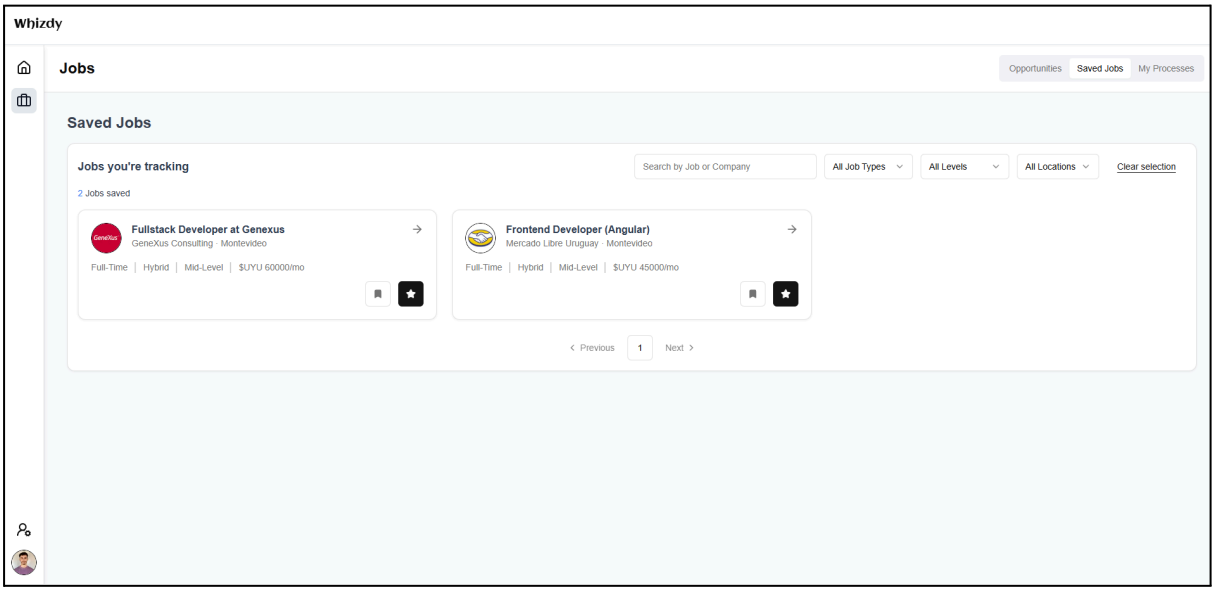
RF59 - Tablero del candidato:

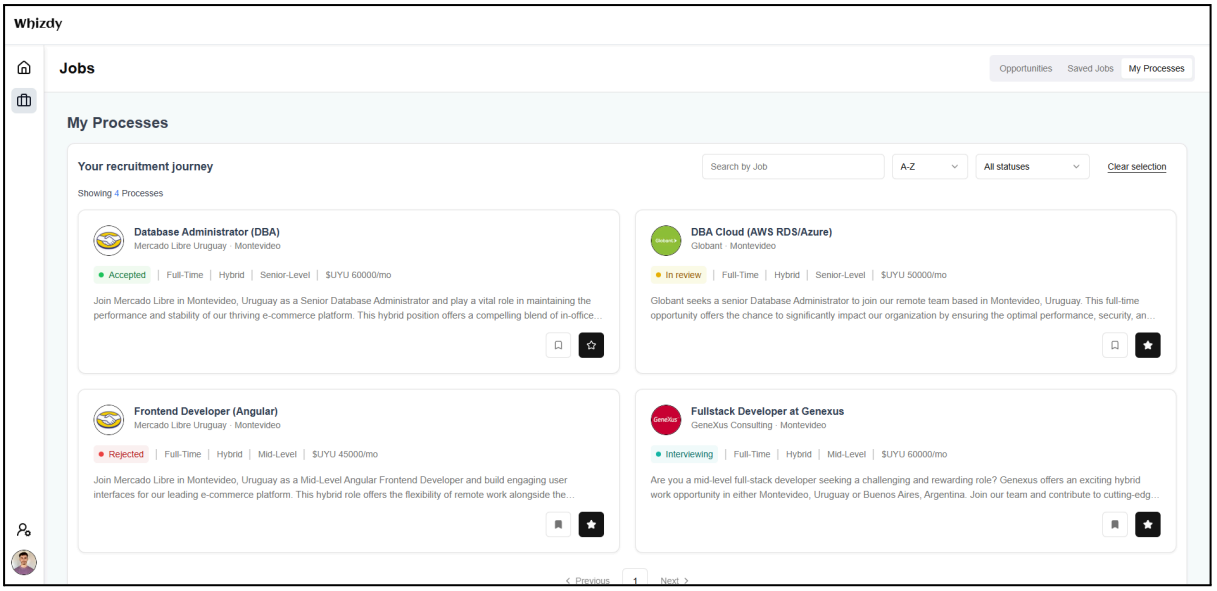




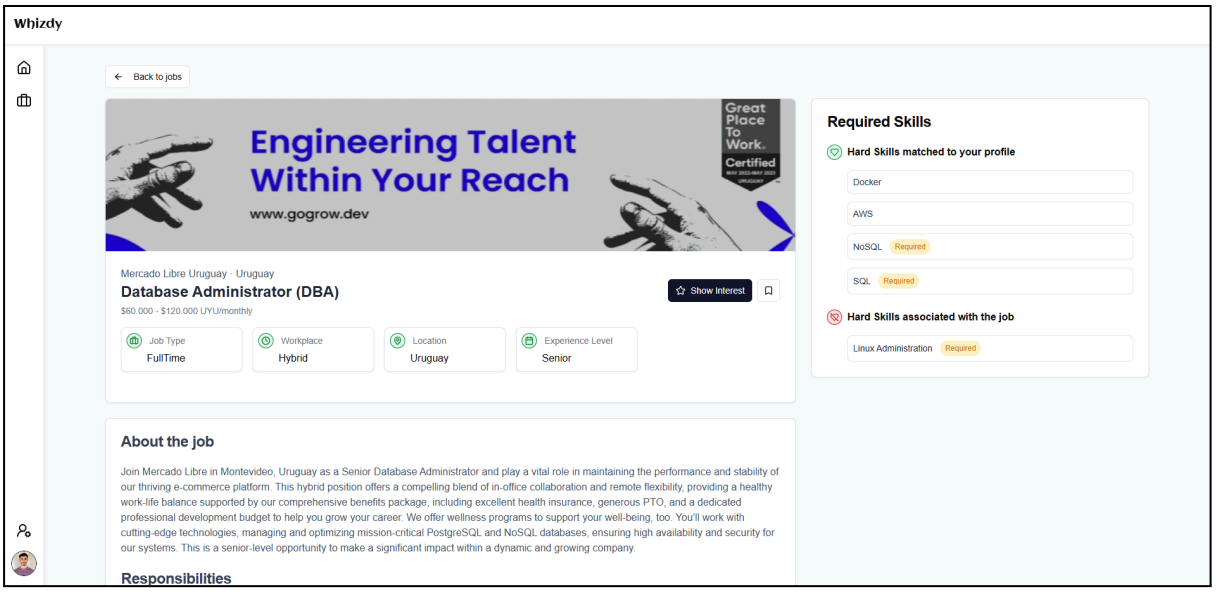
RF60 - Oportunidades y procesos activos:

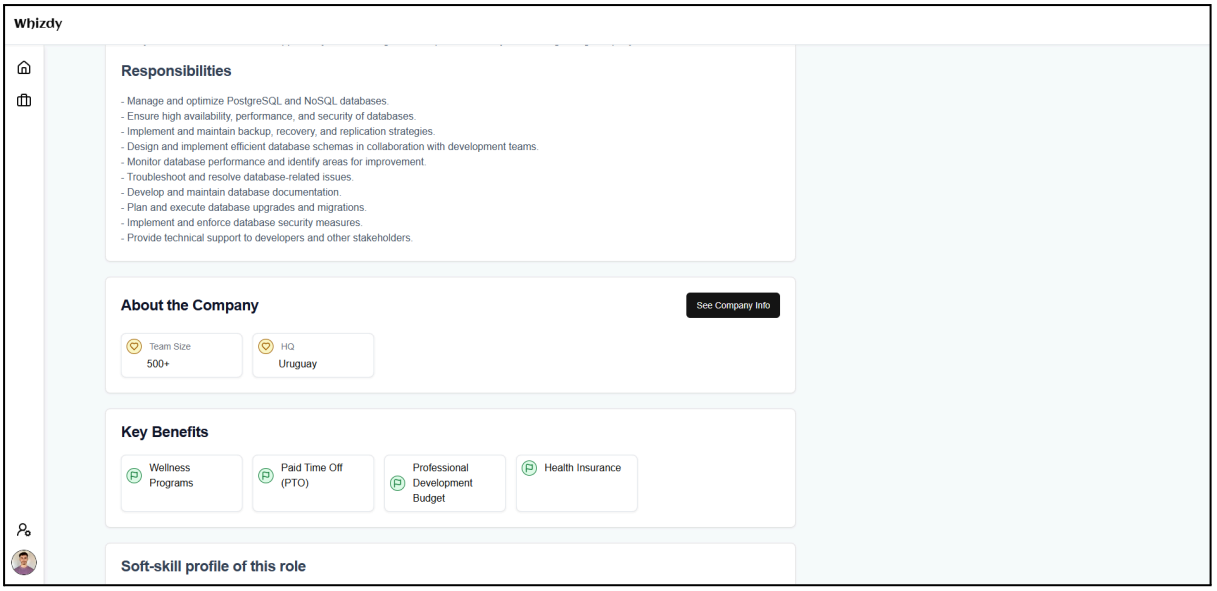


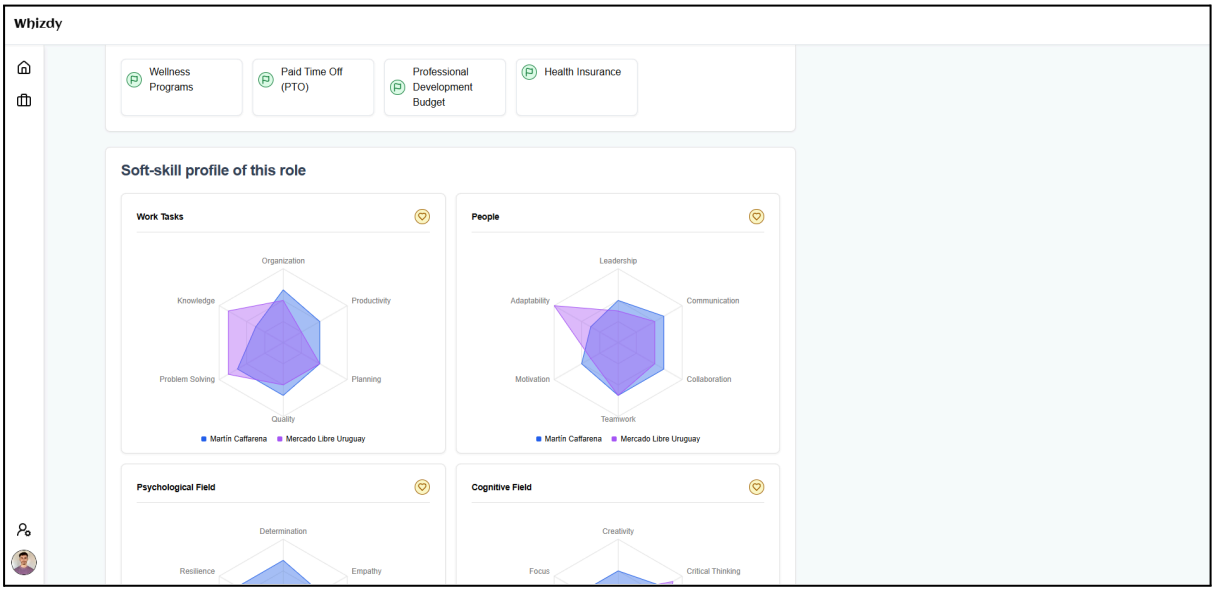




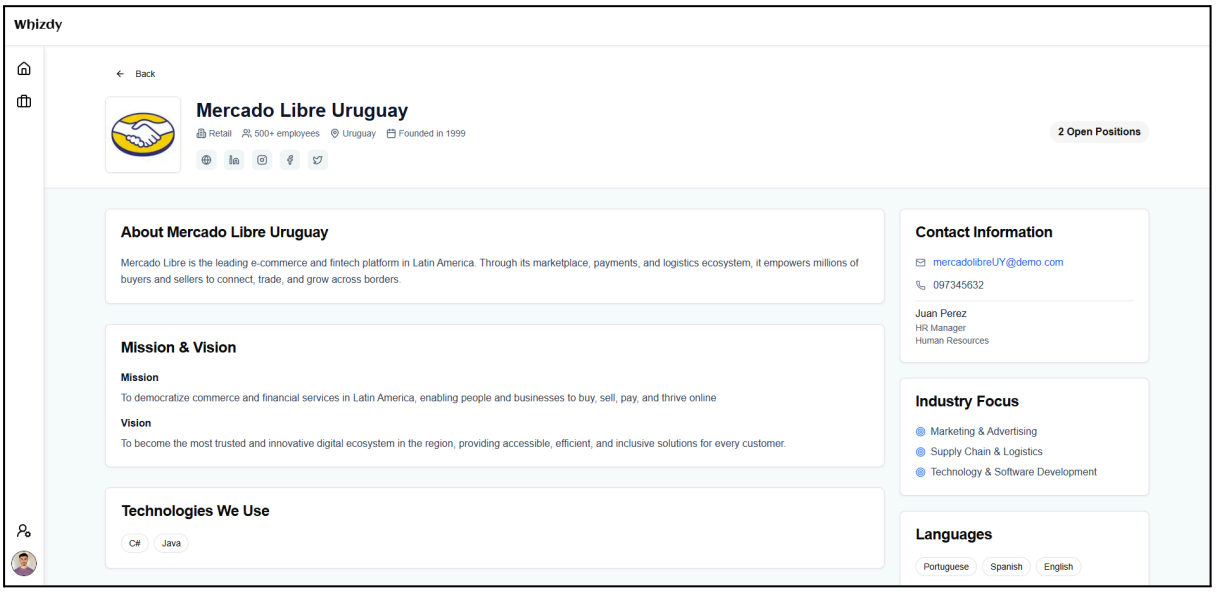
RF63 - Detalle de vacante:

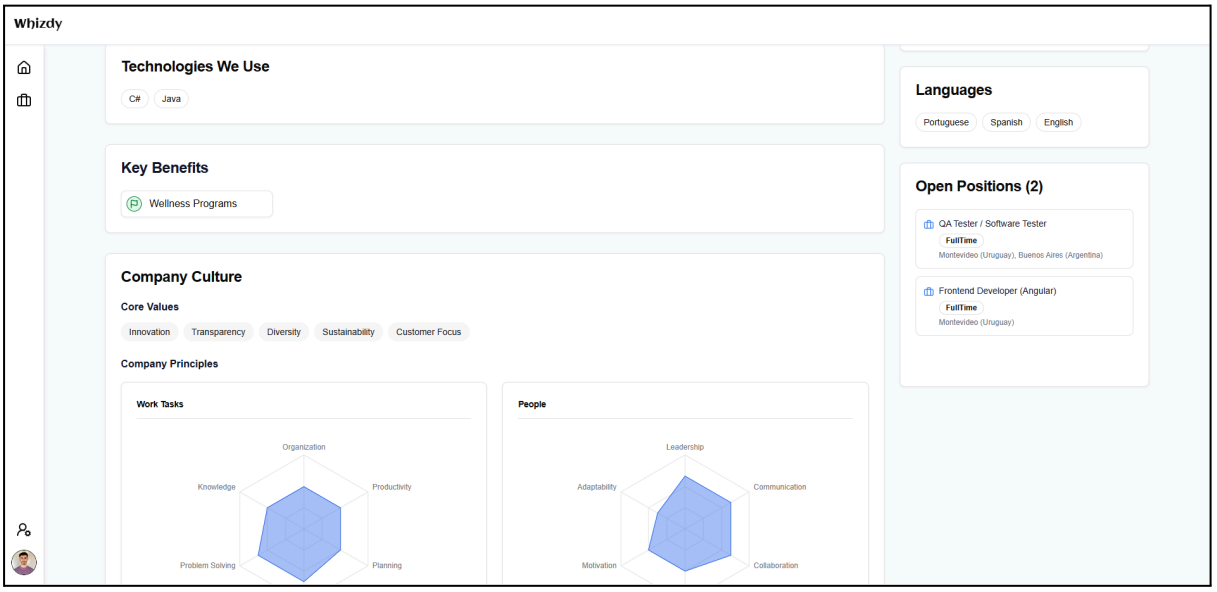



Whizdy
Responsibilities
- Manage and optimize PostgreSQL and NoSQL databases.
- Ensure high availability, performance, and security of databases.
- Implement and maintain backup, recovery, and replication strategies.
- Design and implement efficient database schemas in collaboration with development teams.
- Monitor database performance and identify areas for improvement.
- Troubleshoot and resolve database-related issues.
- Develop and maintain database documentation.
- Plan and execute database upgrades and migrations.
- Implement and enforce database security measures.
- Provide technical support to developers and other stakeholders.
About the Company
See Company Info
Team Size
500+
HQ
Uruguay
Key Benefits
Wellness Programs
Paid Time Off (PTO)
Professional Development Budget
Health Insurance
Soft-skill profile of this role


Whizdy
Wellness Programs
Paid Time Off (PTO)
Professional Development Budget
Health Insurance
Soft-skill profile of this role
Work Tasks
Organization
Productivity
Planning
Quality
Problem Solving
Knowledge
Martin Caffarena
Mercado Libre Uruguay
People
Leadership
Communication
Collaboration
Teamwork
Motivation
Adaptability
Martin Caffarena
Mercado Libre Uruguay
Psychological Field
Determination
Empathy
Resilience
Cognitive Field
Creativity
Critical Thinking
Focus

RF66 - Perfil de la empresa:





16.2. Evidencia - Notas meet validación problema con recruiter

Preguntas:

La idea del proyecto es, mediante IA, hacer match entre un puesto de trabajo y un candidato considerando preferencias del usuario, habilidades técnicas, experiencia, educación y aspectos de personalidad/soft skills. Las skills técnicas son casi cuantitativas (años de experiencia); en soft skills y rasgos personales hay dudas sobre qué datos usar en el match.

- ¿Qué aspectos del *fit* cultural consideran más relevantes (valores, estilo de trabajo, comunicación, liderazgo)?
- ¿Usan frameworks/metodologías? ¿Aplican como *fit* cultural o más como psicometría/soft skills?
- ¿Qué preguntas o situaciones usan para evaluar adaptación a la cultura?
- ¿Cómo evalúan la alineación de valores?
- ¿Los tests de *fit* cultural son genéricos o personalizados por empresa?
- ¿Utilizan herramientas o pruebas psicométricas?
- ¿Qué competencias miden (resiliencia, liderazgo, resolución de conflictos, etc.)?
- ¿Cómo integran esos resultados en el proceso?
- ¿Dependen del tipo de puesto/seniority?
- ¿Qué soft skills se consideran? ¿Cuáles varían según el rol?
- ¿Cómo se evalúan (simulaciones, autoevaluaciones, entrevistas)?
- ¿Cómo se mide su desarrollo a lo largo del tiempo?
- Como recruiter, ¿preferís herramientas tecnológicas (match automático) o entrevistas con especialista?
- Cantidad de entrevistas y participantes (RR. HH., cliente, técnica).
- Filtros rápidos: inglés, salario, etc.

- Qué información real buscan obtener que no siempre está en el CV.
- ¿Qué feedback dan/reciben durante el proceso?
- Si la selección la hace una IA, ¿qué explicación necesitan (por qué fue elegido/descartado)?
- ¿Sería útil ajustar "pesos" por característica al crear la vacante?

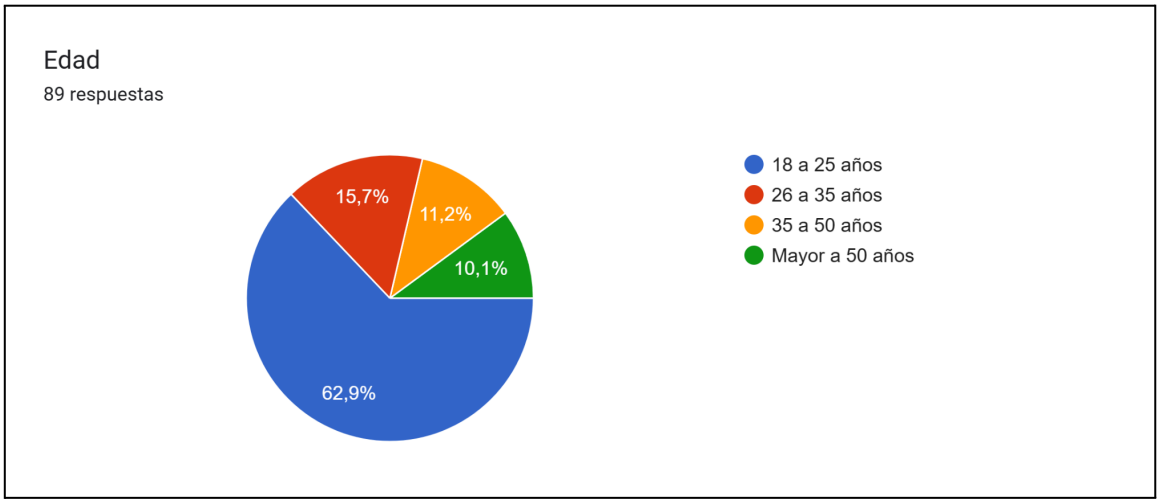
Respuestas:

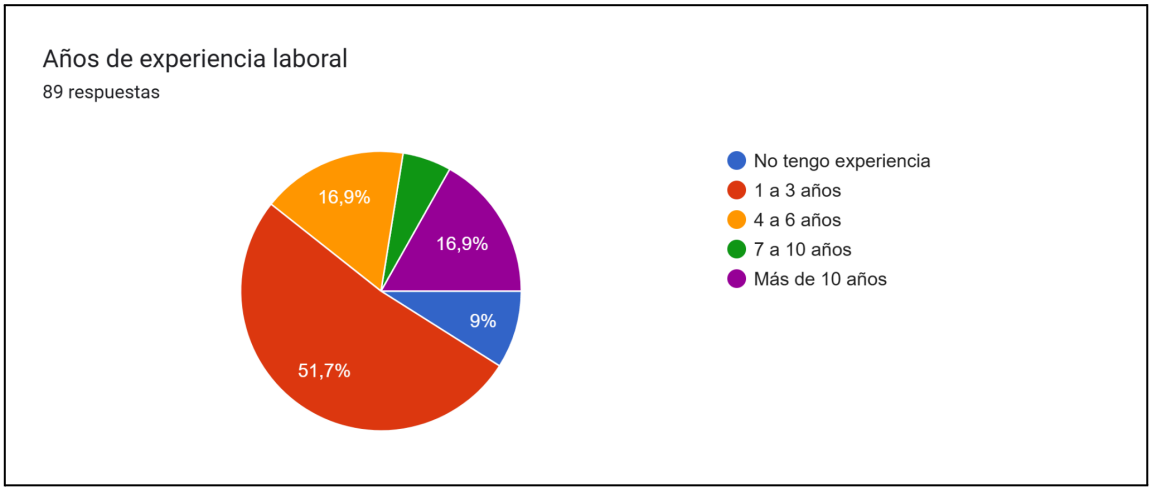
- Suelen ser 2–3 entrevistas: filtro de RR. HH. y entrevista con el cliente según el puesto.
- En perfiles IT, muchas veces no es factible aplicar ciertos tests porque la gente no busca activamente; los recruiters los contactan.
- Motivación muy importante; no buscan gente sólo por cambio económico.
- Valoran permanencias mayores a 6 meses.
- Consideran nivel de inglés y preguntas por defecto (inglés, salario, etc.).
- Se manejan con requerimientos técnicos, tecnologías, experiencia e idioma.
- No siempre son lo principal; se analizan en la entrevista y según cómo responden a preguntas/pruebas.
- Uso del modelo *STAR* (Situación, Tareas, Acciones, Resultados).
- Buscan entender cómo trabaja la persona individual y en equipo.
- Pueden usar tests de personalidad cortos; en algunos casos no lo ven factible en IT.
- Los tests son indicadores; no deberían tomarse con demasiada firmeza.
- Presencial aporta valor (ver cómo lo hacen).
- Puede ser necesario personalizar por empresa.
- La creación de descripciones de trabajo la hacen con GPT.

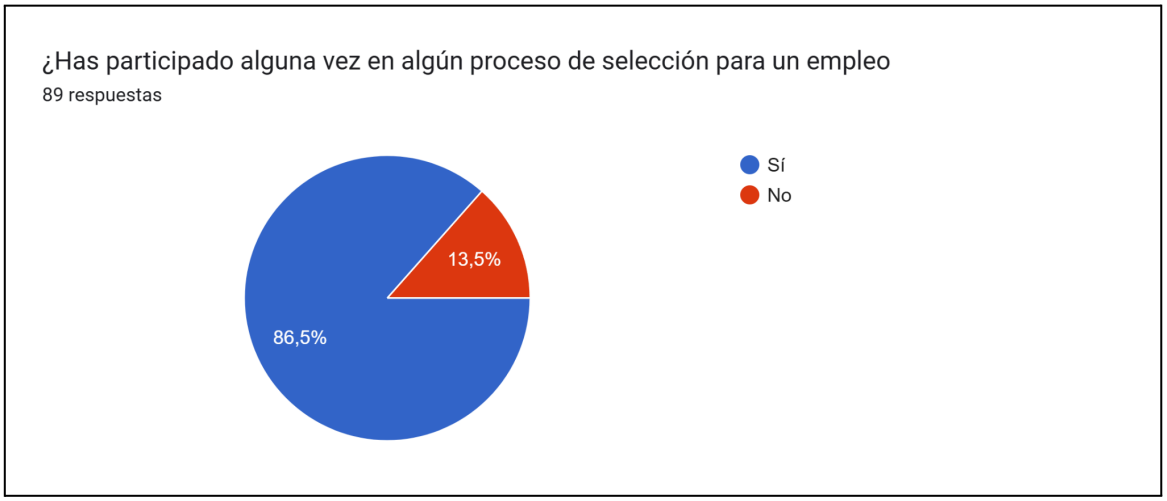
- La IA podría ayudar en etapas iniciales, sobre todo en procesos con volumen.
- Si decide una IA, piden explicación del motivo de selección o descarte.
- Quieren cargar CV y que se autocompleten los campos del perfil.
- Buscan información real que no siempre está en el CV.
- En tests online existe riesgo de que los haga otra persona.
- Prefieren entrevistas en vivo para ver cómo responde.
- Guardan la información del candidato para procesos futuros.
- Suelen dar feedback sobre todo en partes técnicas.
- A veces no le dan feedback al cliente.
- Puede depender del puesto o momento qué se prioriza.
- Sería útil que el recruiter asigne pesos a cada característica al subir una vacante.

16.3 Encuesta validacion problema con usuarios candidatos

La siguiente sección resume los resultados de la encuesta realizada.







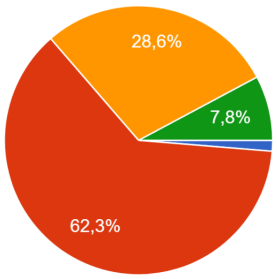
Rubro laboral

89 respuestas

- IT
- Informática
- Desarrollo de software
- Software
- Tecnología
- Administración
- Finanzas
- Sistemas
- Marketing

¿Cuántos procesos de selección has realizado a lo largo de tu vida?

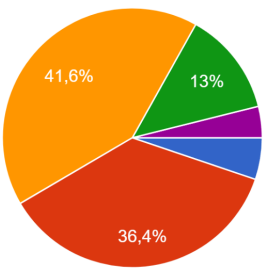
77 respuestas



- 0
- 1 a 5
- 5 a 10
- más de 10

¿Cómo calificarías la comunicación que recibiste durante los procesos (antes, durante y después)?

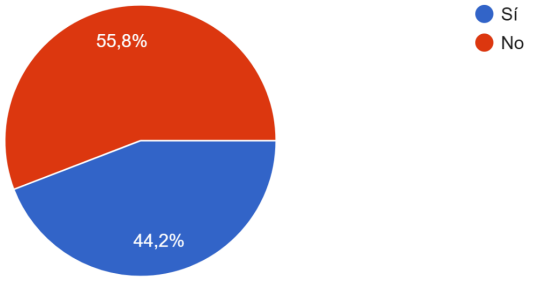
77 respuestas



- Muy buena
- Buena
- Regular
- Mala
- Muy mala

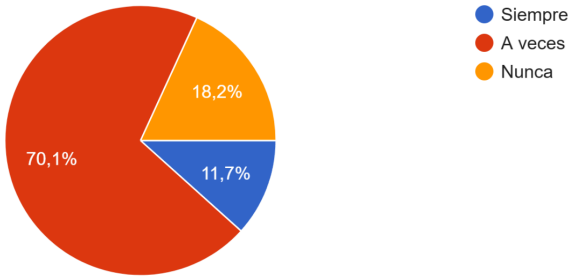
¿Sentiste que se evaluaron correctamente tus capacidades más allá del CV?

77 respuestas



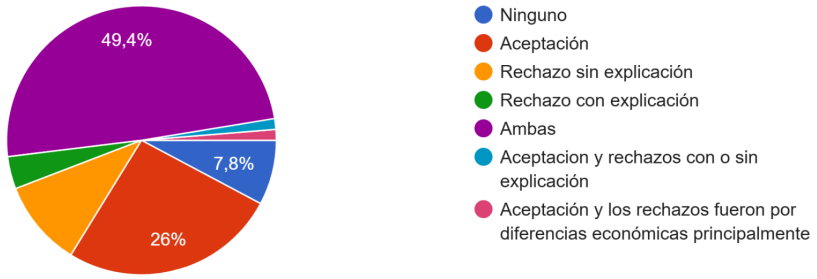
¿Sueles recibir devoluciones una vez terminado el proceso?

77 respuestas



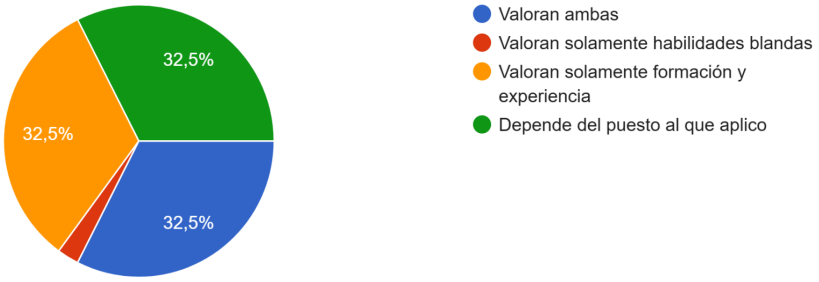
¿Qué tipo de feedback has recibido?

77 respuestas



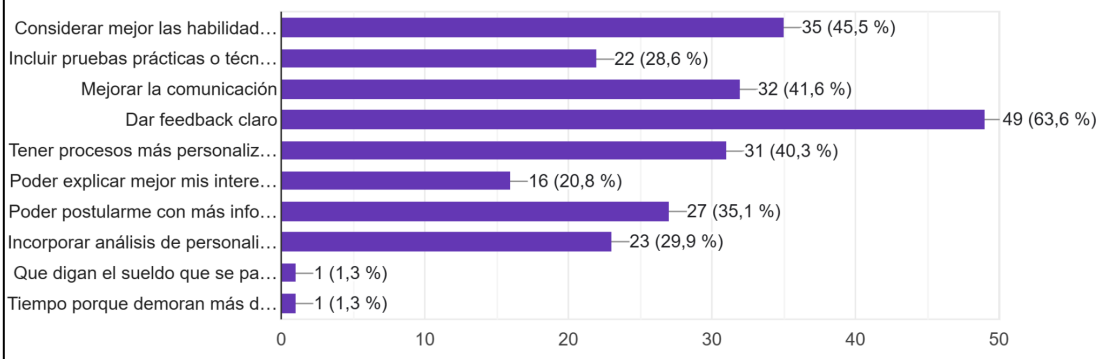
¿Creés que los procesos actuales valoran realmente tus habilidades blandas o solo tu formación y experiencia?

77 respuestas



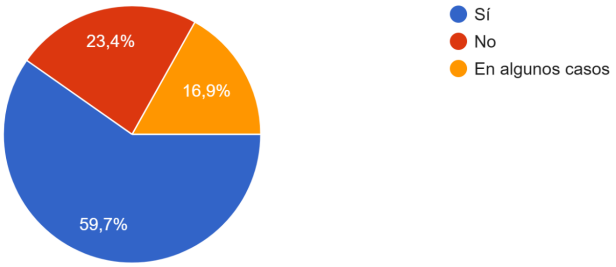
Si pudieras cambiar algo de los procesos de selección tradicionales, ¿Qué cambiarías?

77 respuestas



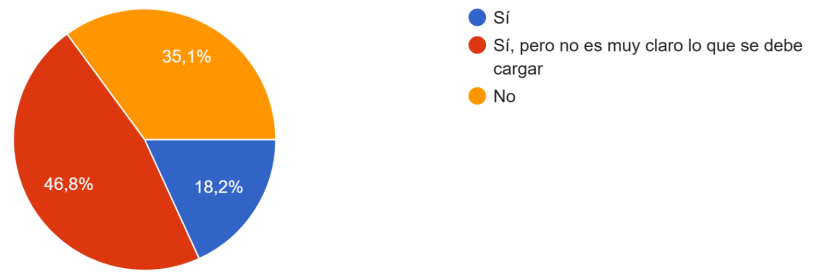
¿Te ha pasado sentirte descartado sin entender el motivo ni obtener una explicación?

77 respuestas



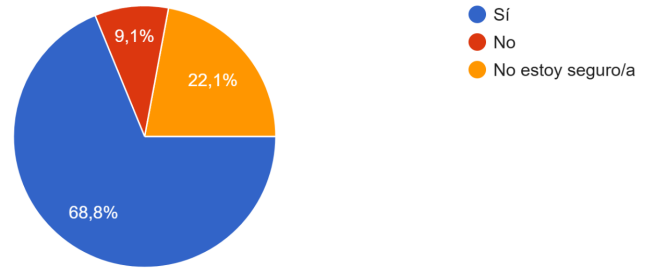
¿Sentís que las plataformas de búsqueda de empleo permiten agregar toda la información valiosa para que sea evaluado correctamente un candidato?

77 respuestas



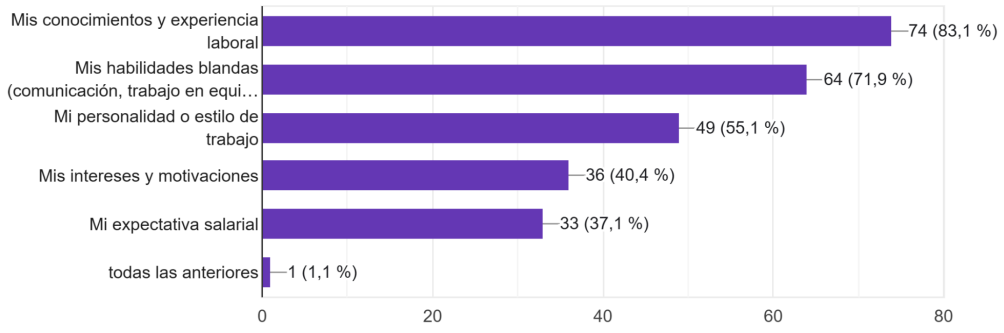
¿Creés que te has perdido oportunidades laborales por no llegar a ver una vacante a tiempo o por haber demasiadas postulaciones?

77 respuestas



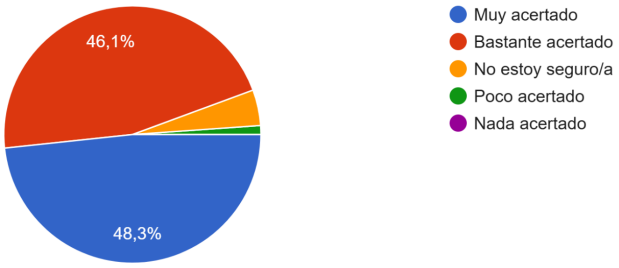
¿Qué aspectos consideras más importantes que se tengan en cuenta al analizar si un puesto encaja con vos?

89 respuestas



¿Qué tan acertado te parecería que se analice tu perfil considerando tus intereses, habilidades blandas y aspiraciones salariales para entender mejor qué tan adecuado es un puesto para vos?

89 respuestas



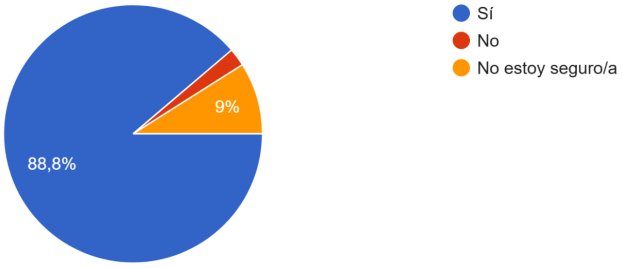
¿Qué tipo de información te gustaría recibir al finalizar un proceso de selección?

89 respuestas

- Que me manifiesten que opinan sobre mis habilidades, mi experiencia, mi personalidad y mis conocimientos
- devolución sobre porque si o porque no me aceptan
- Si cumplo con los requisitos del puesto que estoy aspirando y como me proyecta a futuro
- Me gustaría que se fuera muy explícito con los motivos que determinen la decisión, principalmente en casos de rechazo
- Si quede o no y porque
- El por qué me seleccionaron o no
- dar el por qué no quede seleccionado
- Razón por la que me aceptan y razón por la que no, dependiendo el caso y que puedo mejorar o que "me falto"

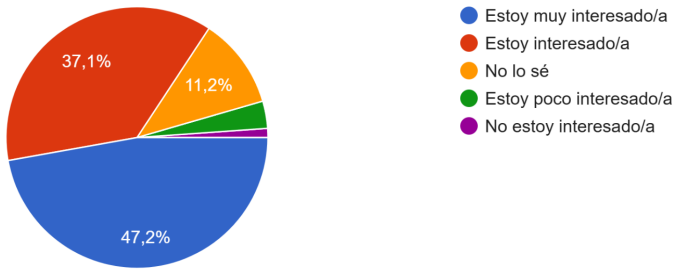
¿Creés que un sistema que considere múltiples aspectos de tu perfil (habilidades, personalidad, experiencia, etc.) le aportaría valor a un proceso de selección?

89 respuestas



¿Cuánto te interesaría que en los procesos se tomen en cuenta aspectos como tus valores personales, tu estilo de trabajo o tu forma de comunicarte?

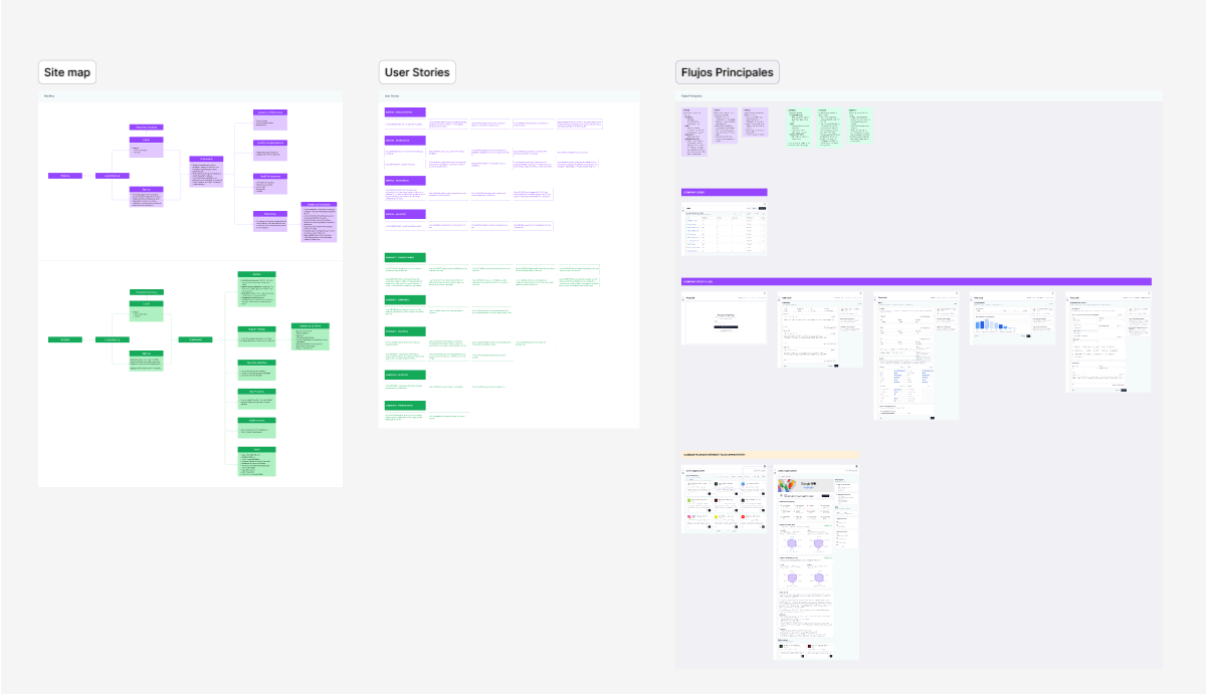
89 respuestas



16.4. Evidencia - Crecimiento de diseño en Figma

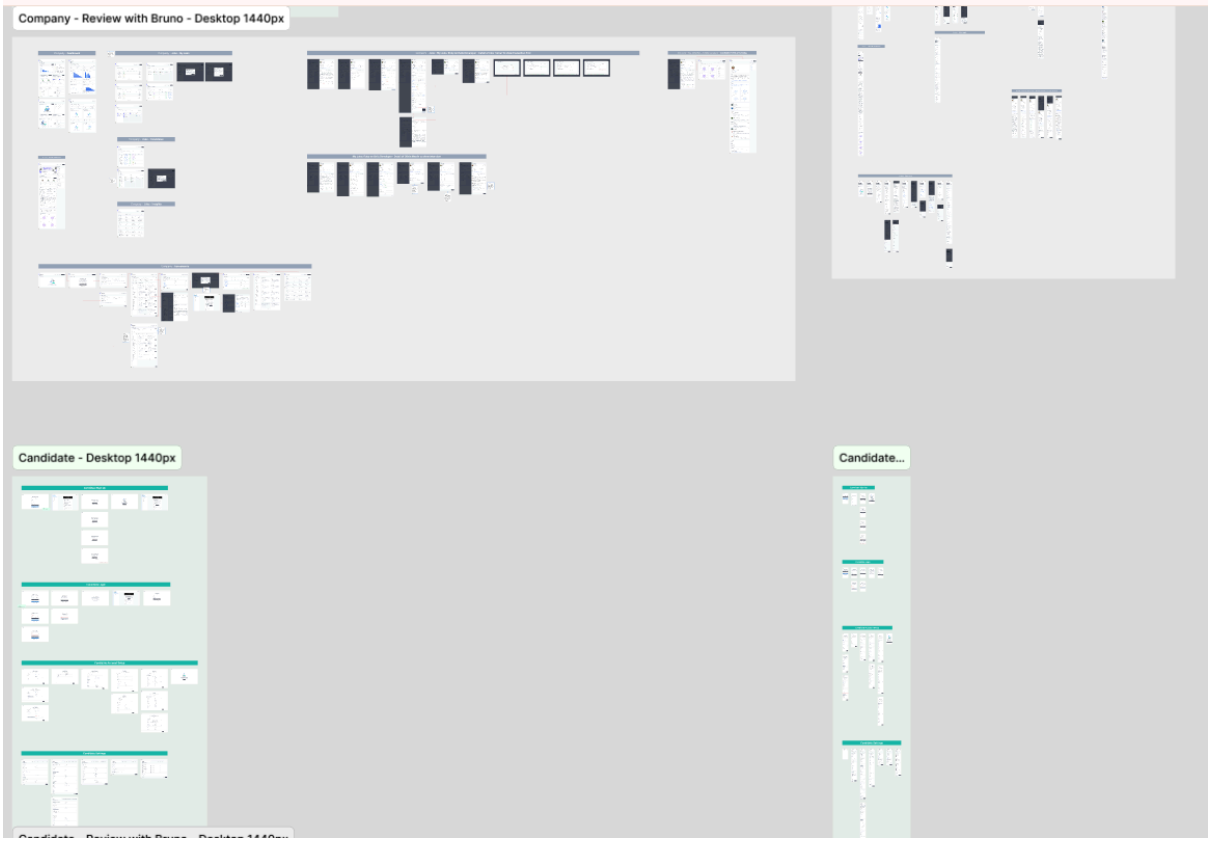
Comparación para comparación visual del crecimiento del diseño en *Figma*, imágenes meramente ilustrativas para contemplar las diferencias de cantidad de contenido.

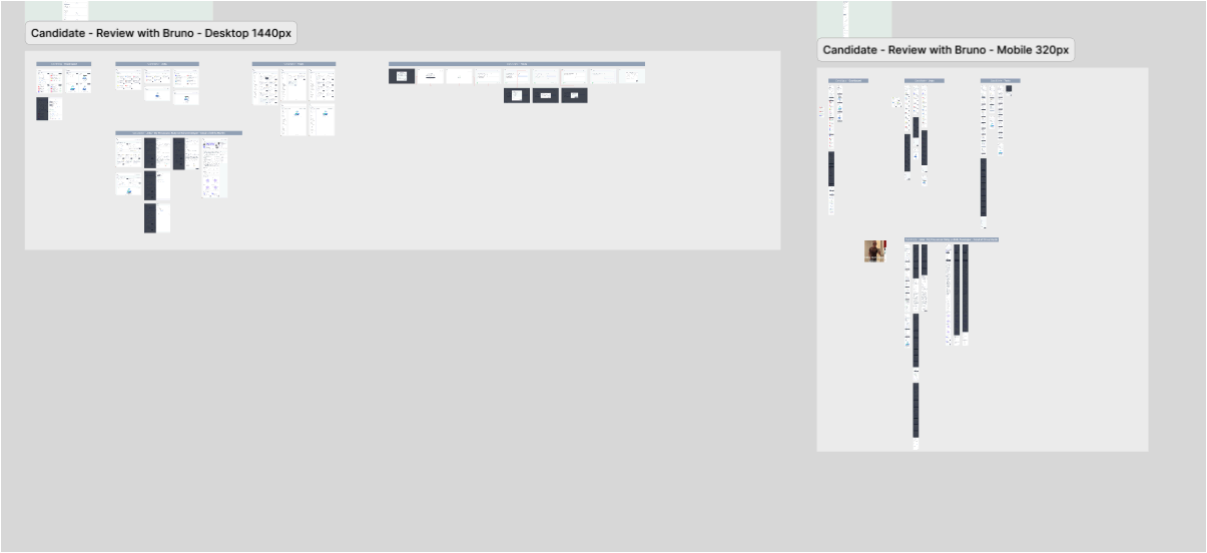
Primer *Figma*:



Figma final:


Company - Desktop 1440px
Company - Mobile 320px
Company - Review with Bruno - Mobile 320px
Company - Review with Bruno - Desktop 1440px


Company - Review with Bruno - Desktop 1440px
Candidate - Desktop 1440px
Candidate...


Candidate - Review with Bruno - Desktop 1440px
Candidate - Review with Bruno - Mobile 320px

16.5. Evidencia - Primeras historias de usuario

A continuación se listan algunas de las primeras versiones de las Historias de Usuario, debido a la cantidad de estas se incluyen a nivel de referencia y evidencia únicamente algunas de ellas. Estas formaron parte de las primer iteración de escritura de Historias de usuario

US1 - Publicar Vacantes de Trabajo

Como EMPRESA, **quiero** publicar vacantes de trabajo, **para** encontrar los mejores candidatos que se adecuen a las necesidades del puesto de trabajo.

Criterios de Aceptación

- La vacante debe incluir los siguientes campos obligatorios:
    - Título de la vacante.
    - Tipo de empleo (Job type).
    - Modalidad de trabajo (workplace type).
    - Nivel (level).
    - Ubicación del empleo (job location).
    - Descripción del puesto.
    - Responsabilidades.
    - Skills requeridas (hard y soft skills).
    - Compensación/Remuneración
- Poder agregar información extra (como puede ser
- Los datos deben validarse antes de publicar la vacante (campos vacíos no permitidos).
- La vacante debe guardarse en el sistema y ser visible para candidatos compatibles.

Prioridad: 1 (Muy alta)

US2 - Marcar Habilidades Blandas y Técnicas

Como EMPRESA, **quiero** seleccionar habilidades blandas y técnicas necesarias para el puesto, **para** definir mejor los requisitos del candidato ideal.

Criterios de Aceptación

- La empresa debe poder seleccionar habilidades de un pool predefinido.
- El sistema debe permitir agregar habilidades personalizadas.
- Cada habilidad debe poder tener un nivel asignado (Begginer, Proficient, Expert).
- Las habilidades seleccionadas deben guardarse correctamente junto con la vacante.

Prioridad: 1 (Muy alta)

US3 - Editar, Archivar, Finalizar o Cancelar Vacantes

Como EMPRESA, **quiero** poder editar, archivar, finalizar o cancelar vacantes, **para** gestionar su ciclo de vida según las necesidades de contratación.

Criterios de Aceptación

- Debe ser posible editar campos de una vacante publicada.
- Archivar una vacante la desactiva, pero mantiene los datos para consultas futuras.
- Finalizar o cancelar una vacante debe notificar a los candidatos interesados.
- Las acciones realizadas deben reflejarse inmediatamente en el sistema.

Prioridad: 1 (Muy alta)

US4 - Reactivar Vacantes Antiguas

Como EMPRESA, **quiero** desarchivar y volver a publicar vacantes antiguas, **para** reutilizar información previamente guardada y poder recuperar la información de candidatos que previamente se habían postulado.

Criterios de Aceptación

- Las vacantes archivadas deben ser accesibles desde una sección "Archivadas".
- El sistema debe permitir actualizar la vacante antes de reactivarla.
- El sistema debe de poder conservar la información de los candidatos y el estado en el que se encontraba el proceso cuando la vacante estaba publicada.

Observaciones

- Opción 1: el historial de candidatos que iniciaron una entrevista podría seguir intacto pero indicando el match actual (ej. si la persona está o no open to work, si hizo nuevos tests o si sus habilidades cambiaron).
- Opción 2 (opción elegida actualmente): si técnicamente es complejo, la otra opción es que figure el historial con el match viejo, y si la empresa lo necesita que pida manualmente que se vuelva a comparar el candidato con la vacante.

Prioridad: 3 (Media)

US5 - Ver datos estadísticos de las interacciones con las vacantes

Como EMPRESA, **quiero** ver datos estadísticos sobre la interacción de los candidatos con mis vacantes, **para** evaluar el rendimiento y la atracción de mis publicaciones.

Criterios de Aceptación

- Los datos estadísticos deben incluir:
    - Número de visualizaciones.
    - Número de interesados.
    - Número de guardados.

    - Número de veces compartidos.
- Los datos deben ser accesibles desde el dashboard de la empresa.
- Las estadísticas deben actualizarse en tiempo real.

Prioridad: 3 (Media)

US6 - Ver Candidatos Compatibles con Vacantes

Como EMPRESA, quiero ver a todos los candidatos compatibles con nuestras vacantes para explorar perfiles que se ajusten a nuestras necesidades.

Criterios de Aceptación:

- Mostrar una lista de candidatos sugeridos, incluyendo nombres, apellidos, y el porcentaje de compatibilidad con la vacante.
- La lista debe ser paginada y permitir búsquedas por nombre, posición o nivel de compatibilidad.
- Incluir un indicador del estado del candidato con la empresa (contactado, en evaluación, descartado, etc.).
- La lista debe actualizarse en tiempo real en base a los criterios de compatibilidad definidos y a los cambios en la información de los candidatos o vacantes.
- Permitir filtrar la lista por criterios como ubicación, experiencia o habilidades clave.

Prioridad: 1 (Muy alta)

US7 - Ver los detalles de los candidatos

Como EMPRESA quiero ver los detalles de los candidatos **para** analizar su perfil completo antes de avanzar en el proceso.

Criterios de Aceptación:

- Al seleccionar un candidato, debe mostrarse un perfil completo con experiencia, habilidades, tests realizados y feedback recibido.
- Incluir un historial de interacciones entre la empresa y el candidato.

Prioridad: 1 (Muy alta)

US8 - Filtrar candidatos por porcentaje de compatibilidad

Como EMPRESA quiero filtrar candidatos por porcentaje de compatibilidad **para** priorizar aquellos que mejor se ajustan a la vacante.

Criterios de Aceptación:

- Incluir un filtro que permita clasificar candidatos por porcentaje de compatibilidad.
- Agregar opciones adicionales de filtro como experiencia, habilidades clave y localización.
- Mostrar resultados ordenados del porcentaje más alto al más bajo.

US9 - Buscar candidatos sin necesidad de publicar una vacante

Como EMPRESA quiero buscar candidatos sin necesidad de publicar una vacante **para** explorar opciones sin crear un llamado formal.

Criterios de Aceptación:

- Permitir búsquedas por palabras clave relacionadas con habilidades, experiencia y ubicación.
- Mostrar sólo candidatos que tengan el estado "Open to Work".
- Garantizar la privacidad de los candidatos que no desean aparecer en las búsquedas (aquellos los cuales tengan apagada la flag “Open to Work”).

Prioridad: 3 (Media)

US10 - Descartar candidatos

Como EMPRESA quiero descartar candidatos **para** gestionar de manera eficiente el proceso de selección y mantener aquellos los cuales se adecuan a la vacante.

Criterios de Aceptación:

- Incluir una opción para marcar a un candidato como descartado.
- Solicitar confirmación antes de realizar esta acción.
- Los candidatos descartados deben aparecer en una sección específica para revisiones futuras.

Prioridad: 3 (Media)

US11 - Guardar candidatos

Como EMPRESA quiero guardar candidatos **para** hacer seguimiento de perfiles interesantes para futuras evaluaciones.

Criterios de Aceptación:

- Incluir una opción para marcar candidatos como "guardados".
- Mostrar una lista separada con los candidatos guardados.
- Permitir búsquedas dentro de la lista de candidatos guardados.

Prioridad: 4 (Baja)

US12 - Coordinar entrevistas

Como EMPRESA quiero coordinar entrevistas con los candidatos **para** avanzar en el proceso de selección de manera organizada.

Criterios de Aceptación:

- Incluir una opción para agendar entrevistas desde el perfil del candidato.
- Integración con herramientas como Calendly para seleccionar fechas disponibles y enviar invitaciones.
- Notificaciones automáticas a los candidatos con detalles de la entrevista.

Prioridad: 3 (Media)

US13 - Actualizar estado de postulaciones y enviar correos a los candidatos

Como EMPRESA quiero actualizar los estados y enviar correos a los candidatos **para** comunicarse de manera directa y eficiente y mantenerlos informados sobre el estado de su postulación.

Criterios de Aceptación:

- Al actualizar una postulación enviar una notificación.
- Incluir plantillas predefinidas para acelerar la comunicación.
- Registrar las notificaciones enviadas en el historial del candidato.

Observación:

En primer lugar se quería poder contar con mensajería interna, pero tener un chat interno nos parece complejo y extenso para el impacto que puede llegar a tener en el producto, teniendo en cuenta que las comunicaciones se van a tener en reuniones y/o por email. Como alternativa podríamos integrar el envío de mensajes con Whatsapp o un envio de email automático, esto nos permitirá concentrar mayor esfuerzo en otras partes como el algoritmo de IA del proyecto.

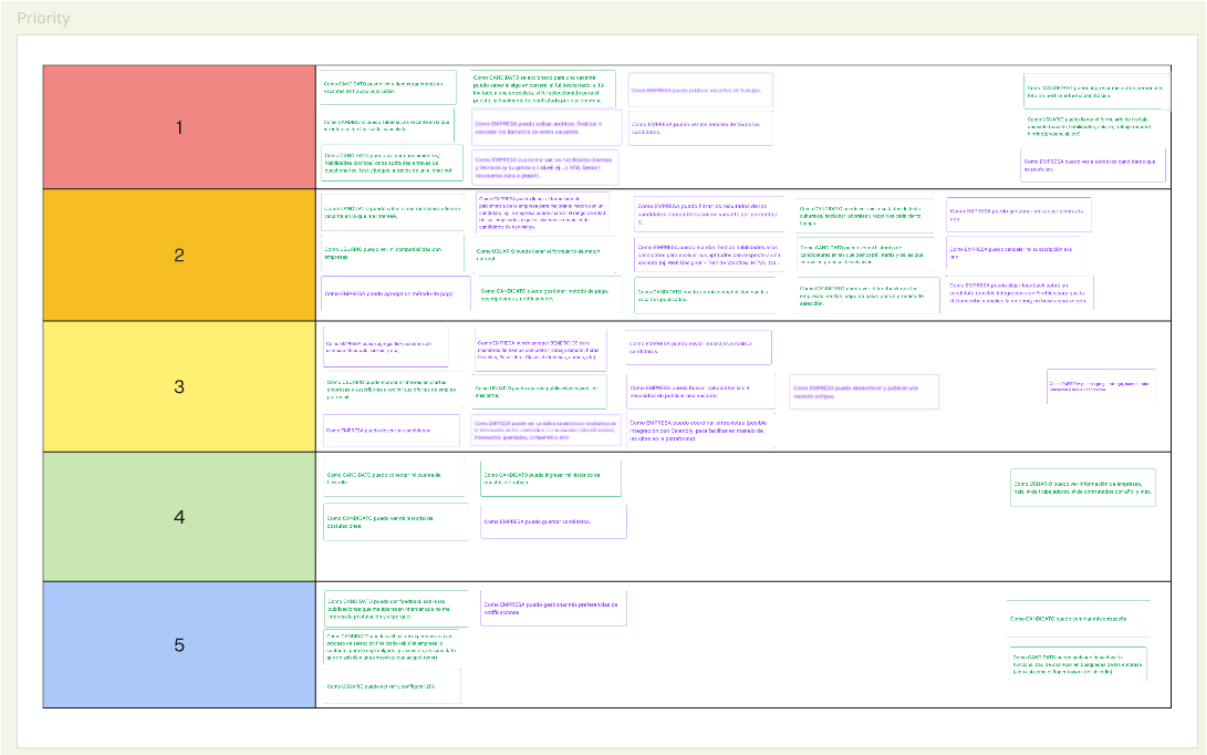
Prioridad: 3 (Media)

16.6. Gráfica inicial esfuerzo vs impacto

Para mayor legibilidad de la imagen dirigirse al documento original mediante el siguiente acceso aquí.

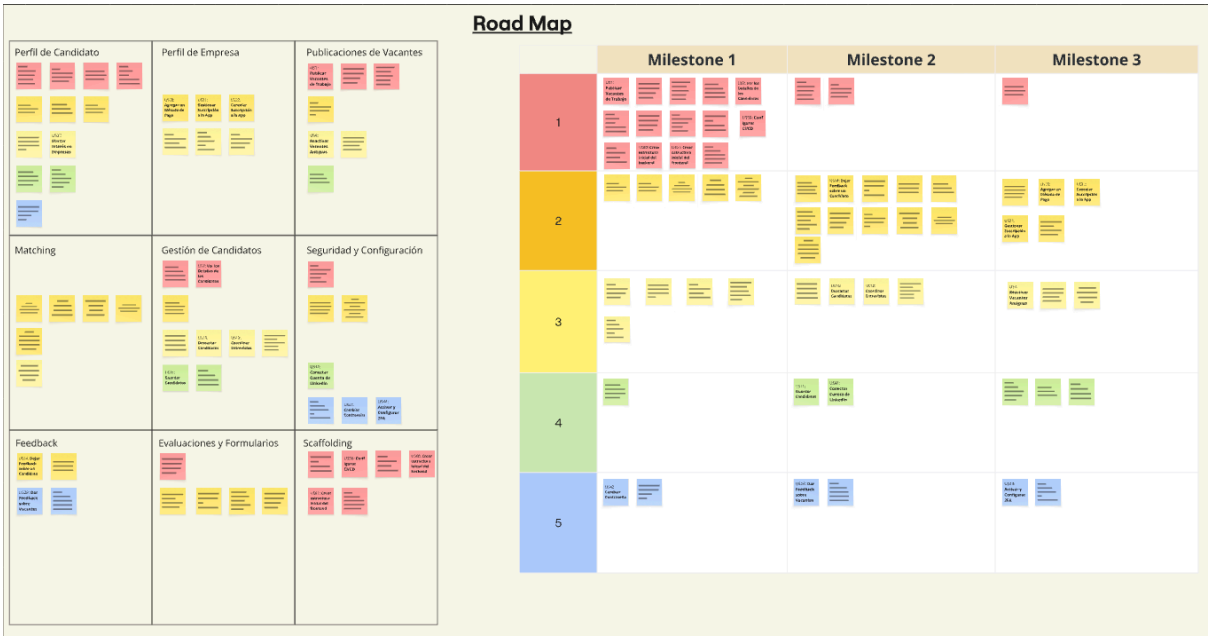
16.7. Primer priorización de historias de usuario

Para mayor legibilidad de la imagen dirigirse al documento original mediante el siguiente acceso aquí.



16.8. *RoadMap* inicial

Para mayor legibilidad de la imagen dirigirse al documento original mediante el siguiente acceso aquí.



También se planteó el siguiente documento con mayor detalle, ambos elementos compartidos con el cliente

Roadmap Whizdy

- Preparación Inicial :
    - Configuración del entorno de desarrollo (repositorios, herramientas de desarrollo, Jira, Clockify).
    - Revisión final de historias de usuario
    - Estimación de historias
    - Planificación del backlog y primeras iteraciones.

Inicio del Desarrollo: Finales de Enero 2025

Milestone 1: Fundamentos del Sistema (Finales Enero - Finales Abril 2025)

Fase 1: Creación de Perfiles y Gestión de Vacantes

1. Gestión de Perfiles:
    - Registro de empleadores.

        - Registro de candidatos.
            - Captura de datos personales, información básica, habilidades técnicas, blandas, experiencia y preferencias.
            - Llenar el formulario de trabajo deseado, indicado puesto, habilidades, salario, etc.
    2. Gestión de Vacantes:
        - Publicar vacantes indicando descripción, habilidades, experiencia, etc para el puesto.
        - Gestión del ciclo de vida de las vacantes (editar, archivar, cerrar, visualizar candidatos postulados).

Fase 2: *Matching* Básico y Pruebas Iniciales

1. Desarrollo del Algoritmo:
    - Implementar el algoritmo de matching inicial basado en embeddings para habilidades técnicas y blandas.
    - Configuración de pesos predeterminados para roles específicos.
2. Pruebas Internas:
    - Validar el flujo entre perfiles, vacantes y el matching.
    - Identificar y corregir errores funcionales.

Entrega de Milestone 1

- Sistema funcional con perfiles, vacantes y matching básico listo para pruebas extendidas.

Milestone 2: Refinamiento y Nuevas Funcionalidades (Finales Abril - Agosto 2025)

Fase 1: Refinamiento del *Matching*

1. Optimización del Algoritmo:
    - Separar embeddings por categorías clave (habilidades técnicas, blandas, fit cultural).
    - Ajustes dinámicos de pesos basados en feedback de pruebas internas.
2. Mejoras en Funcionalidades Existentes:
    - Incorporar feedback al sistema, permitiendo a empleadores evaluar candidatos y entrevistas.

- Ajustes iterativos al flujo de vacantes y perfiles según pruebas.

Fase 2: Sistema de Notificaciones (Junio - Julio 2025)

1. Notificaciones Automatizadas:
    - Implementar notificaciones para informar cambios en vacantes y estados de postulaciones a candidatos y empleadores.

2. Pruebas Extendidas:
    - Validar las nuevas funcionalidades y flujos con escenarios simulados.
    - Corrección de errores y ajustes iterativos.

Entrega de Milestone 2

- Algoritmo optimizado, sistema de feedback y notificaciones implementados y listos para pruebas piloto con usuarios reales.

Milestone 3: Validación y Entrega del MVP (Agosto - Septiembre 2025)
Fase 1: Pruebas Piloto con Usuarios Reales

1. Pruebas con Usuarios:
    - Realizar sesiones piloto con empleadores y candidatos reales.
    - Recopilar feedback cualitativo y cuantitativo para evaluar la precisión del matching y experiencia de usuario.
2. Recopilación y Análisis de Feedback:
    - Ajustar funcionalidades y optimizar la plataforma basándose en resultados y comentarios de usuarios.

Fase 2: Ajustes Finales y Preparación del MVP (Agosto - Septiembre 2025)

1. Optimización de Rendimiento:
    - Pruebas de carga y escalabilidad para garantizar la estabilidad del sistema.
    - Corrección de errores críticos detectados en pruebas piloto.
2. Entrega del MVP:
    - Completar funcionalidades menores.
    - Presentar un sistema estable y funcional que incluya:
    - Gestión de perfiles y vacantes.

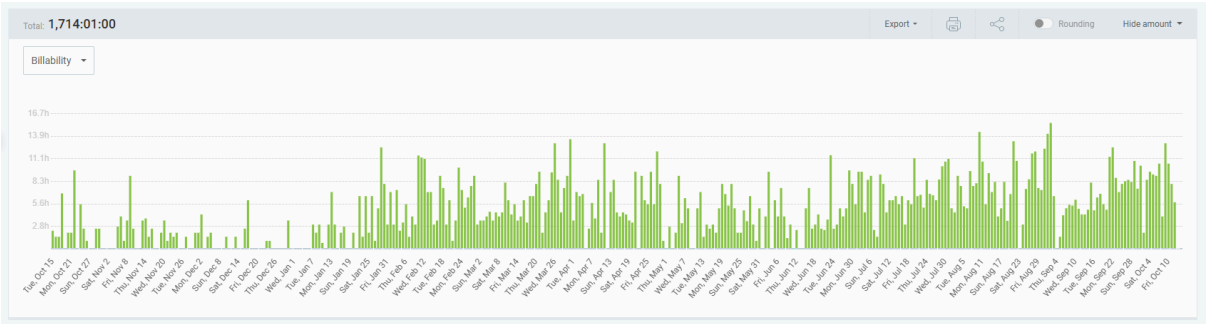
- Algoritmo de matching optimizado.
- Sistema de feedback y notificaciones.
- Panel básico de administración para empleadores y candidatos.

Entrega de Milestone 3 (Finales de Septiembre 2025):

- MVP funcional y validado, listo para presentación final al cliente.

16.9. Registro de horas y métricas de dedicación

Registro detallado de las horas registradas mediante herramienta Clockify
Fecha reporte: 10 de Octubre 2025



Totales por fase: Duración del proyecto: 53 semanas.

Fase	Período	Horas	% del total
Investigación	10/10/2024 – 20/01/2025	**148 h**	**8,02%**
Desarrollo	20/01/2025 – 16/09/2025	**1.477 h 40 m**	**80,10%**
Documentación	16/09/2025 – 16/10/2025	**219 h**	**11,87%**
Total proyecto	10/10/2024 – 16/10/2025	**1.844 h 40 m**	**100%**

Desarrollo - Milestone e iteraciones

Horas por milestone :

- **Milestone 1 (Iter. 1–7): 555 h**
- **Milestone 2 (Iter. 8–14): 530 h**
- **Milestone 3 (Iter. 15–17): 392** *(la Iteración 17 duró 3 semanas)*

Promedio por iteración dentro de cada milestone:

- **M1:** 555 h / 7 = **79 h 16 m** por iteración
- **M2:** 530 h / 7 = **76 h** por iteración
- **M3:** 392 h / 3 = **131 h** por iteración

Promedio general por iteración (Desarrollo, 17 iteraciones):

- **1.478 h / 17 = 87 h**

Tiempo por actividad

- **Gestión (fase de Desarrollo, 15–20%): ≈ 222 h – 296 h**
  (equivale a ~**12,02%–16,02%** del total del proyecto)
- **Reuniones con tutor (todo el proceso): ≈ 26 h 30 m**
- **Reuniones internas de equipo:**
  - **Fase de Investigación: ≈ 73 h**
  - **Fases de Desarrollo + Documentación: ≈ 115 h**
- **Tiempo total estimado en reuniones : ≈ 215 h**

16.10. Evidencia - Notas por Milestone (Checkpoint y final)

La siguiente sección contiene evidencia de la documentación de las notas tomadas por Milestone, estas se tomaban a mitad y final de cada Milestone. Se incluyen algunas de ellas a modo de evidenciar las mismas y la información que estas contenían.

Milestone 1

Checkpoint

- Resumen del Progreso:
    - Hasta la fecha, se han implementado las siguientes funcionalidades principales:
    - Onboarding de compañíay candidato
    - SetUp de compañía y candidato
    - Login
        - Manejo de tokens de autenticación y sesiones.
        - Redireccionamiento automático por auth
    - Poblado de la Base de datos
    - Implementación del menú general de navegación.
    - Creación de una vacante de trabajo
    - *Embeddings* y *Matching*:
        - Se desarrolló una aplicación específica para el macheo (*embeddings*, generación de recomendaciones).
        - Se integró un *matching* básico que sugiere candidatos relevantes según los criterios de vacante.
        - Visualización de resultados para cada candidato y vacante.
    - Poblado de la Base de Datos: Seeds iniciales y datos de prueba para pruebas de front y embedders.<

- Desafíos Encontrados:
    - Dificultades con Next.js:
        - Curva de aprendizaje más alta de lo esperado.
        - Problemas con la estructura inicial del proyecto.
    - Tiempos de desarrollo mayores a lo estimado:
        - Varias tareas tomaron más tiempo del previsto debido a ajustes en definiciones y cambios en la arquitectura.
    - Tickets con un alcance mayor al esperado:
        - Algunas historias de usuario incluían múltiples funcionalidades, lo que generó subestimaciones en los tiempos de desarrollo.
        - Se identificaron dependencias no previstas en ciertos tickets, lo que impactó en la planificación.
    - Definiciones incompletas en algunos tickets:

        - Falta de especificaciones detalladas, lo que llevó a re-trabajo y ajustes en el camino.

- Decisiones Claves Tomadas:
    - Priorización de pruebas y validaciones:
        - Se definió qué aspectos del sistema se testearán en esta fase y cuáles pueden diferirse para iteraciones posteriores.
    - Postergación de algunas funcionalidades:
        - Se decidió no implementar por el momento las tareas y preguntas para una vacante hasta tener una definición más clara.
        - Algunas mejoras del front y definiciones de campos se dejan para la siguiente iteración.
    - Revisión de la estructura del proyecto:
        - Se realizó un refactor en la arquitectura del código para mejorar la organización y facilitar la escalabilidad.

- Riesgos Detectados:
    - Tiempos en la implementación del frontend:
        - La complejidad de Next.js, sumada a ajustes en la estructura, ha impactado los tiempos de desarrollo.
        - Se detectaron áreas donde la estimación inicial fue insuficiente, lo que podría generar futuros retrasos.

- Acciones Pendientes
    - Mejorar la definición de tickets:
        - Asegurar que cada historia de usuario esté bien detallada, incluyendo criterios de aceptación claros y dependencias.
    - Optimizar la estimación de tiempos:
        - Separar tareas grandes en subtareas más manejables para mejorar la precisión de la planificación.
    - Refinar el proceso de documentación y seguimiento:
        - Mantener un registro más detallado de las iteraciones y aprendizajes para mejorar la planificación futura.

Final de Milestone 1

- **Introducción**
  Desde el arranque del Milestone 1, nuestro equipo abordó de manera ordenada la implementación de flujos críticos de la plataforma: onboarding de compañías y candidatos, autenticación, gestión de sesiones y navegación básica. A lo largo de 8 iteraciones de dos semanas cada uno, consolidamos la base para el sistema de *matching* con *embeddings* y sentamos las bases de la arquitectura que sostendrá las iteraciones futuras.

- **Resumen del Progreso (Checkpoint)**
  En el checkpoint intermedio ya habíamos completado el onboarding de usuarios, el setup de compañía y candidato, el login con manejo de tokens y redirecciones automáticas, el poblado inicial de la base de datos (seeds), el menú de navegación y la creación de vacantes. Paralelamente, arrancamos la aplicación de *embeddings* y un *matching* básico que mostraba recomendaciones de candidatos por vacante, junto con la visualización de resultados en el *frontend*.

- **Desafíos Identificados**
  La curva de aprendizaje de Next.js fue superior a lo previsto, lo que nos llevó a repensar la estructura inicial del proyecto. Asimismo, varias historias de usuario resultaron ser más amplias de lo estimado y aparecieron dependencias no contempladas, generando retrasos y retrabajo hasta afinar las definiciones.

- **Decisiones Claves**
  Se priorizo pruebas y validaciones, dejando en segundo plano tareas complementarias como el módulo de assessments y refactorizar la arquitectura para mejorar la organización del código y facilitar la escalabilidad de los servicios.

- **Desarrollo y Aprendizajes**
  En lugar de reseñar cada iteración por separado, a continuación se presenta un relato integrado de cómo evolucionó el trabajo y qué lecciones extrajimos en cada bloque de actividades:

  Puesta en marcha y scaffolding
  Arrancamos definiendo la estructura base del proyecto (scaffolding), los flujos de registro de usuarios (sign-ups) y la autenticación con tokens. En esta fase inicial detectamos que muchas historias de usuario estaban escritas de forma demasiado específica, lo que complicó la estimación. Para solventarlo, instauramos ceremonias de ***backlog grooming*** y empezamos a registrar sistemáticamente los bloqueos y hallazgos técnicos.

Setup de compañías y comunicaciones

Con la base de autenticación lista, implementamos el flujo de creación de compañías y perfiles de candidatos, así como el envío de notificaciones iniciales. Aquí aprendimos a equilibrar la generación de valor inmediato para el cliente (notificaciones que confirmaran los pasos del onboarding) con la necesidad de no sobrecargar una iteración de dependencias. Refactorizamos la estructura de carpetas para acomodar mejor los nuevos servicios de notificación.

Publicación de vacantes y candidate setup

Seguidamente construimos el "post a job" completo -desde el formulario en el *frontend* hasta el endpoint en el backend- y refinamos el candidate setup. Fue clave validar por adelantado los prerrequisitos (token, rutas de navegación, layouts reutilizables) antes de arrancar cada historia; de lo contrario, surgían bloqueos que retrasaban toda la cadena de tareas.

Embeddings* y *matching

Una vez establecida la creación de vacantes, nos enfocamos en la parte esencial de la plataforma: generación de *embeddings* y *matching* básico. Completamos el flujo de procesamiento en segundo plano, la integración con la interfaz y la visualización de recomendaciones. Aquí aprendimos a ajustar el *backlog* conforme evoluciona el *Figma*, creando nuevas historias cuando aparecían features imprevistas.

Investigación continua y refinamientos

Paralelamente a la implementación, dedicamos iteraciones enteras a investigar herramientas y técnicas de *matching* más sofisticadas. En este bloque incorporamos manejadores de errores (error handlers) y refactors de la lógica de candidate setup para robustecer el sistema. Las estimaciones en tiempo real se convirtieron en práctica estándar: si un ticket crecía en complejidad, lo dividimos o posponemos partes no críticas.

Revisiones integrales y testing de integración

Hacia el cierre del Milestone hicimos dos revisiones generales, presentando avances a GoGrow para recibir feedback temprano. Montamos un protocolo de testing de integración que validó todos los flujos principales, desde el registro hasta el *matching*, antes de la demo final.

Cierre de funcionalidades y preparación de demo

En las últimas semanas cerramos todas las historias definidas para este Milestone, manteniendo una velocidad estable de 15–20 Story Points por iteración. Consolidamos nuestras ceremonias de retrospective y review, documentando métricas de rendimiento y bloqueos resueltos. Finalmente,

preparamos la demo para el cliente, destacando las áreas terminadas y dejando claras las secciones despriorizadas (assessments/tests y gestión multiusuario) para iteraciones futuras.

- **Métricas de Entrega**
  - **Iteraciones totales:** 8
  - **Story Points elegidos:** 140
  - **Story Points completados:** 138
  - **Tickets cerrados:** 41
  - **Distribución aproximada por usuario:** 50 SP cada uno

- **Lecciones Aprendidas y Mejora de Procesos**
  - **Refinamiento de historias:** subdividimos grandes tareas, detallamos criterios de aceptación y dependencias.
  - ***Backlog Grooming* continuo:** revisamos y repriorizamos todas las historias antes de cada iteración, alineándose con el valor de negocio.
  - **Ceremonias optimizadas:** planning, daily, review y retro incorporaron métricas de rendimiento y reporte de bloqueos.
  - **Estandarización de plantillas de tickets:** inclusión obligatoria de descripción, dependencias y criterios de prueba.
  - **Flexibilidad frente a cambios de diseño:** adaptamos la implementación conforme *Figma* evolucionó, creando nuevas historias cuando surgieron features imprevistas.

- **Demo Final con el Cliente**
  En la presentación se mostraron:
  - Onboarding y autenticación.
  - Publicación y gestión de vacantes.
  - *Matching* de candidatos con vacantes y su visualización.
  - Navegación principal y datos iniciales desde seeds.

- **Conclusión y Próximos Pasos**
  Damos por cumplido todo lo establecido para Milestone 1. Gracias a las mejoras en estimación, *groomings* y refactors de arquitectura, estamos en posición de arrancar el Milestone 2 con un *backlog* limpio, criterios de aceptación claros y ceremonias consolidadas.

16.11. Evidencia - Informes por Milestone (Checkpoint y Final)

La siguiente sección contiene a modo de evidencia los documentos entregados como informe según instancia por Milestone. No se incluyen todos los informes debido a su extensión, se busca evidenciar las mismas y la información que estas contenían.

Resumen de Avance - Milestone 1 (Checkpoint)

Durante este primer milestone, hemos trabajado en la implementación de funcionalidades clave para la plataforma, enfocándonos en el onboarding de Compañías y Candidatos, autenticación y manejo de sesiones de ambos, así como el Setup de ambas partes y la creación de vacantes de trabajo y por último pero no menos importante la generación de Embbedings y el *matching* de candidatos y vacantes.

Progreso alcanzado

- **Onboarding de empresas y candidatos**: Creación de cuentas, configuración inicial y carga de datos en la base.
- **Autenticación y manejo de sesiones**: Implementación de login, gestión de tokens y redireccionamiento automático según permisos.
- **Navegación y estructura general**: Configuración del menú principal y diseño de la arquitectura para optimizar el desarrollo futuro.
- **SetUp inicial de entidades clave**: Se avanzó en la infraestructura que permitirá a los usuarios gestionar sus datos dentro de la plataforma.
- **Creación de Vacantes de trabajo:** Se realizó una primera versión de la creación de vacantes incorporando las mayores partes de esta
- **Generación y guardado de *Embeddings***: Se implementó la generación de *embeddings* de tanto el candidato como la vacante
- ***Matching*:** Se implementó una primera versión del modelo de *Matching* mediante *embeddings*
- **Investigación del modelo de *Matching*:** Hemos investigado y analizado diferentes configuraciones y variaciones para mejorar el modelo de *matching* considerando nuevas tecnologías y estrategias

Desafíos y mejoras

- **Curva de aprendizaje con Next.js**: Hemos tenido algunos desafíos con la configuración y optimización
- **Revisión de estimaciones**: Algunos tickets resultaron más complejos de lo esperado, lo que nos llevó a mejorar la definición y planificación de tareas.
- **Optimización del proceso de trabajo**: Se implementaron sesiones de *grooming*, revisión de *backlog* y planificación detallada de las próximas iteraciones.
- **Definición de funcionalidades:** nos hemos topado con grandes cambios de lo originalmente planeado debido al gran crecimiento de nuestro punto base, el *Figma*,

por lo que hemos establecido nuevos procesos para la modificación e incorporación de Funcionalidades

Próximos pasos

- En lo que queda del milestone apuntamos a una integración del proceso de creación de vacantes y candidatos y el *matching*
- Apuntamos a mejorar la visualización de estos resultados en la consola.
- A su vez estamos realizando procesos para mejorar la experiencia de usuario general dentro de la plataforma
- Continuaremos afinando el proceso de estimaciones y priorización para mantener un desarrollo más predecible.

Resumen de Avance - Milestone 2 (Checkpoint)

Durante este primer tramo de Milestone 2 se consolidaron funcionalidades que amplían el sistema de *matching* y refuerzan los flujos operativos de la plataforma. Se incorporaron nuevas herramientas para gestión de vacantes y candidatos, así como mejoras generales en la experiencia de uso.

Avances Principales:

- Se profundizó en el sistema de *matching* incorporando criterios adicionales y priorización.

- Se agregaron nuevas funcionalidades relacionadas con el manejo de vacantes:

    - Edición de vacantes.
    - Gestión de estados (Draft, Active, Inverviewing, …).
    - Rechazo y aceptacion del candidatos y cambios de estado.

- Se desarrollaron herramientas complementarias:
    - Sistema de entrevistas y puntajes.
    - Comentarios internos.
    - Generador de resúmenes de candidato con IA
    - Generador de explicación del resultado del *matching* con IA
    - Notificaciones al candidato ante cambio de estados
    - Comunicación de la empresa con el candidato

- Se completaron funcionalidades de edición de información y mayor personalización.
- Se mejoró la visualización de resultados de *matching*.
- Se implementó el registro de métricas, traces, alertas y logging

Progreso en Testing:

- Se realizaron sesiones de testing funcional e integración, donde se detectaron errores menores y se registraron oportunidades de mejoras.

Próximos Pasos:

- Avanzar con las validaciones de usuario.
- Finalizar el despliegue completo de la plataforma.
- Profundizar la visualización de resultados y performance del sistema.
- Consolidar el cierre funcional de los módulos clave.

16.12. Evidencia - Notas por Iteración

A continuación en la siguiente sección se listan a modo de evidencia algunas de las notas tomadas por iteración, debido a su extensa cantidad no se incluyen todas, pero estas sirven a modelo de mostrar la información contenida en ellas.

Iteración 1

- Resumen:Scaffolding y SignUps
- Cantidad de Tickets: -
- Cantidad de Story Points Planeados: 25
- Cantidad de Story Points Completados: 13
- Cantidad de Story Points Pendientes: 12
- Comentarios retro:
    - Surgen complicaciones con Next
    - Surgen complicaciones con definición de tickets, esos resumen features muy específicas
    - Esto llevó a issues subestimados
    - Se llevó la primera meet del Milestone con Go Grow donde se discute similitud con figma y resultados esperados
- Aprendizajes:
    - Mayor generalidad en tickets a veces es necesaria
    - Es necesario definir proceso para identificación de fallas y posibles mejoras
    - Conocimientos en Next Js
- Medidas para la próxima iteración:
    - Se plantean proceso de identificación especificación y estimación de fallas y mejoras
    - Se establecen procesos de registro por Iteración
    - Se establecen sesiones de grooming, revisión de backlog y próxima Iteración así como de priorización
    - Se establecen meet definidas para revisión de avances
    - Se establece sesion de planificación del proyecto

Iteración 2

- Resumen: SetUp Compañía, Envío de notificaciones y refactor estructura

- Cantidad de Tickets: 10
- Cantidad de Story Points Planeados: 36
- Cantidad de Story Points Completados: 29 (26 + uno movido)
- Cantidad de Story Points Pendientes: 7
- Comentarios retro:
    - Se agregan 2 issues, uno de estructura y un bug
    - Se mueve un ticket de Iteración a 3 por no estar terminado
    - Redefinición de estructura del proyecto
    - Se detectan tantos errores como mejoras
    - Se agranda estimación para tickets de notifications
    - Equipo incompleto
- Aprendizajes:
    - Mejoramos medidas de detección de bugs
    - Mejoramos medidas de registro de mejoras y bugs
- Medidas para la próxima iteración:
    - Concentrarnos en valor para el cliente

Iteración 3

- Resumen: Candidate Set-up, Post a Job
- Cantidad de Tickets: 9
- Cantidad de Story Points Planeados: 38 (26 iniciales más tareas extras)
- Cantidad de Story Points Completados: 11 (más 5 de un ticket en progreso)
- Cantidad de Story Points Pendientes: 19 (más 3 de un ticket en progreso)
- Comentarios retro:
    - Dos personas durante la Iteración
    - Colaboración en un mismo ticket
    - US que requieren otros elementos, como manejo de token, navegación en el menu, layouts, carga de datos dinámica
- Aprendizajes:
    - Debemos de prestar mayor atención a los pre-requerimientos de algunas US para evitar sorpresas en el desarrollo y la estimación de estas
- Medidas para para la próxima iteración:
    - Cuidar más los detalles de los tickets para hacer una mejor estimación

    - Acotar el desarrollo a lo que el ticket especifica, si se necesitan muchas otras páginas, endpoint, etc, podemos mockear cosas y crear nuevos tickets que tengan su propia estimación.

Iteración 4

- Resumen: Post a Job, Generación de embeddings
- Cantidad de Tickets: 5
- Cantidad de Story Points Planeados: 27 (26 mas Tartea extra)
- Cantidad de Story Points Completados: 27
- Cantidad de Story Points Pendientes: 0
- Comentarios:
    - Dos personas durante la iteración
    - Mini sesión de testing de integración
    - Organización de estructura de datos
    - Para la siguiente iteración 5 nos dimos cuenta de la falta de varias historias de usuario, las creamos y también editamos otras. Esto se debe a que en el principio cuando hicimos las historias de usuario, nos basamos en una versión inicial del figma (que nos dieron al principio del proyecto), pero que luego de unas semanas evolucionó mucho y aparecieron features y US que no habíamos creado.
- Aprendizajes:
    - Empezamos a aprender nuestro ritmo y capacidad de trabajo
- Medidas para la próxima iteración:
    - Continuaremos concentrandons en estimaciones y priorizando valor para el cliente

Iteración 5

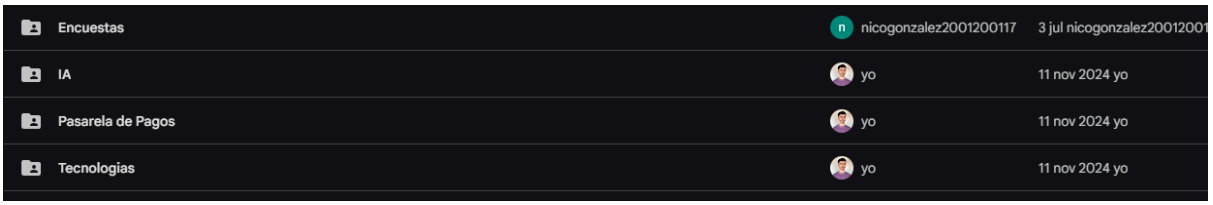
- Resumen: Más investigación matching (herramientas y técnicas), listado de vacantes y candidatos
- Cantidad de Tickets: 7
- Cantidad de Story Points Planeados: 22 + (26 con tareas extras)
- Cantidad de Story Points Completados: 23
- Cantidad de Story Points Pendientes: 3
- Comentarios:

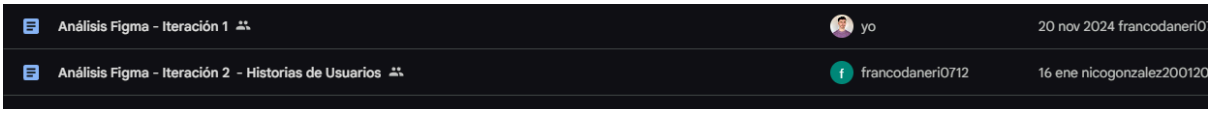
    - Se agrega error handler
    - Se agrega candidate setup refactor
    - se re-estimo una US por similitud frontend
    - Se restimo una US de 5 a 3
- Aprendizajes:
- Medidas para para la próxima iteración:

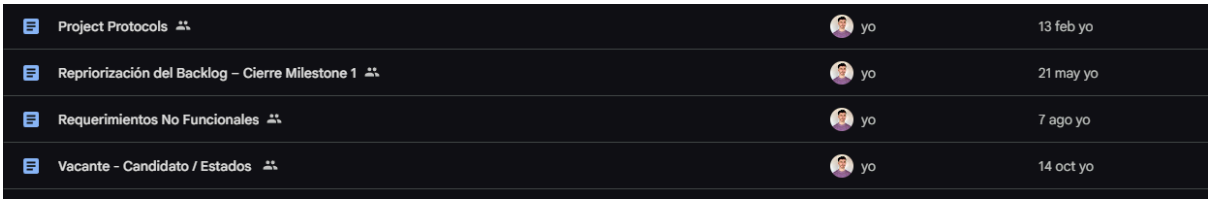
16.13. Evidencia - Documentación continua

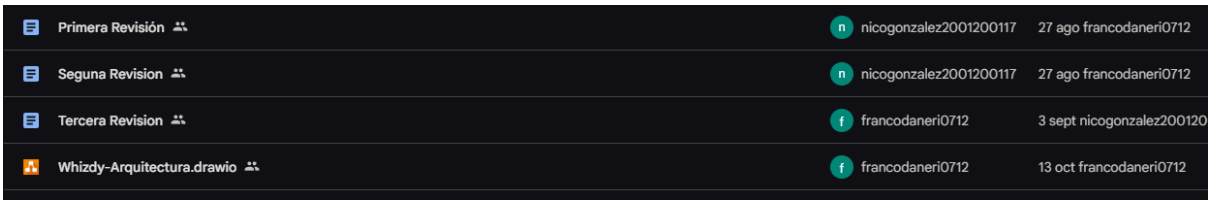
En la siguiente sección se listan a modo de evidencia de documentación continua algunos de los documentos generados. Por obvias razones de cantidad de las mismas se limita la evidencia a un único documento, en este caso evidencia de documentación sobre la decisión tomada para el flujo de los candidatos y vacantes. Tener en cuenta que estos se trataban de documentos internos no desarrollados para su uso directo en la documentación final.

Esta forma de trabajo de documentación continua llevó a una gran cantidad de documentos generados.





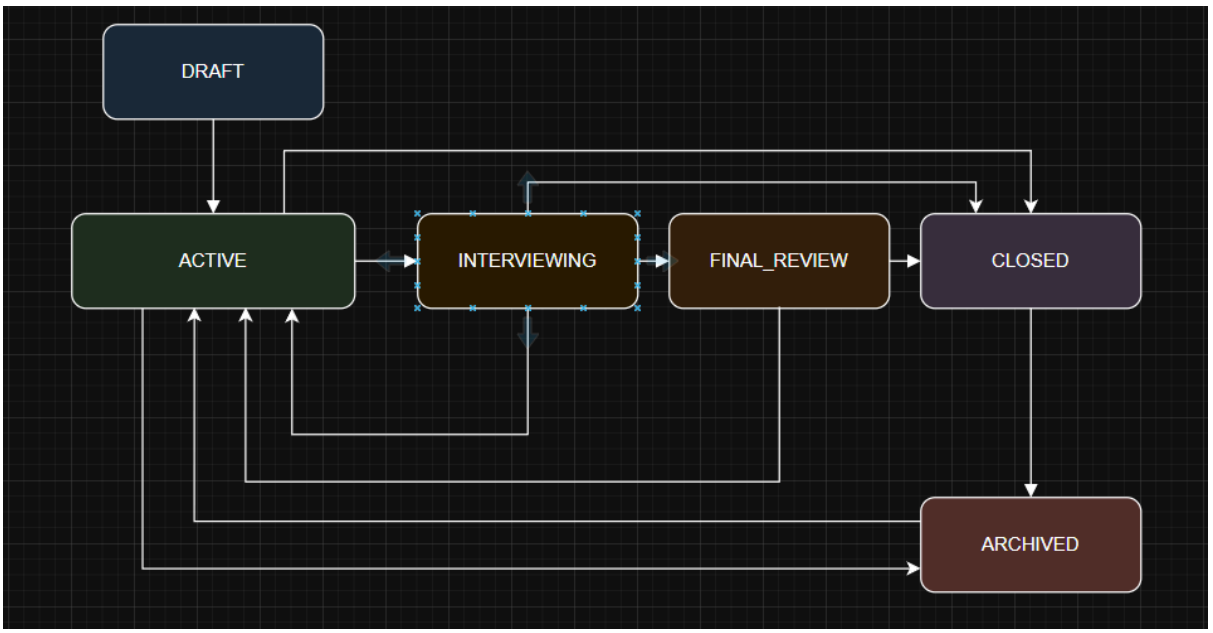


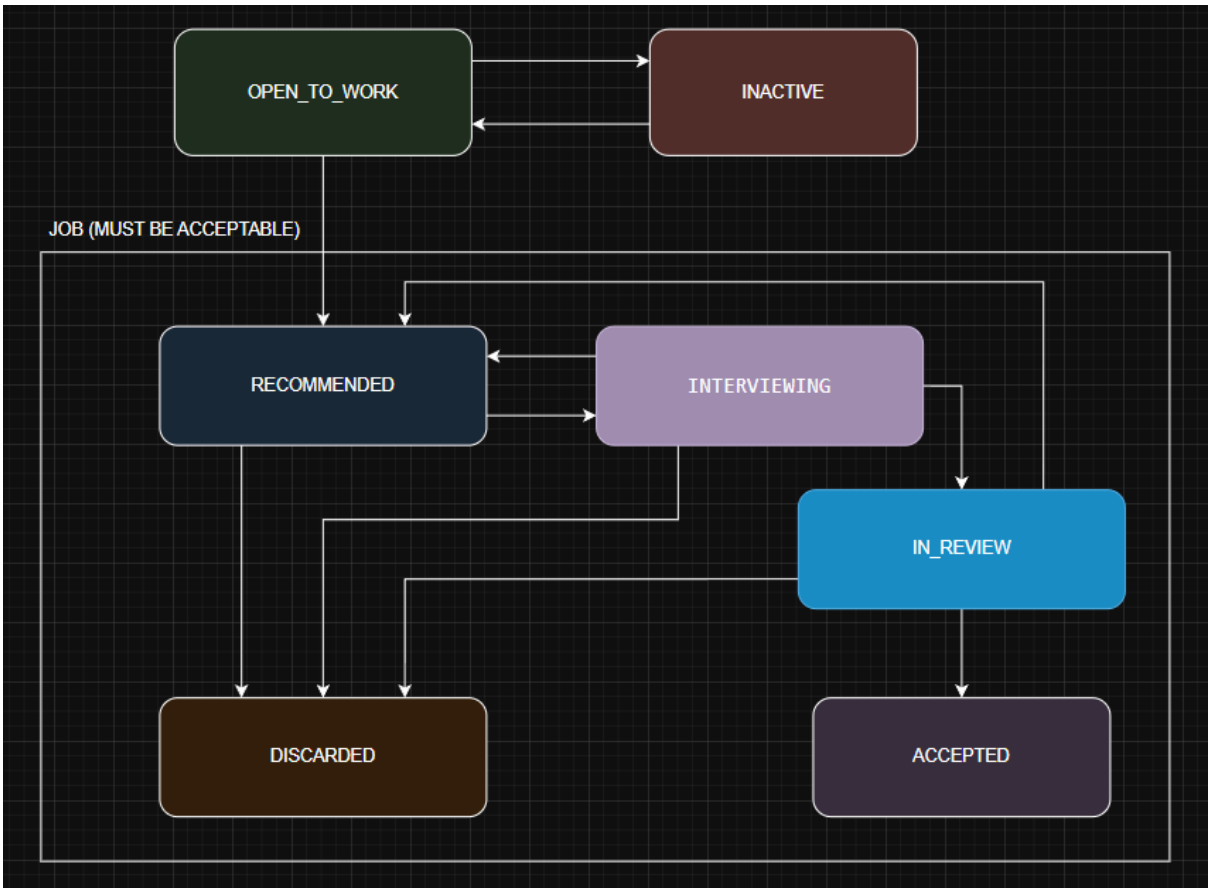


Documento Vacantes - Candidato / Estados:

Estado Vacante	Descripción	Opciones
DRAFT	La vacante no está publicada. Solo visible para la empresa. Editable por la empresa.	ACTIVE
ACTIVE	La vacante está activa y recibiendo nuevos *matchings* automáticamente. Los candidatos pueden demostrar interés. La empresa puede descartar candidatos.	ARCHIVED CLOSED IN_REVIEW
IN_REVIEW	No se solicitan más recomendaciones Se ejecuta un último *matching* con los resultados del *matching* más el resultado de la interview. Es momento de tomar una decisión final. Si se vuelve a active todos los candidatos no descartados pasan a RECOMMENDED No se puede editar. No se aceptan nuevos candidatos.	ACTIVE CLOSED
CLOSED	La vacante fue cerrada. Todos los candidatos no ACCEPTED se pasaran a DISCARDED Los demás fueron descartados. No se puede editar. Solo visible para la empresa. No se aceptan nuevos candidatos.	ARCHIVED
ARCHIVED	La vacante está guardada como histórico. Si se vuelve a active todos los candidatos no descartados pasan a RECOMMENDED No se puede editar. Solo visible para la empresa. No se aceptan nuevos candidatos.	ACTIVE

Estado Candidato	Descripción	Opciones
INACTIVE (GLOBAL)	El candidato no desea ser considerado para vacantes. No participa en nuevos *matchings* Mantiene el historial de vacantes previas	OPEN_TO_WORK
OPEN_TO_WORK (GLOBAL)	El candidato está disponible para ser considerado en vacantes nuevas.	INACTIVE RECOMMENDED
RECOMMENDED (PER JOB)	El sistema lo recomienda para una vacante activa Requiere: OPEN_TO_WORK+ **NO** DISCARDED + ubicación válida + porcentaje mínimo	INTERVIEWING REJECTED
INTERVIEWING (PER JOB)	El candidato no fue DISCARDED Se le agenda o realiza entrevista. Se puede descartar aún Volverá a RECOMMENDED si la vacante se pasa a ACTIVE	IN_REVIEW REJECTED RECOMMENDED
IN_REVIEW (PER JOB)	El candidato fue entrevistado y ahora está siendo evaluado en la decisión final. Volverá a RECOMMENDED si la vacante se pasa a ACTIVE	ACCEPTED REJECTED RECOMMENDED
ACCEPTED (PER JOB)	El candidato fue elegido para ocupar la vacante.	OFFERED
REJECTED (PER JOB)	El candidato fue descartado por la empresa para esta vacante. No vuelve a ser considerado para esta vacante	-
OFFERED		OFFER_REJECTED HIRED
OFFER_REJECTED		-
HIRED		-


DRAFT
ACTIVE
INTERVIEWING
FINAL_REVIEW
CLOSED
ARCHIVED


OPEN_TO_WORK
INACTIVE
JOB (MUST BE ACCEPTABLE)
RECOMMENDED
INTERVIEWING
IN_REVIEW
DISCARDED
ACCEPTED

16.14. Documento re-priorización *Backlog* - Cierre Milestone 1

El siguiente documento fue desarrollado y entregado al cliente durante el cierre del Milestone 1 como forma de documentar de forma clara la priorización planteada y las bases de la misma

Introducción

Al cierre de la primera fase del proyecto (Milestone 1), se realizó una instancia de demostración y puesta en común con el cliente. Esta demo sirvió no solo para mostrar avances en el desarrollo, sino también para abrir un espacio de análisis conjunto sobre el estado del proyecto, el crecimiento del *backlog* y la planificación futura. En este contexto, se propuso una repriorización de funcionalidades con el fin de garantizar un MVP sólido, enfocado y de alto valor.

Análisis del Estado del Proyecto

Durante la instancia se presentaron los avances alcanzados, el estado de las funcionalidades y una visión general del trabajo realizado. Entre los puntos más destacados:

- Se evidenció una evolución significativa del *backlog* en tamaño y complejidad.
- Se realizó un *backlog grooming* completo posterior a los cierres de iteración, logrando mayor claridad sobre el esfuerzo real que implica cada feature.
- Esta revisión permitió visualizar que el alcance actual del proyecto excede significativamente las estimaciones iniciales.

A raíz de esto, se planteó un enfoque más estratégico: priorizar aquello que aporta mayor valor al MVP, orientando los esfuerzos hacia la diferenciación del producto y la calidad de la solución entregada.

Enfoque Propuesto

Se transmitió al cliente que el objetivo principal es posicionar la plataforma como una herramienta diferencial de *scouting* potenciada por inteligencia artificial. En este sentido, se definieron los siguientes pilares:

- Concentrarse en el core del sistema: el algoritmo de *matching* inteligente.
- Apostar a una solución robusta y escalable desde el punto de vista técnico y arquitectónico.
- Cumplir con los requerimientos funcionales y no funcionales con un alto estándar de calidad.
- Evitar construir funcionalidades genéricas que no aporten al diferencial del producto.
- Analizar continuamente qué funcionalidades conviene desarrollar y cuáles integrar mediante terceros.

Funcionalidades Repriorizadas

En función de los pilares anteriores, se propuso al cliente bajar la prioridad de las siguientes funcionalidades:

1. Multicuentas para Empresas

Descripción: Permitir que una cuenta de empresa tenga múltiples usuarios con distintos roles (ej. reclutadores, administradores).

Razón de Repriorización:

- No es indispensable para el funcionamiento del MVP.
- El uso actual de una cuenta administradora principal es suficiente en esta etapa.
- Agregar esta funcionalidad implicaría una complejidad adicional no necesaria al corto plazo.

2. Pasarela de Pagos

Descripción: Implementar un sistema de pagos para habilitar planes freemium/premium.

Razón de Repriorización:

- Representa un desafío técnico importante.
- No aporta valor inmediato al core de la plataforma.

- En esta etapa, se prevé un uso interno de la plataforma, sin necesidad de monetización inmediata.
- Se prioriza dedicar ese tiempo a otros elementos del *backlog* más estratégicos.

3. Psicometría

Descripción: Incluir formularios o tests psicológicos para evaluar aspectos de personalidad o culturales de los candidatos.

Razón de Repriorización:

- Existen herramientas externas especializadas que resuelven esto de forma eficiente.
- No se busca competir en este aspecto, sino integrarse con soluciones existentes en caso de ser necesario.
  El foco está en el *matching*, no en replicar funcionalidades genéricas de evaluación psicométrica.

4. Evaluaciones Técnicas (Assessments)

Descripción: Crear tests técnicos asociados a cada vacante para evaluar las competencias del candidato.

Razón de Repriorización:

- Hay soluciones en el mercado que realizan esto de manera especializada.
- El esfuerzo para desarrollar esta funcionalidad es considerable y se aleja del core diferencial del producto.
- Se plantea integrar herramientas externas, o incluso permitir la carga manual de resultados por parte de la empresa, en lugar de desarrollar un sistema completo desde cero.

Aprobación del Cliente

El planteo fue presentado al cliente como una propuesta sujeta a su validación, considerando que se trata de un software a medida. La respuesta fue positiva:

- Las primeras dos funcionalidades (multicuentas y pasarela de pagos) fueron completamente entendidas y aceptadas como no prioritarias.

- Las funcionalidades relacionadas a psicometría y assessments también fueron aceptadas como postergables, si bien se reconoció que eran de su interés. El cliente entendió que, dada la dimensión del proyecto y del equipo, priorizar estos puntos podría comprometer la calidad y el cumplimiento del MVP.

Conclusión

Las funcionalidades mencionadas seguirán presentes en el *backlog*, pero han sido oficialmente despriorizadas. Esto implica que no formarán parte de las entregas del corto plazo, y su implementación quedará sujeta a futuras iteraciones, en función de la evolución del producto y los recursos disponibles.

Este enfoque permitirá al equipo concentrarse en un MVP bien definido, con una solución diferencial, robusta y validada desde el punto de vista técnico y estratégico.

16.15 Evidencia - Evidencia generación y documentación de procesos del proyecto

A continuación se adjunta documento que contiene algunos de los protocolos y procesos iniciales que el equipo fue definiendo a lo largo del proyecto, estos con el paso del tiempo se fueron modificando y adaptando a las necesidades del equipo y el proyecto, pero esta fue la primer versión del mismo y evidencia el trabajo realizado sobre los procesos del proyecto en sí.

Whizdy *Project Protocols*

Pull Requests & Testing

- Todos los Pull Requests (PRs) deben ser aprobados por al menos dos personas antes de fusionarse.
- Todos los cambios deben incluir pruebas a nivel de lógica de negocio para garantizar estabilidad y calidad.

- Cuando se hace un PR, se envía el enlace por el canal de Discord, indicando cuándo se revisó, se aprobó y se fusionó.
- Usamos ESLint para mantener la calidad del código.
- Usamos GitHub Actions para compilar y correr tests; si las pruebas no pasan, no se fusiona el código.
- Se deben cumplir los criterios de aceptación antes de marcar una tarea como completada.

Gestión de iteraciones y Milestones

- Final de Iteración:
    - Documentar de forma clara cada cambio, consideración o error detectado en la página de cambios.
    - Realizar una reunión para definir y revisar la siguiente iteración.
    - Anotar en un documento los resultados, experiencias, comentarios, cambios grandes, notas y cambios en la gestión.
- *Backlog Grooming*:
    - Al final de cada iteración se realizará una sesión de *grooming* del *backlog* para mantenerlo actualizado y priorizado.
- Inicio de Milestone:
    - Se estiman en grupo las User Stories (US) que entran en el milestone utilizando Poker Estimation.
- Mitad de Milestone: Checkpoint Review
    - Se organiza una reunión para evaluar el progreso, discutir próximos pasos y posibles cambios en el roadmap.
- Final de Milestone:
    - Se realiza una retrospectiva del milestone para identificar mejoras y aprendizajes.
    - Se lleva a cabo una demo con GoGrow para mostrar avances.
- Seguimiento Semanal:
    - Se revisa semanalmente el avance de los tickets y se ajusta la planificación de la iteración según sea necesario.
    - Cualquier situación borde, error o particularidad detectada en el sistema se reporta en el documento de posibles cambios.

    - Al final de la iteración, estos reportes se convierten en tickets según corresponda.

Comunicación y Coordinación

- **Uso de WhatsApp:**
    - Toda comunicación, solicitud de reuniones y coordinación rápida se hace a través de WhatsApp.
    - **Actualización de Tickets:**
        - Los estados de los tickets deben actualizarse a medida que se avanza en el desarrollo, asegurando que siempre reflejen el estado actual.

Reuniones y Comunicación con GoGrow

- Revisión Trimestral con GoGrow:
    - Cada tres iteraciones se realiza una reunión con el cliente para revisar avances, validar conformidad y discutir modificaciones para los próximos iteraciones.

Filosofía del Proyecto

- Se prioriza siempre la entrega de valor para el usuario final.

16.16 Documento cambio stack tecnológico

En la siguiente sección se encuentra el documento compartido con el cliente y donde se documentó el cambio de stack tecnológico propuesto, razones y decisión final.

De Ruby on Rails a JavaScript (Node.js)

Durante esta primera instancia del proyecto, a medida que el equipo profundiza más en la definición de las Historias de Usuario, su implementación y hemos realizado las investigaciones pertinentes para las integraciones necesarias para el proyecto, se ha estado evaluando el stack tecnológico propuesto para el proyecto.
Tras un análisis detallado, hemos decidido realizar un cambio en una de las tecnologías de Stack Tecnológico destinadas para el backend, pasando de Ruby on Rails a Node.js con JavaScript, mientras mantenemos Next.js en el *frontend* como inicialmente se planteó.

A continuación, detallamos las razones detrás de esta decisión, considerando las dificultades actuales y las necesidades específicas del proyecto.

Experiencia con el Stack Actual y Tecnologías Clave

El equipo no tiene experiencia previa en Ruby on Rails, lo que ya de por sí representa un desafío importante. A esto se suman las complejidades del proyecto, como las integraciones con modelos de IA externos (OpenAI, Gemini, etc), Google Calendar, LinkedIn y pasarelas de pago, todas tecnologías con las que no hemos trabajado antes.

Por lo que además de aprender Ruby y su framework, debemos enfrentarnos a nuevas integraciones tecnológicas que son clave para el proyecto. Esto incrementa significativamente la curva de aprendizaje, lo que inevitablemente afectará tanto el tiempo de desarrollo como la calidad del producto final. Sin experiencia previa, existe un riesgo real de cometer errores de implementación y, en última instancia, de ofrecer una experiencia de usuario deficiente.

Optimización del Tiempo de Desarrollo y la Calidad del Producto

Nuestro objetivo es entregar una plataforma de alta calidad dentro del plazo estipulado. Sin embargo, la investigación e integración de modelos de IA y otras herramientas externas, como OpenAI y pasarelas de pago, ya ha ralentizado el progreso del proyecto. Aunque realizamos algunas pruebas preliminares con Ruby, los resultados fueron lentos, y la curva de aprendizaje del lenguaje y framework solo incrementaría las demoras en el futuro, afectando tanto la eficiencia como la calidad del desarrollo.

Si además debemos aprender Ruby desde cero, el tiempo de desarrollo se extendería aún más, comprometiendo la calidad final del producto.

Por otro lado con Node.js y JavaScript, tecnologías con las que ya estamos familiarizados, optimizaríamos los tiempos de desarrollo y podríamos centrarnos directamente en las funcionalidades clave del proyecto, como el algoritmo de *matching* de candidatos y las integraciones con OpenAI y Google Calendar, sin las distracciones de aprender un nuevo lenguaje y framework. Esta decisión no solo mejoraría nuestra agilidad, sino que también mejoraría la calidad del producto, ya que aplicaríamos nuestras habilidades previas en Node.js para reducir errores y retrasos.

Ventajas de Usar Node.js en el Backend

Tras investigar a fondo Ruby y Node.js, y considerando nuestra experiencia previa con Node.js, consideramos que este stack no solo conserva las ventajas de Ruby on Rails, sino que, según nuestras observaciones, ofrece mejoras en rendimiento, escalabilidad y flexibilidad. Nuestra experiencia con Node.js y su ecosistema nos lleva a pensar que está mejor adaptado para las necesidades actuales de integración con APIs externas y rendimiento en tiempo real, además de ofrecer una escalabilidad más eficiente a medida que el proyecto crece.
Específicamente nos parece importante destacar los siguientes puntos:

- Unificación del Lenguaje: Usar JavaScript en todo el stack (*frontend* y *backend*) simplifica la comunicación entre ambas partes y reduce la complejidad, eliminando la necesidad de gestionar dos lenguajes diferentes.

- Ecosistema Robusto: Ecosistema maduro y herramientas bien soportadas.

Conclusión

El cambio de Ruby on Rails a Node.js es una decisión fundamentada en nuestra experiencia previa con JavaScript y las necesidades específicas del proyecto. Aunque el stack propuesto inicialmente incluía tecnologías que desconocemos, como Ruby y Next.js, hemos evaluado el impacto que esto tendría en el tiempo de desarrollo y la calidad del producto.

Node.js nos ofrece ventajas en cuanto a rendimiento, escalabilidad y flexibilidad, alineándose mejor con nuestras habilidades y las integraciones clave, como OpenAI y Google.
Al mantener PostgreSQL como base de datos y Next.js en el frontend, seguimos el stack original propuesto por el cliente, pero optimizamos el proceso de desarrollo y garantizamos un producto más eficiente y escalable.

En resumen, el paso a Node.js es una decisión estratégica que consideramos correcta y bien fundamentada, dadas las circunstancias del equipo y los requerimientos del proyecto.

Next.JS fullstack

Una vez comunicado al cliente el cambio, este nos dio la posibilidad de usar NextJS, no solo para el frontend si no que para el backend igualmente.

Motivaciones para el cambio:

Simplificación y Optimización

El mantener un backend separado en Node.js implica gestionar múltiples proyectos, despliegues y configuraciones, lo que añade complejidad y tiempo al desarrollo. Con Next.js como fullstack, podemos concentrar todas las funcionalidades (frontend y backend) en un único entorno, reduciendo los puntos de fricción.

Enfoque en la Rapidez

El objetivo del proyecto es lanzar una plataforma funcional sin comprometer la calidad pero a la vez permitiéndonos iterar rápido e introducir cambios. Next.js ofrece herramientas preconfiguradas, como API Routes y soporte nativo para conexiones a bases de datos, que permiten implementar tanto el frontend como la lógica backend sin la necesidad de configurar un proyecto Node y levantar un API con sus configuraciones.

Curva de Aprendizaje

Nuestro equipo ya tiene experiencia trabajando con JavaScript y *React*, tecnologías fundamentales en Next.js. Evitamos así invertir tiempo aprendiendo y configurando frameworks adicionales, como Express o NestJS, necesarios para un backend separado.

Comparativa entre Backend + Frontend Separados y Next.js Fullstack

1. **Backend y frontend separados:**

Ventajas:

- Escalabilidad: el backend puede manejar una mayor carga y escalar de manera independiente al frontend.
- Separación de responsabilidades: mejor modularidad del código y permite que equipos distintos trabajen en paralelo.
- Flexibilidad: permite utilizar diferentes tecnologías en el backend según las necesidades futuras. Incluso se parar el backend en microservicios.

Desventajas:

- Complejidad inicial: mayor configuración y coordinación entre backend y frontend.

- Despliegue separado: mayor tiempo y esfuerzo necesario para mantener ambos entornos en producción.
- Desarrollo más lento: implementar la comunicación entre backend y frontend (APIs REST o GraphQL) consume tiempo adicional.

2. **Next.js como Fullstack**

Ventajas:

- Desarrollo rápido: API Routes permiten manejar la lógica backend de manera inmediata, sin necesidad de configurar un servidor externo.
- Integración directa: El frontend y el backend comparten un mismo entorno, eliminando la necesidad de gestionar una comunicación API separada.
- Menor complejidad: Una sola base de código simplifica el mantenimiento y reduce la probabilidad de errores de integración.
- Optimización nativa: Next.js está diseñado para manejar SSR, ISR y APIs, proporcionando una solución unificada para aplicaciones modernas.
- Despliegue sencillo: usar NextJS nos permite desplegar toda la aplicación muy fácilmente en entornos como Vercel

Desventajas:

- Escalabilidad limitada: si el proyecto crece significativamente, el backend integrado puede volverse un cuello de botella de mantenibilidad y performance.
- Dependencia del framework: toda la lógica del backend queda acoplada a Next.js, dificultando migraciones futuras.
- Carga del servidor: Manejar tanto el frontend como el backend en el mismo servidor puede afectar el rendimiento en escenarios de alta concurrencia. Sobre todo si realizamos operaciones pesadas en el backend, para esto podríamos separar un pequeño backend para las operaciones de IA.

16.17 Datos de prueba y resultados de la PoC

16.17.1 Datos de prueba de la primera iteración de la PoC

Vacantes:

Vacante 111: Middle Software Engineer (Uruguay)

"Middle Software Engineer (Uruguay) - 2+ years of experience in JavaScript and Node.js. Experience with MongoDB required. Knowledge of Kafka, Docker, and Angular is a plus."

Vacante 222: Senior Software Engineer (Buenos Aires)

"Senior Software Engineer (Buenos Aires) - 5+ years of experience in Python and Go. Experience with MongoDB required. Experience with React is a plus."

Vacante 333: Frontend Developer (Remote)

"Frontend Developer (Remote) - Strong background in HTML, CSS, and JavaScript. Experience with modern frameworks such as React or Vue is essential. Familiarity with UX/UI design principles is beneficial."

Vacante 444: Marketing Specialist

"Marketing Specialist - 3+ years of experience in digital marketing and content creation. Proficiency in SEO, social media strategies, and email marketing platforms. Strong writing and analytical skills are required."

Candidatos

Candidato 111

"4 years of experience in JavaScript and Node.js. Extensive experience with MongoDB and Redis. Skilled in Docker and Kubernetes for container management. Familiar with Kafka for real-time data processing."

Candidato 222

"6 years of experience in software engineering with a focus on backend development using Python and Go. Proficient with MongoDB and PostgreSQL. Some exposure to frontend technologies like React and Vue."

Candidato 333

"Frontend developer with 3 years of experience in HTML, CSS, and JavaScript. Worked extensively with React and Vue for interactive web applications. Understanding of UX/UI principles and design tools like Figma."

Candidato 444

"Digital marketing expert with over 5 years of experience. Skilled in SEO, content creation, and social media marketing strategies. Knowledge of Google Analytics and experience with email marketing platforms."

Resultados de *matching*

Vacante 111 (*Middle Software Engineer - JavaScript/Node.js*)

Modelo: *all-MiniLM-L6-v2* (384 dimensiones)

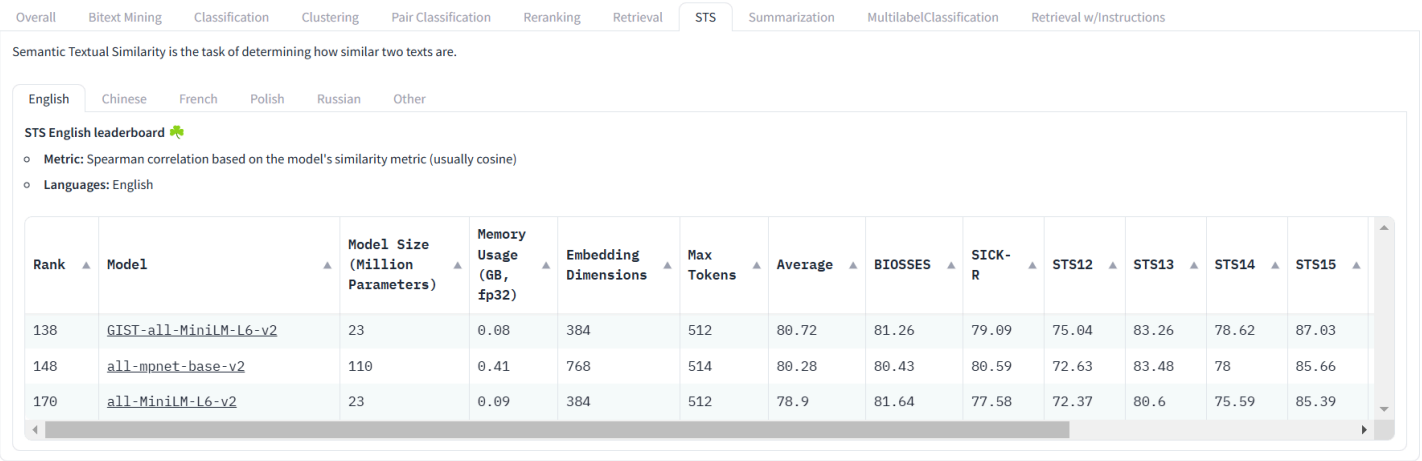
Posición	Candidato ID	Distancia Coseno
1	222	0.556
2	111	0.593

Modelo: *all-mpnet-base-v2* (768 dimensiones)

Posición	Candidato ID	Distancia Coseno
1	222	0.367
2	111	0.385

Nota: En la métrica de distancia coseno, valores menores indican mayor similitud. El Candidato 222 (*Python/Go*) fue rankeado por encima del Candidato 111 (*JavaScript/Node.js*) por ambos modelos, a pesar de que este último poseía el *stack* tecnológico específicamente requerido por la vacante.

Comparación de modelos



Overall | Bitext Mining | Classification | Clustering | Pair Classification | Reranking | Retrieval | STS | Summarization | MultilabelClassification | Retrieval w/Instructions

Semantic Textual Similarity is the task of determining how similar two texts are.

English | Chinese | French | Polish | Russian | Other

STS English leaderboard

- **Metric:** Spearman correlation based on the model's similarity metric (usually cosine)
- **Languages:** English

Rank	Model	Model Size (Million Parameters)	Memory Usage (GB, fp32)	Embedding Dimensions	Max Tokens	Average	BIOSSES	SICK-R	STS12	STS13	STS14	STS15
138	GIST-all-MiniLM-L6-v2	23	0.08	384	512	80.72	81.26	79.09	75.04	83.26	78.62	87.03
148	all-mpnet-base-v2	110	0.41	768	514	80.28	80.43	80.59	72.63	83.48	78	85.66
170	all-MiniLM-L6-v2	23	0.09	384	512	78.9	81.64	77.58	72.37	80.6	75.59	85.39

16.17.2 Datos de prueba de la segunda iteración de la PoC

Nota: La segunda iteración de la Prueba de Concepto incluyó pruebas con un *dataset* extendido de 20 vacantes y 30 candidatos para validar el comportamiento del sistema con mayor volumen de datos. Por razones de brevedad, este anexo presenta los casos más representativos utilizados para el análisis detallado documentado en el cuerpo principal.

Vacante 116: Full Stack Developer (San Francisco)

"We are seeking a talented Full Stack Developer to join our innovative team in San Francisco. In this role, you will design, develop, and maintain scalable web applications that provide exceptional user experiences while collaborating with cross-functional teams to define and deliver high-quality solutions. Your responsibilities will include developing and integrating RESTful APIs, optimizing applications for speed and performance, writing clean and maintainable code, conducting code reviews, and staying updated on emerging technologies and best practices. This position requires proficiency in JavaScript, with React.js and Node.js being core technologies, as well as HTML and CSS, at a minimum level of proficiency and ideally at an expert level. Experience with databases such as MongoDB or PostgreSQL is required, with expert proficiency preferred. Familiarity with Docker for containerized application development and experience in CI/CD pipelines is necessary, with expert-level knowledge ideal. A solid understanding of software architecture patterns is important, and experience with cloud platforms like AWS or Azure is a plus. Familiarity with

TypeScript and GraphQL, even at a beginner level, is valuable, though proficient knowledge is preferred. The ideal candidate will also demonstrate strong problem-solving skills, attention to detail, and effective communication, with adaptability and a collaborative mindset being critical to thrive in this dynamic role. A bachelor's degree in Computer Science or a related field is required, along with a minimum of four years of experience in full stack development. In return, we offer a competitive salary, performance bonuses, comprehensive health, dental, and vision insurance, flexible working hours with hybrid remote options, and opportunities for professional development and growth. If you are passionate about solving challenging problems and delivering exceptional user experiences, we would love to hear from you."

Candidatos

Candidato 301

"Full Stack Developer with 5 years of experience building scalable web applications. Proficient in JavaScript, specifically React.js and Node.js, as well as HTML and CSS. Experienced with databases like MongoDB and PostgreSQL. Skilled in Docker and containerized application development. Strong problem-solving abilities and meticulous attention to detail. Familiar with Git for version control and has a solid understanding of software architecture patterns. Also experienced with TypeScript and GraphQL, and has implemented CI/CD pipelines using Jenkins. Familiar with AWS cloud services."

Candidato 302

"Software Engineer with 4 years of full stack development experience. Proficient in React.js, Node.js, HTML, and CSS. Worked extensively with MySQL and has experience in optimizing database queries. Familiar with Docker and containerization. Strong skills in problem-solving and code optimization. Experienced with Git for version control. Has basic knowledge of TypeScript and has recently started exploring GraphQL. Some exposure to AWS cloud services and CI/CD pipelines."

Candidato 303

"Full Stack Developer with 4 years of experience specializing in front-end technologies. Expert in React.js, HTML, and CSS, with solid experience in Node.js. Worked with PostgreSQL databases and has implemented RESTful APIs. Familiar with Docker and has used Git extensively. Strong attention to detail and a passion for creating exceptional user interfaces. Limited experience with TypeScript but eager to learn. No prior experience with GraphQL or cloud platforms."

Candidato 304

"Experienced Developer with 5 years in full stack development. Proficient in Node.js and React.js, as well as HTML and CSS. Extensive experience with MongoDB and MySQL databases. Skilled in containerized application development using Docker. Strong problem-solving skills and attention to detail. Experienced with Git and has a good

understanding of software architecture patterns. Familiar with CI/CD pipelines and has deployed applications on Azure. Has used TypeScript and GraphQL in previous projects."

Candidato 305

"Full Stack Developer with over 6 years of experience in designing and developing web applications. Highly proficient in JavaScript frameworks including React.js and Node.js. Expert in HTML, CSS, and has extensive experience with MongoDB and PostgreSQL. Skilled in Docker and Kubernetes for containerization and orchestration. Strong understanding of software architecture patterns and design principles. Experienced in setting up CI/CD pipelines using Jenkins and GitLab CI/CD. Proficient with cloud platforms like AWS and Azure. Deep knowledge of TypeScript and GraphQL. Excellent problem-solving skills, attention to detail, and a collaborative team player."

Resultados de *matching*

Resultados del *matching* utilizando el modelo paraphrase-mpnet-base-v2 con métrica de distancia coseno (valores menores indican mayor similitud):

Posición	Candidato ID	Score (Distancia Coseno)
1	305	0.7493
2	301	0.7458
3	304	0.6940
4	302	0.6916
5	303	0.6887

Análisis Comparativo de Habilidades

La siguiente tabla presenta la comparación detallada entre los requisitos de la Vacante 116 y las habilidades de cada candidato:

Requisito	Vacante	Cand. 301	Cand. 302	Cand. 303	Cand. 304	Cand. 305
JavaScript	Required: Proficient, Ideal: Expert	Proficient (5 años, React.js, Node.js)	Proficient (4 años, React.js, Node.js)	Proficient (4 años, React.js, exposición backend)	Proficient (5 años, React.js, Node.js)	Expert (6+ años, React.js, Node.js)

HTML	Required: Proficient, Ideal: Expert	Proficient	Proficient	Proficient	Proficient	Expert
CSS	Required: Proficient, Ideal: Expert	Proficient	Proficient	Proficient	Proficient	Expert
React.js	Required: Proficient, Ideal: Expert	Proficient (Frontend expertise)	Proficient	Proficient (Especializa en frontend)	Proficient	Expert
Node.js	Required: Proficient, Ideal: Expert	Proficient (Backend expertise)	Proficient	Beginner (Exposición básica Node.js)	Proficient	Expert
MongoDB	Required: Proficient, Ideal: Expert	Proficient	No mencionado	No mencionado	Proficient	Expert
PostgreSQL	Required: Proficient, Ideal: Expert	Proficient	Proficient	Proficient	Proficient	Expert

TypeScript	Preferred: Beginner, Ideal: Proficient	Proficient (Experiencia práctica)	Beginner (Conocimiento básico, empezando exploración)	Beginner (Conocimiento limitado, ansioso por aprender)	Proficient	Expert (Conocimiento profundo)
GraphQL	Preferred: Beginner, Ideal: Proficient	Proficient (Experiencia práctica)	Beginner (Conocimiento básico)	Sin experiencia	Proficient	Expert (Conocimiento profundo)
Docker	Required: Proficient, Ideal: Expert	Proficient	Proficient	Familiar (Conocimiento básico)	Proficient	Expert
CI/CD pipelines	Required: Proficient, Ideal: Expert	Proficient (Implementado usando Jenkins)	Familiar (Exposición limitada)	Sin experiencia	Proficient	Expert (Configurado CI/CD con Jenkins y GitLab CI/CD)
Cloud platforms (AWS/Azure)	Preferred: Familiarity	Familiar con AWS	Familiaridad básica con AWS	Sin experiencia	Familiar con Azure	Proficient con AWS y Azure

Problem-solving skills	Required: Proficient, Ideal: Expert	Proficient	Proficient	Proficient	Proficient	Expert
Attention to detail	Required: Proficient, Ideal: Expert	Proficient	Proficient	Proficient	Proficient	Expert
Communication and teamwork	Required: Proficient, Ideal: Expert	Proficient	Proficient	Proficient	Proficient	Expert
Software architecture patterns	Preferred: Proficient	Familiar	No mencionado	No mencionado	Familiar	Proficient

El Candidato 305 demostró la mayor alineación con los requisitos de la vacante, cumpliendo con todas las habilidades requeridas en el nivel ideal (Expert). El Candidato 301 ocupó el segundo lugar con sólida alineación en habilidades básicas pero menor profundidad en requisitos preferidos. Los Candidatos 304, 302 y 303 mostraron decreciente alineación debido a brechas en habilidades requeridas o preferidas.

16.17.3 Datos de prueba de las mejoras post producción

Las pruebas documentadas en este anexo se realizaron utilizando el sistema de *matching* implementado en producción, después de las mejoras técnicas descritas en la sección 8.3.1. A diferencia de las pruebas de concepto previas, estos datos siguen la estructura completa del sistema real, incluyendo vectores de soft skills de 24 dimensiones, metadatos extendidos, y el modelo LoRA fine-tuned de 1024 dimensiones. Por razones de brevedad, se presentan ejemplos representativos de la estructura de datos en lugar del conjunto completo de 5 vacantes y 12 candidatos utilizados en las pruebas.

Ejemplo de vacante: Senior Full Stack JavaScript Developer

La estructura de datos en producción incluye información significativamente más detallada que las pruebas de concepto iniciales. Cada vacante se representa mediante múltiples dimensiones estructuradas:

- **Información básica**: Tipo de empleo (FullTime, Contract, PartTime), modalidad de trabajo (Remote, OnSite, Hybrid), nivel de experiencia (Junior, Mid, Senior), ubicaciones, descripción del puesto, responsabilidades, y beneficios.
- **Habilidades técnicas (hard skills)**: Cada habilidad se especifica con nivel de competencia requerido (Beginner, Intermediate, Proficient, Expert) y si es obligatoria (required: true/false). Por ejemplo: JavaScript - Expert (required), React - Expert (required), Node.js - Proficient (required), TypeScript - Proficient (not required), MongoDB - Intermediate (not required).
- **Habilidades blandas (soft skills)**: Representadas como un vector de 24 competencias agrupadas en cuatro categorías con valores del 1 al 10. Competencias de trabajo (Organization: 4, Productivity: 5, Planning: 4, Quality: 5, Problem-solving: 5, Knowledge: 5). Competencias interpersonales (Leadership: 4, Communication: 4, Collaboration: 4, Teamwork: 4, Motivation: 4, Adaptability: 4). Competencias psicológicas (Determination: 4, Empathy: 3, Openness to Feedback: 4, Calmness: 4, Introspection: 3, Resilience: 4). Competencias cognitivas (Creativity: 4, Critical Thinking: 5, Attention to Detail: 4, Caution: 3, Rationality: 5, Focus: 5).
- **Educación**: Instituciones preferidas, títulos esperados, años de estudio, calificación mínima, y descripción de requisitos educativos.

- **Lenguajes**: Idiomas requeridos con niveles de competencia esperados (Native, Professional Working, Limited Working, Elementary).
- **Experiencia**: Servicios o industrias específicas con nivel de experiencia esperado.
- **Compensación**: Salario base, rango mínimo-máximo, moneda, y frecuencia de pago.
- **Pesos configurables**: Importancia relativa de cada dimensión (basic_info: 0.2, technical: 0.3, soft_skills: 0.2, education: 0.1, industry_focus: 0.1, languages: 0.1).

Ejemplo de candidato: Full Stack Developer Senior

Los perfiles de candidatos siguen una estructura análoga con información detallada:

- **Información básica**: País de residencia, experiencia laboral, ambiente de trabajo preferido, objetivos profesionales primarios.
- **Destacados profesionales**: Descripción personal (aboutMe), tipos de empleo buscados, ubicaciones deseadas, disposición a reubicarse, expectativas salariales, habilidades técnicas con niveles de competencia alcanzados, lenguajes con niveles de dominio.
- **Credenciales**: Educación (institución, título, fechas, campo de estudio, calificación, descripción), certificaciones (organización emisora, nombre, fechas de emisión y expiración, ID de credencial, URL de verificación).
- **Valores fundamentales**: Vector de 24 competencias blandas autoevaluadas en escala 1-10, idéntica en estructura a la requerida por vacantes, permitiendo comparación directa mediante distancia vectorial.

Vacante 1: Senior Full Stack JavaScript Developer

Posición	Candidato ID	Score Global	Basic Info	Technical	Education	Soft Skills	Industry	Languages
1	6ba7b819	0.5643	0.7533	0.9038	0.4283	0.6484	1.0000	0.7403
2	6ba7b810	0.5570	0.7221	0.9112	0.3951	0.6401	1.0000	0.8081
3	6ba7b813	0.5275	0.5793	0.8572	0.5589	0.7284	0.7132	0.7589

El sistema identifica correctamente a dos desarrolladores *full stack senior* (6ba7b819 con 9 años de experiencia y 6ba7b810 con 8 años) como los mejores *matches*, con *scores* técnicos superiores a 0.90 y match perfecto en experiencia (1.0000). El tercer candidato (6ba7b813), un desarrollador *junior* de *frontend* con 2 años de experiencia, presenta *scores* significativamente menores en información básica y experiencia, reflejando correctamente su menor alineación con una posición senior.

Vacante 2: UX/UI Designer

Posición	Candidato ID	Score Global	Basic Info	Technical	Education	Soft Skills	Industry	Languages
1	6ba7b811	0.6018	0.8694	0.9148	0.5379	0.6679	1.0000	0.9749
2	6ba7b817	0.5552	0.6387	0.8790	0.6415	0.6437	0.8450	0.7454
3	6ba7b813	0.4214	0.4496	0.6344	0.4190	0.7241	0.5812	0.8803

El candidato 6ba7b811, UX/UI *designer* con 5 años de experiencia, obtiene el *score* global más alto (0.6018) con match perfecto en industria y *scores* superiores en todas las dimensiones técnicas. El segundo lugar (6ba7b817) es un product designer senior con 7+ años que, aunque más experimentado, tiene menor especialización en UX/UI específicamente. El candidato *junior* de *frontend* aparece tercero con *scores* técnicos significativamente menores (0.6344).

Vacante 3: Backend Python Developer

Posición	Candidato ID	Score Global	Basic Info	Technical	Education	Soft Skills	Industry	Languages
1	6ba7b816	0.5720	0.8101	0.9161	0.3539	0.6545	1.0000	0.8486
2	6ba7b810	0.4718	0.6390	0.6826	0.3992	0.6250	0.8337	0.8770
3	6ba7b819	0.4676	0.6205	0.6770	0.4075	0.6331	0.8337	0.9097

El sistema identifica correctamente al candidato 6ba7b816, desarrollador *backend Python* con 5 años de experiencia, como el mejor *match* con *score* técnico de 0.9161 y *match* perfecto en industria. Los dos candidatos siguientes son desarrolladores *full stack JavaScript* con *scores* técnicos significativamente menores (0.68), demostrando que el sistema distingue apropiadamente entre *Python* y *JavaScript* como lenguajes principales.

Vacante 4: Junior Front-end Developer

Posición	Candidato ID	Score Global	Basic Info	Technical	Education	Soft Skills	Industry	Languages
1	6ba7b813	0.5568	0.6700	0.8747	0.6081	0.6656	0.7896	0.6520
2	6ba7b810	0.5275	0.6910	0.8280	0.4244	0.5780	0.8460	0.6979
3	6ba7b819	0.5155	0.6228	0.8069	0.5062	0.5852	0.8460	0.6361

Para la posición junior, el sistema correctamente prioriza al candidato 6ba7b813, efectivamente un desarrollador frontend junior con 2 años de experiencia. Los dos siguientes son desarrolladores senior full stack, quienes aunque técnicamente competentes, representan sobre-calificación para la posición junior.

Vacante 5: Financial Analyst

Posición	Candidato ID	Score Global	Basic Info	Technical	Education	Soft Skills	Industry	Languages
1	6ba7b820	0.6112	0.9397	0.9487	0.3882	0.6396	1.0000	0.9594
2	6ba7b821	0.5375	0.8812	0.7951	0.2375	0.6873	0.8784	0.8489
3	6ba7b816	0.3053	0.2255	0.4964	0.1202	0.6594	0.5335	0.9292

El caso de la vacante Financial Analyst resulta particularmente ilustrativo de la capacidad del sistema para distinguir entre dominios profesionales completamente diferentes. Los dos candidatos mejor rankeados son profesionales de finanzas (6ba7b820 con 6 años de experiencia y certificación CFA, y 6ba7b821 analista junior con 3 años), con *scores* globales de 0.6112 y 0.5375 respectivamente.

El tercer candidato (6ba7b816), un desarrollador backend Python, obtiene un score global significativamente inferior de 0.3053. La descomposición por dimensiones revela claramente por qué no es un match apropiado: información básica 0.2255 (muy baja, indicando objetivos profesionales incompatibles), habilidades técnicas 0.4964 (medio-bajo, Python/SQL no es equivalente a Excel/Financial Modeling/GAAP), educación 0.1202 (muy baja, Computer Science versus Finance/Accounting), y experiencia 0.5335 (bajo, Web Development versus Finance).

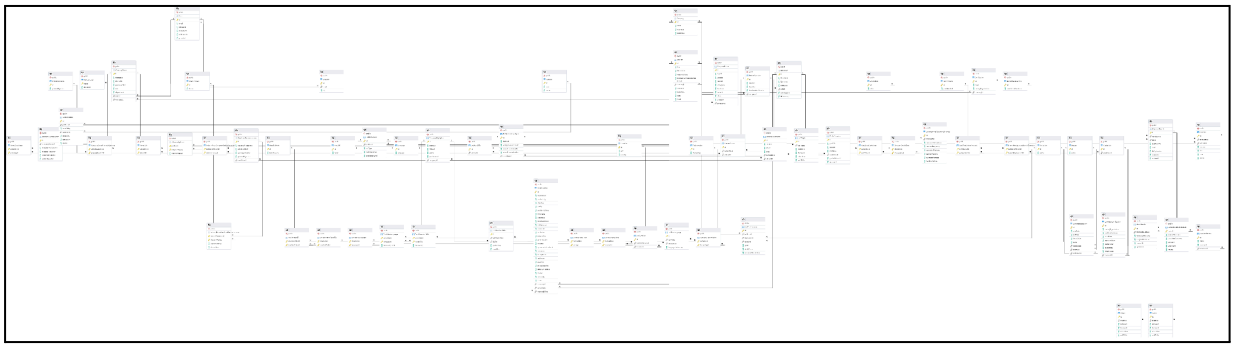
Notablemente, las dimensiones de soft skills (0.6594) y lenguajes (0.9292) tienen scores razonables, lo cual es correcto: las habilidades blandas como pensamiento crítico y atención al detalle son transferibles entre dominios, y el dominio de idiomas es independiente de la especialización profesional. Sin embargo, los pesos configurados (technical: 0.3, basic_info: 0.2, soft_skills: 0.2, education: 0.1, industry: 0.1, languages: 0.1) aseguran que las diferencias fundamentales en especialización dominen el score final.

Este resultado valida que el sistema no solo puede distinguir entre tecnologías específicas dentro de desarrollo de software (como se probó en iteraciones anteriores con JavaScript versus Python), sino también entre campos profesionales completamente distintos como tecnología y finanzas.

Esta distribución prioriza habilidades técnicas (30%) y mantiene balance entre información básica y soft skills (20% cada una), mientras que educación, industria y lenguajes tienen peso menor. Los resultados demuestran que esta configuración produce *rankings* consistentes y apropiados, priorizando match técnico sin ignorar otros factores relevantes.

16.18 Modelo de datos

Debido al tamaño del diagrama, su lectura resulta poco legible en esta vista. Para observar la imagen con mayor claridad y detalle, haga clic aquí.



16.19 Diagramas de secuencia

Detalle de los mensajes de los diagramas de secuencia

Fase 1: detalle de los mensajes

#	Emisor → Receptor	Tipo	Descripción	Notas técnicas
(1)	Usuario → Frontend	Acción de usuario	El usuario completa el formulario de creación de vacante.	Interacción UI. No genera mensaje UML técnicamente, pero se muestra para contexto del flujo.
(2)	Frontend → Backend	Síncrono (—>)	**POST /api/company/job** Envío de datos de la vacante con autenticación JWT en headers.	Request HTTP síncrono. Frontend espera la respuesta mientras está bloqueado. Inicia barra de activación en Backend.
(3)	Backend → Backend	Auto-mensaje (—>)	**Autenticación y validación** Verificación del token JWT y validación de la estructura de datos recibidos.	Self-call que genera barra de activación anidada. Verifica la información de la vacante y los permisos de la cuenta antes de proceder.

(4)	Backend → PostgreSQL	Síncrono (━>)	**Crea registro** Inserta la vacante en la base de datos incluyendo toda la información y las relaciones.	Transacción atómica SQL. Backend espera confirmación de la inserción.
(5)	PostgreSQL → Backend	Retorno (-->)	**Retorna vacante creada** Confirmación de creación exitosa con el ID generado y todos los datos persistidos.	Retorno de la transacción con el objeto Job completo incluyendo ID autogenerado.
(6)	Backend → Frontend	Retorno (-->)	**HTTP 201 Created** Respuesta exitosa con el jobId y datos de la vacante creada.	Código de estado HTTP 201 indica recurso creado exitosamente. Incluye payload con datos de la vacante.
(7)	Frontend → Usuario	Retorno (-->)	**Muestra confirmación de creación** La UI presenta un mensaje de éxito y redirige a la vista de detalle de la vacante.	Fin de la primera barra de activación del Frontend. Usuario ve la confirmación visual.

(8)	Frontend → Backend	Asíncrono (**-->**)	**POST /api/rabbitmq** Solicitud para disparar el procesamiento de *embeddings* mediante message broker.	**El** frontend no espera la respuesta, es un fire-and-forget. Segunda barra de activación del backend.
(9)	Backend → RabbitMQ	Síncrono (**—>**)	**Produce mensaje para generación de *embeddings*** Publicación del mensaje en la cola "embed.job" con datos de la vacante para procesamiento de IA.	Backend espera confirmación AMQP de que el mensaje fue persistido. Mensaje incluye routing key "embed.job".
(10)	RabbitMQ → Backend	Retorno (**-->**)	**Confirmación AMQP** Confirmación de que el mensaje fue recibido y persistido exitosamente en la cola "embed.job".	Confirmación a nivel protocolo AMQP. RabbitMQ confirma que el mensaje está durablemente almacenado y listo para ser consumido por Whizdy Matching.

Fase 2: detalle de los mensajes

#	Emisor → Receptor	Tipo	Descripción	Notas técnicas
(1)	Backend→RabbitMQ	Síncrono (━>)	**Publica mensaje para generación de *embeddings*** Envía datos de vacante a cola "embed.job"	Repetimos el mensaje número 9 de la fase 1 para dar contexto. Backend termina su participación aquí.
(2)	RabbitMQ → Whizdy Matching	Asíncrono (--->)	**Entrega mensaje** Consumer de Whizdy Matching recibe datos de la vacante desde la cola	Consumer escucha la cola "embed.job". RabbitMQ actúa como message broker sin esperar confirmación de procesamiento.
(3)	Whizdy Matching → Whizdy Matching	Auto-mensaje (━>)	**Genera embedding vectorial** Modelo de ML procesa texto y características de la vacante	Self-call con barra anidada. Procesamiento intensivo con modelo de lenguaje para generar representación vectorial.

(4)	Whizdy Matching → *Qdrant*	Síncrono (━>)	**Almacena vector** Guarda embedding de vacante en base de datos vectorial	Operación de inserción en *Qdrant*. Whizdy Matching espera confirmación.
(5)	*Qdrant* → Whizdy Matching	Retorno (-->)	**Confirmación de almacenamiento** Vector guardado exitosamente	Confirma que el embedding está persistido y disponible para búsquedas.
(6)	Whizdy Matching → *Qdrant*	Síncrono (━━━>)	**Búsqueda de similitud** Busca candidatos compatibles comparando vectores	Query de similitud semántica usando el embedding de la vacante para encontrar candidatos.
(7)	*Qdrant* → Whizdy Matching	Retorno (-->)	**Retorna candidatos compatibles** Lista de candidatos con scores de similitud vectorial	*Qdrant* retorna top-N candidatos más cercanos al vector de la vacante.
(8)	Whizdy Matching → Whizdy Matching	Auto-mensaje (━>)	**Calcula scores de matching** Aplica pesos configurados y genera ranking final	Self-call con barra anidada. Combina similitud vectorial con pesos personalizados de la empresa.

(9)	Whizdy Matching → Backend	Asíncrono (-->)	**POST /api/matching/matching-result** Envía resultados de matching vía HTTP callback	**Fire-and-forget**: Whizdy Matching no espera respuesta. Payload incluye lista de candidatos con scores por categoría.
(10)	Backend → PostgreSQL	Síncrono (━>)	**Guarda matches recomendados** Crea registros de candidatos recomendados con scores detallados	Inserción en tabla de matches con estado RECOMMENDED. Incluye scores por habilidades, experiencia, ubicación, etc.
(11)	PostgreSQL → Backend	Retorno (-->)	**Confirma guardado** Retorna confirmación de matches guardados	Confirma que los matches están persistidos y disponibles para consulta del Frontend.

Fase 3A: detalle de los mensajes

#	Emisor → Receptor	Tipo	Descripción	Notas técnicas
(1)	Backend → Redis	Síncrono (━>)	**Encola trabajos de notificación** Añade trabajos a la cola para notificar a la empresa y candidatos destacados.	Usa BullMQ para encolar múltiples trabajos: uno para empresa y otro para candidatos seleccionados.
(2)	Redis → Backend	Retorno (-->)	**Confirma encolado** Confirmación de que los jobs fueron agregados a la cola.	Jobs persistidos en Redis listos para ser procesados.
(3)	Backend (Worker) → Redis	Síncrono (━>)	**Solicita trabajo de la cola** Worker hace polling a Redis pidiendo trabajos pendientes.	Worker de BullMQ consulta activamente la cola de notificationes.
(4)	Redis → Backend (Worker)	Retorno (-->)	**Retorna job de notificación** Entrega job con datos: tipo, destinatarios, template, data.	El trabajo incluye el tipo, recipients, template, subject, etc.
(5)	Backend (Worker) → EmailService	Síncrono (━>)	**Envía notificaciones** Llama a sendEmail() con destinatarios y template.	Envío de emails a empresa (candidatos destacados) y a candidatos (nueva oportunidad). Puede ser

				múltiples llamadas según cantidad de destinatarios.
(6)	EmailService → Backend (Worker)	Retorno (-->)	**Confirmación de envío** Confirma que los emails fueron enviados exitosamente.	Worker marca el job como completado en Redis.
(7)	EmailService → Usuario	Asíncrono (-->)	**Email enviado (compañía)** Notificación con lista de candidatos más compatibles.	Usuario recibe email en su bandeja. Puede incluir resumen de top candidatos.
(8)	EmailService → Candidatos	Asíncrono (-->)	**Emails enviado (candidatos)** Notificación sobre nueva vacante compatible con su perfil.	Múltiples candidatos reciben emails. Se envía solo a candidatos destacados/top.

Fase 3B: detalle de los mensajes

#	Emisor → Receptor	Tipo	Descripción	Notas técnicas
(1)	Usuario → Frontend	Acción de usuario	**Solicita ver candidatos** Usuario accede a la vista de candidatos de la vacante creada.	El usuario puede ingresar a la vacante para ver los candidatos o puede ingresar por medio del correo
(2)	Frontend → Backend	Síncrono (━>)	**GET /api/company/job/{id}/candidates** Solicita lista de candidatos matcheados para la vacante.	Request HTTP con jobId en la URL. Puede ser por polling o acción directa del usuario.
(3)	Backend → PostgreSQL	Síncrono (━>)	**Consulta candidatos recomendados** Query de matches con estado RECOMMENDED y sus scores detallados.	SQL query para obtener datos del candidato y scores por categoría.
(4)	PostgreSQL → Backend	Retorno (-->)	**Retorna lista de candidatos** Lista ordenada de candidatos con scores de compatibilidad.	Incluye datos del candidato y breakdown de scores (habilidades, experiencia, etc.).
(5)	Backend → Frontend	Retorno (-->)	**HTTP 200 OK** Respuesta con array de candidatos y sus scores de *matching*.	JSON con lista completa de candidatos recomendados.

(6)	Frontend → Usuario	Retorno (-->)	**Muestra candidatos recomendados** UI presenta lista ordenada por compatibilidad con visualización de scores.	Usuario puede ver perfiles, scores detallados y tomar acciones (contactar, descartar, etc.).

16.20 Análisis resultados pruebas de carga del sistema

A continuación se visualiza el documento de análisis pos realizadas las pruebas de carga, este documento busca resumir brevemente los resultados obtenidos.

Pruebas de Carga – Whizdy

Se desarrolló en un repositorio dedicado, desacoplado del repositorio principal, un banco de pruebas para evaluar el rendimiento del sistema. Este ejecuta catorce planes de carga construidos con K6 [31], enfocados en las diferentes áreas del sistema, como Whizdy App (Backend y Fronted) y Whizdy Matching.

Cada prueba se lanza con un minuto de separación contra el entorno productivo desplegado en free tier, sin capas de cacheo intermedias, sin auto-escalado y con los límites de cómputo propios del plan gratuito, de modo que las capacidades quedan fijas y acotadas.

También se trabaja sobre una base de datos ya poblada con información real y se organiza en tres categorías:

- Backend – Core Whizdy: endpoints críticos de candidatos, empresas, vacantes y autenticación.
- Backend – Servicio de *Matching*: generación de *embeddings* y búsquedas vectoriales.
- Frontend: acceso a dashboards de candidato y empresa, páginas de login y listados pesados.

Los escenarios reproducen patrones reales de uso mediante tres estrategias:

- Carga sostenida (*maintained*): rampa de 30 s, cinco minutos a 10-20 usuarios concurrentes, rampa de salida de 30 s.
- Picos súbitos (*spike*): ascenso en 30 s hasta 30-50 usuarios, pico sostenido 60 s, descenso 30 s.
- Estrés progresivo (*stress*): calentamiento 1 min hasta 50 usuarios, escalada 50 → 100 → 150 usuarios en etapas de 2-3 minutos, enfriamiento 2 min.

El conjunto permite observar la evolución de latencias, transferencia y uso de recursos bajo condiciones controladas, validando que la arquitectura soporta los niveles de demanda previstos sin beneficiarse de recursos elásticos ni de intermediación de caché.

Los tests de carga se organizan en tres categorías principales según el componente objetivo:

- Tests de *Backend*: Se ejecutan mediante peticiones HTTP directas a las APIs de Whizdy. Incluyen operaciones como autenticación de usuarios, registro de candidatos, gestión de trabajos por parte de empresas, y obtención de datos de candidatos. Estas pruebas evalúan la capacidad de respuesta del backend bajo diferentes cargas de trabajo.

- Tests de *Matching*: Combinan peticiones HTTP directas al sistema de *matching* con el encolado de mensajes en *RabbitMQ*. Algunas operaciones solicitan *matching* de candidatos para trabajos específicos, mientras que otras publican mensajes en colas para procesamiento asíncrono de *embeddings* y análisis de compatibilidad.

- Tests de *Frontend*: Realizan peticiones HTTP GET a las páginas web de la aplicación, simulando usuarios reales navegando por dashboards de candidatos y empresas. Estos tests evalúan el rendimiento de la carga completa de páginas HTML, incluyendo recursos estáticos y contenido dinámico.

Todos los tests miden métricas estándar de rendimiento:

- *Throughput*: Requests por segundo procesados exitosamente

- Latencia: Tiempos de respuesta medidos en percentiles (p90, p95, p99)

- Concurrencia: Número de usuarios virtuales simultáneos

- Tasa de éxito: Porcentaje de requests exitosos vs fallidos

- Volumen de datos: Cantidad de datos enviados y recibidos

- Duración de conexión: Tiempos de establecimiento y mantenimiento de conexiones

Los umbrales de aceptación típicamente requieren tasas de fallo menores al 1%, tiempos de respuesta p95 bajo 1200ms para backend y checks exitosos superiores al 98%.

Cada test se ejecuta bajo tres escenarios diferentes:

- Maintained (Mantenimiento): Pruebas de carga sostenida con 15-20 usuarios virtuales durante 5 minutos, precedidas por 30 segundos de rampa ascendente y seguidas de 30 segundos de rampa descendente. Evalúan el comportamiento bajo carga normal esperada.

- Spike (Picos): Tests de carga con picos de hasta 35 usuarios virtuales, simulando aumentos repentinos de tráfico como los que podrían ocurrir durante horas pico o eventos especiales. Mantienen la carga elevada por períodos más cortos.

- Stress (Estrés): Pruebas con 40 o más usuarios virtuales sostenidos, diseñadas para identificar puntos de quiebre del sistema y evaluar el comportamiento bajo condiciones extremas de carga.

Cada escenario incluye tres etapas: rampa ascendente para alcanzar gradualmente la carga objetivo, mantenimiento de la carga durante el período de evaluación, y rampa descendente para finalizar la prueba de manera controlada.

BACKEND TESTS

[T1] Candidate Setup – POST /api/candidates/setup

Onboarding completo del candidato: carga académica, experiencia, soft-skills, etc. Crítico porque los picos de nuevos registros pueden saturar validaciones y guardado de archivos

Umbrales

p95 (http_req_duration) < 1200 ms

p99 (http_req_duration) < 3000 ms

p95 (waiting) < 900 ms

Error rate < 1 %

Escenarios ejecutados

Mantenido

Perfil: ~5 min · VUs máx.: 15 · RPS: ~39 · Req: 11 802

Métrica	Valor	Cumple
Duración p50 (ms)	348.65	-
Duración p95 (ms)	643.66	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	643.11	✔
Error rate	1.06 %	✘
VUs máx.	15	-
RPS	39.29	-

Spike

Perfil: ~90 s · VUs máx.: 35 · RPS: ~39 · Req: 3 548

Métrica	Valor	Cumple
Duración p50 (ms)	374.91	-
Duración p95 (ms)	2 240	✘
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	2 240	✘
Error rate	1.97 %	✘
VUs máx.	35	-
RPS	39.22	-

Estrés

Perfil: ~7 min · VUs máx.: 40 · RPS: ~40 · Req: 16 886

Métrica	Valor	Cumple
Duración p50 (ms)	382.97	-
Duración p95 (ms)	2 250	✘
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	2 250	✘
Error rate	2.22 %	✘
VUs máx.	40	-
RPS	40.15	-

Conclusión general

En mantenido procesó 11 802 requests (~39.3 RPS, 15 VUs) con latencias dentro de umbral y error rate 1.06–1.07 % por encima del límite. En spike (3 548 req, ~39.2 RPS, 35 VUs) y estrés (16 886 req, ~40.2 RPS, 40 VUs) no cumple latencias p95 (≈2.24–2.26 s) ni error rate.

[T2] Login – POST /api/auth/login

Punto de entrada único para candidatos y empresas. Define el flujo concurrente total de usuarios; cualquier degradación afecta a toda la plataforma.

Umbrales

p95 (http_req_duration) < 600 ms

p99 (http_req_duration) < 1500 ms

p95 (waiting) < 500 ms

Error rate < 1 %

Escenarios ejecutados

Mantenido

Perfil: ~5 min · VUs máx.: 15 · RPS: ~45 · Req: 13 718

Métrica	Valor	Cumple
Duración p50 (ms)	319.95	-
Duración p95 (ms)	449.74	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	449.34	✔
Error rate	0.00 %	✔
VUs máx.	15	-
RPS	45.67	-

Spike

Perfil: ~90 s · VUs máx.: 35 · RPS: ~79 · Req: 7 200

Métrica	Valor	Cumple
Duración p50 (ms)	327.13	-
Duración p95 (ms)	529.86	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	529.07	✘
Error rate	0.00 %	✔
VUs máx.	35	-
RPS	79.68	-

Estrés

Perfil: ~7 min · VUs máx.: 40 · RPS: ~101 · Req: 42 472

Métrica	Valor	Cumple
Duración p50 (ms)	316.95	-
Duración p95 (ms)	381.41	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	381.07	✔
Error rate	0.00 %	✔
VUs máx.	40	-
RPS	101.00	-

Conclusión general

Estable en los tres escenarios, sin errores. Cumple en mantenido (13 718 req, ~45.7 RPS, 15 VUs) y estrés (42 472 req, ~101 RPS, 40 VUs). En spike (7 200 req, ~79.7 RPS, 35 VUs) todas las métricas cumplen salvo waiting p95 = 529 ms (ligeramente sobre umbral).

[T3] Register Candidate – POST /api/candidates

Alta rápida con envío de e-mail de verificación. Paso previo al setup; un ingreso masivo puede generar picos de escritura y notificaciones.

Umbrales

p95 (http_req_duration) < 1000 ms

p99 (http_req_duration) < 2500 ms

p95 (waiting) < 700 ms

Error rate < 1 %

Escenarios ejecutados

Mantenido

Perfil: ~5 min · VUs máx.: 15 · RPS: ~40 · Req: 12 052

Métrica	Valor	Cumple
Duración p50 (ms)	406.51	-
Duración p95 (ms)	497.06	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	496.69	✔
Error rate	0.00 %	✔
VUs máx.	15	-
RPS	40.13	-

Spike

Perfil: ~90 s · VUs máx.: 35 · RPS: ~79 · Req: 7 194

Métrica	Valor	Cumple
Duración p50 (ms)	404.46	-
Duración p95 (ms)	503.66	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	503.35	✔
Error rate	0.00 %	✔
VUs máx.	35	-
RPS	79.17	-

Estrés

Perfil: ~7 min · VUs máx.: 40 · RPS: ~83 · Req: 34 940

Métrica	Valor	Cumple
Duración p50 (ms)	398.66	-
Duración p95 (ms)	484.51	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	484.17	✔
Error rate	0.00 %	✔
VUs máx.	40	-
RPS	83.13	-

Conclusión general

Cumple en los tres escenarios, 0 % de errores. En mantenido (12 052 req, ~40.1 RPS, 15 VUs) el p95 ronda 0.50 s; se mantiene similar en spike (7 194 req, ~79.2 RPS, 35 VUs) y estrés (34 940 req, ~83.1 RPS, 40 VUs).

[T4] Company Create Job – POST /api/company/job

Crea la vacante, encola *embeddings*, dispara *matching* y notifica resultados. Recorre todos los servicios: es el proceso crítico de negocio para generar valor a empresas. Se hace un insert en PostgreSQL, se publica en Rabbit, el servicio *Matching* genera embedding y matchign contra los candados existentes y vuelve a llamar a Whizdy vía http para guardar resultados de *matching* y enviar e-mails.

Umbrales
p95 (http_req_duration) < 1200 ms
p99 (http_req_duration) < 3000 ms
p95 (waiting) < 800 ms
Error rate < 1 %

Escenarios ejecutados

Mantenido
Perfil: ~5 min · VUs máx.: 20 · RPS: ~66 · Req: 20 077

Métrica	Valor	Cumple
Duración p50 (ms)	275.86	-
Duración p95 (ms)	451.11	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	450.66	✔
Error rate	0.00 %	✔
VUs máx.	20	-
RPS	66.92	-

Spike

Perfil: ~90 s · VUs máx.: 35 · RPS: ~61 · Req: 5 558

Métrica	Valor	Cumple
Duración p50 (ms)	281.86	-
Duración p95 (ms)	1 360	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	1 360	✔
Error rate	0.00 %	✔
VUs máx.	35	-
RPS	61.54	-

Estrés

Perfil: ~7 min · VUs máx.: 40 · RPS: ~42 · Req: 18 016

Métrica	Valor	Cumple
Duración p50 (ms)	282.35	-
Duración p95 (ms)	2 760	✘
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	2 720	✘
Error rate	0.07 %	✔
VUs máx.	40	-
RPS	42.89	-

Conclusión general

Cumple en mantenido (20 077 req, ~66.9 RPS, 20 VUs) y spike (5 558 req, ~61.5 RPS, 35 VUs) según los umbrales registrados. En estrés (18 016 req, ~42.9 RPS, 40 VUs) no cumple p95 (≈2.76 s) ni waiting p95, con error rate muy bajo (0.07 %).

[T5] Company Get All Candidates – GET /company/candidates

Listado paginado candidatos por empresa. Consulta pesada (+300 candidatos)

Umbrales

p95 (http_req_duration) < 1500 ms

p99 (http_req_duration) < 2500 ms

p95 (waiting) < 800 ms

Error rate < 1 %

Escenarios ejecutados

Mantenido

Perfil: ~5 min · VUs máx.: 15 · RPS: ~118 · Req: 35 658

Métrica	Valor	Cumple
Duración p50 (ms)	176.93	-
Duración p95 (ms)	231.04	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	217.45	✔
Error rate	0.00 %	✔
VUs máx.	15	-
RPS	118.85	-

Spike

Perfil: ~90 s · VUs máx.: 35 · RPS: ~252 · Req: 22 686

Métrica	Valor	Cumple
Duración p50 (ms)	174.50	-
Duración p95 (ms)	225.50	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	198.29	✔
Error rate	0.00 %	✔
VUs máx.	35	-
RPS	252.08	-

Estrés

Perfil: ~7 min · VUs máx.: 40 · RPS: ~262 · Req: 110 064

Métrica	Valor	Cumple
Duración p50 (ms)	174.28	-
Duración p95 (ms)	225.39	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	197.74	✔
Error rate	0.00 %	✔
VUs máx.	40	-
RPS	262.06	-

Conclusión general

Cumple en los tres escenarios, 0 % de errores. Mantiene p95 ~219–231 ms y escala hasta ~262 RPS en estrés (110 064 req, 40 VUs), con mantenido (15 VUs) y spike (35 VUs) sin degradación.

[T6] Company Get Candidate Details – GET /company/jobs/profiles/{id}

Devuelve el perfil de candidato completo, información personal, académica, profesional, etc.
Gran cantidad de información por consulta

Umbrales
p95 (http_req_duration) < 1500 ms
p99 (http_req_duration) < 2500 ms
p95 (waiting) < 800 ms
Error rate < 1 %

Escenarios ejecutados

Mantenido
Perfil: ~5 min · VUs máx.: 15 · RPS: ~61 · Req: 18 396

Métrica	Valor	Cumple
Duración p50 (ms)	209.53	-
Duración p95 (ms)	278.22	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	275.15	✔
Error rate	0.00 %	✔
VUs máx.	15	-
RPS	61.29	-

Spike

Perfil: ~90 s · VUs máx.: 35 · RPS: ~126 · Req: 11 354

Métrica	Valor	Cumple
Duración p50 (ms)	202.57	-
Duración p95 (ms)	270.13	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	265.83	✔
Error rate	0.00 %	✔
VUs máx.	35	-
RPS	126.09	-

Estrés

Perfil: ~7 min · VUs máx.: 40 · RPS: ~132 · Req: 55 826

Métrica	Valor	Cumple
Duración p50 (ms)	200.30	-
Duración p95 (ms)	250.20	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	242.14	✔
Error rate	0.00 %	✔
VUs máx.	40	-
RPS	132.81	-

Conclusión general

Cumple en los tres escenarios, 0 % de errores. p95 ~250–278 ms; alcanza ~132.8 RPS en estrés (55 826 req, 40 VUs) y se mantiene estable en mantenido (15 VUs) y spike (35 VUs).

[T7] Company Get Job Candidates – GET /api/company/job/candidates

Candidatos filtrados para una vacante específica. Endpoint sumamente importante para perfil empresa; alta frecuencia y grandes respuestas.

Umbrales

p95 (http_req_duration) < 1500 ms

p99 (http_req_duration) < 2500 ms

p95 (waiting) < 900 ms

Error rate < 1 %

Escenarios ejecutados

Mantenido

Perfil: ~5 min · VUs máx.: 15 · RPS: ~58 · Req: 17 676

Métrica	Valor	Cumple
Duración p50 (ms)	219.94	-
Duración p95 (ms)	296.04	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	295.60	✔
Error rate	0.00 %	✔
VUs máx.	15	-
RPS	58.85	-

Spike

Perfil: ~90 s · VUs máx.: 35 · RPS: ~121 · Req: 11 014

Métrica	Valor	Cumple
Duración p50 (ms)	218.12	-
Duración p95 (ms)	262.29	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	261.53	✔
Error rate	0.00 %	✔
VUs máx.	35	-
RPS	121.95	-

Estrés

Perfil: ~7 min · VUs máx.: 40 · RPS: ~119 · Req: 50 200

Métrica	Valor	Cumple
Duración p50 (ms)	220.28	-
Duración p95 (ms)	334.97	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	333.93	✔
Error rate	0.00 %	✔
VUs máx.	40	-
RPS	119.53	-

Conclusión general

Cumple en los tres escenarios, 0 % de errores. p95 ~250–278 ms; alcanza ~132.8 RPS en estrés (55 826 req, 40 VUs) y se mantiene estable en mantenido (15 VUs) y spike (35 VUs).

MATCHING TESTS

[T8] Job Embedding – POST /exchanges/{vhost}/{exchange}/publish

Publica mensaje a *RabbitMQ* tal como lo haría Whizdy tras crear/edita job. Evalúa cuántas vacantes por segundo puede encolar el sistema sin perder mensajes.

Umbrales
p95 (http_req_duration) < 1200 ms
p99 (http_req_duration) < 3000 ms
p95 (waiting) < 900 ms
Error rate < 1 %

Escenarios ejecutados

Mantenido
Perfil: ~5 min · VUs máx.: 20 · RPS: ~108 · Req: 32 516

Métrica	Valor	Cumple
Duración p50 (ms)	162.71	-
Duración p95 (ms)	175.54	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	175.33	✔
Error rate	0.00 %	✔
VUs máx.	20	-
RPS	108.37	-

Spike

Perfil: ~90 s · VUs máx.: 35 · RPS: ~161 · Req: 14 523

Métrica	Valor	Cumple
Duración p50 (ms)	163.19	-
Duración p95 (ms)	182.29	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	182.13	✔
Error rate	0.00 %	✔
VUs máx.	35	-
RPS	161.29	-

Estrés

Perfil: ~7 min · VUs máx.: 40 · RPS: ~165 · Req: 69 424

Métrica	Valor	Cumple
Duración p50 (ms)	163.08	-
Duración p95 (ms)	180.41	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	180.21	✔
Error rate	0.00 %	✔
VUs máx.	40	-
RPS	165.28	-

Conclusión general

Cumple en los tres escenarios, 0 % de errores. Latencias muy estables (p95 ~175–182 ms) y buen throughput: ~108–165 RPS con 20/35/40 VUs (hasta 69 424 req en estrés).

[T9] Job Matching Candidates – POST /matching/v2/job-to-candidates

Fuerza bruta del motor: calcula similitud entre vectores. Determina el techo de trabajos que podemos matchear en paralelo sin degradar tiempos. En la práctica este proceso se ejecuta de forma asincrónica con una cola.

Umbrales
p95 (http_req_duration) < 2000 ms
p99 (http_req_duration) < 3000 ms
p95 (waiting) < 1500 ms
Error rate < 1 %

Escenarios ejecutados

Mantenido
Perfil: ~5 min · VUs máx.: 20 · RPS: ~11 · Req: 3 478

Métrica	Valor	Cumple
Duración p50 (ms)	1 480	-
Duración p95 (ms)	2 900	✘
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	2 900	✘
Error rate	0.00 %	✔
VUs máx.	20	-
RPS	11.58	-

Spike

Perfil: ~90 s · VUs máx.: 35 · RPS: ~15 · Req: 1 569

Métrica	Valor	Cumple
Duración p50 (ms)	1 500	-
Duración p95 (ms)	2 930	✘
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	2 930	✘
Error rate	0.00 %	✔
VUs máx.	35	-
RPS	15.61	-

Estrés

Perfil: ~7 min · VUs máx.: 40 · RPS: ~16 · Req: 7 034

Métrica	Valor	Cumple
Duración p50 (ms)	1 510	-
Duración p95 (ms)	2 820	✘
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	2 820	✘
Error rate	0.00 %	✔
VUs máx.	40	-
RPS	16.73	-

Conclusión general

Procesa sin errores (0 %) pero no cumple p95 en ningún escenario (≈2.82–2.93 s). Throughput moderado: ~11.6–16.7 RPS con 20/35/40 VUs (hasta 7 034 req en estrés).

[T10] Job Register – POST /jobs/

Bypass de Whizdy: registro síncrono de job en matching. Mide la capacidad pura de generar *embeddings* del sistema de Matching, en la práctica este proceso es asincrónico.

Umbrales

p95 (http_req_duration) < 3000 ms

p99 (http_req_duration) < 5500 ms

p95 (waiting) < 2500 ms

Error rate < 1 %

Escenarios ejecutados

Mantenido

Perfil: ~5 min · VUs máx.: 20 · RPS: ~1.5 · Req: 470

Métrica	Valor	Cumple
Duración p50 (ms)	11 010	-
Duración p95 (ms)	27 470	✘
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	27 470	✘
Error rate	9.14 %	✘
VUs máx.	20	-
RPS	1.56	-

Spike

Perfil: ~90 s · VUs máx.: 35 · RPS: ~1.4 · Req: 158

Métrica	Valor	Cumple
Duración p50 (ms)	10 380	-
Duración p95 (ms)	59 830	✘
Duración p99 (ms)	N/D	✘
Waiting p95 (ms)	59 830	✘
Error rate	14.55 %	✘
VUs máx.	35	-
RPS	1.46	-

Estrés

Perfil: ~7 min · VUs máx.: 40 · RPS: ~1.5 · Req: 658

Métrica	Valor	Cumple
Duración p50 (ms)	8 940	-
Duración p95 (ms)	59 830	✘
Duración p99 (ms)	N/D	✘
Waiting p95 (ms)	59 830	✘
Error rate	19.60 %	✘
VUs máx.	40	-
RPS	1.49	-

Conclusión general

No cumple en ningún escenario: p95 muy alto (27–59 s) y error rate elevado (9–20 %). Throughput bajo y estable (~1.5 RPS) con 20/35/40 VUs (470–658 req según escenario).

FRONTEND TESTS

[T11] Candidate Dashboard – GET /candidate/dashboard

Home del candidato: matches, recomendaciones, estado de aplicaciones. Alta sensibilidad a la latencia porque es la primera impresión del producto. Consultas complejas y demandante visualmente.

Umbrales

p95 (http_req_duration) < 800 ms

p99 (http_req_duration) < 2000 ms

p95 (waiting) < 600 ms

Error rate < 1 %

Escenarios ejecutados

Mantenido

Perfil: ~5 min · VUs máx.: 15 · RPS: ~128 · Req: 38 666

Métrica	Valor	Cumple
Duración p50 (ms)	173.67	-
Duración p95 (ms)	208.05	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	195.25	✔
Error rate	0.00 %	✔
VUs máx.	15	-
RPS	128.84	-

Spike

Perfil: ~90 s · VUs máx.: 35 · RPS: ~241 · Req: 21 782

Métrica	Valor	Cumple
Duración p50 (ms)	175.35	-
Duración p95 (ms)	227.42	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	198.42	✔
Error rate	0.00 %	✔
VUs máx.	35	-
RPS	241.91	-

Estrés

Perfil: ~7 min · VUs máx.: 40 · RPS: ~265 · Req: 111 396

Métrica	Valor	Cumple
Duración p50 (ms)	173.26	-
Duración p95 (ms)	225.44	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	192.96	✔
Error rate	0.00 %	✔
VUs máx.	40	-
RPS	265.15	-

Conclusión general

Cumple en los tres escenarios, 0 % de errores. p95 ~208–227 ms con escalamiento hasta ~265 RPS en estrés (111 396 req, 40 VUs), estable también en mantenido (15 VUs) y spike (35 VUs).

[T12] Company Candidates – GET /company/candidates

Explorador general de candidatos para la empresa. (+300 candidatos)

Umbrales

p95 (http_req_duration) < 2500 ms

p99 (http_req_duration) < 5000 ms

p95 (waiting) < 1500 ms

Error rate < 2 %

Escenarios ejecutados

Mantenido

Perfil: ~5 min · VUs máx.: 15 · RPS: ~124 · Req: 37 270

Métrica	Valor	Cumple
Duración p50 (ms)	175.59	-
Duración p95 (ms)	218.90	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	200.33	✔
Error rate	0.00 %	✔
VUs máx.	15	-
RPS	124.17	-

Spike

Perfil: ~90 s · VUs máx.: 35 · RPS: ~249 · Req: 22 456

Métrica	Valor	Cumple
Duración p50 (ms)	174.40	-
Duración p95 (ms)	227.79	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	203.53	✔
Error rate	0.00 %	✔
VUs máx.	35	-
RPS	249.30	-

Estrés

Perfil: ~7 min · VUs máx.: 40 · RPS: ~249 · Req: 104 604

Métrica	Valor	Cumple
Duración p50 (ms)	174.70	-
Duración p95 (ms)	228.69	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	208.84	✔
Error rate	0.00 %	✔
VUs máx.	40	-
RPS	249.03	-

Conclusión general

Cumple en los tres escenarios, 0 % de errores. p95 ~219–229 ms y hasta ~249 RPS en estrés (104 604 req, 40 VUs), consistente en mantenido (15 VUs) y spike (35 VUs).

[T13] Company Dashboard – GET /company/dashboard

Home de empresa con KPIs, gráficos y listados. Reúne múltiples endpoints internos; un alto tráfico con consultas complejas y demandante visualmente.

Umbrales

p95 (http_req_duration) < 2500 ms

p99 (http_req_duration) < 4000 ms

p95 (waiting) < 1200 ms

Error rate < 2 %

Escenarios ejecutados

Mantenido

Perfil: ~5 min · VUs máx.: 15 · RPS: ~128 · Req: 38 548

Métrica	Valor	Cumple
Duración p50 (ms)	173.23	-
Duración p95 (ms)	214.43	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	195.46	✔
Error rate	0.00 %	✔
VUs máx.	15	-
RPS	128.46	-

Spike

Perfil: ~90 s · VUs máx.: 35 · RPS: ~254 · Req: 22 952

Métrica	Valor	Cumple
Duración p50 (ms)	172.63	-
Duración p95 (ms)	225.73	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	192.17	✔
Error rate	0.00 %	✔
VUs máx.	35	-
RPS	254.49	-

Estrés

Perfil: ~7 min · VUs máx.: 40 · RPS: ~253 · Req: 106 640

Métrica	Valor	Cumple
Duración p50 (ms)	173.75	-
Duración p95 (ms)	226.65	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	202.66	✔
Error rate	0.00 %	✔
VUs máx.	40	-
RPS	253.85	-

Conclusión general

Cumple en los tres escenarios, 0 % de errores. p95 ~214–227 ms y hasta ~254 RPS en estrés (106 640 req, 40 VUs), estable con 15/35 VUs en los demás escenarios.

[T14] Company Jobs Candidates – GET /company/jobs/candidates

Página de candidatos postulados a una vacante. Consultas complejas, muy visitada durante procesos de selección activos.

Umbrales

p95 (http_req_duration) < 2500 ms

p99 (http_req_duration) < 5000 ms

p95 (waiting) < 1500 ms

Error rate < 2 %

Escenarios ejecutados

Mantenido

Perfil: ~5 min · VUs máx.: 15 · RPS: ~115 · Req: 34 708

Métrica	Valor	Cumple
Duración p50 (ms)	177.42	-
Duración p95 (ms)	240.50	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	232.44	✔
Error rate	0.00 %	✔
VUs máx.	15	-
RPS	115.67	-

Spike

Perfil: ~90 s · VUs máx.: 35 · RPS: ~216 · Req: 19 545

Métrica	Valor	Cumple
Duración p50 (ms)	178.04	-
Duración p95 (ms)	244.99	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	234.87	✔
Error rate	0.00 %	✔
VUs máx.	35	-
RPS	216.69	-

Estrés

Perfil: ~7 min · VUs máx.: 40 · RPS: ~250 · Req: 105 398

Métrica	Valor	Cumple
Duración p50 (ms)	175.05	-
Duración p95 (ms)	231.03	✔
Duración p99 (ms)	N/D	✔
Waiting p95 (ms)	210.91	✔
Error rate	0.00 %	✔
VUs máx.	40	-
RPS	250.92	-

Conclusión general

Cumple en los tres escenarios, 0 % de errores. p95 ~231–245 ms; escala hasta ~251 RPS en estrés (105 398 req, 40 VUs), con resultados consistentes en mantenido (15 VUs) y spike (35 VUs).

Conclusión general de las pruebas

En términos globales, los endpoints de lectura (*GET*) , tanto del backend como de las páginas de *frontend*. se sostienen con p95 en el rango ~210–245 ms en los tres escenarios (mantenido, *spike* y estrés), sin errores, y con alto throughput sostenido en *frontend*: entre ~240 y ~265 RPS en las vistas principales (Candidate/Company *Dashboard*, listados), con concurrencias de 15–40 VUs. El incremento de VUs no introduce degradaciones relevantes en latencia para estas rutas y el RPS escala a valores altos de forma consistente.

Para las operaciones de creación (*POST*) no intensivas en cómputo, el desempeño típico en mantenido se ubica en p95 ~450–650 ms (por ejemplo, *Register Candidate y Create Job*). En spike/estrés, según el caso, el p95 puede elevarse hacia ~2.3–2.8 s manteniendo errores nulos o bajos, con concurrencias de 15–40 VUs y RPS acordes al perfil de cada prueba. Un caso particular es Candidate Setup, que en mantenido cumple latencias pero no el objetivo de error rate por un margen pequeño; en spike/estrés su p95 se sitúa en ~2.24–2.26 s con ~35–40 VUs y ~39–40 RPS.

En el servicio de *matching*, cuando se ejecuta de forma sincrónica (tal como se probó para establecer el punto de falla y cuantificar su impacto extremo), los tiempos se ubican en p95 ~2.8–2.9 s para el cálculo de similitudes (job-to-candidates) con ~11.6–16.7 RPS y 20–40 VUs, y se registran p95 muy altos (27–60 s) junto con errores significativos (9–20%) en el registro de jobs/*embeddings* (*Job Register*), con ~1.5 RPS y 20–40 VUs. Estos resultados ilustran el alto procesamiento involucrado en el pipeline de *matching* y cómo, cuando se fuerza el camino sincrónico de punta a punta, los procesos pesados degradan el sistema; el comportamiento asíncrono habitual y su capacidad se evalúan por separado en la sección de Create Job. En modo asíncrono -que es el uso esperado- el sistema simplemente encola y responde, sosteniendo el encolado sin errores y con RPS altos (por ejemplo, Job Embedding ~108–165 RPS con 0% de errores), por lo que no se requiere una respuesta inmediata al usuario ni se ve afectada la navegación.

En conjunto, el sistema atiende muy bien lectura y navegación bajo cargas altas (*GET* y vistas *frontend*), mientras que las rutas de creación presentan desempeño estable en mantenido y variaciones en spike/estrés acordes al volumen, y las rutas de *matching* en modo sincrónico exhiben tiempos y errores elevados propios de un flujo de cómputo intensivo. Esta fotografía

resume el estado del sistema con 15–40 VUs y RPS desde ~1.5 (flujos muy pesados) hasta ~265 (vistas y lecturas), sin intervención de mecanismos elásticos ni caché intermedio.

16.21 Validación solución Recruiter

A continuación se detallan las preguntas realizadas a la recruiter durante el proceso de validación de solución tras la realización de la demostración del producto, este documento contiene las preguntas, respuestas y un análisis posterior

Validación con recruiter – GoGrow

1. ¿Qué funcionalidades te parecieron más importantes y cuáles tendrían mayor impacto en tu trabajo diario?

- Disponer de una base de datos fácil de usar, que permita una vista consolidada de todos los candidatos, incluso de procesos anteriores.
- Automatización en la extracción de información del CV al perfil, especialmente valioso frente a plataformas como Team Tailor que no lo logran bien.
- Carga automática del CV para autocompletar datos del perfil, reduciendo la fricción para los candidatos.
- Posibilidad de editar la información luego de subir el CV.
- Incluir campo para indicar si la persona está dispuesta a trabajar en otras áreas o con otros lenguajes.
- Para perfiles junior: es poco realista esperar que completen muchos campos o que conozcan conceptos como soft skills.
- Para perfiles senior: si no están activamente buscando empleo, no completarán formularios extensos.
- Recomendación: minimizar la carga manual y utilizar la IA para organizar y etiquetar la información automáticamente.

2. ¿Qué mejorarías de la experiencia general de uso de la plataforma?

- Necesidad de autocompletado de datos, fluidez, formularios más breves y mejor usabilidad general.

3. ¿El proceso de creación de vacantes te resultó cómodo e intuitivo?

- Sí, resultó claro e intuitivo.

4. ¿Los resultados que se muestran para los candidatos te resultan claros y útiles para tomar decisiones?

- La información debe mostrar qué factores se tuvieron en cuenta para calcular el match.

- Es importante evitar que parezca una "caja negra": mostrar claramente qué dimensiones fueron relevantes para el resultado.

5. ¿Qué tipo de feedback te gustaría poder dejar o recibir sobre los candidatos durante el proceso?

- Agradeció la funcionalidad de cambio automático de estado con envío de email.

- Valoró la opción de enviar notificaciones automáticas semanales, como: “Seguís siendo considerado, aún estamos evaluando”.

- Quiere poder modificar la plantilla de los correos que se envían a los candidatos.

6. ¿El nivel de detalle que mostramos en el informe de compatibilidad (*matching*) cumple con tus expectativas?

- Respondido en punto 4. El foco está en la necesidad de transparencia en el cálculo del *matching*.

7. ¿Creés que la información que se muestra sobre los candidatos es suficiente para avanzar en el proceso de selección?

- Implícitamente, sí. Lo considera útil siempre que pueda editar la información, entender por qué se hace el match, y los datos estén organizados.

8. ¿Qué aspectos del proceso actual te gustaría automatizar o simplificar mediante esta herramienta?

- **Envío automático de correos** (status updates, recordatorios).
- **Autocompletado desde CV.**
- **Personalización de plantillas de correo.**

9. ¿Sentís que la plataforma te ayudaría a ahorrar tiempo o mejorar alguna etapa puntual del proceso de *scouting*?

- Sí, especialmente en procesos con **alto volumen de candidatos**, al facilitar el filtrado inicial y seguimiento.

10. ¿Sentís que el sistema de *matching* es una caja negra en la cual podés confiar para realizar el *matching* correcto?

- Depende del volumen:
    - Para vacantes con pocos candidatos (ej: perfiles senior), preferiría revisar manualmente.
    - Para procesos **masivos**, sí ve valor en usar la IA como **primer filtro confiable**.

11. ¿Te parece útil integrar entrevistas, comentarios y feedback en la misma herramienta?

- Sí. Ya lo respondió positivamente. Valora tener **todo el proceso centralizado**.

12. ¿Qué te aportaría esta plataforma que no te aportan otras como LinkedIn?

- **Análisis automatizado del CV.**
- Una base de datos mejor organizada.
- Posibilidad de enviar **mails automáticos para que los candidatos actualicen su información**.

13. ¿Hay algo que esperabas ver y no te mostramos?

- Le hubiera interesado que el sistema sugiriera automáticamente **candidatos antiguos** que podrían aplicar a nuevas vacantes similares.

Resumen

La recruiter valoró la facilidad de uso, la organización clara de la base de candidatos y la posibilidad de autocompletar datos desde el CV. Agradeció funciones como los mails automáticos, el cambio de estado de candidatos y el potencial para ahorrar tiempo en procesos con gran volumen. Subrayó que la transparencia del sistema de *matching* es clave para confiar en él, y que los formularios deben ser breves y simples, especialmente para perfiles junior o no activamente buscando empleo. Sugirió que el sistema recuerde a los candidatos actualizar sus datos periódicamente.

Conclusiones

Funcionalidades Validadas

- Envío automático de mails y actualización de estado.
- Creación de vacantes fluida.
- *Matching* útil como primer filtro, especialmente en procesos masivos.

Mejoras / Nuevas funcionalidades sugeridas

- Mostrar explícitamente los factores que se utilizaron en el *matching*.
- Permitir personalizar las plantillas de correo.
- Autocompletado de perfil con CV
- Agregar campo de “disponibilidad para otras áreas o lenguajes”.
- Incluir recordatorios para que los candidatos actualicen su perfil.

Cambios sugeridos

- Explicar mejor qué significa cada campo o habilidad en los formularios.
- Hacer que el CV sea obligatorio, pero editable luego de cargarlo.

16.22 Validación solucion con usuarios candidatos

A continuación se detallan las validaciones de la solución realizadas con los candidatos, el proceso, las notas originales y un breve análisis con hallazgos y conclusiones.

Validación de Usuarios – Candidatos

Objetivo de la Validación

- Verificar la funcionalidad general del sistema desde la perspectiva de los candidatos.
- Detectar problemas de usabilidad, puntos de confusión y posibles errores durante la navegación.
- Observar cómo los usuarios reales entienden y utilizan las funciones principales: registro, configuración del perfil, navegación por el dashboard, visualización de vacantes y seguimiento de procesos.
- Usar la instancia también como testing general de la aplicación, identificando *bugs* o flujos rotos antes del cierre del desarrollo.
- Explorar oportunidades de mejora y nuevas funcionalidades que, si bien no necesariamente se implementen en el corto plazo, puedan servir como base para futuras iteraciones y evolución de la plataforma.

Metodología

- Se trató de una validación tardía, cuando el sistema ya estaba prácticamente completo.
- A cada usuario se le presentó la aplicación desde la pantalla de login, con la consigna de que estaban en búsqueda de empleo y habían llegado al sistema por recomendación de un amigo.
- Se les permitió interactuar libremente, sin asistencia directa, para observar el comportamiento natural.
- Únicamente se intervino en caso de bloqueos o errores que no podían resolver por cuenta propia debido al estado de desarrollo.
- Se preparó previamente el entorno con:
    - Datos cargados en la base para simular un uso real.
    - Creación de una cuenta de empresa ficticia, utilizada para publicar vacantes que luego aparecían a los candidatos durante la prueba.

Disposición de la Prueba

1. Creación de cuenta de candidato y configuración inicial.
2. Exploración libre de funcionalidades del sistema (dashboard, perfil, listado de vacantes).
3. Simulación de flujo completo:
    - Publicación de ofertas desde la empresa ficticia.
    - *Matching* y recomendación de puestos al candidato.
    - Seguimiento de estados y recepción de notificaciones.
    - Revisión de feedback, entrevistas y actualizaciones de candidaturas.
4. Observación de interacciones en vistas clave:
    - Dashboard principal.
    - Detalle de vacantes.
    - Configuración y edición de perfil.

Participantes

- Cantidad: 3 usuarios.
- Edad: 23 años (todos).
- Perfil académicos variados, todos con estudios terciarios
- Rol asignado en la prueba: candidatos en búsqueda de empleo.

Notas Originales:

Jimena Rodrigo

- En la pantalla de login piensa que el botón no registra y vuelve a clicar varias veces.
- Quiere elegir más de un primer objetivo en el setup del candidato.
- El setup se le hace largo y tedioso.
- No le parecen claras las proficiency de lenguas, se le hacen poco claras.
- Preferiría que en Education y Certification solo se pida mes y año, no día exacto.
- No entiende qué es Field of Study en educación.
- No entiende qué son las Certifications.
- No entiende algunos campos de Soft Skills (ej. Resilience).
- No entiende la diferencia entre Show Interest y Save Job, sugiere agregar un tooltip.
- No sabe qué es HQ (headquarters).
- No entiende cómo funciona el filtro de Saved Jobs en el dashboard.
- Le molesta que los jobs no estén ordenados por etapa.
- Los tooltips le parecen botones y además demoran mucho.
- No le parece importante la gestión de Interview, pero sí le gusta poder ver agenda y links.
- Quiere usar filtros con múltiples opciones (ej. seleccionar varios estados de job).
- Le gustaría ver la compensation en los detalles de cada job.
- Le gustaría ver notificaciones dentro de la app, no solo por mail o WhatsApp.
- Querría poder apagar notificaciones para no recibir tantos correos.
- Percibe la navegación como intuitiva, la app bonita y fácil de entender.
- Algunos cargadores demoran y no siempre se indica que está cargando.
- Le gusta que el setup no sea automático con IA, prefiere hacerlo manual.
- Sugiere agregar Job Experience al setup.
- En general entiende bien qué se le pide, salvo algunos campos poco claros.
- Le encantó cómo se muestra el estado actual de la vacante y la transparencia del proceso.
- Piensa que hay demasiadas notificaciones, le gustaría personalizarlas.
- Valora poder ver estadísticas y mejorar su perfil con el dashboard.

Federico Méndez

- Siente que demora en cargar las ventanas iniciales y no se indica.
- No entiende qué son los objetivos y quiere elegir más de un Primary Objective.
- No logra entender qué es aleatorio y qué no.
- Parece no leer las descripciones de ayuda en el setup.
- Querría poder escribir en los combos (ej. hard skills) en vez de buscar manualmente.
- El setup se le hace largo.
- Le gustaría cargar CV y luego editar.
- Preferiría solo mes y año en Education y Certification.
- No entiende qué son las Certifications.
- Los combos múltiples no deberían cerrarse al elegir, se le indicó que se va a mejorar.
- No entiende bien las Soft Skills (si son propias o deseadas, ni algunos términos).
- Quiere ver la Compensation en cada trabajo.
- No entiende bien las diferencias entre ventanas de Jobs: Opportunity, Saved, In-Process.
- No lee los mensajes de descripción de cada ventana.
- No entiende bien la navegación en Profile Settings.
- Le gusta mucho ver estados de procesos.
- También piensa que los tooltips son botones.

Santiago Debevec

- Comentarios similares a Federico.
- Siente que demora mucho en cargar.
- No entiende qué son los objetivos.
- Le gustaría poder cargar CV para autocompletar (o usar IA).
- Le gusta ver el estado de la vacante.
- Tiene confusión con campos durante el setup.
- Le resulta largo de más, se muestra aburrido/frustrado.
- No entiende bien la navegación en Profile Settings.
- No entiende la diferencia entre jobs guardados y jobs con interés.
- No entiende qué representa “demostrar interés”.
- No distingue bien entre aplicación de *scouting* y postulación tradicional.
- Problemas con tooltips (piensa que son botones).

- Quiere ver la compensation en cada vacante.
- Le molesta no saber cuánto se paga en cada puesto.
- No entiende algunos campos opcionales.
- Le gustan las notificaciones, pero siente que son demasiadas en procesos múltiples.
- Querría ver notificaciones dentro de la app.
- Le parecen buenas las funcionalidades y la transparencia del sistema.
- Valora recibir feedback y notificaciones de descarte o selección.
- Le gusta ser notificado cuando está en el top 5.
- Duda sobre secciones donde puede seleccionar hasta 5 elementos, no entiende si es exacto o máximo.
- Siente que algunos botones no funcionan, pero es un problema de demora de la UI.

Conclusiones

Impresiones Generales

- Los usuarios comprendieron bien la navegación y funcionalidad principal de la plataforma. La interacción resultó intuitiva y no hubo grandes confusiones sobre el propósito del sistema.

- Se valoró positivamente la propuesta de *scouting* automatizado con IA, en contraste con modelos tradicionales de postulación manual. Se destacó la utilidad de recibir recomendaciones basadas en perfil y no depender solo de ser “el primero en postularse”.

- Los participantes resaltaron la transparencia del sistema, tanto en el porcentaje de compatibilidad con vacantes como en la visibilidad de los estados de cada proceso (rechazado, en revisión, entrevista, etc.). Esto fue señalado como un diferencial frente a otras plataformas que suelen ser percibidas como “cajas negras”.

- En general, las funcionalidades fueron calificadas como suficientes y bien desarrolladas para un MVP, incluso destacando que la plataforma ya se percibe utilizable en un contexto real.

Hallazgos Comunes

- Tiempos de carga y feedback visual insuficiente: los usuarios percibieron demoras en pantallas y botones, generando confusión al no saber si el sistema estaba respondiendo.

- Setup largo y tedioso: aunque comprendido, el proceso inicial fue señalado como extenso. Se pidió simplificación, ayudas adicionales (ejemplos, placeholders) y, en algunos casos, posibilidad de autocompletar con CV o IA.

- Campos poco claros: elementos como Field of Study, Certifications o ciertas Soft Skills no resultaron comprensibles para los usuarios sin contexto extra.

- *Tooltips* y guías: algunos botones y campos fueron malinterpretados (ej. *show interest vs save job*). Se sugirió incorporar tooltips y aclaraciones rápidas.

- Filtros limitados: se pidió poder seleccionar múltiples opciones en filtros (ej. estados de vacantes) y mejorar la comprensión de secciones como Saved, Opportunities, In-Process.

- Notificaciones: fueron valoradas positivamente, pero se mencionó exceso en algunos casos. Los usuarios quieren más control: notificaciones internas dentro de la app y posibilidad real de desactivar correos.

- Compensation: todos coincidieron en que es relevante mostrar la remuneración de cada vacante en la vista de detalle y listados.

Valoraciones Positivas

- Los usuarios consideran que la idea central y la propuesta de valor están bien logradas.

- Destacaron que la aplicación los mantiene informados en todo momento sobre el estado de su postulación.

- Apreciaron poder visualizar métricas personales y estadísticas de vacantes, sintiendo que la plataforma les ayuda a mejorar su perfil y entender su progreso.

- La interfaz fue percibida como clara y estéticamente atractiva.

Posibles Mejoras Identificadas

- Incluir pantallas o indicadores de carga para mejorar la experiencia en tiempos de espera.
- Simplificar y guiar el setup inicial, con ejemplos, explicaciones o placeholders que aclaren campos poco familiares.
- Permitir filtros múltiples y más flexibles en búsquedas y listados.
- Incorporar tooltips o microtextos explicativos en botones clave para evitar confusiones.
- Dar mayor control sobre notificaciones, priorizando notificaciones internas en la app y permitiendo configuración granular.
- Mostrar siempre la compensación en vacantes de manera visible.
- Explorar a futuro opciones como carga de CV para autocompletar información (aunque no es prioridad inmediata).

En resumen, la validación confirmó que la plataforma es funcional y bien recibida, con comentarios centrados más en usabilidad (UI/UX) que en fallas graves de lógica o *matching*. Los hallazgos apuntan a mejoras incrementales en claridad, feedback visual y control de la experiencia del usuario, sin cuestionar la propuesta de valor central.

16.23 Tabla comparativa de LLMs

Nota: La información de precios y características documentada en este anexo corresponde a Octubre de 2024. Los servicios de LLMs actualizan regularmente sus modelos, características, estructuras de precios y límites de tier gratuito, por lo que estos datos podrían no reflejar la actualidad.

OpenAI

- **GPT-4**
    - $30 (entrada) - $60 (salida) por 1M tokens
    - No hay info sobre Cache o Batch
    - Rendimiento rápido
    - Contexto 8K.
    - Opcion Contexto 32K por el doble de precio
- **GPT-3.5-turbo**
    - $1.50 (entrada y salida) por 1M tokens
    - No hay info sobre Cache o Batch
    - Modelo optimizado para costo eficiente
    - Contexto 4K.
- **GPT-4o**: $2.50 (entrada) - $10.00 (salida) por 1M tokens
    - Opción Batch 50% off sin opción de caché
    - Opción con caché 50% off
    - Rendimiento rápido
    - Contexto 128K.
    - Tiene procesamiento de audio
- **GPT-4o mini**
    - $0.15 (entrada) - $0.60 (salida) por 1M tokens
    - Opción Batch 50% off sin opción de caché
    - Opción con caché 50% off
    - Modelo optimizado
    - Tareas menos complejas
- **OpenAI o1-preview**
    - $15.00 (entrada) - $60.00 (salida) por 1M tokens
    - Sin Opción Batch
    - Opción con caché 50% off
    - Optimizado para lógica y complejidad.
    - Razonamiento avanzado para tareas complejas.
- **OpenAI o1-mini**: $3.00 (entrada) - $12.00 (salida) por 1M tokens
    - Sin Opción Batch
    - Opción con caché 50% off
    - Especializado en matemáticas y ciencia.
    - El uso de Batch tiene una demora de hasta 24 hrs

Gemini: versiones gratuitas

- **Gemini 1.5 Flash-8B**
  - Limites: 15 RPM, 1M TPM, 1,500 RPD
- **Gemini 1.5 Flash**
  - Limites: 15 RPM, 1M TPM, 1,500 RPD
- **Gemini 1.5 Pro**
  - Limites: 2 RPM, 32K TPM, 50 RPD

Gemini: versiones de pago

- **Gemini 1.5 Flash-8B**
  - $0.0375 (entrada) - $0.15 (salida) por 1M tokens
  - Límites: 4,000 RPM, 4M TPM
  - $0.01875 por M de tokens de caché por hora
  - Mas de 128K Tokens $0.075 (entrada) - $0.30 (salida) por 1M tokens
- **Gemini 1.5 Flash**
  - $0.075 (entrada) - $0.30 (salida) por 1M tokens
  - Limites:2,000 RPM, 4M TPM
  - $0.01875 por M de tokens de caché por hora
  - Mas de 128K Tokens $0.15 (entrada) - $0.60 (salida) por 1M tokens
  - 1M Tokens de contexto
- **Gemini 1.5 Pro**
  - $1.25 (entrada) - $5.00 (salida) por 1M tokens
  - Limites: 1,000 RPM, 4M TPM
  - $0.3125 por M de tokens de caché por hora
  - Mas de 128K Tokens $2.50 (entrada) - $10.00 (salida) por 1M tokens
  - 2M Tokens de contexto

16.24. Evidencia - Swagger

A continuación se adjuntan secciones de la documentación de swagger generada a modo de evidencia.