Universidad ORT Uruguay

Facultad de Ingeniería

DATIUM: Plataforma para la automatización financiera y gestión de ventas de terrenos a plazo

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

Mateo González – 251555

Federica Cabrera – 255163

Ana Gutman – 260956

Matías Wildbaum – 220100

Tutor: Darío Macchi

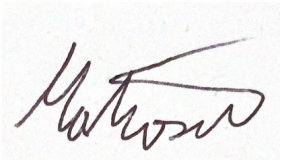
2026

Declaración de autoría

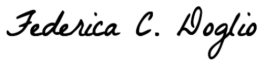
Nosotros, Mateo González, Federica Cabrera, Ana Gutman y Matías Wildbaum, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos el Trabajo Final de Carrera de Ingeniería en Sistemas en la Universidad ORT Uruguay;
- 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 qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Mateo Gonzalez
10/03/2026



Matías Wildbaum
10/03/2026



Federica Cabrera
10/03/2026

Ana Gutman
10/03/2026

Dedicatoria

Este trabajo está dedicado a nuestras familias y seres queridos, quienes nos acompañaron a lo largo de toda la carrera y nos brindaron su apoyo incondicional durante los momentos más desafiantes del proceso.

También lo dedicamos a todos los docentes que formaron parte de nuestra formación académica, cuyo compromiso y dedicación fueron fundamentales para llegar a esta instancia final de nuestra carrera.

Agradecimientos

Queremos agradecer en primer lugar a nuestro tutor, Darío Macchi, por su acompañamiento, orientación y dedicación durante todo el desarrollo del proyecto. Sus aportes y observaciones fueron fundamentales para mejorar la calidad del trabajo y guiarnos en las distintas etapas del proceso.

Agradecemos también a nuestro cliente, Terrazas de Neptunia, por confiar en el equipo para abordar una problemática real de su operativa diaria y por su disposición para participar en entrevistas, revisiones y validaciones durante el desarrollo del sistema.

Asimismo, extendemos nuestro agradecimiento a la Universidad ORT Uruguay y a la Facultad de Ingeniería, por brindarnos el entorno académico y las herramientas necesarias para aplicar de forma práctica los conocimientos adquiridos durante la carrera.

Finalmente, agradecemos a nuestras familias y amigos por su apoyo constante, comprensión y motivación durante todos estos años de formación.

Abstract

El presente trabajo describe el diseño, desarrollo e implementación de DATIUM, una plataforma web orientada a centralizar y automatizar la gestión operativa asociada a la comercialización de terrenos mediante contratos de financiación.

El proyecto surge a partir de la necesidad de un emprendimiento inmobiliario de reemplazar un esquema administrativo basado en planillas de cálculo, el cual generaba fragmentación de la información, dificultades para el seguimiento de contratos y un alto riesgo operativo en el cálculo de cuotas, conversiones monetarias y recargos por mora.

Para abordar esta problemática se diseñó una solución que centraliza la información del negocio y automatiza los principales procesos financieros asociados a la gestión de contratos y cobranzas. El sistema incorpora un motor de cálculo financiero automatizado, capaz de operar con las unidades monetarias que maneja el emprendimiento (dólar estadounidense (USD), Unidad Indexada (UI) y Unidad Reajustable (UR)). Este motor se integra de manera automatizada con datos oficiales del Banco Central del Uruguay (BCU) para garantizar la consistencia y la reproducibilidad histórica de todos los cálculos.

La solución fue desarrollada utilizando una arquitectura web basada en FastAPI para el *backend*, React para el *frontend* y una infraestructura desplegada en AWS, gestionada mediante Infrastructure as Code (IaC) con Terraform. Adicionalmente, el sistema incorpora funcionalidades extendidas orientadas a optimizar la operativa diaria y reducir la fricción con los usuarios. Estas incluyen un canal de autoservicio para compradores integrado mediante la API de WhatsApp, un módulo avanzado de consultas analíticas sobre la base de datos asistido por inteligencia artificial generativa, un panel interactivo (*dashboard*) para el seguimiento de métricas clave y un sistema de notificaciones automáticas por correo electrónico.

El desarrollo del proyecto se llevó a cabo aplicando metodologías ágiles y estándares de ingeniería de software, priorizando la validación iterativa con el cliente y la estabilidad del producto. El aseguramiento de la calidad se sostuvo sobre un modelo robusto que incluye alta cobertura de pruebas automatizadas, integración y entrega continua (CI/CD) y revisiones por pares.

Como resultado, se obtuvo una plataforma operativa utilizada cotidianamente con datos reales del negocio, que permite consolidar la información, reducir los errores operativos manuales y facilitar el análisis integral del estado financiero de sus contratos.

Palabras clave

Gestión de contratos, Cobranzas, Sistema de información, Automatización de procesos, Ingeniería de software, Plataforma web, Gestión inmobiliaria.

Glosario

- **Abitab / Redpagos:** Redes de cobranzas y pagos extrabancarios ampliamente utilizadas en Uruguay para abonar servicios, cuotas y otras obligaciones.
- ***Allowlist*:** Sistema de control de acceso basado en una lista de direcciones de correo electrónico permitidas, gestionada por administradores, que restringe quiénes pueden iniciar sesión en la plataforma
- **Alta, Baja y Modificación (ABM)**: Operaciones para administrar entidades: crear, editar y eliminar registros.
- ***Amazon Web Services* (AWS):** Plataforma de infraestructura en la nube utilizada para desplegar la solución.
- **API Gateway:** Servicio que expone endpoints HTTPS y enruta solicitudes hacia el *backend*.
- ***Application Programming Interface* (API)**: Interfaz para que sistemas se comuniquen mediante *requests/responses*.
- **Atomic Design**: Patrón de arquitectura y diseño de componentes de interfaz de usuario que organiza los elementos visuales jerárquicamente en átomos, moléculas y organismos
- ***Backend:*** Lógica del servidor de la aplicación (desarrollada en FastAPI) encargada de ejecutar las reglas de negocio, realizar los cálculos financieros, conectarse a la base de datos y gestionar las integraciones externas
- **BCU (Banco Central del Uruguay)**: Fuente oficial de cotizaciones y tasas utilizadas en cálculos financieros.

- ***Bot / Chatbot***: Asistente conversacional automatizado. En el sistema se utiliza uno hacia el exterior vía WhatsApp para consultas de compradores, y uno interno asistido por inteligencia artificial para usuarios administradores
- ***Comma-Separated Values* (CSV):** Archivo de texto plano diseñado para almacenar datos tabulares
- **ConfigCat:** Servicio de *feature flags* para habilitar/deshabilitar funcionalidades por configuración.
- ***Continuous Delivery/Deployment* (CD):** Etapa del *pipeline* asociada al despliegue a un entorno (según el flujo definido).
- ***Continuous Integration* (CI)**: Validaciones automáticas al integrar cambios (por ejemplo *tests* y *build*).
- ***Cooldown*:** Intervalo mínimo de espera entre la ejecución repetida de una misma acción sobre un mismo destinatario, utilizado para evitar comportamientos excesivos como el envío repetitivo de notificaciones por correo electrónico.
- ***Daily / Stand-up*:** Reuniones breves de sincronización del equipo de desarrollo orientadas a revisar avances y detectar impedimentos dentro del marco de trabajo ágil
- ***Dashboard***: Panel interactivo en la plataforma orientado al seguimiento visual de indicadores clave de rendimiento ejecutivos, como la morosidad y la recaudación
- ***Date-driven:*** Enfoque de gestión de proyectos donde la fecha de entrega (tiempo) es fija

- ***Design tokens:*** Variables centralizadas que almacenan las decisiones de diseño (paleta de colores, tipografías, espaciados y sombras) para asegurar la consistencia visual y facilitar la mantenibilidad en el desarrollo del *frontend*

- **Docker:** Herramienta de contenedores utilizada para empaquetar y ejecutar el *backend*.

- **EC2:** Servicio de AWS para ejecutar servidores virtuales donde corre la aplicación.

- ***Fallback:*** Mecanismo de contingencia o respaldo utilizado por el sistema para asegurar la continuidad operativa cuando falla un servicio principal (por ejemplo, la obtención automatizada de cotizaciones del BCU en días inhábiles)

- ***Feature flag*:** Configuración que permite habilitar o deshabilitar una funcionalidad sin cambiar el *core*.

- ***Frontend:*** Interfaz visual y aplicación web con la que interactúan los usuarios administradores, responsable de presentar la información

- **Gherkin:** Lenguaje estructurado y legible por humanos utilizado para especificar el comportamiento del software

- **Gitflow:** Estrategia o modelo de gestión de ramas en repositorios de código que organiza el desarrollo en ramas independientes y estructura las entregas a los entornos de integración y producción

- ***Golden Path:*** Secuencia ideal de pasos que recorre un usuario para completar con éxito sus tareas dentro del sistema

- **Impuesto al Valor Agregado (IVA)**: Impuesto indirecto que grava el consumo de bienes y servicios, aplicándose sobre el valor añadido en cada etapa de producción y comercialización

- ***Infrastructure as Code* (IaC):** Definición versionada de infraestructura (por ejemplo con Terraform) para reproducibilidad de entornos.
- ***JSON Web Token* (JWT):** Token utilizado para autenticar solicitudes entre *frontend y backend*.
- **Migraciones:** Scripts versionados para evolucionar el esquema de base de datos de forma controlada.
- **Mora:** Recargo por atraso en el pago de una cuota, calculado según reglas del negocio.
- **MySQL:** Motor de base de datos relacional utilizado para persistencia del sistema.
- **OAuth 2.0**: Protocolo usado para inicio de sesión con Google.
- **Padrón:** Identificador del terreno/inmueble utilizado como referencia en contratos y reportes.
- ***Pipeline / Workflow:*** Secuencia automatizada de pasos ejecutada en herramientas de integración continua (como GitHub Actions) para realizar pruebas, compilar el código y desplegar la aplicación en los distintos entornos
- ***Planning Poker:*** Técnica de estimación ágil basada en el consenso del equipo, utilizada para dimensionar la complejidad y el esfuerzo requerido para implementar una historia de usuario
- ***Prompt engineering:*** Técnica de diseño, estructuración y optimización de las instrucciones (*prompts*) que se envían a un modelo de inteligencia artificial generativa, con el objetivo de obtener resultados precisos y seguros
- ***Pull Request (*PR*):*** Petición formal y documentada para integrar cambios de código de una rama de desarrollo a otra principal

- **Radon:** Herramienta para ejecutar análisis de métricas de código Python.
- ***Release:*** Incremento o versión estable del producto de software que agrupa un conjunto de funcionalidades finalizadas y se entrega para su uso en un entorno productivo
- **S3:** Servicio de almacenamiento usado para *hosting* del *frontend* como sitio estático
- ***Single Page Application* (SPA)**: Arquitectura de aplicación web que carga una única página inicial y actualiza su contenido dinámicamente, permitiendo transiciones fluidas sin necesidad de recargar el navegador
- ***Spike:*** Tarea de investigación exploratoria dentro de un sprint destinada a reducir la incertidumbre técnica o validar la viabilidad de una solución antes de proceder con su implementación definitiva
- ***Sprint / Sprint Planning / Sprint Review:*** Ciclos de trabajo iterativos y sus ceremonias
- ***Story points:*** Unidad de medida relativa utilizada por el equipo para estimar el esfuerzo, la incertidumbre y la complejidad asociados a la implementación de una historia de usuario
- **Structured Query Language (SQL):** Lenguaje de programación estandarizado diseñado para gestionar, manipular y consultar bases de datos relacionales
- **Terraform:** Herramienta de infraestructura como código usada para definir y desplegar recursos en AWS.
- ***Text-to-SQL*:** Patrón de diseño impulsado por inteligencia artificial que traduce consultas formuladas en lenguaje natural (español) a sentencias estructuradas en código SQL para consultar la base de datos

- ***Trade-off*:** Decisión donde se equilibran atributos de calidad en conflicto y se sacrifican beneficios de una característica para mejorar otra
- **UI (Unidad Indexada):** Unidad de cuenta utilizada en Uruguay; su cotización se usa para cálculos y conversiones.
- **UR (Unidad Reajustable)**: Unidad de cuenta utilizada en Uruguay; su cotización se usa para cálculos y conversiones.
- **USD / UYU:** Abreviaturas de monedas (dólar estadounidense / peso uruguayo) usadas en contratos, pagos y reportes.
- ***Webhook*:** Endpoint que recibe eventos desde un sistema externo y dispara procesamiento.
- ***WhatsApp Cloud API* (*Meta*):** API oficial de Meta para integrar mensajería de WhatsApp (*webhooks* y envío de mensajes).
- ***Workaround:*** Solución temporal, manual o alternativa informal desarrollada por los usuarios para sortear una limitación operativa en sus herramientas previas (como las planillas Excel)
- ***Zero Trust:*** Modelo y enfoque de seguridad basado en la premisa de no confiar por defecto en ningún usuario, dispositivo o red, exigiendo una verificación y autorización estricta para acceder a los recursos del sistema

Índice

1. Introducción

El presente trabajo expone el desarrollo de DATIUM, una plataforma web orientada a centralizar y automatizar la operativa asociada a la comercialización de terrenos mediante contratos de financiación y a la gestión de su cobranza. El sistema permite administrar de forma integral la información del negocio, automatizar cálculos financieros relevantes para la operativa y ofrecer mecanismos de seguimiento, control y comunicación a través de reportes, notificaciones automáticas y asistentes conversacionales.

El proyecto se desarrolló en el marco del Trabajo Final de Carrera de Ingeniería en Sistemas de la Universidad ORT Uruguay, dentro del entorno de ORT Software Factory. La solución fue diseñada y construida para un cliente real, con el objetivo de resolver problemáticas concretas de su operativa diaria. A lo largo del desarrollo, la propuesta fue validada de manera iterativa mediante instancias de revisión y demostración, asegurando su alineación con las necesidades del negocio y su aplicabilidad en un contexto real.

1.1. ¿Cómo surge el proyecto?

El proyecto surge a partir de un vínculo previo entre uno de los integrantes del equipo, Matías, y un miembro del equipo de Terrazas de Neptunia. A partir de ese contacto, y conociendo su experiencia en desarrollo de software, se le planteó una necesidad concreta: desarrollar una plataforma que permitiera ordenar y profesionalizar la gestión administrativa del emprendimiento. En un inicio, la propuesta era llevarlo adelante bajo una modalidad *freelance*.

Al analizar la situación y comprender el alcance real del problema, Matías identificó que no se trataba sólo de desarrollar una herramienta puntual, sino de abordar una problemática operativa más amplia, con múltiples aristas técnicas y de negocio. En ese contexto, propuso encarar el desarrollo como Trabajo Final de Carrera, aprovechando

la estructura metodológica y el acompañamiento académico para abordar el problema con mayor profundidad. El cliente aceptó la propuesta.

A partir de esa definición se conformó el equipo y el proyecto comenzó formalmente. Se iniciaron las entrevistas, el relevamiento detallado de información y el análisis exhaustivo de la operativa existente. Sobre esa base se definieron los requerimientos, las prioridades funcionales y las decisiones técnicas que darían forma a la solución.

De esta manera, el proyecto dejó de plantearse como una solución rápida a una necesidad administrativa y pasó a abordarse como el diseño e implementación de un sistema que debía modelar correctamente la lógica del negocio, automatizar reglas financieras y sostener la operativa real del emprendimiento.

1.2. Objetivos

En esta sección se presentan los objetivos definidos por el equipo para el desarrollo de DATIUM. Se organizan en tres niveles: objetivos del producto, objetivos del proyecto y objetivos académicos.

Esta clasificación permite distinguir con claridad:

- qué se busca construir (producto),
- qué resultados se pretende alcanzar con su desarrollo e implementación (proyecto),
- y qué aprendizajes y prácticas se busca consolidar en el marco del Trabajo Final de Carrera (académicos).

Con el fin de definir metas claras y evaluables, los objetivos del producto y del proyecto fueron formulados siguiendo el enfoque SMART [1], que establece que deben ser específicos, medibles, alcanzables, relevantes y acotados en el tiempo. Este enfoque permitió transformar expectativas generales en definiciones concretas que facilitaron su seguimiento durante el desarrollo y su posterior verificación.

Los objetivos académicos, por su parte, se formulan en términos de prácticas y resultados esperados dentro del proceso de formación profesional, vinculados con la aplicación integrada de conocimientos de Ingeniería de Software en un proyecto real.

En los apartados siguientes se detallan los distintos grupos de objetivos. Cada uno se identifica mediante un código que será utilizado posteriormente (ver sección 11.1 Verificación y discusión de cumplimiento de los objetivos)

1.2.1. Objetivos del producto

Los objetivos del producto se orientan a definir las capacidades funcionales principales que debe ofrecer DATIUM como plataforma para apoyar la gestión operativa del emprendimiento.

OProd01) Centralización de la información del negocio: Proveer un punto único de registro y consulta para la información central del negocio (clientes, empresas, etapas, terrenos y contratos), con el fin de facilitar un acceso coherente, mejorar la trazabilidad de las operaciones y reducir la dispersión de datos en planillas y fuentes paralelas.

OProd02) Gestión integral de contratos de venta: Incorporar las funcionalidades necesarias para registrar, consultar y actualizar contratos de venta, contemplando su ciclo operativo y permitiendo el seguimiento del estado y del plan de pagos asociado.

OProd03) Automatización del cálculo y generación de cuotas: Implementar un motor de cálculo que genere cronogramas de cuotas a partir de reglas explícitas del negocio (importes, vencimientos, intereses, moras y conversiones).

OProd04) Registro de pagos con validaciones de consistencia: Permitir el registro de pagos vinculados a contratos, aplicando validaciones que garanticen la integridad de la información financiera y reduzcan la probabilidad de errores derivados de la carga manual.

OProd05) Generación de informes operativos: Brindar herramientas que permitan visualizar información de la operativa relevante para el cliente, y exportar a formatos estándar con el objetivo de facilitar el su trabajo administrativo y de auditoría.

OProd06) Reducción de fricción operativa mediante usabilidad (UX/UI): Diseñar flujos interactivos y vistas consolidadas que reduzcan la carga cognitiva de los usuarios administrativos, aplicando principios de usabilidad para disminuir errores y la dependencia de múltiples planillas.

OProd07) Provisión de un canal de consultas para compradores: Implementar un canal que permita a los clientes consultar su estado de cuenta y vencimientos de forma autónoma, reduciendo la carga operativa derivada de consultas sobre el equipo administrativo.

1.2.2. Objetivos del proyecto

Los objetivos del proyecto se enfocan en los resultados esperados del proceso de desarrollo e implementación de la solución en el contexto real del negocio.

OProy01) Implementación de una solución operativa: Poner en producción una versión operativa de la plataforma dentro del plazo acordado, de modo que sea accesible para el cliente y se utilice en la gestión de datos reales en su operativa cotidiana.

OProy02) Validación iterativa con usuarios: Construir la solución mediante entregas incrementales y validar cada incremento con el cliente y los usuarios operativos, incorporando los ajustes priorizados a partir de su *feedback*.

OProy03) Calidad técnica y sostenibilidad del sistema: Aplicar prácticas de desarrollo que aseguren la robustez del sistema basándose en los atributos del estándar ISO/IEC 25010. Esto abarca la incorporación de pruebas automatizadas, documentación técnica y mecanismos de integración y CI/CD.

OProy04) Satisfacción Operativa del Cliente:

Conseguir que el cliente confirme, mediante encuestas de satisfacción aplicadas al finalizar cada *release*, que DATIUM resulta útil para su operativa cotidiana y contribuye a mejoras concretas respecto al proceso previo basado en planillas de cálculo.

La encuesta evaluará de forma explícita la percepción del cliente según los siguientes indicadores:

- **Adopción:** grado en que la plataforma se consolida como la herramienta principal para la gestión de las nuevas operaciones comerciales y su cobranza asociada, reduciendo la dependencia estructural de planillas.
- **Estabilidad e integridad:** percepción de seguridad y coherencia en los cálculos automatizados (mora, prorrateos, conversiones).
- **Capacidad analítica:** utilidad y accesibilidad de los módulos para la obtención y análisis de información del negocio.
- **Eficiencia operativa interna:** percepción de reducción del esfuerzo y tiempo administrativo requerido para registrar, consultar y procesar la información de ventas y cobranzas.
- **Autonomía del comprador:** disminución percibida en el volumen de consultas repetitivas que recaen sobre el equipo administrativo, atribuible a la disponibilidad del canal de auto-consulta.

Criterio de Éxito: el objetivo se considerará cumplido cuando las encuestas de satisfacción registren un puntaje igual o superior a 4 sobre 5 en cada uno de los indicadores evaluados.

1.2.3. Objetivos académicos

Desde el punto de vista académico, el proyecto tiene como propósito aplicar de manera integrada los conocimientos adquiridos durante la carrera en un contexto real de desarrollo de software.

OAcad01) Ingeniería de requerimientos: Realizar actividades de relevamiento, especificación, validación y verificación de requerimientos funcionales y no funcionales en interacción con un cliente real.

OAcad02) Diseño y justificación técnica de la solución: Diseñar e implementar la arquitectura y el modelo de datos adecuados para la solución, justificando las decisiones técnicas y los *trade-offs* adoptados.

OAcad03) Gestión del proyecto con enfoque ágil: Aplicar prácticas de gestión ágil para la planificación, seguimiento y control del proyecto, promoviendo iteraciones incrementales, adaptación continua y visibilidad del progreso.

OAcad04) Aprendizaje y aplicación de nuevas tecnologías: Aplicar conocimientos adquiridos durante la carrera y explorar nuevas tecnologías, herramientas y prácticas de desarrollo relevantes para la construcción de la solución.

1.3. Entorno conceptual de Software Factory

El Laboratorio de Ingeniería de Software de la Universidad ORT Uruguay, denominado ORT Software Factory (ORTsf) se dedica a la enseñanza de Ingeniería de Software y a la producción de software en forma industrial [2].

ORTsf está abocada fundamentalmente a desarrollar en los alumnos las habilidades que un profesional de las Tecnologías de la Información debe dominar y aplicar. Para esto se ha diseñado un método de enseñanza para estudiantes de fin de carrera, que

apoyados por tutores especializados, trabajan en equipos de desarrollo aplicando prácticas avanzadas de Ingeniería de Software en proyectos reales.

Estos proyectos surgen en colaboración con la industria o como apoyo a las líneas de investigación del departamento. Buscan construir productos que satisfagan a sus clientes, promover el aprendizaje de prácticas reales de ingeniería de software y proveer tecnología probada al mercado.

1.4. Descripción del equipo

El equipo de trabajo se conformada por:

- Mateo González (N.º estudiante: 251555)
- Federica Cabrera (N.º estudiante: 255163)
- Ana Gutman (N.º estudiante: 260956)
- Matías Wildbaum (N.º estudiante: 220100)

El tutor del proyecto fue Darío Macchi, quien acompañó el desarrollo mediante seguimiento y orientación metodológica en el marco de ORT Software Factory.

1.5. Estructura del documento

El presente documento se estructura en once capítulos, además de las referencias bibliográficas y los anexos. En ellos se presenta, de forma progresiva, el problema abordado, la solución desarrollada, el proceso de construcción seguido y las conclusiones obtenidas.

- **Capítulo 1 – Introducción:** origen del proyecto, objetivos del producto, del proyecto y académicos, entorno conceptual de ORT Software Factory, descripción del equipo y estructura del documento.

- **Capítulo 2 – Problema:** contexto del cliente, interesados y principales desafíos identificados al inicio del proyecto.
- **Capítulo 3 – Solución:** solución propuesta, principales funcionalidades y delimitación del alcance, incluyendo sus exclusiones explícitas y su evolución durante el desarrollo.
- **Capítulo 4 – Estrategia de construcción de la solución:** enfoque general adoptado para construir la solución y cronología de las etapas del proyecto.
- **Capítulo 5 – Marco metodológico:** características del proyecto, del cliente, del producto y del equipo, así como la metodología de trabajo y los roles asumidos.
- **Capítulo 6 – Ingeniería de requerimientos:** proceso de relevamiento, especificación, validación y verificación, junto con las funcionalidades definidas y los requerimientos no funcionales considerados.
- **Capítulo 7 – Arquitectura y diseño:** visión general de la arquitectura, atributos de calidad, decisiones arquitectónicas, tecnologías utilizadas, detalle de la solución y prácticas de desarrollo aplicadas.
- **Capítulo 8 – Gestión del proyecto:** herramientas de gestión, etapas e hitos principales, aplicación de Scrum, documentación académica, plan de *releases*, métricas de gestión, comunicación y riesgos del proyecto.
- **Capítulo 9 – Gestión de la calidad:** definición de calidad adoptada en el proyecto, prácticas de aseguramiento aplicadas, métricas y principales conclusiones obtenidas.
- **Capítulo 10 – Gestión de la configuración:** identificación de elementos de configuración, herramientas utilizadas, organización de repositorios, gestión de versiones, cambios e incidentes.

- **Capítulo 11 – Conclusiones:** verificación y discusión del cumplimiento de los objetivos, conclusiones generales, lecciones aprendidas y próximos pasos.
- **Referencias bibliográficas y anexos:** fuentes consultadas y material complementario que respalda y amplía el contenido desarrollado en el cuerpo principal del documento.

2. Problema

2.1. Contexto

El proyecto se desarrolló para Terrazas de Neptunia, un emprendimiento inmobiliario que comercializa terrenos a plazo en el barrio Neptunia (Canelones, Uruguay). Actualmente la empresa administra aproximadamente 130 contratos activos, con plazos de financiación que suelen oscilar entre 120 y 150 cuotas, y una proyección de ocupación del predio que puede alcanzar hasta alrededor de 800 contratos a lo largo del desarrollo del negocio. Cada contrato genera un historial acumulado de cuotas, pagos y cálculos financieros que debe conservarse y consultarse a lo largo de muchos años, por lo que la volumetría de información y persistencia histórica constituyen requisitos operativos relevantes.

La operativa inicial era mayoritariamente manual y distribuida en planillas de Excel: una hoja por padrón agrupadas en archivos por "etapas" (subdivisiones del predio que agrupan padrones), con registros de pagos cargados manualmente desde comprobantes recibidos por WhatsApp o correo, y todos los cálculos financieros (prorrateos, redondeos y reconversiones por cotización) en las mismas hojas. Además, las planillas incluían una columna de notas usadas para registrar aclaraciones o situaciones particulares asociadas a los pagos o contratos.

Este enfoque generaba cuatro problemas estructurales principales:

1) **Riesgo operativo elevado:** la dependencia de entradas y procedimientos manuales incrementa la probabilidad de errores y la dependencia del conocimiento individual.
2) **Falta de consolidación y visibilidad global:** la información fragmentada impide disponer de una fuente única de verdad para consultas y reportes inmediatos.

3) **No escalabilidad:** el crecimiento en cantidad de contratos y en años de financiación aumenta exponencialmente la complejidad operacional sin herramientas que la soporten.

4) **Baja capacidad analítica:** las planillas almacenan datos, pero no proporcionan indicadores ni representaciones visuales que permitan analizar la información rápidamente y apoyar la toma de decisiones.

En el día a día, estos problemas se traducen en dolores operativos concretos: demoras y esfuerzo para responder preguntas básicas del negocio (por ejemplo, cuánto se cobró en un período, cuánto corresponde a cuota versus mora o cuánto se descontó en comisiones), así como una comunicación fragmentada con los clientes a través de WhatsApp, correo electrónico y llamadas. Esta dispersión provoca pérdida de información, duplicación de consultas y dificultades para reconstruir el historial de cada contrato.

Frente a este escenario de dispersión de datos y dependencia de planillas de cálculo, quedó en evidencia que el problema no se resolvía únicamente trasladando la información a una base de datos o a una interfaz web. La situación exigía impulsar una transformación digital más profunda del emprendimiento. Por este motivo, el proyecto define su alcance orientándose al diseño de una plataforma centralizada que permita sustituir progresivamente los procesos informales heredados, consolidándose como el nuevo eje estructurador de la operativa comercial y financiera del negocio.

2.2. Interesados

El ecosistema de usuarios del sistema se divide en cuatro grupos claramente diferenciados según su nivel de acceso, sus responsabilidades y su relación con la plataforma.

- **Núcleo administrativo y de dirección de Terrazas de Neptunia (Usuarios Internos):** Constituye el equipo central de confianza del emprendimiento,

encargado de la administración de ventas, la gestión de cobranzas y la toma de decisiones estratégicas. Dado que este equipo gestiona información financiera sensible, opera bajo criterios estrictos de confidencialidad. Son los operadores exclusivos de la interfaz web del sistema, cuyo acceso se controla mediante un mecanismo de invitaciones gestionado internamente.

- **Compradores (Usuarios Externos)**: Corresponden a los clientes que han adquirido terrenos mediante planes de financiación y representan el mayor volumen de usuarios concurrentes. Su principal necesidad consiste en consultar de forma autónoma su estado de cuenta y los próximos vencimientos. No interactúan con el portal administrativo, sino exclusivamente a través del canal de autoservicio automatizado (*bot* de WhatsApp) y, en menor medida, mediante otros canales de contacto como correo electrónico o llamadas telefónicas.

- **Contador externo y encargados de auditoría**: Son actores periféricos que no intervienen en la operativa diaria ni requieren credenciales de acceso a la plataforma, pero que actúan como consumidores relevantes de la información financiera. Utilizan las exportaciones estructuradas en Excel generadas por el sistema de reportes para realizar conciliaciones, cierres contables y auditorías de los contratos.

- **Equipo de Mantenimiento:** Personal técnico responsable de la administración y soporte del sistema en producción (DATIUM). Sus tareas incluyen el monitoreo de la infraestructura y del rendimiento, la gestión de *feature flags* para habilitar o deshabilitar funcionalidades sin afectar el código productivo, y el mantenimiento evolutivo de la plataforma.

2.3. Desafíos del proyecto

El principal desafío del proyecto estuvo asociado a la madurez del dominio. Al inicio del relevamiento, gran parte de la operativa del negocio se encontraba reflejada en planillas de cálculo y prácticas informales, por lo que numerosos casos reales y reglas

específicas no estaban completamente explicitados. Esta falta de estandarización se hizo evidente durante el análisis de los archivos Excel heredados, los cuales presentaban inconsistencias estructurales, variabilidad en los criterios históricos e incluso errores en los datos.

Este contexto no sólo complejizó la comprensión inicial del negocio, sino que también impactó en decisiones relevantes de la solución. En particular, las irregularidades detectadas en la información histórica condicionaron la estrategia prevista para la migración de datos, aspecto que se retoma en la sección 3.3.2 Evolución y ajuste del alcance

A esto se sumaron demoras en la entrega de información por parte del cliente, especialmente en la definición de reportes y de los flujos esperados para el *bot* de WhatsApp. Asimismo, durante el *feedback* del primer *release* surgieron funcionalidades adicionales y detalles de gestión que no habían sido contemplados inicialmente, lo que implicó replanificaciones y nuevas iteraciones.

En paralelo, la habilitación del *bot* de WhatsApp requirió completar el proceso de validación de la cuenta de negocio en Meta, lo que extendió aproximadamente un mes la posibilidad de realizar pruebas completas en entorno productivo. Como parte de ese proceso, fue necesario además desarrollar y publicar una landing page pública del producto, esfuerzo que no había sido contemplado originalmente y que debió resolverse durante la ejecución del proyecto.

Finalmente, a nivel interno, la mudanza de una integrante del equipo al exterior durante el desarrollo exigió una redistribución de tareas y una mayor coordinación para sostener el ritmo de trabajo.

3. Solución

3.1. Solución propuesta

La solución implementada consiste en una plataforma digital orientada a centralizar, estructurar y automatizar la operativa asociada a la comercialización de terrenos y a la gestión de sus cobranzas. El sistema reemplaza el ecosistema fragmentado de planillas de cálculo y procesos manuales por una fuente única de verdad, una vez completado el proceso de transición, que formaliza las reglas de negocio, asegura la reproducibilidad de los cálculos financieros y mejora la trazabilidad de las operaciones.

Para asegurar que la plataforma optimice los procesos críticos del negocio por encima de otras tareas secundarias, el diseño de la solución y la priorización de sus capacidades se estructuraron en torno a dos *Golden Paths*. El primero corresponde al usuario administrador y abarca el ciclo principal de la operación: desde el alta de contratos y la generación automática de cuotas hasta el registro de pagos, el cálculo de mora y la exportación de reportes para control y auditoría. El segundo flujo corresponde al usuario comprador y está orientado al autoservicio mediante la consulta de su estado de cuenta y de los próximos vencimientos a través de un canal automatizado (ver sección 6.1. Proceso).

A nivel estructural y como se aprecia en la Figura 1, la solución se compone de:

- Una aplicación web para usuarios administradores internos
- Un *backend* que centraliza la lógica de negocio y expone una API REST
- Una base de datos relacional para la persistencia transaccional
- Integraciones con servicios externos
- Habilitación de canales automatizados de consulta.

El modelo de datos formaliza la estructura física y comercial del emprendimiento (clientes, empresas, etapas y terrenos) y soporta el ciclo completo de venta y cobranza. La plataforma permite la creación de contratos altamente flexibles, admitiendo hasta cuatro compradores por padrón y definiendo reglas específicas como la moneda de financiación (USD, UI, UR), plazos y modalidad de pago. A partir de estas condiciones, el sistema genera automáticamente el cronograma de cuotas aplicando prorrateos, redondeos controlados y reglas de vencimiento adaptadas a fin de mes

El cálculo financiero es central en la solución. El *backend* se encarga de validar la consistencia de cada pago y de calcular la mora por atraso en función de las fechas de vencimiento, aplicando una tasa de interés para compra de vivienda específica a la moneda del contrato. Cuando corresponde, este cálculo de mora contempla también la aplicación del IVA.

Para asegurar la reproducibilidad y coherencia temporal de estos cálculos, el sistema obtiene de forma automatizada las cotizaciones (UI, UR, USD) y las tasas de vivienda desde fuentes oficiales del BCU [3]. Estos valores persisten en la base de datos junto con mecanismos de contingencia (*fallback*) ante la falta de datos en días inhábiles, garantizando así la continuidad operativa.

Adicionalmente, el procesamiento financiero contempla la complejidad de los pagos realizados a través de redes de cobranza externas (Abitab / Redpagos). El sistema calcula automáticamente la comisión fija de estas redes expresada en UI, la convierte a UYU según la cotización del día, y valida la consistencia matemática entre el importe bruto pagado por el cliente, la comisión retenida y el importe neto que efectivamente se aplica a la cuota

En términos de control y seguimiento, la plataforma incorpora un módulo de reportes interactivos y exportables a formato Excel. Los reportes cubren cinco perspectivas distintas del negocio: cobranzas, ventas, ingreso por ejercicio, cuotas vencidas y

deudores. Esto se complementa con un *dashboard* ejecutivo, habilitable por configuración, orientado al seguimiento de indicadores de morosidad y recaudación.

Asimismo, se incorporaron dos asistentes conversacionales. Hacia el exterior, se integró un *bot* de WhatsApp mediante la API oficial de Meta, permitiendo a los compradores consultar de forma autónoma el próximo vencimiento, el valor de la próxima cuota, la cantidad de cuotas pagas, pendientes y vencidas, así como su estado de cuenta completo. Hacia el interior, se implementó un *chatbot* web impulsado por inteligencia artificial (*text-to-SQL*) que permite al equipo administrativo realizar consultas analíticas complejas sobre la base de datos utilizando lenguaje natural.

Finalmente, desde el punto de vista de la seguridad, el sistema utiliza un flujo de inicio de sesión con OAuth 2.0, restringido por una lista de habilitados gestionada por administradores. La autorización se gestiona mediante tokens JWT y un modelo de roles binario (administrador / no administrador), lo que, sumado a un registro de eventos y métricas de uso, garantiza un entorno de operación cerrado, trazable y seguro.

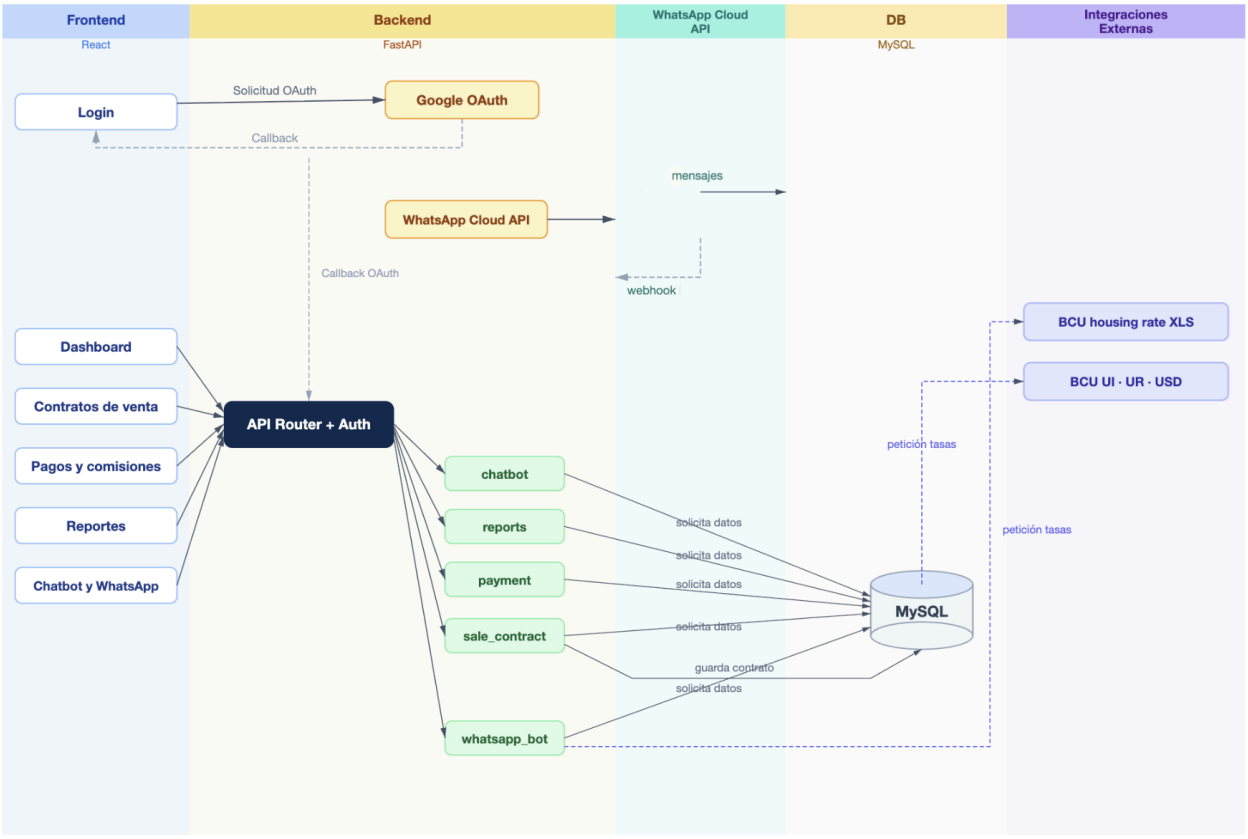


Figura 1. Arquitectura general y principales integraciones externas

3.2. Principales funcionalidades

Para resolver los dolores estructurales identificados en la operativa inicial, las funcionalidades implementadas se estructuraron en siete bloques estratégicos, alineados con los procesos críticos del negocio:

1. **Gestión de entidades maestras (ABM):**

   La plataforma permite administrar los datos maestros necesarios para operar el negocio, asegurando integridad y consistencia entre los elementos de la estructura comercial.

En esta categoría se incluyen:

- Gestión de clientes, con validaciones específicas por tipo (persona física o jurídica) y estandarización de datos de contacto para evitar inconsistencias.
- Gestión de empresas y etapas que estructuran el emprendimiento.
- Gestión de padrones vendidos y validando consistencia al editar.
- Aplicación de restricciones de negocio preventivas al eliminar información (por ejemplo, impedir el borrado de terrenos asociados a contratos activos).

2. **Ventas y contratos:**

El módulo de ventas permite crear contratos asociando compradores con un padrón disponible, administrando su estado durante todo el proceso de financiación.

En esta categoría se incluyen:

- Creación de contratos con asociación de hasta cuatro compradores, validando moneda, importes y fechas.
- Generación automática del cronograma de cuotas mediante prorrateo del capital financiado, redondeo controlado y asignación precisa de vencimientos mensuales.
- Soporte para flujos específicos, como contratos en USD bajo modalidad contado (generando una única cuota) o en cuotas fijas preestablecidas
- Flujo de cancelación de contratos y posterior liberación de sus padrones para su reutilización comercial, así como acceso a los mismos junto con todo su historial de información original.

3. **Cobranza y cálculos financieros:**

   Constituye el motor de la operativa diaria, centrado en el procesamiento transaccional de pagos y la automatización de la lógica financiera.

   En esta categoría se incluyen:

   - Registro de pagos con validaciones de negocio rigurosas y manejo de excedentes o faltantes entre pagos consecutivos.
   - Cálculo automático de recargos por mora y sus intereses asociados, aplicando la tasa de vivienda correspondiente y contemplando el IVA cuando el contrato lo requiere.
   - Deducción y conversión dinámica de comisiones fijas retenidas por redes de cobranza externas (Abitab y Redpagos).
   - Integración automatizada con el BCU para la obtención y persistencia de cotizaciones de moneda (UI, UR, USD) y tasas de vivienda, asegurando la reproducibilidad histórica de los cálculos.

4. **Reportes y exportación:**

   La plataforma incorpora herramientas para el seguimiento, análisis y auditoría, mediante previsualización en pantalla y exportación estructurada a formato Excel.

   En esta categoría se incluyen:

   - Reporte de cobranza: por período y padrón, discriminando totales cobrados, adjudicación a cuotas, mora y comisiones de redes.
   - Reporte de ventas: con totales financiados, señas y posibilidad de conversión a moneda destino.

- Reporte de ingreso del ejercicio: por padrón, separando cobranzas previas, del período y composición de mora e IVA.
- Reporte de cuotas vencidas: a una fecha de corte, indicando cantidad, proporción de riesgo y montos por moneda.
- Reporte de deudores: detallando cantidades de cuotas pagas, pendientes, vencidas y deuda total.

5. **Seguimiento ejecutivo y consultas internas asistidas:**

   El sistema incorpora capacidades analíticas y de automatización avanzadas, habilitables de forma segura mediante configuraciones dinámicas (*feature flags*).

   En esta categoría se incluyen:

   - *Dashboard* interactivo orientado al seguimiento de indicadores de morosidad y recaudación, con filtros por rango de fechas, empresa y etapa.
   - Envío automático de notificaciones por correo electrónico a compradores con cuotas en mora, regulado por reglas de *cooldown*.
   - *Chatbot* interno que permite a los administradores realizar consultas en lenguaje natural, devolviendo respuestas tabulares con visualización del código SQL generado y exportación a CSV.

6. **Herramientas de filtrado y navegación operativa:**

   Para facilitar la gestión diaria de la información, el sistema incorpora mecanismos de filtrado y exploración que permiten navegar grandes volúmenes de datos de forma eficiente.

   En esta categoría se incluyen:

- Filtros aplicables en las principales vistas del sistema para acotar resultados por distintos criterios de negocio.
- Sidebar de filtros avanzada en la lista de padrones activos, que permite ordenar y filtrar por múltiples atributos del terreno o del contrato asociado.
- Navegación contextual entre entidades relacionadas, permitiendo acceder rápidamente desde clientes hacia sus contratos y desde los contratos hacia la información del padrón correspondiente.

7. **Canal automatizado de consultas para compradores vía WhatsApp:**

   El sistema integra un *bot* conversacional conectado en tiempo real a los datos del negocio, orientado a extender el servicio hacia el cliente final.

   En esta categoría se incluyen:

   - Verificación de identidad basada en el número telefónico registrado y confirmación del padrón asociado.
   - Menú de autoservicio que permite consultar los próximos vencimientos, el valor actualizado de la próxima cuota, la cantidad de cuotas pagas, pendientes y vencidas, el estado de cuenta detallado y otros datos relevantes del contrato.

3.3. Alcance del proyecto

El alcance del proyecto comprende la entrega de una plataforma web operativa orientada a usuarios internos, soportada por un *backend* y una base de datos relacional, que permite administrar la información del negocio y ejecutar el flujo completo de venta y cobranza asociado a padrones. El sistema establece su límite funcional estrictamente en la gestión operativa financiera, delegando las tareas

contables a actores externos que se alimentan de los reportes generados, tal como se describe en la sección 5.1.1 Características del Cliente.

La funcionalidad del núcleo abarca la gestión centralizada de las entidades principales del negocio (clientes, empresas, etapas y terrenos) y la administración del ciclo de vida de los contratos de venta, incluyendo sus reglas de cancelación. A nivel de procesamiento, se incluye la automatización de la lógica financiera: generación de cronogramas de cuotas, registro de pagos, cálculo dinámico de recargos por mora y deducción de comisiones de redes externas. Se incluye además la posibilidad de editar la información de los pagos ya registrados, recalculando importes, mora y saldos, y propagando efectos hacia pagos posteriores cuando aplica.

Para soportar los cálculos mencionados de manera reproducible, se incluye dentro del alcance la integración automatizada con el BCU para la obtención y persistencia de cotizaciones monetarias y tasas de vivienda, implementando mecanismos de contingencia (*fallback*) propios.

En términos de seguimiento y control interno, el proyecto abarca la generación de reportes operativos de la información gestionada exportables a formato Excel. Hacia el exterior de la organización, el alcance se extiende a la provisión de información de autoservicio para los compradores. Esto se implementará mediante la integración de un *bot* de WhatsApp conectado a la API oficial de Meta, el cual operará a través de un menú estructurado. Tanto el detalle exacto del formato y los indicadores a incluir en los reportes, como las opciones específicas de consulta que se ofrecerán dentro del menú del *bot* de WhatsApp, serán acordados y refinados en conjunto con el cliente durante las etapas de desarrollo y validación del proyecto.

A nivel de arquitectura y seguridad, el proyecto abarca el despliegue del sistema en un entorno productivo en la nube (AWS), condicionado al cumplimiento de un presupuesto de infraestructura operativo de USD 200 mensuales establecido por el cliente. Esta restricción moldea las decisiones descritas en la sección 7. Arquitectura y Diseño. El

alcance de la seguridad se delimita a un modelo de confianza cerrado, implementando autenticación delegada (OAuth 2.0) con un modelo de autorización binario (administrador y usuario estándar), sobre una lista de correos permitidos (*allowlist*) gestionada por los administradores.

Para asegurar la transición completa de la operativa del cliente hacia la nueva plataforma, se contempla la migración masiva y automatizada del historial operativo. Esto implica sistematizar y trasladar la totalidad de los contratos y pagos preexistentes desde las planillas de cálculo (Excel) actuales hacia la nueva base de datos del sistema.

3.3.1. Exclusiones explícitas del alcance

Queda fuera del alcance el desarrollo de una interfaz web pública para el autoservicio de los clientes del emprendimiento. Esta decisión se fundamenta en el análisis conjunto de la operativa, donde se concluyó que el perfil de los compradores presenta altas barreras tecnológicas para la adopción de una plataforma tradicional. Esta hipótesis se confirmó empíricamente al observar el fracaso previo de un portal desarrollado por la empresa en la plataforma Wix (ver sección 6.1.1.2. Técnicas). En consecuencia, se determinó que la alternativa más eficaz y de menor fricción era canalizar el autoservicio a través de WhatsApp, aprovechando un ecosistema que ya forma parte del flujo de comunicación actual con los usuarios, por lo que ya se encuentran completamente familiarizados con él.

Asimismo, queda excluida del alcance la gestión y el almacenamiento de documentos PDF como respaldo de los comprobantes de pago. Esta decisión se fundamenta en un análisis directo de la carga operativa asociada a dicha funcionalidad. Al inicio del proyecto, previo a la existencia de un canal de comunicación automatizado, guardar comprobantes en formato PDF habría implicado solicitar explícitamente el archivo a

cada comprador, depender de su envío, descargarlo manualmente desde WhatsApp o correo electrónico y finalmente cargarlo de forma individual en la plataforma.

Dado que la necesidad operativa real del cliente consiste únicamente en registrar un identificador alfanumérico que permita realizar controles cruzados con los comprobantes bancarios, ingresar manualmente dicha cadena de texto, tal como el equipo administrativo ya estaba habituado a hacer en las planillas heredadas, resultó ser la alternativa de menor fricción operativa. En consecuencia, la funcionalidad de registro de pagos (RF-06) fue diseñada para aceptar únicamente un identificador textual introducido de forma manual (ver sección 6.2. Funcionalidades).

La gestión de archivos adjuntos se considera una evolución del sistema, viable una vez implementado y estabilizado el canal automatizado de interacción con compradores mediante el *bot* de WhatsApp, lo que permitiría guardar y asociar los comprobantes de manera directa dentro del flujo de gestión de pagos de la plataforma. (ver 11.4. Próximos pasos). Es por esto que se incorporó la opción en el menú del bot con el siguiente aviso: “La funcionalidad de envío de aviso de pago (PDF/imagen) se encuentra en desarrollo y estará disponible próximamente”.

3.3.2. Evolución y ajuste del alcance

Como se mencionó en la sección 2.3 Desafíos del proyecto, el análisis de las planillas heredadas reveló un nivel de inconsistencia estructural y variabilidad en las reglas de negocio que hacía inviable una migración masiva automatizada de forma directa. Para concretarla, era necesario llevar adelante un proceso de revisión conjunta en el que el cliente analizara los registros históricos caso por caso, normalizara la información y definiera criterios de negocio que hasta ese momento no estaban completamente formalizados.

Al tomar dimensión del esfuerzo administrativo que esto implicaba para su operativa cotidiana, el cliente indicó explícitamente postergar la migración masiva para una etapa

posterior a la entrega del Trabajo Final. En consecuencia, el alcance del proyecto se redefinió para priorizar la operativa futura del negocio sobre la regularización del historial.

Cabe destacar que, desde la planificación inicial, ya se había previsto una estrategia de transición gradual para validar la correctitud del nuevo sistema en coexistencia con las planillas. Sin embargo, a partir de esta redefinición, dicha coexistencia adquirió un nuevo propósito: la plataforma pasó a constituirse como fuente de verdad para los contratos nuevos, mientras que la información histórica permaneció temporalmente en Excel.

Esta estrategia de transición tuvo un impacto directo sobre la adopción del canal de autoservicio para compradores (*bot* de WhatsApp). Dado que el sistema solo almacena a los contratos nuevos, habilitar el canal implicaría ofrecer el servicio a un segmento muy reducido de clientes. Por lo tanto, el cliente tomó la decisión estratégica de postergar la exposición masiva del *bot* hasta que se haya concretado la migración definitiva, garantizando así que, al momento de su lanzamiento, la herramienta pueda ser aprovechada por la totalidad de su cartera de compradores de manera equitativa.

Para compensar la reducción del esfuerzo técnico asociada a la postergación de la migración histórica y, al mismo tiempo, sostener la profundidad esperada para el Trabajo Final de Carrera, se incorporó un bloque de funcionalidades extendidas (ver sección 6.2.2. Funcionalidades extendidas (*feature flags*). Estas incluyeron un *chatbot* interno asistido por inteligencia artificial para consultar la base de datos mediante lenguaje natural, un sistema de notificaciones automáticas por correo electrónico para alertar atrasos en los pagos y un *dashboard* interactivo orientado al seguimiento visual de indicadores clave vinculados a la morosidad y la recaudación. Además, se optó por realizarlas bajo un esquema de *feature flags*, de modo de aislarlas del núcleo operativo solicitado por el cliente y no comprometer su estabilidad.

Finalmente, una vez redefinidas las fronteras de la solución, quedaron explícitamente identificados aquellos elementos que no formarían parte de esta fase del proyecto. Estos aspectos, con potencial evolutivo para el sistema, se retoman más adelante en la sección 11.4. Próximos pasos,

4. Estrategia de construcción de la solución

El objetivo de este capítulo es describir con detalle el proceso metodológico y técnico seguido para construir e implementar la solución, mostrando las decisiones de planificación, organización y ejecución que guiaron el desarrollo. La estrategia adoptada se basó en un enfoque iterativo e incremental, orientado a entregar valor de forma temprana y continua y a reducir riesgos técnicos y funcionales mediante validaciones periódicas con el cliente.

Para materializar este enfoque, el trabajo se estructuró a partir de un *product backlog* formalizado en epics y user stories, lo que permitió descomponer el alcance en unidades abordables, priorizables y verificables (ver sección 6.1.2 Especificación). La planificación, priorización y seguimiento de este backlog se gestionaron mediante las herramientas detalladas en la sección 8.1 Herramientas de gestión, utilizando Jira [4] en una primera etapa y posteriormente Linear [5]. El control de versiones, la coordinación del trabajo y la integración continua se centralizaron en GitHub.

Desde la perspectiva de calidad, se incorporaron validaciones de negocio en el *backend* y pruebas automatizadas desarrolladas con pytest [6]. Estas pruebas se ejecutan automáticamente como parte del *pipeline* de integración continua cada vez que se abre o actualiza un PR, y su aprobación constituye una condición necesaria para integrar cambios al repositorio principal. Adicionalmente, se estableció un umbral mínimo de cobertura de código en el *backend* igual o superior al 80%, utilizado como criterio objetivo para asegurar un nivel base de verificación automática en cada incremento.

En paralelo, el proceso de despliegue se automatizó mediante GitHub Actions y la gestión de infraestructura como código con Terraform [7], manteniendo entornos de ejecución separados para desarrollo y producción. Los despliegues se disparan automáticamente a partir de la integración de cambios en las ramas principales del repositorio (develop o main), lo que redujo el riesgo de errores por operaciones

manuales y acortó el tiempo entre la integración del código y su disponibilidad en el entorno objetivo correspondiente.

En cuanto al plan de entregas, la planificación inicial acordada con el cliente contemplaba dividir el proyecto en tres *releases*. Luego de completar el primer *release* y contar con mayor visibilidad sobre el alcance restante, se realizó una revisión conjunta con el cliente para evaluar el progreso y los riesgos asociados a las funcionalidades pendientes. Durante este análisis se identificó el riesgo de postergar para una etapa final componentes con dependencias externas y mayor incertidumbre de integración, particularmente el *bot* de WhatsApp. Abordar estos componentes complejos en un tercer *release* podía comprometer la estabilidad del sistema en etapas tardías del proyecto.

Como resultado de la revisión, se consensuó reordenar prioridades y consolidar el plan de entregas en dos *releases*:

- ***Release* 1:** consolidación del núcleo operativo con las funcionalidades críticas para operar (gestión de ventas y contratos, generación de cuotas, procesamiento de pagos y controles básicos).
- ***Release* 2:** incorporación de funcionalidades incrementales de alto valor y de mayor incertidumbre técnica (módulo de reportes, exportaciones, mejoras de usabilidad, autenticación robusta e integración del *bot* de WhatsApp).

Esta reorganización permitió mitigar riesgos de forma temprana y asegurar que las integraciones externas se desarrollaran sobre una base funcional ya estabilizada.

En paralelo al despliegue del sistema, se definió una estrategia de adopción orientada a mitigar el impacto operativo en el cliente. Inicialmente se preveía que DATIUM operara en paralelo con las planillas de cálculo heredadas, de modo que los cálculos financieros generados por la plataforma pudieran auditarse empíricamente antes de abandonar el sistema anterior. Sin embargo, ante la decisión del cliente de postergar la migración histórica masiva debido a las inconsistencias detectadas en sus registros,

como se explica en la sección anterior, esta coexistencia se redefinió. La plataforma pasó a operar como única fuente de verdad exclusivamente para los nuevos compromisos de compraventa.

Como consecuencia, el esfuerzo técnico reservado para la migración de datos históricos se reorientó hacia la construcción de funcionalidades analíticas y de automatización extendidas, controladas mediante *feature flags*, lo que permitió cumplir con la ambición académica del Trabajo Final sin comprometer el alcance productivo acordado.

Una vez consolidado y estabilizado el núcleo operativo en el primer *release*, el segundo se enfocó en incorporar mejoras derivadas del uso real del sistema e integrar componentes con dependencias externas. Entre estas evoluciones se incluyeron herramientas de reporte y exportación de información para facilitar tareas de auditoría y conciliación contable, mejoras de usabilidad orientadas a reducir fricciones en la operativa administrativa, el fortalecimiento de los mecanismos de autenticación y control de acceso, y la integración del *bot* de WhatsApp como canal automatizado de consulta para los compradores. Dado que esta última característica posee dependencias de terceros (Meta), su desarrollo se abordó sobre un núcleo ya maduro y estabilizado.

En conjunto, la estrategia de construcción del sistema se orientó a tres prioridades fundamentales:

1. Asegurar la consistencia y reproducibilidad de los procesos críticos del negocio (gestión de ventas y contratos, cuotas, pagos y moras).
2. Habilitar un despliegue productivo temprano para validar la solución con usuarios reales.
3. Evolucionar la solución de forma controlada, incorporando mejoras sobre una base estable sin comprometer la confiabilidad del núcleo operativo.

4.1. Cronología de las etapas

La ejecución del proyecto se estructuró en tres etapas principales: investigación, prototipado y validaciones técnicas y desarrollo, tal como se resume en la Figura 2. Esta organización permitió avanzar desde la comprensión del dominio y la reducción de incertidumbre técnica hacia una construcción incremental con validación continua.

Etapa de investigación: En esta fase el esfuerzo se enfocó en comprender los procesos operativos del cliente e identificar actores, entidades principales y reglas de negocio. Los problemas estructurales y dolores operativos relevados se describen en la sección 2.1 Contexto. En paralelo, se realizó una investigación técnica para evaluar alternativas y definir la base tecnológica del sistema, seleccionando FastAPI (Python) para el *backend* y React con TypeScript para el *frontend*, junto con el motor de base de datos relacional. A partir de este relevamiento funcional y técnico se definió el alcance inicial, se modelaron los flujos críticos del sistema y se estructuró el *backlog* en *epics* y *user stories*, estableciendo una primera planificación de entregas. Las justificaciones detalladas de la selección tecnológica se desarrollan en el capítulo correspondiente de decisiones de arquitectura (ver sección 7.3 Architectural decision records (ADRs)) y tecnologías (ver sección 7.4 Tecnologías).

Etapa de prototipado y validaciones técnicas: Con el alcance inicial delineado, se realizaron instancias de prototipado liviano y validaciones técnicas para confirmar decisiones que podían impactar la viabilidad o complejidad del sistema (ver sección 6.1.3 Validación). Estas validaciones permitieron reducir riesgos asociados a la arquitectura inicial, el esquema de persistencia, las integraciones necesarias y el enfoque de despliegue, verificando supuestos clave antes de avanzar con la implementación del producto. El objetivo fue minimizar retrabajo y evitar que decisiones críticas se descubrieran en etapas tardías de construcción.

Etapa de desarrollo: La construcción del sistema se llevó a cabo de manera iterativa en *sprints* de dos semanas, incorporando refinamiento continuo del *backlog* y

revisiones periódicas con el cliente. El *Release* 1 concentró el *core* del sistema y culminó con su despliegue como ambiente productivo en AWS, habilitando el uso real por parte de usuarios administrativos. A partir de ese punto, el cliente comenzó a recibir valor de forma sostenida mediante despliegues frecuentes soportados por automatización de despliegue. El *Release* 2 incorporó funcionalidades incrementales (autenticación, reportes y exportaciones) y mejoras surgidas del uso real del sistema y del *feedback* recibido, manteniendo la estabilidad del producto en producción.

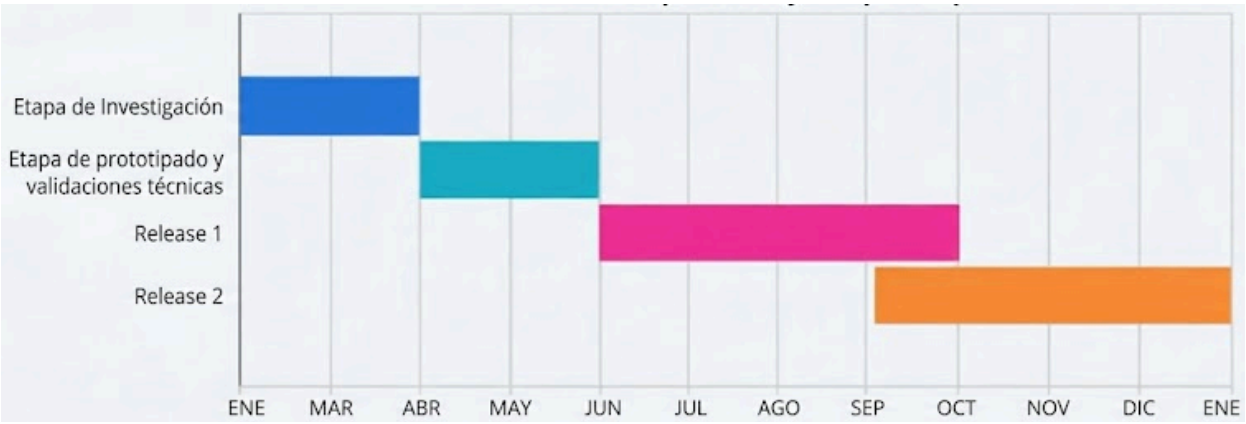


Figura 2. Proyecto por etapas

5. Marco metodológico

5.1. Características del proyecto

5.1.1. Características del cliente

El cliente es un emprendimiento inmobiliario dedicado a la comercialización de terrenos mediante planes de financiación a largo plazo. A nivel organizacional, la empresa no cuenta con un organigrama formal ni con una división estricta por departamentos. La gestión operativa recae sobre un núcleo administrativo y de dirección que asume de forma centralizada los procesos de venta, administración de contratos, cobranza y atención al comprador (ver sección 2.2 Interesados).

Esta falta de formalización organizacional implica que la mayoría de los procesos y reglas financieras no se encontraban documentados, sino que dependían del conocimiento de los operadores y del uso de planillas de cálculo dispersas. Esta característica del cliente justifica la adopción de un marco metodológico iterativo e incremental, ya que el proceso de desarrollo exigió descubrir, validar y formalizar las reglas del negocio de manera progresiva.

Debido a la ausencia de una estructura departamental formal, la operativa del negocio se organiza de manera natural en torno a las fronteras del sistema. En este contexto, el flujo de trabajo de la empresa queda definido por el alcance de la plataforma, estableciendo límites claros entre la gestión operativa interna y las áreas de soporte periférico que se pueden apreciar en la Figura 3:

- Frontera administrativa (*Core*): Constituye el núcleo de la operativa diaria del negocio y es gestionada exclusivamente por el equipo interno de administración y dirección, bajo estrictos criterios de confidencialidad. Esta frontera engloba el flujo de trabajo crítico de la empresa, identificado durante el diseño del sistema como el *Golden Path* [11] del administrador (descrito en la sección 6.1 Proceso).

Sus principales tareas incluyen la estructuración del predio, el alta de compradores, la formalización de contratos, el control de cuotas y el registro de pagos.

La digitalización de esta frontera permite que el sistema automatice la lógica financiera asociada, incluyendo el cálculo de moras e intereses, la deducción de comisiones retenidas por redes de cobranza externas (Abitab / Redpagos) y las conversiones multidivisa basadas en cotizaciones oficiales del BCU. De este modo, la plataforma centraliza estos procesos y se consolida como la única fuente de verdad para las operaciones activas, reduciendo el riesgo operativo y eliminando la dependencia de planillas de cálculo.

- Frontera de atención al cliente: En la operativa original, gran parte del tiempo administrativo se destinaba a responder consultas repetitivas de los compradores, principalmente relacionadas con saldos pendientes, fechas de vencimiento o estado de pagos. Para optimizar este proceso, la plataforma incorpora un canal automatizado de autoservicio que permite resolver este tipo de consultas de forma inmediata. De esta manera, la atención humana queda reservada para situaciones que requieren intervención directa, como lo pueden ser reclamos, aclaraciones específicas o procesos de renegociación.

- Frontera contable y de auditoría: La organización no cuenta con un departamento contable interno. Su operativa finaliza con la generación y exportación estructurada de reportes operativos (ventas, ingresos, cobranzas, deudores, entre otros). A partir de esa información, la conciliación bancaria, la liquidación de impuestos y el resto de las tareas contables se realizan fuera de los procesos diarios de la empresa, siendo asumidas por un contador externo que utiliza dichos reportes como insumo principal.

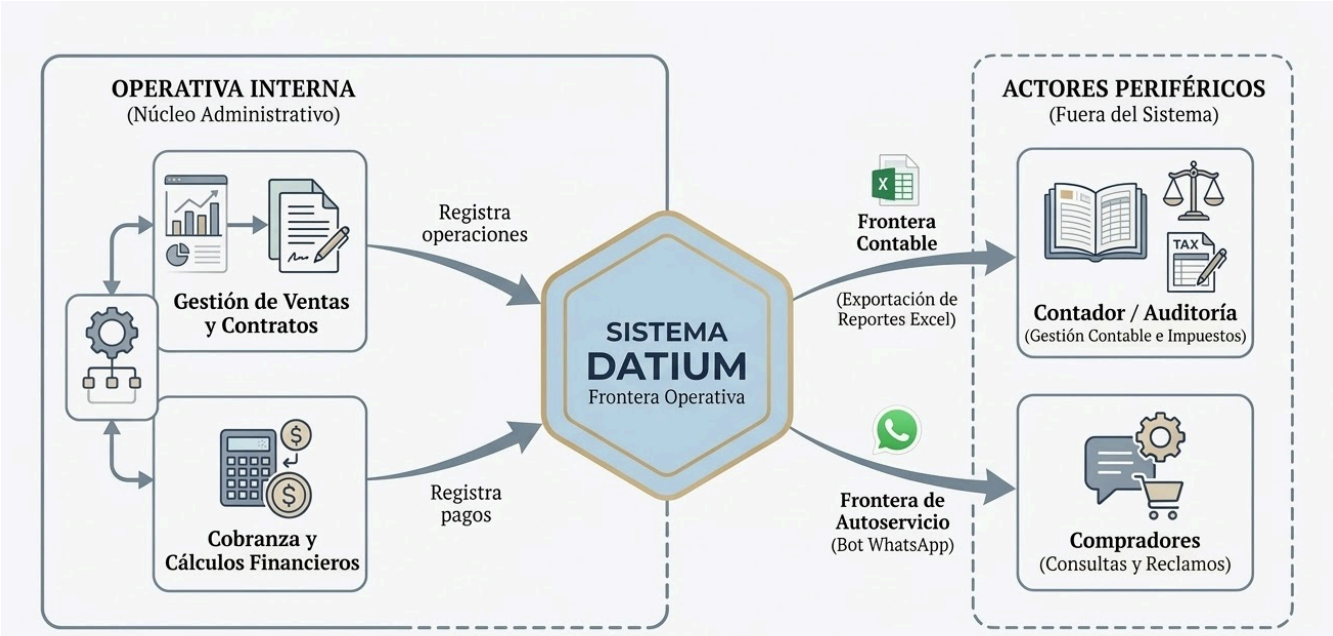


Figura 3. Diagrama de contexto de DATIUM

5.1.2. Características del producto

El producto propuesto se describe en detalle en la sección 3.1 Solución propuesta. Consiste en un sistema web orientado a centralizar y automatizar los procesos operativos del cliente, permitiendo la gestión integrada de la información vinculada a clientes, padrones, contratos, cuotas y pagos. El sistema reemplaza el uso de planillas de cálculo y procesos manuales actualmente utilizados, reduciendo la probabilidad de errores operativos y mejorando la consistencia de la información.

5.1.2.1. Enfoque date-driven

Para este proyecto, considerando la triple restricción [8] (alcance, tiempo, costo), la variable tiempo fue fija debido a la restricción académica. Por lo tanto, se ha optado por el enfoque *date-driven*. Esta decisión se fundamenta en que los proyectos *date-driven* son aquellos que deben ser entregados en una fecha específica, mientras que el conjunto de características es negociable. Esta característica nos permitió conocer de

antemano el número exacto de *sprints* disponibles para la planificación y ejecución del proyecto.

5.1.3. Características del equipo

El equipo de trabajo estuvo conformado por Mateo González, Matías Wildbaum, Federica Cabrera y Ana Gutman, todos estudiantes de la carrera Ingeniería en Sistemas.

Matías contaba con experiencia previa en desarrollo con Python en el ámbito laboral, lo que facilitó la inicialización del proyecto y la estructuración del *backend*. También aportó experiencia en aspectos de usabilidad y experiencia de usuario, contribuyendo a la definición de criterios visuales y mejoras de interacción.

Federica tenía experiencia previa como desarrolladora *frontend*, lo que permitió definir desde etapas tempranas la arquitectura del repositorio de *frontend*, la selección de frameworks y la implementación de una estructura basada en componentes reutilizables (organizados bajo un esquema de *atoms*, *molecules* y *components*). También contaba con experiencia en mecanismos de autenticación, integración con APIs externas y generación de reportes en formato Excel.

Mateo aportó conocimientos en infraestructura en la nube y automatización, adquiridos en la materia electiva Infraestructura en la nube, incluyendo el uso de Terraform. Esta formación permitió incorporar prácticas de infraestructura como código, integración continua y despliegue automático.

Ana se destacó por un fuerte enfoque en desarrollo *backend*, lógica de negocio, procesamiento de datos reales y aseguramiento de calidad del software. Su perfil permitió abordar con rigor técnico el modelado de datos contractuales y la validación de cálculos financieros. Aplicó conocimientos adquiridos en la electiva AI Enablement para

liderar el diseño e implementación de componentes de procesamiento de lenguaje natural.

5.2. Metodología de trabajo

La elección del ciclo de vida y del marco de gestión se realizó considerando las características del producto, del cliente y del contexto del proyecto.

5.2.1. Scrum

La selección del marco de gestión se realizó considerando las características concretas del proyecto al inicio del trabajo. El cliente no contaba con procesos completamente formalizados ni con una especificación detallada y cerrada de requerimientos. Además, la solución implicaba la integración con fuentes externas de datos financieros oficiales y el desarrollo de funcionalidades cuya validación debía realizarse al mismo tiempo con el cliente.

Dado este contexto, se optó por un enfoque iterativo e incremental que permitiera entregar valor de manera progresiva, reducir riesgos asociados a la incertidumbre funcional y ajustar prioridades a medida que se obtenía mayor comprensión del negocio. En consecuencia, se decidió utilizar Scrum como marco de gestión del proyecto.

El desarrollo se organizó en 15 sprints de dos semanas de duración cada uno. En cada sprint se definió un *Sprint Goal* y se trabajó en la construcción de un incremento funcional del sistema, permitiendo validar avances de forma periódica y mantener una planificación adaptable.

Si bien se adoptaron los principios fundamentales de Scrum, se realizaron adaptaciones para ajustarlo al contexto del equipo y del cliente. En particular, las reuniones diarias no se realizaron con frecuencia estrictamente diaria debido a limitaciones de disponibilidad y diferencias de zona horaria entre los integrantes. En su

lugar, se establecieron reuniones sincrónicas dos veces por semana, complementadas con actualizaciones asincrónicas diarias mediante herramientas de mensajería, donde cada integrante reportaba su estado y posibles bloqueos. Esta adaptación permitió mantener la transparencia y la coordinación del equipo sin afectar la continuidad del trabajo.

Por otro lado, las reuniones de revisión con el cliente se realizaron cada dos *sprints*, en lugar de al finalizar cada *sprint*. Dado el contacto frecuente y la comunicación continua con el cliente durante el desarrollo, esta frecuencia resultó suficiente para validar los incrementos y ajustar prioridades cuando fue necesario.

En retrospectiva, el proyecto puede caracterizarse como perteneciente al dominio complejo según el marco Cynefin [9] (ver anexo 1 - Matriz de Cynefin), dado que la comprensión del problema y de las soluciones adecuadas emergió progresivamente a través del desarrollo y la validación continua. En este tipo de contextos, los enfoques iterativos basados en inspección y adaptación resultan adecuados, lo cual refuerza la coherencia de la elección realizada.

5.2.2. Ciclo de vida

El proyecto adoptó un ciclo de vida iterativo e incremental, estructurado en *sprints* de dos semanas, en concordancia con el marco de gestión descrito en la sección anterior. Bajo este enfoque, el sistema evolucionó progresivamente mediante la incorporación sucesiva de funcionalidades, priorizadas en función del valor para el cliente y de la reducción de riesgos técnicos.

Previo al inicio del primer *sprint* se desarrolló una etapa preliminar de definición, en la cual el equipo realizó reuniones internas para establecer decisiones arquitectónicas y tecnológicas, dado que el cliente no contaba con restricciones específicas en este aspecto. Además, se mantuvo una instancia inicial con el cliente destinada a comprender en profundidad el modelo de negocio, los procesos actuales y las

principales problemáticas operativas. Esta fase permitió conformar un backlog inicial y reducir la incertidumbre técnica antes de comenzar el desarrollo iterativo.

Durante los primeros *sprints* el foco estuvo puesto en la construcción del núcleo funcional del sistema, incluyendo la gestión de clientes, padrones, contratos y cuotas. En esta etapa el producto no se encontraba aún desplegado en un entorno accesible para el cliente, ya que se priorizó la consolidación de la base estructural antes de su liberación.

El proyecto contempló un plan de *releases* compuesto por dos grandes hitos. El *Release* 1 estuvo orientado a la entrega del núcleo operativo completo del sistema, permitiendo al cliente comenzar a utilizar la solución para la gestión centralizada de su negocio. A partir de esta liberación se adoptó una dinámica de despliegues con frecuencia semanal, incorporando mejoras y correcciones de manera progresiva.

El *Release* 2 incluyó la incorporación de funcionalidades avanzadas y de mayor complejidad técnica, tales como componentes analíticos y automatizaciones del sistema. En esta fase también se realizaron mejoras y ajustes sobre funcionalidades previamente implementadas, consolidando la estabilidad y la experiencia de uso.

En conjunto, el ciclo de vida adoptado permitió estructurar el desarrollo en etapas claramente diferenciadas: una fase inicial de definición técnica, una fase de construcción del núcleo funcional, y una fase de expansión y optimización. Este enfoque contribuyó a mitigar riesgos, ordenar prioridades y garantizar la entrega progresiva de valor dentro del plazo académico establecido.

5.3. Roles del equipo

Los roles de Scrum fueron los siguientes: el rol de *Scrum Master* fue asumido por Mateo; el rol de *Product Owner* fue asumido por Matías; y el rol de equipo de desarrollo fue asumido por todos los integrantes del equipo.

Más allá de la asignación formal de roles dentro del marco de trabajo adoptado, el equipo distribuyó responsabilidades técnicas y funcionales de acuerdo con las fortalezas, experiencia previa e intereses de cada integrante, procurando además un equilibrio en la carga de trabajo.

Matías asumió responsabilidades vinculadas a la definición funcional del sistema y al desarrollo de componentes orientados a la interacción con el usuario final. Fue responsable de la redacción y mantenimiento de los tickets, y la comunicación continua con el cliente para relevar necesidades y validar funcionalidades. Desarrolló de punta a punta el *bot* de WhatsApp, incluyendo su integración con la lógica de negocio y los flujos de consulta. También participó activamente en decisiones de usabilidad, colaborando en mejoras visuales y en la implementación de los modos claro y oscuro del sistema.

Federica asumió el liderazgo técnico del *frontend*, coordinando las decisiones de diseño e implementación y asegurando la coherencia de la arquitectura definida. Estuvo a cargo del desarrollo del *dashboard*, la generación y exportación de reportes, y la integración con servicios externos para el envío de correos electrónicos. Además, gestionó la configuración del dominio y el despliegue de la landing page de la aplicación.

Mateo concentró sus responsabilidades en la infraestructura y automatización del proyecto. Implementó la infraestructura como código, configuró los entornos necesarios para el despliegue y desarrolló *pipelines* de CI para la ejecución automática de pruebas. Adicionalmente, implementó *pipelines* de despliegue automático y el uso de *feature flags* para habilitar o deshabilitar funcionalidades de forma controlada. Hacia etapas más avanzadas del proyecto, también participó en mejoras de usabilidad del *frontend*, particularmente en la pantalla principal de gestión de contratos.

Ana asumió responsabilidades centrales en el desarrollo *backend*, la lógica de negocio y el procesamiento de datos reales. Fue responsable del análisis y modelado de las

planillas de Excel proporcionadas por el cliente, identificando su estructura y las transformaciones necesarias para su incorporación al sistema. También implementó la obtención de cotizaciones históricas requeridas para los cálculos financieros mediante integración con servicios oficiales, y lideró la validación y corrección de cálculos financieros, incluyendo la detección de inconsistencias numéricas y diferencias de redondeo. Para ello, desarrolló scripts y escenarios reproducibles que permitieron verificar la consistencia de los resultados. Se encargó de agregar pruebas unitarias basadas en casos reales, y diseñó un componente de procesamiento de lenguaje natural que traduce consultas en lenguaje natural a consultas estructuradas sobre la base de datos.

Si bien todos los integrantes tenían ciertas áreas del proyecto a su cargo, fue responsabilidad de todos evaluar el trabajo hecho por sus compañeros, ya que todo el código fue revisado por una persona que no fuera su autor. Así, el equipo entero tenía conocimiento sobre el proyecto en general, y no solamente en aquellas zonas donde cada integrante concentró sus esfuerzos.

6. Ingeniería de requerimientos

6.1. Proceso

El proceso de ingeniería de requerimientos se llevó a cabo de forma iterativa e incremental, alineado con la metodología Scrum adoptada por el equipo. Se estructuró en cuatro etapas, relevamiento, especificación, validación y verificación, las cuales no se ejecutaron de forma estrictamente secuencial, sino que se retroalimentaron a lo largo de los *sprints* del proyecto.

Para seleccionar y organizar las técnicas de relevamiento, el equipo se basó en el framework propuesto por Tsumaki y Tamai [10], que clasifica las técnicas de elicitación de requerimientos según dos ejes ortogonales: estático–dinámico y cerrado–abierto. Este framework se utilizó tanto para guiar la selección inicial de técnicas, buscando cobertura deliberada de los cuatro cuadrantes, como para verificar a posteriori que el conjunto de técnicas efectivamente utilizado logrará un balance adecuado entre necesidades explícitas del cliente, necesidades latentes del dominio e ideas del equipo. La sección 7.1.1 Diagrama de bloques del sistema describe cada técnica junto con su clasificación.

Adicionalmente, se utilizó el concepto de *Golden Path* [11] como herramienta para identificar y priorizar los flujos principales del sistema. El *Golden Path* consiste en definir el camino ideal que un usuario recorre para completar sus tareas clave, permitiendo priorizar las funcionalidades que mayor valor aportan y enfocar tanto el relevamiento como el diseño en lo esencial. Dado que el sistema tiene dos perfiles de usuario claramente diferenciados, se definieron dos *Golden Paths* que se ilustran en la Figura 4:

- **Usuario administrador** (plataforma web): alta de empresa/etapa → alta de terreno → alta de comprador → alta de contrato → generación automática de

cuotas → registro de pago → cálculo de mora → generación de reportes → exportación para auditoría.

- **Usuario comprador** (WhatsApp): verificación de identidad por número de celular → confirmación del padrón asociado → menú principal con las opciones: aviso de pago, próximo vencimiento, valor de próxima cuota, cuotas pagas, cuotas pendientes, cuotas vencidas, estado de cuenta y salir.

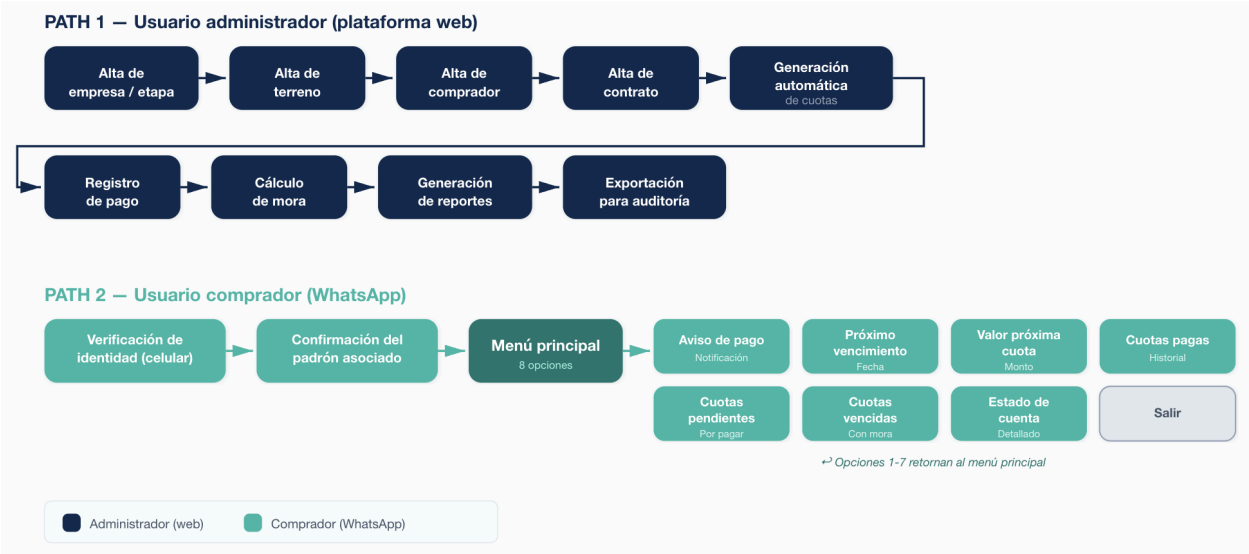


Figura 4. Golden Path

Estos *Golden Paths* guiaron la priorización de los requerimientos funcionales, el orden de desarrollo en los *sprints*, y el diseño de la interfaz del sistema.

6.1.1. Relevamiento

6.1.1.1. Criterios de Clasificación de técnicas

Antes de describir cada técnica, se presentan los dos criterios de clasificación propuestos por Tsumaki y Tamai [10] que el equipo utilizó para asegurar cobertura metodológica.

Criterio 1: Estático vs. Dinámico: Las técnicas estáticas son aquellas que no permiten cambios ni interacciones en el momento de su aplicación: la información se recoge de forma fija, sin modificar el proceso (por ejemplo, analizar un documento existente). Las técnicas dinámicas implican interacción continua con los participantes, lo que permite ajustar el enfoque en función de las respuestas obtenidas o del contexto emergente (por ejemplo, una entrevista donde se profundiza en ciertos temas según lo que surge en la conversación).

Criterio 2: Cerrado vs. Abierto: Las técnicas cerradas se utilizan para investigar aspectos que ya están bien definidos o delimitados (por ejemplo, validar con el cliente una pantalla específica del prototipo). Las técnicas abiertas buscan descubrir nuevos aspectos o problemas que no estaban definidos previamente (por ejemplo, una sesión de brainstorming para explorar alternativas de solución).

El equipo buscó deliberadamente utilizar técnicas en los cuatro cuadrantes resultantes de combinar estos ejes, de modo que el relevamiento no se limitara a seguir únicamente lo planteado por el cliente, sino que también capturará necesidades latentes del dominio y oportunidades de diseño identificadas por el equipo. Estos cuadrantes se visualizan en la Figura 5.

La siguiente tabla resume la clasificación de cada técnica utilizada:

Técnica	**Estático/Dinámico**	**Cerrado/Abierto**	**Etapa del proyecto**
Entrevistas iniciales con el cliente	Dinámico	Abierto	Comprensión del negocio
Análisis de documentación (planillas Excel)	Estático	Cerrado	Comprensión del negocio

Observación de la operativa	Dinámico	Abierto	Comprensión del negocio
Brainstorming interno (Miro)	Dinámico	Abierto	Análisis y diseño
Análisis de soluciones existentes (portal Wix, competidores)	Estático	Abierto	Análisis y diseño
Prototipos de baja y media fidelidad	Dinámico	Abierto	Validación temprana
Prototipos de alta fidelidad	Dinámico	Cerrado	Validación de diseño
Entrevistas de validación funcional	Dinámico	Cerrado	Continua

Tabla 1: Clasificación de técnicas

Como se observa, el equipo logró cobertura en los cuatro cuadrantes: técnicas estáticas-cerradas (análisis de documentación) para comprender las reglas de negocio existentes, estáticas-abiertas (análisis de competidores) para descubrir buenas prácticas externas, dinámicas-abiertas (entrevistas iniciales, brainstorming, prototipos exploratorios) para descubrir necesidades latentes, y dinámicas-cerradas (prototipos de alta fidelidad, entrevistas de validación) para confirmar decisiones concretas.

Figura 5. Clasificación de técnicas

6.1.1.2. Técnicas

El relevamiento se organizó en etapas progresivas. A continuación se describe cada técnica con su justificación, implementación y resultados.

Etapa 1 – Comprensión del negocio

Entrevistas iniciales con el cliente (Dinámico – Abierto)

Figura 6. Reunión inicial con el cliente

El equipo se reunió con el equipo administrativo del emprendimiento inmobiliario (como se muestra en la Figura 6) para comprender el negocio, su funcionamiento y sus principales dolores. Estas entrevistas fueron de carácter semiestructurado [12]: se partió de un guión base, pero se permitió la exploración libre de temas emergentes.

Durante estas instancias se identificaron las problemáticas descritas en la sección 2.1 Contexto: la dependencia de planillas de Excel, el cálculo manual de cuotas e intereses con reglas variables por contrato, la dificultad para obtener una visión consolidada de la cobranza, y la comunicación fragmentada con los compradores. Las entrevistas también permitieron capturar las fricciones cotidianas y los *workarounds* que los

usuarios habían desarrollado para compensar las limitaciones de las herramientas existentes.

Análisis de documentación (Estático – Cerrado)

En paralelo, el equipo analizó las planillas de Excel proporcionadas por el cliente. Cada planilla correspondía a un terreno vendido (identificado por número de padrón), agrupadas en archivos según las etapas del predio. En cada hoja se registraban los datos del contrato y el detalle de pagos mensuales con sus cálculos financieros (ver anexo 2 - Planillas de Excel Proporcionadas).

Este análisis permitió comprender las reglas de negocio subyacentes: la variabilidad de reglas según la moneda del contrato (UI, UR o USD), la aplicación condicional de IVA sobre la mora, la extensión de vencimiento hasta fin de mes para pagos puntuales, las comisiones de redes de cobranza (Abitab / Redpagos), el manejo de excedentes y faltantes entre pagos consecutivos, y la obtención manual de cotizaciones y tasas de vivienda del BCU. También se identificaron inconsistencias: no todas las planillas seguían las mismas reglas, y la documentación no cubría todos los casos reales a los que el cliente se enfrentaba.

Observación de la operativa (Dinámico – Abierto)

Durante las entrevistas (un ejemplo de las cuales se ve en la Figura 7), el equipo observó cómo los usuarios realizaban sus tareas cotidianas: la carga manual de pagos a partir de comprobantes recibidos por WhatsApp y correo electrónico, la consulta de cotizaciones en el sitio del BCU, y la consolidación manual de datos entre múltiples planillas para responder preguntas del negocio. Esta observación fue clave para identificar puntos de fricción que el cliente no articulaba como requerimientos pero que representaban oportunidades claras de mejora.

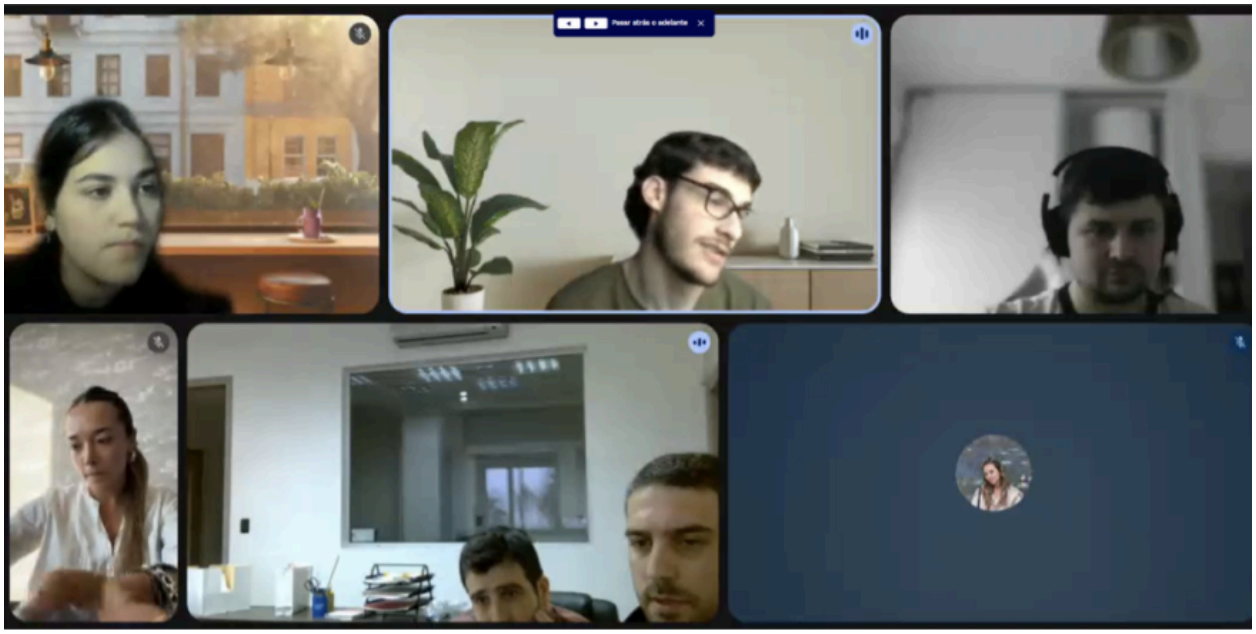


Figura 7. Entrevistas con cliente

Etapa 2 – Análisis y diseño

Brainstorming interno (Dinámico – Abierto)

Con la información de la primera etapa, el equipo realizó una sesión de brainstorming interna utilizando Miro, la cual se puede observar en la Figura 8. Se analizó el problema desde perspectivas técnica y funcional, discutiendo supuestos, identificando preguntas pendientes y explorando alternativas de diseño. Los temas discutidos incluyeron: la viabilidad de interfaces separadas para administradores y compradores, la conveniencia de un *chatbot* de WhatsApp frente a un portal web, la estrategia de automatización de cálculos financieros, y la definición de los perfiles de usuario con sus flujos principales (*Golden Paths*). Esta etapa generó una primera propuesta de solución que fue presentada al cliente para su validación.

Figura 8. Tablero Miro Brainstorming

Análisis de soluciones existentes (Estático – Abierto)

Se realizó un análisis en dos direcciones. Por un lado, se evaluó el portal de clientes que el propio cliente había desarrollado previamente en Wix (Figura 9), cuyo objetivo era centralizar información básica para los compradores.

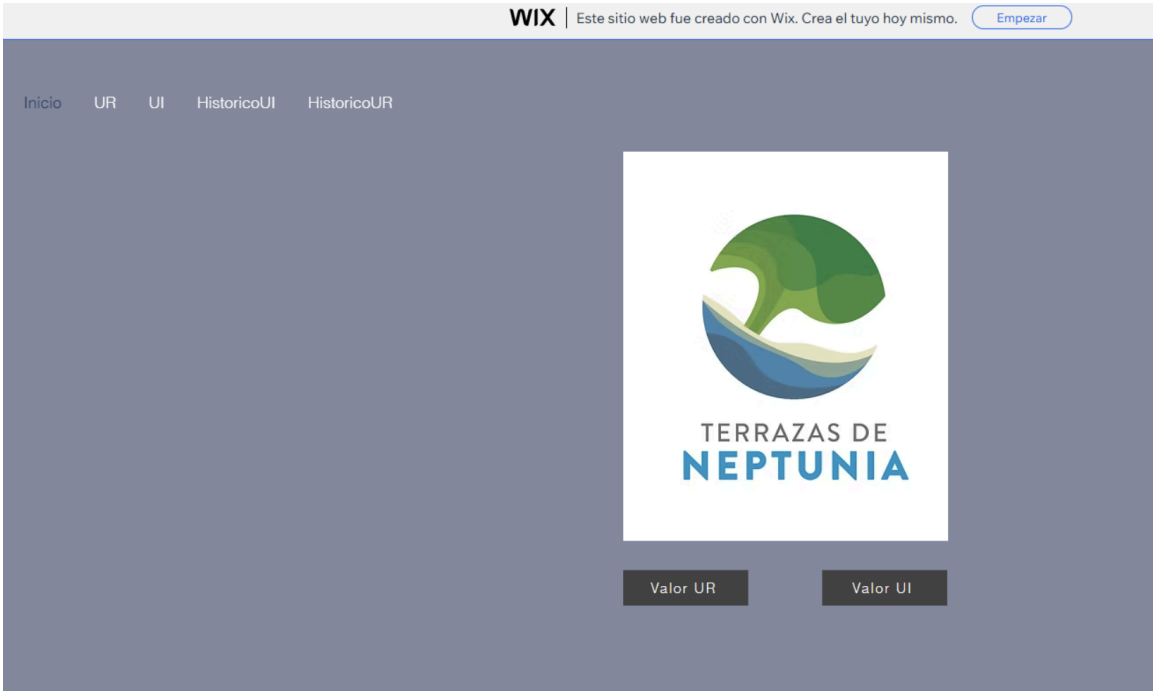


Figura 9. Sitio web Wix

El portal ofrecía funcionalidad limitada: permitía consultar los valores actuales de UR y UI y acceder al histórico de estas unidades, pero no brindaba información personalizada sobre el estado de cuenta de cada comprador. Además, la web presentaba tiempos de carga elevados que generaban fricción en su uso.

Al evaluar su adopción, el equipo detectó que la solución no estaba integrada con los datos reales del negocio, requería actualizaciones manuales y tenía baja utilización. En la práctica, los compradores continuaban realizando consultas por WhatsApp y correo electrónico. Este hallazgo resultó clave para comprender que el problema no era la ausencia de un canal digital, sino la falta de integración con la operativa real del negocio y la inexistencia de información personalizada para cada cliente. A partir de este diagnóstico se redefinieron los requerimientos de interacción, priorizando la incorporación de un *bot* de WhatsApp conectado directamente al *backend* del sistema.

Por otro lado, se realizó un análisis exploratorio de productos disponibles en el mercado que abordan la gestión de ventas a plazo y cobranzas en el ámbito inmobiliario. Se evaluaron soluciones pertenecientes a distintas categorías:

- Módulos de ERP generalistas con funcionalidad de cuotas (Odoo Installment Management).
- Software especializado en property management (AppFolio, Buildium).
- Suites ERP regionales (TOTVS).
- Plataformas de gestión contractual (Webdox).
- Alternativas *low-code* combinadas con integradores (Airtable + Zapier).

El principal hallazgo del análisis fue que ninguna de las soluciones evaluadas ofrecía soporte nativo para para la UI y la UR ni para las reglas de reajuste y cálculo de mora específicas del dominio. En todos los casos habría sido necesario realizar

personalizaciones significativas, con el costo de implementación y la dependencia tecnológica que ello implica.

Asimismo, la integración de un canal de autoservicio a través de WhatsApp conectado directamente con los datos de cobranza no estaba presente en las soluciones analizadas.

Cabe destacar que, a pesar de estas limitaciones, el análisis permitió identificar buenas prácticas en términos de navegación, uso de filtros, estructura de *dashboards* y presentación de reportes financieros, las cuales fueron incorporadas posteriormente al diseño de la plataforma.

Etapa 3 – Validación temprana

Prototipos de baja y media fidelidad (Dinámico – Abierto)

El equipo desarrolló prototipos de baja y media fidelidad para validar tempranamente las ideas de solución con el cliente desde una perspectiva de usabilidad (Ver anexo 3 - Prototipos baja fidelidad). El foco estuvo en evaluar la estructura de las pantallas, el orden y la jerarquía de la información, la disposición de los elementos en pantalla, y la navegación entre secciones. Esto permitió detectar problemas de comprensión y de flujo antes de avanzar con definiciones visuales más cerradas, reduciendo el riesgo de invertir esfuerzo de diseño en una estructura que no resultara clara o intuitiva para el usuario.

Etapa 4 – Validación de diseño

Prototipos de alta fidelidad (Dinámico – Cerrado)

Se avanzó con prototipos de alta fidelidad presentados al cliente como primer acercamiento visual a la solución final. Estos prototipos permitieron validar la disposición de la información en pantalla, criterios de usabilidad, jerarquía visual y

claridad de los flujos principales: gestión de clientes, terrenos, contratos y pagos (Ver anexo 4 - Prototipos alta fidelidad).

Dado que el sistema iba a ser utilizado de forma intensiva en la operativa diaria, la experiencia de usuario fue considerada un aspecto clave desde el inicio. Se realizaron varias instancias de iteración con el cliente incorporando *feedback* antes de avanzar a la etapa de desarrollo. Este proceso fue donde se aplicaron por primera vez las heurísticas de Nielsen [13] como criterio de evaluación del diseño (ver sección 7.2.2 Usabilidad (RNF-02)).

Entrevistas de validación (Dinámico – Cerrado)

Adicionalmente, de forma transversal a todo el proyecto, se realizaron instancias periódicas de validación con el cliente utilizando los avances del sistema como base para la revisión (Dinámico – Cerrado). Estas instancias se centraron en decisiones funcionales: reglas de cálculo, manejo de casos borde, comportamiento ante pagos parciales y priorización de funcionalidades. Como se puede observar en la Figura 10.

El cliente se mostró altamente disponible y proporcionó *feedback* rápido al comenzar a testear el sistema, lo que permitió ajustar requerimientos de forma ágil. Esta práctica no constituyó una etapa aislada del relevamiento, sino una actividad continua que acompañó todo el ciclo de desarrollo.

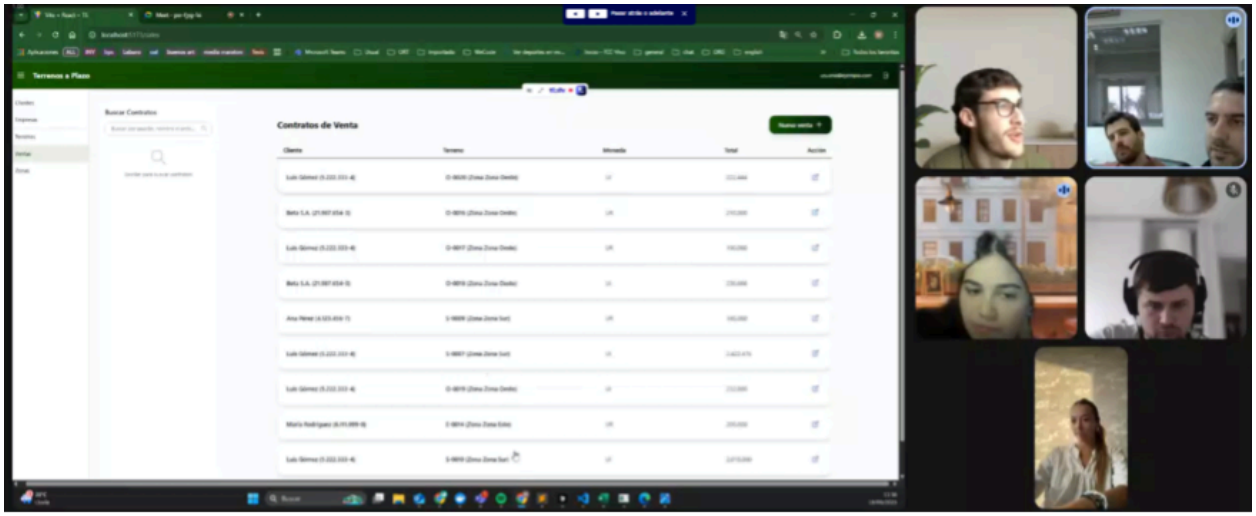


Figura 10. Entrevistas de validación

Este proceso fue donde se aplicaron por primera vez las heurísticas de Nielsen como criterio de evaluación del diseño (ver sección 7.2.2 Usabilidad (RNF-02) y ejemplos visuales en el anexo 7 - Capturas de pantalla del sistema implementado).

6.1.2. Especificación

Los requerimientos funcionales fueron especificados como historias de usuario en formato estándar (*Como [rol], quiero [funcionalidad], para [beneficio]*), complementadas con criterios de aceptación redactados en formato Gherkin (*Dado que... Cuando... Entonces...*) [14]. Este formato fue seleccionado por su claridad, su trazabilidad directa con las pruebas de aceptación, y su alineación natural con la metodología Scrum.

Cada historia de usuario fue estimada en *story points* [15] mediante *Planning Poker* [15] durante las sesiones de *Sprint Planning*, y asociada a la épica correspondiente dentro del *Product Backlog* gestionado en Linear [5].

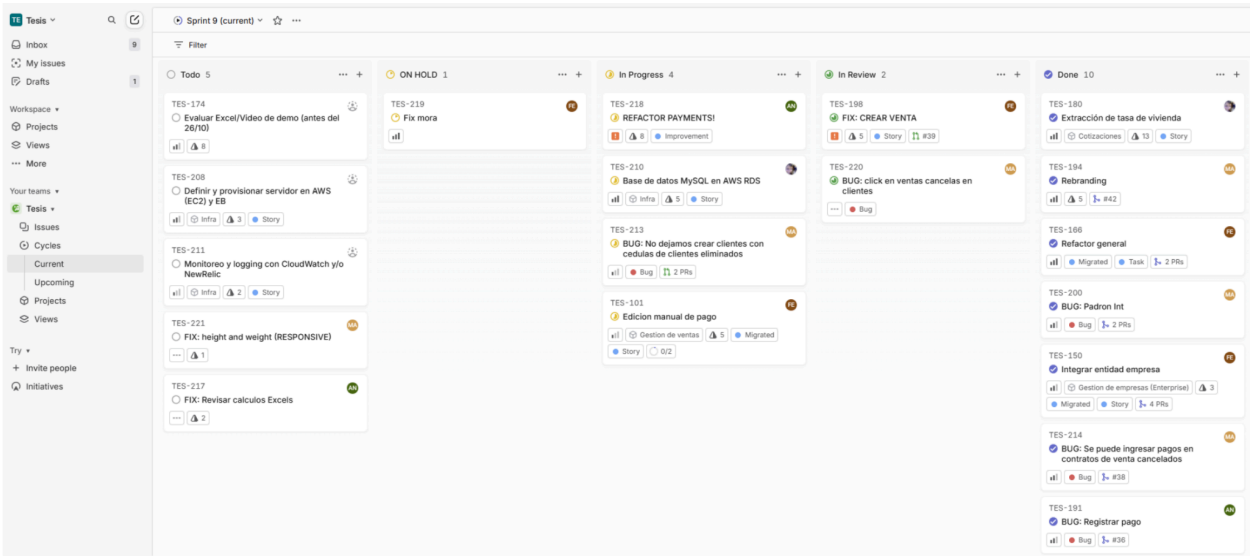


Figura 11. Tablero del Sprint Backlog en Linear

Las historias fueron redactadas por el *Product Owner* y revisadas por el equipo antes de ingresar al *Sprint Backlog* que se ve en la Figura 10, como parte del cumplimiento de la *Definition of Ready* (DoR) [16]:

- Historia estimada en *story points*.
- Criterios de aceptación redactados en formato Gherkin.
- Historia dividida en sub-tareas de implementación.
- Asociada a la épica correspondiente.

Para los requerimientos no funcionales, se utilizó un formato estructurado que incluye: descripción, prioridad (MoSCoW) [17], atributo de calidad según ISO/IEC 25010 [18], razón de ser, criterio de aceptación medible, y tácticas arquitectónicas empleadas. Este formato permite trazar cada RNF con las decisiones de arquitectura y con las métricas que lo verifican, respondiendo al principio de que todo requerimiento debe ser verificable (ver anexo 5 - Ejemplo de Historia de usuario).

6.1.3. Validación

La validación de la solución se realizó de forma incremental a lo largo del proyecto, combinando instancias de revisión temprana con el cliente y validación directa del sistema en uso real. Estas instancias permitieron verificar tanto la adecuación funcional de la solución como su alineación con la operativa diaria del negocio.

6.1.3.1. Validación de prototipos

Durante la etapa de diseño se utilizaron prototipos de baja y media fidelidad para validar las ideas de solución con el cliente. Estas representaciones permitieron discutir la organización de la información, los flujos de uso principales y las prioridades operativas antes de avanzar con la implementación.

Posteriormente se elaboraron prototipos de mayor fidelidad que fueron presentados al cliente para revisar aspectos de usabilidad, navegación y jerarquía visual. Estas instancias de validación permitieron detectar ajustes necesarios y reducir el riesgo de retrabajo durante la fase de desarrollo.

6.1.3.2. Validación durante el desarrollo

A lo largo del desarrollo se realizaron revisiones periódicas con el cliente para presentar avances funcionales del sistema y recibir retroalimentación. Estas revisiones permitieron confirmar que las funcionalidades implementadas respondieran a las necesidades reales del negocio y ajustar decisiones de diseño cuando fue necesario.

Este proceso iterativo facilitó la incorporación temprana de mejoras y contribuyó a mantener una alineación continua entre el sistema desarrollado y la operativa del cliente.

6.1.3.3. Validación en producción

Una vez desplegado el *Release* 1 en ambiente productivo con datos reales, el sistema comenzó a ser utilizado de forma continua por el cliente en su operativa diaria. Esta instancia constituyó la validación más robusta del sistema, ya que permitió evaluar su funcionamiento en condiciones reales de uso.

Durante esta etapa se habilitó además un documento compartido con el cliente para registrar observaciones, dudas y sugerencias detectadas durante el uso cotidiano de la plataforma. Este mecanismo permitió centralizar el *feedback* recibido y priorizar ajustes en función de su impacto en la operativa.

A partir de esta retroalimentación surgieron distintos ajustes funcionales que fueron incorporados en *sprints* posteriores mediante el proceso de integración continua. Por ejemplo, se realizaron adaptaciones para contemplar la variabilidad de reglas financieras según la moneda del contrato, la correcta aplicación de comisiones asociadas a redes de cobranza y la posibilidad de registrar múltiples compradores asociados a un mismo terreno.

Estos ajustes permitieron alinear aún más el sistema con las particularidades reales del negocio y mejorar su adecuación a los procesos operativos del cliente.

6.1.3.4. Encuestas de satisfacción post-release

Luego de cada *release* se aplicaron encuestas estructuradas al administrador del negocio responsable de la transición hacia DATIUM, con el objetivo de evaluar de forma empírica la experiencia de uso, la adopción de la herramienta y el cumplimiento del objetivo OProy04 (ver sección 1.2.2. Objetivos del proyecto). El instrumento combinó indicadores operativos medidos mediante escala Likert de 1 a 5 [19], definidos como umbral formal de cumplimiento del objetivo, con preguntas de satisfacción

general e intención de continuidad uso en la escala de 1 a 10 y preguntas abiertas destinadas a relevar retroalimentación cualitativa.

Los resultados cuantitativos muestran una asimilación exitosa de la herramienta en la operativa diaria y el cumplimiento del criterio de éxito definido, que exigía una puntuación media igual o superior a 4 sobre 5 en los indicadores evaluados. En ambas entregas, las métricas vinculadas a eficiencia y usabilidad, tales como la capacidad de realizar tareas operativas sin asistencia, la reducción del tiempo necesario para registrar pagos, el ahorro de trabajo manual generado por la automatización de cuotas y la claridad de la interfaz, obtuvieron de forma unánime la calificación máxima de 5 sobre 5.

Resulta especialmente relevante la evolución observada en la percepción de la estabilidad y la integridad de los cálculos financieros. Luego del primer *release*, la coincidencia entre los cálculos del sistema y los controles manuales realizados por la empresa obtuvo un puntaje de 5 sobre 5, mientras que la confianza declarada por el usuario en la exactitud de la automatización de cuotas, intereses y mora alcanzó inicialmente una calificación de 4 sobre 5. Luego del despliegue del segundo *release* y de un mayor tiempo de uso continuo de la plataforma, el nivel de confianza en el motor de cálculos ascendió a la calificación máxima de 5 sobre 5. Este incremento valida la estrategia de transición gradual adoptada durante la implementación, que permitió al equipo administrativo verificar la correctitud del sistema en condiciones reales antes de consolidar su uso como referencia operativa.

En la sección cualitativa las respuestas abiertas confirman que la solución logró mitigar los principales problemas detectados durante la fase de relevamiento. Ante la consulta sobre los aspectos más valorados, el usuario destacó en el primer *release* la eliminación del error manual en los cálculos, mientras que en el segundo *release* señaló como principal atributo la simpleza de uso de la plataforma. Asimismo, al indagar sobre posibles dificultades, confusiones o comportamientos inesperados, las

respuestas indicaron la ausencia de fricciones operativas relevantes, expresadas mediante comentarios como “Nada” y “Por el momento nada”.

Finalmente, el impacto global de la evolución iterativa queda reflejado en las métricas de satisfacción: la valoración general aumentó de 8/10 al finalizar el *Release* 1 a 9/10 luego del *Release* 2. De forma coherente, la probabilidad declarada de continuar utilizando DATIUM a largo plazo también subió de 8/10 a 9/10, lo que confirma una adopción estructural favorable de la plataforma en el modelo de negocio del emprendimiento.

En resumen, las encuestas validan que se alcanzaron los objetivos operativos planteados: aceptación por parte del usuario administrativo, mejora de la eficiencia en tareas clave y consolidación de la confianza en los cálculos financieros automatizados.

El detalle completo de las preguntas y respuestas de estas encuestas se presenta en el anexo 6 - Encuestas de satisfacción.

6.1.4. Verificación

La verificación (confirmación de que los requerimientos están correctamente especificados, son consistentes y no ambiguos) se realizó mediante las siguientes prácticas.

Definition of Ready (DoR): Antes de que una historia de usuario pudiera ingresar a un *sprint*, debía cumplir con la Definition of Ready definida por el equipo: redacción en formato Gherkin con criterios de aceptación claros, estimación realizada mediante *Planning Poker*, y división en tareas técnicas. Este mecanismo aseguraba que ningún requerimiento ingresara al *sprint* con ambigüedades o información faltante.

Revisión cruzada de historias de usuario: Antes de ingresar al *Sprint Backlog*, cada historia era revisada por al menos un integrante distinto del *Product Owner*, verificando la claridad de los criterios de aceptación y la completitud de la especificación.

Definition of Done (DoD) [20]**:** Cada historia implementada debía cumplir: ejecución exitosa de todos los *tests* (unitarios y de integración) con cobertura de código lógico ≥ 80% en el *backend* y revisión del PR por un integrante distinto del autor siguiendo el paso a paso definido en su descripción que cubría todos los casos lógicos afectados (ver anexo 13.8. Anexo 8 - Template de Pull Request). Estas especificaciones en el PR garantizan que las pruebas sean reproducibles.

Trazabilidad requerimiento–prueba: Los criterios de aceptación en formato Gherkin se utilizaron como base directa para la escritura de pruebas automatizadas, asegurando que cada requerimiento fuera verificable de forma objetiva y reproducible.

Validación en entorno de desarrollo: Luego de la integración de cada cambio en la rama develop, el equipo verificaba el comportamiento del incremento en el entorno de desarrollo en AWS, que replica la configuración del entorno productivo. Esta práctica permitía detectar problemas de integración antes de promover los cambios a producción

6.2. Funcionalidades

Los requerimientos funcionales se organizan en dos grupos: funcionalidades del núcleo del sistema (*core*), que responden directamente a los dolores operativos del cliente, y funcionalidades extendidas, incorporadas en el marco académico de la tesis y controladas mediante *feature flags* para no afectar la estabilidad del sistema productivo. Estas últimas no fueron solicitadas por el cliente y se incorporaron con el objetivo de demostrar la extensibilidad de la arquitectura. La especificación completa de cada requerimiento funcional, incluyendo sus criterios de aceptación en formato Gherkin y la estimación en *story points*, se encuentra en la herramienta de gestión del proyecto (Linear). En el anexo 5 - Ejemplo de Historia de Usuario se incluye un ejemplo representativo de historia de usuario con el nivel de detalle utilizado.

Para proveer respaldo empírico sobre la materialización de las funcionalidades descritas, la evidencia visual de las interfaces desarrolladas se encuentra documentada de forma exhaustiva en el anexo 7 - Capturas de pantalla del sistema implementado. Dicho anexo agrupa las capturas de pantalla respetando la misma categorización de los requerimientos funcionales que se expone en las tablas siguientes

A continuación se presenta la tabla consolidada de requerimientos funcionales con su descripción y priorización

6.2.1. Funcionalidades del núcleo (core)

Estas funcionalidades cubren la totalidad de las operaciones del núcleo del sistema. Incluyen tanto las que forman parte del *Golden Path* del usuario administrador (desde el alta de terrenos y compradores hasta la generación de reportes para auditoría) como las que forman parte del *Golden Path* del usuario comprador a través del canal de WhatsApp, además de funcionalidades complementarias necesarias para la operativa completa.

ID	Funcionalidad	Descripción	Prioridad
RF-01	Gestión de empresas y etapas	Alta, consulta (con filtros) y edición de empresas y sus etapas. Una empresa puede contener múltiples etapas; cada etapa agrupa un conjunto de terrenos (padrones).	Must
RF-02	Gestión de terrenos (padrones)	Alta, consulta (con filtros) y edición de terrenos identificados por número de padrón y asociados a una etapa. El padrón constituye la entidad base sobre la que se registran los contratos de venta.	Must

RF-03	Gestión de compradores	Alta, consulta (con filtros) y edición de compradores con datos personales y de contacto. Desde la ficha de cada comprador se puede acceder al listado completo de sus contratos (activos y cancelados).	Must
RF-04	Gestión de contratos de venta	Registro, edición y consulta de los términos del compromiso de compraventa: padrón, comprador(es) (hasta 4), moneda, monto total, cantidad de cuotas, seña, fechas y condiciones específicas (IVA, modalidades de pago). Incluye un sidebar de filtrado avanzado que permite buscar por texto (nombre, cédula o padrón), filtrar por categorías cruzadas (moneda, estado de financiación, empresa, etapa) y ordenar resultados por padrón, próximo vencimiento o cantidad de cuotas vencidas.	Must
RF-05	Generación automática de cuotas	A partir de un contrato registrado, el sistema genera el cronograma de cuotas con fechas de vencimiento y montos distribuidos en la moneda del contrato. Para los cálculos de conversión a UYU al momento del pago, el sistema consulta bajo demanda las cotizaciones de UI, UR y USD desde el BCU, con un mecanismo de *fallback* de tres	Must

		niveles: intento con la fecha exacta, retroceso de hasta siete días hacia atrás para fines de semana y feriados, y como último recurso consulta del último cierre disponible sin restricción de fecha. Las cotizaciones obtenidas se persisten en base de datos para asegurar consistencia histórica.	
RF-06	Gestión de pagos	Registro y administración de pagos asociados a contratos, con validaciones de negocio sobre monto, moneda y fecha. También se registra el identificador del comprobante, el medio de pago y la cuenta bancaria utilizada. Contempla distintas casuísticas operativas, incluyendo contratos en UI, UR o USD, pagos de cuotas o cancelaciones al contado, y la opción de prórroga hasta fin de mes. Calcula automáticamente excedentes o faltantes y actualiza los saldos y moras correspondientes. Permite la edición posterior del pago, recalculando los efectos sobre el mismo pago y sobre las cuotas posteriores cuando corresponda.	Must
RF-07	Cálculo automático de	Cálculo de recargos por mora según las reglas del negocio y la moneda del contrato, utilizando la tasa de vivienda publicada por el BCU. Contempla días de atraso, moneda de	Must

	mora e intereses	la deuda, aplicación condicional de IVA sobre la mora, y comisiones de redes de cobranza (Abitab / Redpagos).	
RF-08	Reportes operativos	El sistema permite generar reportes operativos consolidados, aplicando filtros por fecha, padrón y moneda, con visualización interactiva en pantalla y opción de exportación a Excel. Incluye los siguientes reportes: *1)* Cobranza*:* resume los montos cobrados en un período, discriminando cuota, mora y comisiones, con posibilidad de conversión a UYU según la cotización correspondiente a la fecha del pago. *2)* Ventas*:* presenta el detalle y los resúmenes de los contratos registrados, con opción de conversión a una moneda de destino. *3)* Cuotas vencidas: informa el monto total vencido por moneda para una fecha de corte determinada. *4)* Deudores: identifica a los compradores con deuda vigente.	Must

		5) Ingreso del ejercicio por padrón: muestra los ingresos asociados a cada padrón dentro del ejercicio.	
RF-09	Exportación a Excel	Exportación de reportes a formato Excel con hojas de detalle y resúmenes, preparados para análisis contable y auditoría.	Must
RF-10	Consulta de estado de cuenta por WhatsApp	*Bot* de WhatsApp que permite a los compradores consultar de forma autónoma información vinculada a su contrato y a su estado de cuenta. El flujo comienza con la verificación del número de celular del comprador; si este no se encuentra registrado en el sistema, se le indica que debe contactar a la empresa. En caso contrario, se le presenta el padrón asociado como mecanismo de confirmación de identidad. Una vez validado, accede a un menú principal con opciones definidas por el cliente: aviso de pago, próximo vencimiento, valor de la próxima cuota, cantidad de cuotas pagas, cantidad de cuotas pendientes, cantidad de cuotas vencidas, estado de cuenta completo y salida. La opción “aviso de pago” se encuentra prevista en el flujo, pero al cierre del proyecto responde mediante un mensaje informativo, ya que la recepción de comprobantes	Should

		adjuntos quedó definida como una evolución posterior (ver 3.3.2 Evolución y ajuste del alcance)	

Tabla 2: Funcionalidades

6.2.2. Funcionalidades extendidas (feature flags)

Estas funcionalidades se controlan mediante *feature flags* gestionados a través de ConfigCat, que permite activarlas o desactivarlas de forma independiente sin afectar el núcleo del sistema ni requerir nuevos despliegues.

Fueron incorporadas al *Product Backlog* como respuesta directa a la exclusión de la migración histórica masiva. Dado que el cliente prefirió postergar la normalización de sus datos, fue necesario sustituir el esfuerzo técnico originalmente previsto para la migración con el fin de cumplir con las exigencias académicas del proyecto. (Ver sección 3.3.2 Evolución y ajuste del alcance)

Si bien estas herramientas no formaban parte de las necesidades operativas explícitamente solicitadas al inicio, fueron diseñadas a partir de un análisis exhaustivo del dominio del negocio. Su desarrollo se llevó a cabo con un doble propósito: demostrar la extensibilidad de la arquitectura propuesta y aportar valor directo a la operativa del cliente, proporcionándole capacidades analíticas y de automatización que optimizan su seguimiento y control diario.

ID	**Funcionalidad**	**Descripción**	**Prioridad**
RF-11	*Chatbot* interno con IA	Consultas internas en lenguaje natural sobre la base de datos, con traducción controlada a SQL, devolución de resultados e interpretación. Incluye control de acceso por	Could

		feature flag, registro de consultas y resultados en logs de aplicación, y mecanismo de *feedback* por consulta (útil / no útil).	
RF-12	Notificaciones automáticas por email	Envío de notificaciones por correo electrónico a compradores con cuotas en mora. Se dispara automáticamente al registrar un pago con atraso, con un *cooldown* de siete días entre notificaciones al mismo comprador (definido como constante en el código). Cada envío se registra en base de datos con estado (pendiente, enviado, fallido), fecha y destinatario. El envío es no bloqueante: si falla, no afecta la operación de pago.	Could
RF-13	*Dashboard* interactivo	Panel con métricas clave del negocio: estado de cobranza, morosidad y recaudación, visible de un vistazo sin necesidad de generar reportes individuales.	Could

Tabla 3: Funcionalidades extendidas

Ver ejemplo de historia de usuario completa con criterios de aceptación en formato Gherkin en el anexo 5 - Ejemplo de Historia de usuario.

6.3. Requerimientos no funcionales

Los requerimientos no funcionales se definieron tomando como referencia la norma ISO/IEC 25010. Se seleccionaron los atributos de calidad con mayor impacto en el uso real del sistema en producción, considerando el contexto operativo del cliente: un

equipo administrativo que utiliza el sistema diariamente para gestionar operaciones financieras sensibles.

6.3.1. RNF-01 – Integridad de datos

Descripción: El sistema debe garantizar la consistencia total de los datos financieros en todas las operaciones, sin pérdidas, duplicaciones ni alteraciones. Toda operación financiera debe ejecutarse de forma atómica y dejar un registro auditable.

Prioridad: Must

Atributo de calidad (ISO 25010): Fiabilidad – Madurez.

Razón de ser: El sistema gestiona información financiera real. Cualquier inconsistencia en cálculos de cuotas, mora o balances puede derivar en pérdidas económicas, conflictos con compradores o problemas de auditoría.

Criterio de aceptación: Conciliación diaria sin diferencias entre los datos del sistema y los cálculos financieros esperados. Ninguna operación financiera puede completarse parcialmente.

Tácticas arquitectónicas: Constraints de integridad referencial en MySQL, transacciones ACID para operaciones financieras, validaciones de negocio en la capa de servicio de FastAPI, marcas temporales de creación en entidades financieras.

6.3.2. RNF-02 – Usabilidad

Descripción: La interfaz debe ser intuitiva, eficiente y orientada a reducir la carga cognitiva del usuario, reemplazando la fricción operativa de las planillas de Excel por flujos guiados y claros. El sistema debe ser adoptable con capacitación mínima.

Prioridad: Must

Atributo de calidad (ISO 25010): Usabilidad – Facilidad de aprendizaje, Operabilidad, Protección contra errores de usuario.

Razón de ser: El dolor principal del cliente era la fricción y dificultad para comprender información dispersa en múltiples planillas. La usabilidad fue la preocupación central de cara al producto final: todo aspecto que repercutiera en la interfaz fue sometido a evaluación y cambio constante. Un sistema técnicamente correcto pero difícil de usar no lograría la adopción necesaria para reemplazar las planillas.

Este criterio también guió la elección del canal de comunicación con los compradores: la experiencia previa del cliente con un portal web en Wix evidenció baja adopción, lo que llevó a priorizar WhatsApp como canal de autoservicio por ser la plataforma que los compradores ya utilizan diariamente (ver sección 7.3, ADR-06).

Criterio de aceptación: Los usuarios pueden realizar tareas clave (registrar pagos, consultar contratos, generar reportes) sin asistencia del equipo de desarrollo. Resultado positivo en encuestas de satisfacción *post-release*. Cumplimiento de las heurísticas de Nielsen [13] evaluadas *(ver "Evaluación usando heurísticas de Nielsen" a continuación).*

Tácticas de Implementación: Diseño iterativo con prototipos de baja, media y alta fidelidad validados con el cliente; flujos guiados para tareas críticas; jerarquía visual con información priorizada; videos de demostración; iteración continua basada en *feedback* de uso real en producción.

Evaluación de cumplimiento del RNF-02 mediante heurísticas de Nielsen

Las heurísticas de Nielsen se utilizaron como criterio de evaluación en dos instancias: durante el diseño de prototipos de alta fidelidad (evaluación a priori) y luego de la puesta en producción del sistema con datos reales (evaluación a posteriori). A continuación se listan las heurísticas que seleccionamos por ser las más relevantes

para el contexto operativo, donde los usuarios realizan tareas financieras repetitivas y la eficiencia y prevención de errores son críticas:

#1 – Visibilidad del estado del sistema: El sistema informa al usuario en todo momento sobre el estado de las operaciones mediante indicadores visuales claros: estados de cuotas diferenciados por color (vencida, pendiente, pagada), contadores actualizados y confirmaciones explícitas luego de acciones que modifican datos (registro de pagos, alta de contratos, cancelaciones).

#2 – Coincidencia entre el sistema y el mundo real: La terminología utiliza el vocabulario del negocio inmobiliario del cliente (padrón, cuota, mora, prórroga, comprador), evitando jerga técnica. El glosario de términos del dominio utilizado en la interfaz se incluye al inicio de este documento. Las pantallas reflejan los conceptos que los administradores ya manejan en su operativa diaria.

#5 – Prevención de errores: Las operaciones financieras críticas incluyen validaciones previas al envío, campos obligatorios señalizados y confirmaciones antes de ejecutar acciones irreversibles.

#6 – Reconocimiento antes que recuerdo: La información necesaria para tomar decisiones está visible en pantalla (detalle del contrato, cuotas pendientes, historial de pagos) sin requerir que el usuario recuerde datos de otras pantallas.

#7 – Flexibilidad y eficiencia de uso: Filtros por fecha, padrón y moneda para adaptar la visualización a la tarea en curso, y accesos directos a las acciones más frecuentes (ingresar pago, ver detalle de contrato).

A modo ilustrativo, en el anexo 7 - Capturas de pantalla del sistema implementado se incluyen comentarios sobre imágenes seleccionadas que evidencian visualmente la aplicación práctica de estas heurísticas en las distintas interfaces de la plataforma.

6.3.3. RNF-03 – Seguridad y trazabilidad

Descripción: El sistema debe proteger los datos financieros y personales mediante un control de acceso estricto por niveles, y proporcionar mecanismos de registro que permitan reconstruir la secuencia temporal de las operaciones de negocio.

Prioridad: Must

Atributo de calidad (ISO 25010): Seguridad – Confidencialidad, Integridad, Responsabilidad (*Accountability*).

Razón de ser: El sistema maneja datos financieros y personales sensibles. La confianza del cliente depende de percibir que los datos están seguros y que existe respaldo ante cualquier discrepancia.

Criterio de aceptación: Solo usuarios previamente autorizados por un administrador pueden acceder. Las entidades principales registran marcas temporales de creación que permiten reconstruir la cronología de las operaciones. No existen rutas de la API sin protección de autenticación.

Nota sobre Accountability: El sistema no implementa una auditoría formal con atribución de usuario por operación (registro de quién modificó qué y cuándo). Esta limitación fue una decisión deliberada dado el contexto operativo cerrado del cliente, y fue identificada como mejora para una eventual escala del sistema (ver sección 11.3 Lecciones aprendidas).

Tácticas arquitectónicas: El detalle de las tácticas implementadas para satisfacer este requerimiento se encuentra en la sección 7.2.3 Seguridad trazabilidad (RNF-03) y en los ADR-03 y ADR-04.

6.3.4. RNF-04 – Mantenibilidad

Descripción: El sistema debe poder evolucionar y desplegarse de forma segura sin depender del equipo de desarrollo original.

Prioridad: Must

Atributo de calidad (ISO 25010): Mantenibilidad – Modularidad, Analizabilidad, Modificabilidad, Testabilidad.

Razón de ser: Puede que el equipo de tesis eventualmente deje de mantener el sistema. Es fundamental que cualquier persona con conocimientos técnicos pueda retomar el proyecto, comprender su estructura y desplegarlo siguiendo la documentación.

Criterio de aceptación: Cobertura de *tests* ≥ 80% en lógica de negocio, *backend*. Un desarrollador externo puede desplegar una nueva versión siguiendo la documentación, sin asistencia del equipo.

Tácticas arquitectónicas: *Pipeline* de CI con verificación de cobertura y análisis estático, infraestructura como código con Terraform, CI en GitHub Actions, arquitectura modular alineada al dominio.

6.3.5. RNF-05 – Observabilidad

Descripción: El sistema debe contar con mecanismos centralizados de monitoreo y diagnóstico que permitan detectar problemas de infraestructura y performance rápidamente, y responder con evidencia ante incidentes o reportes del cliente.

Prioridad: Should

Atributo de calidad (ISO/IEC 25010): Fiabilidad – Madurez. Mantenibilidad – Analizabilidad.

Razón de ser: Con el sistema en producción, el equipo necesita visibilidad sobre su comportamiento para diagnosticar incidentes con evidencia concreta.

Criterio de aceptación: Todos los *requests* a la API quedan registrados con trazas de punta a punta. Ante un incidente reportado, el equipo puede diagnosticar la causa raíz utilizando los logs y métricas disponibles.

Tácticas arquitectónicas: Centralización de logs y métricas con Amazon CloudWatch [21], monitoreo de performance (APM) con New Relic [22].

6.3.6. RNF-06 – Resiliencia

Descripción: El sistema debe contar con mecanismos de respaldo y recuperación que garanticen la continuidad del negocio ante un incidente.

Prioridad: Should

Atributo de calidad (ISO 25010): Fiabilidad – Recuperabilidad.

Razón de ser: El sistema es la única fuente de verdad del negocio. La pérdida de datos sería un evento crítico.

Criterio de aceptación: Existencia de backups verificados por 24hs. La recuperación ha sido validada restaurando un backup en un entorno separado (dev) y verificando la consistencia de los datos.

Tácticas arquitectónicas: Backups automatizados de MySQL, validación de recuperación en entorno alternativo, infraestructura reproducible con Terraform. Se tienen backups diarios de la base de datos en AWS con retención de 24 horas, para siempre retener la versión más reciente de la misma.

6.3.7. RNF-07 – Performance

Descripción: Las operaciones clave deben ejecutarse con tiempos de respuesta que no introduzcan fricción en la operativa diaria.

Prioridad: Should

Atributo de calidad (ISO 25010): Eficiencia de desempeño – Comportamiento temporal.

Razón de ser: Un sistema que se percibe lento en tareas cotidianas generaría resistencia a abandonar las planillas de Excel.

Criterio de aceptación: Las operaciones clave (registro de pagos, consulta de contratos, generación de reportes) se completan en tiempos que no generan percepción de lentitud. Además de una ausencia de quejas de rendimiento.

Tácticas arquitectónicas: Monitoreo de tiempos de respuesta mediante APM (New Relic) [22], optimización bajo demanda, *backend* con FastAPI [23].

6.4. Conclusiones y lecciones aprendidas

El proceso de ingeniería de requerimientos fue uno de los aspectos más desafiantes del proyecto, no por la complejidad de las funcionalidades en sí, sino por la complejidad del dominio financiero y la inmadurez de los procesos del negocio del cliente. De esta etapa se derivan las siguientes lecciones clave.

Descubrimiento progresivo de requerimientos: Al profundizar en la operativa se identificaron reglas y casos borde que no estaban explicitados al inicio. La documentación del cliente era incompleta y la información faltante emergió durante las sesiones de validación. Esto generó retrabajo, pero confirmó la necesidad y el valor del enfoque iterativo con validación continua.

La usabilidad como requerimiento transversal: La usabilidad no fue un requerimiento aislado sino un criterio que influyó en todas las decisiones: desde la elección de WhatsApp como canal (descartando un portal web con baja adopción) hasta las múltiples iteraciones del diseño validadas con usuarios. Evaluar sistemáticamente la usabilidad (por ejemplo, aplicando las heurísticas de Nielsen [13]) permitió tomar decisiones más objetivas y reducir errores en tareas financieras repetitivas.

Cobertura metodológica usando el enfoque de Tsumaki: La clasificación de técnicas según los ejes estático/dinámico y cerrado/abierto resultó útil no solo como ejercicio académico, sino como herramienta práctica para verificar que el relevamiento no se limitará a confirmar lo que el cliente ya sabía. Las técnicas del cuadrante dinámico-abierto (brainstorming, observación, prototipos exploratorios) fueron las que más descubrimientos generaron, mientras que las del cuadrante dinámico-cerrado (prototipos de alta fidelidad, validaciones funcionales) fueron las que más retrabajo evitaron.

***Golden Paths* como herramienta de priorización:** La definición de dos *Golden Paths* (administrador web y comprador por Whatsapp) desde etapas tempranas del proyecto permitió mantener el foco en las funcionalidades que realmente aportan valor, tanto para la priorización del backlog como para el diseño de la interfaz y la planificación de los *releases*.

El proyecto fue más complejo que el producto: En retrospectiva, el producto final no presenta una complejidad funcional extraordinaria. Sin embargo, el proyecto fue complejo por la necesidad de regularizar las inconsistencias del negocio para traducirlas a un sistema, y por el foco constante en la usabilidad que sometió cada aspecto de la interfaz a evaluación constante.

7. Arquitectura y diseño

7.1. Visión general de la arquitectura

Para describir la arquitectura del sistema, el equipo se basó en el enfoque *Views and Beyond* del Software Engineering Institute de Carnegie Mellon University [24], que propone documentar la arquitectura a través de múltiples vistas complementarias, cada una orientada a un conjunto de preocupaciones (*concerns*) distinto. Este enfoque permite presentar la arquitectura de forma progresiva: primero una visión de alto nivel y luego profundizaciones sucesivas en cada componente.

7.1.1. Diagramas del sistema

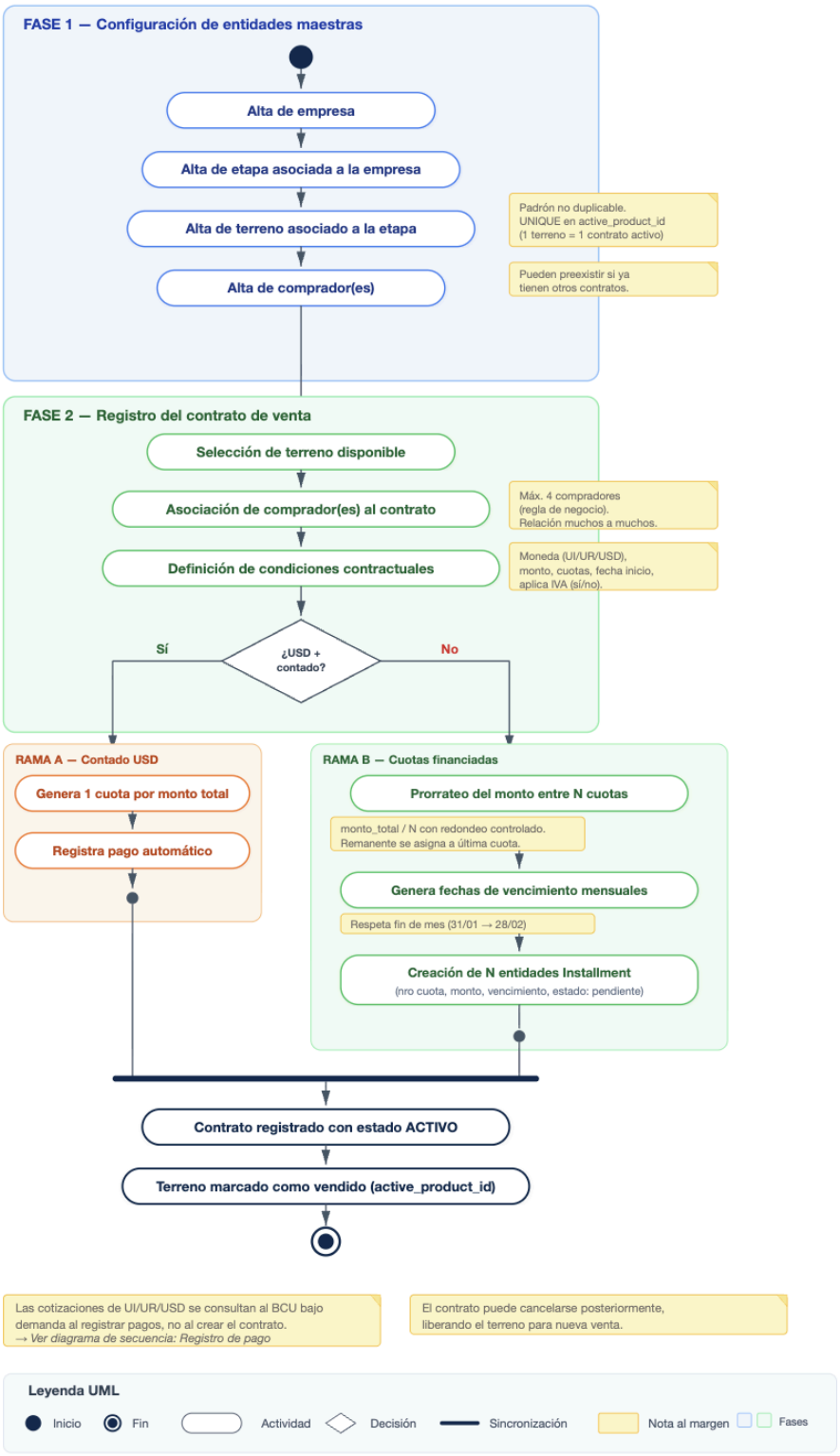


Figura 12. Diagrama de actividad: desde el alta de terreno hasta la generación de cuotas

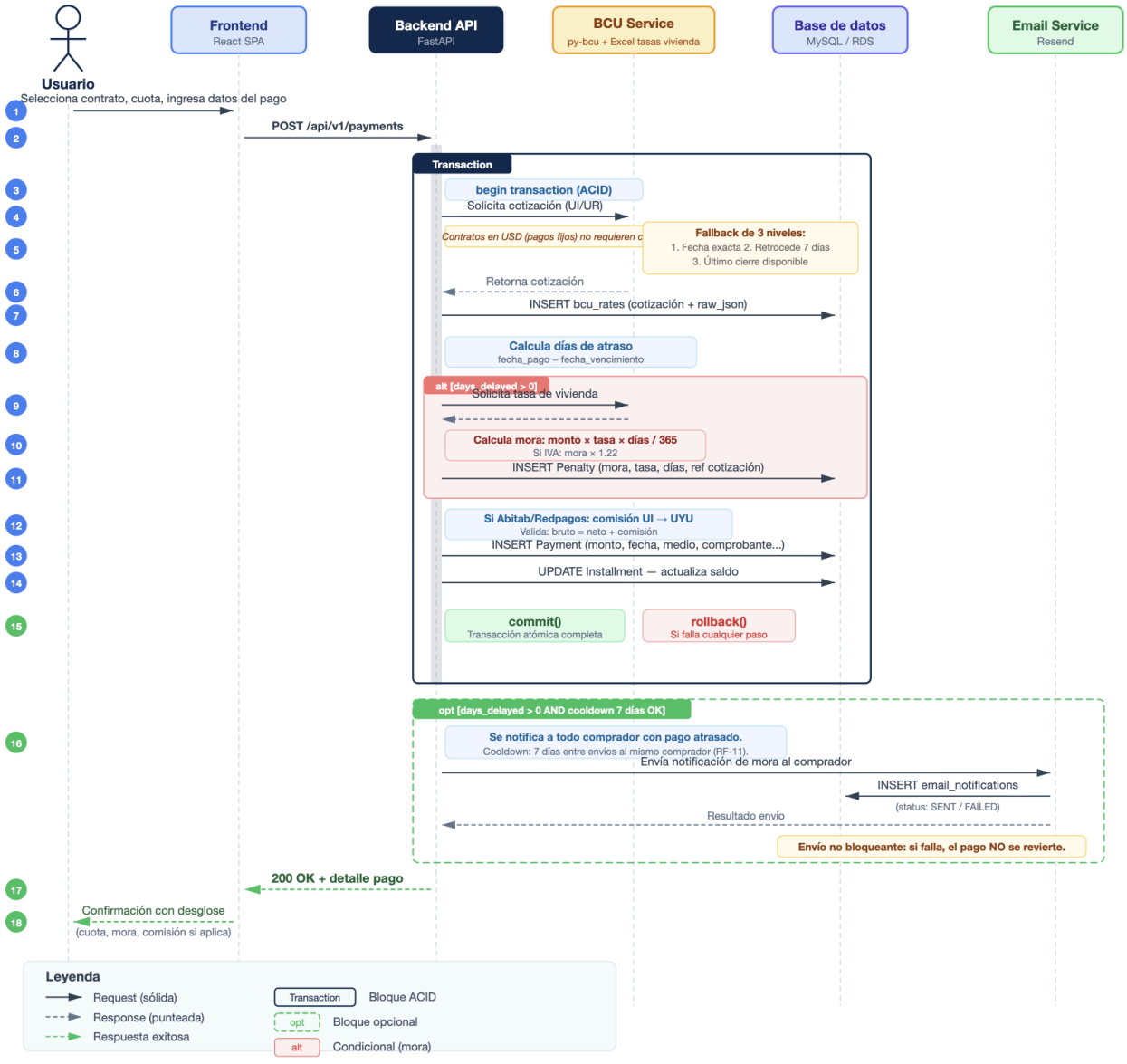


Figura 13. Diagrama de secuencia: Registro de pago con cálculo de mora

La solución se compone de los siguientes bloques principales (ilustrados en la Figura 14):

***Frontend* (SPA):** Aplicación web de página única (SPA) [25] desarrollada en React con TypeScript, desplegada como sitio estático en Amazon S3 y servida a los usuarios a través de Amazon CloudFront (CDN). Se comunica con el *backend* exclusivamente a través de una API REST.

API Gateway (AWS [26]): Punto de entrada HTTPS al *backend* en producción. Actúa como proxy entre el *frontend* y el servidor de aplicación, proporcionando terminación TLS de forma transparente.

***Backend* (API REST):** Servidor de aplicación desarrollado en FastAPI (Python), que expone una API REST bajo el prefijo `/api/v1`. Contiene la lógica de negocio, las validaciones, los cálculos financieros y la orquestación de integraciones externas. Se ejecuta en una instancia EC2 mediante Docker Compose. Previo al inicio del desarrollo, el equipo elaboró investigaciones internas sobre las integraciones clave del sistema. De forma similar, se documentaron guías de trabajo sobre convenciones de commits (*Conventional Commits*) [27] y organización del código *frontend* siguiendo Atomic Design [28]. Estas investigaciones se realizaron durante la etapa de validación técnica descrita en la sección 4.1 Cronología de las etapas y sirvieron como insumo para las decisiones arquitectónicas formalizadas en los ADRs de la sección 7.3 Architectural decision records (ADRs).

Base de datos (MySQL 8.4.8): Base de datos relacional que almacena toda la información del negocio, incluyendo contratos, cuotas, pagos, mora, cotizaciones del BCU, configuración del sistema y registros operativos. En producción corre en una instancia RDS de AWS con backups diarios automatizados; en desarrollo local corre como servicio Docker en la misma instancia.

Integraciones externas: El *backend* se conecta con servicios externos para funcionalidades específicas: el *web service* SOAP del BCU (vía la librería `py-bcu` [29])

para obtener cotizaciones y tasas de vivienda; la Meta WhatsApp Cloud API [30] para el canal de autoservicio de compradores; OpenAI [31] (modelo `gpt-4o-mini`) para el *chatbot* interno de consultas en lenguaje natural; ConfigCat [32] para la gestión de *feature flags*; Resend para el envío de notificaciones por correo electrónico; y OAuth 2.0 para la autenticación de usuarios.

Monitoreo: New Relic APM para trazas de *requests* de punta a punta y monitoreo de performance en producción, complementado con Amazon CloudWatch para monitoreo de los recursos de infraestructura (EC2, RDS, S3).

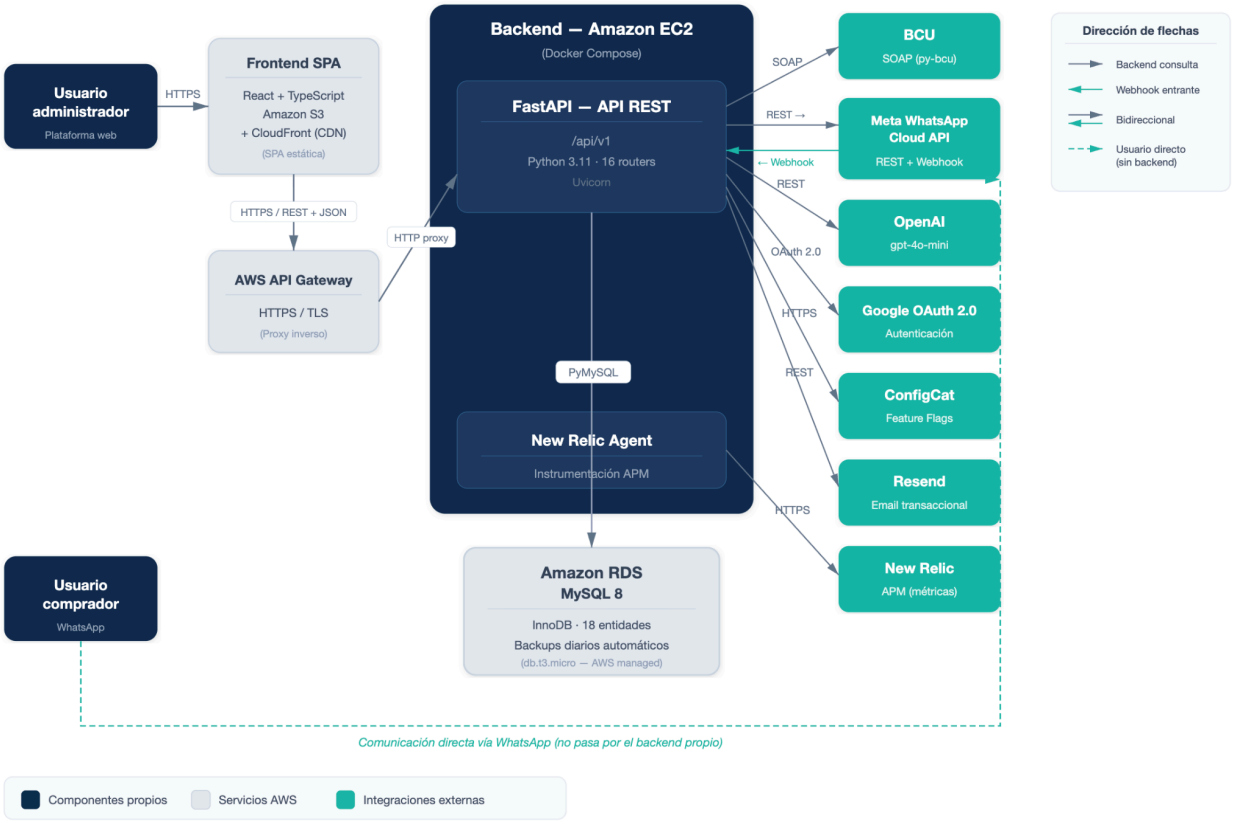


Figura 14. Diagrama de arquitectura del sistema

7.1.2. Organización de repositorios

Al inicio del proyecto, el código fuente se organizó en dos repositorios Git independientes: uno para el *backend* (Python/FastAPI) y otro para el *frontend* (React/TypeScript). Esta separación permite ciclos de desarrollo, *testing* y despliegue independientes para cada componente. Posteriormente, al implementar la automatización de despliegues a los entornos de AWS, se creó un tercer repositorio orquestador (all) que incorpora los dos repositorios anteriores como submodules de Git y contiene la infraestructura como código (Terraform), los *workflows* de despliegue a dev y prod, y los scripts auxiliares de despliegue del *frontend*. Esta organización permite que cada *pipeline* de despliegue opere sobre una foto consistente del *backend* y el *frontend* al momento de ejecutarse. El detalle de la estructura de cada repositorio y la estrategia de ramas se describe en la sección 10.3 Organización de los repositorios.

7.1.3. Flujo de comunicación

En ambiente local, el *frontend* (Vite dev server en puerto 5173) se comunica directamente con el *backend* (FastAPI en puerto 8000). Tanto en producción como en dev, el *frontend* desplegado en S3 realiza las peticiones HTTP a través de AWS API Gateway, que las redirige al *backend* en EC2. Toda la comunicación es API REST con formato JSON. La autenticación se gestiona mediante tokens JWT que el *frontend* incluye en el header `Authorization: Bearer` de cada *request*, obtenidos luego del flujo de OAuth 2.0.

7.2. Atributos de calidad

Los atributos de calidad priorizados en la sección 6.3 Requerimientos no funcionales guiaron las decisiones arquitectónicas del sistema. A continuación se describe cómo cada atributo se traduce en decisiones concretas de diseño y tecnología, estableciendo la trazabilidad entre los requerimientos no funcionales y la arquitectura implementada.

7.2.1. Integridad de datos (RNF-01)

El sistema maneja información financiera real sobre la cual el cliente toma decisiones de negocio. Para garantizar la integridad de los datos, se adoptaron las siguientes decisiones:

La base de datos MySQL fue seleccionada por su soporte nativo de transacciones ACID con el motor InnoDB, permitiendo que las operaciones financieras complejas (como el registro de un pago, que involucra la cuota, el cálculo de mora, la creación de la penalidad y la actualización del balance) se ejecuten de forma atómica. Las operaciones se organizan con `flush()` para las operaciones intermedias y `commit()` una sola vez al final, con `rollback()` explícito en caso de error.

Se definieron constraints de integridad referencial entre las tablas del modelo de datos (foreign keys con `ON DELETE CASCADE` donde corresponde), y un constraint `UNIQUE` en `active_product_id` que garantiza a nivel de base de datos que un terreno solo pueda tener un contrato activo simultáneamente.

Los cálculos financieros utilizan tipos `DECIMAL` con precisión adecuada: `DECIMAL(20,4)` para montos en moneda del contrato y `DECIMAL(15,2)` para montos en UYU. Las cotizaciones del BCU se persisten con `Numeric(30,10)` para preservar la precisión de las tasas. Cada pago y cada penalidad almacenan una referencia (*foreign key*) a la cotización utilizada en el momento de la operación, asegurando reproducibilidad histórica de los cálculos.

7.2.2. Usabilidad (RNF-02)

La usabilidad fue la preocupación central del producto. A nivel de arquitectura, las decisiones que la soportan incluyen:

La elección de una SPA con React permite transiciones fluidas entre pantallas sin recargas completas, resultando en una experiencia más ágil que las aplicaciones web

tradicionales. El uso de TanStack React Query [33] como gestor de *server state* habilita *caching* automático de datos del *backend*, reduciendo la latencia percibida en navegaciones repetidas.

El *frontend* fue construido con un sistema de componentes propio siguiendo el patrón Atomic Design (átomos, moléculas y organismos) [28], respaldado por *design tokens* centralizados que definen la paleta de colores, tipografía, espaciado y sombras. Esta decisión permite mantener consistencia visual en toda la plataforma y facilita la iteración del diseño sin impacto estructural. Por ejemplo, si se necesita cambiar el color primario de la aplicación o el espaciado entre elementos, basta con modificar el valor del token correspondiente en un único archivo, y el cambio se propaga automáticamente a todos los componentes que lo utilizan, sin necesidad de recorrer pantalla por pantalla.

7.2.3. Seguridad y trazabilidad (RNF-03)

Para satisfacer el RNF-03, se implementó un esquema de seguridad en múltiples capas que abarca autenticación, control de acceso, gestión de secretos y trazabilidad de operaciones. La justificación técnica y las alternativas evaluadas para las decisiones principales se documentan en los ADR-03 (autenticación con OAuth 2.0 y sistema de invitaciones) y ADR-04 (control de acceso binario). A continuación se describe la implementación.

La protección de datos financieros y personales se implementó en múltiples capas:

Autenticación: Se implementó un flujo OAuth 2.0 con Google como proveedor de identidad, complementado con un sistema de invitaciones (*allowlist*): solo las cuentas de correo electrónico previamente autorizadas por un usuario administrador pueden acceder al sistema. Concretamente, luego de completar el flujo de autenticación con Google, el *backend* verifica que el correo electrónico del usuario exista en la tabla AllowedEmail antes de conceder acceso; si no está en la lista, se redirige con un error indicando que el usuario no fue invitado. Luego de la autenticación, el *backend* emite un token JWT firmado con HMAC-SHA256 y expiración de 24 horas, que el *frontend* almacena y envía en cada request. El flujo OAuth 2.0 utiliza *signed state* con HMAC y timestamp para prevenir ataques CSRF sin necesidad de sesiones del lado del servidor. Este flujo se ilustra a continuación en la Figura 15.

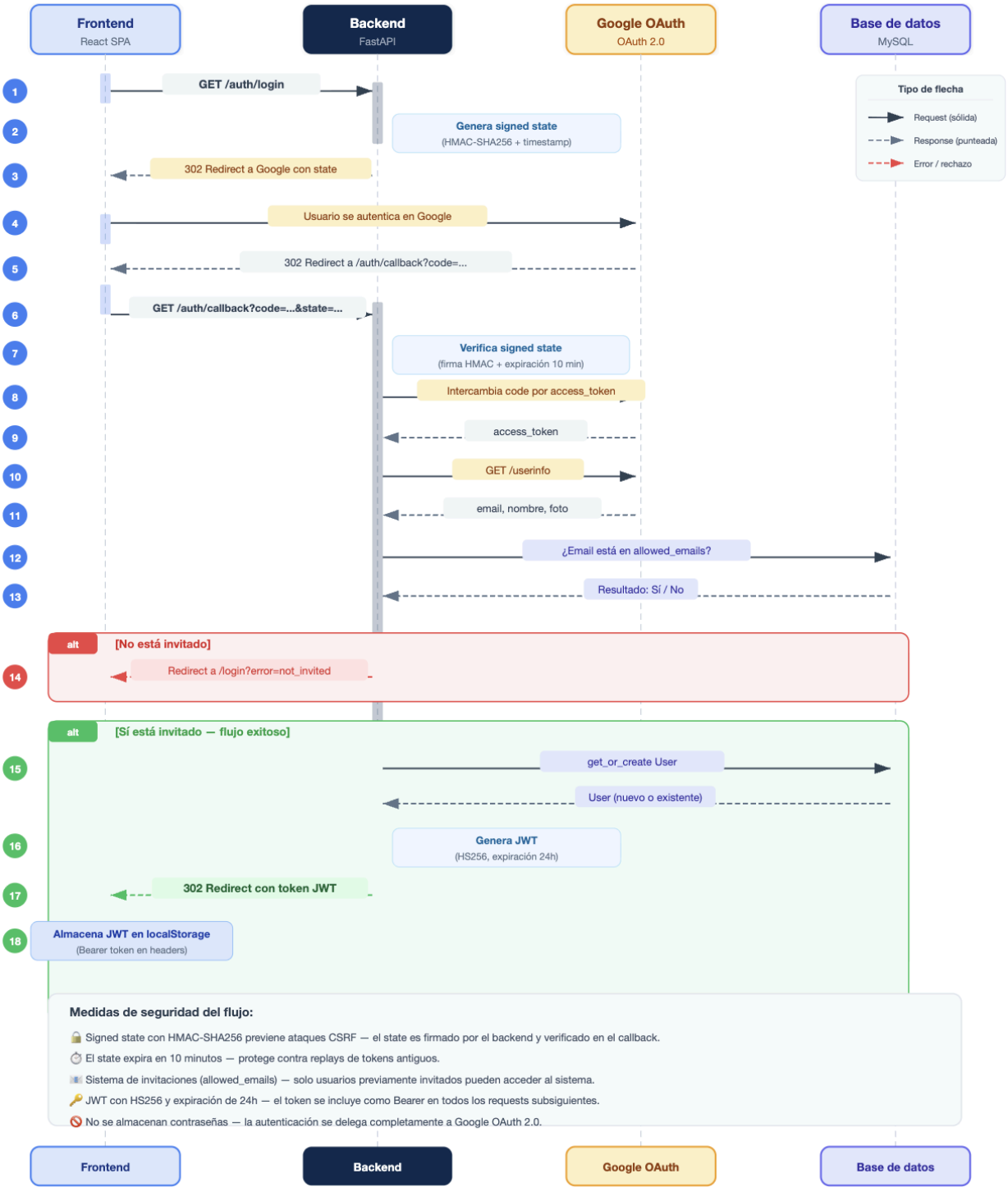


Figura 15. Flujo de autenticación

Control de acceso por niveles: El sistema implementa un esquema de dos niveles: usuario estándar y administrador. La autorización se resuelve mediante dependencias declarativas de FastAPI (`get_current_user` para cualquier usuario autenticado, `get_admin_user` para operaciones administrativas como gestión de invitaciones y configuración). Este esquema fue una decisión deliberada: dado que el sistema es operado por un equipo administrativo reducido y cerrado, un sistema de roles granulares habría añadido complejidad sin beneficio proporcional.

Gestión de secretos: Los secretos del sistema (credenciales de base de datos, claves de API, tokens de servicios externos) se gestionan como variables de entorno, diferenciadas por nivel: archivo `.env` en desarrollo local, GitHub Secrets para el *pipeline* de CI (donde los secretos se inyectan como variables de entorno `TF_VAR_` durante el despliegue y se propagan a la infraestructura mediante Terraform), y un archivo de variables por entorno (`terraform.prod.tfvars` y `terraform.dev.tfvars`) para valores no sensibles de infraestructura. En ambos entornos (dev y prod), los secretos siguen el mismo flujo: GitHub Secrets → variables de entorno del *workflow* → Terraform apply. La configuración de New Relic ejemplifica la práctica adoptada: el archivo de configuración no contiene el *license key*, que se delega a una variable de entorno.

Trazabilidad: Las entidades principales del sistema registran marcas temporales de creación (`created_at`). El *bot* de WhatsApp cuenta con registro de mensajes, métricas de uso y estado conversacional en la base de datos del sistema, complementado con las métricas que la propia plataforma de Meta provee sobre el canal (volumen de mensajes recibidos y enviados, entregas exitosas y tasas de lectura). Las notificaciones por email registran estado de envío y errores. Las cotizaciones del BCU persisten con la respuesta cruda (campo `raw_json`) como evidencia de los valores consultados. El sistema no implementa una tabla de auditoría formal con atribución de usuario por operación (quién modificó qué y cuándo). Esta decisión se tomó dado que el equipo administrativo es cerrado y de confianza, y que el

sistema ya registra marcas temporales de creación en las entidades principales. Una auditoría formal por acción de usuario fue identificada como mejora para una eventual escala del sistema (ver sección 11.3 Lecciones aprendidas). Igualmente, los registros existentes permiten reconstruir la secuencia temporal de las operaciones realizadas.

7.2.4. Mantenibilidad (RNF-04)

La mantenibilidad se aborda desde tres perspectivas: código, infraestructura y procesos.

Código: La API REST del *backend* fue diseñada con endpoints orientados a las entidades del dominio (/clients, /sale-contracts, /payments, /reports), utilizando nombres que reflejan el vocabulario del negocio y no abstracciones técnicas. El *backend* se organiza en capas claramente diferenciadas (routers, CRUD/lógica de negocio, modelos, schemas). Los schemas Pydantic [35] validan la entrada y salida de cada endpoint, y las excepciones de dominio están definidas por entidad con códigos y mensajes descriptivos. El *pipeline* de CI ejecuta análisis estático con `flake8` y formateo automático con `black`, que detecta errores de sintaxis, *imports* no utilizados y violaciones de estilo que dificultan la lectura del código. Los *tests* automatizados con cobertura mínima del 80% actúan como red de seguridad ante cambios: no garantizan la calidad del diseño, pero aseguran que las modificaciones en un módulo no rompan el comportamiento existente en otros, reduciendo el riesgo de regresiones al modificar o extender el sistema. El *frontend* utiliza TypeScript con verificación de tipos como condición de compilación en el *pipeline* de CI.

Infraestructura: La infraestructura en AWS se define como código mediante Terraform, gestionado en un repositorio separado por razones de seguridad. Esto permite versionar los cambios de infraestructura y reproducir el entorno de forma controlada.

Procesos: El equipo adoptó un flujo de trabajo basado en Gitflow [36], con dos ramas principales (develop y main) y ramas de feature para cada historia o tarea. Cada cambio se integra mediante PRs estandarizados, con revisión obligatoria de otro

integrante y ejecución automática de pruebas antes de la fusión. Para garantizar el orden y la trazabilidad, se implementó una plantilla predefinida de uso obligatorio para la creación de estos PRs (ver anexo 8 - Template de *Pull Request*).

7.2.5. Observabilidad (RNF-05)

La observabilidad del sistema se implementa en dos niveles. A nivel de aplicación, se integró New Relic APM como herramienta de monitoreo de producción. El agente se ejecuta como *wrapper* del proceso principal de Uvicorn (`newrelic-admin run-program uvicorn`), capturando trazas de requests de punta a punta, tiempos de respuesta y errores. La configuración de *distributed tracing* permite correlacionar requests del *frontend* con operaciones del *backend* y consultas a la base de datos. A nivel de infraestructura, Amazon CloudWatch recopila métricas y logs de los recursos de AWS (EC2, RDS, API Gateway, CloudFront), proporcionando visibilidad sobre el estado de los servicios, uso de recursos y alertas operativas.

7.2.6. Resiliencia (RNF-06)

Existen distintos niveles de resiliencia según la estrategia adoptada: recuperación ante fallos (*recovery*), donde el sistema se restaura luego una caída; reinicio automático (*self-healing*), donde el sistema detecta la caída y se levanta sin intervención; y redundancia activa (*high availability*), donde múltiples instancias corren en paralelo y absorben el tráfico si una falla.

El sistema actual opera en el primer nivel, con una estrategia centrada en la capacidad de recuperación. La base de datos en producción se respalda mediante backups diarios automatizados configurados en la instancia RDS de AWS, lo que permite restaurar el estado de la base a cualquier punto dentro del período de retención.

La infraestructura reproducible con Terraform permite reconstruir el entorno completo ante un fallo de infraestructura, y el sistema de migraciones *custom* garantiza que el esquema de base de datos pueda recrearse de forma idempotente a partir de los 28

scripts SQL versionados. No se implementaron estrategias de reinicio automático ni redundancia activa, dado que el volumen de usuarios y la criticidad temporal de la operación no lo justificaban en esta etapa.

7.2.7. Performance (RNF-07)

FastAPI fue seleccionado como framework de *backend* por su rendimiento basado en ASGI (el protocolo asíncrono de Python) y por la validación automática con Pydantic [35] que evita procesamiento innecesario de datos malformados. Según los benchmarks publicados por TechEmpower (ronda 22, 2023) [37], los frameworks ASGI de Python se ubican entre los de mayor throughput dentro del ecosistema Python, superando significativamente a frameworks sincrónicos como Django y Flask en escenarios de I/O concurrente. El uso de TanStack React Query [33] en el *frontend* reduce las peticiones redundantes al *backend* mediante *caching* con tiempos de expiración configurables. New Relic APM provee métricas de tiempos de respuesta a nivel de aplicación para identificar y corregir degradaciones en endpoints específicos, mientras que Amazon CloudWatch monitorea los recursos de infraestructura (uso de CPU y memoria en EC2, métricas de RDS, latencia del API Gateway), permitiendo detectar cuellos de botella que no se originan en el código sino en la capacidad del entorno.

7.3. Architectural decision records (ADRs)

Para documentar y fundamentar las decisiones arquitectónicas que dan forma a la solución, se utilizó el formato de *Architectural Decision Records* (ADRs) [38]. Cada ADR registra el contexto, la decisión tomada, las alternativas consideradas y las consecuencias, proporcionando trazabilidad entre los requerimientos no funcionales y las soluciones arquitectónicas implementadas.

ADR-01: Separación frontend/backend en repositorios independientes

Contexto: El sistema requiere una interfaz web para usuarios administrativos y una API *backend* con lógica de negocio compleja. El equipo necesita poder desarrollar, testear y desplegar cada componente de forma independiente.

Decisión: Se adoptó una arquitectura de *frontend* y *backend* separados en repositorios Git independientes, comunicados exclusivamente por API REST.

Alternativas consideradas:

- Aplicación monolítica con rendering del lado del servidor (Django, Flask con templates).
- Monorepo con ambos proyectos.

Consecuencias:

- Ciclos de desarrollo independientes.
- Despliegues desacoplados (el *frontend* se despliega como sitio estático en S3, el *backend* en EC2).
- Clara separación de responsabilidades.
- Requiere definir y mantener un contrato de API estable entre ambos componentes.

RNFs asociados: RNF-04 (Mantenibilidad), RNF-07 (Performance — SPA con *caching*).

ADR-02: MySQL como motor de base de datos

Contexto: El sistema gestiona operaciones financieras que requieren consistencia transaccional estricta y tipos numéricos de alta precisión.

Decisión: Se seleccionó MySQL 8.4.8 [39] con motor InnoDB como base de datos relacional.

Alternativas consideradas:

- PostgreSQL (funcionalidades avanzadas, mayor curva de aprendizaje para el equipo).
- SQLite (sin soporte de concurrencia adecuada).
- Base de datos NoSQL (sin soporte transaccional nativo).

Consecuencias:

- Soporte nativo de transacciones ACID.
- Tipos `DECIMAL` con precisión configurable para cálculos financieros.
- Amplia documentación y tooling; el equipo tenía experiencia previa con MySQL.

RNFs asociados: RNF-01 (Integridad de datos).

ADR-03: Google OAuth 2.0 con sistema de invitaciones (*Allowlist*)

Contexto: El sistema maneja datos financieros, contratos y saldos de deuda altamente sensibles. El acceso a la plataforma web debe estar restringido exclusivamente al núcleo administrativo del cliente.

Decisión: Se implementó autenticación mediante Google OAuth 2.0 [34] combinada con un sistema de control de acceso por *allowlist* (lista de permitidos). Exclusivamente las direcciones de correo electrónico registradas e invitadas de forma manual por un administrador pueden acceder a la plataforma. Para resolver el problema de inicialización del sistema (*bootstrapping*), el primer usuario con rol de administrador se ingresó directamente en la base de datos. A partir de este primer acceso, dicho usuario pudo operar la plataforma web para gestionar e invitar al resto del equipo administrativo

del cliente. Luego de la autenticación exitosa de un usuario permitido, se emite un token JWT con expiración de 24 horas.

Alternativas consideradas:

- Autenticación tradicional con usuario y contraseña administrados por el sistema (requiere gestión compleja de contraseñas y recuperación).
- Auth0 o Firebase Auth (costo adicional, dependencia externa para función *core*).
- Autenticación abierta con registro libre (descartada por representar un riesgo de seguridad inaceptable y aumentar innecesariamente la superficie de ataque).

Consecuencias: Esta decisión se alinea de manera óptima con la estructura de la empresa: al tratarse de un núcleo administrativo cerrado y de alta confidencialidad, se priorizó blindar el acceso (*Zero Trust* [40]) en lugar de automatizar el alta masiva de usuarios. Exponer un formulario público de registro representaba un riesgo de seguridad injustificado frente a la naturaleza privada de la operativa del cliente.

RNFs asociados: RNF-03 (Seguridad y trazabilidad).

ADR-04: Control de acceso binario (administrador / usuario estándar)

Contexto: El sistema es operado por un núcleo administrativo consolidado y altamente colaborativo. Se requiere proteger ciertas configuraciones críticas sin entorpecer la fluidez de las tareas cotidianas.

Decisión: Se implementó un esquema de autorización de dos niveles mediante un campo booleano `is_admin` en el modelo de usuario, resuelto con dependencias declarativas de FastAPI (`get_current_user, get_admin_user`).

Alternativas consideradas:

- RBAC con tabla de roles y permisos (complejidad desproporcionada para el alcance del proyecto).
- Permisos por endpoint, donde cada ruta de la API define individualmente qué usuarios pueden acceder (a medida que la API crece, la cantidad de reglas se multiplica y se vuelve propenso a errores de omisión, un problema documentado en la literatura de seguridad de APIs como *broken access control*, clasificado como el riesgo #1 en OWASP API Security Top 10 2023 [41]).

Consecuencias: Se logra una implementación robusta, fácil de auditar y perfectamente adaptada al modelo operativo del cliente. Todos los miembros del núcleo administrativo pueden operar funcionalmente el negocio (ventas, cobranzas), mientras que el rol de administrador queda reservado únicamente para la gestión de acceso (invitar nuevos miembros).

RNFs asociados: RNF-03 (Seguridad y trazabilidad), RNF-04 (Mantenibilidad — simplicidad).

ADR-05: Feature flags con ConfigCat para funcionalidades extendidas

Contexto: Las funcionalidades extendidas (RF-11 a RF-13) no fueron solicitadas por el cliente y se incorporaron en el marco académico. Es necesario poder activarlas o desactivarlas sin afectar el núcleo productivo del sistema.

Decisión: Se integró ConfigCat como servicio externo de *feature flags*. Tres flags (`isChatbotEnabled`, `isDashboardEnabled`, `isEmailNotificationsEnabled`) controlan la disponibilidad de las funcionalidades extendidas tanto en el *backend* (bloqueando endpoints) como en el *frontend* (ocultando elementos de navegación).

Alternativas consideradas:

- Flags en base de datos (requiere UI propia de administración).
- Variables de entorno (requiere redespliegue para cambiar).
- LaunchDarkly (mayor costo).

Consecuencias:

- Cambio de estado sin redespliegue ni modificación de código.
- Aislamiento claro entre funcionalidades *core* y extendidas.
- Dependencia de un servicio externo (mitigada por valores *default* `false`).
- Evaluación centralizada mediante SDK con caché local.

RNFs asociados: RNF-04 (Mantenibilidad — extensibilidad sin riesgo).

ADR-06: Bot de WhatsApp en lugar de portal web para compradores

Contexto: El cliente necesita un canal de comunicación con los compradores de terrenos para consultas de estado de cuenta. Existía un portal web en Wix que no logró adopción.

Decisión: Se implementó un *bot* de WhatsApp conectado al *backend* del sistema mediante la Meta WhatsApp Cloud API, con flujo conversacional basado en máquina de estados (ver anexo 9 - Proceso de habilitación del *chatbot* de WhatsApp mediante Meta para el proceso de habilitación del *bot*).

Alternativas consideradas:

- Portal web para clientes (descartado por la experiencia previa con Wix explicada en la sección 6.1.1.2. Técnicas (Análisis de soluciones existentes (Estático – Abierto)).
- Aplicación móvil (complejidad desproporcionada, requiere instalación).

Consecuencias:

- Canal de comunicación en la plataforma que los compradores ya usan diariamente; cero fricción de adopción (no requiere instalación ni registro).
- Datos en tiempo real conectados al *backend.*
- Limitaciones del canal (interacción basada en texto y botones, no permite interfaces complejas); dependencia de la disponibilidad de la API de Meta y de la política de ventana de 24 horas.
- Como requisito del proceso de verificación de Meta para habilitar la WhatsApp Cloud API en producción, fue necesario desarrollar y publicar una landing page pública del producto (https://www.datium.live/), que presenta la propuesta de valor del sistema y cumple con las políticas de Meta para la verificación de cuentas de negocio. Este requisito no estaba contemplado inicialmente y fue identificado durante el proceso de aprobación.

RNFs asociados: RNF-02 (Usabilidad), RNF-07 (Performance — respuestas directas sin navegación web).

ADR-07: Chatbot text-to-SQL con OpenAI y pipeline de seguridad en capas

Contexto: Se requiere un *chatbot* interno que permita realizar consultas en lenguaje natural sobre la base de datos, como funcionalidad extendida del sistema.

Decisión: Se implementó un *pipeline* de cuatro pasos:

1. pregunta natural
2. SQL parametrizado
3. Ejecución
4. Respuesta en español

Utilizando el modelo `gpt-4o-mini` de OpenAI, con un sistema de seguridad en siete capas para prevenir consultas maliciosas, como se puede ver en la Figura 15:

1. ***Feature flag* (envolvente global):** el *chatbot* se controla mediante el flag `isChatbotEnabled` en ConfigCat, que permite desactivar la funcionalidad completamente sin necesidad de redespliegue.

2. **Detección de preguntas fuera del dominio (antes de generar SQL):** valida que la pregunta sea sobre contratos, pagos o cuotas antes de invocar al modelo, rechazando todo lo demás para reducir el riesgo de alucinaciones.

3. ***Prompt engineering* con JSON response mode (durante la generación de SQL):** 12 *templates* modulares con reglas estrictas que limitan la generación exclusivamente a sentencias `SELECT` con placeholders obligatorios, combinado con JSON response mode que restringe la salida a un formato estructurado y dificulta que instrucciones inyectadas generen SQL arbitrario.

4. **SQL Guard (entre generación y ejecución):** validación del SQL generado con una lista de 14 keywords prohibidos (`DROP`, `DELETE`, `INSERT`, `UPDATE`, `TRUNCATE`, `ALTER`, `CREATE`, entre otros) y verificación de estructura válida.

5. **Ejecución parametrizada (durante la ejecución):** uso de `text()` de SQLAlchemy [42] con bind parameters, sin concatenación de strings, previniendo inyección de SQL incluso si el modelo genera contenido malicioso.

6. **Eliminación de IDs internos (después de la ejecución):** los identificadores internos de la base de datos se remueven de los resultados antes de enviarlos al

modelo para generar la respuesta, evitando la exposición de información sensible del modelo de datos.

7. **Reintentos con backoff exponencial (transversal):** si la generación de SQL o la ejecución fallan, el sistema reintenta hasta 3 veces con backoff, proporcionando resiliencia ante errores transitorios de OpenAI.

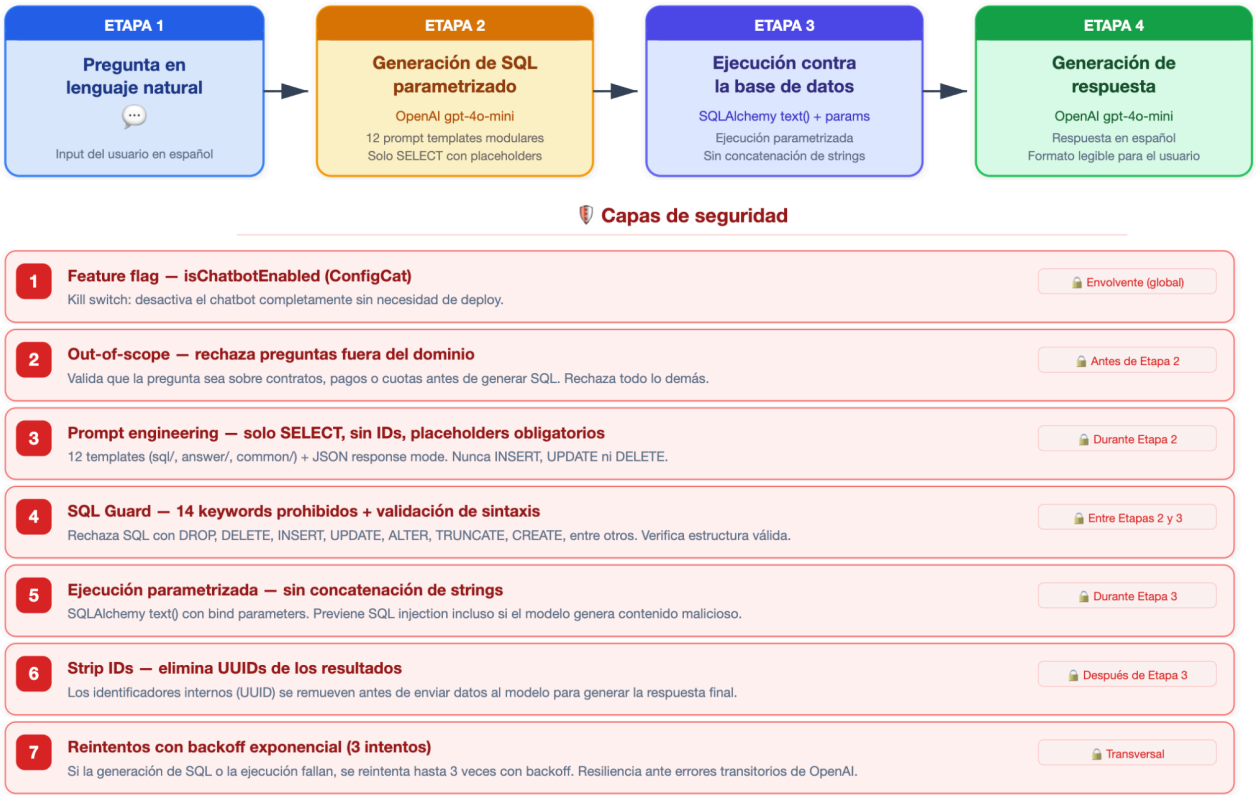


Figura 16. Pipeline del chatbot

Alternativas consideradas:

- Consultas predefinidas en un *dashboard* (menos flexible)
- Acceso directo a la base de datos para usuarios técnicos (riesgo de seguridad, no aplica para usuarios no técnicos).

Consecuencias: El *chatbot* permite realizar consultas analíticas sobre los datos del negocio sin que estas deban estar predefinidas en el sistema ni codificadas manualmente, ya que el modelo genera dinámicamente las sentencias SQL necesarias a partir de instrucciones en lenguaje natural, incluyendo filtros, agregaciones, joins y otras operaciones de consulta. Esto aporta flexibilidad para responder preguntas no anticipadas durante el diseño. Como contrapartida, cada interacción tiene un costo asociado al uso del modelo de OpenAI, y la generación automática de SQL a partir de lenguaje natural introduce un riesgo inherente de seguridad, mitigado mediante las siete capas definidas en la decisión. La funcionalidad se controla mediante un *feature flag*, lo que permite desactivarla de forma inmediata si se detecta un comportamiento no deseado, sin afectar al resto del sistema.

RNFs asociados: RNF-03 (Seguridad — múltiples capas de protección), RNF-04 (Mantenibilidad — controlado por *feature flag*).

ADR-08: Migraciones SQL manuales con control de aplicación

Contexto: El esquema de base de datos evolucionó a lo largo de 15 *sprints*. Se necesita un mecanismo de migraciones que sea reproducible y se ejecute automáticamente en cada despliegue.

Decisión: Se implementó un sistema *custom* de migraciones basado en scripts SQL versionados con convención de nombres cronológicos (`YYYYMMDD_descripcion.sql`), un script bash (`migrate.sh`) que los aplica en orden, y una tabla de control (`schema_migrations`) que registra qué scripts fueron aplicados.

Alternativas consideradas:

- Alembic (herramienta estándar para SQLAlchemy mayor complejidad de setup).
- Su autogeneración de migraciones presenta limitaciones documentadas por el propio proyecto para cambios complejos como renombrados de columnas,

cambios de tipo o modificaciones de constraints, que requieren intervención manual del desarrollador) migraciones manuales sin control (riesgo de inconsistencias entre entornos).

Consecuencias:

- Control total sobre el SQL ejecutado.
- Idempotencia (el script puede re-ejecutarse sin riesgo).
- Ejecución automática al arrancar via Docker Compose (servicio `migrator`).
- 28 migraciones aplicadas a lo largo del proyecto.
- No hay mecanismo de *rollback* automático (las reversiones requieren scripts SQL manuales).

RNFs asociados: RNF-01 (Integridad de datos — evolución controlada del esquema), RNF-04 (Mantenibilidad — despliegues reproducibles).

7.4. Tecnologías

A continuación se presentan las tecnologías utilizadas en el proyecto, organizadas por capas. La selección de las tecnologías principales se realizó durante la etapa de validación técnica (ver Sección 4.1 Cronología de las etapas).

7.4.1. Frontend

Tecnología	Versión	Rol
React	19.1.0	Framework de interfaz de usuario (SPA)
TypeScript	5.8.3	Tipado estático sobre JavaScript

Vite	6.3.5	Bundler y servidor de desarrollo
Tailwind CSS	4.1.8	Framework de estilos *utility-first*
Axios	1.6.5	Cliente HTTP para comunicación con el *backend*
TanStack React Query	5.33.1	Gestión de *server state* con *caching*
React Router	6.22.3	Enrutamiento del lado del cliente
Recharts	3.6.0	Visualización de gráficos para el *dashboard*
react-hot-toast	2.5.2	Notificaciones visuales (toasts)
clsx + tailwind-merge	—	Composición condicional de clases CSS

Tabla 4: Tecnologías Frontend

7.4.2. Backend

Tecnología	**Versión**	**Rol**
Python	3.12	Lenguaje de programación del *backend*

FastAPI	—	Framework web ASGI para la API REST
SQLAlchemy	—	ORM para acceso a base de datos
Pydantic	v2	Validación de schemas de entrada/salida
PyMySQL	—	Driver de conexión a MySQL
py-bcu	—	Cliente para el *web service* SOAP del BCU
python-jose	—	Generación y validación de tokens JWT
Authlib	—	Flujo OAuth 2.0
OpenAI SDK	—	Integración con el modelo gpt-4o-mini
Resend	—	Envío de notificaciones por email
configcatclient	—	SDK de ConfigCat para *feature flags*
New Relic APM	—	Monitoreo de performance en producción
pytest	8.4.0	Framework de *testing*
flake8	7.0.0	Análisis estático de código Python

Black	24.4.2	Análisis estático de código Python

Tabla 5: Tecnologías Backend

7.4.3. Base de datos

Tecnología	**Versión**	**Rol**
MySQL	8.4.8	Motor de base de datos relacional

Tabla 6: Tecnologías Base de Datos

7.4.4. Infraestructura y despliegue

Tecnología	**Rol**
AWS EC2	Servidor de aplicación (*backend* + base de datos)
AWS API Gateway	Proxy HTTPS y punto de entrada en producción
AWS S3	*Hosting* del *frontend* como sitio estático
AWS CloudFront	Content delivery network (CDN) para el *frontend*
AWS RDS	Base de datos

Docker Compose	Orquestación de servicios en el servidor (*backend*, MySQL, migrator)
AWS Lambda	Ejecución *serverless* de funciones
Terraform	Infraestructura como código (repositorio separado)
GitHub Actions	*Pipelines* de integración continua (*tests* + *build*)

Tabla 7: Tecnologías Infraestructura

7.5. Detalle de la arquitectura

7.5.1. Arquitectura del backend

Como se ve a continuación en la Figura 17, el *backend* sigue un patrón de tres capas pragmático, organizado de la siguiente manera:

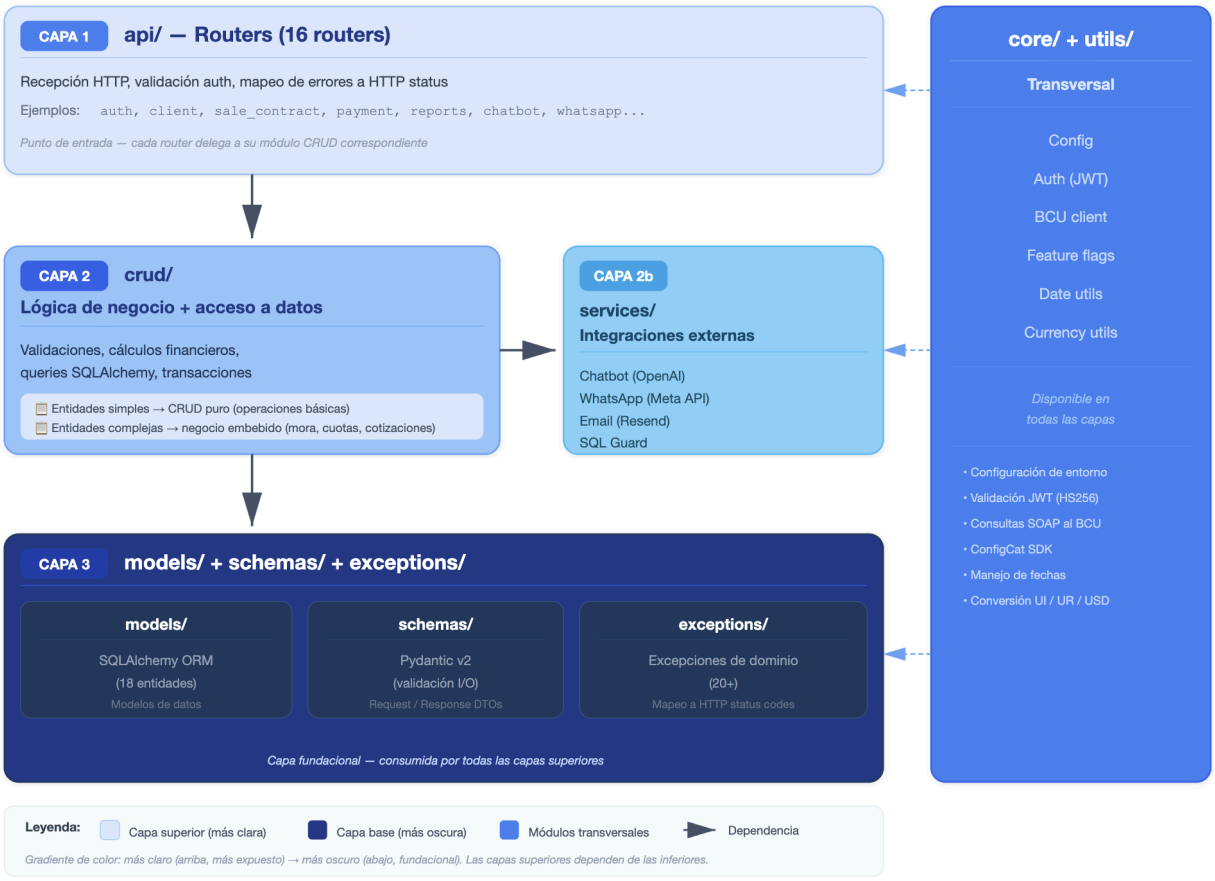


Figura 17. Arquitectura en capas del backend

Capa de presentación (`api/`): Contiene los routers de FastAPI organizados por entidad de dominio. Cada router recibe las peticiones HTTP, delega en la capa de lógica de negocio, y mapea las excepciones de dominio a códigos HTTP apropiados. Se registran 16 routers bajo el prefijo `/api/v1`, cubriendo: autenticación, clientes, productos (terrenos), contratos de venta, pagos, cotizaciones BCU, reportes (con sub-módulo para cada tipo de reporte), *dashboard*, *chatbot*, WhatsApp, cuentas bancarias, configuración del sistema, *feature flags* e invitaciones.

Capa de lógica de negocio y acceso a datos (`crud/` y `services/`): Para entidades con lógica simple (clientes, productos, empresas, etapas), los módulos CRUD combinan acceso a datos y validaciones básicas. Para entidades complejas (contratos de venta y pagos), los módulos CRUD contienen también la lógica de negocio:

validaciones de reglas financieras, cálculos de mora, generación de cuotas, recálculo en cascada de pagos posteriores al ajustar uno previo, y orquestación de transacciones. La carpeta `services/` se reserva exclusivamente para integraciones con servicios externos: el *pipeline* del *chatbot* con OpenAI (7 archivos que implementan las etapas de generación de SQL, ejecución, generación de respuesta y protección), el servicio del *bot* de WhatsApp (máquina de estados conversacional + cliente HTTP para la API de Meta), el servicio de envío de emails (Resend), y el servicio de consulta de información para el *bot* (`ClientInfoService`).

Capa de modelos y validación (`models/`, `schemas/`, `exceptions/`): Los modelos SQLAlchemy definen el esquema de base de datos con 18 entidades. Los schemas Pydantic v2 definen los contratos de validación para cada endpoint, con `from_attributes=True` para mapear directamente desde los modelos ORM. Las excepciones de dominio (20+) están organizadas por entidad y contienen atributos `code` y `message` descriptivos.

Capa transversal (*`core/`*, `utils/`): Incluye la configuración centralizada, la lógica de autenticación y autorización, el cliente del BCU, el servicio de *feature flags*, y utilidades para manejo de decimales, formateo de monedas y fechas.

Inyección de dependencias: Se utiliza el sistema `Depends()` de FastAPI para tres cross-cutting concerns: la sesión de base de datos (`get_db`), la autenticación (`get_current_user`, `get_admin_user`), y el servicio de *feature flags*. Los módulos CRUD y servicios se importan directamente como funciones, sin inyección formal.

Manejo de errores: Las excepciones de dominio se lanzan en la capa CRUD/services y se capturan en la capa de routers, donde se convierten a `HTTPException` con el código de estado HTTP apropiado. Existe un handler global para errores de validación de Pydantic (`RequestValidationError`) que normaliza la respuesta de errores de validación de entrada.

7.5.2. Modelo de datos

El modelo de datos del sistema está compuesto por 18 entidades organizadas en cuatro grupos funcionales. A continuación se describen las entidades principales y sus relaciones que se ilustran en la Figura 18.

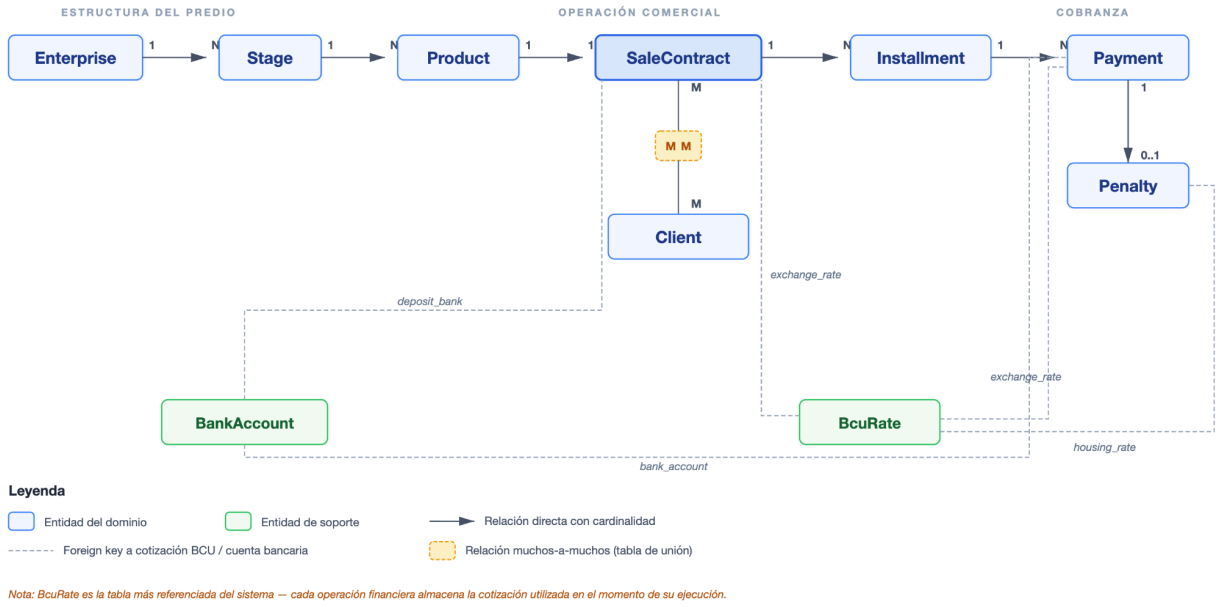


Figura 18. Modelo de datos simplificado

Nota: El modelo entidad-relación (MER) completo se incluye en el anexo 10 Modelo Entidad-Relación / MER.

Dominio del negocio: El núcleo del modelo sigue la cadena: Una empresa tiene múltiples etapas (sectores del predio), cada etapa tiene múltiples productos (terrenos identificados por padrón catastral), y cada producto puede tener a lo sumo un contrato de venta activo (garantizado por el constraint `UNIQUE` en `active_product_id`). La relación entre contratos y compradores es muchos-a-muchos (un contrato puede tener hasta cuatro compradores, y un comprador puede estar asociado a múltiples contratos) mediante la tabla de unión `sale_contract_clients`. Cada contrato genera N cuotas, y cada cuota puede tener N pagos (para soportar pagos parciales y ajustes).

Cada pago puede tener a lo sumo una penalidad asociada (mora), calculada cuando la fecha de pago supera la fecha de vencimiento de la cuota.

Cotizaciones y tasas (`BcuRate`): Esta es la tabla más referenciada del sistema: almacena tanto cotizaciones de moneda (USD, UR, UI) como tasas de vivienda por moneda (USD_HOUSING, UR_HOUSING, UI_HOUSING). Es referenciada por: `SaleContract.exchange_rate_id` (tasa al momento de firma), `Payment.exchange_rate_id` (tasa al momento del pago), `Payment.commission_ui_rate_id` (tasa UI para comisión de red de cobranza), y `Penalty.housing_rate_id` (tasa de vivienda para cálculo de mora). Cada registro incluye el valor, la fecha de cierre del BCU, la fecha de extracción, y la respuesta cruda del *web service* (`raw_json`) como evidencia.

Usuarios y autenticación: El modelo `User` almacena los datos del usuario autenticado (email, nombre, foto de Google, flags `is_active` e `is_admin`, último *login*). El modelo `AllowedEmail` implementa la *allowlist* de invitaciones. `BankAccount` almacena las cuentas bancarias disponibles para el registro de pagos, con constraint de unicidad compuesto.

WhatsApp y notificaciones: El *bot* de WhatsApp utiliza cuatro tablas: `WhatsAppConversationState` (estado de la máquina de estados por usuario), `WhatsAppClientAssociation` (vinculación entre número de WhatsApp y cliente/padrón), `WhatsAppMessageLog` (deduplicación y registro de mensajes), `WhatsAppRateLimit` (control de frecuencia), y `WhatsAppBotMetric` (métricas de uso). Las notificaciones por email se registran en `EmailNotification` con estado de envío (PENDING/SENT/FAILED) y datos de contexto.

Configuración: `SystemConfig` almacena parámetros globales del sistema en formato clave-valor (por ejemplo, la comisión de redes de cobranza expresada en UI).

7.5.3. Arquitectura del frontend

El *frontend* es una SPA desarrollada en React 19 con TypeScript, construida con Vite como bundler.

Patrón de componentes: Atomic Design: El sistema de componentes sigue Atomic Design con tres niveles, documentado internamente en el repositorio:

- **Átomos** (13 componentes): elementos UI base sin lógica de dominio — `Button` (5 variantes, 3 tamaños, estado de carga), `Input` (con label, error display, accesibilidad vía `aria-invalid`), `Select` (con búsqueda), `Modal` (portal con overlay, cierre con Escape), `Table` (genérica con columnas tipadas), `Card`, `Checkbox`, `TextArea`, `Alert`, `Icon`, `InfoField`, `Skeleton`, `StatCard`.
- **Moléculas** (28 componentes): composiciones de átomos con lógica de formularios y modales, drawers para creación de entidades (`CreateSaleDrawer`, `CreateClientDrawer`), modales de pago (`PaymentModal`, `PaymentAdjustModal`), secciones de detalle (`ContractDetailSection`, `InstallmentsSection`, `FinancialSummary`), filtros (`SaleSearchSidebar`, `ReportFilters`), y skeletons de carga.
- **Organismos** (14 componentes): secciones autónomas de página que consumen datos del *backend*, *layout* (`Topbar`, `Sidebar`), listas de entidades (`SaleList`, `ClientList`), componentes del *dashboard* (indicadores de negocio, gráficos con Recharts, tablas), tablas de reportes, y el widget del *chatbot*.

Sistema de estilos: Todo el diseño visual se construyó con Tailwind CSS sin utilizar librerías de componentes externas. Se definieron *design tokens* centralizados (`design-tokens.json`) que especifican la paleta de colores (azul marino como color primario, verde teal como acento), tipografía (Inter como fuente principal), espaciado y sombras. La utilidad `cn()` (basada en `clsx + tailwind-merge`) permite componer

clases condicionalmente sin conflictos. El sistema soporta modo oscuro mediante *toggle* manual con persistencia en `localStorage`.

Gestión de estado: Se utilizan dos estrategias complementarias: React Context para el estado global de autenticación (token JWT, usuario actual, sincronización entre tabs), y TanStack React Query para el *server state* (datos del *backend* con *caching*, revalidación automática y manejo de estados de carga/error).

Comunicación con el *backend*: La capa `api/` (16 archivos) centraliza todas las llamadas HTTP usando Axios con interceptores para: inyección automática del token JWT en cada request, y redirección a *login* ante respuestas 401. Cada archivo exporta funciones asíncronas tipadas que encapsulan las llamadas HTTP.

Manejo de formularios: Los formularios se implementan con un patrón *custom* basado en `useState`: estado del formulario como interface tipada, estado de errores como `Partial<Record<keyof RequestType, string>>`, validación síncrona antes del submit, y captura de errores de servidor. No se utiliza librería de formularios externa.

Enrutamiento: React Router v6 con rutas declarativas organizadas en dos niveles: rutas públicas (`/login, /auth/callback`) y rutas protegidas por `ProtectedRoute` que verifican la autenticación antes de renderizar el *layout* principal con *sidebar* y *topbar*.

7.5.4. Integraciones externas

Banco Central del Uruguay: El sistema obtiene cotizaciones (UI, UR, USD) y tasas de vivienda mediante la librería `py-bcu`, que consume el *web service* SOAP del BCU. Se implementó un mecanismo de *fallback* de hasta 7 días hacia atrás para días sin cotización (fines de semana, feriados), y un último recurso que consulta el último cierre disponible sin fecha específica. Las tasas de vivienda se obtienen mediante un script Python ejecutado como subproceso; dicho script extrae estas tasas desde archivos Excels públicos (debido a que el BCU, si bien publica estos archivos, no expone un

servicio web formal para obtener sus datos). Cada cotización obtenida se persiste en la tabla `bcu_rates` con la respuesta cruda del servicio.

Meta WhatsApp Cloud API: El *bot* de WhatsApp se conecta mediante la API v18.0 de Meta. Un endpoint webhook recibe los mensajes entrantes (verificación inicial con token, deduplicación por `message_id`), y un cliente HTTP envía respuestas en cinco formatos: texto simple, botones interactivos y listas de opciones. El flujo conversacional se implementa como máquina de estados con 5 pasos principales (`START` → `SELECT_CONTRACT` → `VERIFY_CENSUS` → `MAIN_MENU` → `END`) y protecciones de *rate limiting* (10 mensajes/minuto), horario de servicio configurable, y *timeout* de conversación de 24 horas. En la Figura 19 se aprecian las máquinas de estado relevantes.

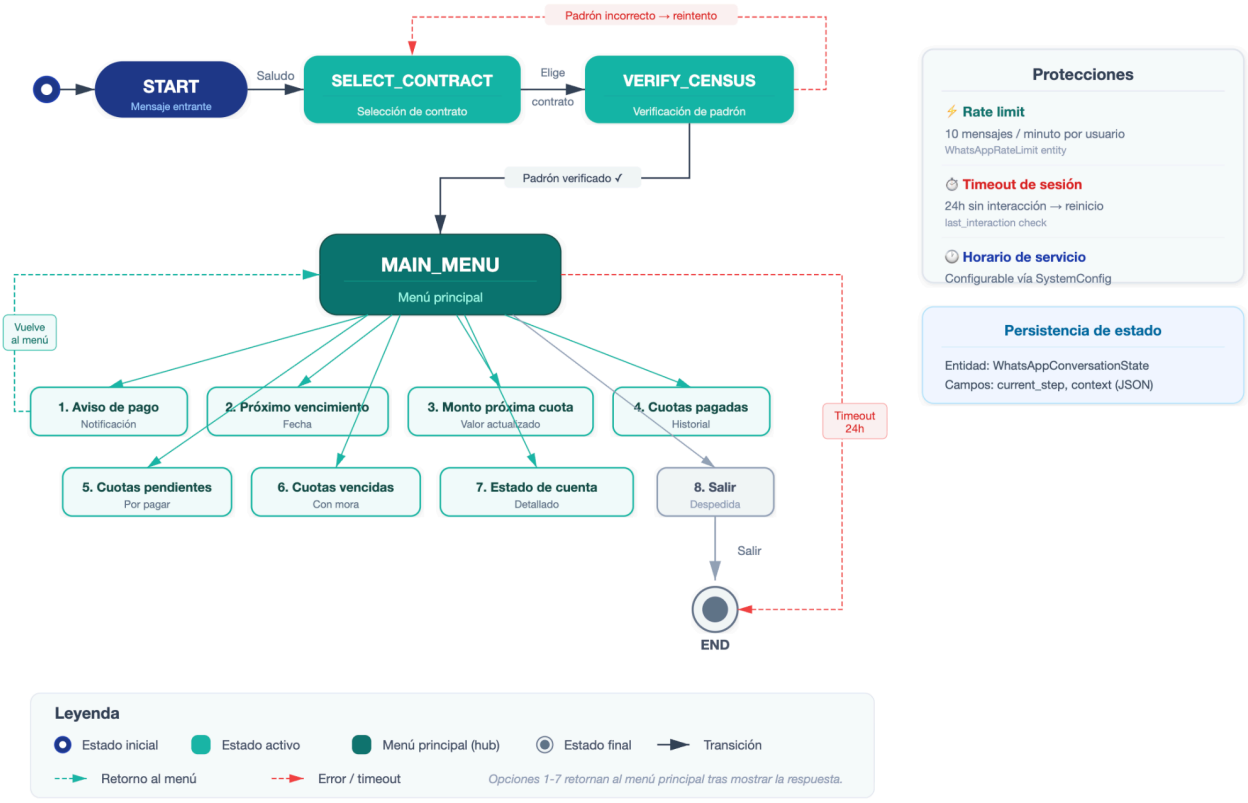


Figura 19. Máquinas de estados del bot de Whatsapp

OpenAI (*chatbot text-to-SQL*): El *chatbot* interno utiliza un *pipeline* de 4 pasos: generación de SQL parametrizado a partir de la pregunta en lenguaje natural, ejecución controlada contra la base de datos, generación de respuesta interpretativa, y formateo de títulos de columnas. Los *prompt templates* están modularizados en 12 archivos markdown organizados en carpetas (`sql/`, `answer/`, `common/`). La seguridad se implementa en las capas detalladas en el ADR-07: *Chatbot* text-toSQL con OpenAI y *pipeline* de seguridad en capas.

ConfigCat: El SDK oficial se inicializa como *singleton* con `@lru_cache` y expone tres flags evaluados en *backend* y *frontend*. Los endpoints controlados por *feature flag* devuelven error si el flag está deshabilitado, y el *frontend* oculta los elementos de navegación correspondientes. (ver anexo 11 - Captura de configuración de *feature flags* con ConfigCat)

Resend: Servicio de envío de emails utilizado para notificaciones automáticas de mora. Los emails se construyen con templates HTML inline y se envían exclusivamente cuando un pago registra atraso, con protecciones de *cooldown* (7 días mínimo entre emails al mismo cliente) y registro del envío en base de datos.

7.5.5. Infraestructura de despliegue

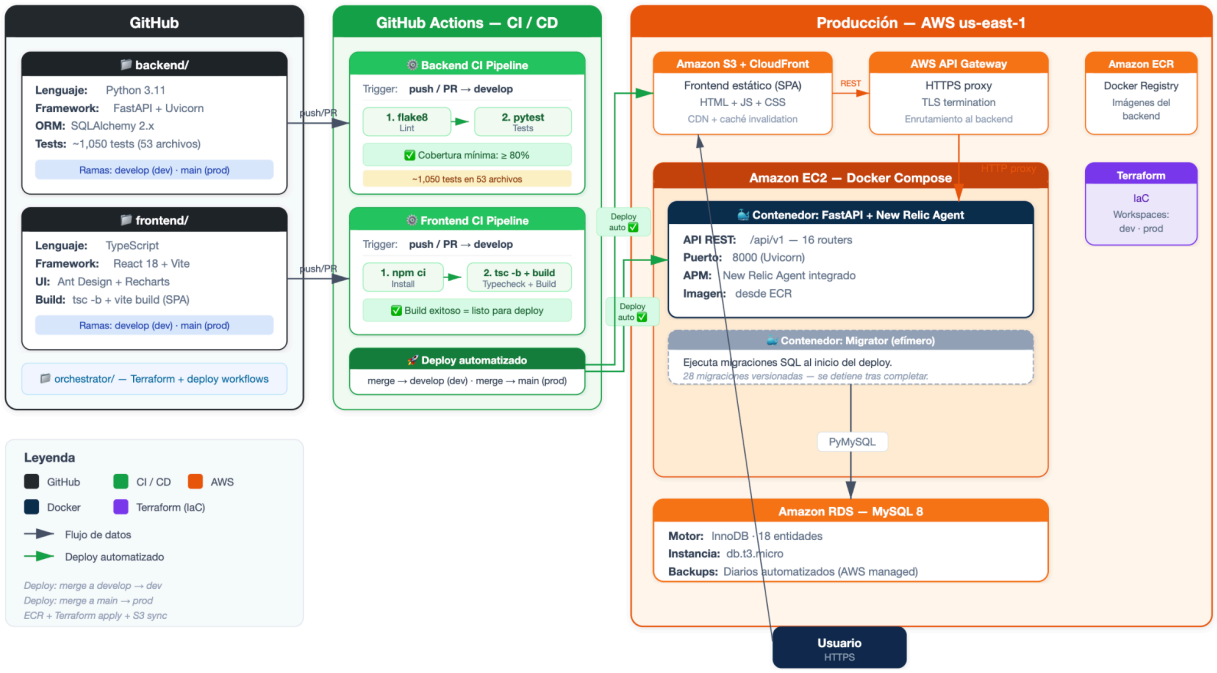


Figura 20. Infraestructura de Despliegue

Entornos: Como se visualiza en la Figura 20, el sistema opera en tres entornos: desarrollo local (Docker Compose con servicios de backend, MySQL y migrador, Vite dev server para el *frontend*), dev en AWS (mismo stack que producción, desplegado automáticamente al mergear a *develop*) y producción en AWS (EC2 con Docker Compose para el *backend*, RDS para la base de datos con backups diarios, S3 para el *frontend* estático, CloudFront como CDN y API Gateway como proxy HTTPS). El entorno de desarrollo en AWS permite validar los cambios en un ambiente equivalente al productivo antes de promoverlos a *main*.

***Pipeline* de CI:** Se configuraron dos *workflows* de GitHub Actions, uno por repositorio, activados en push y PR a la rama **develop**. El *pipeline* del *backend* ejecuta: *lint* con **flake8** (errores de sintaxis bloquean el *build*, *warnings* se reportan sin bloquear) y *tests* con **pytest** (falla si la cobertura cae por debajo del 80%). El *pipeline* del *frontend* ejecuta: instalación de dependencias y *build* completo (**tsc -b** + **vite build**), verificando

la compilación de TypeScript como condición de integración (ver anexos 12 - Captura de *Pipeline* de CI del *backend* y 13 - Captura de *Pipeline* de CI del *frontend*).

Despliegue: El despliegue se automatizó mediante dos *workflows* de GitHub Actions definidos en el repositorio orquestador: **dev-deploy.yml** (disparado al mergear a *develop*) y **prod-deploy.yml** (disparado al mergear a *main*). En ambos casos, el *pipeline* construye y publica la imagen Docker del *backend* en ECR, ejecuta terraform apply contra el workspace correspondiente (dev o prod), y ejecuta el script de despliegue del *frontend* que inyecta la URL de la API en el *build*, sube los artefactos a S3 e invalida la caché de CloudFront. Adicionalmente, se conservan scripts de despliegue manual (**backend-deploy.ps1** y **frontend-deploy.ps1**) como mecanismo alternativo, aunque la forma principal y preferida de desplegar es a través de los *pipelines* automatizados.

Base de datos en producción: MySQL 8.4.8 corre en una instancia RDS de AWS con backups diarios automatizados.

7.5.6. Análisis y Discusión de Costos de AWS

La elección de la infraestructura en Amazon Web Services (AWS), si bien brindó escalabilidad y robustez (discutida en las secciones 7.2.6 Resiliencia (RNF-06) y 7.2.7 Performance (RNF-07)), tuvo una consideración activa sobre la gestión de los costos operativos, ya que fue necesario tener acceso pago ya que el nivel de acceso que disponibilizaba la universidad tenía restricciones que imposibilitaban el uso de Terraform. El diseño arquitectónico se basó en los siguientes servicios con sus principales implicancias económicas:

- Amazon EC2 (Instancias Virtuales): El *backend* se desplegó en una instancia de tipo t3.micro, seleccionada por su balance entre rendimiento y el nivel de consumo gratuito.

- Amazon RDS (MySQL): Para la base de datos relacional, se optó por una instancia de RDS db.t3.micro. Aunque RDS simplifica la gestión, su costo tiende a ser mayor al de una base de datos autogestionada en EC2. Esta decisión se tomó priorizando la mantenibilidad (sección 7.2.4 Mantenibilidad (RNF-04)), justificando el costo adicional.

- Amazon S3 (Almacenamiento y *Hosting*): El *frontend* (sitio estático) y los assets se alojaron en S3. Este servicio representa uno de los costos más bajos y predecibles de la arquitectura, cobrando por almacenamiento y peticiones, lo cual es ideal para un *hosting* estático.

El costo mensual inicialmente proyectado para la arquitectura base (*Backend* en EC2, DB en RDS, *Frontend* en S3/CloudFront) se estimó en USD 100 mensuales, asumiendo un nivel de tráfico bajo-moderado post-lanzamiento. Si bien este valor se encontraba dentro del presupuesto máximo de USD 200 fijado por el cliente (comentado en la sección 3.3 Alcance del proyecto), el equipo identificó una oportunidad de optimización para maximizar la rentabilidad operativa de la solución. Por ello, se tomó la iniciativa de analizar la infraestructura en busca de alternativas más eficientes, logrando implementar mejoras que redujeron el costo operativo final.

En consecuencia, se identificó que el mayor costo estaba en la base de datos RDS, cuyos gastos son debidos al tiempo que está encendida y disponible para responder a consultas. Se evaluaron alternativas, como una base de datos *serverless* que cobra por consulta y no por tiempo de encendido, hasta que finalmente se optó (con el consentimiento del cliente) por la opción de mantener la base de datos encendida solo durante el horario de trabajo del cliente (de 8:00 a 22:00 UTC-3).

Adicionalmente, se descubrió que la versión de MySQL usada en la instancia RDS podría aumentar el costo si se encontraba deprecada por ser demasiado antigua, por lo que se aseguró que se estuviese en una versión mayor que tuviera soporte extendido (la versión escogida 8.4.8 tiene soporte hasta el año 2029).

Con estas mejoras, se notó una reducción significativa en los costos, dejando el costo de ambos ambientes (dev/prod) en un total de entre 40 y 50 USD mensuales.

7.6. Desarrollo

7.6.1. Prácticas de desarrollo

Flujo de trabajo con Git: La estrategia de ramificación e integración continua se rige por el modelo Gitflow, cuyo diseño y justificación arquitectónica se detallan en la sección 7.2.4. Mantenibilidad (RNF-04). Operativamente, el equipo estandarizó la integración de código mediante el uso estricto de una plantilla para la creación de PRs (ver anexo 8 - Template de *Pull Request*). Este formato exige que el desarrollador documente explícitamente el contexto del cambio (resumen de la funcionalidad o defecto), la solución técnica implementada y, de manera crítica, un bloque estructurado de instrucciones de prueba ("*Testing Steps for Reviewers*" en el *template*). Esta práctica asegura que cada modificación propuesta esté contextualizada y cuente con un mecanismo de verificación reproducible antes de ser sometida a la revisión por pares.

***Testing*:** El *backend* cuenta con un suite de aproximadamente 1,050 *tests* organizados en 53 archivos, cubriendo todos los dominios del sistema. Los *tests* utilizan dos estrategias de aislamiento: `MagicMock` [43] con *fake sessions* para la mayoría de los *tests* (simulando la sesión de base de datos y las respuestas de queries), y SQLite in-memory para *tests* que requieren operaciones reales de base de datos (WhatsApp, BCU). Las dependencias externas (BCU, WhatsApp API, OpenAI) se aíslan mediante `monkeypatch` de pytest. El `conftest.py` define *factories* para las entidades principales del dominio y fixtures de autenticación para tres niveles de acceso (usuario regular, administrador, sin autenticación).

El *frontend* no cuenta con *tests* unitarios automatizados. Esta fue una decisión deliberada del equipo: los *tests* funcionales de interfaz son costosos de construir y resultan frágiles ante cambios estéticos (un ajuste visual menor puede romper decenas

de *tests* sin que exista un defecto real). Dado que el equipo contó con un cliente con buena disposición a las pruebas, se priorizó la verificación funcional manual mediante scripts de prueba que cubren los flujos principales del sistema (por ejemplo: dado un contrato en UI con cuota vencida, al registrar un pago parcial, verificar que el sistema calcule correctamente la mora, actualice el saldo y genere la notificación). La verificación automatizada del *frontend* se limitó a la compilación de TypeScript (`tsc-b`) en el *pipeline* de CI, que garantiza la consistencia de tipos, complementada con ESLint [44] y Prettier [45] para la calidad del código.

Calidad de código: El *backend* utiliza `flake8` como *linter* (integrado en CI) y `black` como formatter (uso local). El *frontend* utiliza ESLint con TypeScript-ESLint y Prettier (ambos de uso local). La verificación de tipos de TypeScript se ejecuta en CI como condición de *build*.

7.6.2. Evolución del sistema

La evolución técnica del sistema a lo largo de los quince *sprints* quedó materializada empíricamente en la ejecución de 28 migraciones de la base de datos. Esta evolución fue impulsada principalmente por el avance progresivo en la implementación de las funcionalidades definidas durante la fase de ingeniería de requerimientos (ver sección 6 Ingeniería de requerimientos), en la cual el equipo generó una visión amplia y estructurada de las necesidades del negocio.

Sobre esta base planificada, el esquema de datos evolucionó de forma iterativa para soportar la creciente complejidad de la operativa a medida que se construían los distintos módulos. Esto incluyó desde ajustes críticos en la precisión decimal para tolerar cálculos financieros multidivisa, hasta reestructuraciones estructurales más profundas, como la habilitación de múltiples compradores simultáneos por contrato mediante el uso de tablas intermedias. Posteriormente, las migraciones acompañaron la expansión del alcance del sistema, incorporando la persistencia de cuentas

bancarias, el sistema de invitaciones para el control de acceso y las tablas necesarias para operar la máquina de estados del *bot* de WhatsApp.

Un aspecto fundamental de esta evolución técnica consistió en el esfuerzo por resolver la ambigüedad terminológica detectada durante el relevamiento, alineando el código fuente con el modelo mental del negocio. Dado que el cliente utilizaba ciertos conceptos de forma intercambiable (por ejemplo, "terreno", "padrón" y "producto"), el equipo acordó una estandarización semántica interna para la implementación. Desde el diseño arquitectónico se definió que, a nivel de *backend*, las entidades adoptarían nombres en inglés con un enfoque orientado a la generalización de la plataforma (por ejemplo, modelando los terrenos como *products*), tal como se refleja formalmente en el Modelo Entidad-Relación (ver sección 7.5.2. Modelo de datos).En contraste, la interfaz de usuario (*frontend*) fue diseñada utilizando estrictamente la jerga en español con la que el cliente opera diariamente.

Finalmente, esta dualidad terminológica presentó un desafío arquitectónico particular durante la integración del *chatbot* interno asistido por inteligencia artificial (6.2.2. Funcionalidades extendidas (*feature flags*) (RF-11)). Para que el asistente pudiera traducir correctamente las consultas formuladas en español por los usuarios hacia consultas SQL estructuradas sobre la base de datos, fue indispensable explicitar este vínculo conceptual mediante técnicas de *prompt engineering*. Se instruyó algorítmicamente al modelo para que fuera capaz de mapear la jerga operativa ingresada desde el *frontend* ("padrón", "terreno") con las entidades generalizadas del *backend* (*products*). Adicionalmente, el modelo fue configurado con instrucciones estrictas para que la cláusula `SELECT` de las consultas resultantes devolviera siempre identificadores de negocio legibles para los usuarios administrativos (como el número de padrón o el documento del comprador), omitiendo los identificadores internos transaccionales de la base de datos (como los UUIDs), los cuales carecen de valor para el análisis humano.

7.6.3. Uso de herramientas de Inteligencia Artificial

Durante el ciclo de vida del proyecto, se integraron de forma explícita y sistemática diversas herramientas de Inteligencia Artificial (IA) generativa como apoyo al flujo de trabajo del equipo. Su incorporación tuvo como objetivo optimizar los tiempos de ejecución, ampliar la capacidad de análisis y acelerar las iteraciones del desarrollo. Su uso se repartió en las siguientes cuatro áreas principales de la ingeniería del software:

1. Ideación y análisis: En las etapas tempranas y de diseño, se utilizó ChatGPT [46] como un "sparring técnico". La herramienta se empleó para contrastar enfoques arquitectónicos, explorar alternativas para el modelo de datos y evaluar tecnologías antes de proceder con su implementación formal. Esto permitió estructurar conceptos abstractos y descartar soluciones inviables con mayor rapidez.

2. Desarrollo y programación: Para la construcción del software, el equipo se apoyó en herramientas de inteligencia artificial generativa, tanto en plataformas conversacionales como ChatGPT como en asistentes integrados directamente en el entorno de desarrollo Visual Studio Code (Claude Code y GitHub Copilot Chat, este último con acceso a múltiples modelos). Su uso se concentró principalmente en la asistencia durante la programación: generación de código, resolución de errores y, especialmente en la escritura de pruebas automatizadas, estas herramientas resultaron clave para lograr una cobertura significativa de los diversos módulos del sistema.

A nivel metodológico, la adopción de estas herramientas operó bajo la premisa de que la IA funciona como asistente y no como autoridad. Todo el código generado fue verificado por los desarrolladores y sometido al flujo formal del proyecto. Durante este proceso de control de calidad, se integraron las capacidades de IA de GitHub Copilot para asistir en la revisión de PRs. Esta herramienta no solo permitió generar resúmenes automáticos de los cambios introducidos, sino que también formuló sugerencias directas de modificación y optimización sobre el código evaluado, facilitando la corrección y la detección temprana de anomalías estructurales. No

obstante, la aceptación de dichas sugerencias y la aprobación final en la revisión por pares siempre recayó sobre el análisis y criterio humano.

3. Redacción y revisión de documentación: Para la elaboración de la documentación técnica y del presente documento monográfico, se utilizaron ChatGPT, Gemini, NotebookLM, Claude Code y GitHub Copilot. Las tres primeras herramientas fueron utilizadas como apoyo para revisar redacción, mejorar claridad, sugerir alternativas de organización y verificar consistencia entre secciones. En particular, NotebookLM facilitó el análisis cruzado del documento para asegurar que no existieran contradicciones entre los requerimientos, la arquitectura y las conclusiones.

Por su parte, Claude Code y GitHub Copilot Chat fueron usados para apoyar la elaboración de secciones técnicas específicas de la tesis. Al operar integrados en el entorno de desarrollo, estas herramientas tuvieron la capacidad de acceder al repositorio, interpretar el código fuente implementado y brindar asistencia de descripciones sobre el funcionamiento de los componentes y las decisiones arquitectónicas plasmadas en el software.

4. Generación de imágenes: Para la creación de imágenes ilustrativas y recursos visuales complementarios requeridos durante el proyecto, se utilizó la herramienta de generación por IA Nano Banana Pro.

7.7. Conclusiones y lecciones aprendidas

La arquitectura y el diseño de DATIUM se consolidaron a partir de decisiones pragmáticas orientadas a satisfacer los atributos de calidad priorizados, equilibrando las exigencias académicas del proyecto con las necesidades operativas de un entorno financiero real. En este contexto, el uso de Architecture Decision Records (ADR) resultó una práctica particularmente valiosa, ya que permitió profesionalizar el proceso de toma de decisiones arquitectónicas y documentar de forma explícita los criterios considerados en cada caso. Este enfoque facilitó la justificación estructurada de los

trade-offs evaluados a lo largo del proyecto y contribuyó a mantener trazabilidad sobre las decisiones adoptadas.

En primer lugar, la definición de la arquitectura del *backend* evidenció que no siempre resulta necesario ni conveniente adoptar patrones de diseño de forma estrictamente purista. En este caso, se decidió no implementar una arquitectura hexagonal completa y optar por un modelo pragmático en el que la lógica de negocio compleja, asociada principalmente a pagos y contratos, reside junto al acceso a datos dentro de la capa CRUD. Únicamente las integraciones externas fueron aisladas en una capa de servicios. Separar toda la lógica de negocio en servicios independientes habría requerido introducir niveles adicionales de abstracción sin aportar beneficios claros para un dominio relativamente acotado. En cambio, el enfoque adoptado permitió que el flujo completo de una operación pudiera leerse y analizarse dentro de un mismo archivo, facilitando la comprensión del sistema.

En segundo lugar, la adopción de *feature flags* como patrón arquitectónico resultó fundamental para la gestión de riesgos y el despliegue continuo. Este mecanismo permitió desarrollar, integrar y evaluar funcionalidades extendidas de carácter experimental o académico aislando su impacto sobre el sistema. De esta forma, fue posible garantizar que el núcleo operativo estable, ya utilizado por el cliente, no se viera comprometido por cambios en desarrollo.

Por otra parte, la integración de Inteligencia Artificial Generativa dentro del sistema, mediante un *chatbot* basado en el patrón *text-to-SQL*, evidenció que el uso de modelos de lenguaje en dominios corporativos introduce nuevos desafíos en materia de seguridad. La generación automática de consultas SQL a partir de lenguaje natural implica vulnerabilidades inherentes que no pueden mitigarse mediante una única barrera de protección. Como resultado, se diseñó un *pipeline* con múltiples capas de defensa independientes, siete en total, con el objetivo de reducir el riesgo de

inyecciones maliciosas o consultas incorrectas derivadas de posibles alucinaciones del modelo.

A nivel de infraestructura de datos, el sistema de migraciones SQL personalizado funcionó adecuadamente para el volumen y la complejidad del proyecto, permitiendo mantener un control preciso sobre la evolución del esquema de base de datos. No obstante, se identificó como una limitación arquitectónica la ausencia de un mecanismo automatizado de *rollback*, lo que implicó que cualquier reversión requiriera la ejecución de scripts manuales. En caso de que el volumen de cambios estructurales aumente en el futuro, la adopción de herramientas especializadas como Alembic representaría una evolución natural para este componente.

Finalmente, el proyecto también permitió validar la incorporación estratégica de herramientas de Inteligencia Artificial como apoyo al flujo de desarrollo. La experiencia demostró que estas herramientas pueden actuar como un acelerador cognitivo y un multiplicador de productividad, siempre que se utilicen como asistentes del proceso de desarrollo y no como sustitutos del criterio técnico humano.

Entre las prácticas identificadas para maximizar su valor de forma responsable se destacan las siguientes:

- **Aceleración de tareas repetitivas:** La generación de estructuras base de código, plantillas de pruebas unitarias y borradores de documentación permitió concentrar el esfuerzo del equipo en decisiones de arquitectura y en la formalización de reglas de negocio.
- **Exploración técnica acelerada:** La IA demostró ser especialmente útil como herramienta de contraste para evaluar alternativas de diseño, analizar enfoques posibles o prototipar soluciones preliminares antes de su implementación definitiva.

- **Uso de la IA como asistente, no como autoridad**: En un dominio financiero fue imprescindible mantener el control humano sobre todas las decisiones críticas. Ninguna sugerencia generada por herramientas de IA fue incorporada al sistema sin validación técnica, revisión de código y verificación funcional sobre casos reales.
- **Integración con los mecanismos de control de calidad:** Todo artefacto generado con apoyo de IA se sometió a los mismos procesos de validación que el resto del código del proyecto, incluyendo revisión por pares y ejecución de los *pipelines* automatizados.
- **Protección de la información:** Se estableció una política estricta de no utilizar datos productivos reales en las interacciones con modelos públicos, empleando únicamente datasets anonimizados o sintéticos para la generación de ejemplos y escenarios de prueba

8. Gestión del proyecto

8.1. Herramientas de gestión

La gestión del proyecto se apoyó en un conjunto de herramientas seleccionadas para facilitar el trabajo colaborativo, garantizar coherencia entre requerimientos y código, y asegurar calidad en el proceso de desarrollo y despliegue. A continuación se detallan las herramientas utilizadas.

Se comenzó utilizando Jira como herramienta central para la administración del backlog del proyecto. En esta plataforma se organizaron las épicas, historias de usuario y tareas técnicas, permitiendo visualizar el estado del trabajo, asignar responsables y mantener un registro histórico de cambios. Más tarde en el proyecto, dado que la herramienta no fue considerada sencilla de usar y presentaba errores en sus gráficas, se terminó cambiando a Linear, que desempeñó las mismas funciones pero dejando al equipo más satisfecho con su uso.

El código fuente fue gestionado mediante Git, utilizando GitHub como repositorio remoto. Esto permitió mantener el versionado completo del sistema, trabajar en ramas independientes y realizar revisiones formales mediante PRs antes de integrar cambios.

Se implementaron *pipelines* automáticos mediante GitHub Actions, que ejecutaban pruebas automáticas y validaciones antes de permitir la integración de cambios. Además, se configuraron despliegues automáticos a los entornos de desarrollo y producción, reduciendo riesgos asociados a despliegues manuales.

La infraestructura en la nube fue definida utilizando Terraform, bajo el paradigma de Infraestructura como Código. Esto permitió versionar la configuración de AWS junto con el código del sistema, garantizando reproducibilidad y consistencia entre entornos.

La comunicación interna del equipo combinó WhatsApp para comunicación asíncrona diaria;
Slack para discusiones técnicas; y Google Meet para reuniones sincrónicas y ceremonias formales. La comunicación con el cliente se realizó mediante mensajes por WhatsApp, correos, reuniones periódicas por videoconferencia y un documento compartido de seguimiento de *feedback*.

La documentación funcional y académica se gestionó en Google Drive, permitiendo trabajo colaborativo y control de versiones del documento monográfico.

8.2. Etapas y principales hitos del proyecto

El proyecto se desarrolló de forma incremental, estructurado en dos grandes etapas: investigación y desarrollo, las cuales marcaron la evolución desde el entendimiento inicial del dominio hasta la puesta en producción y expansión funcional del sistema.

8.2.1. Etapa 1: Investigación

En esta fase se concentró el esfuerzo en comprender el dominio del problema y delimitar el alcance funcional del sistema, el cual ya fue especificado en la sección 3.3 Alcance del proyecto.

Se realizaron entrevistas con el cliente con el objetivo de comprender la operativa actual, los procesos administrativos, las reglas de negocio y los problemas asociados al uso de planillas Excel y cálculos manuales.

A partir del relevamiento se realizó una sesión interna de brainstorming para estructurar la información obtenida. Los resultados fueron compartidos y validados con el cliente, lo que permitió definir requerimientos funcionales y no funcionales, así como modelar los flujos críticos del sistema y estructurar el backlog inicial en épicas y user stories.

Relacionado con la definición del stack tecnológico, durante esta etapa también se investigaron y seleccionaron las tecnologías que se utilizarían en el desarrollo del

sistema. Dado que el cliente no impuso restricciones sobre las mismas, el equipo contó con la libertad para tomar esta decisión considerando múltiples factores: la experiencia, familiaridad y preferencia previa de los integrantes, así como también las particularidades funcionales del software a implementar.

Para el *backend*, se optó por Python debido a su sintaxis clara y legible, lo que reduce la curva de aprendizaje y facilita la iteración sobre funcionalidades durante el desarrollo. Además, considerando que el sistema debía interactuar con archivos Excel provenientes de la operativa del cliente, se evaluó la disponibilidad de librerías adecuadas dentro de su ecosistema, el cual ofrece herramientas maduras para este tipo de procesamiento de datos. La elección de Python también estuvo influida por la intención de incorporar funcionalidades basadas en inteligencia artificial, ya que se trata de uno de los ecosistemas más consolidados para este tipo de integraciones, tanto por la disponibilidad de librerías como por la madurez de sus herramientas.

En cuanto a frameworks, FastAPI fue seleccionado para la construcción de APIs REST, dado que permite generar documentación automáticamente a partir del código y ofrece un alto rendimiento, comparable al de frameworks utilizados en entornos Node.js. Finalmente, se utilizó SQLAlchemy para el acceso a la base de datos relacional, ya que proporciona capacidades de *Object-Relational Mapping* (ORM) y herramientas robustas para la definición y gestión de esquemas.

Para el *frontend* se eligió React, principalmente porque una de los integrantes del equipo, Federica, contaba con experiencia previa en su uso. Esto permitió acelerar el desarrollo inicial y facilitar la transferencia de conocimiento al resto del equipo. Además, la sintaxis de React, basada en JavaScript XML (JSX), extiende el lenguaje JavaScript, que ya era conocido por todos los integrantes, lo que contribuyó a reducir la curva de aprendizaje. Como framework de estilos se adoptó Tailwind CSS, que permite aplicar estilos mediante clases utilitarias predefinidas directamente en los elementos HTML, facilitando la construcción rápida y consistente de interfaces.

8.2.2. Etapa 2: Desarrollo incremental y releases

La construcción del sistema se llevó a cabo de manera iterativa mediante *sprints* de dos semanas, incorporando refinamiento continuo del backlog y revisiones periódicas con el cliente.

Se implementó el *core* del sistema, incluyendo la gestión de contratos, cuotas y pagos, junto con la automatización de cálculos financieros. También se incorporaron mecanismos de control de acceso y registro de eventos operativos.

Con la infraestructura desplegada en AWS y los entornos listos, el sistema alcanzó un estado productivo y comenzó a ser utilizado activamente por el cliente. A partir de este punto se incrementó la frecuencia de *feedback* y se redujeron los ciclos de mejora, dado el uso real del sistema.

Luego del primer *release*, se incorporaron funcionalidades adicionales tales como reportes, *bot* de WhatsApp, *chatbot* para consultas en lenguaje natural y *dashboard* de visualización. Para controlar la introducción progresiva de estas funcionalidades se implementaron *feature flags*, permitiendo mantener la estabilidad del sistema en producción.

8.3. Aplicación de Scrum

Para la gestión del desarrollo se adoptó Scrum como marco de trabajo, con adaptaciones acordes a la realidad del equipo y del contexto académico. El objetivo no fue aplicar Scrum en su forma estrictamente teórica, sino utilizar sus principios fundamentales de trabajo iterativo, inspección, adaptación, y entrega incremental de valor.

Las principales adaptaciones fueron:

- Reducción de la frecuencia de *Sprint Reviews* formales de una vez por *Sprint* a una vez cada 2 *Sprints*, debido al contacto continuo con el cliente.

- Sustitución del *daily* presencial por un modelo híbrido síncrono/asíncrono, donde 2 veces por semana se tienen llamadas síncrona con todo el equipo y el resto de días se envían mensajes a un grupo de WhatsApp con todos los integrantes para informar de avances e impedimentos. Este fue debido a que no era factible reunirse diariamente debido a los horarios complejos y en conflicto de los integrantes que eran estudiantes y adicionalmente trabajan a tiempo completo; también había diferencia horaria con Federica que residió en Francia durante la mayoría del proyecto y este hecho complicaba aún más la coordinación síncrona.

La metodología, cuyo flujo se ve en la Figura 21, permitió estructurar el trabajo, mantener foco en entregables incrementales y generar espacios formales de mejora continua.

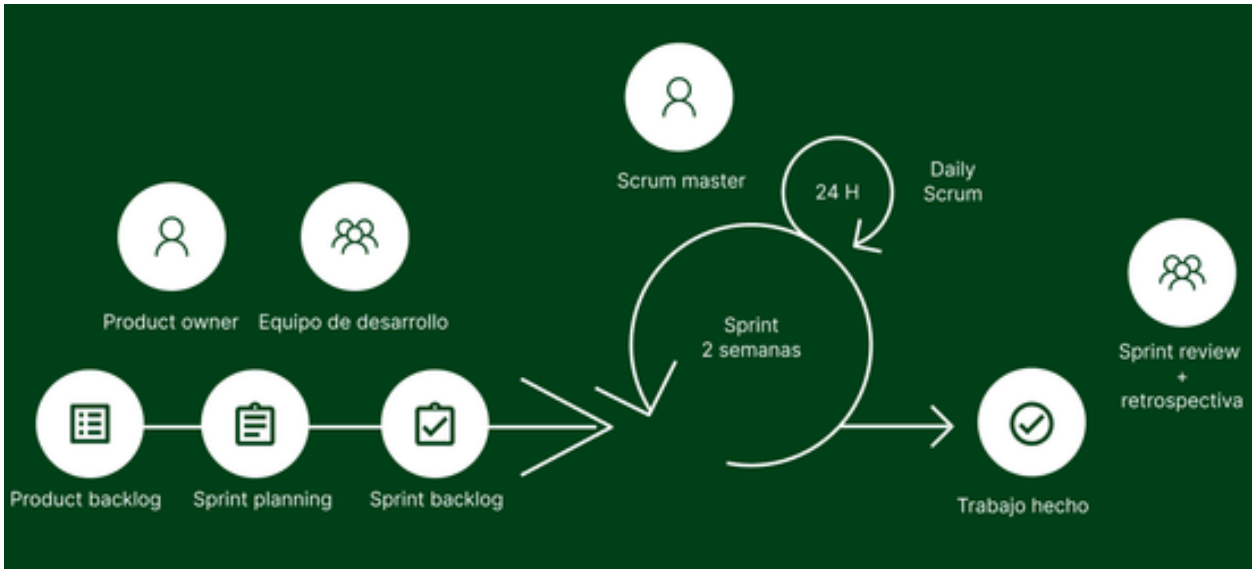


Figura 21. Proceso de Scrum

8.3.1. Planificación de las iteraciones

Las iteraciones tuvieron una duración fija (*sprints* de dos semanas). Al inicio de cada *sprint* se realizaba una instancia de *Sprint Planning* donde:

- El *Product Owner* presentaba las historias priorizadas.
- El equipo analizaba el alcance y complejidad.
- Se estimaban las historias mediante *Planning Poker* [15] utilizando *Story Points.*
- Se evaluaba el ingreso de tareas técnicas, *bugs* con severidad definida, *spikes* de investigación y correcciones.
- Se definía el *Sprint Backlog*.

Para que una historia pudiera ingresar al *sprint* debía cumplir con la DoR:

- Redacción en formato Gherkin [14].
- Estimación realizada.
- División en tareas técnicas.
- Claridad en criterios de aceptación.

Durante los primeros *sprints* no se contaba con una capacidad histórica definida. A partir del *sprint* 5 se comenzó a utilizar el promedio de puntos completados de los 3 *sprints* anteriores como referencia para planificar el alcance.

Cuando existía alta incertidumbre técnica se planificaban *spikes* exploratorios previos a la implementación definitiva.

8.3.2. Ejecución de las iteraciones

Durante el *sprint,* el trabajo se desarrollaba en ramas feature independientes de acuerdo a los lineamientos de Gitflow [36].

Se implementó un flujo basado en:

- Desarrollo incremental.
- PRs obligatorios.
- Revisión por otro integrante.
- Ejecución automática de *tests* y validaciones antes de integrar.

El seguimiento diario del avance se realizaba mediante:

- Dos reuniones síncronas semanales (*stand-ups*).
- Comunicación asíncrona diaria para actualizaciones rápidas.

Este esquema permitió mantener la alineación del equipo sin exigir sincronización completa en todos los días de la semana.

8.3.3. Evaluación de las iteraciones

Se presentaban los incrementos funcionales al cliente en la ceremonia de Sprint Review. Dado el contacto frecuente con el cliente, estas revisiones funcionaban más como validaciones estructuradas que como primer punto de exposición.

La técnica de retrospectiva fue cambiando; se empezó usando “el bueno, el malo, y el feo” (ver anexo 14 - Ejemplo de retrospectiva “el bueno, el malo, y el feo”) para identificar lo que salió bien, mal, o feo. El equipo sentía que esta técnica no era suficientemente introspectiva, por lo que se pasó a “tres cerditos” (ver anexo 15 - Ejemplo de retrospectiva “tres cerditos”), donde se evalúan los elementos frágiles y que

podrían desmoronarse en cualquier momento (“casa de paja”), los que funcionan pero tienen lugar para mejorar (“casa de palos”), y los que son fuertes y estables (“casa de ladrillos”). Si bien esta fue más útil, tampoco dejó al equipo satisfecho.

Finalmente, se terminó usando la dinámica de retrospectiva del “bote” que se muestra en la Figura 23, la cual permite identificar explícitamente:

- Qué impulsa al equipo.
- Qué lo retrasa.
- Qué lo pone en riesgo.
- Hacia dónde quiere dirigirse.

Esta retrospectiva fue la que más valor aportó a la evaluación del *sprint* concluido, por lo que fue la que se siguió usando hasta el final del proyecto.

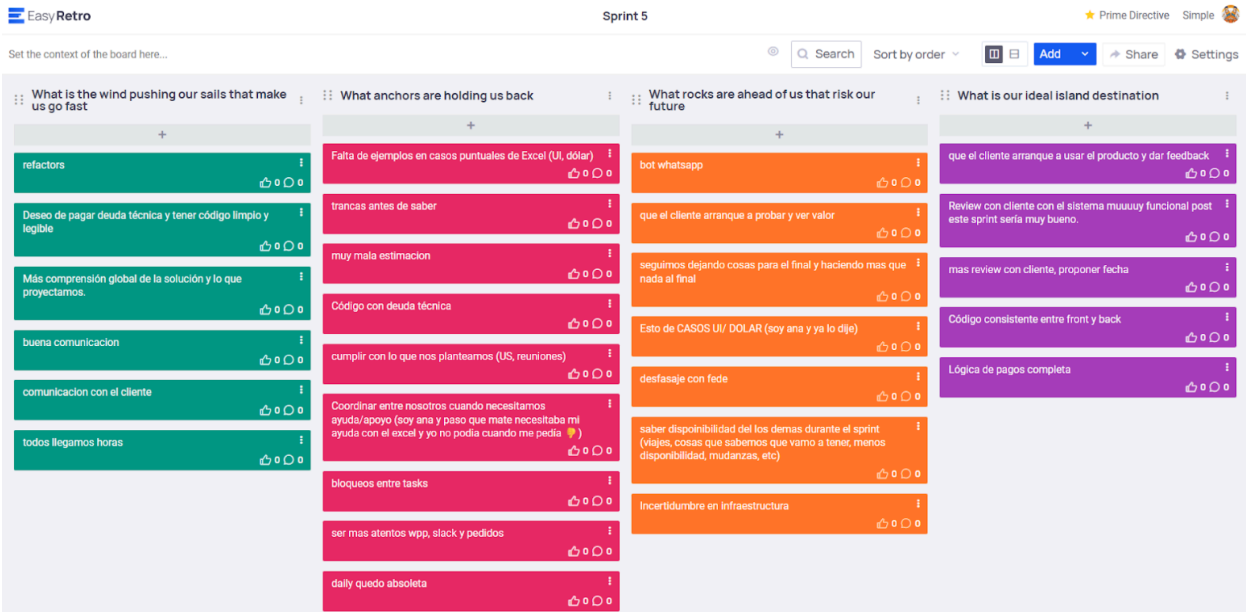


Figura 22. Retrospectiva del Sprint 5 usando la técnica del bote

A partir de estas instancias se implementaron mejoras concretas, como priorizar la revisión de PRs para evitar acumulación de trabajo bloqueado.

8.4. Gestión de la documentación académica

Se fueron creando documentos puntuales en el Drive compartido a lo largo del proyecto, en los cuales se detallaban:

- Decisiones para los ADRs.
- Resultados de investigaciones y *spikes.*
- Retrospectivas y accionables resultantes.
- Explicaciones técnicas sobre tecnologías.

Para la gestión del documento formal del proyecto se adoptaron las siguientes prácticas:

- Uso de la plantilla oficial establecida por la Facultad.
- Aplicación del formato IEEE para referencias bibliográficas.
- Control sistemático de cumplimiento de normas antes de cada entrega.
- Versionado del documento en repositorio compartido.
- Revisión cruzada entre integrantes antes de presentaciones parciales.

Además, se planificaron entregas intermedias correspondientes a las distintas revisiones académicas, permitiendo incorporar *feedback* del tutor y los revisores de manera incremental.

De esta forma, la documentación no se abordó como una tarea final aislada, sino como un proceso continuo integrado al desarrollo del proyecto.

8.5. Plan de *releases*

El plan de *releases* consistió en 2 *releases*: el primero, que estaba compuesto por las funcionalidades núcleo indispensables para el negocio; y el segundo, que traía los "nice-to-have" que agregaban valor al producto ya existente. En la sección 4.1 Cronología de las etapas se muestra la línea de tiempo para los *releases*, y en la sección 6.2 Funcionalidades se detallan las funcionalidades que se listan a continuación:

El *release* 1 incluía las siguientes funcionalidades:

- Gestión de clientes
- Gestión de terrenos y predios

- Gestión de ventas
- Gestión de recargos
- Gestión de pagos recibidos.
- Gestión de etapas.
- Despliegue de infraestructura

Como se puede ver, este *release* trató con las principales features que el cliente necesita para poder mantener las mismas capacidades que maneja en su operativa previa al producto. El *release* 1 culmina con el despliegue en un entorno productivo del sistema, y de este punto en el tiempo hasta el final del proyecto inclusive se mantiene funcional este entorno.

El *release* 2, por otra parte, tuvo estas funcionalidades:

- Autenticación de usuario
- Reportes
- Exportación a Excel y preview
- *Dashboard*
- *Feature Flags*
- *Bot* Whatsapp
- *Chatbot* Web

Este segundo *release* abarca el tiempo entre que se libera el *release* 1 y el fin del proyecto.

8.6. Métricas de gestión

Con el objetivo de evaluar la evolución del proceso de trabajo y la madurez del equipo a lo largo del proyecto, se definieron y analizaron métricas cuantitativas basadas en los datos registrados en Linear y el registro de horas de esfuerzo. Las métricas seleccionadas fueron:

- Velocidad promedio del equipo.
- Cumplimiento de alcance por *sprint.*
- Predictibilidad del *sprint.*
- Distribución de esfuerzo.

8.6.1. Velocidad del equipo

La velocidad se definió como la cantidad promedio de *story points* completados por *sprint* a lo largo del proyecto.

Considerando los 15 *sprints* ejecutados, el equipo completó un total de 251 *story points*, lo que arroja una velocidad promedio de aproximadamente 17 story points por *sprint.*

Durante los primeros *sprints* no se contaba con información histórica suficiente para definir una capacidad comprometida. A partir del *Sprint* 5 se comenzó a utilizar el promedio de *story points* completados como referencia para definir el alcance tentativo de los *sprints* siguientes, ajustando en función de la presencia de *spikes* técnicos u otras tareas que no generaban valor funcional directo al cliente.

8.6.2. Cumplimiento de alcance por sprint

En el anexo 16 - Gráfica de trabajo completado vs estimado por *sprint* se presenta la gráfica de trabajo completado versus alcance comprometido por *sprint*.

Durante los primeros cuatro *sprints* no se definió formalmente un alcance comprometido, debido a la ausencia de datos históricos que permitieran justificar una capacidad estimada. A partir del *Sprint* 5 se comenzó a establecer un alcance explícito.

Se observa que en varios *sprints* el trabajo completado fue inferior al alcance comprometido. A continuación se detallan las razones.

En determinados *sprints* quedaron historias técnicamente finalizadas pero pendientes de revisión de código, lo que impidió su cierre formal dentro del *sprint*. Esta situación fue particularmente evidente en los *Sprints* 1 y 5, donde no se registraron *story points* completados. Como medida correctiva, el equipo acordó priorizar la revisión de PRs como actividad crítica dentro del *sprint*, evitando que trabajo terminado quedara bloqueado por falta de validación.

En el *Sprint* 6, si bien se cerraron pendientes del *sprint* anterior, la cantidad de *story points* completados no aumentó proporcionalmente. Esto se debió a la incorporación de múltiples tareas técnicas y ajustes de infraestructura que no fueron estimados en *story points*, pero que consumieron capacidad efectiva del equipo. Luego del desfasaje observado en el *Sprint* 5, el equipo adoptó un enfoque más metódico, priorizando calidad y estabilidad sobre volumen de entrega.

En historias con alto grado de incertidumbre, incluso habiendo realizado *spikes* exploratorios previos, surgieron imprevistos que afectaron la planificación. Un ejemplo relevante fue la implementación del *bot* de WhatsApp en el *Sprint* 12, donde la integración con dependencias externas introdujo variabilidad que no fue anticipada.

En las primeras iteraciones resultó complejo estimar con precisión durante el *Sprint* Planning, debido a la adopción de tecnologías nuevas y a la falta de experiencia previa en el dominio específico del negocio. En algunos casos el alcance resultó excesivo, generando incumplimiento; en otros, el alcance fue insuficiente y se incorporaron historias adicionales durante el *sprint*, afectando la estabilidad del compromiso inicial. Esta variabilidad disminuyó progresivamente a medida que el equipo acumuló experiencia y datos históricos, evidenciando una mejora en la capacidad de planificación.

8.6.3. Predictibilidad por sprint

La predictibilidad se definió como el porcentaje de cumplimiento del alcance comprometido en cada *sprint*. En el anexo 17 - Gráfica de predictibilidad por *sprint* se presenta la evolución de la predictibilidad a lo largo de los *sprints*.

Se observa una alta variabilidad inicial, propia de un equipo que aún no contaba con métricas históricas de desempeño. A partir del *Sprint* 8 la predictibilidad comienza a estabilizarse en valores cercanos al 90–100%, lo que indica una mejora sustancial en la capacidad de estimación y planificación. Los descensos puntuales, como el anteriormente observado en el *Sprint* 12, se explican por la aparición de imprevistos técnicos asociados a historias de alta incertidumbre.

8.6.4. Distribución de esfuerzo

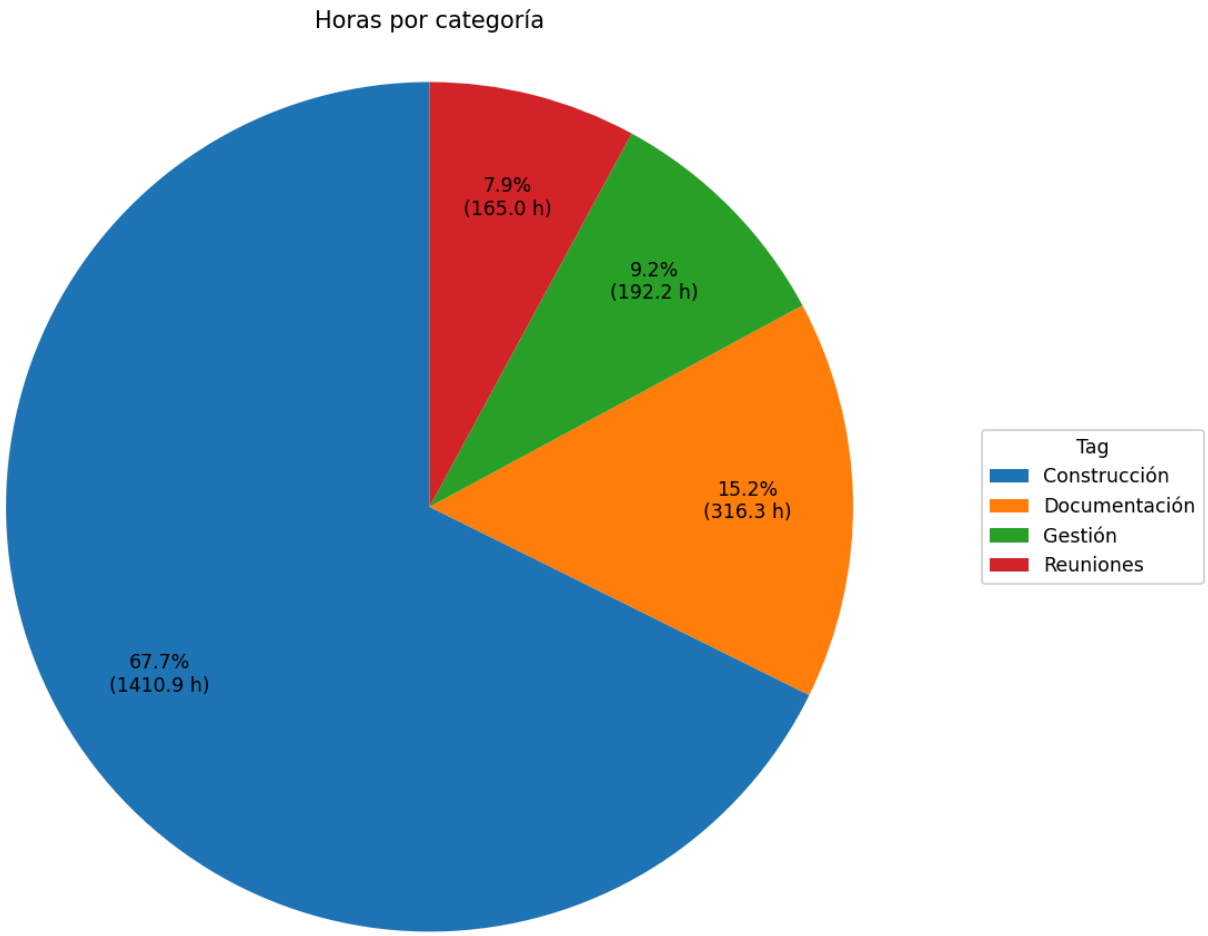


Figura 23. Pie chart de la distribución de horas durante el proyecto

Las horas dedicadas al proyecto fueron registradas de acuerdo a las siguientes categorías:

- Construcción: debido al marco de trabajo (Scrum) comprende programación, aseguramiento de la calidad (QA), gestión de la configuración del software (SCM).

- Documentación: tareas relacionadas a la creación y mantenimiento de la documentación del proyecto.
- Gestión: incluye las actividades relacionadas a la metodología Scrum y sus ceremonias.
- Reuniones: elicitación de requerimientos y procesos de trabajo del cliente.

En la Figura 22, se ve que la gran mayoría de las horas fueron dedicadas a tareas de desarrollo, lo cual es esperable dado el contexto del proyecto: en Scrum el desarrollo involucra un proceso riguroso de planificación y refinamiento del trabajo antes de poder arrancarlo, y a esto se le agrega un esfuerzo adicional para eliminar las inconsistencias heredadas por los procesos no maduros del cliente. A esto le sigue la documentación, que también es razonable debido a la necesidad de presentar un documento final que detalle el proyecto en su totalidad. Luego vienen las tareas de gestión de Scrum, que fueron actividades recurrentes y consistentes a lo largo del proyecto. Finalmente, se tienen las reuniones, que también fueron teniendo lugar a lo largo del ciclo de vida del proyecto y ocurrían frecuentemente en intervalos cortos.

8.7. Gestión de la comunicación

8.7.1. Comunicación interna del equipo

Se utilizaron distintos medios según el tipo de interacción:

- WhatsApp para comunicación rápida, *daily* scrum, y updates asincrónicos.
- Slack para discusiones técnicas más estructuradas y seguimiento de temas específicos.
- Google Meet principalmente para ceremonias de Scrum (*sprint* planning, *sprint* review, *sprint* retrospective), preparación de revisiones, o solucionar problemas puntuales.

8.7.2. Comunicación con interesados

La interacción con el cliente combinó comunicación cotidiana e instancias formales de validación. Se mantuvo comunicación frecuente vía WhatsApp y correo electrónico para consultas operativas, aclaraciones funcionales y coordinación. En cuanto a instancias formales, se realizaron *Sprint* Reviews cada 2 *sprints*; durante estas revisiones se presentaban los incrementos desarrollados, se validaban funcionalidades y se recogía *feedback*. Así, se verificaba que el equipo estaba alineado con las expectativas del cliente, se ajustaban prioridades del backlog, y se confirmaban hipótesis antes de seguir desarrollando.

Con el tutor académico se mantuvo un esquema de comunicación constante y estructurado. Se tuvo contacto frecuente vía WhatsApp para consultas puntuales, dudas metodológicas, y coordinación. Además, se tenía una reunión cada *sprint* dedicada a recibir *feedback* más elaborado, validar decisiones sobre la metodología, resolver dudas técnicas o académicas, e informar sobre eventos relevantes del proyecto.

8.8. Gestión de riesgos

Como parte de la planificación del proyecto, se realizó una identificación temprana de los principales riesgos técnicos y operativos, con el objetivo de definir estrategias de mitigación y planes de contingencia antes de su posible materialización.

Para cada riesgo se evaluó:

- Probabilidad de ocurrencia.
- Impacto en caso de materializarse.
- Magnitud resultante (en función de probabilidad e impacto).

- Estrategia de respuesta (mitigación y contingencia).

En el anexo 18 - Matriz de análisis de riesgo se presenta la matriz de riesgos elaborada durante la etapa inicial del proyecto.

8.8.1. Riesgo 1: Integración del bot de WhatsApp

Probabilidad: Probable
Impacto: Bajo
Magnitud: Riesgo moderado
Estrategia: Mitigar

Descripción:

Dado que la integración con WhatsApp depende de la aprobación de Meta

Cuando la aprobación es demorada

Entonces se retrasa la funcionalidad del *bot*

Disparador:
Demora prolongada en la aprobación del *bot* por parte de Meta o cambios en las APIs externas que impidan avanzar con la integración.

Acción preventiva:

- Realización de un *spike* técnico temprano para validar la viabilidad del flujo completo.
- Investigación previa sobre requisitos y proceso de aprobación de Meta, lo que permitió identificar la necesidad de contar con una landing page pública del producto como requisito de verificación de la cuenta de negocio.
- Contactar un experto con la integración con Whatsapp.

Acción correctiva:

- Reorganización del backlog y paralelización de tareas mientras se espera la aprobación.
- Alternativa definida: reemplazar el *bot* por notificaciones vía email (no fue necesario activarla).

Este riesgo fue considerado el más relevante debido a la dependencia de APIs externas y del proceso de aprobación de Meta, factores que estaban fuera del control directo del equipo. Se realizó un *spike* técnico temprano para validar la viabilidad del flujo completo de integración. Esto permitió reducir la incertidumbre técnica respecto al desarrollo, aunque no eliminó el riesgo asociado a las aprobaciones externas.

El riesgo efectivamente se materializó: las aprobaciones de Meta demoraron más de lo planificado. Se había definido como alternativa el reemplazo temporal del canal de notificaciones por correo electrónico. Sin embargo, no fue necesario activarla. En su lugar, se reorganizó el trabajo y se paralelizaron tareas. Mientras se aguardaban las aprobaciones, el equipo avanzó en otras funcionalidades del sistema. Una vez obtenidas las mismas, la integración se realizó sin impacto significativo en el cronograma.

8.8.2. Riesgo 2: Infraestructura en AWS

Probabilidad: Bastante probable
Impacto: Moderado
Magnitud: Riesgo alto
Estrategia: Mitigar

Descripción:

Dado que la configuración de la infraestructura es necesaria para el despliegue productivo del sistema en AWS

Cuando la configuración no es realizada

Entonces el sistema no puede ser desplegado

Disparador:
Dificultades técnicas reiteradas en el despliegue o configuración de la infraestructura que afecten el cronograma del *Release* 1.

Acción preventiva:

- Asignación de dedicación exclusiva al diseño y configuración de la infraestructura.
- Uso de infraestructura como código mediante Terraform para garantizar entornos reproducibles.
- Investigación técnica previa sobre arquitectura y despliegue.

Acción correctiva:

- Postergar la infraestructura productiva al *Release* 2 en caso de no llegar a tiempo al *Release* 1

La infraestructura en AWS representaba inicialmente una curva de aprendizaje significativa para el equipo, lo que podía afectar tiempos de entrega y estabilidad del sistema. Se decidió asignar dedicación específica a esta área y adoptar infraestructura como código mediante Terraform, con el objetivo de:

- Garantizar entornos reproducibles.

- Reducir errores manuales.
- Facilitar despliegues consistentes.

El riesgo no se materializó de forma crítica, por lo que la infraestructura pudo ser implementada dentro de los plazos previstos.

8.8.3. Riesgo 3: Sistematización de datos heredados de Excel

Probabilidad: Probable
Impacto: Alto
Magnitud: Riesgo moderado
Estrategia: Mitigar

Descripción:

Dado que existen inconsistencias en la documentación provista inicialmente

Cuando una inconsistencia significativa es detectada y corregida

Entonces se hace retrabajo

Disparador:
Detección de inconsistencias significativas en los datos históricos (por ejemplo: uso de una tasa de vivienda correspondiente a un mes incorrecto para el cálculo de mora, o propagación de un valor erróneo en una celda de la planilla de Excel)

- Implementación de validadores automáticos en cada paso de los cálculos intermedios.

Acción correctiva:

- Corrección manual asistida de datos inconsistentes.

- Ajustes iterativos en las reglas de negocio a medida que se detecten nuevos casos límite.

La calidad e inconsistencia de los datos heredados representaba un desafío importante, tanto desde el punto de vista técnico como de comprensión del dominio; estos datos incompletos, inconsistentes o con criterios históricos no documentados podían afectar la confiabilidad del sistema.

Se implementaron validadores automáticos en cada etapa de los cálculos intermedios, con el objetivo de:

- Detectar inconsistencias tempranamente.
- Prevenir la propagación de errores.
- Garantizar coherencia en la información procesada.

Si bien surgieron inconsistencias durante el desarrollo, el sistema de validaciones permitió controlarlas sin afectar la operación ni la confiabilidad del sistema final.

8.8.4. Riesgo 4: Cálculo de tasas y cotizaciones

Probabilidad: Muy probable
Impacto: Bajo
Magnitud: Riesgo moderado
Estrategia: Mitigar

Descripción:

Dado que la integración con el BCU depende de un servicio externo

Cuando el servicio no está disponible

Entonces no se pueden conseguir las tasas y cotizaciones de las monedas

Disparador:
Indisponibilidad temporal del servicio del BCU o cambios en el formato de respuesta de la API.

Acción preventiva:

- Construcción de un servicio propio intermedio para el consumo de tasas.
- Implementación de mecanismo de caché para reducir dependencia en tiempo real.

Acción correctiva:

- *Fallback* manual para carga de tasas en caso de indisponibilidad del servicio externo.
- Uso de última tasa válida almacenada hasta restauración del servicio.

El cálculo de tasas y cotizaciones dependía de fuentes externas, como las publicadas por el BCU, que podían cambiar, no estar disponibles o modificar su formato.

Se desarrolló un servicio propio que:

- Implementa almacenamiento en caché.
- Permite *fallback* manual en caso de indisponibilidad externa.
- Desacopla el sistema de cambios directos en la fuente.

Inicialmente existieron complejidades con la validación de certificaciones para poder acceder al BCU, lo que requirió hacer ajustes al servicio. A pesar de esto, no se registraron interrupciones críticas, y el sistema mantuvo continuidad operativa incluso ante variaciones en la fuente externa.

8.9. Conclusiones y lecciones aprendidas

La gestión del proyecto evidenció que el éxito no radica en la aplicación rígida de una metodología, sino en la capacidad del equipo para inspeccionar su proceso de trabajo y adaptarlo continuamente al contexto del proyecto y del cliente.

La comunicación dentro del equipo resultó determinante para el avance del proyecto. Cuando fue deficiente, la productividad se vio afectada directamente. Esto evidenció la importancia de contar con instancias regulares de sincronización, particularmente reuniones breves y frecuentes que permitieran verificar el estado de las tareas, identificar bloqueos y resolverlos con rapidez.

Inicialmente el equipo utilizó un único canal de comunicación, por ser el más conocido y familiar. Sin embargo, a medida que el proyecto avanzó se hizo evidente que distintos tipos de interacción requerían espacios diferenciados. Definir herramientas específicas según su propósito permitió ordenar el flujo de trabajo: canales para seguimiento diario asincrónico, espacios para discusiones técnicas estructuradas y reuniones sincrónicas para ceremonias del proceso. Esta separación contribuyó significativamente a mejorar la organización y la eficiencia en la resolución de problemas.

Dentro de Scrum se respetaron los tres pilares empíricos: transparencia, inspección y adaptación. Esto implicó realizar ajustes concretos en la forma de trabajo cuando el contexto lo requirió, como adaptar las dinámicas de seguimiento diario para contemplar diferencias de disponibilidad entre los integrantes o modificar herramientas de gestión cuando generaban fricciones operativas. La experiencia demostró que el valor de Scrum radica en su capacidad para facilitar la mejora continua más que en la adhesión estricta a prácticas predeterminadas.

El seguimiento sistemático de métricas de avance permitió observar la evolución del proceso de trabajo y validar con datos la maduración del equipo. Durante las primeras iteraciones existió una variabilidad esperable en el cumplimiento de los compromisos

de *sprint*. Sin embargo, a medida que se acumuló historial y experiencia compartida, el equipo logró estabilizar su capacidad de estimación y ejecución, alcanzando niveles de cumplimiento cercanos al 90–100 % en las últimas iteraciones. Esta visibilidad sobre el progreso facilitó la detección temprana de desvíos y permitió tomar decisiones correctivas con rapidez.

Finalmente, decidir un plan de *releases* temprano fue un gran acierto, ya que permitió al equipo explicitar aquellos elementos que pondrían en riesgo dicho plan. En base a esa identificación fue que se realizaron investigaciones para decidir cuáles expresaban mayor incertidumbre, cómo reducir dicha incertidumbre, y qué pasos tomar en caso de imprevistos riesgosos. Además, cuando algunas dependencias externas generaron demoras, el equipo pudo reorganizar el backlog y continuar avanzando en otras funcionalidades sin comprometer el progreso general del proyecto.

9. Gestión de la calidad

9.1. ¿Qué es la calidad para DATIUM?

DATIUM fue desarrollado para reemplazar un esquema basado en planillas de cálculo, caracterizado por procesos manuales, cálculos financieros complejos e información dispersa. En este contexto, la calidad no puede limitarse a la implementación correcta de funcionalidades, sino que debe garantizar que el sistema reduzca los errores humanos, centralice los datos en una única fuente de verdad y proporcione información confiable para la toma de decisiones.

Para estructurar el concepto de calidad, se tomó como referencia el modelo de calidad definido por la norma ISO/IEC 25010, que propone un conjunto de características y subcaracterísticas para evaluar la calidad de productos de software. No obstante, dado el dominio específico del proyecto, se priorizaron aquellos atributos que resultan críticos para un sistema financiero en producción.

Desde este punto de vista, la calidad en DATIUM se define en torno a los siguientes ejes:

Primero, la correctitud funcional del sistema, alineada con la característica de adecuación funcional de ISO/IEC 25010. Este aspecto es crítico, ya que se trata de un sistema que gestiona contratos financieros, cuotas, moras y conversiones entre monedas. Los cálculos deben ser matemáticamente correctos, reproducibles en el tiempo y consistentes con las reglas de negocio definidas. Un error en este nivel impacta directamente en la confianza del sistema y en la operación del negocio.

Segundo, la integridad de los datos (relacionado a la pérdida de datos, uno de los dolores mencionados en la sección 2.1. Contexto), vinculada con las características de fiabilidad y seguridad. La información financiera debe mantenerse coherente y protegida frente a inconsistencias, accesos indebidos o modificaciones no controladas.

Desde esta perspectiva, la posibilidad de auditar operaciones sensibles y garantizar consistencia transaccional es un indicador de calidad.

Tercero, la trazabilidad (atributo de calidad nombrado en la sección 7.2.3. Seguridad y trazabilidad (RNF-03)), también relacionada con la seguridad y la responsabilidad en el uso del sistema. Dado que DATIUM reemplaza procesos manuales donde la información podía quedar dispersa, es importante poder reconstruir la secuencia temporal de las operaciones realizadas. El sistema actual registra marcas temporales de creación en las entidades principales y eventos específicos (mensajes de WhatsApp, notificaciones por email), lo que permite reconstruir la cronología de las operaciones. No obstante, no se implementó una auditoría formal con atribución por usuario (registro de quién realizó cada acción), lo cual fue una decisión deliberada para el contexto operativo actual y fue identificada como mejora futura (ver sección 11.3. Lecciones aprendidas).

Cuarto, la usabilidad (atributo de calidad mencionado en la sección 7.2.2. Usabilidad (RNF-02)), correspondiente a la característica con el mismo nombre del modelo ISO/IEC 25010. La calidad no solo depende de que el sistema funcione correctamente, sino de que la información sea presentada de forma clara, comprensible y eficiente. Una interfaz que cumpla esto mejora la experiencia del usuario y reduce la carga cognitiva en tareas operativas.

Quinto y último, la mantenibilidad (atributo nombrado en la 7.2.4. Mantenibilidad (RNF-04)), otro atributo explícitamente definido por la norma. Dado que se prevé que el sistema continúe creciendo luego de terminado el proyecto académico, la calidad implica que sea posible incorporar nuevos cambios sin comprometer las funcionalidad existentes ni perjudicar la estabilidad del sistema.

9.2. Prácticas de aseguramiento de la calidad

9.2.1. Aplicación de estándares

Con el objetivo de garantizar los atributos de calidad definidos previamente, el desarrollo de DATIUM se apoyó en la adopción de estándares tanto a nivel conceptual como técnico. La aplicación de estos lineamientos permitió reducir la variabilidad en las decisiones de implementación, mejorar la consistencia del sistema y facilitar su futuro crecimiento.

Para empezar, se tomó como referencia el modelo de calidad definido por la norma ISO/IEC 25010, el cual establece características como adecuación funcional, fiabilidad, seguridad, usabilidad y mantenibilidad. Si bien el proyecto no siguió una certificación formal bajo dicha norma, su estructura sirvió como marco para orientar las decisiones de diseño y priorizar atributos críticos para un sistema en producción.

Se adoptaron convenciones explícitas de estilo y organización del código. En el *backend* desarrollado en Python se siguieron las recomendaciones de la guía PEP 8 [47] para mantener uniformidad y legibilidad. Asimismo, se promovió una arquitectura modular con clara separación de responsabilidades, diferenciando la lógica de negocio, la capa de acceso a datos y la capa de exposición de servicios. Esta organización favorece la mantenibilidad y reduce el acoplamiento entre componentes.

La gestión del código fuente se realizó mediante Git, utilizando un flujo de trabajo basado en ramas separadas (Gitflow [36]) para desarrollo y producción (develop y main), con el objetivo de aislar cambios, realizar revisiones previas a la integración y disminuir riesgos asociados a despliegues directos en producción.

Relacionado a seguridad, se implementó autenticación mediante OAuth 2.0 con emisión de JSON Web Tokens (JWT) y un esquema de control de acceso binario (administrador /usuario estándar) mediante dependencias declarativas de FastAPI, asegurando que cada usuario acceda únicamente a las funcionalidades

correspondientes a su perfil. Además, las credenciales y variables sensibles fueron gestionadas mediante variables de entorno diferenciadas por nivel, para evitar exponerlas en el código fuente:

- Archivo .env para desarrollo local.
- GitHub Secrets en el *pipeline* de integración continua.
- Variables de entorno en el servidor de producción.

Con respecto a la infraestructura, ya se ha mencionado el uso de Terraform como herramienta de infraestructura como código, permitiendo definir de manera declarativa los recursos desplegados en AWS. Con esto se garantiza la reproducibilidad de los entornos, reduce errores derivados de configuraciones manuales y facilita la trazabilidad de cambios en la infraestructura al estar versionado como cualquier otro archivo en Git.

Finalmente, se configuraron *pipelines* de integración y despliegue continuo (CI/CD) en el repositorio orquestador all (ver sección 10.3 Organización de los repositorios), de modo que cada cambio integrado en la rama develop o main desencadene automáticamente procesos de validación y despliegue para el entorno correspondiente. Esta automatización disminuye la intervención manual y asegura consistencia entre entornos.

9.2.2. Revisiones

Se estableció como práctica obligatoria el uso de PRs para la integración de cambios en las ramas principales del repositorio Git (develop y main). Para estructurar este proceso y dotarlo de mayor rigor metodológico, se implementó una plantilla estandarizada (ver anexo 8 - Template de *Pull Request*). Este formato exige que el autor del código incluya una descripción del problema, la justificación técnica de la

solución y una sección explícita denominada *Testing* Steps for Reviewers (Pasos de prueba para revisores).

Ninguna funcionalidad o corrección era incorporada a las ramas de integración sin que otro integrante del equipo ejecutara este guion de pruebas de forma independiente. Este proceso de *testing* cruzado, guiado por las instrucciones de la plantilla, garantizó la reproducibilidad de las pruebas manuales. De este modo, fue posible evaluar de manera objetiva la correctitud funcional de la implementación, verificar el cumplimiento de las convenciones definidas y analizar el posible impacto sobre los componentes existentes antes de autorizar la fusión del código.

Durante las revisiones de código se ponía especial énfasis en la lógica de negocio vinculada a los cálculos financieros, dado que pequeños errores en este nivel podrían generar inconsistencias significativas en la operación del sistema. Asimismo, se evaluaba la claridad del código, la correcta separación de responsabilidades y la presencia de pruebas automatizadas asociadas a la nueva funcionalidad.

Además, se realizaron revisiones funcionales previas a la liberación de nuevas versiones al entorno productivo. Estas instancias consistían en validar manualmente los flujos completos de uso, verificando que el comportamiento observado coincidiera con los requerimientos definidos y que no se hubieran introducido regresiones. Esta práctica permitió complementar las pruebas automatizadas con una evaluación integral del funcionamiento del sistema.

El proyecto también incluyó revisiones periódicas con el cliente (las *Sprint Reviews*, explicadas en la sección 8.3. Aplicación de Scrum), en las cuales se presentaban los incrementos funcionales desarrollados durante cada iteración. Estas instancias no solo permitieron validar que la solución se alineara con las expectativas del negocio, sino que también facilitaron la detección temprana de ajustes necesarios en reglas o procesos. De esta forma, la revisión no se limitó al código, sino que abarcó también la adecuación funcional del sistema al contexto real de uso.

9.2.3. Pruebas

Se implementaron pruebas automatizadas sobre la lógica de negocio del sistema. Estas pruebas, mayoritariamente unitarias, verifican el comportamiento de funciones críticas tales como el cálculo de cuotas, intereses, mora y conversiones entre monedas. El objetivo principal fue garantizar la correctitud matemática de los algoritmos y su reproducibilidad ante distintos escenarios; estas pruebas abarcan al menos un 80% de líneas de código de la lógica de negocio.

Además, se desarrollaron pruebas de integración orientadas a validar la interacción entre componentes, particularmente en aquellos casos donde intervienen múltiples capas del sistema (por ejemplo, persistencia de datos y reglas de negocio). Este enfoque permitió detectar inconsistencias que no serían visibles al evaluar cada módulo de forma aislada.

Las pruebas automatizadas fueron integradas al *pipeline* de integración continua (CI), de modo que cada cambio incorporado al repositorio ejecutará automáticamente el conjunto de *tests*. En caso de que alguna prueba fallara o se incumpliera el mínimo de 80% de cobertura aplica estrictamente a la lógica de negocio del *Backend*, el proceso de integración se detenía, impidiendo la propagación de defectos hacia ramas de mayor estabilidad. Esta práctica redujo el riesgo de regresiones y fortaleció la confiabilidad del sistema a lo largo de su evolución.

Para complementar, se realizaron pruebas manuales orientadas a validar flujos completos de uso desde la perspectiva del usuario. El objetivo de estas fue evaluar escenarios reales del negocio, verificar la coherencia de la información presentada en la interfaz y detectar posibles problemas de usabilidad que no necesariamente emergen en pruebas unitarias.

Previo a cada despliegue en producción se realizaron validaciones en el entorno de desarrollo, que tiene la misma configuración que el productivo. Esto permitió verificar el

comportamiento integral del sistema en condiciones cercanas a las reales, disminuyendo la probabilidad de errores asociados a diferencias de configuración entre entornos.

9.3. Métricas

9.3.1. Análisis estructural y dependencias entre módulos

Para analizar la estructura arquitectónica del sistema se generó un grafo de dependencias entre módulos del paquete **`/app`** dentro del repositorio del *backend* (ver anexo 19 - Resumen de métricas de calidad por un resumen). A partir de dicho grafo se evaluaron los niveles de acoplamiento y estabilidad relativa entre módulos, lo que permitió identificar los componentes centrales del dominio.

Se evidencia una arquitectura organizada en capas claramente diferenciadas:

- Infraestructura: **`app.db`**
- Modelo de dominio: **`app.models`**
- Acceso a datos (CRUD): **`app.crud`**
- Interfaces y DTOs: **`app.schemas`**
- Servicios de negocio: **`app.services`**
- Utilidades transversales: **`app.utils`**
- Excepciones específicas de dominio: **`app.exception`**

Las dependencias observadas siguen el patrón esperado:

- Los módulos de *schemas* y *crud* dependen de *models*.
- Los servicios dependen de *crud*, *schemas* y utilidades.

- La infraestructura (db) es transversal.

No se identificaron ciclos de dependencias significativos entre capas principales, lo cual constituye un indicador positivo de modularidad y separación de responsabilidades.

Tomando como referencia el modelo de estabilidad propuesto por Robert C. Martin [48], se entiende que la inestabilidad se asocia a un alto número de dependencias salientes, mientras que la estabilidad se asocia a un alto número de dependencias entrantes. Del análisis se observó que los módulos con mayor cantidad de dependencias salientes fueron:

- `app.db.db`
- `app.db`
- `app.utils`

Esto es consistente con su rol arquitectónico: la infraestructura y las utilidades son naturalmente dependientes de múltiples componentes.

Por otro lado, los módulos con mayor cantidad de dependencias entrantes fueron:

- `app.crud.sale_contract`
- `app.models`
- `app.crud.payment`

Esto indica que el núcleo del dominio (se destaca la lógica relacionada a contratos de venta, la cual es central al negocio) constituye el centro estable del sistema, siendo ampliamente referenciado por otros módulos. Este comportamiento es coherente con los principios de arquitectura limpia, donde las dependencias deben dirigirse hacia el núcleo del dominio y no a la inversa.

El análisis del grafo de dependencias permite concluir que:

- Existe separación clara de responsabilidades.
- No se evidencian ciclos estructurales críticos.
- La infraestructura no invade el dominio.
- El núcleo del negocio se encuentra correctamente centralizado y estabilizado.

9.3.2. Complejidad ciclomática

La complejidad ciclomática (CC) fue calculada mediante Radon [49] sobre el código fuente del *backend*, excluyendo los *tests* (ver anexo 20 - Mapa de calor de Complejidad Ciclomática). El promedio de complejidad ciclomática de las 551 funciones y clases analizadas es 3,83, lo que indica que la mayoría de las funciones poseen estructuras de control simples, con bajo nivel de ramificación condicional. Además, la vasta mayoría de dichas funciones y clases (82,9%) entra en la categoría A de complejidad ciclomática [50], que comprende valores entre 1 y 5 y es considerado un rango de complejidad bajo.

Distribución de complejidad ciclomática:

- Categoría A (riesgo bajo): 82,9%
- Categoría B–D (riesgo bajo-moderado): minoritarias
- Categoría E/F (riesgo alto): 4 funciones

Las funciones con valores elevados de complejidad (mayor a 15) se concentran principalmente en:

- Módulos de *dashboard*
- Ajustes de pagos

- Generación de reportes financieros

Esta concentración es consistente con la presencia de reglas de negocio más complejas en dichas áreas, lo cual es esperable dado el dominio del sistema. No se observa una distribución caótica de alta complejidad a lo largo del código, sino una localización puntual en módulos específicos.

9.3.3. Índice de mantenibilidad

El índice de mantenibilidad fue calculado mediante Radon [49] sobre el código fuente del *backend* (ver anexo 21 - Gráfica de Índice de Mantenibilidad por archivo). El valor promedio obtenido fue 75,87, lo cual indica un nivel alto de mantenibilidad.

Según la clasificación utilizada por Microsoft [51], los valores comprendidos entre 20 y 100 se consideran indicativos de buena mantenibilidad.

Distribución del índice de mantenibilidad:

- Categoría A: 98,2%
- Categoría B: 1,8%
- Categoría C o inferior: 0%

Estos resultados indican que la gran mayoría de los módulos presentan buena mantenibilidad considerando tamaño, complejidad y estructura. Se observa además que los módulos con menor índice de mantenibilidad coinciden con aquellos que presentan mayor complejidad ciclomática, lo que demuestra coherencia entre métricas.

9.4. Conclusiones y lecciones aprendidas

Se concluye que la calidad del proyecto queda reflejada de forma consistente en tres puntos principales: la separación clara de responsabilidades del sistema, el bajo nivel general de complejidad ciclomática y el alto índice de mantenibilidad alcanzado. En

conjunto, estos resultados muestran que fue posible construir una solución estructuralmente sólida y preparada para evolucionar.

Las métricas obtenidas evidencian que la complejidad del sistema se mantiene en niveles relativamente bajos. Los casos de mayor complejidad se concentran en módulos asociados a la presencia de reglas de negocio más exigentes, como lo son el *dashboard*, los ajustes de pagos y la generación de reportes financieros. Entonces, no hay una dispersión descontrolada de complejidad, sino una localización puntual en componentes del dominio que lo requieren.

Asimismo, el índice de mantenibilidad obtenido permite afirmar que la gran mayoría de los módulos presenta buenas condiciones de mantenimiento considerando tamaño, complejidad y estructura. La coincidencia entre los módulos más complejos y aquellos con menor mantenibilidad también muestra coherencia entre las métricas aplicadas, reforzando la validez del análisis realizado.

Una de las principales lecciones aprendidas fue la importancia de incorporar herramientas de análisis estático tempranamente como parte del aseguramiento de la calidad. Esto permitió establecer criterios objetivos sobre la estructura y la calidad del código desde etapas iniciales del desarrollo. Del mismo modo, las revisiones de código tuvieron un papel central para sostener estos estándares y detectar problemas antes de integrar cambios al sistema.

10. Gestión de la configuración

Este capítulo describe los mecanismos adoptados para controlar, versionar y desplegar, garantizando trazabilidad de cambios, reproducibilidad de entornos y un proceso de publicación consistente. La gestión de la configuración se apoya en GitHub como repositorio remoto y plataforma de integración continua (CI), en Infraestructura como Código (Terraform) y en una separación explícita de entornos de ejecución.

10.1. Identificación de elementos de configuración

Se consideraron elementos de configuración (CIs) todos aquellos artefactos cuya modificación impacta el comportamiento del sistema o su operación:

- Código fuente
    - *Backend*: API REST (Python/FastAPI)
    - *Frontend*: aplicación web (React/TypeScript)
- Infraestructura y automatización
    - Definiciones Terraform de recursos AWS (dev/prod).
    - Scripts operativos asociados al despliegue (por ejemplo, despliegue del frontend).
    - Workflows de GitHub Actions utilizados para CI y para despliegue.
- Base de datos
    - Definición del esquema y migraciones SQL (controladas por el mecanismo de migraciones del proyecto).

- Configuración por entorno
    - Variables de entorno de *backend* (DB, OAuth 2.0, WhatsApp, OpenAI, correo, etc.).
    - Variables de *build* del frontend.
    - Parámetros de IaC por entorno (tfvars).
- Credenciales y secretos
    - GitHub Secrets (AWS roles, ECR repos, credenciales de DB, tokens y API keys de integraciones).

Esta clasificación permitió definir qué cambios requieren revisión formal (PR) y qué información debe mantenerse fuera del repositorio (secretos).

10.2. Herramientas utilizadas

- Git + GitHub: versionado, historial, PRs y trazabilidad de cambios.
- GitHub Actions: automatización de CI (*tests*/*build*) y CD (*deploy*).
- Terraform (Infraestructura como Código): versionado y reproducibilidad de la infraestructura AWS.
- Docker: empaquetado y ejecución del *backend* como imagen, publicada en ECR y desplegada en AWS.
- AWS: recursos de infraestructura para ambientes dev/prod (sitio estático, distribución, cómputo, red, base de datos, etc.).
- MySQL: motor de persistencia del sistema, con migraciones versionadas.

10.3. Organización de los repositorios

La solución se implementó en tres repositorios dentro de la organización tesis-ort-terrenos:

- *backend*: contiene el código del *backend*.
- *frontend*: contiene el código del *frontend*.
- all: repo orquestador que contiene la infraestructura (Terraform), los workflows de despliegue (dev/prod) y scripts de despliegue del frontend.

El repositorio all integra *backend* y frontend como submódulos (*checkout* recursivo), lo que permite que los workflows de despliegue operen sobre una "foto" consistente del *backend* y el frontend al momento de ejecutar el *pipeline*.

10.4. Gestión de versiones

Se adoptó un flujo de trabajo alineado con Gitflow [36], utilizando las ramas principales:

- *develop*: rama de integración y desarrollo continuo.
- *main*: rama estable asociada al entorno productivo.

El desarrollo se realiza en ramas de feature y se integra mediante PRs con revisión por pares y validaciones automáticas previas a la integración, según se describe en la sección 8.3. Aplicación de Scrum. En términos generales, el flujo seguido fue:

1. *feature*
2. PR a *develop*
3. verificación en *dev*
4. *merge* de develop a main cuando corresponde liberar a producción

10.5. Gestión de cambios

Los cambios se administraron asegurando trazabilidad desde la implementación hasta su despliegue:

- Trazabilidad y control: cada cambio queda registrado como *commits* y PRs, permitiendo reconstruir el historial y justificar modificaciones.
- Verificación automática (CI):
    - *Backend* (backend/.github/workflows/tests.yml):
        - *Lint* con flake8.
        - Formateo automático con black.
        - *Tests* con pytest y cobertura mínima exigida.
        - Se definió un umbral mínimo de cobertura del 80%, y su cumplimiento es parte del *pipeline* automatizado.
    - *Frontend* (frontend/.github/workflows/build.yml):
        - Validación de *build* (npm ci + npm run *build*).
- Publicación (CD) a entornos
    - El despliegue se automatizó con *workflows* en el repo all:
        i. dev-deploy.yml al pushear/mergear en *develop* (*dev*).
        ii. prod-deploy.yml al pushear/mergear en *main* (*prod*).
    - En ambos casos, el *pipeline*:
        i. Construye y publica la imagen Docker del *backend* en ECR

ii. Ejecuta Terraform *apply* contra el *workspace* correspondiente *(dev/prod)*

iii. Ejecuta el *script* de despliegue del *frontend* (frontend-deploy.ps1) inyectando la API URL en el *build* y subiendo artefactos a S3, con invalidación de CloudFront.

10.5.1. Gestión de incidentes

Los incidentes se gestionaron como *issues/tickets* etiquetados, utilizando las siguientes etiquetas para priorizar:

- *Bug*: identifica que el ticket corresponde a un defecto del sistema.
- Severidad: se asigna una etiqueta según impacto:
    - *High severity:* impacto crítico (bloquea operación, pérdida/corrupción de datos, fallo de flujo principal).
    - *Medium severity:* impacto relevante pero con alternativa o degradación parcial.
    - *Low severity:* impacto menor (casos borde, problemas cosméticos o de baja frecuencia).

Esta clasificación permitió ordenar el trabajo de corrección y definir urgencia de resolución y promoción hacia producción, manteniendo trazabilidad desde el incidente hasta el PR y el despliegue.

10.6. Conclusiones y lecciones aprendidas

La gestión de la configuración en DATIUM no se limitó al versionado de código, sino que se consolidó como una práctica orientada a garantizar la reproducibilidad de la solución, la consistencia entre entornos y la confiabilidad del proceso de despliegue.

La separación del *frontend* y el *backend* en repositorios Git independientes resultó adecuada para acompañar la dinámica de trabajo del equipo, ya que permitió

desacoplar los ciclos de cambio, adaptar los flujos de integración continua a las particularidades de cada tecnología y facilitar la evolución de cada componente sin introducir interferencias innecesarias. No obstante, la principal lección técnica surgió al complementar esta independencia con un repositorio adicional de carácter orquestador, encargado de integrar ambos componentes mediante submódulos. Esta decisión permitió conservar la autonomía durante el desarrollo y, al mismo tiempo, garantizar que cada despliegue se realizará sobre una versión explícita y sincronizada del sistema completo. En este sentido, se comprobó que la independencia entre componentes aporta flexibilidad, pero que dicha flexibilidad sólo resulta sostenible cuando existe un mecanismo claro que preserve la coherencia global de la solución al momento de su liberación.

A su vez, la utilización de Terraform permitió gestionar la infraestructura de forma declarativa, versionada y reproducible. Esto redujo la dependencia de configuraciones manuales y ayudó a mantener una mayor coherencia entre los entornos de desarrollo y producción, disminuyendo la incertidumbre asociada a los despliegues.

Finalmente, la gestión de configuraciones sensibles mediante variables de entorno y GitHub Secrets evitó exponer credenciales en el código fuente. Asimismo, la automatización de los despliegues mediante GitHub Actions eliminó procedimientos manuales y permitió establecer controles previos a la publicación de nuevas versiones, como la compilación correcta del sistema y el cumplimiento del umbral mínimo de cobertura de pruebas.

11. Conclusiones

11.1 Verificación y discusión de cumplimiento de los objetivos

En esta sección se analiza el nivel de cumplimiento de los objetivos planteados al inicio del proyecto (definidos en la sección 1.2. Objetivos).

El análisis se organiza según las tres categorías planteadas originalmente: objetivos del producto, objetivos del proyecto y objetivos académicos. En cada caso se evalúa no solo si el objetivo fue alcanzado, sino también de qué manera la evidencia obtenida durante el desarrollo, la validación con el cliente, la puesta en producción y los resultados observados en uso real permiten sostener esa conclusión.

Dado que los objetivos del producto y del proyecto fueron formulados bajo enfoque SMART, su verificación se apoya en resultados concretos y observables. En el caso de los objetivos académicos, el análisis se centra en la aplicación efectiva de prácticas, métodos y conocimientos de Ingeniería de Software en un contexto real de desarrollo.

11.1.1 Objetivos del producto

OProd01) Centralización de la información del negocio – Completamente cumplido.

El sistema logró erradicar la dependencia de información fragmentada mediante la implementación de una base de datos MySQL relacional. El modelo de datos resultante consta de 18 entidades que estructuran el flujo principal del negocio de forma robusta, garantizando que la información de clientes, etapas, padrones y contratos se mantenga centralizada y relacionada como una única fuente de verdad.

(ver sección 7.5.2. Modelo de datos).

OProd02) Gestión integral de contratos de venta – Completamente cumplido.

La solución incorpora las funcionalidades necesarias para registrar, consultar y actualizar contratos de venta, contemplando su ciclo operativo y el seguimiento del plan de pagos asociado. La plataforma logró modelar adecuadamente la complejidad real del negocio, incluyendo múltiples compradores por contrato, manejo de distintas monedas y estados contractuales, lo que permitió sustituir una operatoria que antes dependía de registros manuales y criterios no estandarizados. En este sentido, el cumplimiento del objetivo radica tanto en la cobertura funcional alcanzada como en la capacidad del sistema para representar de forma consistente la lógica contractual del negocio.

(ver secciones 3.2. Principales funcionalidades y 6.2. Funcionalidades (RF-04)).

OProd03) Automatización del cálculo y generación de cuotas – Completamente cumplido.

El sistema implementa un motor de cálculo capaz de generar cronogramas de cuotas a partir de reglas explícitas del negocio, contemplando importes, vencimientos, moras, intereses y conversiones entre monedas. Su valor es especialmente significativo porque uno de los principales problemas del contexto inicial estaba asociado a la fragilidad y al esfuerzo de los cálculos manuales.

(Ver secciones 6.2. Funcionalidades (RF-05) y 7.5.4. Integraciones externas)

OProd04) Registro de pagos con validaciones de consistencia – Completamente cumplido.

La plataforma permite registrar pagos asociados a contratos aplicando validaciones de negocio orientadas a preservar la integridad de la información financiera.

Este flujo se implementó utilizando transacciones ACID soportadas por el motor InnoDB de MySQL, garantizando que operaciones como registrar un pago, calcular la mora

correspondiente y actualizar los balances se ejecuten de forma atómica y no queden en estados inconsistentes ante fallas.

Además del registro en sí, el sistema automatiza controles que anteriormente dependían del manejo manual. Entre ellos se incluyen el tratamiento de excedentes o faltantes entre pagos consecutivos, la aplicación de prórrogas hasta fin de mes y el recálculo en cascada cuando se modifica un pago anterior.

También se implementó una validación específica para pagos realizados a través de redes de cobranza externas (Abitab y Redpagos). En estos casos el sistema verifica la coherencia entre el importe bruto ingresado, la comisión fija definida en la interfaz y el importe neto resultante.

En consecuencia, el objetivo se considera alcanzado no solo desde la funcionalidad disponible, sino también desde la confiabilidad del proceso de cobranza, reduciendo significativamente el riesgo de errores manuales.

(Ver secciones 6.2. Funcionalidades (RF-06), 6.3.1. RNF-01 Integridad de datos y 7.2.1 Integridad de datos (RNF-01))

OProd05) Generación de informes operativos – Completamente cumplido.

La necesidad analítica del cliente fue resuelta a través de un módulo dedicado que previsualiza e interactúa con cinco reportes operativos clave: cobranzas, ventas, ingreso por ejercicio, cuotas vencidas y deudores. El cumplimiento de este objetivo se complementa con la funcionalidad de exportación estructurada a formato Excel, la cual resulta indispensable para las tareas contables y de auditoría externa del cliente.

(Ver sección 6.2. Funcionalidades (RF-08 y RF-09))

OProd06) Reducción de fricción operativa mediante usabilidad (UX/UI) – Completamente cumplido.

El diseño de la solución priorizó explícitamente la claridad de los flujos, la organización de la información y la reducción de la carga cognitiva de los usuarios administrativos. Este objetivo se trabajó de forma iterativa mediante prototipos y revisiones funcionales, lo que permitió ajustar tempranamente la interfaz antes de consolidarla en desarrollo.

Para evitar que la usabilidad quedara sujeta a criterios subjetivos, las decisiones de diseño se fundamentaron en la aplicación explícita de las heurísticas de Jakob Nielsen [13]. De esta forma se buscó garantizar consistencia en la interacción, claridad en la presentación de la información y prevención de errores en las tareas operativas más frecuentes.

La evidencia posterior de uso y satisfacción del cliente refuerza esta conclusión, ya que se valoró positivamente la facilidad de uso del sistema, la claridad de la información presentada y la simplicidad para seguir los flujos principales de trabajo. En consecuencia, la usabilidad no fue un atributo accesorio del sistema, sino un componente efectivo del valor entregado.

(Ver secciones 6.1. Proceso , 6.3.2. RNF-02 Usabilidad, 7.2.2 Usabilidad (RNF-02), y anexo 6 – Encuestas de satisfacción y anexo 13.7. Anexo 7 - Capturas de pantalla del sistema implementado)

OProd07) Provisión de un canal de autoservicio para compradores – Mayormente cumplido.

Desde el punto de vista técnico, se logró desarrollar e integrar un canal de autoservicio automatizado mediante un *bot* conversacional utilizando la WhatsApp Cloud API de Meta. Esta herramienta permite a los compradores autenticarse de forma segura y

consultar de manera autónoma información relevante, como próximos vencimientos, cuotas pendientes y su estado de cuenta.

El cumplimiento se califica como mayormente cumplido porque, si bien el canal se encuentra completamente implementado y validado por el cliente en el entorno de desarrollo, se acordó con el negocio no habilitarlo todavía para los usuarios finales en producción. Esta decisión se vincula directamente con la postergación de la migración masiva de los datos históricos.

Actualmente la plataforma gestiona únicamente los contratos nuevos generados dentro del sistema. En este contexto, habilitar el *bot* implicaría ofrecer el canal de autoservicio solo a un segmento muy reducido de la cartera de compradores, por lo que se decidió posponer su activación hasta que el sistema concentre una mayor proporción de la información operativa del negocio.

(Ver secciones 3.3.2 Evolución y ajuste del alcance, 6.2. Funcionalidades (RF-10), 7.3. Architectural decision records (ADRs) (ADR-06) y anexo 9 - Proceso de habilitación del chatbot de WhatsApp mediante Meta)

11.1.2 Objetivos del proyecto

OProy01) Implementación de una solución operativa – Completamente cumplido.

El proyecto finalizó con el despliegue de la plataforma en los servidores de Amazon Web Services (AWS), utilizando infraestructura como código (Terraform) para asegurar su reproducibilidad. La solución se encuentra operando en un ambiente productivo estable y ya fue adoptada por el cliente para el registro cotidiano de sus nuevas operaciones comerciales.

(Ver secciones 4. Estrategia de construcción de la solución y 10. Gestión de la configuración)

OProy02) Validación iterativa con usuarios – Completamente cumplido.

La construcción se ejecutó en 15 *sprints*, incorporando demostraciones y revisiones funcionales constantes con el cliente. El uso de prototipos desde las etapas tempranas de diseño permitió validar flujos de pantalla antes de programarlos, y las sesiones de revisión fueron vitales para descubrir reglas de cálculo financiero que el cliente no había explicitado al inicio (como por ejemplo los distintos modos de pago para contratos financiados en USD).

(Ver secciones 5. Marco metodológico y 6.1.1. Relevamiento)

OProy03) Calidad técnica y sostenibilidad del sistema – Completamente cumplido.

El proyecto incorporó prácticas concretas de aseguramiento de la calidad orientadas a la robustez, mantenibilidad y confiabilidad del sistema. La evaluación técnica se apoyó en atributos tomados de ISO/IEC 25010, pruebas automatizadas, documentación técnica y métricas de calidad. Entre los resultados más relevantes se destaca que el *backend* superó el umbral del 80% de cobertura en pruebas unitarias e integradas y mantuvo una complejidad ciclomática promedio baja (3,83), lo que constituye evidencia objetiva de un producto sostenible desde el punto de vista técnico.

(Ver secciones 9. Gestión de la calidad y 10.5. Gestión de cambios)

OProy04) Satisfacción Operativa del Cliente – Completamente cumplido

Las encuestas de satisfacción aplicadas luego de cada *release* muestran que DATIUM fue percibido por el cliente como una herramienta útil, confiable y valiosa para su operativa cotidiana, superando además el criterio de éxito definido inicialmente, que establecía un puntaje igual o superior a 4 sobre 5 en los indicadores evaluados correspondientemente en las encuestas de satisfacción *post-release*.

En el primer *release*, el cliente valoró con 5/5 aspectos como facilidad de uso, rapidez de respuesta, utilidad de los reportes y coincidencia entre los cálculos del sistema y los controles manuales, mientras que la confianza en la exactitud de los cálculos obtuvo 4/5. En el segundo *release*, esta percepción evolucionó favorablemente y la confianza en los cálculos pasó a 5/5, acompañada por una mejora en la satisfacción general y en la intención de seguir utilizando la plataforma.

A su vez, el sistema fue adoptado como herramienta principal para la gestión de nuevos contratos, manteniéndose una convivencia transitoria con planillas para información histórica, en línea con el alcance previsto. Por lo tanto, el objetivo se considera completamente cumplido porque existe evidencia tanto de adopción como de satisfacción sostenida en uso real.

(Ver *secciones* 3.3. Alcance del proyecto, y 4. Estrategia de construcción de la solución y anexo 6.2 – Encuesta de satisfacción (*Release* 2))

11.1.3 Objetivos académicos

OAcad01) Ingeniería de requerimientos – Completamente cumplido.

El relevamiento se ejecutó con rigor metodológico utilizando el framework de Tsumaki y Tamai, lo que aseguró una aplicación equilibrada de técnicas estáticas (análisis de planillas de Excel) y dinámicas (entrevistas, *brainstorming y* validación continua de prototipos). Posteriormente, formalizamos los descubrimientos derivados de este proceso mediante historias de usuario estimadas y respaldadas con criterios de aceptación en formato Gherkin. Esta práctica nos permitió evitar ambigüedades y enlazar directamente las necesidades del cliente con nuestras pruebas automatizadas, garantizando la trazabilidad total del requerimiento.

(Ver secciones 6.1.1. Relevamiento y 6.1.2. Especificación)

OAcad02) Diseño y justificación técnica de la solución – Completamente cumplido.

Se fundamentó cada decisión estructural en base a los atributos de calidad priorizados según el estándar ISO/IEC 25010. La arquitectura del sistema fue estructurada bajo el enfoque *Views and Beyond* y todas las decisiones estructurales críticas fueron formalmente fundamentadas utilizando *Architectural Decision Records* (ADRs). Decisiones complejas, como la separación en múltiples repositorios o la implementación del control de acceso, fueron evaluadas documentando detalladamente sus alternativas, consecuencias y su impacto en los atributos de calidad requeridos. Esto nos permitió demostrar un enfoque pragmático y maduro, evaluando alternativas y *trade-offs* reales en la arquitectura en lugar de adoptar patrones de diseño teóricos a ciegas

(*Ver secciones* *7.1. Visión general de la arquitectura* y 7.3. Architectural decision records (ADRs))

OAcad03) Gestión del proyecto con enfoque ágil – Completamente cumplido.

Se usó Scrum como marco de trabajo ágil para gestionar el proyecto. El equipo utilizó herramientas industriales (GitHub, Linear) y métricas cuantitativas que demostraron empíricamente cómo, a partir del octavo *Sprint*, lograron madurar su proceso hasta alcanzar una predictibilidad de cumplimiento de estimaciones cercana al 100%. Además, se adaptaron activamente las ceremonias para maximizar el valor del proceso, pasando a un modelo híbrido para el seguimiento diario y evolucionando las dinámicas de retrospectiva hasta encontrar en el formato del "bote" la herramienta ideal para identificar riesgos y oportunidades de mejora.

(Ver *secciones* *5.2.1. Scrum*, 8.3 Aplicación de Scrum y 8.6. Métricas de gestión)

OAcad04) Aprendizaje y aplicación de nuevas tecnologías – Completamente cumplido.

El objetivo se considera completamente cumplido, ya que durante el proyecto se investigaron, integraron y desplegaron tecnologías que trascienden el currículo básico de la carrera, enfrentando desafíos técnicos propios de un entorno de desarrollo profesional.

El equipo consolidó experiencia práctica en infraestructura en la nube, incluyendo la gestión y optimización activa de los costos operativos en AWS. Asimismo, se incorporaron herramientas de observabilidad utilizadas en entornos productivos, como New Relic APM y Amazon CloudWatch, que permitieron mejorar la visibilidad del comportamiento del sistema y facilitar el monitoreo de su funcionamiento.

El principal desafío técnico abordado fue la implementación de un *chatbot* interno basado en el patrón *text-to-SQL*, impulsado por Inteligencia Artificial Generativa mediante el modelo OpenAI GPT-4o-mini. Esta integración requirió el diseño de una arquitectura de seguridad compuesta por múltiples capas de validación, orientada a prevenir inyecciones SQL y mitigar posibles alucinaciones del modelo. De esta forma, el proyecto demostró la capacidad del equipo para incorporar tecnologías emergentes bajo criterios estrictos de seguridad y control en un contexto de datos financieros.

(Ver secciones 7.3. Architectural decision records (ADRs) (ADR-05 y ADR-07), 7.5.4 Integraciones externas y 7.5.6. Análisis y Discusión de Costos de AWS)

En términos generales, el análisis realizado permite concluir que los objetivos definidos al inicio del proyecto fueron alcanzados satisfactoriamente. Los objetivos del producto evidencian que la solución cubre las capacidades funcionales centrales necesarias para atender la problemática identificada. Los objetivos del proyecto muestran que dichas capacidades no quedaron solo en el plano de la implementación, sino que

fueron validadas con usuarios, puestas en producción y respaldadas por evidencia concreta de uso y satisfacción. Finalmente, los objetivos académicos confirman que el proyecto constituyó una instancia real de integración y aplicación de conocimientos de Ingeniería de Software, tanto desde la perspectiva metodológica como técnica. En conjunto, la verificación de objetivos permite afirmar que el trabajo no solo cumplió con lo planificado, sino que además logró traducir esas metas en una solución concreta, adoptada y valiosa para el cliente.

11.2. Conclusiones generales

En esta sección se ofrece una visión integradora del proyecto, complementaria a las conclusiones específicas por disciplina (Ingeniería de Requerimientos, Arquitectura, Gestión y Calidad) detalladas a lo largo del documento.

A continuación se reunen los resultados globales alcanzados, el valor real generado para el cliente y el impacto de la solución como respuesta tecnológica a un problema de negocio:

- **Transformación digital efectiva:** El proyecto culminó con la entrega y puesta en producción exitosa de DATIUM, una plataforma web que centraliza y automatiza los procesos operativos críticos del cliente (gestión de contratos, cuotas, pagos y reportes). La solución permitió sustituir un ecosistema frágil basado en planillas de cálculo y procesos manuales por un sistema transaccional robusto. Sus reglas de negocio explícitas erradican los puntos de falla frecuentes, como los errores de carga humana, las inconsistencias en cálculos multidivisa y los constantes reprocesos financieros.
- **La estrategia iterativa como motor de adopción:** La decisión de construir la solución mediante un plan de *releases* demostró ser el enfoque metodológico adecuado para mitigar riesgos. Liberar tempranamente un núcleo funcional estable, permitió que el cliente comenzara a operar la plataforma en escenarios

reales desde las primeras etapas. Sobre esta base validada, fue posible iterar e incorporar módulos complementarios de manera controlada y progresiva, integrando el *feedback* directo del usuario.

- **Implantación real y convivencia planificada:** Como evidencia definitiva del éxito y adopción del proyecto, el sistema se encuentra operando en producción y ha sido asimilado en el flujo de trabajo cotidiano de la empresa. Tal como acordamos al redefinir el alcance, la plataforma gestiona hoy la totalidad de los contratos nuevos como única fuente de verdad, conviviendo en una transición estructurada con las planillas de datos históricos. Este escenario no solo confirma la viabilidad operativa del producto, sino que traza una línea evolutiva natural hacia la futura consolidación y migración definitiva de la información.
- **La importancia de la calidad técnica**: Se comprobó empíricamente que, en un dominio financiero, incluso desvíos mínimos pueden tener un impacto directo tanto en el capital como en la confianza del usuario. En este contexto, resultó indispensable sostener una estrategia de calidad basada en validaciones estrictas de negocio, pruebas automatizadas y procesos de despliegue controlados. Este rigor técnico permitió asegurar la consistencia de los cálculos y mantener la estabilidad del sistema a medida que el proyecto crecía en alcance y complejidad.

En suma, el equipo considera que este Trabajo Final de Carrera ha cumplido plenamente con su propósito. Logramos trascender el ejercicio netamente académico para entregar una solución de software profesional, validada y adoptada en un entorno productivo real. Puede afirmarse que DATIUM no solo le aporta a nuestro cliente mejoras medibles en la eficiencia y confiabilidad operativa de su emprendimiento, sino que logramos construirlo sobre una base arquitectónica madura, dejándolo preparado para su futura evolución.

11.3. Lecciones aprendidas

A lo largo del proyecto no solo se obtuvo una solución funcional para un problema real, sino también un conjunto de aprendizajes relevantes desde el punto de vista técnico, metodológico y de negocio. Estas lecciones surgieron de la experiencia concreta de trabajar con un dominio financiero-operativo real, con requerimientos que evolucionaron durante el desarrollo, dependencias externas significativas y la necesidad de equilibrar velocidad de entrega con confiabilidad.

En esta sección se sintetizan los principales aprendizajes transversales del proyecto, entendidos no como observaciones aisladas, sino como conclusiones derivadas de decisiones, dificultades, validaciones y resultados efectivamente observados durante la construcción e implantación de la solución. La inclusión de estas lecciones busca aportar una reflexión crítica sobre el proceso recorrido y explicitar aquellos criterios que resultarían valiosos para proyectos futuros de características similares.

11.3.1. Validar temprano en contexto real acelera el aprendizaje

Una de las lecciones más relevantes del proyecto fue comprobar que la validación temprana con el cliente, apoyada primero en prototipos y luego en entregas funcionales reales, acelera el aprendizaje del equipo y mejora la calidad de las decisiones. En este caso, muchas reglas del dominio no estaban completamente explicitadas al inicio, no porque estuvieran mal definidas deliberadamente, sino porque formaban parte de una operativa históricamente sostenida en planillas, excepciones y criterios implícitos.

En ese contexto, intentar cerrar completamente el entendimiento del negocio antes de avanzar hubiera sido poco realista y probablemente ineficiente. La estrategia que resultó más efectiva fue construir una base funcional temprana, ponerla frente al cliente y usar su interacción real con el sistema como insumo de aprendizaje. Esto permitió detectar ambigüedades, identificar excepciones no previstas y ajustar tanto

funcionalidades como decisiones de interfaz sobre evidencia concreta, reduciendo retrabajo posterior.

La evolución observada en la satisfacción del cliente y en su confianza respecto al sistema refuerza esta lección. Las encuestas posteriores a los *releases* muestran no solo aceptación, sino una mejora progresiva en la confianza sobre la exactitud de los cálculos y en la valoración general del producto, lo que sugiere que la validación incremental fue efectiva no solo para aprender más rápido, sino también para consolidar adopción.

11.3.2. En dominios financieros, la “correctitud” no es negociable

Otra lección central del proyecto fue entender que, en un sistema orientado a gestionar contratos, cuotas, pagos, mora e intereses, la correctitud de los cálculos no constituye un atributo más de calidad, sino una condición mínima de viabilidad. A diferencia de otros contextos donde ciertos desajustes funcionales pueden corregirse sin comprometer la confianza general del usuario, en este dominio un error en un cálculo impacta directamente en la credibilidad de toda la solución.

Esto obligó a tratar la lógica financiera como un núcleo crítico del sistema, con especial cuidado en el modelado de reglas, el manejo de monedas, la persistencia de valores históricos y la trazabilidad de cada operación. También implicó comprender que la validación de estos aspectos no podía descansar exclusivamente en pruebas técnicas internas, sino que debía contrastarse con controles reales del cliente.

La evidencia obtenida durante la implantación confirmó la relevancia de este aprendizaje. En el primer *release*, el cliente ya evaluó con 5/5 la coincidencia entre los cálculos del sistema y sus controles manuales, aunque la confianza subjetiva sobre la exactitud total del motor de cálculo comenzó en 4/5. Luego, después de un uso más sostenido, esa confianza evolucionó a 5/5. Esta diferencia entre “cálculo correcto” y “confianza plenamente consolidada” permitió aprender que, en dominios sensibles, la

correctitud debe demostrarse técnicamente, pero la confianza se construye también con tiempo de uso y consistencia sostenida.

11.3.3. La calidad sostenida requiere procesos, no esfuerzos puntuales

El proyecto también dejó como aprendizaje que la calidad del software no puede depender de revisiones tardías ni de esfuerzos intensivos cerca del cierre, sino que debe construirse de manera continua mediante prácticas sistemáticas. En un sistema con reglas de negocio complejas, múltiples módulos e integraciones externas, confiar solo en pruebas manuales o en revisiones informales hubiera introducido un riesgo alto de regresiones y problemas difíciles de detectar.

Por ese motivo, resultó clave sostener prácticas de aseguramiento de calidad durante todo el proyecto, incluyendo revisiones, pruebas automatizadas, métricas técnicas y criterios explícitos para evaluar mantenibilidad y robustez. Más que ver la calidad como una etapa, el equipo aprendió a tratarla como una disciplina transversal que acompaña la evolución del producto.

Este aprendizaje también tuvo una dimensión organizacional: establecer procesos estables permitió reducir dependencia de esfuerzos individuales y mejorar la previsibilidad técnica del desarrollo. En otras palabras, la calidad se volvió más sostenible cuando dejó de depender del cuidado puntual de algunas personas y pasó a estar respaldada por mecanismos repetibles, visibles y compartidos por el equipo.

11.3.4. Separar core de extensiones reduce riesgo

Una lección importante a nivel de producto y arquitectura fue comprobar el valor de diferenciar claramente entre el núcleo operativo indispensable y las funcionalidades complementarias o extendidas. En este proyecto, esa distinción permitió priorizar primero un *core* sólido, estable y útil para la operativa diaria, centrado en contratos,

cuotas, pagos y reportes base, antes de avanzar sobre componentes adicionales o más variables.

Esta estrategia tuvo un efecto positivo doble. Por un lado, permitió entregar valor temprano al cliente con una base funcional confiable. Por otro, evitó que la incorporación prematura de funcionalidades accesorias o con mayor incertidumbre comprometa la estabilidad del sistema principal. En proyectos con restricciones de tiempo y con un dominio todavía en maduración, esta separación entre “lo esencial” y “lo evolutivo” resultó especialmente valiosa.

El aprendizaje de fondo es que priorizar el *core* no implica resignar ambición, sino secuenciarla mejor. Primero hay que consolidar la columna vertebral del negocio y recien luego, sobre esa base validada, pueden incorporarse extensiones con menor riesgo y mayor claridad sobre su valor real.

11.3.5. Priorizar integraciones externas por riesgo

El proyecto también mostró que las integraciones externas deben priorizarse no solo según el “valor percibido” que aportan al usuario, sino también según el riesgo técnico y operativo que introducen. Componentes como integraciones con servicios de terceros, canales externos o dependencias con configuraciones ajenas al equipo pueden generar incertidumbres que no siempre son evidentes al estimar funcionalidad pura.

La experiencia concreta del proyecto llevó a revisar el plan inicial de *releases* y reordenar prioridades, justamente porque algunas funcionalidades con dependencias externas importantes no debían dejarse demasiado para el final. El aprendizaje fue que postergar estos componentes por considerarlos “complementarios” puede ser contraproducente si concentran riesgos de integración, habilitación o estabilidad que luego afectan al conjunto del proyecto.

11.3.6. Costos de infraestructura en la nube

Un aspecto secundario que empezó a ser una preocupación al momento de desplegar la infraestructura en AWS fue el tema de los costos, discutido en la sección 7.5.6 (Análisis y Discusión de Costos de AWS).

Esta experiencia fue valiosa ya que obligó al equipo a considerar seriamente la parte económica del proyecto, y a realizar un esfuerzo agregado por reducir los costos a un mínimo sacrificando la menor cantidad de conveniencia y trabajo razonablemente posible.

La enseñanza principal fue que una buena decisión de infraestructura no es la más sofisticada, sino la que logra un equilibrio adecuado entre robustez, mantenibilidad, costo y proyección de crecimiento

11.3.7. Explicitar limitaciones fortalece la propuesta

Finalmente, una de las lecciones más valiosas fue entender que reconocer con claridad las limitaciones de la solución no debilita el trabajo, sino que lo fortalece. A lo largo del proyecto surgieron decisiones de alcance, exclusiones y transiciones necesarias, como la convivencia temporal con planillas para ciertos datos históricos o la postergación de funcionalidades cuya implementación requería mayor madurez del dominio.

Lejos de ser un problema discursivo, explicitar estas limitaciones permitió presentar una propuesta más honesta, realista y académicamente sólida. También ayudó a distinguir entre lo efectivamente resuelto, lo que quedó encaminado y aquello que corresponde tratar como evolución futura, evitando sobreprometer resultados que no se alineaban con el estado real del producto.

Este aprendizaje fue importante tanto para la relación con el cliente como para la redacción del documento final. En ambos planos, quedó claro que una solución robusta no es la que afirma resolverlo todo, sino la que define con precisión qué problema

resuelve, bajo qué condiciones lo hace y cuáles son sus próximos pasos razonables de evolución.

En conjunto, estas lecciones aprendidas muestran que el valor del proyecto no se limitó al producto entregado, sino que también residió en el proceso de comprensión, decisión y ajuste que permitió construirlo. El trabajo realizado dejó aprendizajes sobre validación, calidad, arquitectura, priorización, infraestructura y manejo del alcance que trascienden este caso particular y resultan aplicables a otros proyectos de software con alta dependencia del dominio, necesidad de adopción real y exigencias de confiabilidad.

En ese sentido, el proyecto no solo produjo una solución útil para el cliente, sino también una experiencia formativa completa para el equipo, al enfrentarlo con problemas reales cuya resolución requirió criterio técnico, capacidad de adaptación y reflexión crítica sobre las decisiones tomadas.

11.4. Próximos pasos

A partir de la experiencia obtenida durante el desarrollo y la puesta en producción de DATIUM, se identifican las líneas de evolución que permitirán proyectar la continuidad y el crecimiento de la plataforma. Estas iniciativas se apoyan en los aprendizajes técnicos y operativos del proyecto y derivan directamente de la arquitectura, del producto construido y del proceso real de adopción por parte del cliente.

Las propuestas que se presentan a continuación no corresponden a funcionalidades pendientes del alcance original, sino a direcciones razonables de evolución que emergen del uso real del sistema. Para ordenar las iniciativas se distinguen dos horizontes de evolución: (i) acciones concretas orientadas a consolidar y optimizar la operativa en el corto y mediano plazo, y (ii) oportunidades estratégicas de largo plazo orientadas a escalabilidad, comercialización y analítica avanzada.

11.4.1. Consolidación y optimización operativa (corto y mediano plazo):

Esta línea de trabajo representa un compromiso formal asumido por el equipo con el cliente para continuar el desarrollo una vez entregada y defendida la presente tesis. Su objetivo principal es afianzar el núcleo del sistema y facilitar la transición definitiva del cliente, eliminando la dependencia de procesos manuales heredados.

- **Importación de datos históricos y migración definitiva:** Actualmente DATIUM coexiste temporalmente con las planillas históricas, gestionando únicamente los nuevos compromisos. El paso esencial es completar mecanismos robustos de importación masiva desde Excel que permitan consolidar todo el historial en la plataforma. Esto implicará diseñar y desarrollar procesos automatizados de validación, normalización y detección de inconsistencias, y establecer un flujo de trabajo colaborativo en el que el cliente revise y valide registros caso por caso. Solo una vez completado este proceso y validada la consistencia de la información será posible dejar atrás a las planillas definitivamente.
- **Evolución del registro de comprobantes y del canal WhatsApp:** Una vez consolidada la adopción del canal automatizado de WhatsApp, se prevé evolucionar la opción de “Aviso de pago” actualmente prevista en el menú del *bot*. Esta mejora permitirá que el comprador adjunte su comprobante de pago en formato PDF o imagen, de modo que el sistema lo reciba y lo asocie automáticamente al contrato correspondiente dentro de DATIUM. De esta forma, será posible sustituir gradualmente la práctica actual de registrar manualmente un identificador alfanumérico, incorporando una gestión documental integrada al flujo de cobranzas.

11.4.2. Exploración y oportunidades a largo plazo

Las siguientes iniciativas representan direcciones estratégicas que, aunque no forman parte de los compromisos inmediatos, abren posibilidades significativas de valor técnico, analítico y comercial.

- **Generalización del producto (Modelo SaaS):** La problemática asociada a la gestión de productos financiados en cuotas y a las conversiones monetarias a UYU no es exclusiva de este emprendimiento. Existe un potencial tecnológico y comercial significativo para generalizar DATIUM hacia un producto multi-inquilino (*Software as a Service*). Esto implicaría parametrizar estructuralmente las reglas de negocio y los flujos operativos para adaptarlos a otras organizaciones e incluso a distintos rubros. Para concretar esta visión sería necesario realizar un análisis de mercado exhaustivo que permita evaluar su viabilidad comercial y definir una estrategia de entrada a la industria.

- **Fortalecimiento de la auditoría y control de acceso:** A medida que crezca el volumen de usuarios o aumente la criticidad de los procesos, el esquema actual de permisos de dos niveles (administrador y usuario estándar) deberá evolucionar hacia un modelo de roles más granulares, diferenciados por funcionalidad. Asimismo, será clave implementar un mecanismo de auditoría formal para cada acción relevante del sistema, registrando el estado *before/after* de las entidades modificadas. Esta capacidad resulta indispensable para garantizar una trazabilidad completa de las operaciones críticas y facilitar eventuales auditorías contables.

- **Predicción de riesgo de impago con Machine Learning (ML):**Aprovechando la consolidación de los datos históricos de pagos y del comportamiento de la mora en la base de datos relacional del sistema, es posible construir datasets analíticos a partir de consultas y procesos de extracción. Estos datos podrían utilizarse para entrenar modelos de Machine Learning orientados a estimar la

probabilidad de retraso o impago de cada contrato, permitiendo priorizar el seguimiento de deudores y mejorar la gestión preventiva de la cobranza.

- **Automatización integral del registro de pagos:** Como evolución de largo plazo, y una vez consolidada la recepción y asociación automática de comprobantes a través del canal de WhatsApp, se identifica la oportunidad de incorporar mecanismos de procesamiento documental para asistir el registro de los pagos. A partir del archivo enviado por el comprador, el sistema podría aplicar técnicas como reconocimiento óptico de caracteres (OCR) o análisis asistido por inteligencia artificial para extraer datos relevantes del comprobante, tales como monto, fecha, moneda o número de referencia. Esta información podría utilizarse para pre completar el registro del pago dentro de la plataforma y reducir significativamente la carga administrativa, manteniendo instancias de validación antes de su confirmación definitiva. En una etapa posterior, y sujeto a resultados satisfactorios de precisión y confiabilidad, esta capacidad podría evolucionar hacia mayores niveles de automatización del flujo de cobranza.

12. Referencias bibliográficas

[1] G. T. Doran, "There's a S.M.A.R.T. way to write management's goals and objectives," Management Review, vol. 70, no. 11, pp. 35-36, nov. 1981.

[2] Laboratorio ORT Software Factory. (2013, Mar.) [Online]. Disponible: http://fi.ort.edu.uy/innovaportal/v/3393/5/fi.ort.front/inicio.html

[3] Banco Central del Uruguay, "Cotizaciones y Tasas Oficiales", mar. 2026. [Online]. Available: http://www.bcu.gub.uy/. Accessed on: Mar. 9, 2026.

[4] Atlassian, "Plataforma Cloud de Atlassian (Jira, Confluence)", feb. 2026. [Online]. Available: https://www.atlassian.com/es. Accessed on: Mar. 9, 2026.

[5] Linear Orbit, Inc., "Documentación Oficial de Linear: Guías y Flujos de Trabajo", feb. 2026. [Online]. Available: https://linear.app/docs. Accessed on: Mar. 10, 2026.

[6] Pytest Development Team, "Pytest: helps you write better programs," 2026. [Online]. Available: https://docs.pytest.org/. Accessed on: Mar. 11, 2026.

[7] HashiCorp, "Documentación Oficial de Terraform", ene. 2026. [Online]. Available: https://developer.hashicorp.com/terraform. Accessed on: Mar. 9, 2026.

[8] Project Management Institute (PMI), A Guide to the Project Management Body of Knowledge (PMBOK Guide), 7th ed. Newtown Square, PA, USA: Project Management Institute, 2021.

[9] International Agile Federation, "The Cynefin Framework", jul. 2023. [Online]. Available: https://medium.com/@internationalagilefederation/the-cynefin-framework-8e3c4711d2c3. Accessed on: Mar. 9, 2026.

[10] T. Tsumaki and T. Tamai, "Framework for matching requirements elicitation techniques to project characteristics," Software Process: Improvement and Practice, vol. 11, no. 5, pp. 505–519, sep. 2006. [Online]. Available: https://doi.org/10.1002/spip.293. Accessed on: Mar. 9, 2026.

[11] Google, "The Golden Path: Definition and methodology for user-centric design," Google Design Resources, 2024. [Online]. Available: https://design.google/resources/. Accessed on: Mar. 9, 2026.

[12] S. B. Merriam and E. J. Tisdell, Qualitative Research: A Guide to Design and Implementation, 4th ed. San Francisco, CA, USA: Jossey-Bass, 2015.

[13] J. Nielsen, "10 Usability Heuristics for User Interface Design," Nielsen Norman Group, nov. 2020. [Online]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/. Accessed on: Mar. 9, 2026.

[14] Cucumber, "Gherkin Syntax Reference," Cucumber Documentation, 2024. [Online]. Available: https://cucumber.io/docs/gherkin/. Accessed on: Mar. 10, 2026.

[15] Mountain Goat Software, "Planning Poker," 2023. [Online]. Available: https://www.mountaingoatsoftware.com/agile/planning-poker. Accessed on: Mar. 10, 2026.

[16] Scrum Alliance, "Definition of Ready," Resource Library, 2023. [Online]. Available: https://www.scrumalliance.org/resources/definition-of-ready. Accessed on: Mar. 10, 2026.

[17] Agile Business Consortium, "The DSDM Agile Project Framework: MoSCoW Prioritisation," Agile Business Consortium Resources, 2014. [Online]. Available: https://www.agilebusiness.org/resource/moscow-prioritisation.html. Accessed on: Mar. 10, 2026.

[18] International Organization for Standardization (ISO), "Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models," ISO/IEC Standard 25010:2011, 2011. [Online]. Available: https://www.iso.org/standard/35733.html. Accessed on: Mar. 9, 2026.

[19] ESIC, "Escala de Likert: qué es y cómo utilizarla en encuestas," 2026. [Online]. Available: https://www.esic.edu/rethink/comercial-y-ventas/escala-de-likert-que-es-y-como-usarla-en-encuestas-c. Accessed on: Mar. 11, 2026.

[20] K. Schwaber and J. Sutherland, "The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game," Scrum.org, nov. 2020. [Online]. Available: https://scrumguides.org/scrum-guide.html. Accessed on: Mar. 10, 2026.

[21] Amazon Web Services (AWS), "Documentación de Amazon CloudWatch: Guía del usuario", feb. 2026. [Online]. Available: https://docs.aws.amazon.com/cloudwatch/. Accessed on: Mar. 10, 2026.

[22] New Relic, "New Relic APM Solutions and Documentation", ene. 2026. [Online]. Available: https://docs.newrelic.com/. Accessed on: Mar. 10, 2026.

[23] S. Ramírez, "FastAPI Documentation," 2026. [Online]. Available: https://fastapi.tiangolo.com. Accessed on: Mar. 10, 2026.

[24] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, Documenting Software Architectures: Views and Beyond, 2nd ed. Boston, MA, USA: Addison-Wesley, 2010.

[25] M. S. Mikowski and J. C. Powell, Single Page Web Applications: JavaScript end-to-end. Shelter Island, NY, USA: Manning Publications, 2013.

[26] Amazon Web Services (AWS), "Documentación de Servicios (EC2, S3, API Gateway)", mar. 2026. [Online]. Available: https://aws.amazon.com/es/documentation/. Accessed on: Mar. 9, 2026.

[27] Conventional Commits, "Conventional Commits 1.0.0 Specification," 2022. [Online]. Available: https://www.conventionalcommits.org/en/v1.0.0/. Accessed on: Mar. 9, 2026.

[28] B. Frost, Atomic Design. Massachusetts, USA: Brad Frost, 2016. [Online]. Available: https://atomicdesign.bradfrost.com/. Accessed on: Mar. 9, 2026.

[29] M. Benitez, "py-bcu: Librería para el consumo de los servicios web del Banco Central del Uruguay," 2024. [Online]. Available: https://github.com/m-benitez/py-bcu. Accessed on: Mar. 10, 2026.

[30] Meta, "Documentación de WhatsApp Cloud API", feb. 2026. [Online]. Available: https://developers.facebook.com/docs/whatsapp/cloud-api. Accessed on: Mar. 9, 2026.

[31] OpenAI, "Documentación de APIs y Modelos", mar. 2026. [Online]. Available: https://platform.openai.com/docs/. Accessed on: Mar. 9, 2026.

[32] ConfigCat, "Documentación Oficial de ConfigCat", feb. 2026. [Online]. Available: https://configcat.com/docs/. Accessed on: Mar. 9, 2026.

[33] TanStack, "TanStack Query: Powerful asynchronous state management for TS/JS," 2026. [Online]. Available: https://tanstack.com/query/latest/docs/framework/react/overview. Accessed on: Mar. 10, 2026.

[34] Google, "Guía de Google Sign-In para Servidores (OAuth 2.0)", 2025. [Online]. Available: https://developers.google.com/identity/sign-in/web/server-side-flow. Accessed on: Mar. 9, 2026.

[35] S. Colvin et al., "Pydantic: Data validation and settings management using Python type hints," 2026. [Online]. Available: https://docs.pydantic.dev/. Accessed on: Mar. 10, 2026.

[36] V. Driessen, "A successful Git branching model," nvie.com, ene. 2010. [Online]. Available: https://nvie.com/posts/a-successful-git-branching-model/. Accessed on: Mar. 9, 2026.

[37] TechEmpower, "Web Framework Benchmarks - Round 22," TechEmpower Blog, may. 2023. [Online]. Available: https://www.techempower.com/benchmarks/. Accessed on: Mar. 9, 2026.

[38] M. Nygard, "Documenting Architecture Decisions," nypower.org, nov. 2011. [Online]. Available: https://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions. Accessed on: Mar. 10, 2026.

[39] Oracle, "Documentación Oficial de MySQL", ene. 2026. [Online]. Available: https://dev.mysql.com/doc/. Accessed on: Mar. 9, 2026.

[40] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "*Zero Trust* Architecture," NIST Special Publication 800-207, Aug. 2020. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-207. Accessed on: Mar. 10, 2026.

[41] OWASP Foundation, "OWASP API Security Top 10 2023," OWASP Project, jun. 2023. [Online]. Available: https://owasp.org/www-project-api-security/. Accessed on: Mar. 9, 2026.

[42] M. Bayer, "SQLAlchemy: The Python SQL Toolkit and Object Relational Mapper," 2026. [Online]. Available: https://docs.sqlalchemy.org/. Accessed on: Mar. 10, 2026.

[43] Snehagiranje, "Unveiling the Magic: Understanding Mock and MagicMock in Python," Medium, Jun. 18, 2024. [Online]. Available:

https://medium.com/@snehagiranje05/unveiling-the-magic-understanding-mock-and-magicmock-in-python-ecadf1f1013c. Accessed: Mar. 11, 2026.

[44] ESLint Foundation, "ESLint: Pluggable JavaScript linter", 2026. [Online]. Available: https://eslint.org/. Accessed on: Mar. 11, 2026.

[45] Prettier, "Prettier: An opinionated code formatter", 2026. [Online]. Available: https://prettier.io/. Accessed on: Mar. 11, 2026.

[46] OpenAI, "ChatGPT: Optimizing Language Models for Dialogue", 2026. [Online]. Available: https://openai.com/chatgpt. Accessed on: Mar. 11, 2026.

[47] G. van Rossum, B. Warsaw, and A. Coghlan, "PEP 8 – Style Guide for Python Code," Python Enhancement Proposals, PEP 8, Jul. 2001. [Online]. Available: https://peps.python.org/pep-0008/. Accessed: Mar. 11, 2026.

[48] R. C. Martin, "Design Principles and Design Patterns", 2000. [Online]. Available: https://web.archive.org/web/20150906155800/http://www.objectmentor.com/resources/articles/oodmetrc.pdf. Accessed on: Mar. 9, 2026.

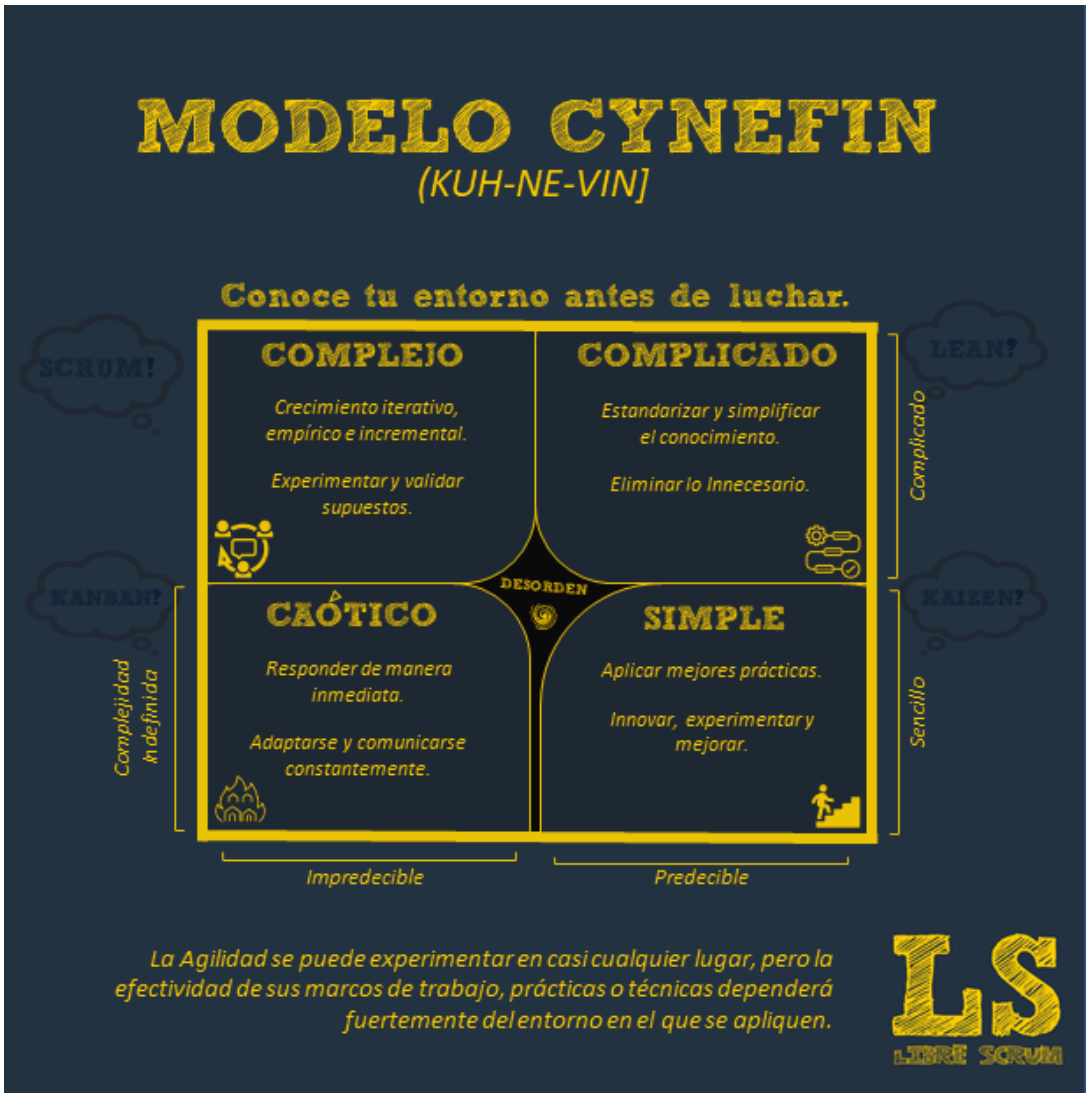
[49] F. Woss, "Documentación Oficial de Radon: A Python Code Metric Tool", 2024. [Online]. Available: https://radon.readthedocs.io/. Accessed on: Mar. 9, 2026.

[50] F. Woss, "Radon Documentation: Complexity Granularity and Grades," 2024. [Online]. Available: https://radon.readthedocs.io/en/latest/api.html#module-radon.complexity. Accessed on: Mar. 11, 2026.

[51] Microsoft, "Code metrics - Maintainability index range and meaning - Visual Studio (Windows)", dic. 2025. [Online]. Available: https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-maintainability-index-range-and-meaning. Accessed on: Mar. 9, 2026.

13. Anexos

13.1. Anexo 1 - Matriz de Cynefin

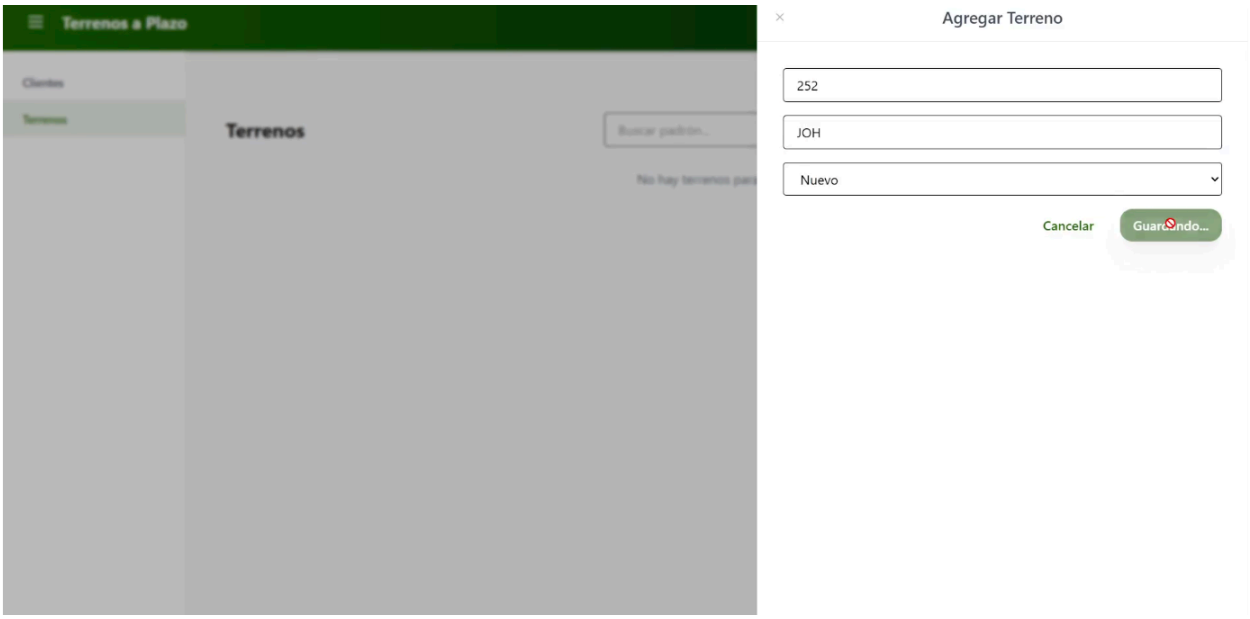


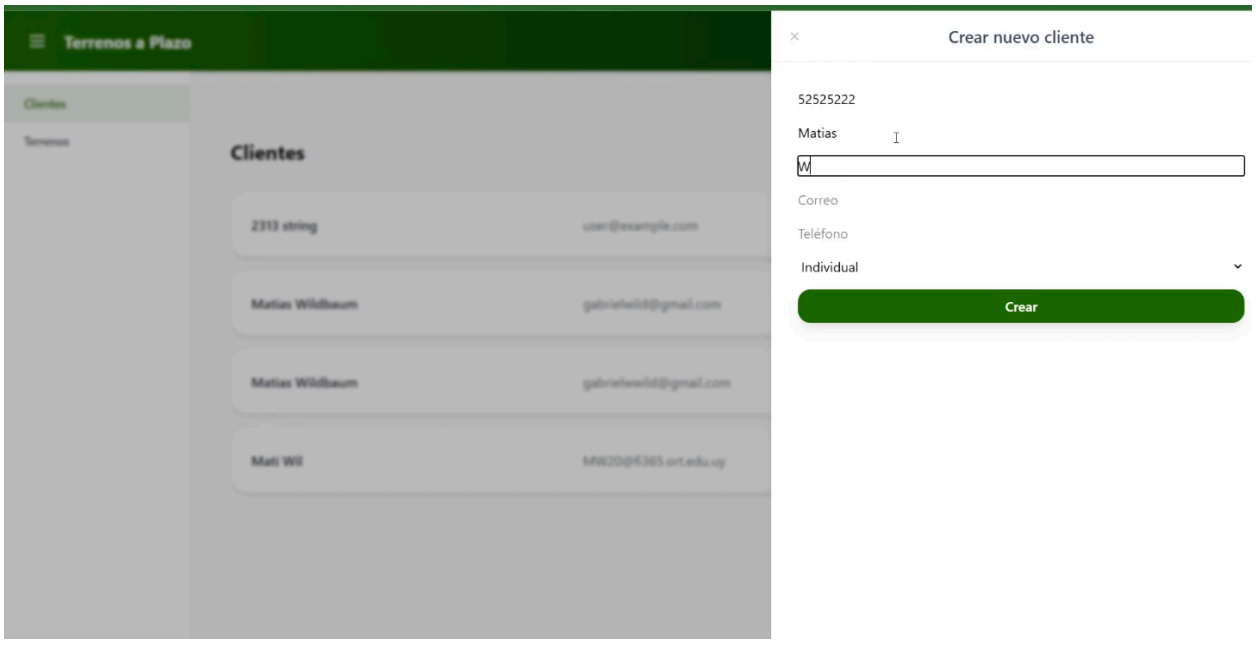
13.2. Anexo 2 - Planillas de Excel Proporcionadas

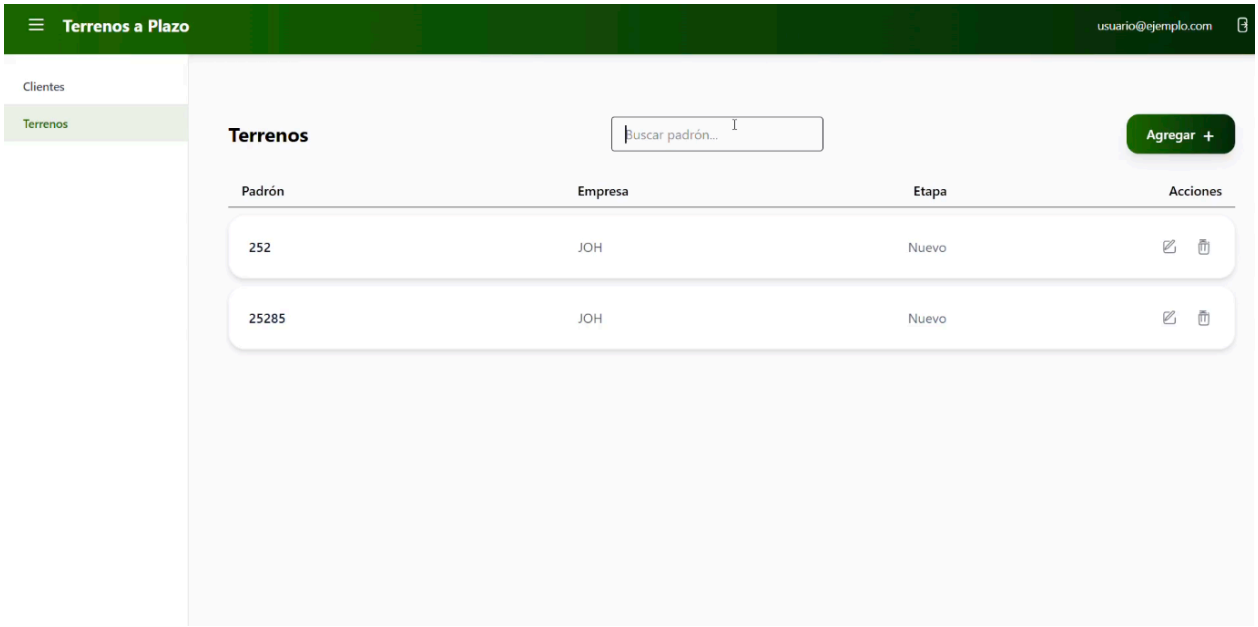
	Cuota	Fecha	Monto UR	Valor UR	Monto $U	hasta	días	tasa	Mora UR	Mora $	Total UR	Total $	Pago $	Pago sin mora	Pago UR	A favor UR	Comprobante
	Entrega			-	$U 200.000,00								$U 200.000,00		-	-	
2022	1	20/6/2022	16	1.428,01	$U 22.848,16						16,0000	$U 22.848,16	$U 22.860,00	$U 22.860,00	16,0083	0,0083	CRE CAMBIOSOP..217
	2	20/7/2022	16	1.445,81	$U 23.132,96						15,9917	$U 23.120,97	$U 23.150,00	$U 23.150,00	16,0118	0,0201	CRE CAMBIOSOP..754
	3	20/8/2022	16	1.446,76	$U 23.148,16						15,9799	$U 23.119,11	$U 23.150,00	$U 23.150,00	16,0013	0,0213	CRE CAMBIOSOP..619
	4	20/9/2022	16	1.497,30	$U 23.796,80						15,9787	$U 23.765,05	$U 23.150,00	$U 23.150,00	15,5651	-0,4135	CRE CAMBIOSOP..130
	5	20/10/2022	16	1.494,32	$U 23.909,12						16,4135	$U 24.527,07	$U 24.600,00	$U 24.600,00	16,4623	0,0488	CRE CAMBIOSOP..635
	6	20/11/2022	16	1.496,69	$U 23.947,04						15,9512	$U 23.874,00	$U 24.000,00	$U 24.000,00	16,0354	0,0842	CRE CAMBIOSOP..154
	7	20/12/2022	16	1.498,67	$U 23.978,72						15,9158	$U 23.892,55	$U 24.000,00	$U 24.000,00	16,0142	0,0984	CRE CAMBIOSOP..708
	8	20/1/2023	16	1.501,26	$U 24.020,16						15,9016	$U 23.872,45	$U 24.100,00	$U 24.100,00	16,0532	0,1516	CRE CAMBIOSOP..293
	9	20/2/2023	16	1.502,25	$U 24.036,00						15,8484	$U 23.808,30	$U 24.100,00	$U 24.100,00	16,0426	0,1942	CRE CAMBIOSOP..912
	10	20/3/2023	16	1.579,57	$U 25.273,12						15,8058	$U 24.966,41	$U 25.300,00	$U 25.300,00	16,0170	0,2112	CRE CAMBIOSOP..368
	11	20/4/2023	16	1.584,25	$U 25.348,00						15,7888	$U 25.013,42	$U 25.400,00	$U 25.400,00	16,0328	0,2440	CRE CAMBIOSOP..359
	12	20/5/2023	16	1.586,04	$U 25.376,64						15,7560	$U 24.989,63	$U 25.500,00	$U 25.500,00	16,0778	0,3218	CRE CAMBIOSOP..846
2023	13	20/6/2023	16	1.596,82	$U 25.549,12						15,6782	$U 25.035,28	$U 25.600,00	$U 25.600,00	16,0319	0,3537	CRE CAMBIOSOP..393
	14	20/7/2023	16	1.597,15	$U 25.554,40						15,6463	$U 24.989,56	$U 25.600,00	$U 25.600,00	16,0286	0,3822	CRE CAMBIOSOP..898
	15	20/8/2023	16	1.597,62	$U 25.561,92						15,6178	$U 24.961,30	$U 25.600,00	$U 25.600,00	16,0238	0,4060	CRE CAMBIOSOP..434
	16	20/9/2023	16	1.620,91	$U 25.934,56						15,5940	$U 25.276,40	$U 25.950,00	$U 25.950,00	16,0095	0,4156	CRE CAMBIOSOP..960
	17	20/10/2023	16	1.626,80	$U 26.028,80						15,5844	$U 25.352,76	$U 26.030,00	$U 26.030,00	16,0007	0,4163	CRE CAMBIOSOP..468
	18	20/11/2023	16	1.629,65	$U 26.074,40						15,5837	$U 25.395,97	$U 26.030,00	$U 26.030,00	15,9728	0,3891	CRE CAMBIOSOP..046
	19	20/12/2023	16	1.634,00	$U 26.144,00						15,6109	$U 25.508,28	$U 26.150,00	$U 26.150,00	16,0037	0,3927	CRE CAMBIOSOP..681
	20	20/1/2024	16	1.638,35	$U 26.213,60						15,6073	$U 25.570,17	$U 26.200,00	$U 26.200,00	15,9917	0,3844	CRE CAMBIOSOP..216
	21	20/2/2024	16	1.642,33	$U 26.277,28						15,6156	$U 25.645,92	$U 26.300,00	$U 26.300,00	16,0138	0,3983	CRE CAMBIOSOP..768
	22	20/3/2024	16	1.717,20	$U 27.475,20						15,6017	$U 26.791,30	$U 26.400,00	$U 26.400,00	15,3739	-0,2279	CRE CAMBIOSOP..342
	23	20/4/2024	16	1.719,57	$U 27.513,12						16,2279	$U 27.904,96	$U 28.000,00	$U 28.000,00	16,2831	0,0553	CRE CAMBIOSOP..848
	24	20/5/2024	16	1.719,98	$U 27.519,68						15,9447	$U 27.424,62	$U 27.504,00	$U 27.504,00	15,9909	0,0462	CRE CAMBIOSOP..415
	25	20/6/2024	16	1.724,29	$U 27.588,64	1/7/2024	11	5,6160%	0,0270	$U 46,56	15,9808	$U 27.555,62	$U 27.700,00	$U 27.700,00	$U 27.553,44	16,0646	0,0837	CRE CAMBIOSOP..048
	26	20/7/2024	16	1.724,69	$U 27.595,04	28/7/2024	13	6,1920%	0,0351	$U 60,54	15,9514	$U 27.511,16	$U 27.600,00	$U 27.600,00	$U 27.539,46	16,0029	0,0515	CRE CAMBIOSOP..687
2024	27	20/8/2024	16	1.740,23	$U 27.843,68	2/9/2024	13	6,7680%	0,0384	$U 66,90	15,9869	$U 27.820,94	$U 27.600,00	$U 27.600,00	$U 27.533,10	15,8600	-0,1270	CRE CAMBIOSOP..220
	28	20/9/2024	16	1.740,23	$U 27.843,68	1/10/2024	11	7,3080%	0,0355	$U 61,81	16,1625	$U 28.126,43	$U 28.121,00	$U 28.121,00	$U 28.059,19	16,1594	-0,0031	CRE CAMBIOSOP..824
	29	20/10/2024	16	1.742,93	$U 27.886,88	11/11/2024	22	7,3080%	0,0705	$U 122,86	16,0736	$U 28.015,18	$U 27.850,00	$U 27.850,00	$U 27.727,14	15,9788	-0,0948	CRE CAMBIOSOP..612
	30	20/11/2024	16	1.742,93	$U 27.886,88						16,0948	$U 28.052,06	$U 27.880,00	$U 27.880,00	15,9961	-0,0987	CRE CAMBIOSOP..944
	31	20/12/2024	16	1.744,40	$U 27.910,40	2/1/2025	13	7,3080%	0,0419	$U 73,09	16,1406	$U 28.155,70	$U 27.908,00	$U 27.908,00	$U 27.834,91	15,9986	-0,1420	CRE CAMBIOSOP..635
	32	20/1/2025	16	1.819,41	$U 29.110,56	28/4/2025	98	7,3080%	0,3167	$U 576,26	16,4587	$U 29.945,18	$U 30.000,00	$U 30.000,00	$U 29.423,74	$U 16,49	0,0301	CRE CAMBIOSOP..784
	33	20/2/2025	16	1.827,85	$U 29.245,60	2/6/2025	102	7,3080%	0,3261	$U 596,14	16,2960	$U 29.786,66	$U 29.700,00	$U 29.700,00	$U 29.103,86	$U 16,25	-0,0474	CRE CAMBIOSOP..467
	34	20/3/2025	16	1.829,02	$U 29.264,32	31/8/2025	164	7,3080%	0,5269	$U 963,77	16,5743	$U 30.314,81						
	35	20/4/2025	16	1.829,02	$U 29.264,32	12/8/2025	114	7,3080%	0,3652	$U 667,96	16,3652	$U 29.932,28						
	36	20/5/2025	16	1.829,02	$U 29.264,32	13/8/2025	85	7,3080%	0,2723	$U 498,04	16,2723	$U 29.762,96						
	37	20/6/2025	16	1.829,02	$U 29.264,32	14/8/2025	55	7,3080%	0,1762	$U 322,26	16,1762	$U 29.586,58						
2025	38	20/7/2025	16	1.829,02	$U 29.264,32	15/8/2025	26	7,3080%	0,0833	$U 152,34	16,0833	$U 29.416,66						
	39	20/8/2025	16	1.829,02	$U 29.264,32	15/8/2025	-5	7,3080%	-0,0160	-$U 29,30	15,9840	$U 29.235,02						
	40	20/9/2025	16														
	41	20/10/2025	16														
	42	20/11/2025	16														
	43	20/12/2025	16														
	44	20/1/2026	16														
	45	20/2/2026	16														
	46	20/3/2026	16														
	47	20/4/2026	16														
	48	20/5/2026	16														
	49	20/6/2026	16														

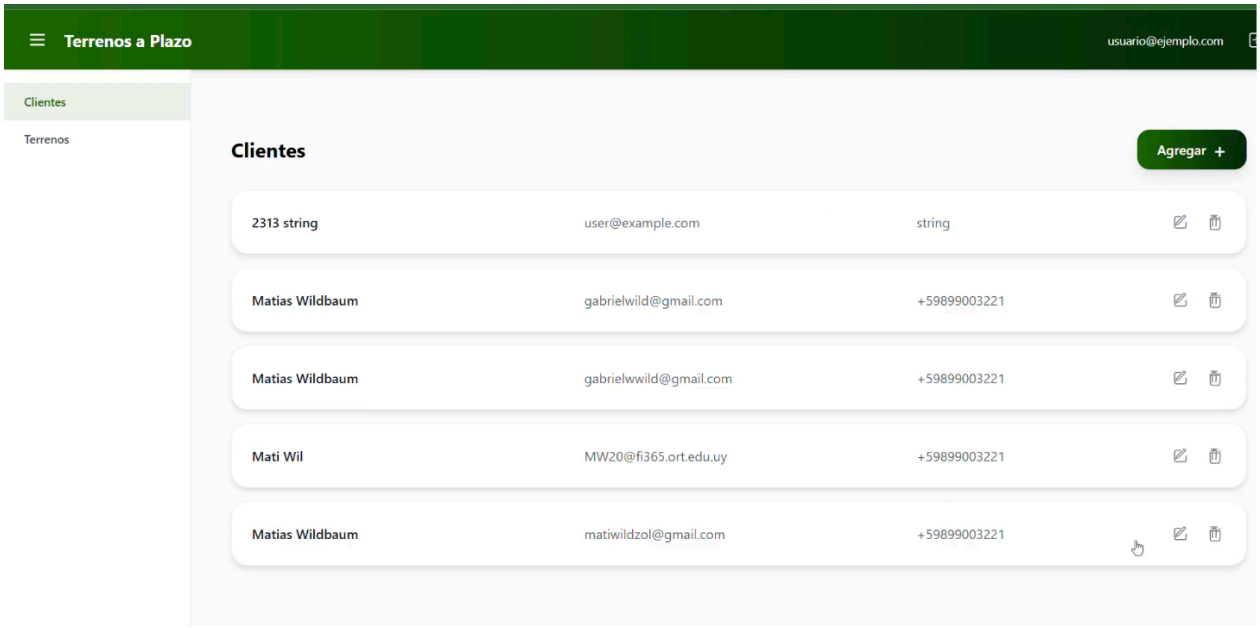
13.3. Anexo 3 - Prototipos baja fidelidad

13.4. Anexo 4 - Prototipos alta fidelidad






Terrenos a Plazo
usuario@ejemplo.com
Clientes
Terrenos
Terrenos
Buscar padrón...
Agregar +
Padrón
Empresa
Etapa
Acciones
252
JOH
Nuevo
25285
JOH
Nuevo


Terrenos a Plazo
usuario@ejemplo.com
Clientes
Terrenos
Clientes
Agregar +
2313 string
user@example.com
string
Matias Wildbaum
gabrielwild@gmail.com
+59899003221
Matias Wildbaum
gabrielwwild@gmail.com
+59899003221
Mati Wil
MW20@fi365.ort.edu.uy
+59899003221
Matias Wildbaum
matiwildzol@gmail.com
+59899003221

13.5. Anexo 5 - Ejemplo de Historia de usuario

13.6. Anexo 6 – Encuestas de satisfacción

Luego de la entrega de cada *release* se realizaron encuestas de satisfacción para evaluar la experiencia de uso del sistema en condiciones reales de operación.

Las preguntas se organizaron en distintas secciones orientadas a evaluar:

- uso y adopción del sistema
- experiencia de uso y usabilidad
- utilidad de las funcionalidades entregadas
- nivel de satisfacción general
- comentarios y sugerencias abiertas

Las siguientes subsecciones presentan el detalle de las preguntas realizadas y las respuestas obtenidas.

13.6.1. Anexo 6.1 – Encuesta de satisfacción (Release 1)

SECCIÓN 1 - Uso y adopción del sistema (ítems 1–5)

(Items respondidos en escala 1–5: 1 = totalmente en desacuerdo, 5 = totalmente de acuerdo)

1. Puedo realizar mis tareas diarias en el sistema sin pedir ayuda - Respuesta: 5
2. Registrar un pago me lleva menos tiempo que antes - Respuesta: 5
3. La generación automática de cuotas me ahorra trabajo manual - Respuesta: 5
4. La información del contrato, pagos y cuotas es clara y fácil de encontrar - Respuesta: 5

SECCIÓN 2 - Experiencia y usabilidad

5. Los reportes generados por el sistema son útiles para mi trabajo diario.
   - Respuesta: 5
6. El sistema responde rápido y no siento demoras innecesarias.
   - Respuesta: 5
7. Me siento confiado/a con la exactitud de los cálculos (cuotas, intereses, moras).
   - Respuesta: 4
8. El sistema me resulta más confiable que las planillas Excel anteriores.
   - Respuesta: 5
9. La interfaz es sencilla e intuitiva
   - Respuesta: 5
10. Los flujos principales (crear contrato, registrar pago) son fáciles de seguir.
    - Respuesta: 5
11. Los cálculos del sistema coinciden con los controles que realizamos manualmente.
    - Respuesta: 5

SECCIÓN 3 - Funcionalidades entregadas

12. ¿Qué funcionalidades del *Release* 1 utilizaste?
    - Alta de contrato - (marcado)
    - Registro de pagos - (marcado)
    - Cálculo automático de mora - (marcado)
    - Todavía no usé ninguna

Nivel de satisfacción

13. En una escala del 1 al 10, ¿qué tan satisfecho/a estás con el sistema hasta ahora?
    - Respuesta: 8
14. En una escala del 1 al 10, ¿qué tan probable es que quieras seguir usándolo a largo plazo?
    - Respuesta: 8

***Feedback* abierto**

¿Qué fue lo que más te gustó del sistema?

- “Eliminar el error manual de los cálculos”.

¿Qué te resultó más difícil o confuso?

- “Nada”.

¿Si pudieras cambiar una sola cosa del sistema?

- “No cambiaría, seguiría sumando funcionalidades”.

¿Qué funcionalidad te gustaría para el próximo *release*?

- “Reportes”.

13.6.2. Anexo 6.2 – Encuesta de satisfacción (Release 2)

Preguntas

La estructura de la encuesta del *Release* 2 mantuvo las mismas secciones que la del *Release* 1, evaluando:

- uso del sistema
- experiencia de uso
- funcionalidades entregadas
- satisfacción general
- *feedback* abierto

SECCIÓN 1 - Uso y adopción del sistema (ítems 1–5)

(Items respondidos en escala 1–5: 1 = totalmente en desacuerdo, 5 = totalmente de acuerdo)

15. Puedo realizar mis tareas diarias en el sistema sin pedir ayuda
    - Respuesta: 5
16. Registrar un pago me lleva menos tiempo que antes
    - Respuesta: 5
17. La generación automática de cuotas me ahorra trabajo manual
    - Respuesta: 5
18. La información del contrato, pagos y cuotas es clara y fácil de encontrar - Respuesta: 5

SECCIÓN 2 - Experiencia y usabilidad

19. Los reportes generados por el sistema son útiles para mi trabajo diario.
    - Respuesta: 5
20. El sistema responde rápido y no siento demoras innecesarias.
    - Respuesta: 5

21. Me siento confiado/a con la exactitud de los cálculos (cuotas, intereses, moras).
    - Respuesta: 5
22. El sistema me resulta más confiable que las planillas Excel anteriores.
    - Respuesta: 5
23. La interfaz es sencilla e intuitiva
    - Respuesta: 5
24. Los flujos principales (crear contrato, registrar pago) son fáciles de seguir.
    - Respuesta: 5
25. Los cálculos del sistema coinciden con los controles que realizamos manualmente.
    - Respuesta: 5

SECCIÓN 3 - Funcionalidades entregadas

26. ¿Qué funcionalidades del *Release* 2 utilizaste?
    - Alta de contrato - *(marcado)*
    - Registro de pagos - *(marcado)*
    - Cálculo automático de mora - *(marcado)*
    - Reportes - *(marcado)*
    - *Dashboard* - *(marcado)*
    - *Chatbot* - *(marcado)*
    - *Bot* de WhatsApp - *(marcado)*

Nivel de satisfacción

27. En una escala del 1 al 10, ¿qué tan satisfecho/a estás con el sistema hasta ahora?
    - Respuesta: 9
28. En una escala del 1 al 10, ¿qué tan probable es que quieras seguir usándolo a largo plazo?
    - Respuesta: 9

Respuestas

Comentarios destacados:

¿Qué fue lo que más te gustó del sistema?

- “Simpleza de uso”.

¿Qué te resultó más difícil o confuso?

- “Nada”.

¿Cambiarías algo del sistema?

- “Aún no lo he usado suficiente”.

¿Hay fallas o comportamientos inesperados?

- “Por el momento nada”.

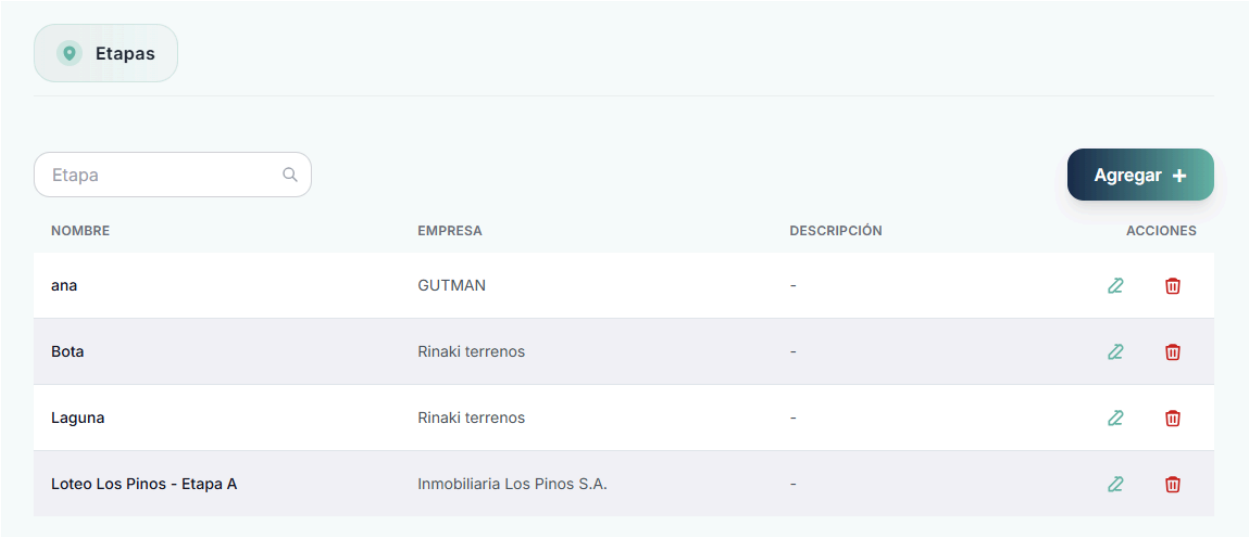
13.7. Anexo 7 - Capturas de pantalla del sistema implementado

Las capturas incluidas en este anexo tienen como objetivo principal ilustrar visualmente las funcionalidades implementadas en el sistema. A modo complementario, se incluyen comentarios debajo de imágenes seleccionadas para evidenciar la aplicación práctica de los criterios de usabilidad definidos en el RNF-02, evaluados mediante las heurísticas de Nielsen.

Cabe destacar que la Heurística de Nielsen #2 (Coincidencia entre el sistema y el mundo real) se aplica de forma transversal en todas las pantallas, ya que la interfaz utiliza exclusivamente los conceptos propios del dominio del cliente (como padrón, cuota, mora y prórroga), por lo que su mención no se reitera en cada captura individual.

13.7.1. Anexo 7.1 – Gestión de entidades maestras (RF-01, RF-02, RF-03)

Capturas de las vistas de listado, filtros y formularios de alta/edición de empresas, etapas, terrenos y compradores



Empresas

Nombre, RUT o descrip...

Agregar +

NOMBRE	RUT	DESCRIPCIÓN	ACCIONES
Inmobiliaria Los Pinos S.A.	RUT-TEST-0002	Empresa inmobiliaria - Los Pinos	
GUTMAN	1111	-	
Rinaki terrenos	11155555956565	-	

Terrenos

Padrón

Padrón

Empresa

Todas

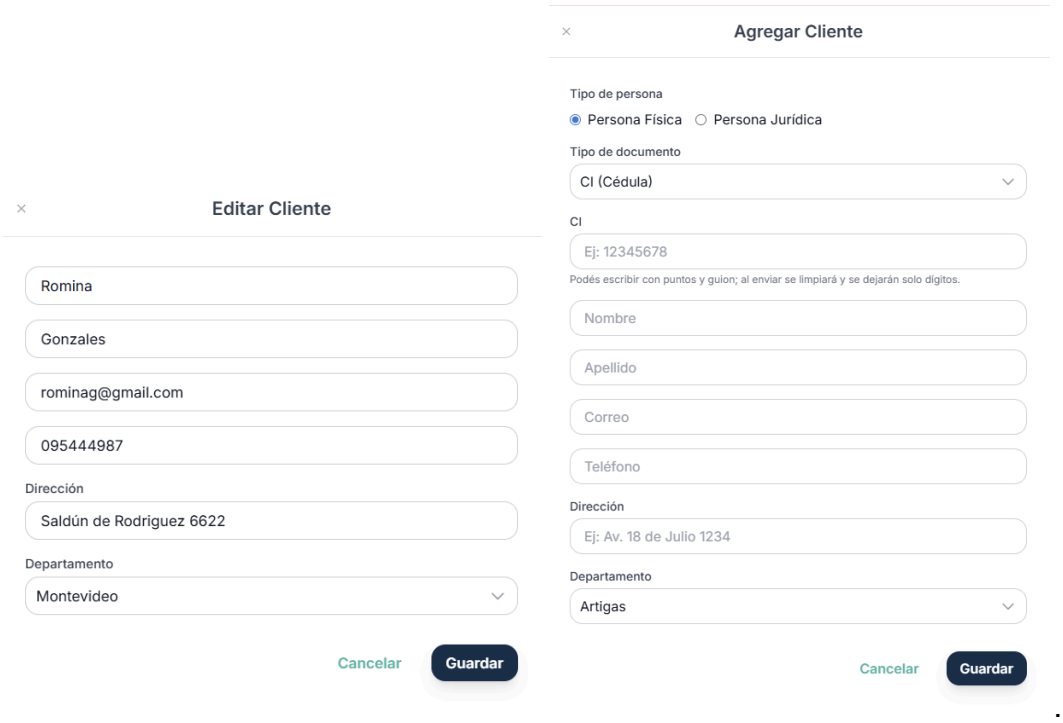
Etapa

Todas

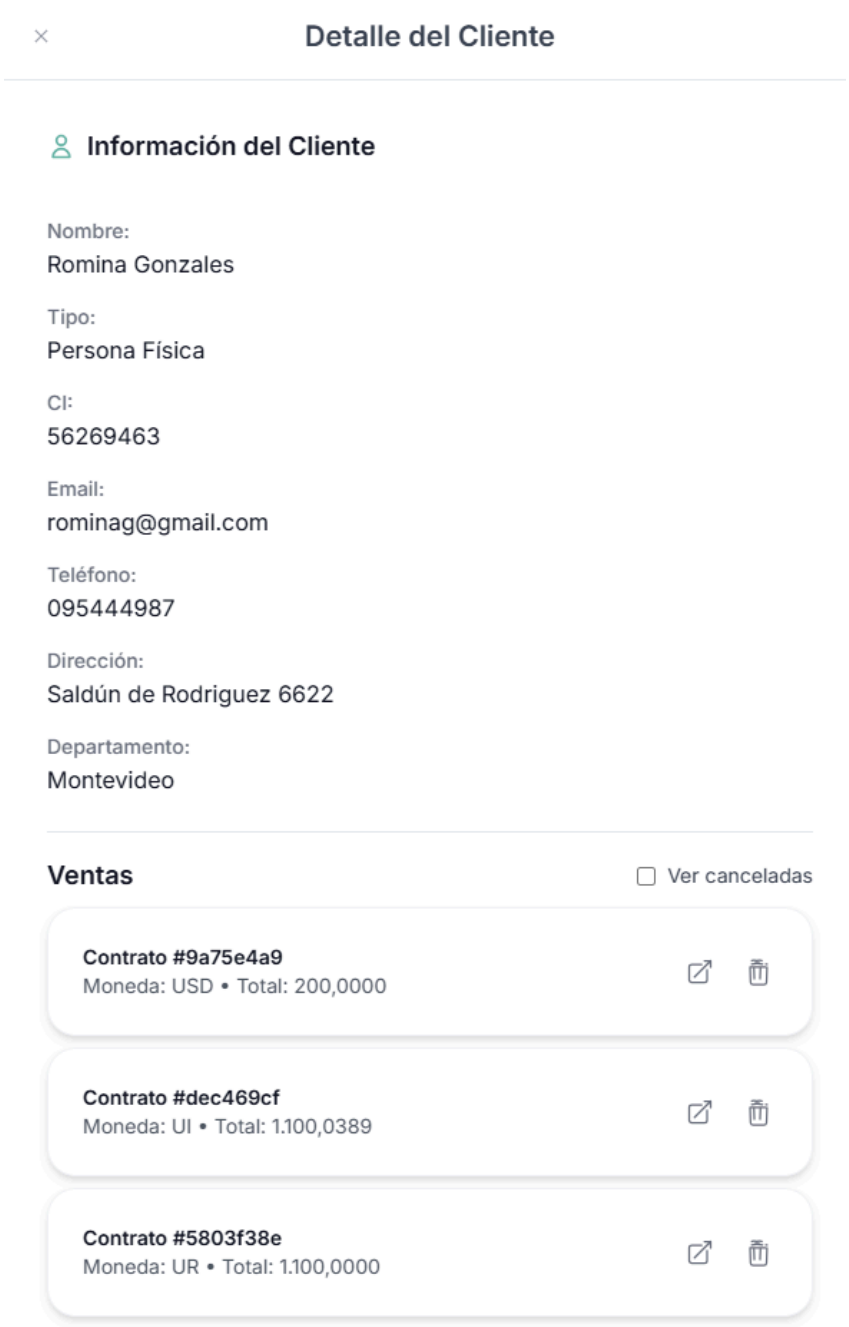
Agregar +

PADRÓN	ETAPA	VENDIDO	ACCIONES
0	ana	Sí	
1	Loteo Los Pinos - Etapa A	Sí	
11	ana	Sí	
22	Loteo Los Pinos - Etapa A	Sí	
115	ana	Sí	
667	ana	Sí	
2002	Loteo Los Pinos - Etapa A	Sí	
12345	Laguna	Sí	
66632	Bota	Sí	
996658	Laguna	Sí	

Aplicación de Heurística de Nielsen #7 (Flexibilidad y eficiencia de uso): La vista incorpora filtros combinables (por fecha, padrón, moneda, etapa, etc.) para adaptar rápidamente la información a la tarea en curso.



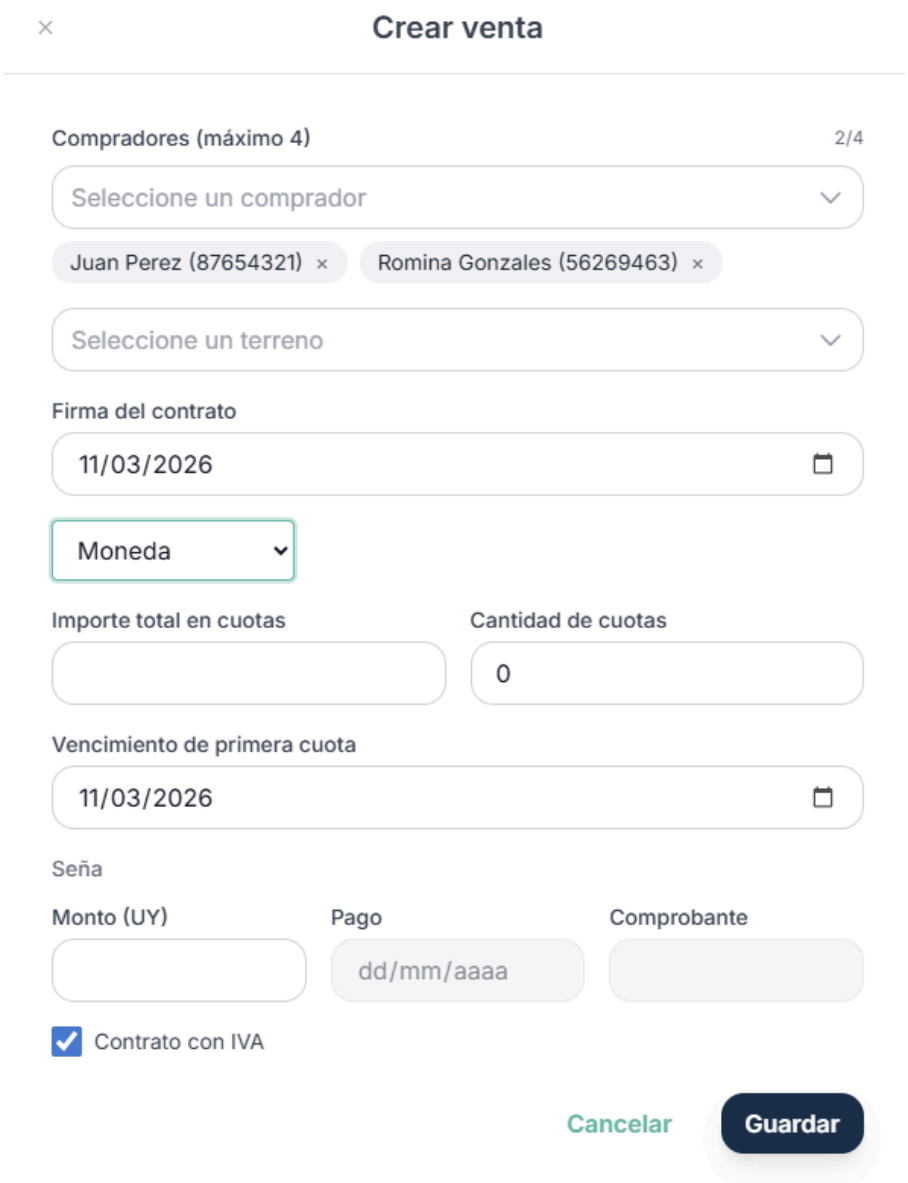
Aplicación de Heurística de Nielsen #5 (Prevención de errores): El sistema deshabilita la confirmación hasta completar los campos obligatorios y valida los datos ingresados antes de ejecutar la acción.



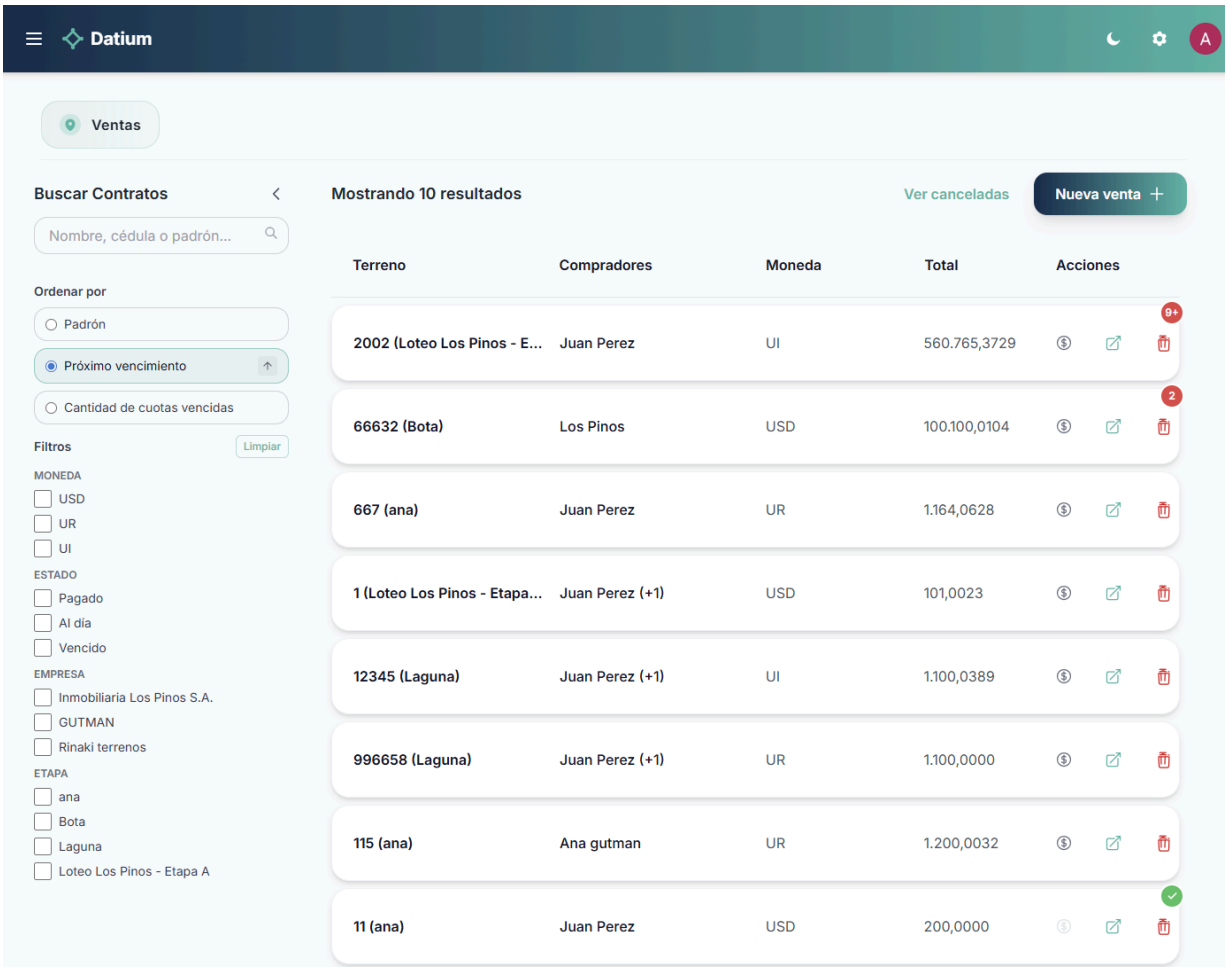
Aplicación de Heurística de Nielsen #6 (Reconocimiento antes que recuerdo): en la sidebar de detalle del cliente se listan sus contratos vendidos y cancelados a modo de agrupar esta información y no obligar a recordarla o buscarla individualmente.

13.7.2. Anexo 7.2 – Ventas, contratos y cuotas (RF-04, RF-05):

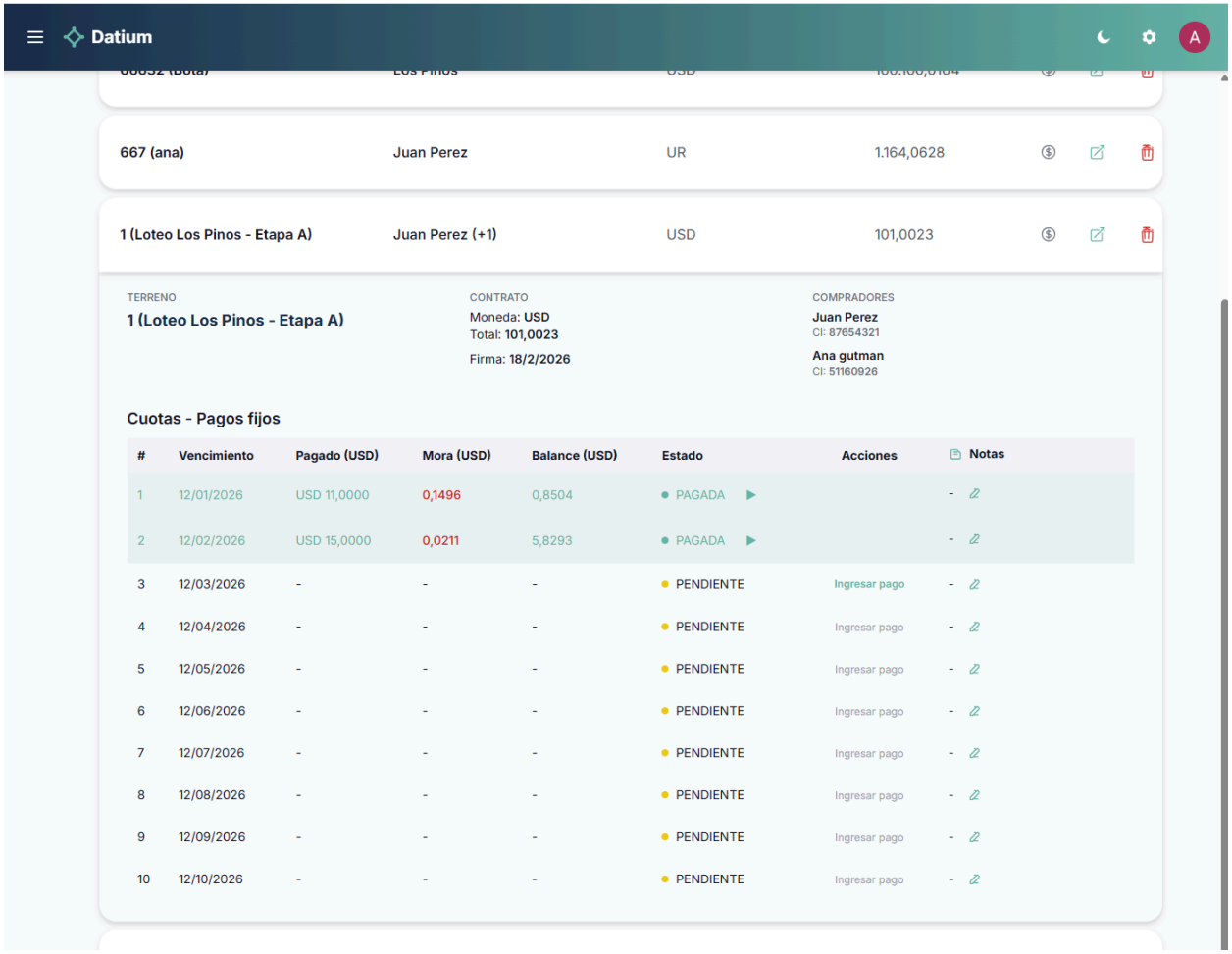
Capturas del flujo de creación de un contrato (evidenciando la capacidad de asociar hasta cuatro compradores), la vista del detalle del contrato generado y el cronograma de cuotas automático renderizado en la interfaz.



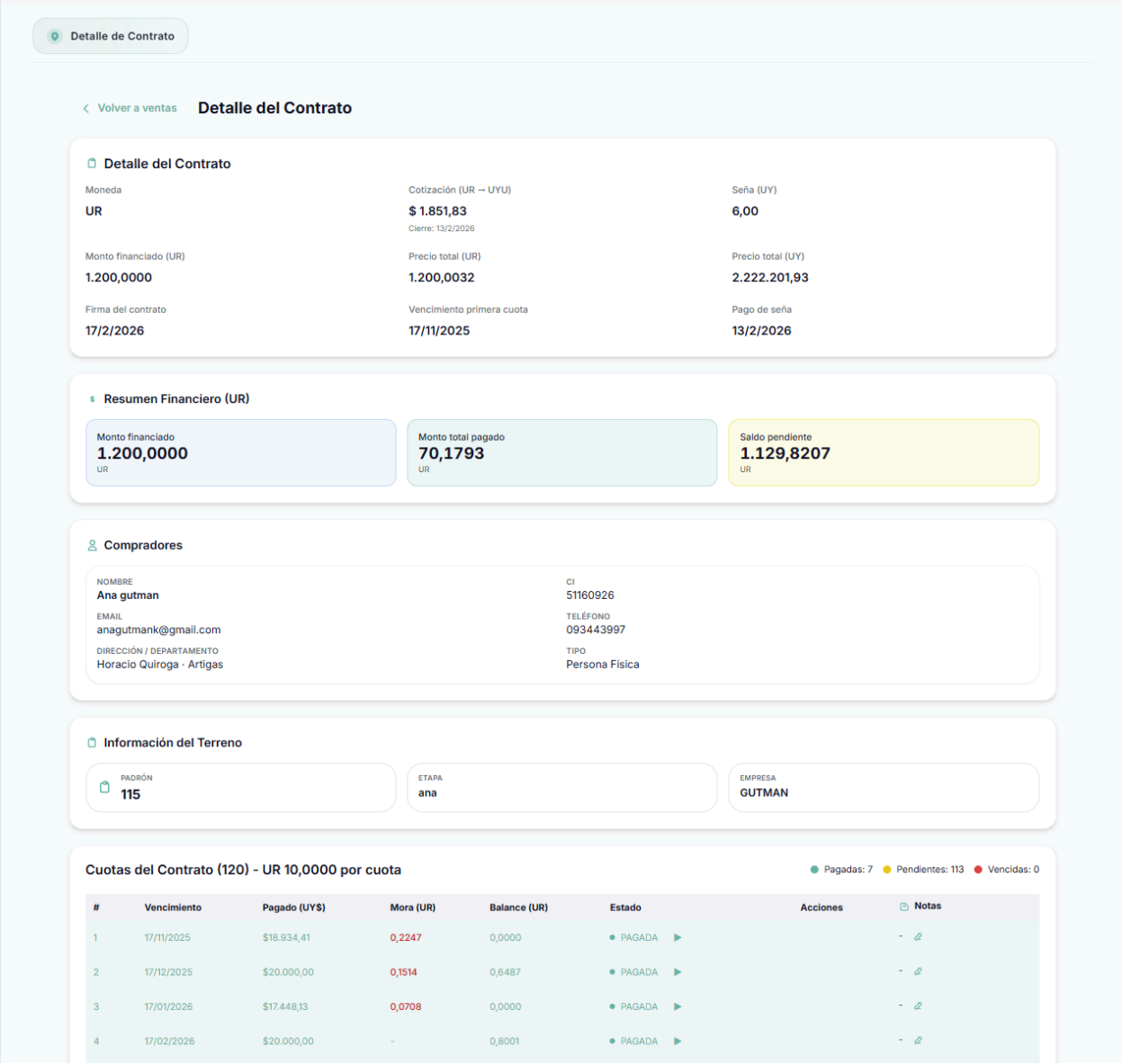
Aplicación de Heurística de Nielsen #5 (Prevención de errores): El sistema deshabilita la confirmación hasta completar los campos obligatorios y valida los datos ingresados antes de ejecutar la acción..



Aplicación de Heurística de Nielsen #7 (Flexibilidad y eficiencia de uso): la pantalla de ventas incorpora la sidebar de filtros avanzados para adaptar rápidamente la información a la tarea en curso.



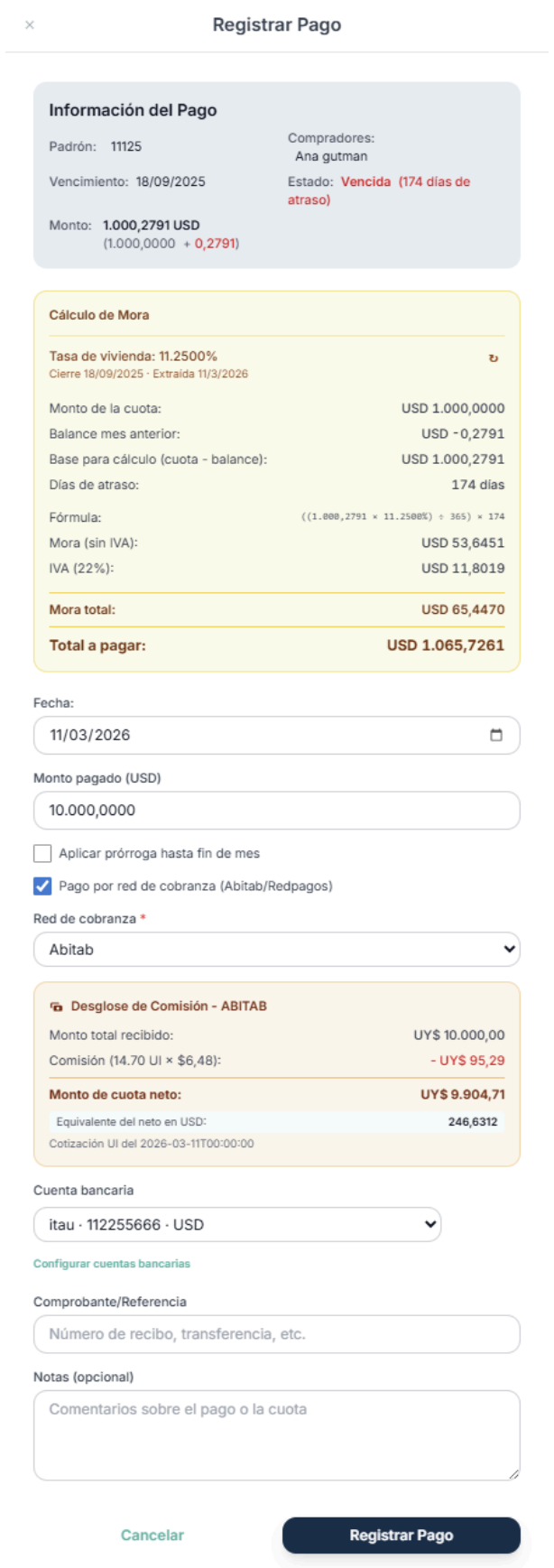
Aplicación de Heurística de Nielsen #6 (Reconocimiento antes que recuerdo): La información relevante (detalle del contrato, cuotas pendientes, historial de pagos) se consolida en la misma vista, evitando que el usuario deba recordar datos de pantallas anteriores.



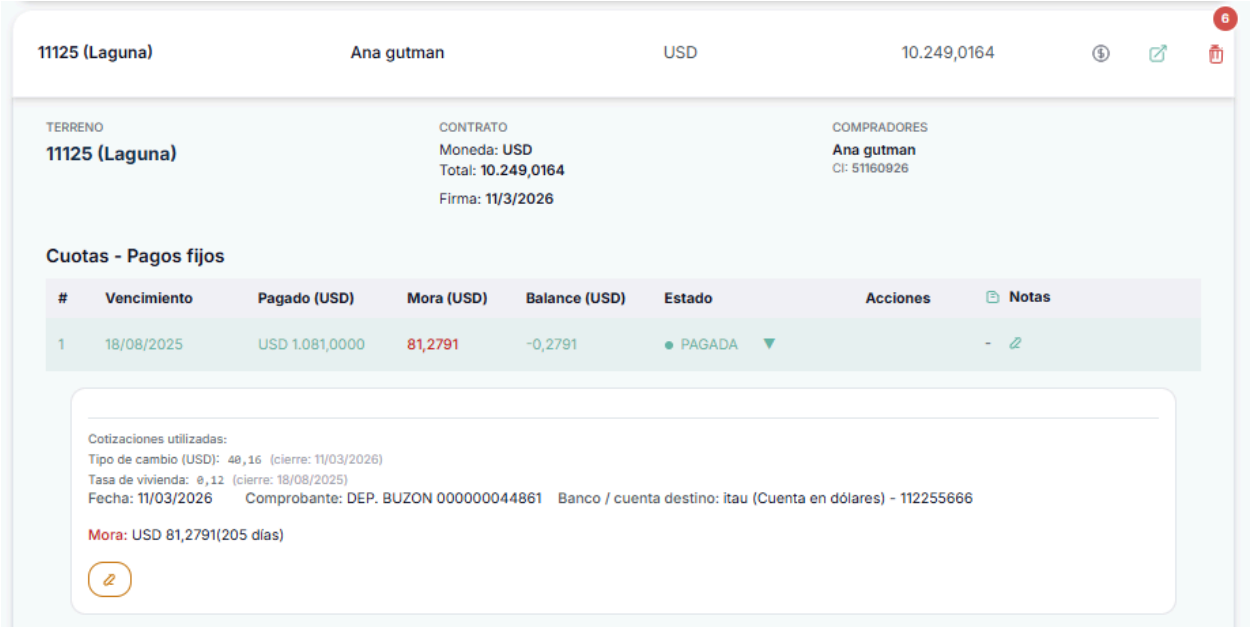
Aplicación de Heurística de Nielsen #6 (Reconocimiento antes que recuerdo): La información relevante (detalle del contrato, cuotas pendientes, historial de pagos) se consolida en la misma vista, evitando que el usuario deba recordar datos de pantallas anteriores.

13.7.3. Anexo 7.3 – Cobranza y validaciones (RF-06, RF-07):

Capturas del modal de registro de pagos, evidenciando las alertas de negocio, el cálculo automático de mora en pantalla y la aplicación de comisiones de redes de cobranza


Registrar Pago
Información del Pago
Padrón: 11125
Compradores:
Ana gutman
Vencimiento: 18/09/2025
Estado: Vencida (174 días de atraso)
Monto: 1.000,2791 USD
(1.000,0000 + 0,2791)
Cálculo de Mora
Tasa de vivienda: 11.2500%
Cierre 18/09/2025 · Extraída 11/3/2026
Monto de la cuota: USD 1.000,0000
Balance mes anterior: USD -0,2791
Base para cálculo (cuota - balance): USD 1.000,2791
Días de atraso: 174 días
Fórmula: ((1.000,2791 × 11.2500%) ÷ 365) × 174
Mora (sin IVA): USD 53,6451
IVA (22%): USD 11,8019
Mora total: USD 65,4470
Total a pagar: USD 1.065,7261
Fecha:
11/03/2026
Monto pagado (USD)
10.000,0000
Aplicar prórroga hasta fin de mes
Pago por red de cobranza (Abitab/Redpagos)
Red de cobranza *
Abitab
Desglose de Comisión - ABITAB
Monto total recibido: UY$ 10.000,00
Comisión (14.70 UI × $6,48): - UY$ 95,29
Monto de cuota neto: UY$ 9.904,71
Equivalente del neto en USD: 246,6312
Cotización UI del 2026-03-11T00:00:00
Cuenta bancaria
itau · 112255666 · USD
Configurar cuentas bancarias
Comprobante/Referencia
Número de recibo, transferencia, etc.
Notas (opcional)
Comentarios sobre el pago o la cuota
Cancelar
Registrar Pago

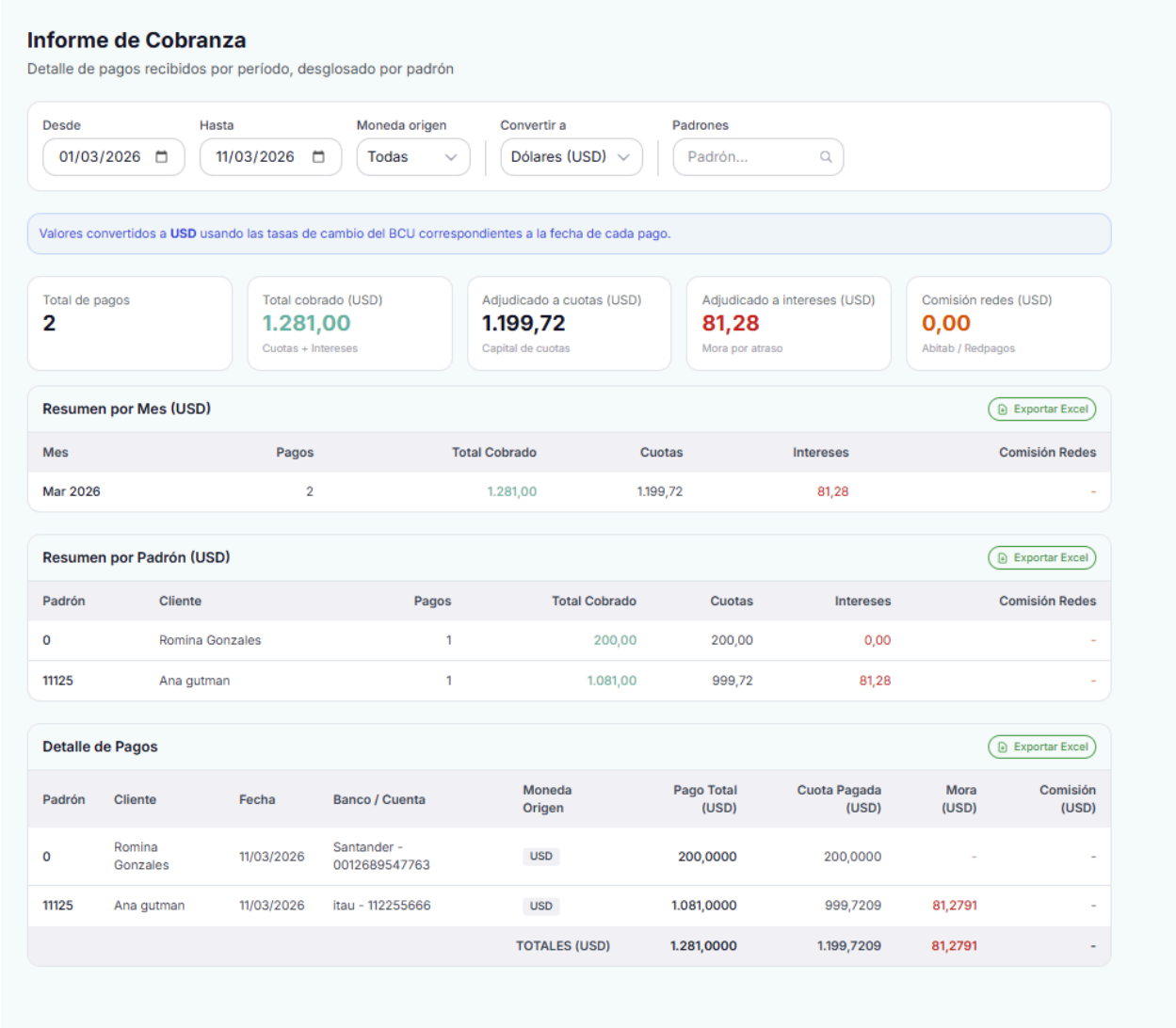
Aplicación de Heurística de Nielsen #5 (Prevención de errores): El sistema deshabilita la confirmación hasta completar los campos obligatorios y valida los datos ingresados antes de ejecutar la acción.



Aplicación de Heurística de Nielsen #6 (Reconocimiento antes que recuerdo): La información relevante (detalle del pago) es desplegable desde el mismo.

13.7.4. Anexo 7.4 – Reportes operativos y exportación (RF-08, RF-09):

Capturas de las vistas interactivas de reportes (cobranza, ventas, ingreso, cuotas vencidas y deudores) demostrando la aplicación de filtros paramétricos y la visualización de datos estructurados previa a la exportación.



Informe de Cobranza

Detalle de pagos recibidos por período, desglosado por padrón

Desde: 01/03/2026 | Hasta: 11/03/2026 | Moneda origen: Todas | Convertir a: Dólares (USD) | Padrones: Padrón...

Valores convertidos a **USD** usando las tasas de cambio del BCU correspondientes a la fecha de cada pago.

Total de pagos	Total cobrado (USD)	Adjudicado a cuotas (USD)	Adjudicado a intereses (USD)	Comisión redes (USD)
2	1.281,00	1.199,72	81,28	0,00
	Cuotas + Intereses	Capital de cuotas	Mora por atraso	Abitab / Redpagos

Resumen por Mes (USD) — Exportar Excel

Mes	Pagos	Total Cobrado	Cuotas	Intereses	Comisión Redes
Mar 2026	2	1.281,00	1.199,72	81,28	-

Resumen por Padrón (USD) — Exportar Excel

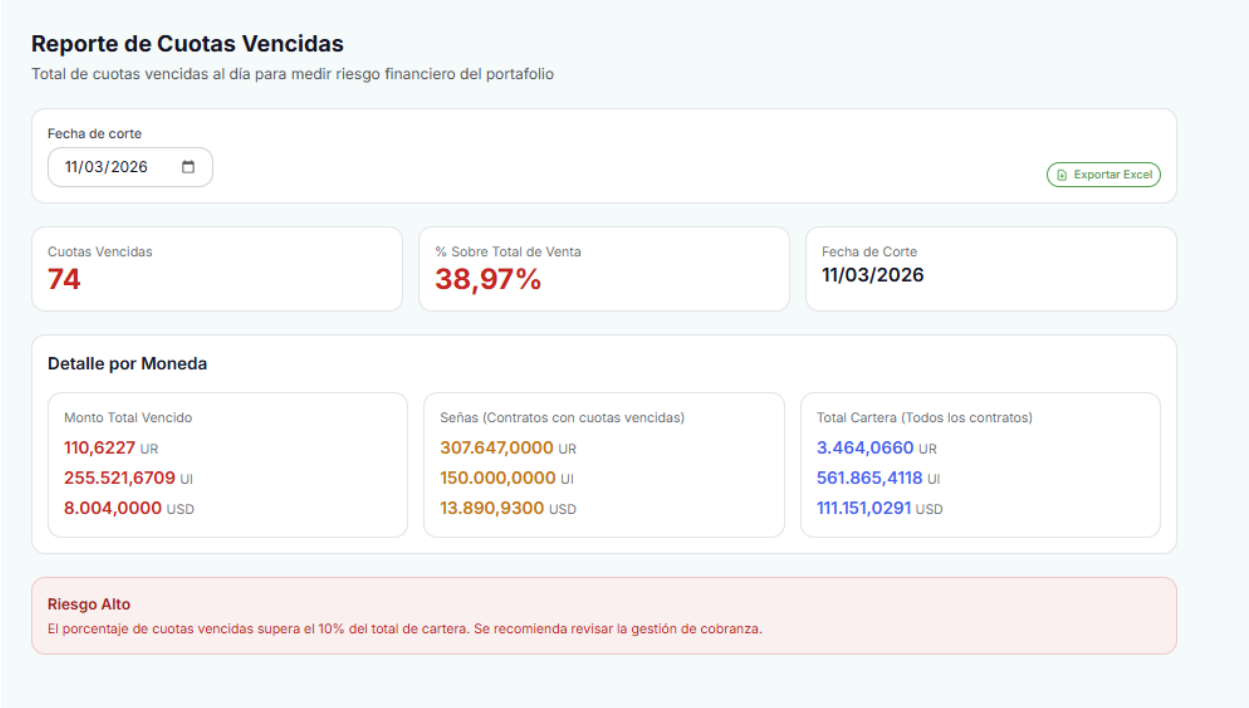
Padrón	Cliente	Pagos	Total Cobrado	Cuotas	Intereses	Comisión Redes
0	Romina Gonzales	1	200,00	200,00	0,00	-
11125	Ana gutman	1	1.081,00	999,72	81,28	-

Detalle de Pagos — Exportar Excel

Padrón	Cliente	Fecha	Banco / Cuenta	Moneda Origen	Pago Total (USD)	Cuota Pagada (USD)	Mora (USD)	Comisión (USD)
0	Romina Gonzales	11/03/2026	Santander - 0012689547763	USD	200,0000	200,0000	-	-
11125	Ana gutman	11/03/2026	itau - 112255666	USD	1.081,0000	999,7209	81,2791	-
				TOTALES (USD)	1.281,0000	1.199,7209	81,2791	-

Aplicación de Heurística de Nielsen #7 (Flexibilidad y eficiencia de uso): La vista incorpora filtros combinables (por fecha, padrón, moneda, etapa, etc.) para adaptar rápidamente la información a la tarea en curso.

Aplicación de Heurística de Nielsen #6 (Reconocimiento antes que recuerdo): La información relevante se consolida en el reporte.



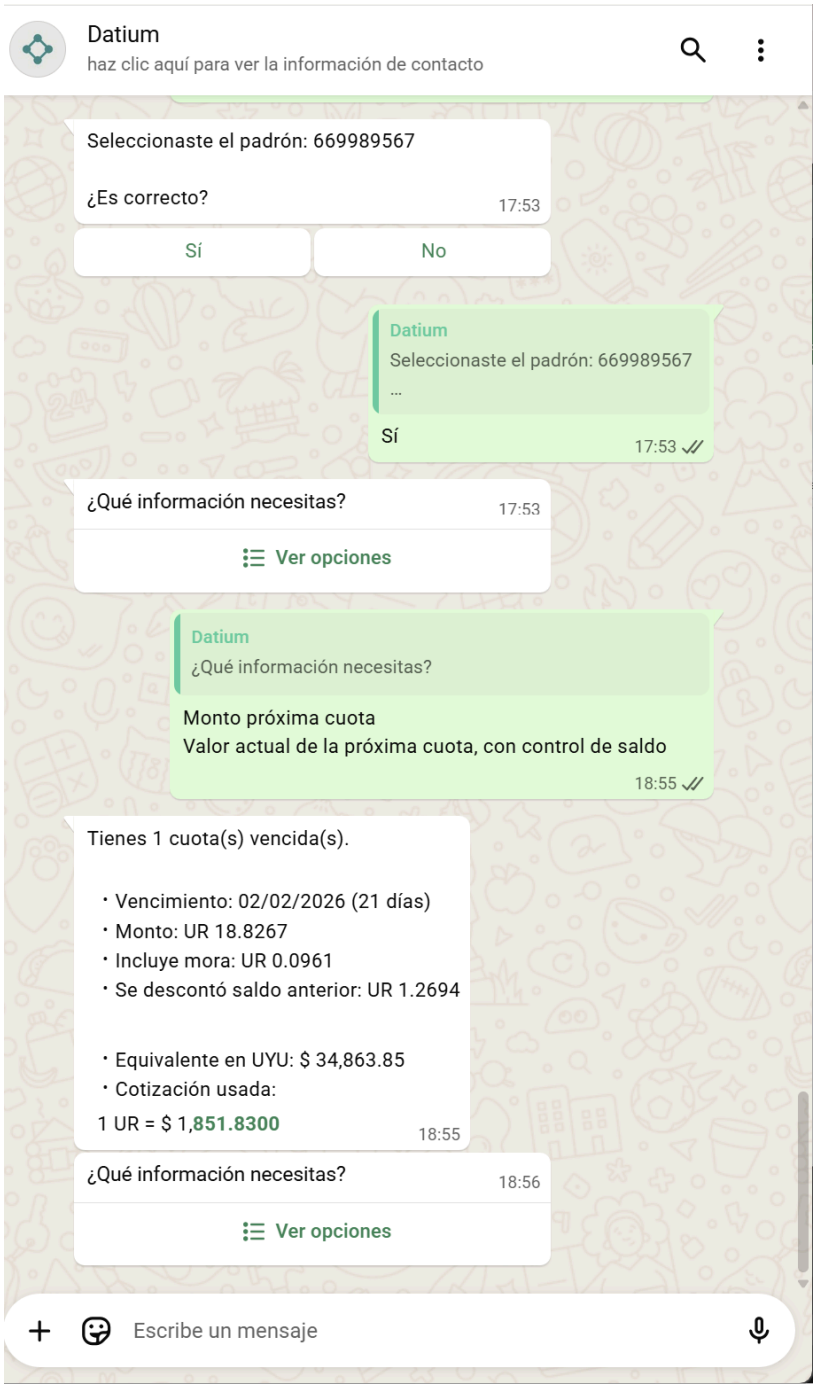
Aplicación de Heurística de Nielsen #7 (Flexibilidad y eficiencia de uso): la vista incorpora un filtro por fecha para adaptar rápidamente la información a la tarea en curso.

	A	B	C	D	E
	Concepto	**Valor**	**UR**	**UI**	**USD**
	Fecha de Corte	11/03/2026			
	Cantidad de Cuotas Vencidas	74			
	% Sobre Total de Venta	38,97%			
	Monto Total Vencido		110,6227	255.521,6709	8.004,0000
	Señas (Contratos con cuotas vencidas)		307.647,0000	150.000,0000	13.890,9300
	Total Cartera (Todos los contratos)		3.464,0660	561.865,4118	111.151,0291
	Indicador de Riesgo	Riesgo Alto - El porcentaje de cuotas vencidas supera el 10%			

.

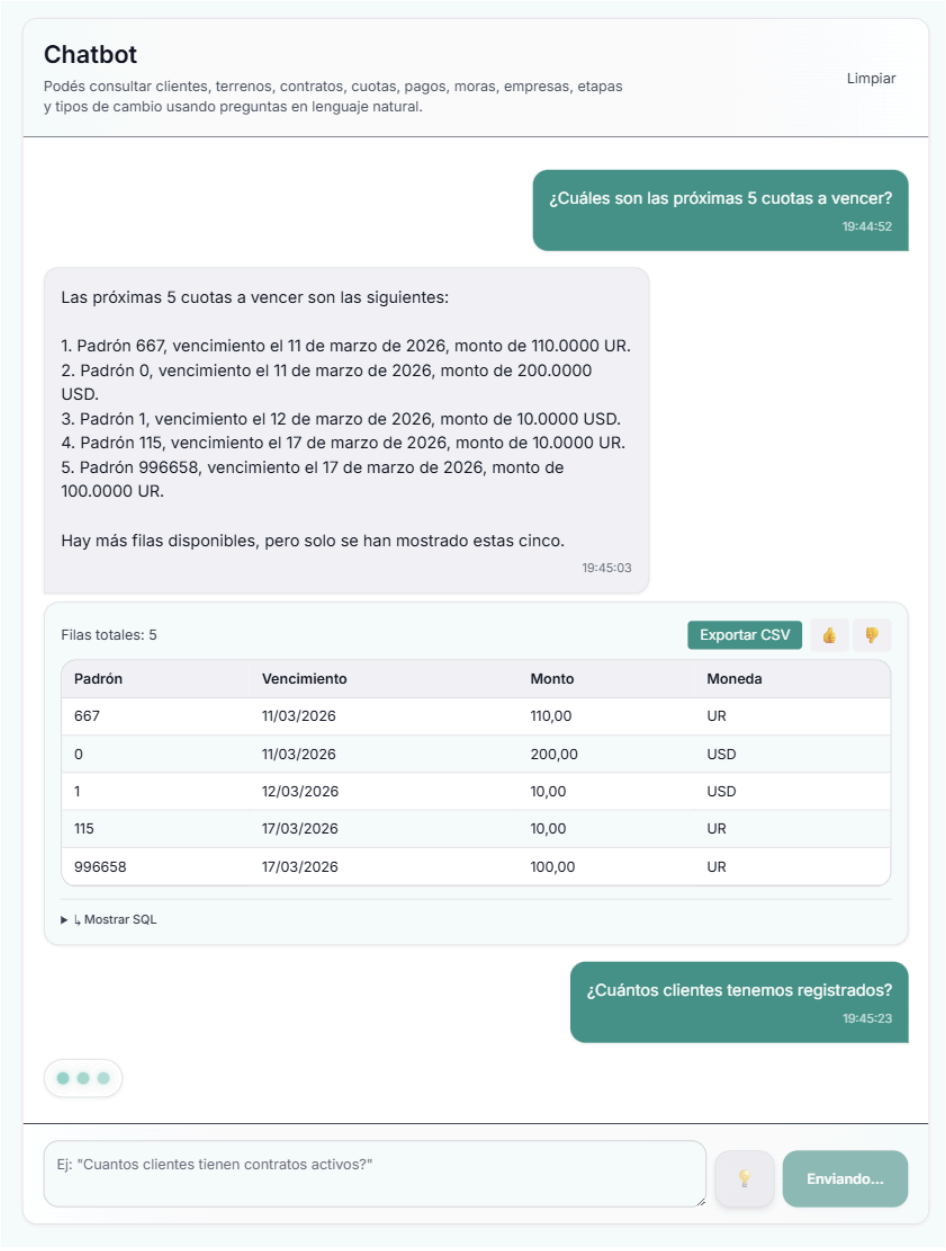
13.7.5. Anexo 7.5 – Canal de autoservicio vía WhatsApp (RF-10):

Capturas de pantalla desde la interfaz de un dispositivo móvil mostrando la máquina de estados del *bot*, la verificación de identidad mediante *product_census* y la respuesta estructurada de próximos vencimientos

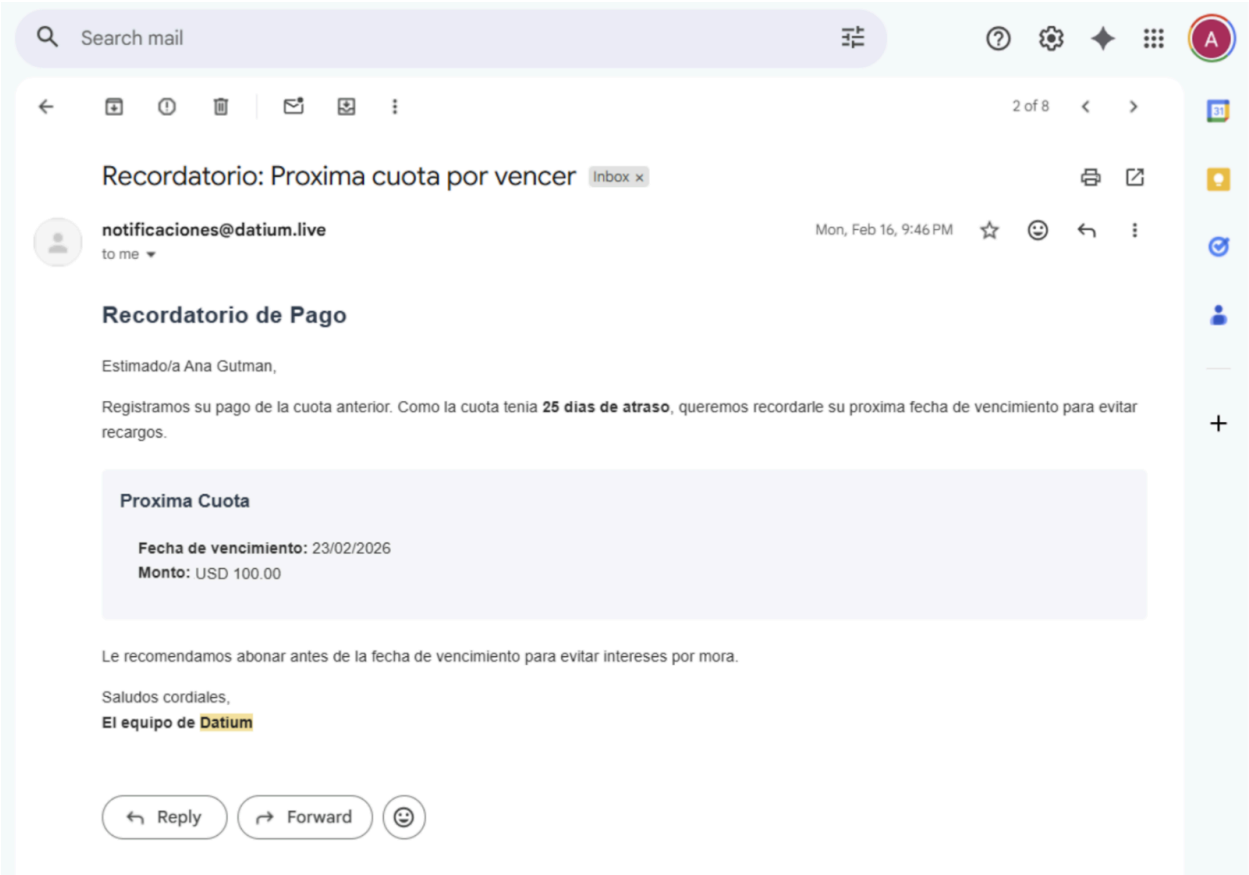


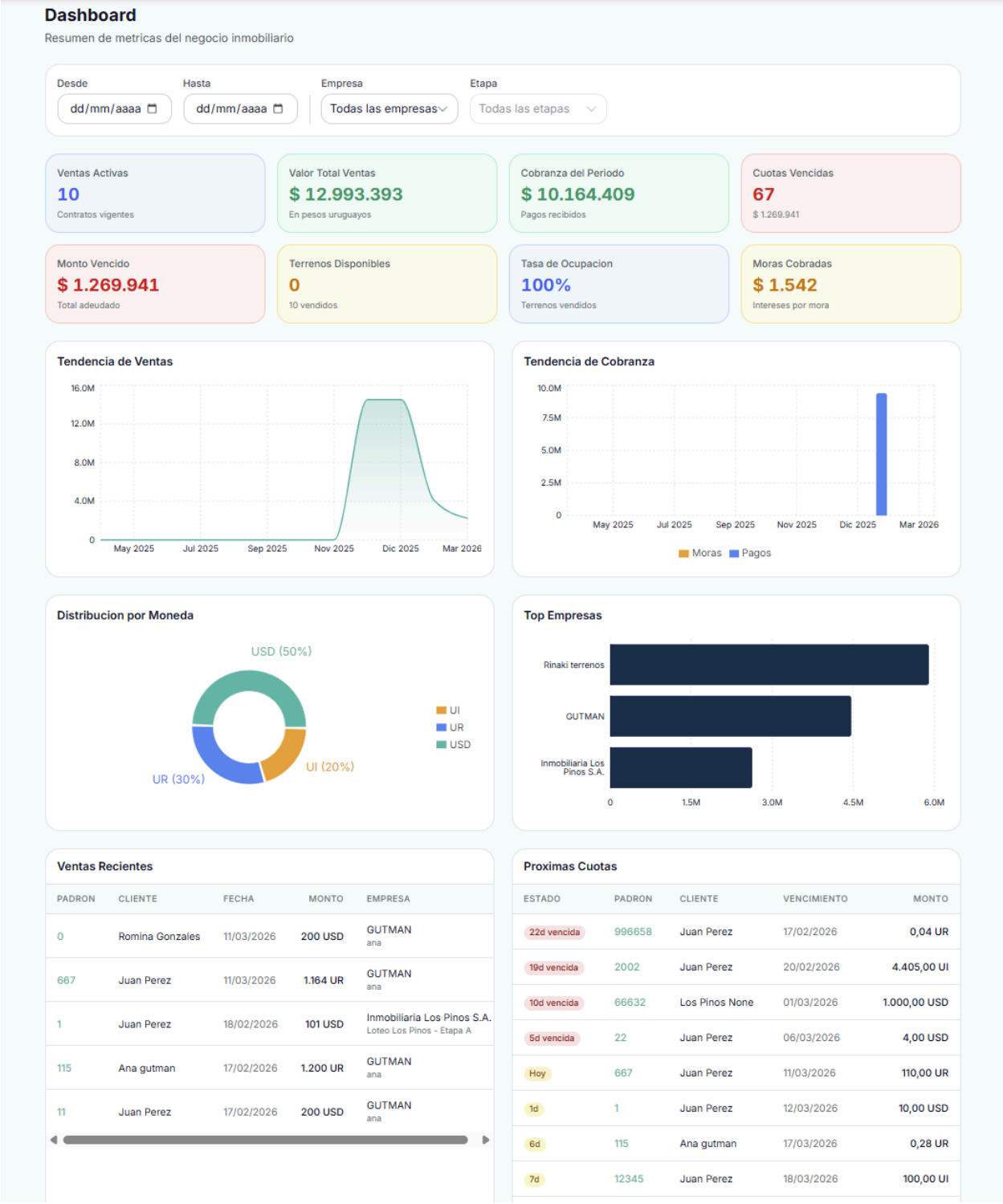
13.7.6. Anexo 7.6 – Funcionalidades extendidas (RF-11, RF-12, RF-13):

Capturas del widget del *chatbot* interno procesando una consulta en lenguaje natural y devolviendo la tabla SQL, una captura de la bandeja de entrada mostrando el formato del correo electrónico enviado por Resend para notificar la mora, y la vista general del *dashboard* interactivo con sus tarjetas de indicadores de negocio y gráficos



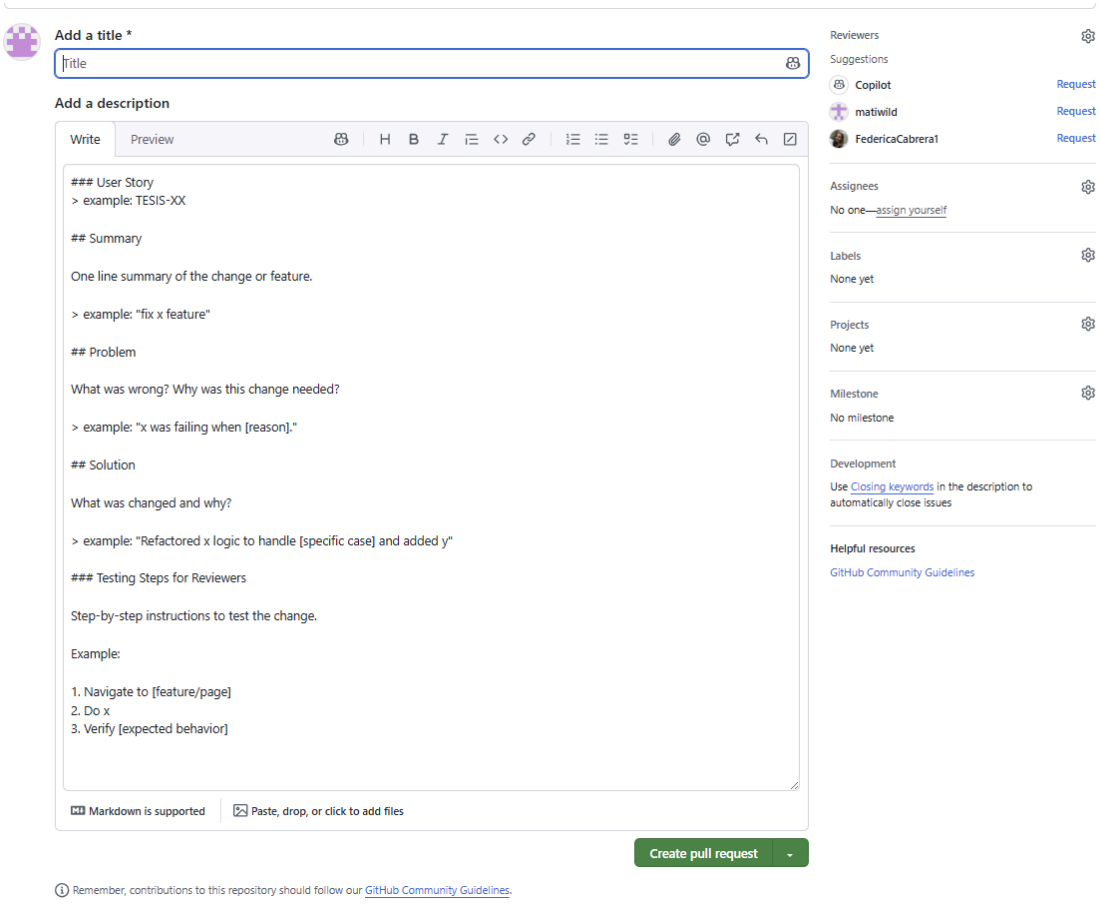
Aplicación de Heurística de Nielsen #1 (Visibilidad del estado del sistema): La interfaz informa de forma inmediata el estado de la operación mediante indicadores visuales claros, como la animación de "..." mientras el modelo procesa la consulta.





Aplicación de Heurística de Nielsen #6 (Reconocimiento antes que recuerdo): La información relevante se consolida en el dashboard.

13.8. Anexo 8 - Template de *Pull Request*



13.9. Anexo 9 - Proceso de habilitación del *chatbot* de WhatsApp mediante Meta

Este anexo describe el proceso técnico y administrativo necesario para habilitar la integración del sistema con WhatsApp Cloud API (Meta), utilizada para implementar el *chatbot* de consultas para compradores. La puesta en marcha del canal requirió configurar activos en el ecosistema de Meta, registrar y activar un número de WhatsApp, y exponer un *webhook* público para la recepción de eventos de mensajería.

A diferencia de otras funcionalidades del proyecto, esta integración implicó dependencias externas y validaciones fuera del control del equipo (por ejemplo, requisitos de configuración del negocio y activación del número). Como consecuencia, el trabajo incluyó múltiples iteraciones de configuración y resolución de incidencias hasta lograr una comunicación end-to-end estable entre WhatsApp, la plataforma de Meta y el backend.

La configuración y las validaciones del flujo completo de mensajería se realizaron principalmente en el entorno de desarrollo. Esto responde a la decisión comercial del cliente de no habilitar el *bot* en producción hasta haber completado la migración de sus datos históricos, asegurando que la herramienta esté disponible para toda su cartera de clientes de forma simultánea y no únicamente para los contratos nuevos

1. Configuración inicial en Meta (Developer + WhatsApp Cloud API)

Se inició la integración creando y configurando los activos necesarios dentro del ecosistema de Meta:

- Creación y configuración de una aplicación en Meta for Developers.
- Habilitación del producto WhatsApp Cloud API para esa aplicación.
- Obtención de credenciales e identificadores necesarios para operar (por ejemplo, token de acceso y el identificador del número/phoneNumberId).

Objetivo de esta etapa: disponer de un entorno habilitado en Meta para poder registrar un número de WhatsApp y comenzar la integración técnica con el backend.

2. Verificación del negocio y requisitos de habilitación (Meta Business)

Para poder operar el canal de WhatsApp de forma completa, Meta exige cumplir requisitos vinculados al negocio asociado a la app. Durante el proceso se observaron restricciones que bloquearon el avance y requirieron iteraciones de configuración.

En particular, se detectó que:

- La habilitación del número podía quedar en estado pendiente durante días.
- Fue necesario completar configuraciones del negocio para destrabar el proceso, incluyendo:
    - agregar un método de pago, y
    - completar/actualizar información vinculada al negocio (por ejemplo, presencia web y activos asociados).

El impacto operativo fue que la activación del número y la habilitación completa de la cuenta dependieron de completar esta verificación, lo cual introdujo demoras no controlables por el equipo.

3. Registro y activación del número en WhatsApp Cloud API

Luego de obtener los identificadores necesarios, se realizó el registro y activación del número telefónico dentro de WhatsApp Cloud API.

En los primeros intentos, el número se vinculó bajo la modalidad WhatsApp Business (aplicación), lo cual no resultó compatible con el enfoque definido para la integración mediante WhatsApp Cloud API. Como consecuencia, fue necesario adquirir y registrar nuevos números hasta contar con uno correctamente habilitado y asociado a Cloud API.

La activación efectiva del número se completó aplicando un procedimiento de tres pasos ejecutado desde el Graph API Explorer (o mediante llamadas equivalentes a la Graph API), utilizando como insumos el WABA ID (WhatsApp Business Account ID) y el Phone Number ID:

Paso 1 – Obtener el Phone Number ID a partir del WABA ID

Se consultó la lista de números asociados a la cuenta de WhatsApp Business para identificar el Phone Number ID correspondiente.

Operación (genérica): GET /{WABA_ID}/phone_numbers

Paso 2 – Registrar el número con un PIN de verificación

Con el Phone Number ID obtenido en el paso anterior, se ejecutó el registro del número utilizando un PIN. El PIN puede ser arbitrario, pero debe conservarse para reintentos o verificaciones posteriores.

Operación (genérica): POST /{PHONE_NUMBER_ID}/register

Parámetros: messaging_product=whatsapp, pin=<PIN>

Paso 3 – Suscribir la aplicación a los eventos del WABA

Finalmente, se suscribió la aplicación a los eventos de la cuenta (necesario para la recepción de callbacks).

Operación (genérica): POST /{WABA_ID}/subscribed_apps

Tras completar estos tres pasos, el número quedó reflejado como “Aprobado” en el panel de Business Manager, habilitando su uso dentro de WhatsApp Cloud API.

Como lección clave, el registro/activación del número es un paso obligatorio previo a la recepción de mensajes; sin completar este procedimiento (y sin utilizar la modalidad correcta desde el inicio), el *bot* no puede operar ni recibir eventos entrantes.

4. Configuración del *webhook* del *backend*

Para que el sistema pudiera recibir mensajes entrantes desde WhatsApp, fue necesario configurar un *webhook* en la aplicación de Meta. El *webhook* consiste en un endpoint

HTTP público expuesto por el *backend*, al cual Meta envía eventos cada vez que ocurre una interacción relevante (por ejemplo, un mensaje entrante).

La configuración incluyó:

- Implementación del endpoint del *webhook* en el *backend*, con soporte para:
    - Handshake de verificación (mecanismo inicial mediante el cual Meta valida que el endpoint pertenece al desarrollador).
    - Recepción de eventos (mensajes entrantes y cambios de estado).
- Publicación del endpoint en una URL accesible públicamente (HTTPS).
- Registro de la URL del *webhook* en el panel de Meta, junto con el verify token requerido para completar el handshake.
- Suscripción a los eventos de mensajería necesarios para que Meta enviara callbacks al *backend*.

Como resultado, quedó operativo un endpoint público del *backend* para la recepción de eventos.

5. Pruebas end-to-end (E2E) del flujo de mensajería

Una vez completada la activación del número y la configuración del *webhook*, se realizaron pruebas end-to-end para validar la integración completa entre WhatsApp, Meta y el *backend* del sistema.

El flujo validado fue el siguiente:

1. El comprador envía un mensaje al número de WhatsApp asociado al *bot*.
2. La plataforma de WhatsApp (Meta) genera un evento y lo envía al *webhook* configurado.
3. El *backend* recibe el evento, valida su estructura y determina el estado conversacional correspondiente.

4. El *backend* consulta la base de datos del sistema cuando corresponde (por ejemplo, para obtener estado de cuenta o próximos vencimientos).
5. El *backend* envía la respuesta al usuario mediante la API oficial de WhatsApp.

Estas pruebas confirmaron que el sistema era capaz de recibir eventos de mensajería y responder correctamente a través de la API, habilitando la implementación progresiva de la lógica del *chatbot*.

6. Incidencias principales y resolución

A continuación se resumen las incidencias más relevantes encontradas durante el proceso de habilitación, junto con su resolución.

6.1. Fallo en la verificación del *webhook* (“Invalid verify token”)

- Síntoma / error: Invalid verify token.
- Causa: el verify token configurado en Meta no coincidía con el token esperado por el *backend*.
- Acciones realizadas: revisión de la configuración del *webhook*, reinicios/*deploy* del *backend* y verificación del endpoint expuesto.
- Solución: alinear el mismo verify token en Meta y en la variable de entorno del *backend*.
- Lección aprendida: el verify token debe coincidir exactamente entre Meta y el *backend* para completar la verificación.

6.2. Mensajes no llegaban al *backend* (eventos no recibidos)

- Síntoma: al enviar mensajes desde WhatsApp, no se recibían eventos en el *backend*.
- Causa: URL de *webhook* incorrecta, incompleta o apuntando a un path distinto al endpoint real.

- Acciones realizadas: revisión de la URL configurada, validación de accesibilidad pública y verificación del despliegue.
- Solución: configurar en Meta la URL completa del API Gateway incluyendo el path exacto del endpoint.
- Lección aprendida: el *webhook* debe ser accesible públicamente y apuntar exactamente al endpoint esperado.

6.3. Activación del número bloqueada (pendiente durante varios días)

- Síntoma: el número permanecía en estado de activación pendiente por un período prolongado
- Causa: configuración incompleta del negocio asociado a la app en Meta.
- Acciones realizadas: reintentos de configuración, eliminación y re-alta del número, incorporación de método de pago y actualización de información de negocio requerida por Meta.
- Solución: completar los requisitos del negocio (incluyendo método de pago y datos solicitados por Meta) hasta destrabar la habilitación.
- Lección aprendida: la activación de WhatsApp Cloud API depende de verificaciones y requisitos externos que exceden lo estrictamente técnico.

6.4. Dependencias de infraestructura en pruebas E2E

- Síntoma: se evaluó si la falta de respuesta del *bot* podía deberse a componentes no disponibles (*backend*/base de datos).
- Causa: el flujo end-to-end depende de que el *backend* esté operativo y accesible públicamente; fallas de infraestructura pueden confundirse con errores de configuración en Meta.
- Solución: pruebas incrementales por capas (handshake del *webhook* → recepción de evento → respuesta del *bot*).
- Lección aprendida: en integraciones externas conviene aislar pasos para reducir tiempo de diagnóstico.

6.5. Consideraciones de costos y uso previsto sin costos adicionales

Se realizó un análisis del modelo de conversaciones de WhatsApp Cloud API y se diseñó el flujo para que el canal funcione principalmente bajo un patrón buyer-initiated (la conversación es iniciada por el comprador). En el uso previsto, las consultas se resuelven dentro de la ventana de conversación, evitando mensajes salientes innecesarios y el uso de plantillas cuando no fueran requeridas, reduciendo el riesgo de costos operativos adicionales.

7. Configuración final lograda

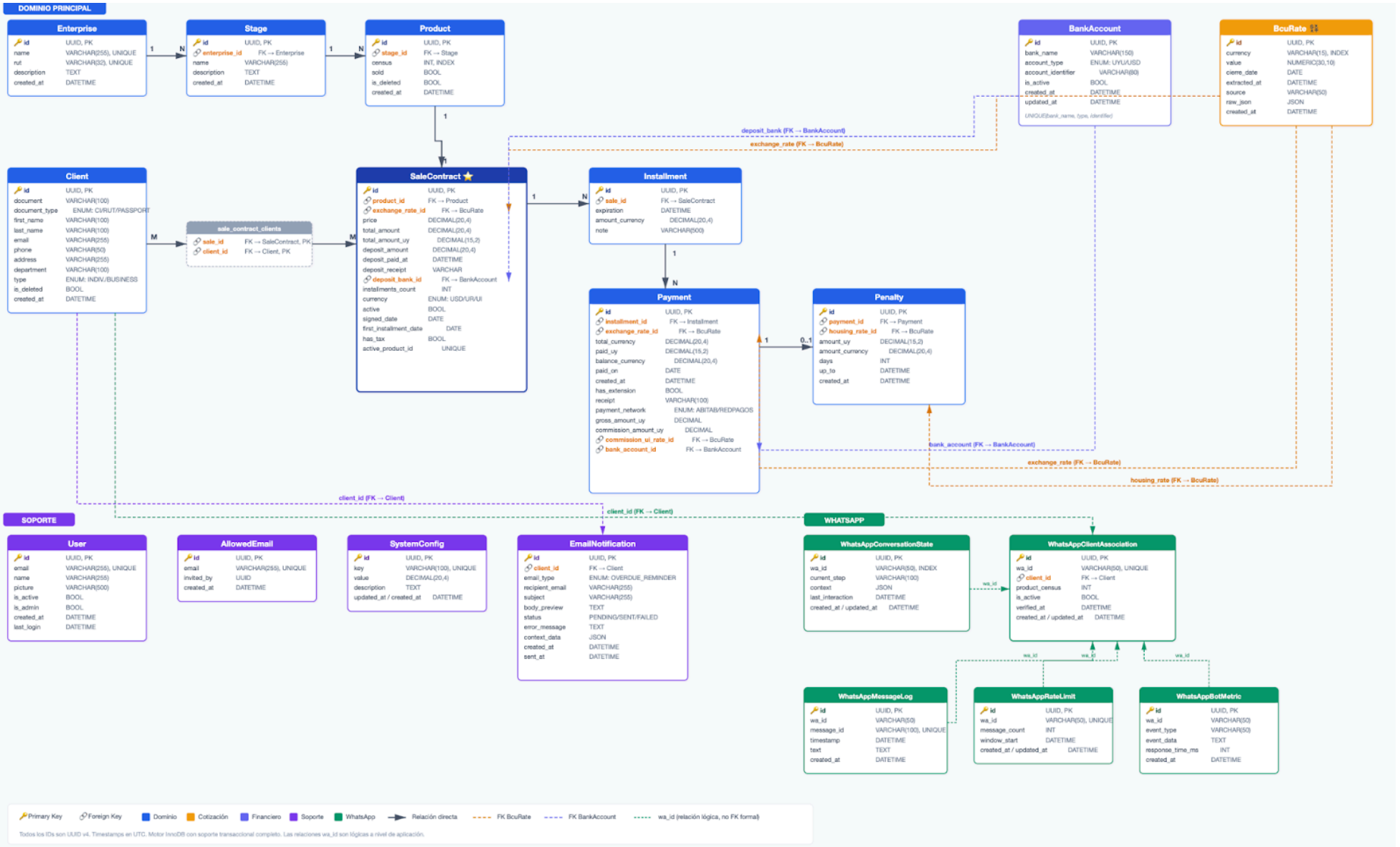
- Verify token: configurado en Meta y alineado con la configuración del *backend* (sin exponer el valor).
- Tokens / permisos: se utilizó el token de acceso generado en Meta Developer junto con los permisos necesarios para recepción de eventos y envío de respuestas.
- Entornos: la integración se configuró y validó inicialmente en desarrollo, dado que el cliente aún no utilizaba WhatsApp en producción
- Estado final: el *backend* recibe correctamente eventos entrantes desde WhatsApp y procesa los mensajes para responder mediante el *chatbot*.

8. Decisiones y *trade-offs*

- Configurar primero en dev: se decidió validar toda la integración en desarrollo antes de moverla a producción, minimizando riesgo operativo.
- API oficial (sin intermediarios): se eligió WhatsApp Cloud API oficial para mantener control directo de la integración y evitar dependencias adicionales.
- Priorizar el flujo mínimo operativo: se prioriza habilitar primero el circuito base (número habilitado + *webhook* verificado + recepción de eventos) antes de agregar lógica avanzada del *bot*.

- *Webhook* único por aplicación: Meta opera con un *webhook* activo por app; por lo tanto, la transición dev→prod se realiza modificando la URL configurada en Meta.
- Diseño orientado a minimizar costos: se diseñó el *bot* para operar principalmente con conversaciones iniciadas por el comprador (buyer-initiated), evitando notificaciones salientes y plantillas cuando no fueran necesarias.

13.10. Anexo 10 - Modelo Entidad-Relación / MER


DOMINIO PRINCIPAL
Enterprise
id UUID, PK
name VARCHAR(255), UNIQUE
rut VARCHAR(32), UNIQUE
description TEXT
created_at DATETIME
Stage
id UUID, PK
enterprise_id FK → Enterprise
name VARCHAR(255)
description TEXT
created_at DATETIME
Product
id UUID, PK
stage_id FK → Stage
census INT, INDEX
sold BOOL
is_deleted BOOL
created_at DATETIME
BankAccount
id UUID, PK
bank_name VARCHAR(150)
account_type ENUM: UYU/USD
account_identifier VARCHAR(80)
is_active BOOL
created_at DATETIME
updated_at DATETIME
UNIQUE(bank_name, type, identifier)
BcuRate
id UUID, PK
currency VARCHAR(15), INDEX
value NUMERIC(30,10)
cierre_date DATE
extracted_at DATETIME
source VARCHAR(50)
raw_json JSON
created_at DATETIME
deposit_bank (FK → BankAccount)
exchange_rate (FK → BcuRate)
Client
id UUID, PK
document VARCHAR(100)
document_type ENUM: CI/RUT/PASSPORT
first_name VARCHAR(100)
last_name VARCHAR(100)
email VARCHAR(255)
phone VARCHAR(50)
address VARCHAR(255)
department VARCHAR(100)
type ENUM: INDIV./BUSINESS
is_deleted BOOL
created_at DATETIME
sale_contract_clients
sale_id FK → SaleContract, PK
client_id FK → Client, PK
SaleContract
id UUID, PK
product_id FK → Product
exchange_rate_id FK → BcuRate
price DECIMAL(20,4)
total_amount DECIMAL(20,4)
total_amount_uy DECIMAL(15,2)
deposit_amount DECIMAL(20,4)
deposit_paid_at DATETIME
deposit_receipt VARCHAR
deposit_bank_id FK → BankAccount
installments_count INT
currency ENUM: USD/UR/UI
active BOOL
signed_date DATE
first_installment_date DATE
has_tax BOOL
active_product_id UNIQUE
Installment
id UUID, PK
sale_id FK → SaleContract
expiration DATETIME
amount_currency DECIMAL(20,4)
note VARCHAR(500)
Payment
id UUID, PK
installment_id FK → Installment
exchange_rate_id FK → BcuRate
total_currency DECIMAL(20,4)
paid_uy DECIMAL(15,2)
balance_currency DECIMAL(20,4)
paid_on DATE
created_at DATETIME
has_extension BOOL
receipt VARCHAR(100)
payment_network ENUM: ABITAB/REDPAGOS
gross_amount_uy DECIMAL
commission_amount_uy DECIMAL
commission_ui_rate_id FK → BcuRate
bank_account_id FK → BankAccount
Penalty
id UUID, PK
payment_id FK → Payment
housing_rate_id FK → BcuRate
amount_uy DECIMAL(15,2)
amount_currency DECIMAL(20,4)
days INT
up_to DATETIME
created_at DATETIME
bank_account (FK → BankAccount)
exchange_rate (FK → BcuRate)
housing_rate (FK → BcuRate)
client_id (FK → Client)
client_id (FK → Client)
SOPORTE
User
id UUID, PK
email VARCHAR(255), UNIQUE
name VARCHAR(255)
picture VARCHAR(500)
is_active BOOL
is_admin BOOL
created_at DATETIME
last_login DATETIME
AllowedEmail
id UUID, PK
email VARCHAR(255), UNIQUE
invited_by UUID
created_at DATETIME
SystemConfig
id UUID, PK
key VARCHAR(100), UNIQUE
value DECIMAL(20,4)
description TEXT
updated_at / created_at DATETIME
EmailNotification
id UUID, PK
client_id FK → Client
email_type ENUM: OVERDUE_REMINDER
recipient_email VARCHAR(255)
subject VARCHAR(255)
body_preview TEXT
status PENDING/SENT/FAILED
error_message TEXT
context_data JSON
created_at DATETIME
sent_at DATETIME
WHATSAPP
WhatsAppConversationState
id UUID, PK
wa_id VARCHAR(50), INDEX
current_step VARCHAR(100)
context JSON
last_interaction DATETIME
created_at / updated_at DATETIME
wa_id
WhatsAppClientAssociation
id UUID, PK
wa_id VARCHAR(50), UNIQUE
client_id FK → Client
product_census INT
is_active BOOL
verified_at DATETIME
created_at / updated_at DATETIME
wa_id
wa_id
wa_id
WhatsAppMessageLog
id UUID, PK
wa_id VARCHAR(50)
message_id VARCHAR(100), UNIQUE
timestamp DATETIME
text TEXT
created_at DATETIME
WhatsAppRateLimit
id UUID, PK
wa_id VARCHAR(50), UNIQUE
message_count INT
window_start DATETIME
created_at / updated_at DATETIME
WhatsAppBotMetric
id UUID, PK
wa_id VARCHAR(50)
event_type VARCHAR(50)
event_data TEXT
response_time_ms INT
created_at DATETIME
Primary Key
Foreign Key
Dominio
Cotización
Financiero
Soporte
WhatsApp
Relación directa
FK BcuRate
FK BankAccount
wa_id (relación lógica, no FK formal)
Todos los IDs son UUID v4. Timestamps en UTC. Motor InnoDB con soporte transaccional completo. Las relaciones wa_id son lógicas a nivel de aplicación.

DOMINIO — ESTRUCTURA DEL PREDIO

Enterprise — enterprises

PK	id CHAR(36)	UUID
UQ	name VARCHAR(255)	NOT NULL
UQ	rut VARCHAR(32)	NOT NULL
	description TEXT	nullable
	created_at DATETIME	

Stage — stages

PK	id CHAR(36)	UUID
FK	enterprise_id CHAR(36)	→ enterprises
	name VARCHAR(255)	NOT NULL
	description TEXT	nullable
	created_at DATETIME	

Product — products

PK	id CHAR(36)	UUID
FK	stage_id CHAR(36)	→ stages
IDX	census INT UNSIGNED	padrón
	sold BOOLEAN	default false
	is_deleted BOOLEAN	soft delete
	created_at DATETIME	

DOMINIO — OPERACIÓN COMERCIAL

Client — clients

PK	id CHAR(36)	UUID
IDX	document VARCHAR(100)	
	document_type ENUM	CI|RUT|PASSPORT
IDX	first_name VARCHAR(100)	NOT NULL
	last_name VARCHAR(100)	nullable
	email VARCHAR(255)	nullable
	phone VARCHAR(20)	nullable
	address VARCHAR(255)	nullable
	department VARCHAR(50)	nullable
	type ENUM	INDIVIDUAL|BUSINESS
	is_deleted BOOLEAN	soft delete
	created_at DATETIME	

SaleContract — sales

PK	id CHAR(36)	UUID
FK	product_id CHAR(36)	→ products
FK	exchange_rate_id CHAR(36)	→ bcu_rates (firma)
FK	deposit_bank_acct_id CHAR(36)	→ bank_accounts
	price DECIMAL(20,4)	moneda contrato
	total_amount DECIMAL(20,4)	
	total_amount_uy DECIMAL(15,2)	en UYU
	deposit_amount DECIMAL(20,4)	seña
	deposit_paid_at DATETIME	nullable
	deposit_receipt VARCHAR(100)	nullable
	installments_count INTEGER	
	currency ENUM	USD|UR|UI
	active BOOLEAN	default true
	signed_date DATETIME	NOT NULL
	first_installment_date DATETIME	NOT NULL
	has_tax BOOLEAN	default false
UQ	active_product_id CHAR(36)	1 contrato/lote

sale_contract_clients — M:M

PK FK	sale_id CHAR(36)	→ sales
PK FK	client_id CHAR(…	→ clients

DOMINIO — COBRANZA

Installment

installments

PK	**id** CHAR(36)	UUID
FK	**sale_id** CHAR(36)	→ sales
	expiration DATETIME	vencimiento
	amount_currency DECIMAL(20,4)	moneda contrato
	note VARCHAR(500)	nullable

Payment

payments

PK	**id** CHAR(36)	UUID
FK	**installment_id** CHAR(36)	→ installments
FK	**exchange_rate_id** CHAR(36)	→ bcu_rates (pago)
FK	**commission_ui_rate_id** CHAR(36)	→ bcu_rates (UI)
FK	**bank_account_id** CHAR(36)	→ bank_accounts
	total_currency DECIMAL(20,4)	pagado moneda
	paid_uy DECIMAL(15,2)	pagado UYU
	balance_currency DECIMAL(20,4)	saldo cuota
	paid_on DATETIME	fecha pago
	created_at DATETIME	
	has_extension BOOLEAN	prórroga
	receipt VARCHAR(100)	nro recibo
	payment_network ENUM	ABITAB|REDPAGOS
	gross_amount_uy DECIMAL(15,2)	bruto UYU
	commission_amount_uy DECIMAL(15,2)	comisión red
	commission_ui_value DECIMAL(20,4)	comisión UI

Penalty

penalties

PK	**id** CHAR(36)	UUID
FK	**payment_id** CHAR(36)	→ payments
FK	**housing_rate_id** CHAR(36)	→ bcu_rates (viv.)
	amount_uy DECIMAL(15,2)	mora UYU
	amount_currency DECIMAL(20,4)	mora moneda
	days INTEGER	días atraso
	up_to DATETIME	calculado hasta
	created_at DATETIME	

SOPORTE — COTIZACIONES Y CUENTAS BANCARIAS

BcuRate

bcu_rates — tabla más referenciada del sistema

PK	**id** CHAR(36)	UUID
IDX	**currency** VARCHAR(15)	USD|UR|UI|*_HOUSING
	value NUMERIC(30,10)	alta precisión
	cierre_date DATE	fecha cierre BCU
	extracted_at DATETIME	cuándo se consultó
	source VARCHAR(50)	default "BCU"
	raw_json JSON	respuesta cruda
	created_at DATETIME	

BankAccount

bank_accounts

PK	**id** CHAR(36)	UUID
	bank_name VARCHAR(150)	NOT NULL
	account_type ENUM	UYU|USD
	account_identifier VARCHAR(80)	NOT NULL
	is_active BOOLEAN	default true
	created_at DATETIME	
	updated_at DATETIME	auto-update

⚡ UNIQUE(bank_name, account_type, account_identifier)

SOPORTE — AUTENTICACIÓN Y CONFIGURACIÓN

User — users

PK	id CHAR(36)	UUID
UQ	email VARCHAR(255)	NOT NULL
	name VARCHAR(255)	nullable
	picture VARCHAR(500)	URL avatar
	is_active BOOLEAN	default true
	is_admin BOOLEAN	default false
	created_at DATETIME	
	last_login DATETIME	nullable

AllowedEmail — allowed_emails

PK	id CHAR(36)	UUID
UQ	email VARCHAR(255)	NOT NULL
	invited_by CHAR(36)	sin FK
	created_at DATETIME	

SystemConfig — system_config

PK	id CHAR(36)	UUID
UQ	key VARCHAR(100)	
	value DECIMAL(20,4)	
	description VARCHAR(500)	nullable
	updated_at DATETIME	auto-update
	created_at DATETIME	

EmailNotification — email_notifications

PK	id CHAR(36)	UUID
FK	client_id CHAR(36)	→ clients
	email_type ENUM	OVERDUE_REMINDER
	recipient_email VARCHAR(255)	
	subject VARCHAR(500)	
	body_preview TEXT	nullable
	status ENUM	PENDING|SENT|FAILED
	error_message TEXT	nullable
	context_data TEXT	JSON
	created_at DATETIME	
	sent_at DATETIME	nullable

WHATSAPP BOT

WA Conversation... — whatsapp_conversation_states

PK	id VARCHAR(36)	UUID
IDX	wa_id VARCHAR(50)	nro WhatsApp
	current_step VARCHAR(100)	paso del flujo
	context JSON	datos diálogo
	last_interaction DATETIME	
	created_at DATETIME	
	updated_at DATETIME	

WA ClientAssocia... — whatsapp_client_associations

PK	id VARCHAR(36)	UUID
UQ	wa_id VARCHAR(50)	1 wa = 1 client
FK	client_id VARCHAR(36)	→ clients
	product_census VARCHAR(20)	nullable
	is_active BOOLEAN	
	verified_at DATETIME	nullable
	created_at / updated_at DATETIME	

WA MessageLog — whatsapp_message_logs

PK	id CHAR(36)	UUID
IDX	wa_id VARCHAR(50)	
UQ	message_id VARCHAR(100)	ID msg WA
	timestamp DATETIME	
	text TEXT	contenido
	created_at DATETIME	

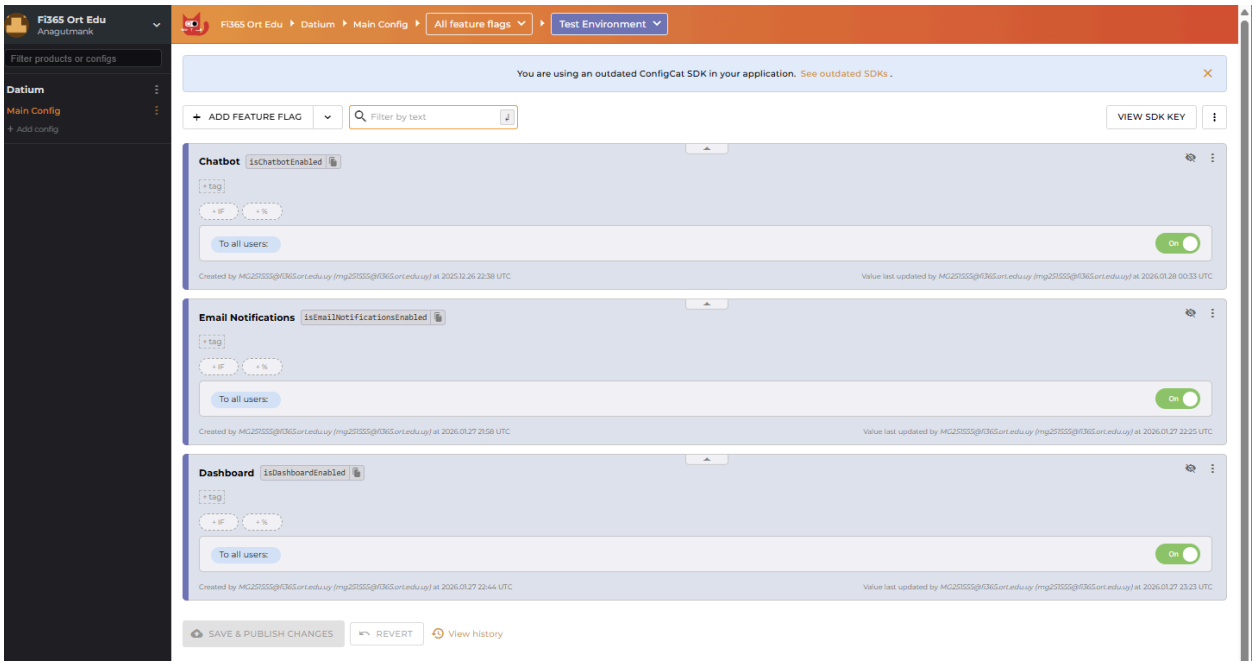
WA RateLimit — whatsapp_rate_limits

PK	id VARCHAR(36)	
UQ	wa_id VARCHAR(50)	
	message_count INTEGER	default 0
	window_start DATETIME	
	created_at / updated_at DATETIME	

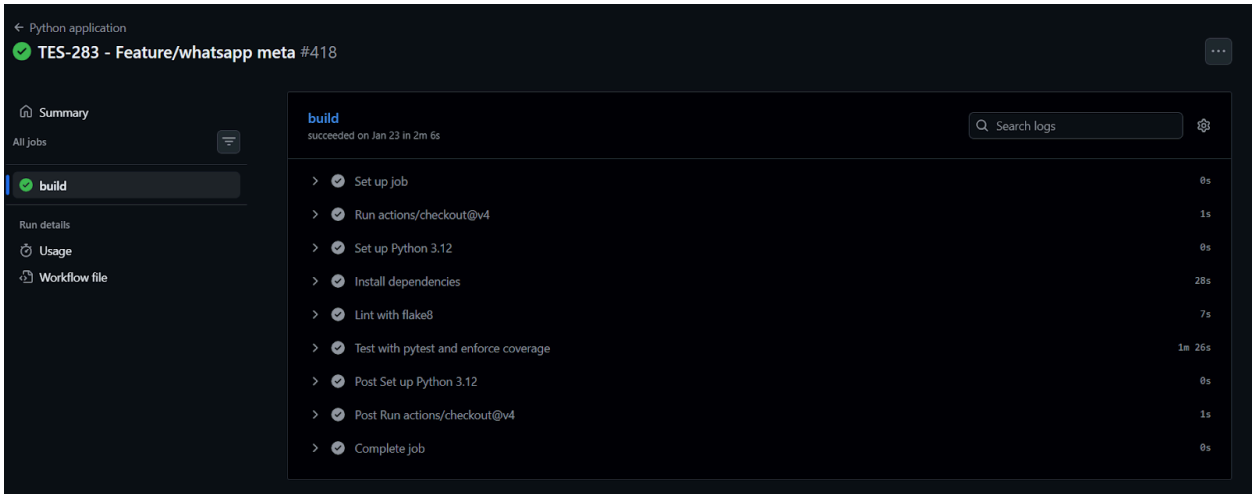
WA BotMetric — whatsapp_bot_metrics

PK	id VARCHAR(36)	
IDX	wa_id VARCHAR(50)	
IDX	event_type VARCHAR(50)	start|menu|error
	event_data TEXT	JSON string
	response_time_ms INTEGER	nullable
IDX	created_at DATETIME	

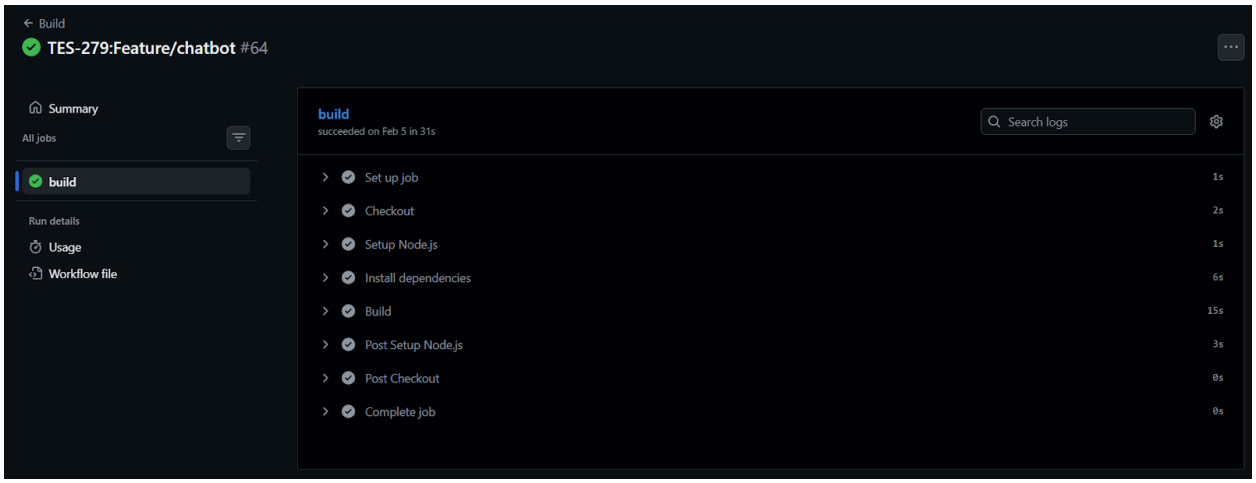
13.11. Anexo 11 - Captura de configuración de *feature flags* con ConfigCat



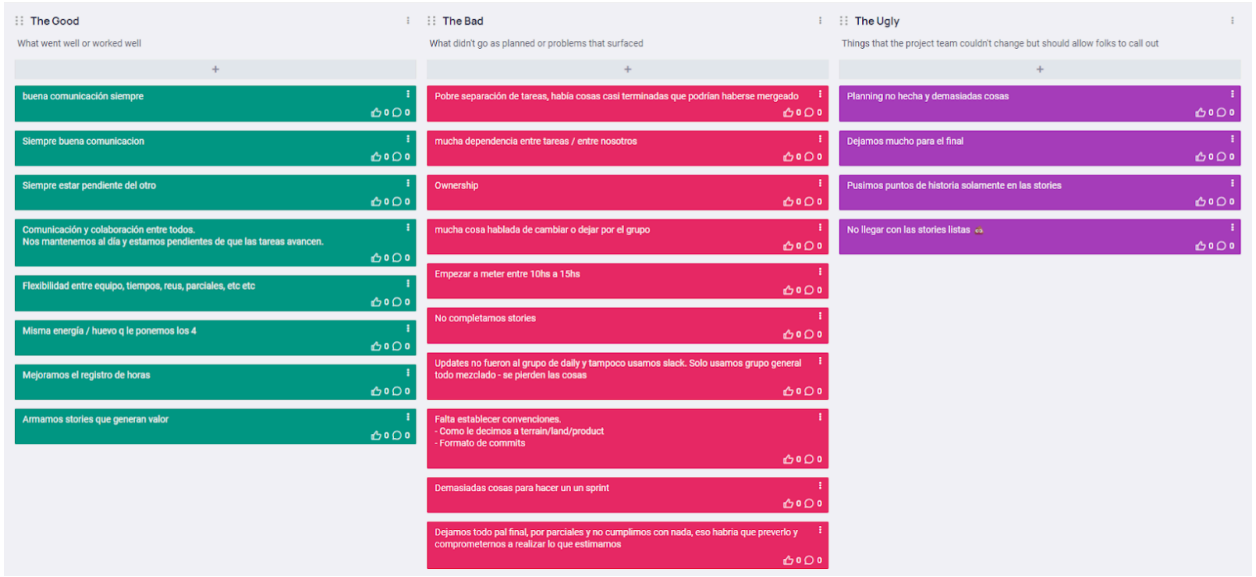
13.12. Anexo 12 - Captura de *Pipeline* de CI del *backend*



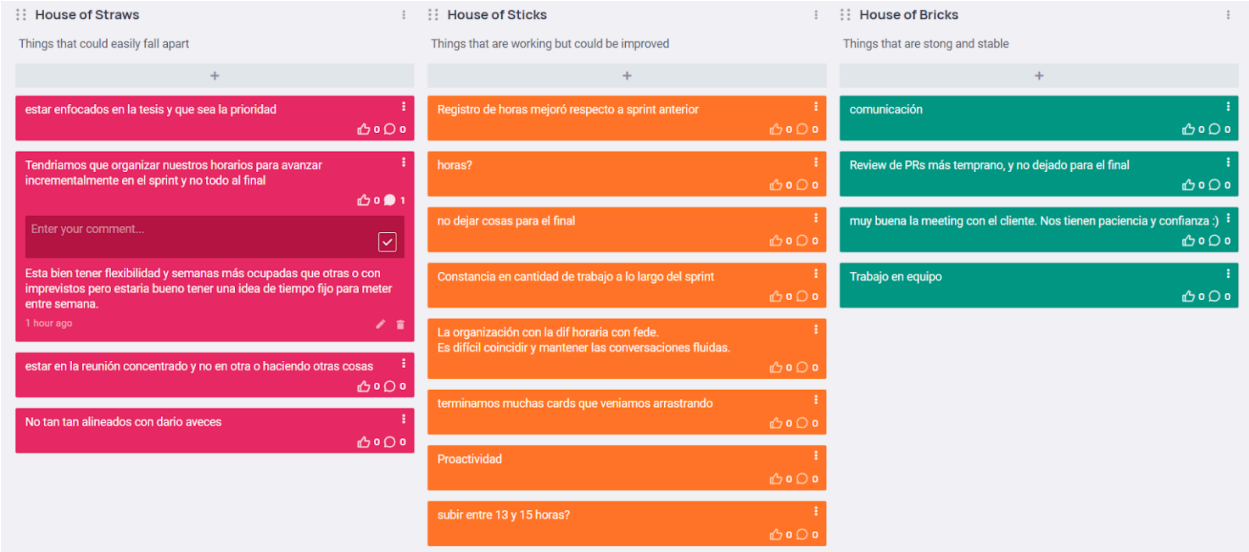
13.13. Anexo 13 - Captura de *Pipeline* de CI del *frontend*



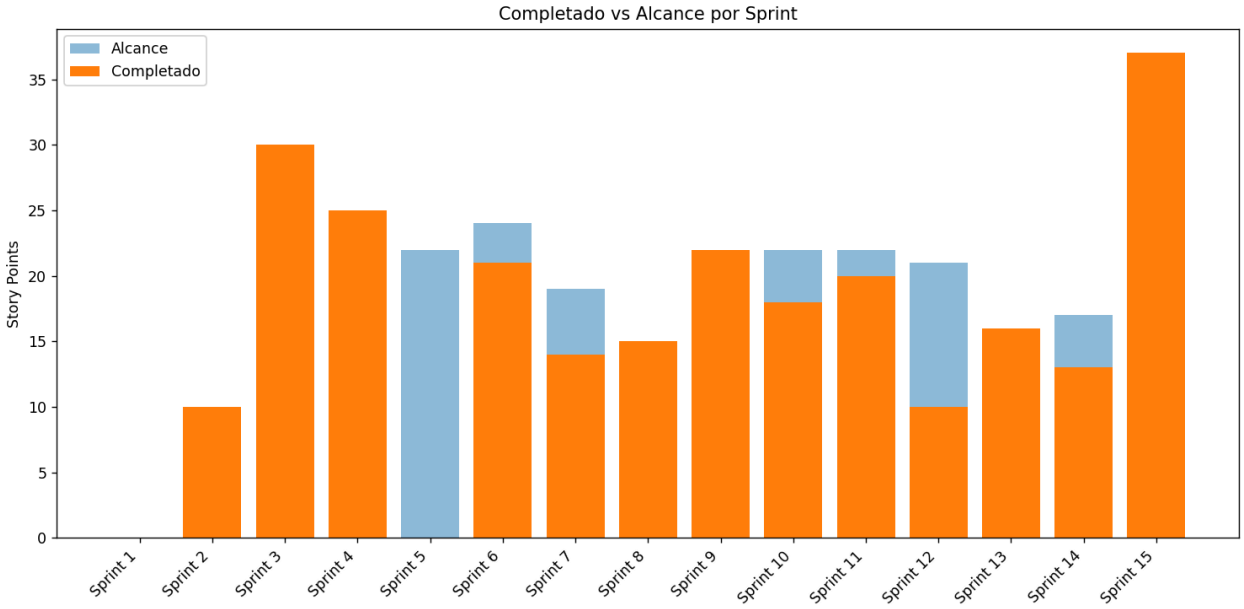
13.14. Anexo 14 - Ejemplo de retrospectiva “el bueno, el malo, y el feo”



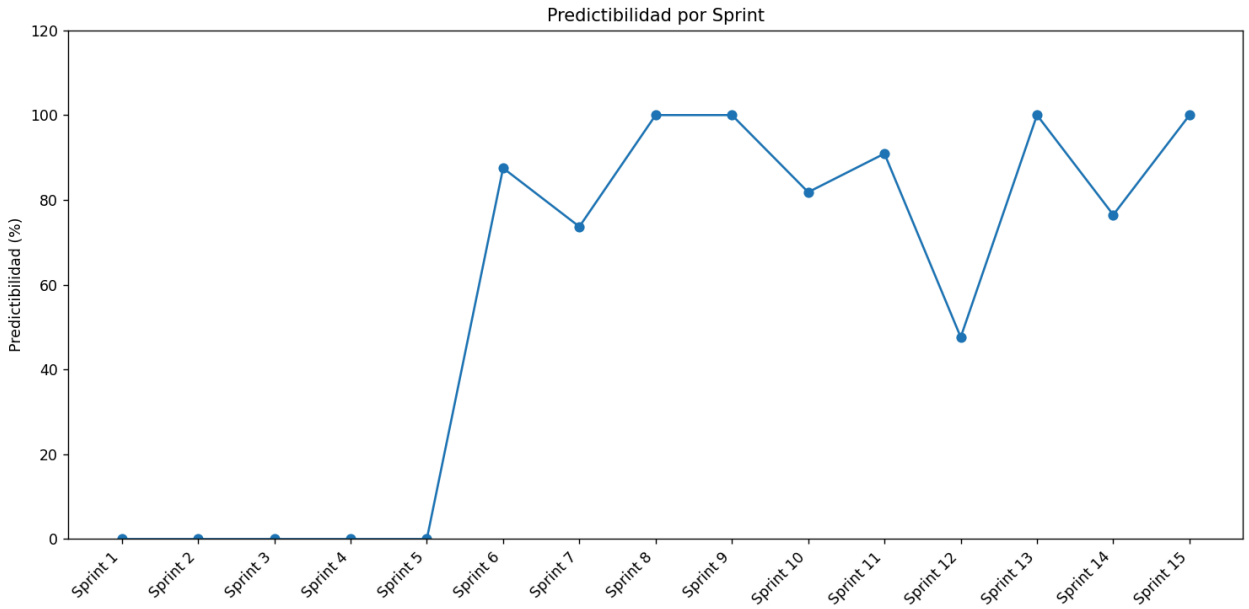
13.15. Anexo 15 - Ejemplo de retrospectiva “tres cerditos”



13.16. Anexo 16 - Gráfica de trabajo completado vs estimado por *sprint*



13.17. Anexo 17 - Gráfica de predictibilidad por *sprint*



13.18. Anexo 18 - Matriz de análisis de riesgo

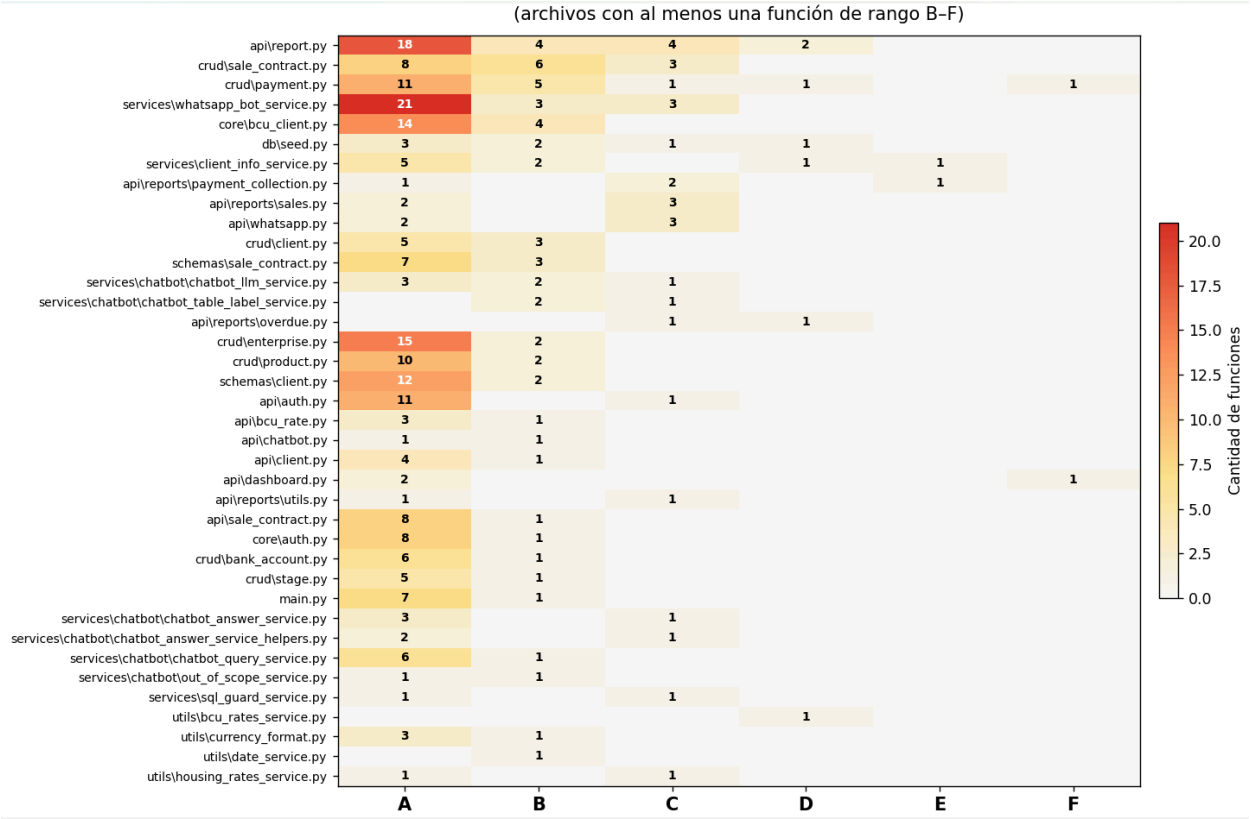
Actividad	**Probabilidad**	**Impacto**	**Magnitud**	**Respuesta**
Bot de WhatsApp	Probable	Bajo	Riesgo moderado	Investigación previa
Infraestructura en AWS	Bastante probable	Moderado	Riesgo alto	Investigación previa, dedicación exclusiva
Sistematización de Excel	Probable	Moderado	Riesgo moderado	Realizar pruebas de validación
Tasa de vivienda	Muy probable	Bajo	Riesgo moderado	Investigación previa, evaluación de alternativas

13.19. Anexo 19 - Resumen de métricas de calidad

quality_summary.txt

```
Quality Metrics Summary (app/)

Average cyclomatic complexity: 3.83
Functions/classes with CC > 15: 24
Average maintainability index: 75.87

Modules with highest outgoing dependencies:
  app.db.db: 27
  app.db: 26
  app.utils: 24
  app.utils.timezone: 23
  app.schemas: 13

Most referenced modules (incoming dependencies):
  app.crud.sale_contract: 19
  app.models: 18
  app.crud.payment: 17
  app.db.seed: 13
  app.crud.product: 10

Artifacts:
  radon_cc_report.txt
  radon_mi_report.txt
  pydeps_graph.dot
  pydeps_graph.svg
```

13.20. Anexo 20 - Mapa de calor de Complejidad Ciclomática



13.21. Anexo 21 - Gráfica de Índice de Mantenibilidad por archivo