Universidad ORT Uruguay
Facultad de Ingeniería

Feedback Assistant:

Sistema de retroalimentación continua

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

Bruno Bellizzi – 178712
Nicolás Eiris – 182713
María Inés Fernández – 186503

Tutor: Leonardo Scafarelli

2018

Declaración de auditoría

Nosotros, Bruno Bellizzi, Nicolás Eiris y María Inés Fernández, declaramos que el trabajo que se presenta en esta obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos el proyecto de grado de la carrera Ingeniería en Sistemas;

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

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

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

- Cuando la obra se basa en trabajos realizados 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.

Bruno Bellizzi	Nicolás Eiris	María Inés Fernández
18 de Agosto de 2018	18 de Agosto de 2018	18 de Agosto de 2018

Agradecimientos

A nuestras familias y amigos, quienes nos motivaron y apoyaron incondicionalmente a lo largo de estos años de carrera.

A Leonardo Scafarelli, quien con su profesionalismo buscó sacar lo mejor de cada uno de nosotros, compartiendo su conocimiento y guiándonos permanentemente hacia el camino correcto. Reconocer también a los docentes Gastón Mousqués y Sergio Yovine por los aportes brindados ante dudas de distintos aspectos tecnológicos.

Por último a Pablo Arreche, quien en su rol de product owner se encontró siempre a disposición siendo un gran nexo entre el equipo y la empresa cliente.

A todos ustedes, muchas gracias.

Abstract

El objetivo de Feedback Assistant es incentivar a los empleados y a sus asesores de talento a proveer retroalimentación en tiempo real dentro de la organización a la cual pertenecen. Para lograr esto, el acto de requerir o proveer retroalimentación tiene que ser lo más natural y libre de esfuerzo posible, y el mecanismo para hacerlo tiene que estar ubicado donde el usuario pasa la mayor parte de su tiempo. Por ejemplo, para una gran cantidad de empresas que eligieron la plataforma G-Suite de Google, ese lugar es Gmail, las herramientas de calendario o alguna de las restantes aplicaciones de G-Suite. Esto mismo aplica para otras plataformas corporativas como Office 365 o SalesForce.

El cliente es una empresa estadounidense líder en desarrollo de software para la gestión del capital humano. Dentro de sus principales productos se encuentra TMS (Talent Management System), un sistema de gestión de talento laboral. Feedback Assistant se integra con TMS, aportando valor a éste enviandole el procesamiento asociado al análisis del sentimiento del *feedback* provisto por empleados y asesores de talento dentro de la organización.

Se utilizó una metodología ágil debido a las características del proyecto. Mediante videoconferencias semanales con el cliente como actividad fundamental se lograron relevar los requerimientos. Si bien la mayoría de ellos fueron definidos al comienzo del proyecto, a lo largo de éste se fueron agregando nuevos requerimientos negociados con el cliente de forma de aportar mayor valor al mismo.

El flujo principal de la solución contempla la sincronización con los calendarios de los empleados, detectando cuándo finalizan sus eventos y solicitando *feedback* a los asistentes de los mismos mediante distintos canales de comunicación. Luego, el *feedback* provisto se analiza utilizando tecnologías de procesamiento de lenguaje natural y los resultados son enviados al producto del cliente como fue mencionado previamente.

La arquitectura consiste en un ecosistema de servicios distribuidos. Uno de los atributos de calidad más importantes que el equipo decidió priorizar fue la

extensibilidad, debido a que en el futuro se busca integrarse con otras tecnologías para recibir *feedback*, como ser Slack, Whatsapp o Microsoft Teams.

Se ha logrado construir un producto capaz de generar un cambio radical sobre las metodologías tradicionales de evaluación del desempeño, generando datos certeros y objetivos de forma contínua para lograr potenciar las habilidades de los individuos y su crecimiento dentro de la organización a la que pertenecen.

Palabras clave

Feedback; Feedback Assistant; Manejo de talentos; Asesores de talento; Ascentis; Java; Spring Framework; Inteligencia Artificial; Procesamiento de Lenguaje Natural; Google G-Suite; Office 365; Microsoft Azure; IBM Watson; Microservicios; *Scrum;* Universidad ORT Uruguay.

Índice

1 Introducción

En este capítulo se abordarán generalidades del proyecto, abarcando la selección del mismo, los distintos objetivos que se fijaron, la descripción del cliente, la composición del equipo y la estructura del presente documento.

1.1 Selección de proyecto

Luego de evaluar todas las propuestas presentadas en la feria de proyectos de la Universidad ORT Uruguay, el equipo optó por el proyecto Feedback Assistant. La posibilidad de aplicar tecnologías de última tendencia como lo es la Inteligencia Artificial (IA) incentivó dicha elección. Este campo ha tenido un crecimiento exponencial en los últimos años, y los referentes de la industria afirman que revolucionará la mayoría de los procesos de negocio en la próxima década [1]. El Procesamiento de Lenguaje Natural, rama de la IA, representaba un área desconocida para el equipo, lo que motivó enormemente la elección de este proyecto.

Otro factor que incidió en la elección de Feedback Assistant fue la procedencia del cliente (estadounidense), añadiendo de esta forma un desafío adicional de comunicación. También resultó interesante aprender sobre su proceso de negocio y experimentar la forma de trabajo de una organización con cultura estadounidense.

1.2 Objetivos

A continuación se detallarán los distintos objetivos que el equipo se planteó para el proyecto, el producto y lo académico.

1.2.1 Objetivos del proyecto

Cumplir y superar las expectativas del cliente

El objetivo principal de este proyecto fue agregar el mayor valor posible a la *suite* de productos para la gestión del capital humano que distribuye el cliente, buscando incentivar a los empleados y a sus respectivos asesores de talento a utilizar con mayor frecuencia las herramientas de trabajo que el mismo provee. Para evaluar si el valor agregado cumplió con las expectativas del cliente, se realizaron distintas encuestas midiendo la satisfacción del mismo respecto al trabajo realizado (ver sección 7.3.7 Satisfacción del cliente).

Construcción de un Producto Mínimo Viable

El equipo se planteó cumplir con las metas establecidas por el cliente. Estas metas consisten en la construcción de un producto mínimo viable (MVP) para luego seguir agregándole funcionalidades. Asimismo el equipo apuntó a superar este MVP.

Profesionalismo

Se buscó llevar a cabo una metodología de trabajo idéntica a las aplicadas en el mercado laboral, con el fin de generar un producto de calidad que le aporte real valor al cliente. También se apuntó a que éste sea capaz de utilizar y mantener el producto en etapas posteriores a las del proyecto. Para esto se llevó a cabo un proceso de ingeniería de software estandarizado donde los integrantes del equipo fueron capaces de gestionar y medir la calidad a lo largo del proyecto, utilizando distintas métricas que serán detalladas en el presente documento.

1.2.2 Objetivos académicos

Aplicación de conocimientos

Aplicar los conocimientos adquiridos en las distintas cátedras a lo largo de la carrera, buscando converger estos conocimientos en un único proyecto.

Aprobar el proyecto de grado como requisito final de la carrera

Este proyecto surge como requerimiento para la obtención del título de Ingeniero en Sistemas. De aquí la motivación por aprobar esta evaluación con una calificación de excelencia.

Autogestión

A pesar de contar con el soporte del tutor, se aprovechó esta instancia para poner a prueba la autosuficiencia del equipo en aplicar una metodología de trabajo que permita realizar un proyecto de este calibre, demostrando así la capacidad del equipo en llevar a cabo proyectos de esta índole luego de adquirida esta experiencia.

1.2.3 Objetivos del equipo

Filosofía de trabajo

Para lograr construir un producto de alta calidad, el equipo procuró ejecutar las tareas siguiendo una filosofía de trabajo en la cual se destacan los siguientes valores y actitudes:

- Proactividad
- Compromiso
- Reflexión
- Flexibilidad
- Compañerismo
- Superación

Adquisición de nuevos conocimientos

En este proyecto no se buscó únicamente aplicar los conocimientos adquiridos hasta el momento, sino también utilizar esta instancia formal para obtener nuevos conocimientos técnicos y de negocio que se están utilizando actualmente en el mercado de las TI (Tecnologías de la Información).

Se buscó incursionar en la rama del Procesamiento de Lenguaje Natural. Por lo tanto, fue necesario realizar una investigación para interiorizarse en el tema y ver las soluciones existentes en el mercado. De esta manera el equipo fue capaz de aplicar la mejor solución al problema.

Cumplimiento de expectativas personales

Se buscó aprovechar esta última instancia académica para trabajar de forma profesional y concluir el ciclo universitario buscando la excelencia en todas las dimensiones del proyecto.

1.3 Descripción del cliente

Ascentis es una compañía estadounidense que desarrolla soluciones del estilo Software-as-a-Service para facilitar la gestión del Capital Humano, cuyos principales mercados son Estados Unidos y Canadá. Durante los últimos 30 años se han dedicado a ayudar a las pequeñas y medianas empresas a maximizar el retorno de una de sus inversiones más importantes, el Capital Humano. Ascentis cree que la forma más efectiva de lograr este objetivo es mejorar la experiencia laboral de los empleados dentro de las organizaciones a las que distribuye sus soluciones. Para ello, ofrecen una *suite* de productos que simplifica las interacciones con los departamentos de Recursos Humanos de las empresas poniendo a disposición toda la información corporativa clave, ayudando así a desarrollar y mejorar las habilidades profesionales e interpersonales, promoviendo la comunicación asertiva dentro de la organización.

En octubre del año 2017, la empresa expande sus oficinas abriendo una nueva dependencia en Uruguay. Ascentis se comprometió a destinar recursos tanto en Uruguay como en Estados Unidos para fortalecer la mutua colaboración con el equipo y la universidad. El compromiso y atención por parte de Ascentis han sido excelentes a lo largo de todo el proyecto, mostrándose siempre a disposición ante las necesidades del equipo.

1.4 Descripción del equipo de trabajo

El equipo está integrado por tres estudiantes avanzados de Ingeniería en Sistemas, fueron compañeros reiteradas veces durante la carrera y están acostumbrados a trabajar juntos, aspectos que permitieron una comunicación fluida y un entorno de confianza.

Los integrantes son:

- Bruno Bellizzi
- Nicolás Eiris
- María Inés Fernández
- Leonardo Scafarelli (Tutor)

1.5 Entorno conceptual de Software Factory y sus objetivos

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.

1.6 Estructura del documento

Capítulo 1: Introducción

Pretende introducir al lector sobre los aspectos más generales del proyecto, como la selección del mismo, la composición del equipo y una breve descripción del cliente. Además presenta los distintos objetivos que el equipo se propuso cumplir.

Capítulo 2: El problema y la solución

Expone el contexto del problema junto a la solución planteada y sus principales características.

Capítulo 3: Ingeniería de requerimientos

Se describen las actividades realizadas dentro de la ingeniería de requerimientos, incluyendo cómo se relevaron los mismos y las técnicas aplicadas para validarlos. Además se listan los requerimientos funcionales y no funcionales relevados.

Capítulo 4: Descripción de la solución de *software*

Abarca tanto la presentación de la arquitectura a alto nivel, como la descripción detallada desde distintos puntos de vista de la arquitectura resultante. Además expone los patrones de diseño utilizados y las tácticas aplicadas asociadas a favorecer cada atributo de calidad.

Capítulo 5: Análisis tecnológico

Presenta las discusiones y decisiones que enfrentó el equipo a la hora de seleccionar las tecnologías a utilizar, tomando en cuenta los costos y beneficios que las mismas proveen respecto a la realidad del proyecto. Asimismo, expone los principales desafíos técnicos a los cuales el equipo se enfrentó.

Capítulo 6: Gestión de proyecto

Explica el proceso de gestión aplicado. Se detalla el marco de trabajo, ciclo de vida, la metodología que se aplicó y las herramientas de gestión utilizadas. Adicionalmente se describe la planificación y evolución de los distintos *sprints*, junto con la gestión del alcance y esfuerzo. Por otro lado, se muestran los riesgos identificados, los planes de acción para cada uno de ellos y su evolución a lo largo del tiempo.

Capítulo 7: Gestión de calidad

Se especifican las actividades realizadas para el aseguramiento de la calidad tanto del producto como del proceso. Se da a conocer el plan de calidad y cómo se llevaron a cabo sus actividades.

Capítulo 8: Gestión de la configuración

Se detallan las herramientas utilizadas para la configuración del *software*. Incluye los mecanismos para la administración del control de versiones, la gestión de defectos y control de cambios.

Capítulo 9: Conclusiones

Por último se expondrán las conclusiones finales del proyecto y las lecciones aprendidas en el proceso, dando una reflexión por parte del equipo respecto a esta experiencia de trabajo.

2 El problema y la solución

Este capítulo describe el contexto del problema a resolver, para luego exponer cuál fue la solución de alto nivel desde el punto de vista del negocio, explicando sus características fundamentales.

2.1 Contexto del problema

En los últimos años la evolución de la tecnología, sumado al hecho de que millones de *millennials* están ingresando al mercado laboral, han producido un cambio notable en cómo y dónde las empresas desarrollan sus negocios.

Mientras que diez años atrás la mayoría de las empresas ejecutaba toda su infraestructura de negocios crítica *in-house*, ahora vemos que utilizan herramientas *cloud* como Office 365 [3], G-Suite [4], SalesForce [5], etc.

Al mismo tiempo, los empleados han elevado sus expectativas con respecto a las aplicaciones corporativas. Requieren que sean rápidas, intuitivas, simples y que estén disponibles dónde y cuándo las necesiten.

Como parte de esta tendencia, se quiere explorar la integración de conceptos de manejo de talentos con plataformas de negocios en la nube. El manejo de talentos es la capacidad de una organización para reclutar, retener y producir empleados talentosos y competitivos en el mercado laboral [6]. Busca fortalecer el capital humano de forma estratégica para aumentar el valor de las organizaciones.

Una de las formas más simples y efectivas del manejo de talentos es el *feedback*. Éste puede proveer consejos en tiempo real, apoyo y correcciones. El *feedback* continuo puede reemplazar completamente las revisiones de rendimiento anuales/semestrales/trimestrales/etc. mientras mejora la comunicación entre los jefes y los empleados, así como también entre los colegas. En un ambiente de trabajo real, el *feedback* es típicamente entregado por vía verbal o por email y no es común para los individuos proveer *feedback* en forma de documentos sin que el mismo sea solicitado y vinculado a un evento significativo. Un estudio reciente de Price Waterhouse Coopers reveló que casi el 60% de los encuestados prefiere

feedback diariamente o semanalmente. Ese número aumenta al 72% para los *millennials* [7].

Dentro de la suite de productos que distribuye Ascentis, se encuentra un sistema para el manejo de talentos denominado TMS (*Talent Management System*) [8]. El mismo fue desarrollado hace muchos años y no ha recibido actualizaciones tecnológicas, por lo que existe una amenaza ante la aparición de nuevos competidores en el mercado con herramientas más innovadoras. Por esto, el cliente requiere de una solución que le brinde un valor adicional a su producto agregando interacción con tecnologías de última generación.

2.2 Objetivos generales del producto

A continuación se exponen los objetivos que se buscaron cumplir con la solución diseñada.

- **Incentivar empleados y asesores de talento a proveer feedback**
  La tarea de enviar *feedback* no es necesariamente obligatoria; por esta razón es necesario que la acción de enviar *feedback* sea sencilla e intuitiva para el usuario.

- **Fácil adición de nuevos canales de comunicación**
  Debido a la gran variedad de canales de comunicación que utilizan las organizaciones, el producto debe permitir agregar nuevas formas de proveer *feedback* con facilidad. Para lograr este objetivo se hizo énfasis en la extensibilidad de la arquitectura, la cual se expone en el capítulo 4. Descripción de la solución.

- **Autosuficiencia**
  Se busca minimizar la dependencia de un administrador del sistema, eliminando la mayor cantidad de interacciones manuales necesarias para que el producto continúe funcionando correctamente. Por ende, Feedback Assistant debe funcionar de manera autónoma, manteniéndose permanentemente sincronizado con los datos del capital humano de la organización.

- **Personalización**

  Considerando que los usuarios finales serán los miembros de las organizaciones clientes de Ascentis, se requiere que Feedback Assistant cumpla con la propiedad de ser *white label*. Esto implica que la estética pueda adaptarse a gusto del consumidor, generando sensación de pertenencia por parte de la organización.

2.3 Principales aspectos de la solución

Feedback Assistant tiene el propósito de lograr una retroalimentación más fluida dentro de las organizaciones. Para ello, el mismo se integra con las herramientas que los empleados utilizan a diario, como el correo electrónico y el calendario.

Para más de 5 millones de organizaciones [9] estas herramientas son Gmail [10] y Google Calendar [11], posicionando a las plataformas de trabajo de Google como referentes en el mercado.

Según la experiencia del cliente, los mejores *feedbacks* surgen luego de eventos con la participación de múltiples asistentes, como reuniones de trabajo, presentaciones de productos, informes de avance, exposición de resultados comerciales y transferencia de conocimientos. Todos estos suelen agendarse en los calendarios de cada asistente.

Para poner al lector en contexto, se ejemplifica a continuación un flujo básico de la aplicación con los usuarios:

- Se agenda una reunión laboral del equipo de marketing conformado por seis integrantes, donde uno de ellos expondrá una estrategia a ejecutar en la próxima campaña.

- Cinco de los integrantes aceptan la invitación mientras que uno de ellos no lo hace debido a que coincide con su licencia.

- Luego de realizada y finalizada la reunión, el sistema detecta este acontecimiento y dispara solicitudes de *feedback* a los cinco asistentes mediante correo electrónico o llamada telefónica.

- Algunos de ellos responden esta solicitud brindando su opinión respecto a la presentación realizada por el colega (en caso de respuesta por parte del expositor se considera como autoevaluación).

- Al final de la jornada el sistema envía un correo electrónico al gerente de marketing conteniendo un reporte del análisis de los feedbacks provistos. Este reporte contiene los comentarios brindados y asociado a cada uno cuán positiva o negativa fue la retroalimentación.

Todo este proceso de interacción se hace posible gracias a las funcionalidades que se detallan a continuación.

Integración con Google G-Suite

Como se mencionó anteriormente, el producto debe basarse en la integración con las herramientas laborales que se utilizan a diario. En primera instancia se hizo foco en las tecnologías de Google, por lo que la solución se integra con la plataforma Google G-Suite con el objetivo de sincronizar los datos asociados al capital humano de la organización.

Detección de eventos finalizados

Feedback Assistant tiene constante conocimiento sobre los usuarios activos dentro de la organización, así como también de los eventos en sus calendarios. Llevando una permanente sincronización de estos datos, el producto es capaz de identificar la finalización de un evento laboral para luego disparar una solicitud de *feedback* a sus asistentes.

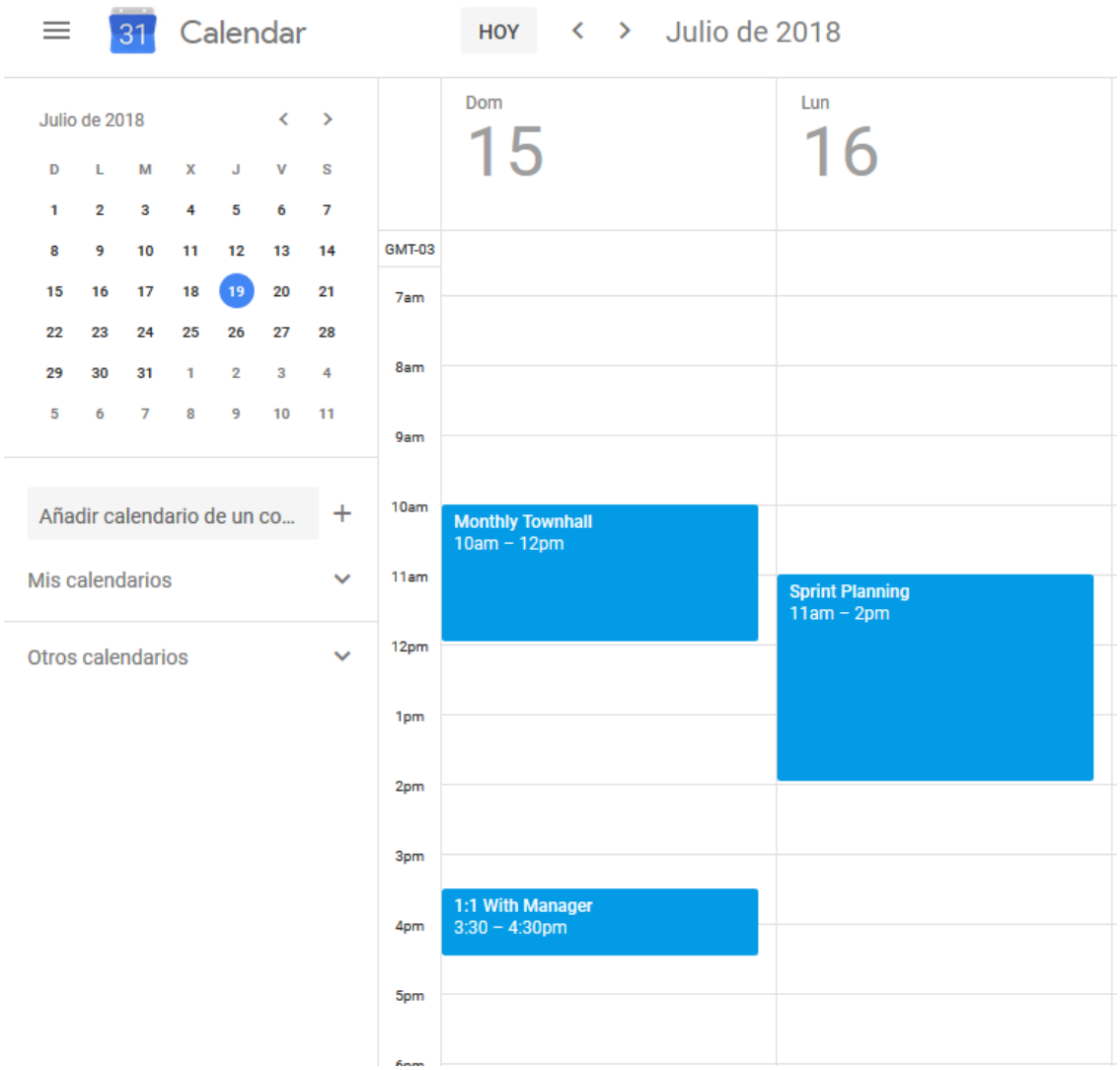


Ilustración 2.1 Visualización de eventos en Google Calendar

Solicitud de feedback vía correo electrónico

Una vez detectada la finalización de un evento, el sistema envía un correo electrónico solicitando *feedback* a todos los integrantes de la organización que hayan participado en el mismo. El correo tiene una presentación amigable para motivar al destinatario a proveer su *feedback*, conteniendo información suficiente para que se pueda identificar rápidamente a cuál evento se está haciendo referencia.

Además, el producto ofrece la opción de personalizar el diseño del correo electrónico para que la organización pueda adaptarlo a gusto, cambiando los colores, logos, textos y formato para que se adecúe a la estética deseada.

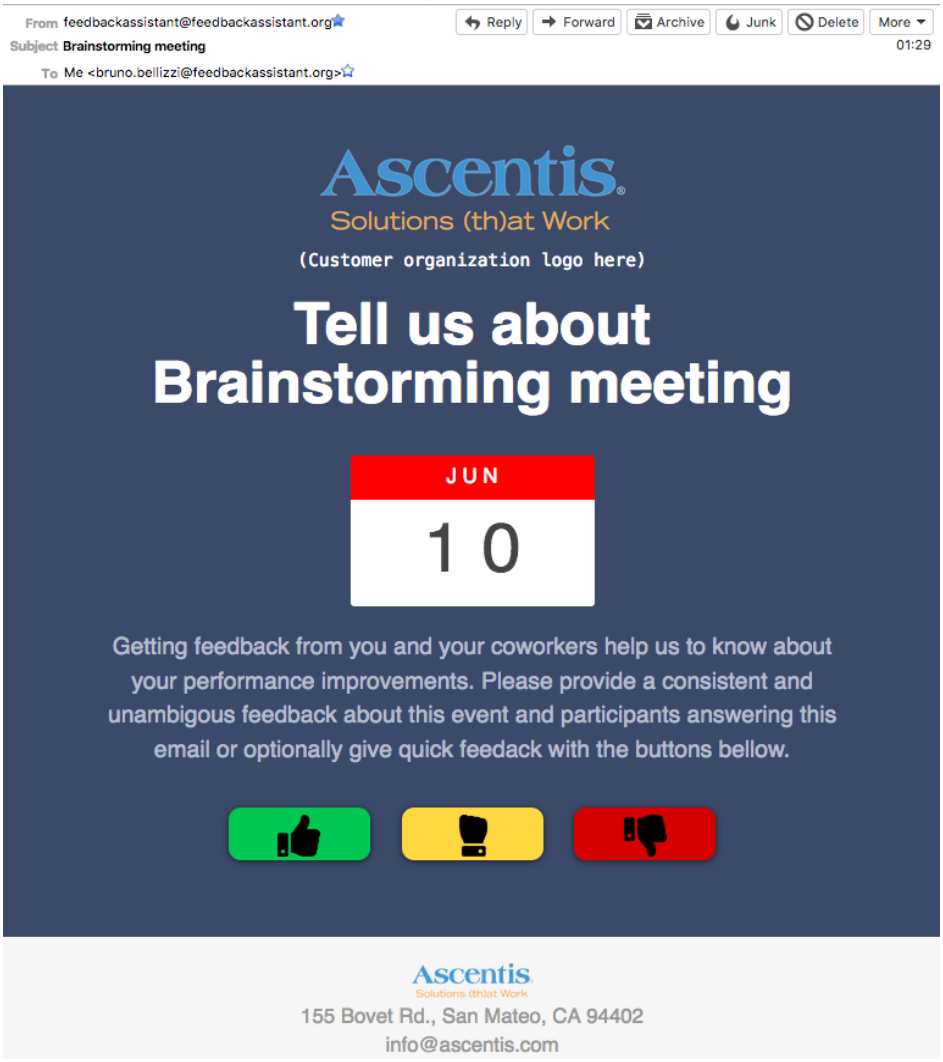


Ilustración 2.2 Solicitud de feedback mediante correo electrónico

Tomando en cuenta la necesidad de que el acto de proveer *feedback* sea lo más simple posible, la solución permite al usuario final enviar *feedback* de dos formas distintas dentro del mismo correo electrónico.

En primer lugar, el usuario puede redactar una respuesta al correo recibido, siendo esta la más efectiva ya que permite al usuario expresarse con su lenguaje natural y hacer referencia a lo que sea necesario del evento o individuo en particular.

Por otro lado, si el usuario lo desea, existe un mecanismo complementario (y vale aclarar, no excluyente) al de respuesta redactada. Este se denomina *feedback* instantáneo, el cual consiste en seleccionar una de las tres opciones que se presentan en el correo de solicitud. Con una escala de tres niveles (positivo, neutral y negativo), el usuario puede dar su opinión respecto al evento sin perder demasiado tiempo. Luego de que se selecciona una de las opciones de *feedback instantáneo,* el sistema despliega una nueva pestaña en el navegador notificando la recepción del *feedback* provisto. Esta notificación es también personalizable por la organización.

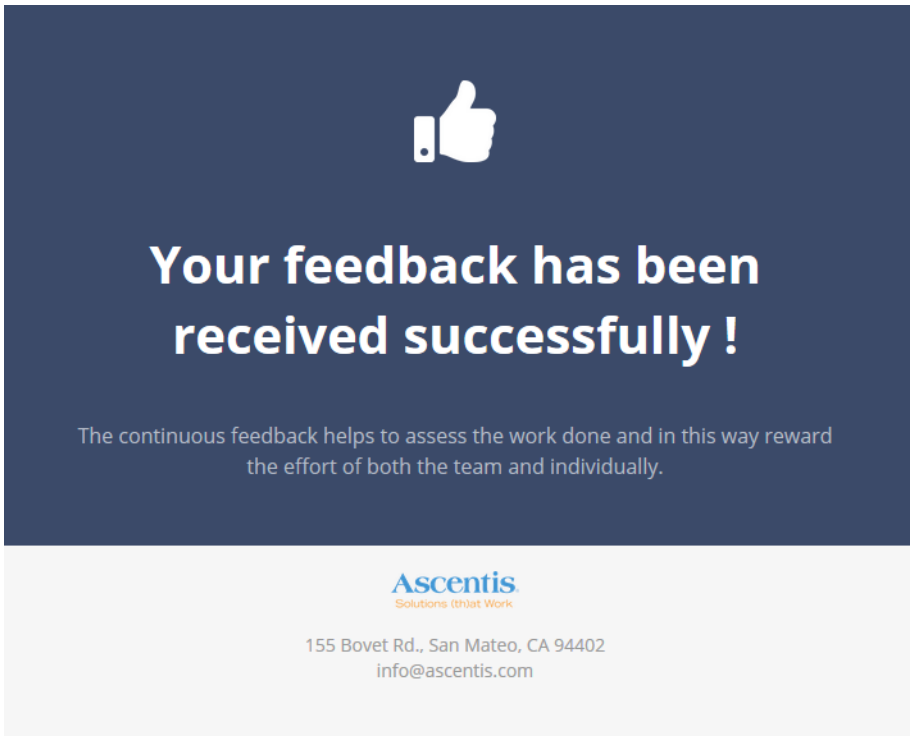


Ilustración 2.3 Notificación de recepción exitosa para feedback instantáneo.

Solicitud de feedback vía llamada telefónica

El equipo toma la decisión de aportar aún más valor a la solución, agregando la funcionalidad de solicitud y recepción de *feedback* mediante una llamada telefónica (ver capítulo 3. Ingeniería de requerimientos). Cuando se detecta la finalización de cierto evento en el calendario del usuario, el sistema realiza una llamada telefónica al mismo utilizando el teléfono de contacto extraído desde G-Suite.

Cuando el usuario atiende la llamada telefónica, un *bot* responde en representación del sistema para solicitarle *feedback* indicando el evento al que se hace referencia. Por último, el sistema reconoce la finalización del diálogo, el *bot* agradece al usuario por su tiempo y le notifica la recepción exitosa o errónea del *feedback* provisto.

Ilustración 2.4 Llamada telefónica solicitando feedback.

Siguiendo con la filosofía de que los puntos de interacción con el usuario son totalmente personalizables, tanto el mensaje de solicitud como el de agradecimiento son configurables, permitiendo a cada organización cambiar el diálogo dinámico del *bot* a gusto.

Este método de solicitud podría considerarse un tanto invasivo, por lo que un administrador puede habilitar o deshabilitar este canal y configurar cuáles usuarios son candidatos válidos para recibir la llamada.

Notificación de errores en la recepción de *feedback*

Siempre se debe mantener informado al remitente en caso de que haya ocurrido cualquier tipo de falla dentro del sistema. El mismo mantiene notificado al usuario respecto al estado del *feedback*, enviándole una notificación por correo electrónico. Ascentis busca que los sistemas sean usables y confiables, desprendiéndose esta necesidad como requisito del proyecto.

Esta notificación, al igual que las anteriores, cumple con la posibilidad de poder personalizarse a gusto de la organización.

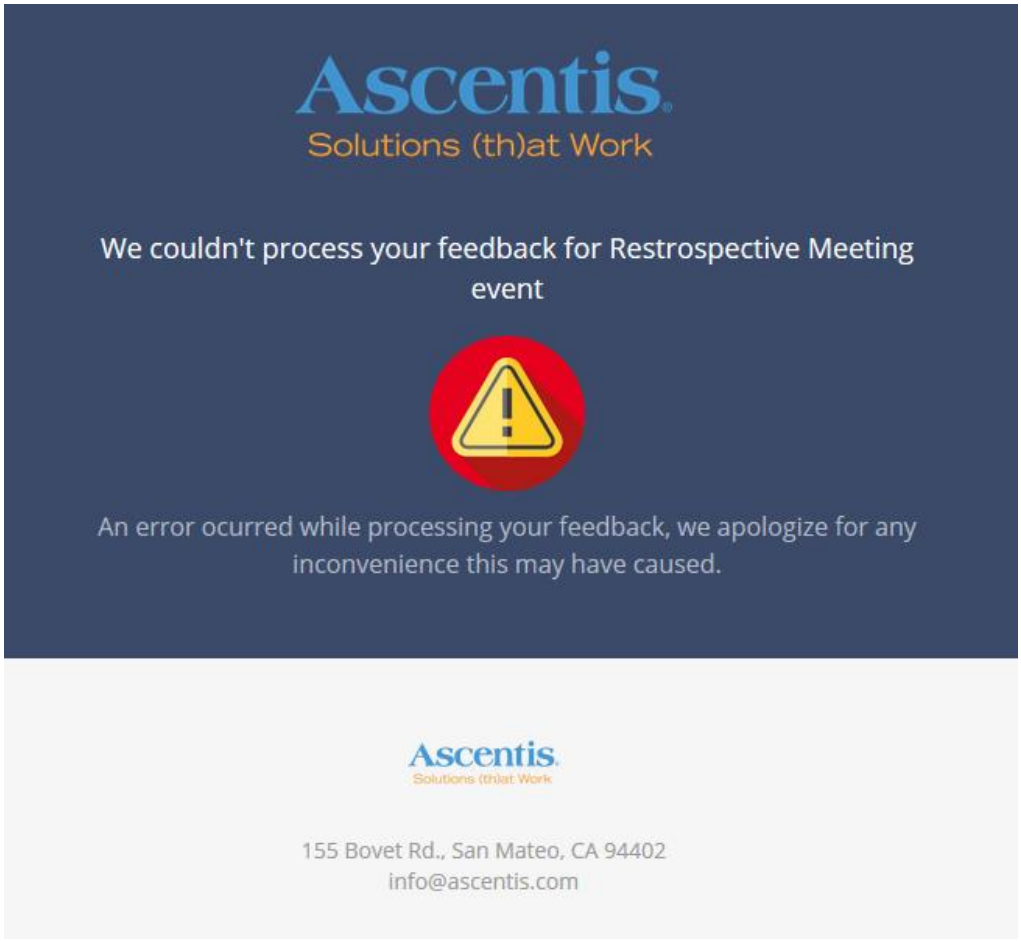


Ilustración 2.5 Notificación de recepción errónea

Procesamiento de feedback

Cualquiera sea el canal de comunicación por el cual el usuario provea *feedback*, el sistema elabora un análisis del sentimiento del mismo mediante la utilización de tecnologías para el Procesamiento del Lenguaje Natural (PLN) [12].

Este análisis es realizado mediante la integración con los mejores servicios de PLN del mercado (ver capítulo 5. Análisis tecnológico). Los resultados de estos servicios son promediados para lograr un análisis más preciso y de esta forma disminuir el margen de error (no olvidar que el lenguaje natural tiende a ser ambiguo).

En el caso de la llamada telefónica existe un desafío adicional de transcribir el mensaje grabado a texto, para que luego continúe su procesamiento como si hubiese sido redactado de forma escrita.

El contenido del *feedback* es categorizado en uno de los siguientes grupos:

- Muy negativo
- Negativo
- Neutral
- Positivo
- Muy positivo

Teniendo en cuenta que éste puede ser ambiguo, se registra también la precisión del análisis indicando qué tan seguro está respecto a la categorización realizada.

Un valor adicional que se le agrega a este aspecto de la solución es la capacidad de procesar *feedbacks* en múltiples idiomas.

Integración con TMS

Dado que el propósito principal es agregarle un valor diferencial al producto del cliente, Feedback Assistant se integra con TMS volcando a éste todo el análisis realizado.

Generación de reportes

El sistema cuenta con la capacidad de proporcionar a los gerentes un correo electrónico diario con el detalle de lo analizado. Éste contiene para cada feedback recibido en el día, el evento, el remitente y el sentimiento asociado. El propósito es poder comprimir la información y mostrarla de una forma sencilla y visual.

También se generan reportes semanales y mensuales, donde se expone para cada persona el sentimiento promedio y la cantidad de *feedbacks* provistos. La utilidad del mismo es poder visualizar las tendencias en un determinado tiempo.

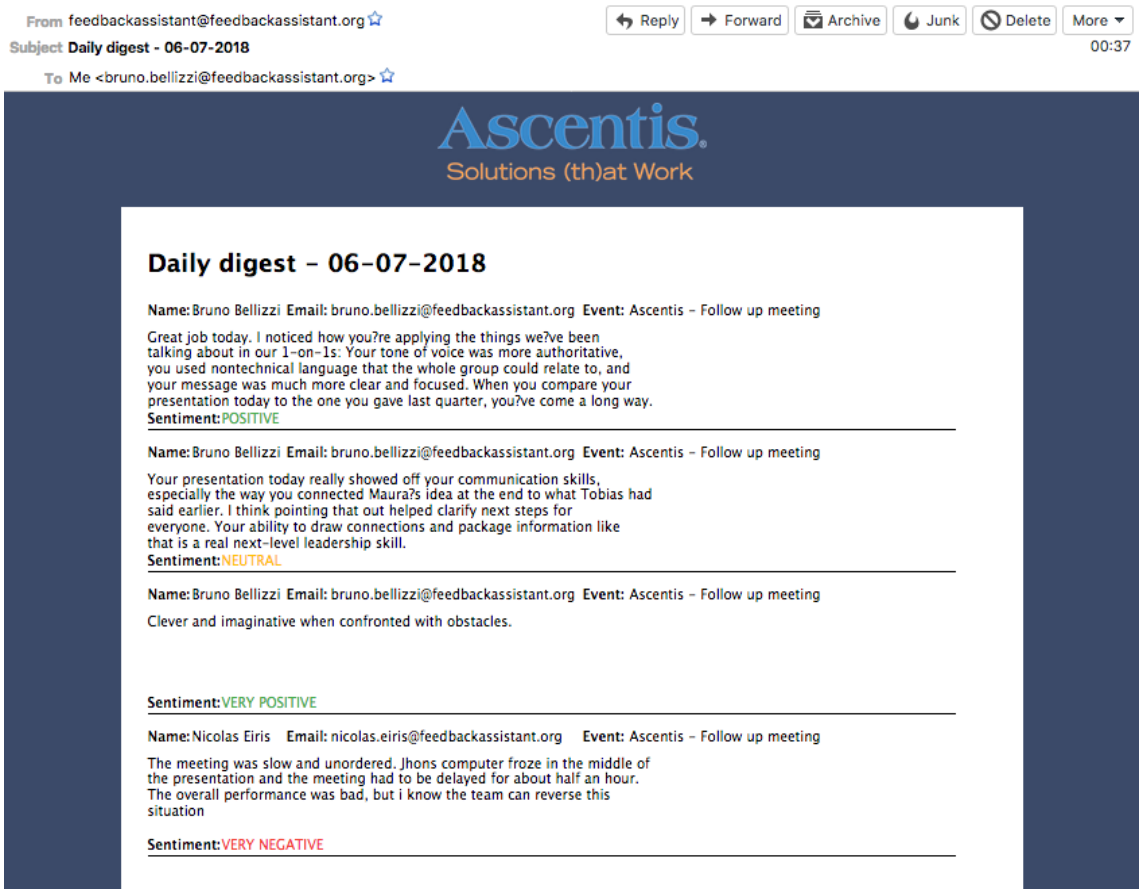


Ilustración 2.6 Reporte diario

3 Ingeniería de requerimientos

En el presente capítulo se describen las actividades que fueron llevadas a cabo para realizar el relevamiento y posterior validación de los requerimientos del sistema.

En primera instancia se presentan los métodos utilizados para la obtención y análisis de los requerimientos, así como los mecanismos de comunicación utilizados para ello. Luego se explican las tácticas utilizadas para la validación de los requerimientos relevados.

Posteriormente se dan a conocer propuestas adicionales presentadas por el equipo para agregar mayor valor al producto solicitado por el cliente.

Finalmente se detallan los distintos requerimientos funcionales, no funcionales y restricciones identificados durante las etapas de relevamiento, junto con la prioridad acordada con el cliente.

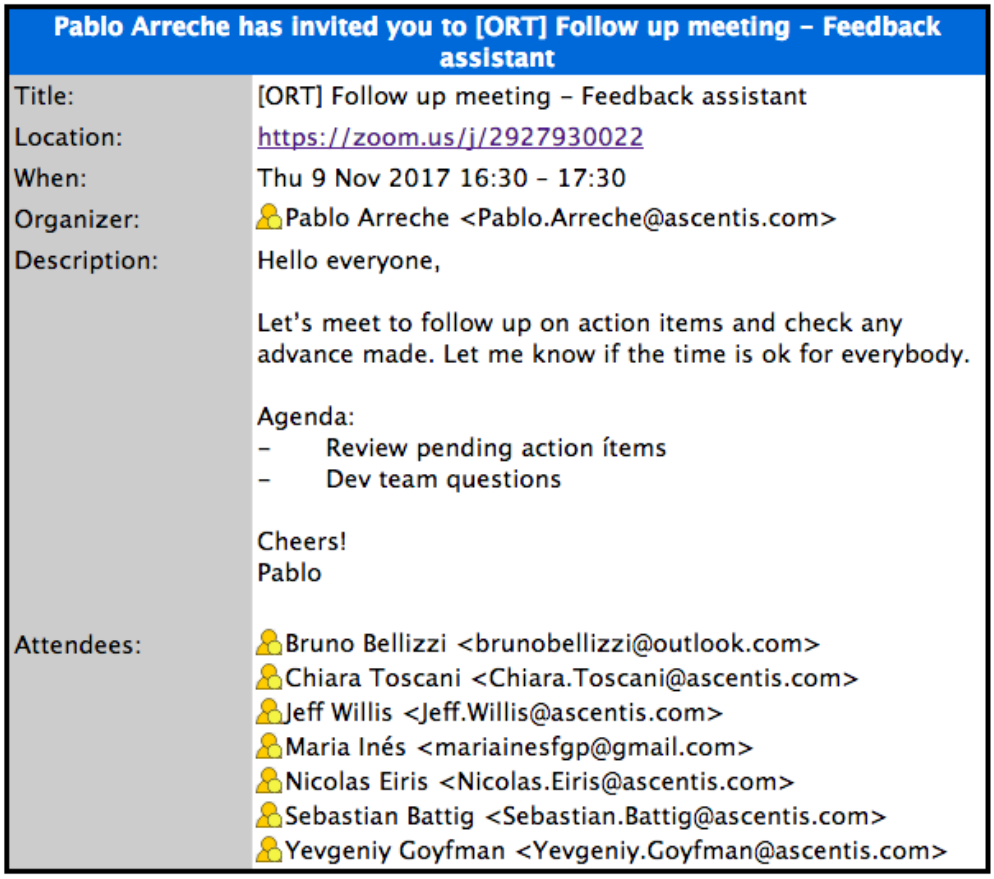
3.1 Relevamiento de requerimientos

El primer acercamiento al problema fue mediante un documento formal entregado por el cliente previo a que la Universidad ORT Uruguay apruebe el proyecto al equipo. Este documento especificaba la solución que el cliente pretendía obtener junto al contexto que lo motivaba a realizarlo.

Si bien el documento introducía la idea general de forma clara, no se profundizaba en aspectos y detalles importantes para lograr entender mejor qué era lo que el cliente esperaba obtener, motivo por el cual el equipo solicitó una reunión presencial para profundizar en el tema. Esta reunión se llevó a cabo junto a Sebastián Battig (gerente general de Ascentis Uruguay), quien explicó con mayor detalle el proyecto propuesto y aclaró algunas de las dudas del equipo. Luego de esta instancia Ascentis y el equipo formalizaron la relación de trabajo y se dió comienzo al proyecto en cuestión.

Videoconferencias semanales

Para las actividades de relevamiento se consideró que si bien el cliente se encontraba abriendo una oficina en Uruguay, los expertos del dominio radican en Estados Unidos, motivo por el cual las siguientes interacciones fueron realizadas mediante correos electrónicos y videoconferencias.



Pablo Arreche has invited you to [ORT] Follow up meeting – Feedback assistant

Title:	[ORT] Follow up meeting – Feedback assistant
Location:	https://zoom.us/j/2927930022
When:	Thu 9 Nov 2017 16:30 – 17:30
Organizer:	Pablo Arreche <Pablo.Arreche@ascentis.com>
Description:	Hello everyone, Let's meet to follow up on action items and check any advance made. Let me know if the time is ok for everybody. Agenda: – Review pending action ítems – Dev team questions Cheers! Pablo
Attendees:	Bruno Bellizzi <brunobellizzi@outlook.com> Chiara Toscani <Chiara.Toscani@ascentis.com> Jeff Willis <Jeff.Willis@ascentis.com> Maria Inés <mariainesfgp@gmail.com> Nicolas Eiris <Nicolas.Eiris@ascentis.com> Sebastian Battig <Sebastian.Battig@ascentis.com> Yevgeniy Goyfman <Yevgeniy.Goyfman@ascentis.com>

Ilustración 3.1 Correo electrónico de coordinación de reunión

Se pactaron reuniones semanales para poder comprender aún más los objetivos que se buscaban cumplir y las funcionalidades que se deseaban obtener. Estas reuniones fueron de suma importancia para comprender el problema en su totalidad y familiarizarse con los conceptos de manejo de talento involucrados, así como también para conocer el producto de Ascentis (TMS) con el que se debía integrar la solución.

El equipo se reunía por lo menos una vez antes de cada reunión con el cliente para repasar los temas y preguntas a tratar, con el fin de que estas reuniones fueran lo

más provechosas posibles teniendo en cuenta que su duración estaba limitada a una hora.

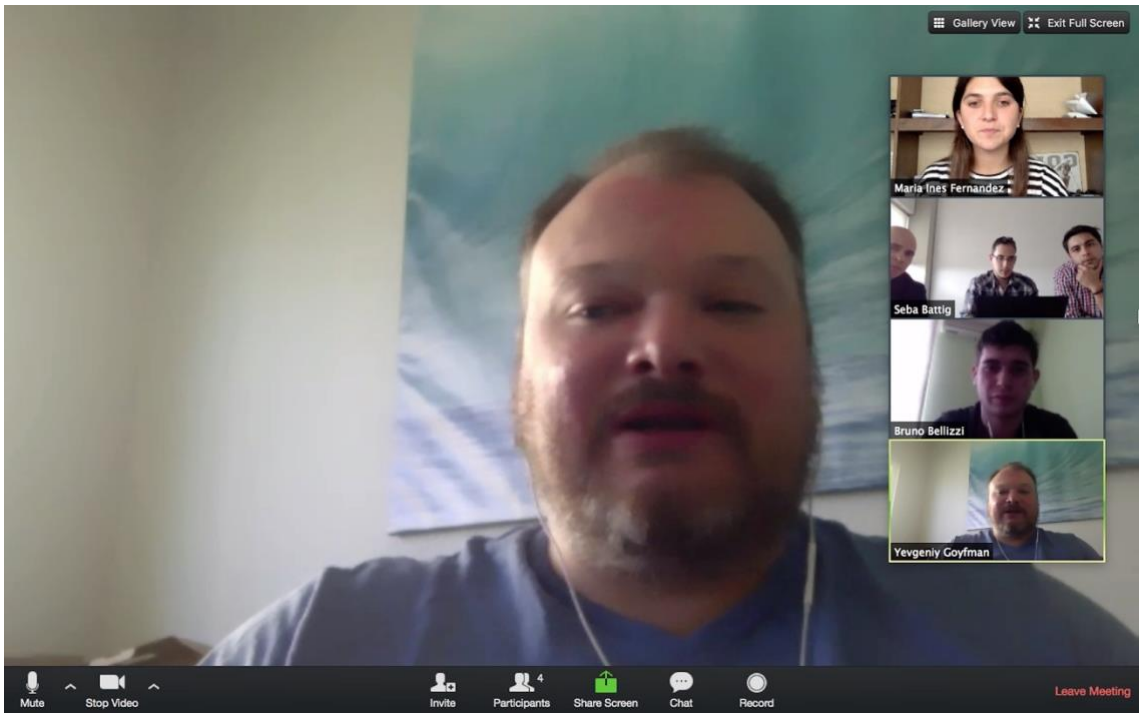


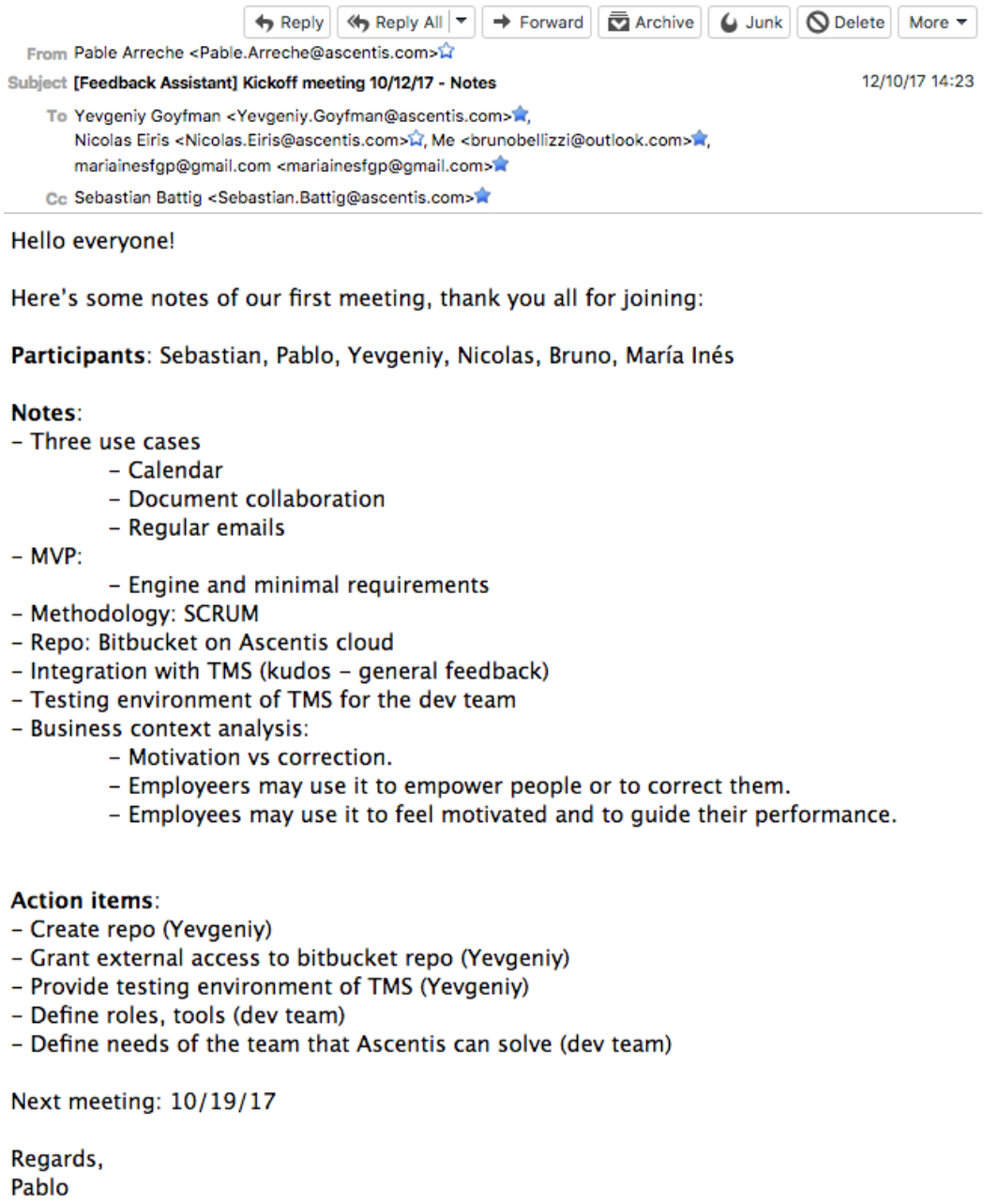
Ilustración 3.2 Videoconferencia con el cliente

Es relevante dejar en claro de que toda comunicación con el cliente debió realizarse en inglés. Afortunadamente el dominio de este idioma por parte de los integrantes no generó mayores dificultades, pudiéndose expresar de manera fluida y comprender los temas discutidos.

Las reuniones también se aprovecharon para detectar fortalezas y debilidades de la solución desde el punto de vista comercial, discutiéndose temas cómo por qué un empleado o asesor de talento usaría Feedback Assistant y por qué no. Según la experiencia del cliente, la fortaleza de la solución radica en la motivación que provocaría en los empleados para mejorar su desempeño y en la utilidad para los asesores de talento en poder corregir o incentivar a sus empleados a trabajar de mejor manera.

Luego de cada videoconferencia se realizaba un acta donde se exponían los temas tratados y las acciones pendientes para la próxima reunión, la cual era enviada a todos los participantes mediante correo electrónico. Las mismas fueron de gran

utilidad para organizar las tareas pendientes entre el cliente y el equipo así como también para dejar registro de los temas tratados en cada reunión.



Reply | Reply All | Forward | Archive | Junk | Delete | More

From Pable Arreche <Pable.Arreche@ascentis.com>

Subject **[Feedback Assistant] Kickoff meeting 10/12/17 - Notes** 12/10/17 14:23

To Yevgeniy Goyfman <Yevgeniy.Goyfman@ascentis.com>,
Nicolas Eiris <Nicolas.Eiris@ascentis.com>, Me <brunobellizzi@outlook.com>,
mariainesfgp@gmail.com <mariainesfgp@gmail.com>

Cc Sebastian Battig <Sebastian.Battig@ascentis.com>

Hello everyone!

Here's some notes of our first meeting, thank you all for joining:

Participants: Sebastian, Pablo, Yevgeniy, Nicolas, Bruno, María Inés

Notes:
- Three use cases
    - Calendar
    - Document collaboration
    - Regular emails
- MVP:
    - Engine and minimal requirements
- Methodology: SCRUM
- Repo: Bitbucket on Ascentis cloud
- Integration with TMS (kudos – general feedback)
- Testing environment of TMS for the dev team
- Business context analysis:
    - Motivation vs correction.
    - Employeers may use it to empower people or to correct them.
    - Employees may use it to feel motivated and to guide their performance.

Action items:
- Create repo (Yevgeniy)
- Grant external access to bitbucket repo (Yevgeniy)
- Provide testing environment of TMS (Yevgeniy)
- Define roles, tools (dev team)
- Define needs of the team that Ascentis can solve (dev team)

Next meeting: 10/19/17

Regards,
Pablo

Ilustración 3.3 Acta de reunión

Sesiones de brainstorming

Mientras se iba recabando información en las reuniones con Ascentis, el equipo se reunía para realizar sesiones de brainstorming y discutir lo relevado. De esta manera se iban identificando requerimientos funcionales y no funcionales de la solución a implementar. Durante estas sesiones se realizaba una puesta en común de lo discutido con el cliente y se anotaban diferentes dudas que iban surgiendo sobre el problema.

Investigación del producto TMS

Se realizó una investigación sobre las funcionalidades que brinda el producto del cliente y a qué apunta en cuanto a aspectos de negocio, de forma de poder generar una solución que aporte real valor a Ascentis. Para esto se utilizó un *webinar* provisto por el cliente en donde se explica el producto a integrarse y como se utiliza [13]. Para obtener mayor información acerca de esta investigación ver Anexo 11.1 Investigación del producto TMS.

Resultado del relevamiento

Luego de realizadas las actividades explicitadas previamente, se procedió a generar un documento tentativo de requerimientos funcionales y no funcionales. Esto permitió converger todas las necesidades identificadas durante las distintas actividades en un único documento para ser luego discutido y validado con el cliente. El mismo puede encontrarse en el Anexo 11.2 Documento de validación de requerimientos.

3.2 Validación de requerimientos

Ésta validación tuvo como objetivo asegurar que los requerimientos relevados reflejaran realmente lo que el cliente solicitaba. Consistieron principalmente de dos actividades:

Documento de validación de requerimientos

El documento de validación de requerimientos mencionado en la sección anterior fue enviado por correo electrónico al cliente solicitando que sea analizado para

poder validarlo en la próxima reunión. En esta reunión los requerimientos fueron discutidos siendo aprobados o no por el cliente.

Un hecho interesante que surgió de la reunión fue que el cliente rechazó los requerimientos que referían a una herramienta web auxiliar que le permitiera obtener información o realizar búsquedas sobre los *feedbacks* procesados. De aquí surge una de las restricciones asociada a la no utilización de interfaz gráfica. Ésto se debe a que la información procesada por Feedback Assistant sería utilizada como complemento al producto del cliente (TMS) siendo éste el encargado de presentar la información al usuario final.

Esta actividad tuvo como ventaja que al enviarse un documento con la descripción del sistema días antes de la reunión, el cliente pudo dedicar tiempo para leerlo y pensar si lo que se relevó era realmente lo que se buscaba, evitando así tener que tomar la decisión rápida de aceptar o no un requerimiento durante la reunión.

Prueba de concepto

Una vez validado el documento mencionado se prosiguió con otra instancia que sirvió tanto de validación como de investigación, ésta fue la generación de una prueba de concepto de la integración con GSuite. Esta prueba de concepto tuvo varios objetivos:

- Validar los requerimientos relevados, presentando al cliente el curso normal del sistema desde el punto de vista del usuario en busca de que la solución a construir cumpla con lo esperado de una forma más tangible que un documento escrito.

- Verificar la viabilidad de la solución discutida durante las etapas de relevamiento. Antes de comenzar con el desarrollo del MVP fue necesario asegurarse que era posible cumplir con lo que se estaba comprometiendo, siendo esta prueba de concepto un primer acercamiento al problema tecnológico.

- Construir un prototipo funcional evolutivo para las siguientes etapas del desarrollo. La prueba de concepto consistió en construir los primeros requerimientos del sistema pudiendo luego ser reutilizada para continuar con el desarrollo. Este prototipo consistía de una arquitectura monolítica que debía ser refactoreada para formar parte del producto final, pero ya contenía lógica que pudo ser reutilizada en las siguientes fases.

Esta prueba de concepto contuvo las siguientes funcionalidades:

- Impersonalización de usuarios de G-Suite mediante cuenta de servicio.
- Integración con Google Directory para obtener los usuarios de una organización.
- Integración con Google Calendar para obtener los calendarios de los usuarios.
- Identificación de un evento recientemente finalizado.
- Solicitud de feedback vía correo electrónico.

Para guiar la demo de la prueba de concepto se elaboró una presentación la cual puede encontrarse en el Anexo 11.3 Presentación prueba de concepto.

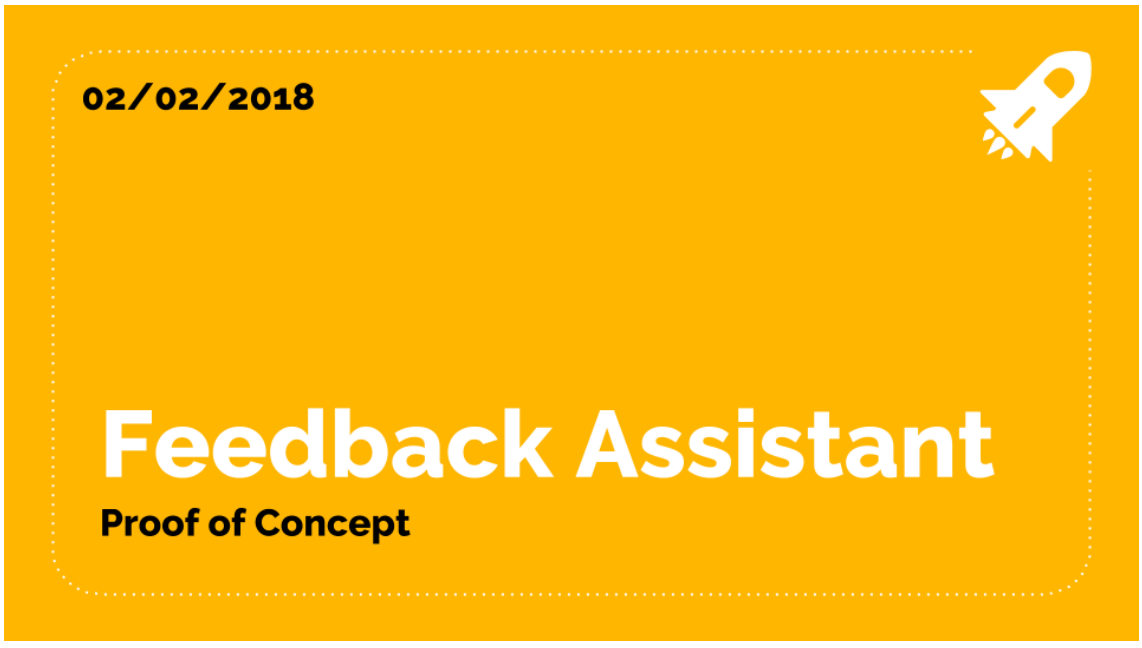


Ilustración 3.4 Presentación prueba de concepto

Además de servir para validar los requerimientos, la prueba de concepto fue de suma utilidad para la investigación de las integraciones con G-Suite. Aquí el equipo descubrió la complejidad que implicaría acceder a los datos de los usuarios de G-Suite y eventos de Google Calendar sin la previa autorización manual de cada usuario, aspecto que se profundiza en la sección 5.2 Principales desafíos tecnológicos.

Se puede concluir que ésta fue una actividad de suma importancia para la validación de requerimientos, en vista de que el cliente fue capaz de experimentar empíricamente con un prototipo operativo que le dió una idea de cómo se utilizaría el producto final. El cliente expresó que el equipo había entendido correctamente lo que se buscaba y sus comentarios fueron muy positivos, aprobando el trabajo realizado.

3.3 Propuestas adicionales

A medida que se realizaba la investigación y desarrollo de los requerimientos validados para el MVP, el equipo observó que el tiempo del proyecto daba para incluir nuevos requerimientos que aporten valor agregado a la solución acordada. Es por esto que mientras se avanzaba en el desarrollo, los integrantes ya se encontraban pensando nuevas ideas para incluir a la solución existente una vez realizada la primer entrega formal, buscando agregar alternativas innovadoras en pro de obtener mejores niveles de satisfacción por parte de Ascentis.

Se realizaron varias instancias de *brainstorming* a lo largo de las reuniones presenciales del equipo donde cada uno exponía nuevas ideas, y lo que fue muy importante, muchas veces sucedía que uno presentaba una idea y otro la complementaba aportando aún más valor a la idea inicial.

De aquí surgieron los siguientes requerimientos adicionales:

- Solicitud de feedback vía llamada telefónica.
- Análisis más preciso del sentimiento del feedback.
- Integración con Office 365.

Solicitud de *feedback* vía llamada telefónica

Se hizo mucho énfasis en la empatía con el usuario final, donde el equipo hizo foco en minimizar lo máximo posible el esfuerzo a realizar para proveer *feedback*. De aquí surgieron ideas muy interesantes como por ejemplo la interacción mediante distintos canales de comunicación como Whatsapp, Slack y Microsoft Teams. Luego de varias discusiones e intercambio de opiniones surge llevar esta idea a un mayor nivel de complejidad. Esta consistía en la interacción con el usuario final mediante la utilización de llamadas telefónicas, donde el equipo concuerda que de esta forma se minimiza por completo el esfuerzo necesario por parte del usuario para proveer *feedback*. Mediante esta solución el usuario no debe redactarlo de forma escrita, sino que lo provee de forma oral, siendo una interesante alternativa al correo electrónico.

Al exponer esta idea al cliente, éste expresó un gran entusiasmo y aprobación respecto a la nueva funcionalidad, concordando en la adición de la misma a la solución.

Análisis más preciso del sentimiento del *feedback*

Luego de analizar ideas para mejorar la interacción con el usuario, se buscaron alternativas para mejorar el resultado del análisis de sentimiento del *feedback*. Hasta el momento se contaba únicamente con la integración del servicio de procesamiento de lenguaje natural de Google. En algunas instancias los resultados retornados por el servicio no contaban con la precisión que se esperaba (ver sección 7.1.2 Objetivos de calidad del producto), por lo que se buscaron alternativas con el fin de revertir esta situación.

Una primer alternativa que surgió fue la implementación de un modelo de inteligencia artificial capaz de aprender de forma autónoma mediante el suministro de grandes cantidades de muestras de datos. Ésta opción llamó la atención debido al gran desafío tecnológico involucrado. La razón por la cual no se pudo llevar a cabo fue la relación costo-beneficio, ya que no se contaba con experiencia previa en la implementación de modelos inteligentes, lo que demandaría mucho tiempo para interiorizar los conocimientos necesarios. El equipo consultó con el cliente y éste expresó que hubiese sido un gran desafío

pero que dada la realidad del proyecto no era viable. Se debía hacer foco en otros aspectos del proyecto como ser la interacción con el usuario y la calidad del producto a generar. Además el equipo era consciente que la implementación de esta idea no aseguraba la precisión del análisis.

Se continuaron buscando ideas que permitan mejorar el análisis del *feedback*. Aquí surge otra solución por parte del equipo, la cual consistió en la integración con otros servicios de procesamiento de lenguaje natural (sin descartar la existente) para luego generar un promedio entre los resultados y así disminuir la incertidumbre respecto al posible margen de error existente. Esta idea interesó al cliente ya que la relación costo-beneficio era sumamente positiva teniendo en cuenta los objetivos a corto y mediano plazo.

Integración con Office 365

Por otro lado, se ofreció aportar mayor valor a la solución abarcando una mayor cantidad de público objetivo respecto a la utilización de Google G-Suite como única plataforma de herramienta laboral. Luego de investigar acerca de los competidores de la plataforma utilizada, se concluye que Microsoft Office 365 era la plataforma indicada a integrarse luego de finalizada la integración con la inicial. El equipo se acercó al cliente respecto a esta idea, obteniendo una respuesta de total aprobación por parte de este, teniendo en cuenta que internamente Ascentis utiliza Microsoft Office 365 para su gestión laboral y junto con Google dominan el mercado de estos servicios.

3.4 Listado de requerimientos

A continuación se listan los requerimientos funcionales, no funcionales y restricciones que se relevaron durante las actividades descritas en el presente capítulo.

Requerimientos funcionales

Id	Requerimiento	Prioridad	Descripción
RF01	Sincronizar usuarios de Google GSuite.	Alta	Obtener todos los usuarios pertenecientes a un dominio de GSuite, para luego poder monitorear sus eventos.
RF02	Sincronizar eventos con Google Calendar	Alta	Obtener eventos de Google Calendar mediante un proceso periódico y configurable, reconociendo nuevos eventos y los recientemente finalizados.
RF03	Solicitar feedback mediante correo electrónico al finalizar un evento	Alta	El sistema deberá ser capaz de enviar un correo electrónico solicitando feedback a todos los usuarios del evento (pertenecientes a la organización) luego de que un evento finaliza. Este correo electrónico deberá poder ser customizable por el administrador del sistema
RF04	Recibir feedback por correo electrónico	Alta	El sistema deberá leer periódicamente una casilla de correo, recibiendo el feedback enviado por los usuarios para luego procesarlo.
RF05	Delegación de autoridad a cuenta de servicio de Gsuite.	Alta	Se deberá evitar la necesidad de solicitar consentimiento a cada usuario para acceder a sus datos laborales. Para esto el sistema deberá consumir los servicios de Google utilizando una cuenta de servicio.
RF06	Analizar el sentimiento de un feedback con Google - Natural Language API	Alta	Se deberá analizar el sentimiento de un *feedback* recibido utilizando el servicio de procesamiento de lenguaje natural de Google. Luego se deberá promediar el resultado con lo retornado por los demás servicios de PLN.
RF07	Notificar al TMS luego que el feedback es procesado	Alta	El sistema deberá enviar la información procesada al sistema TMS de Ascentis.
RF08	Generar reportes diarios	Alta	El sistema deberá generar reportes diarios de todos los *feedbacks* procesados en el día, enviándolo luego por correo electrónio a los asesores de talento
RF09	Generar reportes semanales	Media	El sistema deberá generar reportes semanales de todos los *feedbacks* procesados en la semana, enviándolo luego por correo electrónio a los asesores de talento.
RF10	Generar reportes mensuales	Media	El sistema deberá generar reportes mensuales de todos los *feedbacks* procesados en el mes, enviándolo luego por correo electrónio a los asesores de talento.

RF11	Feedback instantáneo	Media	La solicitud de *feedback* vía correo electrónico deberá contar con la opción de proporcionar *feedback* instantáneo mediante tres opciones: positivo, neutro y negativo. Una vez seleccionada una opción se deberá mostrar un mensaje de agradecimiento customizable por el administrador del sistema.
RF12	Solicitar feedback mediante llamada telefónica al finalizar un evento	Media	El sistema deberá ser capaz de realizar una llamada telefónica a un usuario para solicitarle *feedback*. El diálogo de la llamada telefónica deberá poder ser customizable por el administrador del sistema.
RF13	Recibir feedback mediante llamada telefónica	Media	El sistema deberá ser capaz de recibir *feedback* proveniente de llamadas telefónicas.
RF14	Análisis de sentimiento con IBM Watson - Natural Language Understanding	Media	Se deberá analizar el sentimiento de un *feedback* recibido utilizando el servicio de procesamiento de lenguaje natural de IBM Watson. Luego se deberá promediar el resultado con lo retornado por los demás servicios de PLN.
RF15	Análisis de sentimiento con Microsoft Azure - Cognitive Services	Media	Se deberá analizar el sentimiento de un *feedback* recibido utilizando el servicio de procesamiento de lenguaje natural de Microsoft Azure. Luego se deberá promediar el resultado con lo retornado por los demás servicios de PLN.
RF16	Sincronizar usuarios de Office 365	Baja	Obtener todos los usuarios pertenecientes a un dominio de Office 365, para luego poder monitorear sus eventos.
RF17	Sincronizar eventos de Office 365	Baja	Obtener eventos de Office 365 mediante un proceso periódico configurable, reconociendo nuevos eventos y los recientemente finalizados.
RF18	Solicitud de feedback recurrente sobre cierto tema	Baja	Solicitar feedback periódicamente sobre cierto tópico en particular. Se deberá poder admitir múltiples tópicos.
RF19	Limitar solicitudes a ciertas áreas de la organización	Baja	Limitar los usuarios a los que se le solicita feedback en base a la unidad organizacional a la cual pertenecen.

Tabla 3.1 Requerimientos Funcionales

Requerimientos no funcionales

Id	Requerimiento	Prioridad	Descripción	Atributo de calidad
RNF01	Habilidad de ejecutar sobre cualquier sistema operativo.	Alta	El sistema deberá ser capaz de ejecutarse sobre cualquier ambiente posible para adaptarse a la infraestructura del cliente, sin importar el sistema operativo o hardware involucrado.	Portabilidad
RNF02	Facilidad de integración con nuevos servicios de terceros	Alta	El sistema deberá contener una arquitectura y diseño que permita la fácil adición de nuevas comunicaciones con servicios de terceros. En futuras etapas del proyecto, es deseable que el sistema pueda integrarse con nuevos canales de comunicación para obtener *feedback* desde distintas plataformas.	Extensibilidad
RNF03	Facilidad para cambiar los pasos de procesamiento del *feedback*.	Alta	El esfuerzo necesario para cambiar los pasos del procesamiento del *feedback* deberá ser bajo.	Modificabilidad Extensiblidad
RNF04	Esfuerzo casi nulo por parte del usuario para proveer *feedback*	Alta	Considerando que proveer *feedback* es una acción no obligatoria, el esfuerzo necesario para ello deberá ser mínimo, buscando motivar al usuario final a proveerlo sin demandar demasiado tiempo.	Usabilidad
RNF05	Velocidad de respuesta	Alta	Cualquiera sea el canal de comunicación con el usuario, la notificación de recepción exitosa o errónea del *feedback* provisto deberá ser instantánea, con el fin de no hacer esperar al usuario por cualquier procesamiento interno.	Eficiencia
RNF06	Adaptable a organizaciones de distintos tamaños.	Alta	El sistema deberá manejar una gran cantidad de usuarios, debido a que el mismo será utilizado por empresas estadounidenses de pequeño y mediano porte.	Escalabilidad
RNF07	Manejo de errores efectivo	Media	Se deberá tener un manejo detallado y organizado de los errores del sistema, siendo capaz de alertar al usuario ante la ocurrencia y motivo de cualquier error que lo afecte	Disponibilidad

Tabla 3.2 Requerimientos no funcionales

Restricciones

- Toda información que sea entregada a Ascentis debe estar redactada en inglés. Esto incluye tanto los documentos como el código fuente.

- No implementar interfaz de usuario. Aprovechar la interfaz gráfica que proveen Google Calendar y los clientes de correo electrónico.

3.5 Conclusiones y lecciones aprendidas

Se concluye que las actividades realizadas por el equipo si bien fueron muy dispares, fueron también muy útiles y complementarias para el proceso de ingeniería de requerimientos, ya que se lograron entender y validar correctamente tanto los requerimientos funcionales como los no funcionales.

Amerita recalcar la gran utilidad que proporcionó la prueba de concepto, permitiendo al equipo validar los requerimientos relevados, investigar las tecnologías a utilizar, verificar la viabilidad del producto, identificar posibles riesgos y construir un prototipo funcional evolutivo, todo en la misma actividad.

Fue una enorme satisfacción ver cómo el cliente se involucraba cada vez más en el proyecto debido a las ideas innovadoras presentadas por el equipo, viéndose esto reflejado en las instancias de medición de satisfacción del cliente (ver sección 7.3.7 Satisfacción del cliente). En particular la funcionalidad de solicitud de *feedback* mediante llamada telefónica logró atraer notablemente la atención del cliente. Esto se debió a la correcta priorización de la necesidad del cliente asociada a que la solicitud de *feedback* debía ser lo más sencilla y libre de esfuerzo posible.

4 Descripción de la solución de software

4.1 Introducción

En este capítulo se hace foco en la presentación de la solución tecnológica implementada. Se exponen las distintas herramientas, tácticas y conocimientos de arquitectura de software aplicados para generar una solución de calidad que supere las expectativas del cliente.

En primer lugar se presentan los enfoques arquitectónicos que se manejaron para la solución, describiendo ventajas y desventajas de cada uno teniendo en cuenta las necesidades de calidad identificadas. Luego se detalla la decisión que el equipo tomó para la elección de la arquitectura, justificando los motivos por los que se alinea más a los requerimientos relevados.

Luego de detallar las decisiones, se presenta la descripción de la arquitectura para dar conocimiento de las distintas vistas y en cada una de ellas los patrones de diseño y arquitectónicos utilizados.

Posteriormente se presentan para cada atributo de calidad a favorecer, las tácticas aplicadas que justifican el favorecimiento de los mismos.

Se presenta también la instancia de validación de la arquitectura final con el cliente, mencionando su opinión, comentarios y sugerencias luego de la exposición por parte del equipo.

Por último se exponen las conclusiones y lecciones aprendidas relacionadas a la arquitectura resultante.

4.2 Monolito vs. Microservicios

En las primeras fases del proyecto, luego de finalizar con el proceso de ingeniería de requerimientos, no se contaba con un enfoque arquitectónico definido. No se tenía claro si era conveniente contener todas las funcionalidades en una única aplicación desplegable, o distribuir las funcionalidades en un ecosistema de aplicaciones.

Para cada atributo de calidad asociado a los distintos requerimientos no funcionales se realizó un balance entre ambas arquitecturas para evaluar cuál se adecuaba de mejor manera a la realidad del proyecto.

- **Portabilidad**

La portabilidad, es decir, la característica que posee el o los elementos de software para ejecutarse en diferentes plataformas, es favorecido de igual forma ante cualquiera de las dos arquitecturas.

Igualmente, vale aclarar que la arquitectura monolítica presenta una gran desventaja sobre la basada en microservicios. Al tratarse de una única aplicación, la arquitectura monolítica se acopla a una única tecnología, mientras que la arquitectura basada en múltiples servicios permite manejar un *stack* de múltiples tecnologías e implementar cada uno en el lenguaje que sea más apropiado para el propósito deseado.

- **Extensibilidad**

Lo que se busca al favorecer este atributo de calidad es requerir el mínimo esfuerzo posible para agregar nuevos elementos de software sin efectos colaterales indeseados.

Para favorecer la extensibilidad la arquitectura debe incentivar la modularidad, es decir, manejar unidades lógicas lo suficientemente granulares. Este atributo impone escasas y limpias dependencias durante el desarrollo, así como también estructuras altamente cohesivas y levemente acopladas. Además, se debe contar con una buena legibilidad del código fuente, con esto se busca ubicar la nueva funcionalidad en el sector adecuado dentro de la arquitectura.

La arquitectura monolítica no tiende a favorecer la extensibilidad. Al ubicar todos los elementos de software en un mismo componente, suele existir una alta dependencia entre distintos elementos de la arquitectura. Además, al ubicarse todos los módulos dentro del mismo proyecto, se pierde legibilidad, lo que puede ocasionar la incorrecta ubicación de nuevos elementos dentro de la solución, produciendo por consiguiente una complejidad adicional en modificaciones futuras.

En contraparte, un ecosistema de servicios donde cada uno tiene una única responsabilidad provee una gran ventaja asociada a la extensibilidad ya que cada servicio cuenta con una cantidad baja de módulos. Se contará con un bajo acoplamiento y será sumamente intuitivo ubicar nuevos elementos dentro del servicio sin necesidad de eventualmente afectar a los existentes.

Por tanto, podemos concluir que la extensibilidad se ve más favorecida con la arquitectura basada en microservicios respecto a la monolítica.

- **Modificabilidad**

La modificabilidad se relaciona con el costo del cambio y la facilidad con que el sistema se adapta a los cambios. El mismo se relaciona de gran manera con la extensibilidad. Para favorecerlo se debe contar con elementos de software altamente cohesivos y levemente acoplados. Debe existir una única razón por la cual se desea modificar un elemento y las dependencias entre los mismos debe ser escasa.

Si se optara por la arquitectura monolítica, aunque el código fuente se encontrase perfectamente estructurado, igualmente se tendería a contar con un alto acoplamiento debido a la cantidad de dependencias entre los módulos del sistema. Este enfoque dificulta el desarrollo en equipo ya que todos los desarrolladores trabajan sobre el mismo proyecto. Además, a medida que el proyecto crece con el tiempo, se vuelve cada vez más tedioso realizar un cambio o agregar nueva funcionalidad al sistema. Al contar con tantos módulos en el mismo proyecto también se vuelve costoso entender el propósito del sistema para una persona que no participó de la construcción desde las primeras fases del proyecto (el cliente continuará trabajando sobre el producto en el futuro).

Por otro lado, si se construye una arquitectura basada en microservicios, cada uno de ellos contará con una baja cantidad de módulos, y por ende, si se genera una correcta estructura modular, se consigue un bajo acoplamiento. Introducir un cambio suele ser una tarea sencilla en comparación con una arquitectura monolítica. Si se diseñan los servicios en base a su responsabilidad, cada componente será granular y al efectuar un cambio sobre el mismo será menos probable que provoque efectos negativos sobre los demás servicios.

El equipo considera que este es uno de los atributos de calidad más favorecido por la arquitectura basada en microservicios.

- **Disponibilidad**

El mismo busca que el sistema se encuentre la mayor cantidad de tiempo posible en condición de funcionamiento operativo.

He aquí otro gran favorecimiento por parte de la arquitectura en microservicios por sobre la monolítica. Si se cuenta con un único artefacto de software, y el mismo sufre una falla que ocasiona la detención de su ejecución, la totalidad de las funcionalidades del sistema dejarán de estar operativas. En cambio, si se contara con múltiples servicios y uno de ellos sufriera una falla que lleve a detener su ejecución, los otros servicios seguirán operativos provocando solamente una pérdida parcial de funcionalidades operativas.

En conclusión, el ecosistema de servicios aporta mayor valor a la disponibilidad en comparación al monolito.

- **Usabilidad**

Ninguno de los enfoques involucrados en la discusión influyen directamente sobre los mecanismos que faciliten el uso del sistema por parte del usuario. Debido a que la interacción del usuario se da mediante correo electrónico o llamada telefónica, construir una única aplicación o múltiples servicios no influirá en la mencionada interacción.

Igualmente vale aclarar que los atributos de calidad influyen entre ellos. En el contexto actual existen más de un canal de comunicación con el usuario, por lo que la disponibilidad puede ser un atributo influyente sobre la usabilidad. Si el usuario no fuera capaz de proveer feedback debido a la indisponibilidad total del sistema (considerar lo comentado en Disponibilidad), no existiría ningún mecanismo de interacción, impactando esto también en la confiabilidad.

Teniendo esto presente, podemos concluir que una arquitectura basada en microservicios aportaría mayor favorecimiento a la usabilidad de Feedback Assistant en comparación con una única aplicación monolítica.

- **Escalabilidad**

Ascentis desea instalar Feedback Assistant en los distintos ambientes de sus clientes. Por esto, es requerido que el mismo sea capaz de escalar para soportar un gran número de usuarios concurrentes interactuando con el sistema (de 100 a 500 usuarios). De aquí surge la necesidad de escalar el producto con facilidad.

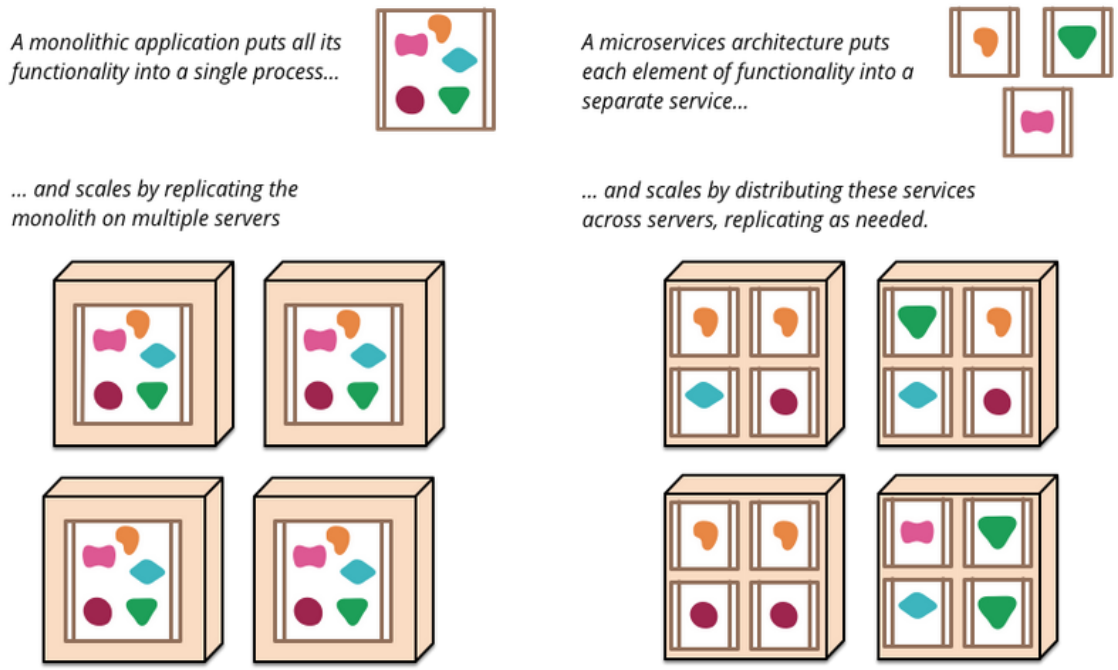


Ilustración 4.1. Escalabilidad microservicios vs monolítico [14]

En cuanto a la escalabilidad horizontal, tal como se presenta en la ilustración, vemos como la arquitectura basada en microservicios favorece de mayor manera este atributo de calidad en comparación con la monolítica, proporcionando la posibilidad de replicar solamente los nodos referentes a los servicios más críticos ante alta concurrencia. Dicho esto, llevando a cabo una escalabilidad horizontal en un ecosistema de aplicaciones se aprovechan mejor los recursos.

Por otro lado, utilizando una arquitectura monolítica, el artefacto desplegable contiene todos los componentes del sistema siendo imposible escalar solamente los sectores críticos. Por ende, al escalar horizontalmente se estará utilizando capacidad de carga innecesaria para determinadas funcionalidades del sistema.

Para el caso de escalabilidad vertical, se puede afirmar que la arquitectura monolítica es más favorable por sobre la basada en microservicios. El hecho de tener múltiples aplicaciones interactuando ya sea por HTTP o mecanismos de mensajería implica una carga adicional en los recursos de la infraestructura donde se ejecuta la aplicación. Por tanto, es necesario mantener una infraestructura de mayor porte para los microservicios en comparación con el monolito.

Poniendo ambos argumentos en consideración, el equipo decide inclinarse por una escalabilidad horizontal, y por ende un ecosistema basado en múltiples servicios es el camino para el favorecimiento de este atributo de calidad.

- **Eficiencia**

Una exigencia que propone la realidad del proyecto es la necesidad de respuesta inmediata al usuario luego de proveer *feedback* al sistema. Independientemente del canal de comunicación, se requiere evitar la espera por parte del usuario en recibir la notificación de recepción exitosa o errónea del *feedback*. Específicamente para la comunicación por llamada telefónica, la respuesta dando a conocer la recepción exitosa del feedback provisto debe ser instantánea.

A primera vista, se puede afirmar que la arquitectura monolítica favorece más la eficiencia que la basada en microservicios. En un ecosistema de aplicaciones, los mismos suelen comunicarse mediante protocolos web, por lo que existiría una latencia entre las comunicaciones afectando notoriamente este atributo de

calidad. Por otro lado, si se maneja una única aplicación donde la comunicación se da a nivel de dependencias modulares, este problema de latencia es evitado generando menores tiempos de respuesta.

De todas formas, este problema que presentan los microservicios en cuanto a eficiencia puede ser evitado. Un mecanismo posible es incluir asincronismo en la comunicación entre los servicios. De esta forma se puede evitar esperar por el procesamiento de la transacción para notificar la recepción, llevando a cabo primero una respuesta instantánea al usuario previo al procesamiento. Teniendo en cuenta que no es requerido notificar el estado del procesamiento, sino la correcta o en su defecto errónea recepción del *feedback*, agregando por ejemplo una cola de mensajes entre el productor del *feedback* (servicio que recibe el *feedback* luego de ser enviado por el usuario) y el consumidor del mismo (servicio con la responsabilidad de procesar el *feedback*) se puede solucionar este problema que presentan los microservicios.

En conclusión, la arquitectura que más favorece este atributo de calidad es la monolítica. Igualmente, si se le agrega un poco de complejidad a la arquitectura basada en microservicios será posible favorecer este atributo de calidad.

- **Otros aspectos a considerar**

Independientemente de los requerimientos no funcionales y sus atributos de calidad asociados, en el contexto del equipo existen otros aspectos que pueden influir en el éxito o no del proyecto y ambos enfoques arquitectónicos se inclinan a apoyar distintos aspectos.

Uno de ellos es la necesidad de Ascentis en obtener entregables incrementales en tiempos cortos. Ésto se ve más favorecido por la arquitectura monolítica por sobre la basada en microservicios. El hecho de tener un único repositorio, no utilizar protocolos de comunicación para los procesos internos (sino dependencias modulares) y manejar el despliegue de un único artefacto favorecen la productividad permitiendo entregables con mayor funcionalidad en las primeras fases del proyecto.

Otro aspecto que favorece de mayor forma la aplicación monolítica es la integridad de los datos. Si contamos con un ecosistema de aplicaciones, los repositorios de datos serán también distribuidos, y de esta forma podrían generarse problemas de integridad entre los datos debido al manejo de claves foráneas de otro esquema independiente.

Conclusión

Luego de analizar todas las consideraciones previamente detalladas, el equipo toma la decisión de aplicar una arquitectura basada en microservicios.

Claramente, los requerimientos no funcionales apuntan a atributos de calidad altamente favorecidos por este enfoque arquitectónico. A pesar de las desventajas y complejidades que el mismo presenta, el equipo concluye que los resultados finales en cuanto a calidad serán mayores respecto a la arquitectura monolítica, presentando además un desafío técnico en la implementación de estos mecanismos de arquitectura que motiva al equipo a llevarla a cabo.

4.3 Descripción de la arquitectura

En esta sección se lleva a cabo una apreciación detallada de la arquitectura del sistema desde distintos puntos de vista. El objetivo es lograr expresar las decisiones tomadas en los distintos sectores del sistema junto con los distintos patrones de diseño y arquitectónicos aplicados en cada uno de ellos.

A continuación se presenta una ilustración de la arquitectura a alto nivel para dar una idea general al lector de cómo se compone la solución arquitectónica y luego detallar las distintas estructuras que lo componen.

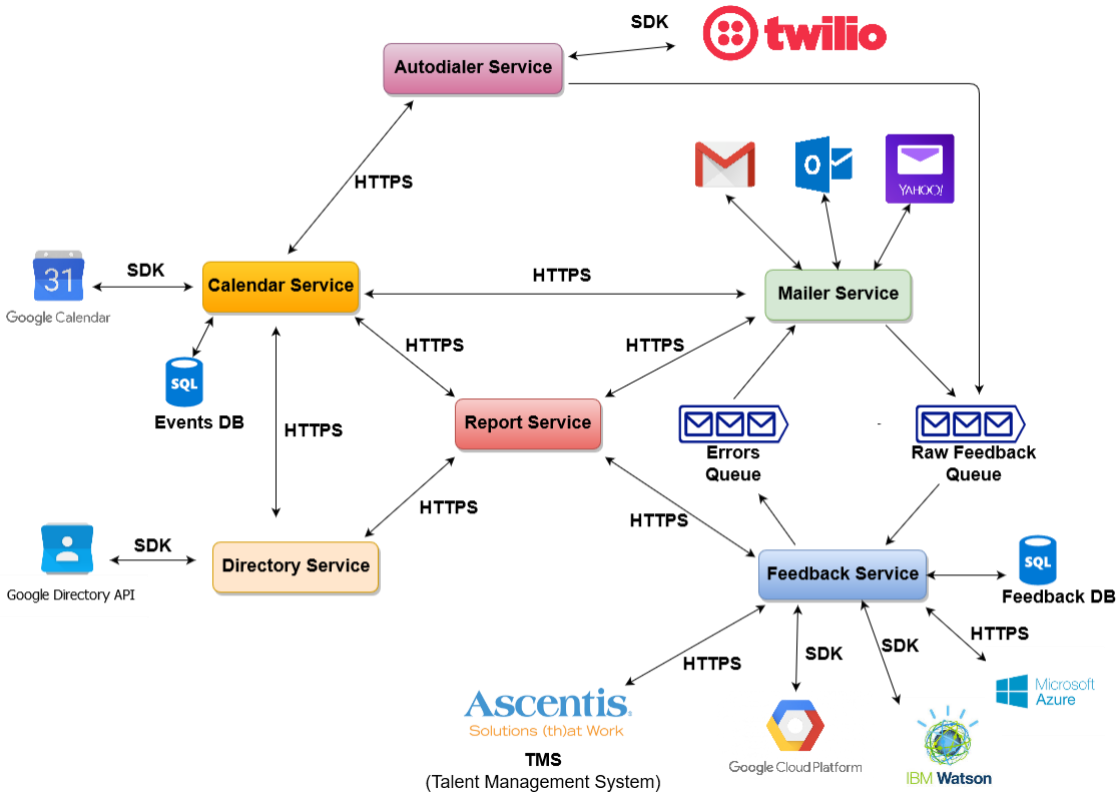


Ilustración 4.2. Solución arquitectónica

A lo largo de esta sección se expone información con ayuda de distintos diagramas cuando la situación lo amerite, con el fin de detallar la solución desde un aspecto lógico hasta uno físico del sistema en producción.

Para lograr este cometido, el equipo toma la decisión de documentar la arquitectura basándose en el enfoque presentado en 'Documenting Software Architectures: Views & Beyonds' [15]. El mismo provee tácticas de documentación donde se analiza la estructura arquitectónica desde distintos enfoques. Vale aclarar que una vista representa un conjunto de elementos del sistema y las relaciones asociadas a ellos.

Se comienza mostrando las vistas de módulos para dar a conocer la organización lógica del sistema. Posteriormente se detallan las vistas de componentes y conectores con el objetivo de mostrar las estructuras del sistema y la relación entre las mismas en tiempo de ejecución. Por último, se expone la vista de alocación para documentar la distribución física de la solución en el ambiente de producción, aquí se estarán dando a conocer los artefactos involucrados y la infraestructura donde los mismos se alojan.

La información que se encontrará en cada una de las distintas vistas será:

- **Representación primaria:** Expone diagramas que muestran las estructuras que se desean detallar en la vista.

- **Catálogo de elementos:** Describe las responsabilidades de cada elemento que compone la representación primaria.

- **Justificación:** Explica las decisiones de diseño mencionando los patrones utilizados y su debida justificación.

4.3.1 Vista de módulos

A continuación se detallan las vistas referentes a las estructuras lógicas de Feedback Assistant. Se da a conocer tanto la descomposición de los paquetes así como también las dependencias entre los mismos.

Vista de Descomposición

Representación Primaria

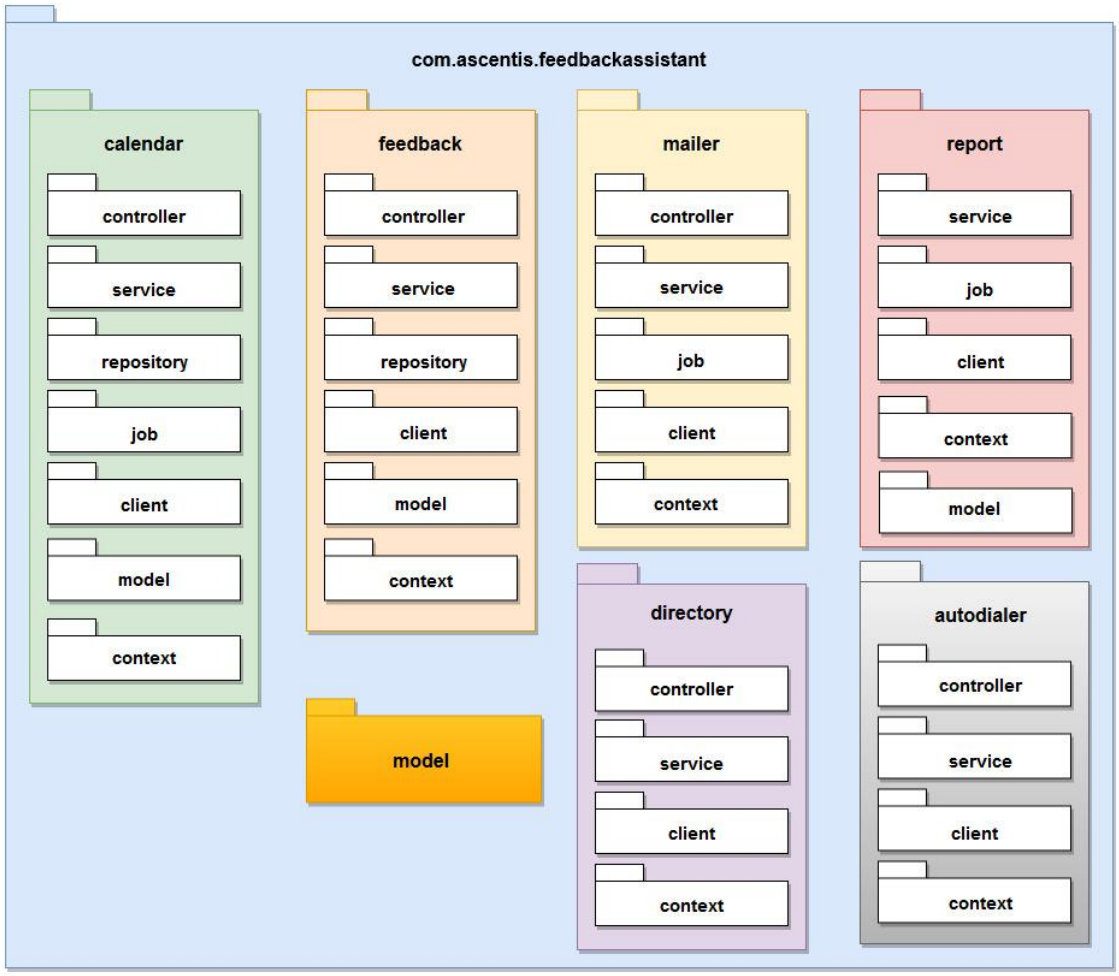


Ilustración 4.3 Diagrama de descomposición

Catálogo de Elementos

En esta sección se describen los elementos lógicos o paquetes pertenecientes al sistema y sus responsabilidades con el objetivo de dejar en claro la razón de existencia de cada uno de ellos.

Elemento	Responsabilidades
calendar	➢ Responsable de contener las estructuras que gestionan los eventos laborales extraídos de los calendarios y llevar a cabo la persistencia de los mismos en el respectivo repositorio de datos. ➢ Cada cierto período configurable de tiempo sincroniza los eventos de los usuarios y dispara solicitudes de *feedback* ante la aparición de nuevos eventos finalizados.
feedback	➢ Se encuentran ubicados los elementos encargados de realizar el procesamiento del *feedback*. A medida que se reciben nuevos *feedbacks*, éste se comunica con los distintos servicios de procesamiento de lenguaje natural, para luego generar un promedio de los resultados obtenidos. ➢ Envía la información obtenida al sistema TMS. ➢ Persiste el procesamiento generado.
mailer	➢ Sirve como vía de comunicación entre el sistema y el usuario vía correo electrónico. Solicita y recepciona *feedbacks*, envía reportes y notifica ante fallas internas en el procesamiento. ➢ Una vez recibido el *feedback*, el mismo es enviado de forma asincrónica al servicio encargado del procesamiento. ➢ Responsable de manejar las plantillas customizables de solicitud de *feedback* y notificación de procesamiento erróneo ubicadas en los recursos.
report	➢ Los elementos encargados de generar los distintos reportes periódicos se ubican bajo este módulo, generando las solicitudes pertinentes a los demás servicios con el fin de recabar la información necesaria para generar los reportes.
model	➢ Contiene las entidades de negocio asociadas al modelo de datos de la solución.
directory	➢ Contiene las estructuras que cumplen con la responsabilidad de gestionar los usuarios del sistema, realizando una sincronización de los mismos mediante la comunicación con el servicio externo de contactos (Google Directory).
autodialer	➢ Se contienen los elementos encargados de servir como vía de comunicación entre el sistema y los usuarios, utilizando llamadas telefónicas. Estas estructuras reciben la orden de

	solicitar *feedback* y realizan un llamado telefónico al usuario correspondiente. ➢ Luego de interactuar con el usuario y obtener el feedback del mismo, se envía esta información de forma asincrónica al servicio encargado del procesamiento.
controller	➢ Estructuras encargadas de manejar los recursos HTTP, donde se hará posible la recepción de solicitudes para la comunicación entre los distintos servicios del ecosistema.
service	➢ Contiene los elementos que permiten encapsular la lógica de negocio.
repository	➢ Agrupa las estructuras encargadas de manejar la interacción con el repositorio de datos.
job	➢ Contiene los elementos que ejecutan tareas bajo tiempos periódicos configurables.
client	➢ Contiene las implementaciones de los clientes HTTP para lograr la comunicación entre los distintos servicios del ecosistema o con servicios externos.
context	➢ Agrupa la configuración que utiliza *Spring framework* para dar inicio a la correcta ejecución del servicio y cargar los *beans* asociados al mismo.

Tabla 4.1. Catálogo de elementos del sistema

NOTA: Este catálogo de elementos debe ser tenido en cuenta para la vista de uso

Justificación

Como se puede observar en la representación primaria, se genera una descomposición modular por funcionalidad. Esto se alinea con la filosofía del patrón arquitectónico de microservicios, lo que brinda la gran ventaja de que cada módulo cumpla con un conjunto de responsabilidades concreto, aumentando la coherencia semántica de los mismos.

Dentro del módulo asociado a cada microservicio, se puede apreciar una separación lógica aplicando un patrón arquitectónico en capas. Se agrupan los elementos con misma responsabilidad con el fin de estandarizar la modularización a lo largo de los servicios. El equipo se cuestionó si era conveniente utilizar este patrón o no, pero dada la cantidad de aplicaciones existentes, se consideró útil generar agrupaciones lógicas que sean idénticas a lo

largo del ecosistema. Además, se logra mediante este enfoque un desarrollo en varios niveles, produciendo dos grandes ventajas:

- En caso de que sobrevenga alguna modificación al microservicio, éste afectará solo al nivel o capa involucrada, dado que se redujo el acoplamiento utilizando abstracciones para la comunicación entre capas, lo que claramente favorece la modificabilidad. A modo de ejemplo, si en un futuro se desea cambiar toda la implementación de la interacción con la base de datos, simplemente se debe modificar lo existente en la capa de repositorio sin necesidad de modificar nada existente en los otros niveles de la aplicación.

- Brinda la gran ventaja de poder distribuir el trabajo de manera eficiente entre desarrolladores que trabajan sobre el mismo microservicio. Al contar con capas bien definidas dentro de la aplicación, cada desarrollador se encuentra totalmente desacoplado del resto de las capas, de modo que basta con conocer la abstracción existente entre dos capas para poder desarrollar en paralelo.

En cuanto a lo que concierne a las entidades de negocio, en la ilustración 4.3 Diagrama de descomposición se puede ver cómo aparece el módulo *model* múltiples veces. Al contar con varios microservicios independientes, se torna necesario un buen manejo de entidades de negocio a través del ecosistema. Una opción era alojar un módulo de entidades de negocio en cada microservicio, de esta forma, dados dos microservicios que compartan una entidad de negocio, la clase correspondiente a la misma se encontraría duplicada en ambos microservicios. El equipo considera esta opción como desfavorable debido a que se estaría comprometiendo totalmente a la modificabilidad causado por la duplicación y mantenimiento de código fuente en múltiples ubicaciones. La otra opción que se manejó, y que se tomó la decisión de llevar a cabo, fue la de generar un módulo conteniendo las entidades de negocio que se ubique en un nivel de especificidad idéntico al de los microservicios. De esta forma, se logra encapsular todas las entidades de negocio en un espacio genérico para que luego los microservicios dependan de ellas cuando sea necesario, favoreciendo el reúso y evitando mantener código duplicado a lo largo de los microservicios.

Igualmente, se puede observar en la ilustración 4.3 Diagrama de descomposición cómo algunos microservicios contienen un módulo *model* en su interior, independiente del genérico previamente justificado. Los microservicios que requieren interacción con un repositorio de datos utilizan la implementación de JPA provista por la herramienta Hibernate. Además, éste mecanismo exige que toda entidad a persistir cuente con una serie de anotaciones en su definición para que el mismo sea capaz de definir y manejar automáticamente el esquema de datos. Pero como varias de las entidades que se persisten son también compartidas a lo largo de varios microservicios, el equipo decide alojar una subclase (extiende de la entidad de negocio genérica) en el microservicio donde se desea persistir a la misma, colocando en ésta las anotaciones necesarias para que el *framework* gestione la misma en el repositorio de datos interno del microservicio. Para obtener mayor información acerca de esta estrategia utilizada ver la sección Manejo de Entidades contenida en el capítulo 5.2 Principales desafíos tecnológicos.

Vista de Uso

Esta vista describe las relaciones entre las estructuras lógicas previamente presentadas en la vista de descomposición.

A continuación se muestran los diagramas y justificaciones referentes a la vista de uso, donde para cada microservicio del sistema se detallan las interrelaciones modulares.

Representación primaria

Microservicio Calendar

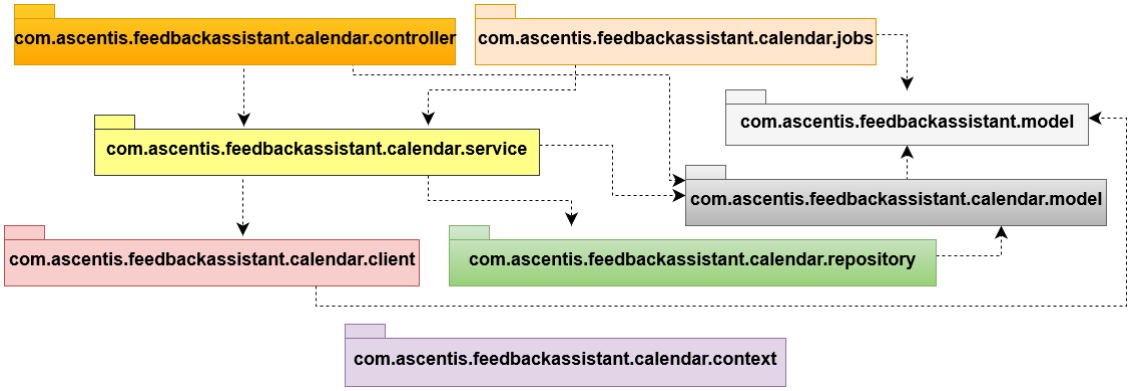


Ilustración 4.4 Vista de uso microservicio Calendar

Microservicio Feedback

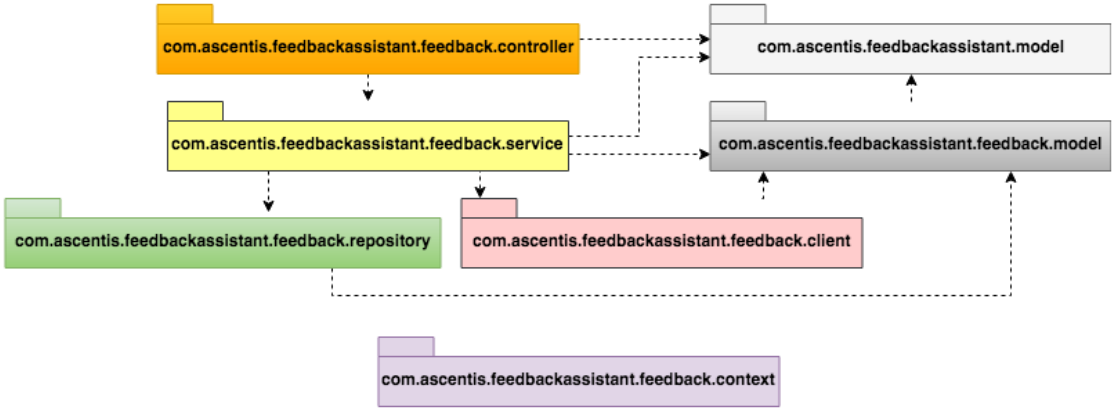


Ilustración 4.5 Vista de uso microservicio Feedback

Microservicio Report

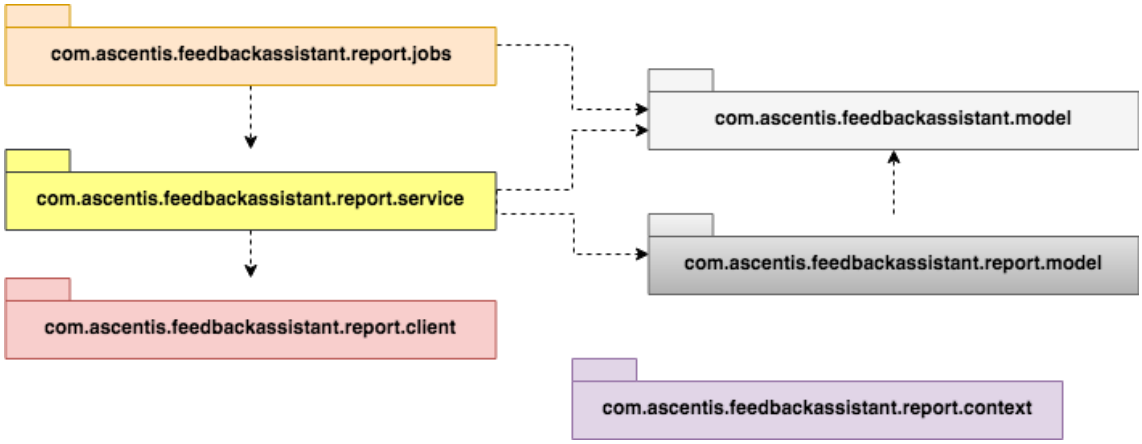


Ilustración 4.6 Vista de uso microservicio Report

Microservicio Directory

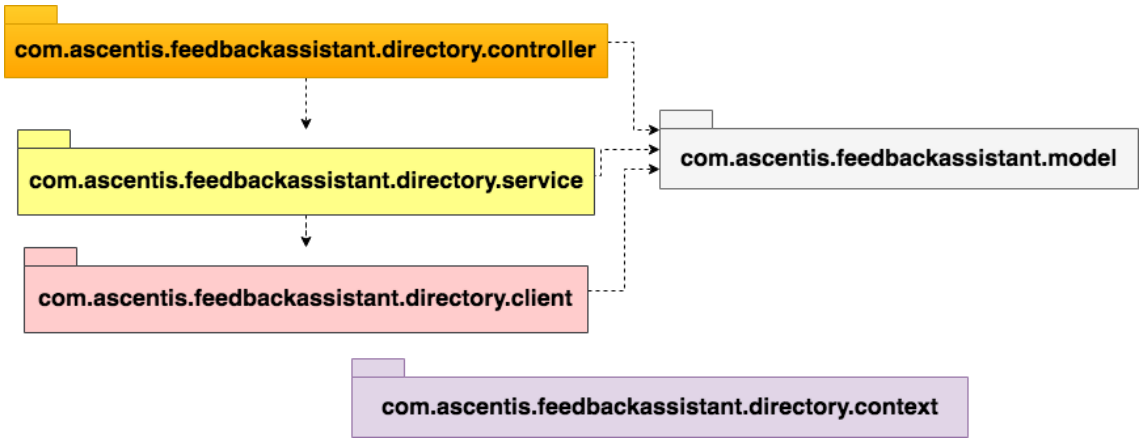


Ilustración 4.7 Vista de uso microservicio Directory

NOTA: Considerar esta ilustración para el microservicio *Autodialer*.

Microservicio Mailer

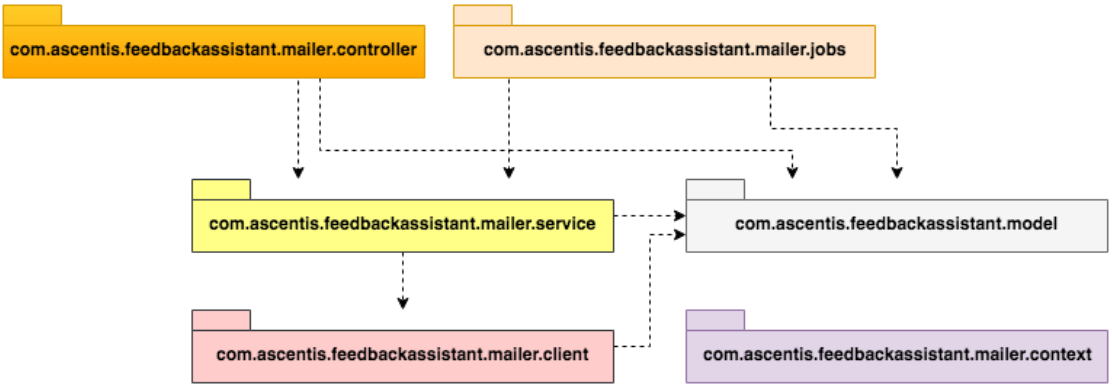


Ilustración 4.8 Vista de uso microservicio Mailer

Catálogo de Elementos

Idéntico al de vista de descomposición.

Justificación

Las dependencias entre módulos reflejan la aplicación del patrón en capas. El flujo de comunicación entre las capas tiene siempre un único sentido, y además, una decisión importante que el equipo tomó fue la de no generar relaciones entre capas pasando por encima de otras existentes. Un claro ejemplo de esta situación es cuando se inyecta una abstracción expuesta por la capa de repositorio en la capa de controlador, sin hacer uso de la capa de servicio existente entre las mismas. El equipo entiende que a pesar de ser válido para el patrón aplicado, lo más conveniente es no realizar saltos por encima de las capas, en busca de reducir el acoplamiento entre las mismas y así maximizar el favorecimiento de la modificabilidad.

Como se puede observar en las distintas ilustraciones referentes a las interdependencias modulares, siempre se cumple con el principio de módulos donde nunca ocurre que un módulo dependa de otro más específico teniendo en cuenta la profundidad del nombre de ambos paquetes.

Considerando la necesidad de favorecer la mantenibilidad, el equipo priorizó tener especial cuidado a la hora de decidir qué clases van en cada módulo, incidiendo directamente en la cohesión modular. Por esto, se diseñó la solución aplicando el Patrón GRASP [16] de alta cohesión a nivel de clase para cumplir con el Principio de Clausura Común a nivel de módulo. Esto implica que las clases que cambian juntas, se agrupan juntas. Asignar una responsabilidad de modo que la cohesión continúa siendo alta es fundamental. Una clase con baja cohesión provoca problemas de modificabilidad, reutilización y comprensión de su existencia. La aplicación del patrón en capas ayuda también a cumplir con este principio, debido a que se encapsulan las estructuras que tienden a cambiar juntas. Retomando el ejemplo referente a la necesidad de cambiar por completo la interacción con el repositorio de datos, se debería modificar únicamente el módulo *repository*. Aquí vemos como las clases que cambian juntas son agrupadas dentro de la misma estructura.

El equipo sacó provecho de la funcionalidad de inyección de dependencias provista por *Spring Framework,* logrando de esta forma con el cumplimiento del principio SOLID [17] referente a la inversión de dependencias, generando acoplamientos hacia abstracciones.

```
public void process(RawFeedback rawFeedback) {
    LOG.info("Processing feedback for event: " + rawFeedback.getEventId() + ", from: "
            + rawFeedback.getfromEmail());

    for (int i = 0; i < steps.size(); i++) {
        ProcessorStep step = steps.get(i);
        try {
            step.process(rawFeedback);
            LOG.info(String.format("[%1$d/%2$d] Step %3$s: OK", i + 1, steps.size(), step.getClass().getName()));
        } catch (FeedbackException fe) {
            LOG.error(String.format("[%1$d/%2$d] Step %3$s: ERROR. Aborting", i + 1, steps.size(),
                    step.getClass().getName()));
            errorFeedbackMQSenderService.pushErrorFeedbackToMQ(fe);
            break;
        }
    }
}

@PostConstruct
public void loadSteps() {
    steps = new ArrayList<>();

    ClassLoader loader = RawFeedbackMQReceiverService.class.getClassLoader();
    int stepCount = Integer.parseInt(environment.getProperty("feedbackassistant.processor.step_count"));

    for (int i = 1; i <= stepCount; i++) {
        String className = environment.getProperty("feedbackassistant.processor.step_" + i);
        try {
            Class<?> stepClass = loader.loadClass(className);
            ProcessorStep step = (ProcessorStep) stepClass.getDeclaredConstructor().newInstance();
            AutowireCapableBeanFactory bf = this.context.getAutowireCapableBeanFactory();
            bf.autowireBean(step);

            steps.add(step);
        } catch (ReflectiveOperationException ex) {
            LOG.error("Could not load step " + i + ", class '" + className + "': " + ex.getMessage(), ex);
        }
    }
}
```

Ilustración 4.9 Implementación canal de procesamiento

El equipo implementó el procesamiento del *feedback* en una serie de pasos donde dado un *feedback*, éstos realizan modificaciones sobre el mismo. Teniendo en cuenta que se busca favorecer la extensibilidad y modificabilidad, el equipo aplica el patrón de diseño *Chain of Responsability* [18]. Mediante un único canal de procesamiento y múltiples manejadores, éstos últimos son intercambiados dinámicamente en tiempo de ejecución, recibiendo el *feedback* a procesar, realizando algún cambio sobre el mismo, y retornándolo nuevamente hacia el canal de procesamiento. En cuanto a la implementación, utilizando *Reflection* se cargan los *beans* referentes a cada paso del procesamiento [19]. Para esto, se define en el archivo de configuración el nombre del *bean* asociado a cada paso. Cada paso implementa una abstracción, y luego el manejador utiliza esta abstracción con el fin de evitar conocer cada implementación. Luego de obtenidos los pasos, se itera sobre los mismos ejecutando el método `process()` definido en la abstracción, y de esta forma actuar como canal de procesamiento. El equipo es consciente de la desventaja que genera este mecanismo, siendo esta el uso de *Reflection* con el fin de manejar dinámicamente el uso de *beans* por su nombre.

Igualmente se toma la decisión de asumir este riesgo y mitigarlo realizando una adecuada detección y manejo de excepciones.

Se tomó la decisión de encapsular el análisis del sentimiento utilizando los distintos servicios de PLN en un único paso del procesamiento. Se utiliza un servicio que aplica el patrón *Strategy* para manejar una familia de algoritmos encargados de comunicarse con los distintos servicios de PLN e intercambiarlos con facilidad [20].

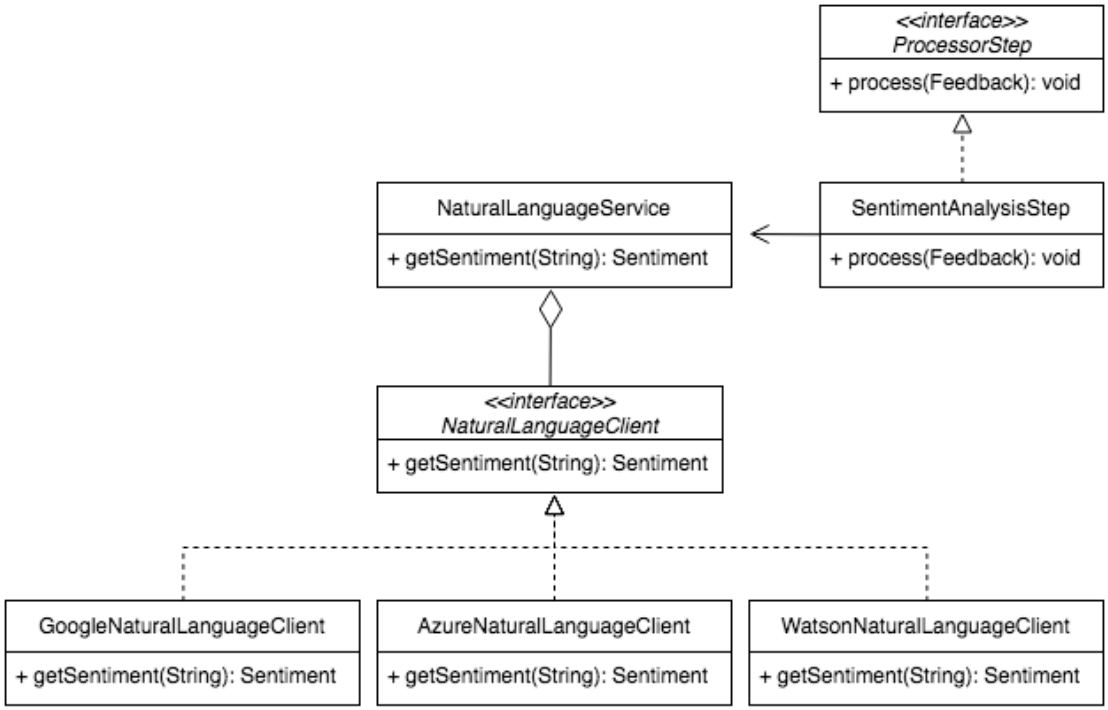


Ilustración 4.10 UML Procesamiento de sentimiento

Otra decisión importante que se tomó fue la de no generar un módulo específico para las excepciones. Crear un módulo donde alojar las excepciones es una tarea muy común en el desarrollo de software, pero el equipo tomó la decisión de no generar este módulo y ubicar las mismas donde sea necesario. Se cuestiona qué responsabilidad tendría este módulo conviviendo con los demás dentro del sistema. Luego de varias discusiones se llega a la conclusión de que no tiene una responsabilidad que aporte a la funcionalidad del ecosistema de módulos; se estaría creando un módulo simplemente para generar cierto orden en la

arquitectura, provocando un módulo levemente cohesivo. No se estaría cumpliendo con el principio de clausura común ya que las excepciones no cambian juntas, por lo que no deben agruparse juntas.

Por tanto, se considera importante dejar de lado la costumbre o la necesidad de generar cierto orden en la arquitectura y hacer foco en el acoplamiento. Si se crea un módulo de excepciones se estarán generando tantas dependencias como tantos módulos necesiten conocerlo, produciendo un amplio desfavorecimiento en la modificabilidad causado por el alto acoplamiento.

4.3.2Vista de componentes y conectores

Vista de ecosistema de microservicios

Representación primaria

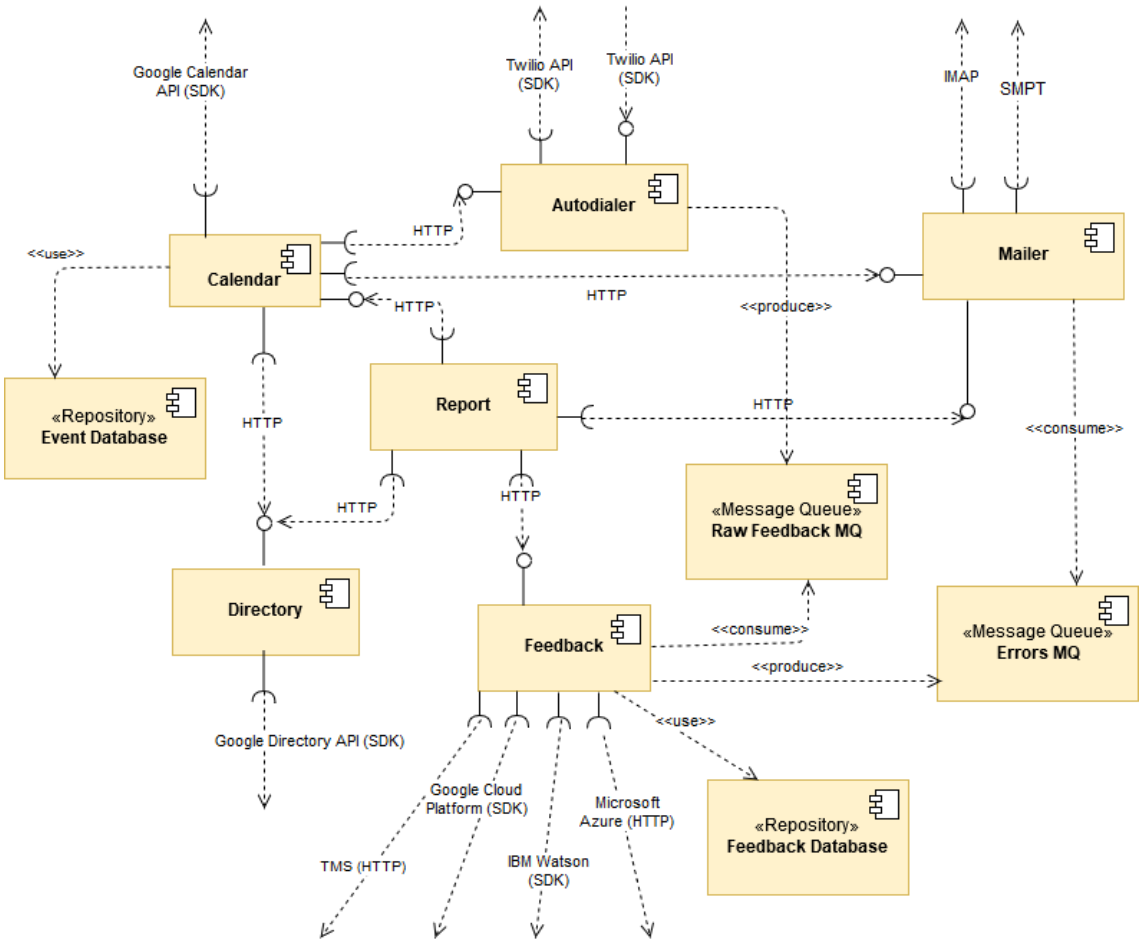


Ilustración 4.11 Diagrama de Componentes y Conectores

Catálogo de Elementos

Elemento	Responsabilidades
directory	➢ Este expone interfaz para obtener los usuarios y sus datos. Para esto consume el servicio *Google Directory* mediante el uso del SDK de *Google APIs*.
calendar	➢ Sincroniza los eventos de los usuarios con el servicio de *Google Calendar* mediante el SDK de Google APIs. ➢ Expone una interfaz web al resto de los microservicios para obtener los eventos del sistema. ➢ Consume los componentes *autodialer* y *mailer* con el fin de disparar solicitudes de *feedback* y se consume *directory* para obtener los usuarios y sus datos.
mailer	➢ Vía de comunicación mediante correo electrónico utilizando los protocolos de correo SMTP e IMAP. ➢ Expone interfaz para disparar solicitud de *feedback*. ➢ Produce *feedback bruto* en *Raw Feedback MQ* ➢ Consume errores a ser notificados desde *Errors MQ*.
autodialer	➢ Expone una interfaz para disparar solicitud de *feedback* vía llamada telefónica y otra para ser consumido por el servicio Twilio luego de finalizada la llamada. ➢ Consume al servicio Twilio mediante SDK para solicitar una nueva llamada telefónica.
feedback	➢ Consume el feedback bruto de *Raw Feedback MQ* para llevar a cabo su procesamiento y produce a Error MQ cuando haya errores internos. Para el procesamiento se consumen los distintos servicios de PLN, persistiendo luego el análisis generado. ➢ Consume el servicio de TMS para integrarse con Ascentis.
report	➢ Consume interfaces expuestas por los demás servicios con el fin de recabar los datos necesarios para generar reportes y enviar los mismos mediante correo electrónico.
Event Database	➢ Base de datos conteniendo los datos de los eventos.
Feedback Database	➢ Base de datos conteniendo los datos de los feedbacks.

Raw Feedback MQ	➢ Cola de mensajes para enviar feedback bruto de manera asincrónica.
Error MQ	➢ Cola de mensajes para manejar errores de manera asincrónica.

Tabla 4.2 Catálogo de elementos

Justificación

En la ilustración 4.11 Diagrama de Componentes y Conectores se puede observar una imagen del ecosistema de microservicios en tiempo de ejecución. Aquí se puede notar el favorecimiento de varios atributos de calidad. Considerando la posible detención de la ejecución de algún microservicio, los demás continuarán operativos sin ningún tipo de efecto colateral ante la falla ocurrida, lo que refleja el alto grado de disponibilidad. Por otro lado, como se comenta en la sección Monolito vs. Microservicios, escalar horizontalmente los servicios más críticos ante alta concurrencia es sumamente sencillo y eficiente. También se puede observar en esta vista el gran favorecimiento de la extensibilidad, generando componentes granulares utilizados como punto de interacción con el usuario (*autodialer y mailer*) sin incidir de ninguna manera en el funcionamiento del resto del ecosistema.

Tomando en cuenta que en mayor parte éste es un sistema de integración, el equipo toma la decisión de generar un híbrido de UML. Por eso se presentan las comunicaciones hacia sistemas externos mediante flechas de comunicación apuntando hacia el exterior del ecosistema. No se presentan los sistemas externos como componentes pero sí las comunicaciones hacia éstos, aclarando el protocolo o mecanismo de comunicación.

4.3.3 Vista de alocación

Vista de Despliegue

Representación Primaria

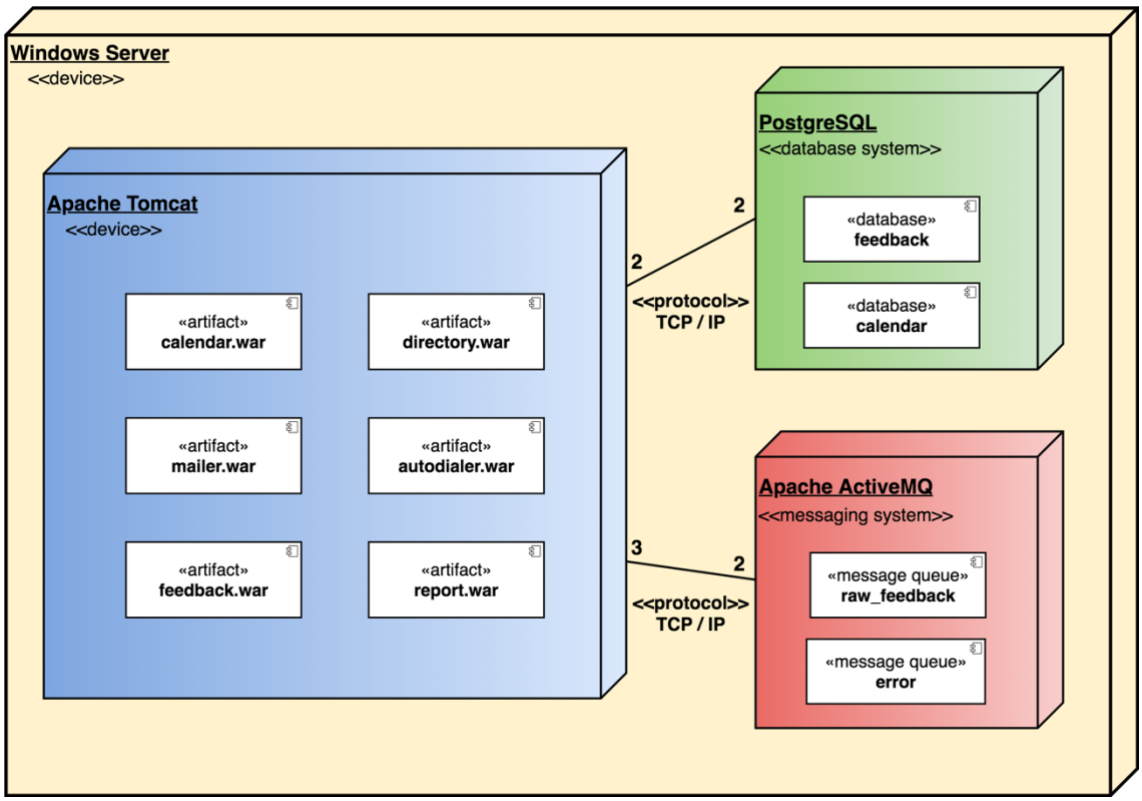


Ilustración 4.12 Diagrama de despliegue

Catálogo de Elementos

Elemento	Responsabilidades
Windows Server	Sirve como sistema operativo de la máquina virtual provista por Ascentis para el ambiente de producción. Se utiliza la versión **Windows Server 2016**.
Apache Tomcat	Servidor Web utilizado para instalar los artefactos de Feedback Assistant. En este se encuentra el ambiente de ejecución de la aplicación. Se utiliza la versión **Apache Tomcat 9.0**.
PostgreSQL	Motor de base de datos utilizado para persistir tanto los *feedbacks* como los eventos que se sincronizan de Google Calendar. Se utiliza la versión **PostgreSQL Server 10**.
Apache ActiveMQ	Servidor de mensajería para las colas de mensaje asociadas al *raw feedback* y a los errores asociados al procesamiento interno. Se utiliza la versión **Apache ActiveMQ 15.5.2**

calendar.war	Artefacto asociado al microservicio Calendar.
directory.war	Artefacto asociado al microservicio Directory.
mailer.war	Artefacto asociado al microservicio Mailer.
feedback.war	Artefacto asociado al microservicio Feedback.
autodialer.war	Artefacto asociado al microservicio Autodialer.
report.war	Artefacto asociado al microservicio Report.
feedback	Base de datos relacional que contiene la tabla de *feedbacks* procesados por el microservicio Feedback.
calendar	Base de datos relacional que contiene la tabla de eventos persistida por el microservicio Calendar.
raw_feedback	Cola de mensajes para transmitir asincrónicamente el *raw feedback* ya sea desde Mailer o Autodialer al microservicio Feedback.
errors	Cola de mensajes para transmitir asincrónicamente los errores desde Feedback a Mailer para notificar al usuario ante una falla interna en el procesamiento.

Tabla 4.3 Catálogo de elementos

Justificación

El ambiente de producción consiste de una máquina virtual alojada en los servidores físicos de Ascentis Uruguay. Esta ejecuta un sistema operativo Windows Server 2016 provisto por el cliente, pero vale aclarar que la solución podría ejecutarse sobre cualquier sistema operativo. Luego, el equipo se hizo responsable de instalar todos los componentes necesarios para ejecutar Feedback Assistant. La aplicación funciona actualmente bajo la plataforma Java 8.

Como se puede ver en la ilustración 4.12 Diagrama de despliegue, los microservicios son artefactos totalmente independientes. Esta es una gran ventaja que provee el patrón utilizado ya que brinda la capacidad de escalar horizontalmente los artefactos que sean más críticos en cuanto a carga de uso.

Para el actual sistema en producción se utiliza un único servidor web Apache Tomcat, pero aplicar el patrón de microservicios brinda la facilidad de manejar tantos servidores para alojar los artefactos como sea necesario. Se podría haber instalado cada artefacto en un servidor distinto (ver Anexo 11.7 Despliegue ideal), pero para mantener el despliegue simple en etapas de desarrollo se optó por

ubicar todos los microservicios de momento bajo el mismo servidor. Aquí una gran ventaja de la arquitectura en cuanto al despliegue; la libertad de instalación es totalmente abierta.

Otras alternativas evaluadas para el servidor web fueron JBoss y Glassfish, siendo éstas muy utilizadas en el mercado, pero con demasiadas funcionalidades innecesarias para las necesidades de este sistema. Éstas fueron desarrolladas para aplicaciones empresariales con implementación total de JavaEE, mientras que Feedback Assistant solo utiliza una pequeña sección de la misma. Apache Tomcat exige menos configuración inicial y es más liviano que las otras alternativas tecnológicas mencionadas.

4.4 Atributos de calidad

A continuación se presentan las tácticas aplicadas para favorecer los principales atributos de calidad, asociados a los requerimientos no funcionales relevados.

- **Modificabilidad**
    - Aumentar cohesión
        - Incrementar la coherencia semántica: ésta implica generar módulos donde sus elementos sirvan al mismo propósito. La aplicación del patrón en capas permitió generar una correcta separación de responsabilidades dentro de cada microservicio.

    - Disminuir acoplamiento
        - Encapsulamiento: Esta táctica refiere a disminuir la probabilidad de que el cambio en un módulo se propague a otros. Mediante la utilización de abstracciones entre dependencias se reduce la fuerza de acoplamiento debido a que las interfaces limitan las formas en las que interactúan los módulos. Éstas esconden el detalle interno que tiende a ser afectado por los cambios.

- Uso de intermediarios: La utilización de colas de mensajes entre microservicios permitieron introducir un mecanismo de publicador-subscriptor. Por ejemplo, cuando se recibe el feedback en *mailer* luego de que el usuario responde la solicitud del mismo, este no tiene absoluto conocimiento del módulo encargado de procesarlo, evitando una dependencia innecesaria. Lo mismo sucede cuando *autodialer* encola el feedback recibido telefónicamente.

- Restricción de dependencias: Como se comenta anteriormente en la presente sección, el equipo toma la decisión de respetar el orden de las capas establecidas, restringiendo por ejemplo el uso de clases del módulo *repository* en el módulo *controller* donde se reciben las peticiones HTTP.

  - Diferir Enlaces
    - Archivos de recursos: La utilización de archivos de configuración es un mecanismo para diferir enlaces en tiempo de ejecución. Mediante el uso de estos, se evita tener que realizar cambios en el código y tener que generar nuevamente el artefacto.

- **Portabilidad**
  - Entorno de ejecución
    - Minimizar dependencias hacia la plataforma: Esta implica generar software que ejecute sobre una máquina virtual. Feedback Assistant ejecuta sobre la JVM (Java Virtual Machine), lo que permite encapsular las dependencias sobre la plataforma. De esta forma, la aplicación es capaz de ser ejecutada en cualquier sistema operativo.

- **Extensibilidad**
  - White-Box
    - Glass-Box: White-Box implica agregar funcionalidades modificando o agregando código fuente. Dentro de esta, el equipo toma la decisión de aplicar Glass-Box, la cual implica restringir al desarrollador en modificar código existente al agregar una nueva funcionalidad. La arquitectura diseñada aplica esta táctica, debido a que las funcionalidades que se tienden a extender son los canales de comunicación con el usuario. Teniendo un microservicio por cada medio de comunicación, al agregar uno nuevo no surge la necesidad de modificar código existente. Este atributo de calidad está directamente asociado a la modificabilidad, y dado que también se hizo énfasis en ésta, la extensibilidad se vió notoriamente favorecida.

- **Eficiencia**
  - Gestión de recursos
    - Introducción de concurrencia: Mediante las colas de mensajes la aplicación es capaz de desacoplar asincrónicamente la responsabilidad de recepcionar el *feedback* (ubicada en los microservicios que sirven como canales de comunicación) con la de procesarlo. Por tanto se evita realizar todas estas acciones en el mismo hilo de ejecución y de esta forma optimizar los tiempos de respuesta. Si no se hubiese optado por aplicar esta táctica, la respuesta de recepción exitosa (o errónea) del *feedback* no sería entregada hasta no finalizar con todo el procesamiento del *feedback* (escenario sincrónico).

- **Disponibilidad**
  - Recuperación ante fallas
    - Manejo de excepciones: Cuando el sistema experimenta cualquier tipo de falla, se identifican y manejan las excepciones asociadas a la misma. Mediante este mecanismo no se ignora la falla, sino que se delega el error hasta el nivel de interacción con el usuario.

    - Rollback: El motor de base de datos PostgreSQL implementa internamente mecanismos de rollback para gestionar correctamente la integridad de los datos.

  - Prevención de fallas
    - Transacciones: Al utilizar bases de datos relacionales y colas de mensaje se agrega el concepto de transaccionalidad, cumpliendo de esta forma con las propiedades *ACID* [21].

4.5 Validación de la arquitectura

Luego de que el equipo consolidó el diseño final de la arquitectura general de la solución, consideró fundamental contar con una instancia de validación de la misma con el cliente.

Recordando lo mencionado en la sección 1.3 Descripción del Cliente, Ascentis abre una nueva oficina en Uruguay en el año 2017. Por esto, requieren de intercambios culturales entre miembros de la organización ya sea para transferencia de conocimientos o unificar procesos y culturas de trabajo. Dicho esto, el equipo aprovecha la visita de los arquitectos de software líderes de la empresa a Uruguay, agendando una instancia de validación de la arquitectura en las oficinas de Ascentis Uruguay para el 14 de Marzo del 2018.

Ilustración 4.13 Validación de arquitectura

Esta fue una experiencia realmente positiva para todos los integrantes del equipo, no sólo por el hecho de exponer el trabajo realizado en el diseño de la arquitectura, sino también debido a las interesantes discusiones que se llevaron a cabo a lo largo de la reunión. Como se puede ver en la ilustración, el equipo diagramó la arquitectura de alto nivel en el pizarrón, mostrando los flujos y procesos principales de la solución. De esta forma, se pudieron justificar las técnicas utilizadas para lograr favorecer la calidad y cumplir con los requerimientos no funcionales.

Afortunadamente, los expertos se mostraron muy satisfechos respecto a los mecanismos utilizados y el enfoque de solución propuesto. Igualmente, éstos presentaron comentarios y sugerencias sumamente trascendentes que hicieron aún más efectiva y eficiente la solución arquitectónica. A continuación se detallan las sugerencias más importantes propuestas por el cliente para agregar valor a la arquitectura:

- Evitar cachear lo máximo posible ya que mantener la integridad de los datos que provienen de un servicio externo es un gran desafío. Por ende, estos expertos recomiendan al equipo evitar cacheos innecesarios. Esto surgió debido a la persistencia de usuarios sincronizados con Google G-Suite en el microservicio *directory*. El equipo había tomado la decisión inicial de utilizar una base de datos para persistir los usuarios sincronizados, cuando en realidad ya se contaba con el servicio externo que expone los datos. No tenía ningún sentido mantener esa información en el sistema cuando Google Directory ya cuenta con la responsabilidad de gestionar estos datos.

- Se recomienda el uso de conectores para interactuar con servicios externos. El equipo había logrado buenos niveles en cuanto a extensibilidad, que fueron mejorados aún más gracias a este encuentro con los arquitectos líderes de Ascentis. Al inicio se acoplaban la lógica de negocio y los clientes encargados de interactuar con sistemas externos. Por esto, recomiendan generar una separación entre ambas responsabilidades. El equipo revierte esta situación agregando el módulo *client* en cada

microservicio. De esta forma la lógica de negocio no tiene conocimiento de las comunicaciones.

4.6 Conclusiones y lecciones aprendidas

Se logró cumplir tanto con el objetivo de proyecto asociado a la satisfacción de Ascentis en cuanto a la solución arquitectónica (teniendo en cuenta su futura intervención sobre Feedback Assistant para continuar brindándole valor), así como también con el objetivo académico asociado a la incorporación de nuevos conocimientos e incursión de tecnologías novedosas.

A lo largo de la carrera se han dado a conocer de forma teórica las características de la arquitectura de microservicios. Cuando se debía implementar una solución de este tipo, nunca se exigió una solución totalmente basada en microservicios, sino que se partía de una arquitectura monolítica (primera versión del proyecto) y luego se solicitaba generar una solución híbrida de microservicios (segunda versión del proyecto) migrando cierto sector del monolito a microservicios. Dicho esto, generar la solución inicial de Feedback Assistant fue un gran desafío y dejó muchos aprendizajes a los integrantes del equipo. A modo de ejemplo, uno de los aprendizajes más relevantes obtenidos fue el manejo de entidades de negocio a lo largo del ecosistema. Hay que realizar consideraciones en base a los atributos de calidad a favorecer sobre si alojarlas en un módulo independiente o repetirlas en los servicios cuando sea necesario, mostrando aquí el enfrentamiento entre modificabilidad y reuso.

Por tanto, aplicar esta arquitectura y generar las discusiones constructivas a lo largo del diseño permitió ir tomando decisiones cada vez más sólidas a lo largo del tiempo.

Se concluye también que pudieron ser contemplados todos los requerimientos no funcionales aplicando distintas tácticas que apuntaron a favorecer los atributos de calidad asociados a éstos.

5 Análisis tecnológico

En esta sección se describen las tecnologías utilizadas en la solución. También se presentan los desafíos técnicos más relevantes a los cuales el equipo se enfrentó junto con las actividades realizadas para lograr la correcta resolución de los mismos.

5.1 Descripción de las tecnologías utilizadas

Teniendo en cuenta la necesidad de Ascentis en favorecer la portabilidad del sistema, se debió manejar un *stack* de tecnologías que permitan la ejecución de Feedback Assistant en todos los sistemas operativos existentes.

Dicho esto, en las primeras fases del proyecto el equipo procede a realizar la prueba de concepto sobre las *APIs* a integrarse mediante la tecnología *Java EE*. Los resultados fueron positivos teniendo en cuenta las demos realizadas al cliente y la satisfacción del mismo. Igualmente el equipo percibió una baja productividad durante el desarrollo de las mismas por lo que realizó un análisis de alternativas tecnológicas para obtener una mayor productividad.

Aquí surge la opción de utilizar *Spring Framework*. Ésta es una herramienta basada en *Java EE* la cual provee soluciones a problemas típicos que se viven en la programación. Esto permite al desarrollador enfocarse en su tarea y generar así una mayor productividad.

Algunas de las ventajas más importantes del *framework* son:

- Permite programación orientada a aspectos, habilitando la implementación de rutinas transversales encapsulando responsabilidades como por ejemplo el *logging*, seguridad, etc.

- Provee inversión de control ya que el mismo se encarga de crear e inyectar los *beans* entre ellos teniendo en cuenta las dependencias existentes. Así se logra cumplir con el principio de diseño asociado a la inversión de dependencias.

- Proporciona un contenedor para administrar el ciclo de vida de los *beans* [22].

- Cuenta con una capa genérica de gestión de transacciones para facilitar el manejo con los repositorios de datos o mecanismos de mensajería.

- Provee un enfoque de convención sobre configuración para facilitar la configuración de la aplicación.

- Mediante anotaciones se definen controladores de recursos HTTP, servicios de lógica de negocio, repositorio de datos y tareas periódicas permitiendo al desarrollador olvidarse por completo de las implementaciones internas para cumplir con estas responsabilidades.

A continuación se presenta una ilustración para dar a conocer la arquitectura interna de *Spring Framework*.

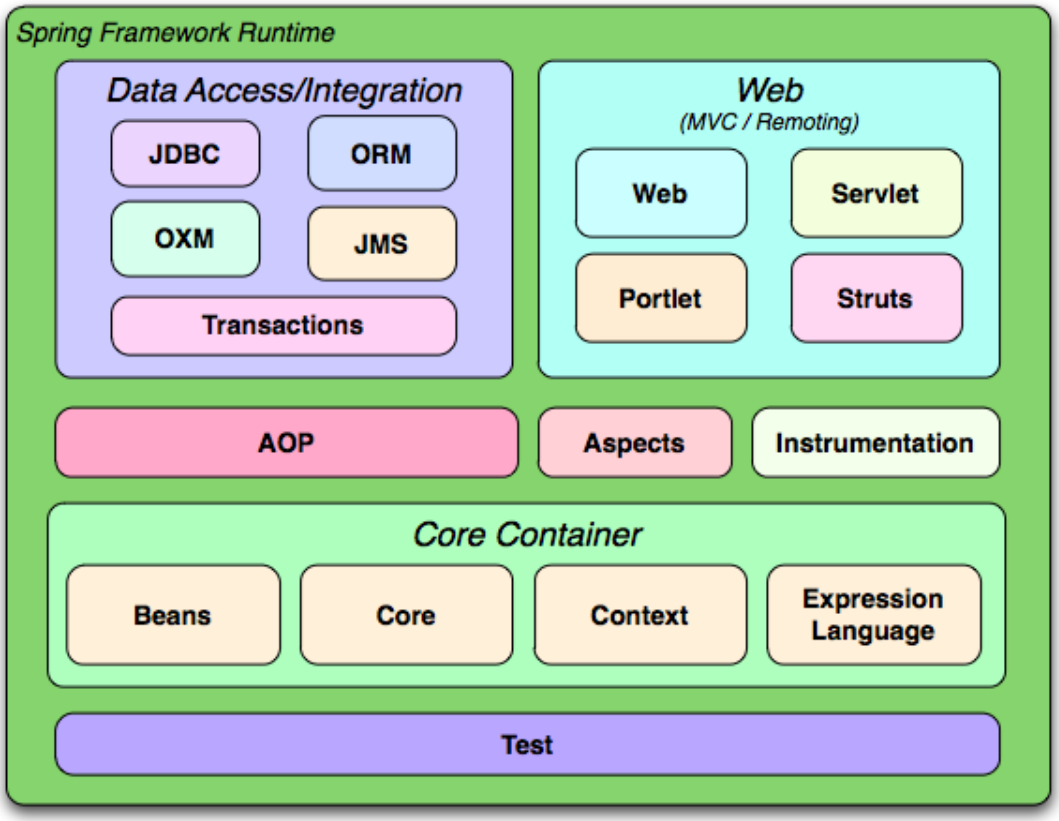


Ilustración 5.1 Arquitectura de Spring Framework [23]

Además del framework para el desarrollo, se utilizaron otras herramientas basadas en la tecnología *Java*:

- Para las colas de mensaje se utilizó *Apache ActiveMQ* [24]. Éste es un *broker* de mensajería de código abierto que implementa totalmente la especificación de *JMS (Java Message Service)*.

- Para la gestión de dependencias dentro del sistema se utilizó *Apache Maven* [25]. Mediante ésta también se generan los artefactos para ser alojados en el servidor de producción. Se basa en el concepto de un modelo de objeto de proyecto denominado POM (*Project Object Model*) para definir las características de la aplicación utilizando formato XML. A continuación se presenta una ilustración a modo de ejemplo mostrando el POM de uno de los microservicios.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.ascentis.feedbackassistant</groupId>
    <artifactId>directory</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>

    <name>com-ascentis-feedbackassistant-directory</name>
    <description>Feedback Assistant Directory</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.8.RELEASE</version>
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>com.ascentis.feedbackassistant</groupId>
            <artifactId>model</artifactId>
            <version>0.0.1-SNAPSHOT</version>
        </dependency>

         <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
```

Ilustración 5.2 POM de microservicio directory

- Como se menciona en la sección 4.3.3 Vista de Alocación, se utiliza como servidor web Apache Tomcat [26] el cual se basa en la tecnología *Java* para el despliegue de los servicios.

En cuanto al IDE para el desarrollo del producto, se evaluaron las herramientas IntelliJ IDEA y Eclipse. Se optó por la utilización de la primera debido a que presenta una experiencia de desarrollo (intuición respecto a navegación y codificación en la herramienta) superior respecto a la segunda teniendo en cuenta la experiencia previa sobre ambas. Se cree que la característica más importante para esta elección de herramientas es la usabilidad de la misma y todos los integrantes concordaron en que IntelliJ IDEA es sumamente superior en cuanto a usabilidad respecto a Eclipse.

5.2 Principales desafíos tecnológicos

- **Impersonalización de usuarios**

Dos de las características más importantes del proyecto a recalcar son que Feedback Assistant se basa en la integración con *Google G-Suite* y a su vez no utiliza interfaz gráfica para la interacción con los usuarios finales. Como se menciona en el capítulo *3. Ingeniería de Requerimientos*, Ascentis solicitó que la interacción entre el usuario y el sistema se realice mediante las herramientas colaborativas que se utilizan a diario, como ser el calendario y correo electrónico.

Dicho esto, la solución debía ser capaz de obtener todos los datos referentes a los contactos y calendarios de los usuarios dentro de la organización. El gran problema al cual el equipo se enfrentó fue la necesidad de acceder a los datos de los usuarios (los cuales son absolutamente confidenciales debido a las estrictas políticas de seguridad de Google [27]) sin interfaz gráfica de por medio. Es fundamental que la obtención de datos no deba involucrar ninguna interacción por parte de los usuarios, sino que se realice de forma transparente para los mismos. El sistema no debe depender de la aprobación por parte de cada

empleado para acceder a sus datos, ni de la operación de ningún administrador que deba dar de alta o baja a nuevos usuarios.

Si hubiese sido válida la utilización de una UI se hubiese redireccionado al usuario a Google para que inicie sesión y apruebe la solicitud de acceso a sus datos como se muestra en la siguiente ilustración.

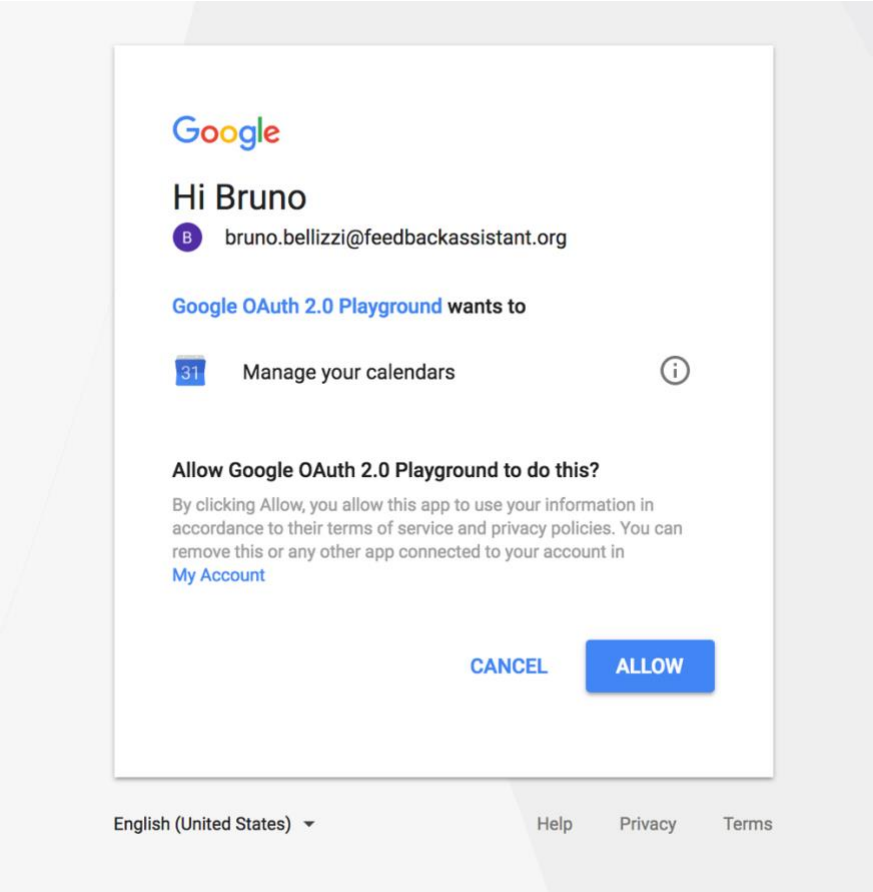


Ilustración 5.3 Solicitud de consentimiento para acceso a datos de Google

Por tanto se debió a buscar otra solución a este problema de seguridad. Luego de una detallada investigación, se llegó a la conclusión de que el único camino posible para la obtención de los datos sin autorización manual del usuario es la utilización de una cuenta de servicio dentro de una organización de Google G-Suite.

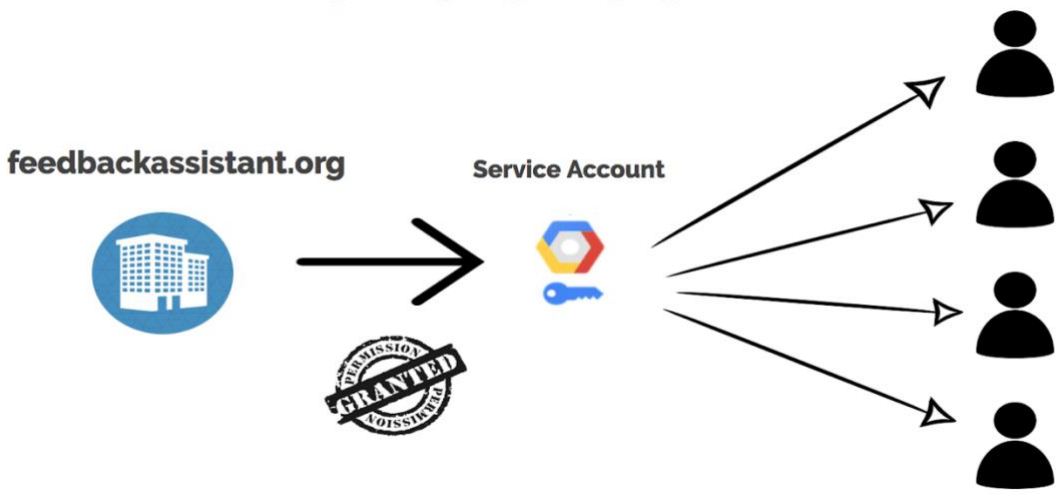


Ilustración 5.4 Delegación de autoridad en Google G-Suite

Para esto el equipo registró la organización Feedback Assistant en la plataforma mediante un plan básico a modo de simular una organización real y poder llevar a cabo el desarrollo basándose en la misma. Ésta se encuentra bajo el dominio *feedbackassistant.org* y tiene un costo de U$S 5 mensuales por usuario [28].

Posteriormente se debió crear una cuenta especial que pertenece a una aplicación o máquina virtual en lugar de pertenecer a un individuo denominada cuenta de servicio [29]. La misma se identifica por un correo electrónico y se comunica directamente con los servicios de Google sin involucrar usuarios en la interacción.

Una vez creada la cuenta de servicio, el administrador de la organización (integrante del equipo) debió brindar los permisos necesarios a la misma para que sea posible acceder a los datos de contactos y calendarios de los usuarios dentro de la organización (delegación de autoridad). Se debió descargar una clave pública y una privada con el fin de hacer posible la comunicación segura entre Feedback Assistant y Google G-Suite. Esta comunicación se realiza mediante OAuth 2.0 [30].

A continuación se presenta una ilustración del formato de las credenciales alojadas en el sistema para la comunicación con *Google Directory* y *Google Calendar*.

```
{
  "type": "service_account",
  "project_id": "feedbackassistant2018",
  "private_key_id": "0571840f465fd322078e70b58c14025bc1551295",
  "private_key": "-----BEGIN PRIVATE KEY-----\nMIIEvgIBADANBgkqhkiG9w0BAQEFAASCE
  "client_email": "servaccount2@feedbackassistant2018.iam.gserviceaccount.com",
  "client_id": "109524699734630723036",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://accounts.google.com/o/oauth2/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/ser
}
```


Ilustración 5.5 Ejemplo credenciales Google G-Suite

De esta forma se logró resolver el problema de la impersonalización de usuarios para la sincronización con GSuite, sin necesidad de la intervención manual por parte de cada usuario.

- **Análisis de sentimiento impreciso**

Cuando el equipo se encontraba inmerso en el desarrollo del MVP, período en el que se llevó a cabo la integración con el servicio de procesamiento de lenguaje natural de *Google* para analizar el sentimiento de los feedbacks recibidos, se notó la baja precisión en algunos de los resultados retornados por el servicio. Como se puede observar en la siguiente ilustración a modo de ejemplo, se enviaba un texto positivo y la API retornaba un sentimiento neutro.

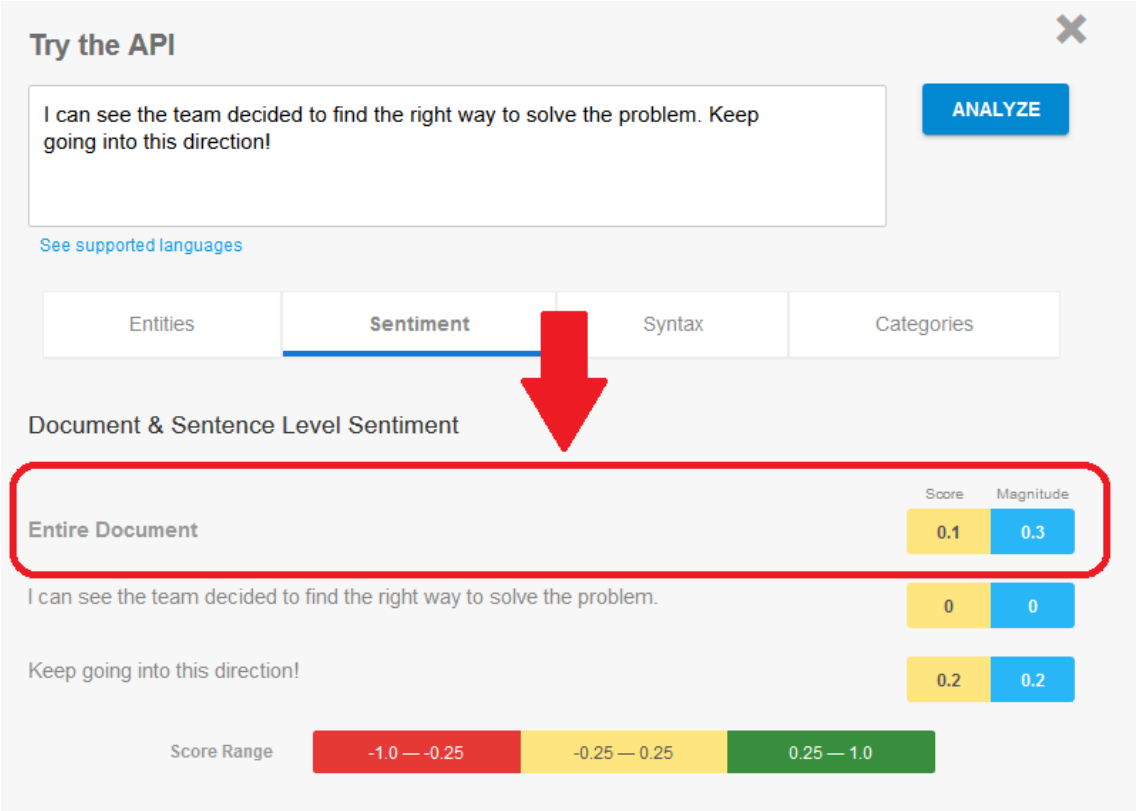


Ilustración 5.6 Análisis de sentimiento utilizando Google Natural Language API

Esta situación afectaba de forma negativa al producto, debiendo ser revertida de alguna forma con el fin de generar análisis más precisos sobre el sentimiento del *feedback*.

Luego de varias discusiones el equipo toma la decisión de abordar este problema agregando dos nuevas integraciones con servicios de PLN, obteniendo de esta forma el análisis proveniente de tres fuentes distintas y luego generando un promedio entre ellos para obtener un resultado final de análisis con menor margen de error.

Se llevó a cabo una investigación de alternativas tomando en cuenta que era necesario interactuar con los mejores servicios en cuanto a precisión en el análisis de sentimiento. Esta investigación se basó en la opinión de la comunidad respecto a su experiencia y mediante empirismo al utilizar las demos provistas por los servicios y evaluando sus resultados.

Finalmente se optó por agregar la integración con los servicios Microsoft Azure *(Text Analytics API* [31]*)* e IBM Watson *(Natural Language Understanding API* [32]*)*.

Luego de implementar integración con éstos y llevar a cabo el promedio entre los resultados de los tres servicios, los resultados comenzaron a ser más precisos respecto a lo expresado en el *feedback* provisto.

Considerando el mismo ejemplo de *feedback* utilizado en la Ilustración 5.6 Análisis de sentimiento utilizando Google Natural Language API se muestran los resultados de IBM Watson y Microsoft Azure para luego presentar el promedio de los tres resultados y así dar evidencia de la mejoría en el análisis final.

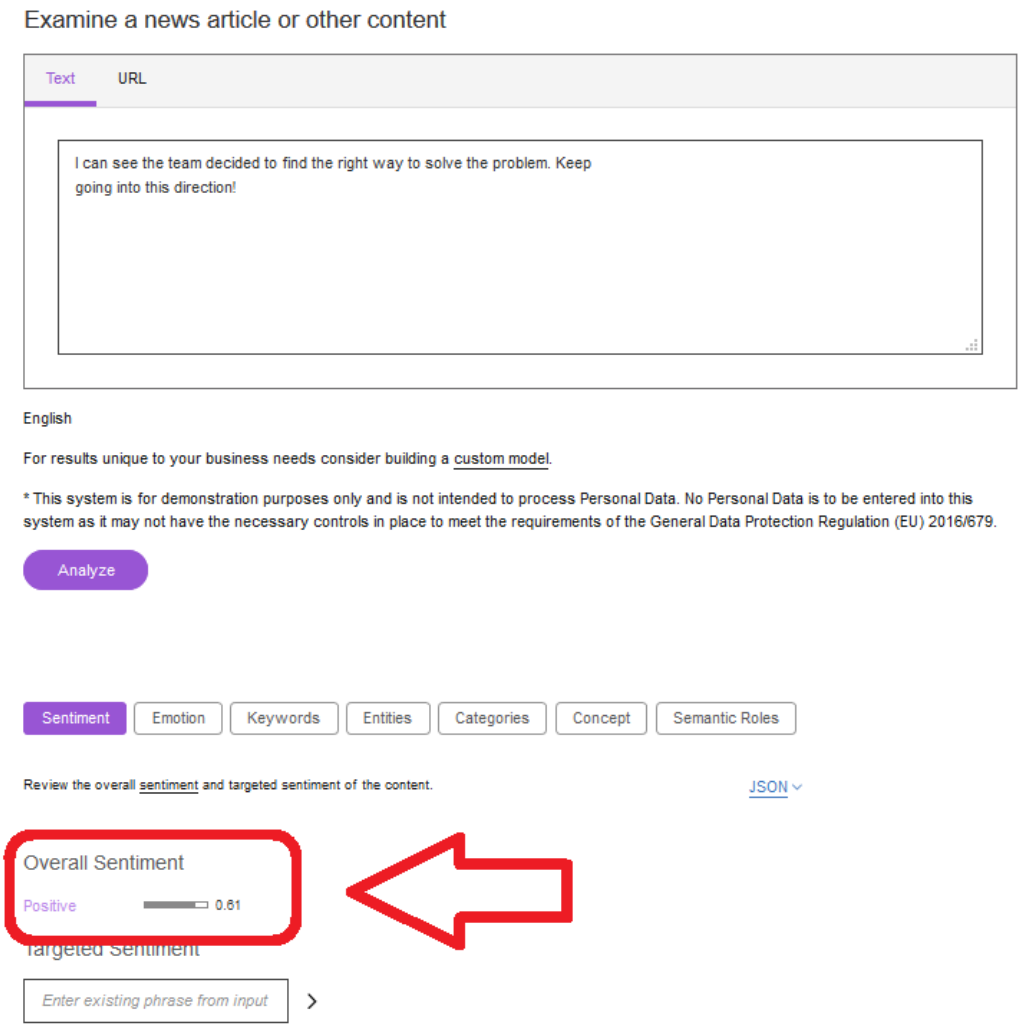


Ilustración 5.7 Análisis de sentimiento utilizando IBM Watson - *Natural Language Understanding API*

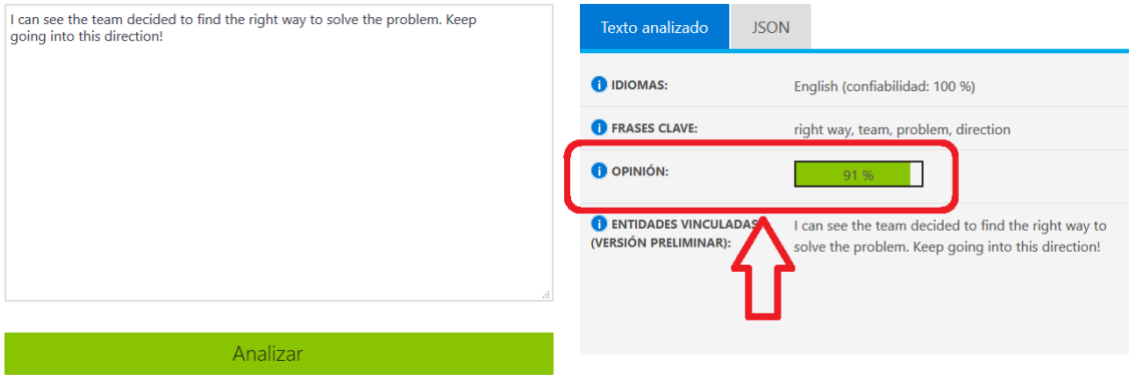


Ilustración 5.8 Análisis de sentimiento utilizando *Microsoft Azure* - Text Analytics API.

En la siguiente tabla se muestra el resultado del análisis en porcentaje para cada servicio sobre cuán positivo fue el *feedback* provisto.

Servicio	Cuán positivo fue el feedback asociado al ejemplo (%)
Google	55%
IBM Watson	80%
Microsoft Azure	91%

Tabla 5.1 Tabla de porcentajes de análisis de ejemplo presentado.

Nota: Cada servicio maneja una escala distinta por lo que el equipo convirtió cada valor a porcentaje para facilitar la comprensión al lector.

Por ende, el promedio de los porcentajes obtenidos en el ejemplo anterior resulta ser de 75%. Aquí se tiene evidencia de que este mecanismo utilizado proporciona resultados más precisos respecto a utilizar únicamente el servicio de Google para el análisis del sentimiento. El *feedback* provisto inicialmente tenía un porcentaje del 55% cuando se utilizaba únicamente el servicio de Google y luego mediante el promedio del resultado de los tres servicios fue de 75%, lo que es sumamente coherente respecto al texto procesado. El hecho de que Microsoft Azure haya retornado un 91% no implica que sea mejor que el promedio entre todos si se

tiene en consideración el contenido del *feedback* (no es tan positivo como para obtener un análisis del 91%).

Para evidenciar la mejoría del análisis en la mayoría de los casos respecto a utilizar únicamente Google Natural Language API se repite la comparación de resultados de análisis con múltiples ejemplos de *feedback*. Estas comparaciones se encuentran en el Anexo 11.6 Comparación de análisis de sentimiento de *feedbacks.*

- **Manejo de entidades de negocio**

Una de las complejidades que introduce el enfoque arquitectónico utilizado es el manejo de entidades de negocio. Dada la inexperiencia del equipo respecto a los microservicios, inicialmente se ubicó un módulo de entidades de negocio en cada aplicación. Esto generó un gran desfavorecimiento de la modificabilidad ya que cuando se debía modificar una entidad, el cambio debía ser realizado en cada microservicio que la contenía. Además esto trae también la desventaja de tener código repetido a lo largo del sistema.

Para revertir esta situación se creó una nueva biblioteca de clases (*model*) con la única responsabilidad de servir como proveedor de entidades de negocio al resto del sistema. De esta forma se lograron encapsular todas las entidades de negocio en único sector de la arquitectura y se evitaron los problemas previamente mencionados. Esta biblioteca es inyectada mediante *Maven* [33] en los demás servicios.

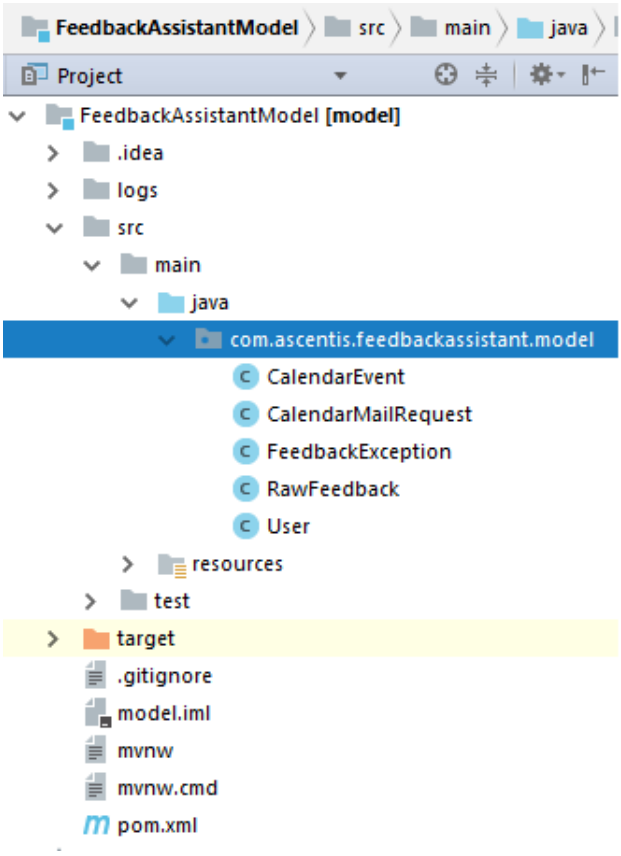


Ilustración 5.9 Estructura modular de la biblioteca de entidades model

Igualmente se continuó experimentando problemas con las entidades de negocio. Como es mencionado en el capítulo 4. Descripción de la solución de *software*, el microservicio *calendar* y *feedback* manejan interacción con dos repositorios de datos independientes (una base de datos por microservicio). El equipo utilizó la implementación de JPA (Java Persistence API) [34] provista por la herramienta *Hibernate* [35] para la interacción con la base de datos. En JPA una entidad de persistencia es una clase Java, cuyo estado es persistido asociándolo a una tabla en una base de datos relacional [36]. La herramienta detecta las entidades de persistencia mediante la presencia de anotaciones. Las anotaciones son una forma de añadir metadatos al código fuente para manejar distintos comportamientos en tiempo de ejecución. Para definir una entidad de persistencia basta con agregar la anotación @Entity previo a la definición de la clase como se muestra en la siguiente ilustración.

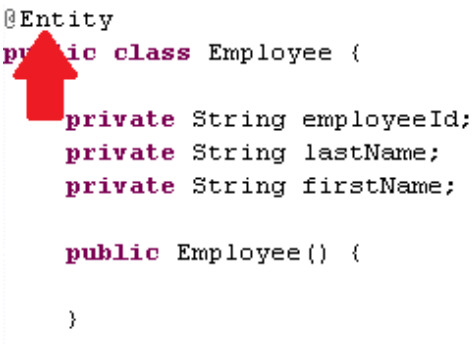


Ilustración 5.10 Ejemplo entidad de persistencia JPA

Por lo tanto las anotaciones para la persistencia se ubicaron en las entidades *CalendarEvent* y *RawFeedback* de la aplicación *model* ya que son las únicas entidades que se persisten en el sistema (las persisten los microservicios *calendar* y *feedback* respectivamente). Dado que estas entidades también son utilizadas en otros microservicios como por ejemplo en *report* para generar los reportes de *feedbacks* analizados, se experimentaron errores en los mismos ya que no contaban con la dependencia *Hibernate* y al contar con la presencia de la anotación @Entity estas aplicaciones fallaban al inicializar.

Este problema pudo ser solucionado mediante la ubicación de una subclase por cada entidad de persistencia en los respectivos microservicios que manejan persistencia de datos. Utilizando una nueva anotación denominada *@MappedSuperclass* en las entidades genéricas utilizadas por todos los microservicios se logró corregir este error de inicialización de las aplicaciones.

Una clase designada con la anotación MappedSuperclass se puede correlacionar de la misma manera que una entidad, excepto que las asignaciones se aplicarán sólo a sus subclases, ya que no existe una tabla para la propia superclase mapeada. Cuando se aplica a las subclases, las asignaciones heredadas se aplicarán en el contexto de las tablas de las subclases [37].

En las siguientes ilustraciones se evidencia el uso de esta anotación.

```java
@MappedSuperclass
public class CalendarEvent implements Serializable {

    @Id
    private String id;

    @Column
    private String summary;

    @Column(name = "startDate")
    @JsonFormat(pattern = "yyyy-MM-dd@HH:mm")
    private Date start;

    @Column(name = "endDate")
    @JsonFormat(pattern = "yyyy-MM-dd@HH:mm")
    private Date end;

    @Column
    private Boolean notified;

    @ElementCollection(fetch = FetchType.EAGER)
    private Map<Integer, String> attendees;
```

Ilustración 5.11 Entidad CalendarEvent en model

```java
package com.ascentis.feedbackassistant.calendar.model;

import javax.persistence.Entity;

@Entity(name = "event")
public class CalendarEvent extends com.ascentis.feedbackassistant.model.CalendarEvent {

}
```

Ilustración 5.12 Entidad CalendarEvent en microservicio calendar

- **Solicitud de feedback vía llamada telefónica**

En busca de que la solución sea totalmente innovadora y que sorprenda positivamente al cliente, el equipo se enfrentó a un gran desafío tecnológico siendo este uno de los más complejos en el proyecto. Esta complejidad estaba dada principalmente por la nula experiencia en la implementación de autómatas que realicen llamadas telefónicas.

A continuación se listan las complejidades más notorias que el equipo debió abordar:

- Conseguir un operador de telefonía para realizar las llamadas.
- Implementar un bot que dado un texto lo emita en la llamada.
- Grabar la voz del usuario durante la llamada.
- Transformar el audio grabado a texto.
- Recibir el texto en el sistema.

Se realizaron múltiples *spikes* previo al desarrollo de esta funcionalidad con el fin de buscar una solución que facilite todos los problemas listados.

Finalmente se optó por utilizar una plataforma de comunicación en la nube denominada *Twilio* [38]. Este servicio provee múltiples funcionalidades y entre ellas las listadas previamente, lo que permitió delegar estas responsabilidades a un servicio en la nube de suma confianza. Éste es utilizado por una gran cantidad de empresas de alto porte como Coca-Cola, DELL, Uber, entre otros, lo que genera confianza en su utilización, brindando también la ventaja de evitar el manejo de infraestructura adicional ya que se aloja en la nube [39].

La comunicación con el servicio *Twilio* se lleva a cabo mediante una *API REST* y además éste provee un *SDK* para *Java* con el objetivo de facilitar el trabajo al desarrollador.

A continuación se presenta una ilustración mostrando la interacción entre la aplicación, *Twilio* y el usuario final.

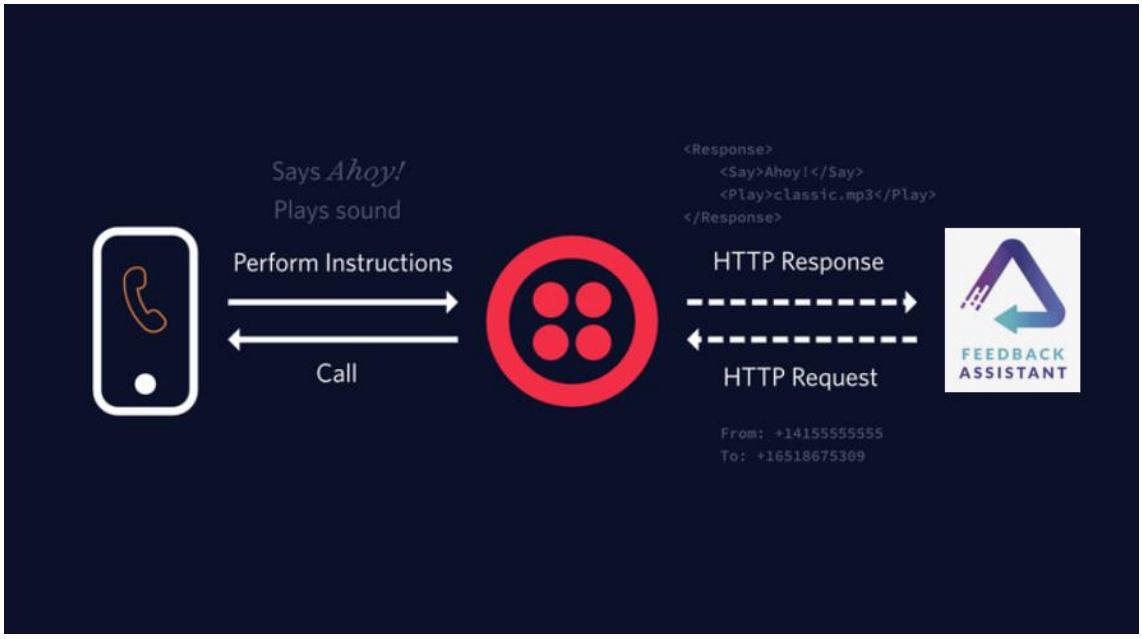


Ilustración 5.13 Comunicación con Twilio

Este mecanismo consiste de dos comunicaciones independientes. Primero la aplicación solicita a Twilio la realización de una llamada telefónica. Luego éste lleva a cabo la misma comunicandose con el usuario final y una vez finalizada, Twilio genera otra solicitud HTTP a la aplicación conteniendo toda la información de la llamada. Al tratarse Twilio de un servicio alojado en internet, se debió solicitar a Ascentis la redirección de un puerto específico en el router al host en donde el equipo alojó la aplicación. Con esto se permitió la recepción de la solicitud HTTP correspondiente a la segunda comunicación del proceso descrito mediante un puerto específico.

5.3 Conclusiones y lecciones aprendidas

Para cerrar este capítulo se quiere expresar el alto nivel de satisfacción interno respecto a la resolución de los problemas tecnológicos planteados. Se ha logrado trabajar en equipo complementando las habilidades de cada integrante para lograr solucionar las complejidades presentadas. También vale recalcar la mejoría en el desempeño de la solución mediante la integración con múltiples proveedores de PLN promediando luego los resultados obtenidos.

Se desean recalcar los conocimientos obtenidos asociados a la seguridad en cuanto a la impersonalización de usuarios, donde el equipo no era consciente de esta funcionalidad de G-Suite. Por otro lado hacer énfasis en el aprendizaje en cuanto al manejo del modelo de datos a lo largo de una arquitectura basada en microservicios.

Por último pero no menos importante, destacar la adquisición de nuevos conocimientos relacionados a la implementación de autómatas que realicen llamadas telefónicas dinámicas en base a datos provistos al mismo.

6 Gestión de proyecto

Desde el inicio del proyecto, el equipo coincidió en la importancia de una eficiente organización para la asignación de tareas, registro del trabajo y seguimiento del mismo. Se buscó obtener en todo momento una perspectiva clara y objetiva que permita identificar rápidamente desvíos y realizar acciones correctivas y preventivas en etapas tempranas del proyecto.

En este capítulo se exponen los procesos, herramientas de trabajo y metodologías con sus variaciones que el grupo utilizó para lograr una correcta gestión. También se detalla cómo se gestionaron los potenciales riesgos del proyecto, así como su análisis cualitativo y planes de respuesta. Finalmente, se exponen las conclusiones y lecciones aprendidas en cuanto a la gestión aplicada.

6.1 Marco de trabajo

En esta sección se define el marco de trabajo utilizado durante el proyecto, indicando el ciclo de vida y metodologías empleadas para la organización del mismo. Se tomaron en cuenta factores como la volatilidad de los requerimientos, la experiencia del equipo y las sugerencias del cliente.

6.1.1 Asignación de roles

Al comienzo del proyecto, surgió la necesidad de definir las responsabilidades de cada integrante del equipo mediante la asignación de roles. Esto se debe principalmente a que las tareas y actividades a realizar en un proyecto con tal alcance son variadas e importantes.

Es por lo anterior que el encargado de cada área tuvo la responsabilidad de asegurar que se lleven a cabo todas las actividades de su ámbito, pero la ejecución de dichas actividades fueron realizadas de manera colaborativa. El responsable también debía asegurar el seguimiento y completitud de los documentos relacionados a su rol, actualizandolos cuando ocurrían cambios que lo ameritaba (como cambios en los requerimientos, re-evaluaciones de riesgos o cambios arquitectónicos).

Para la asignación se consideraron las fortalezas y preferencias de cada integrante con el rol asignado. Los roles fueron asignados de la siguiente manera:

Integrante	**Roles**
Bruno Bellizzi	Gerente de proyecto *Scrum Master* Desarrollador
Nicolás Eiris	Arquitecto de *software* Encargado de SCM Desarrollador
María Inés Fernández	Gerenta de Calidad Ingeniera en requerimientos Desarrolladora

Tabla 6.1 Asignación de roles

6.1.2 Ciclo de vida

Debido a que los principales requerimientos fueron propuestos por el cliente al inicio del proyecto y que los mismos fueron relativamente estables, se decidió trabajar siguiendo un ciclo de vida iterativo e incremental. Incremental en la medida que el producto de *software* fue incorporando un subconjunto de nuevas funcionalidades en intervalos pequeños de tiempo o iteraciones, permitiendo contar con pequeños incrementos funcionales que podrían validarse con el cliente de forma continua. E iterativo considerando que además de agregar nuevas funcionalidades, también se hizo énfasis en mejorar la calidad del producto en cada iteración.

Con este enfoque primero se decidió qué se hará, y luego qué se hará primero. Después de cada iteración se planeó lo próximo a desarrollar, repitiendo este proceso hasta que el proyecto finaliza de acuerdo a lo pactado con el cliente.

Se descartó un ciclo de vida en cascada debido a que el cliente exigía muestras continuas en tiempos cortos para conocer el estado del producto periódicamente. Además, no se contaba con un enfoque arquitectónico definido en las primeras etapas, por lo que sería sumamente costoso revertir un error en las decisiones

arquitectónicas en etapas avanzadas del proyecto o realizar cualquier otro cambio sobretodo teniendo en cuenta las múltiples interrogantes que planteaba el proyecto al inicio.

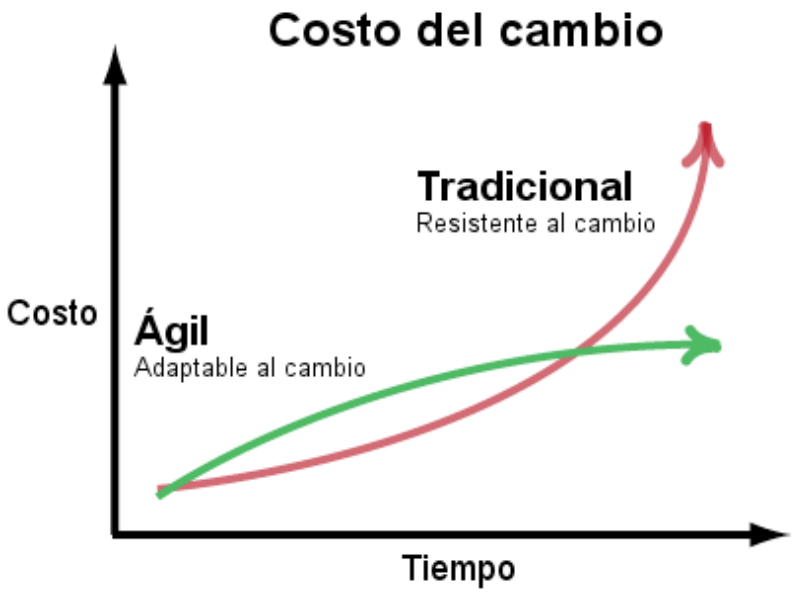


Ilustración 6.1 Costo del cambio Ágil vs. Tradicional

6.1.3 Metodología de referencia

Considerando las características del proyecto, el equipo optó por utilizar elementos de metodologías ágiles en conjunto con elementos de metodologías tradicionales.

De las metodologías tradicionales se tomaron elementos que permitían una planificación a largo plazo, como por ejemplo el plan de *release*. También se utilizaron otros elementos de esta metodología que aporton a la fiabilidad del proyecto, como el plan de calidad y el análisis de riesgos.

Por otro lado, la inexperiencia del equipo respecto a las tecnologías contempladas en la solución generaron la necesidad de una forma de trabajo flexible, que permitiera reorganizar las tareas en base a lo aprendido durante el transcurso del proyecto. Se consideró apropiado trabajar utilizando una variación del *framework Scrum* [40] para cumplir con las siguientes consideraciones:

- **Flexibilidad:** el *framework Scrum* propone una metodología adaptable al equipo de desarrollo, permitiendo realizar modificaciones en la planificación en base a los conocimientos obtenidos en la iteración anterior. También permite la flexibilidad sobre el producto desarrollado, pudiéndose generar variaciones en las funcionalidades según las consideraciones del cliente.

- **Transparencia:** se asignó a Pablo Arreche (integrante de Ascentis Uruguay) el rol de *product owner*. Esto involucró al cliente directamente en el proceso de trabajo, donde el mismo fue parte de la evolución del proyecto y se mantenía informado sobre el estado del mismo. Esta colaboración constante entre el cliente y el equipo fue clave para la retroalimentación continua.

También se tuvo en cuenta que Ascentis sugirió trabajar con este *framework*, considerando que es el utilizado dentro de su organización.

No se siguió *Scrum* puro debido a que se añadieron variaciones como por ejemplo la presencia de roles y documentos tradicionales.

6.1.4 Ceremonias

Se adoptaron las siguientes ceremonias de *Scrum*:

Ceremonia	**Utilizada**
Sprint planning meeting	✓
Daily meeting	✓
Sprint review meeting	✓
Sprint retrospective meeting	✓

Tabla 6.2 Ceremonias de Scrum

Debido a que por distintos factores le resultaba difícil al equipo reunirse todos los días, se optó por realizar las *Daily meetings* de distintas formas, siendo reemplazadas por mensajes de WhatsApp, videollamadas o reuniones presenciales cuando se consideraba conveniente. Estas últimas se realizaban cuando el *sprint* no se desarrollaba tal cual había sido planeado y el equipo se reunía para revertir la situación.

Se acordó una duración por *sprint* de dos semanas y una dedicación por integrante de 30 horas por *sprint*. Esto último se consideró flexible, tomando en cuenta las responsabilidades académicas, laborales y personales de cada integrante, pudiendo existir *sprints* con menor o mayor esfuerzo por alguno de los integrantes que debía ser compensado en futuros *sprints*. De todas formas, en la práctica el esfuerzo fue en su mayor medida homogéneo como se indica en la sección 6.4 Gestión de esfuerzo.

Además de las ceremonias de *Scrum*, se añadieron dos reuniones adicionales al proceso de trabajo:

- **Reuniones de tutoría**

  Estas reuniones se realizaron en la Universidad ORT Uruguay junto al tutor del equipo al inicio y final del proyecto bajo una frecuencia semanal, y cada quince días durante el transcurso del mismo. En éstas se mostraba la situación actual del proyecto, los avances realizados y las dificultades encontradas. El tutor aportaba distintas sugerencias sobre cómo superar las dificultades encontradas y su visión del estado del equipo y del proyecto. Además servía como guía para cumplir con los objetivos y expectativas de ORTsf (*ORT software factory*), identificando las fortalezas y debilidades del proyecto. El equipo luego se reunía para reorganizarse tomando en cuenta los aportes realizados por el tutor, permitiendo retomar el camino correcto ante determinados desvíos de gestión. De todas formas, las decisiones siempre fueron tomadas por el equipo con ayuda del tutor, y no a la inversa.

- **Reuniones con el cliente**

  Inicialmente se realizaron reuniones semanales (durante relevamiento de requerimientos) y luego cada dos semanas con Ascentis mediante videoconferencia y en algunas instancias presenciales. Éstas se realizaban al cierre de un *sprint* y previo a la ceremonia de planificación del próximo. Los principales propósitos eran aclarar dudas técnicas y/o de negocio, validar el trabajo realizado y medir la satisfacción del cliente (ver sección 7.3.7 Satisfacción de cliente).

  En las reuniones participaban los tres integrantes del equipo, el *product owner* (recurso ubicado en Uruguay) y un referente tecnológico en Estados Unidos. La comunicación se realizaba en idioma inglés.

  Generalmente la videoconferencia se realizaba en el horario de la tarde (debido a la diferencia horaria con el asistente estadounidense) y el equipo se reunía en la noche para la puesta en común y realización de la *retrospective meeting* junto con la *sprint review meeting* asociadas al *sprint* que se cerraba.

La siguiente ilustración muestra el proceso de *Scrum* utilizado con las adaptaciones empleadas:

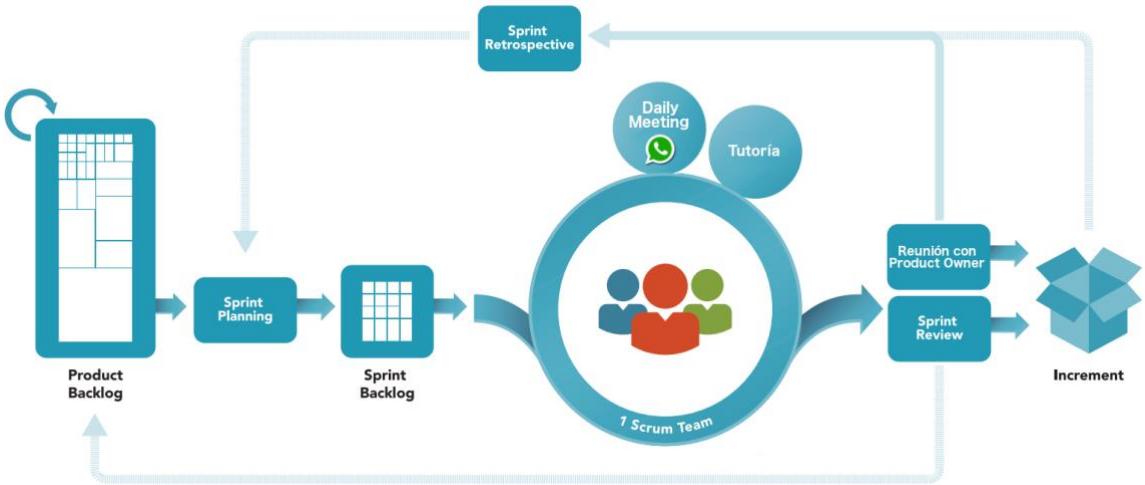


Ilustración 6.2 Metodología de trabajo [41]

6.1.5 Elección de herramientas

En las primeras fases del proceso, el equipo tomó la decisión de llevar a cabo la gestión de las historias de usuario y sus defectos asociados utilizando la herramienta Trello [42]. Esto se debió a que era una herramienta con la cual todos los integrantes se encontraban familiarizados, habiéndose utilizado en proyectos anteriores durante la carrera. Trello también cumple con la propiedad de ser una herramienta gratuita, lo que motivó al equipo a utilizarla.

Ilustración 6.3 Logotipo de Trello

Sin embargo, Trello está basado en la metodología Kanban [43], que comparte algunas similitudes con Scrum como el flujo de las historias "Pendiente/En progreso/Terminado", pero no contempla el concepto de iteraciones. Esto sumado al hecho de que tampoco proporciona un mecanismo simple para la gestión de defectos, llevó al equipo a tomar la decisión de buscar otra herramienta de trabajo.

Luego de investigar las herramientas disponibles basadas en el *framework Scrum*, se tomó la decisión de migrar a la herramienta Jira [44] de Atlassian [45]. Esta herramienta no solo permite la organización del trabajo en *sprints*, sino que también facilita la gestión de defectos (ver capítulo 7.3.6 Gestión de defectos).

Ilustración 6.4 Logotipo de Jira

6.1.6 Planificación de iteraciones

Estimación y priorización de historias de usuario

Para la estimación del esfuerzo necesario en cada historia de usuario, el equipo utilizó la técnica de estimación *Poker Planning* [46]. Esta es una técnica ágil que se basa en el consenso por parte de los integrantes del equipo para medir el tamaño en *story points* para cada *user story* del *product backlog*.

Ilustración 6.5 Instancia de Poker Planning.

Estas historias de usuario fueron priorizadas por el *product owner*, definiendo el orden de las mismas dentro del *product backlog*.

Cronograma de iteraciones

Como se mencionó en la sección 6.1.4. Ceremonias, cada *sprint* tuvo una duración de dos semanas, con una dedicación por integrante de 30 horas por sprint. A continuación se muestra la distribución de los *sprints* en el calendario.

Sprint	Fecha
Sprint 1	01/01/18 - 13/01/18
Sprint 2	15/01/18 - 27/01/18
Sprint 3	29/01/18 - 10/02/18
Carnaval	12/02/18 - 17/02/18
Sprint 4	19/02/18 - 03/03/18
Sprint 5	05/03/18 - 17/03/18
Sprint 6	19/03/18 - 31/03/18
Sprint 7	02/04/18 - 14/04/18

Sprint	Fecha
Sprint 8	16/04/18 - 28/04/18
Sprint 9	30/04/18 - 12/05/18
Sprint 10	14/05/18 - 26/05/18
Sprint 11	28/05/18 - 09/06/18
Sprint 12	11/06/18 - 23/06/18
Sprint 13	25/06/18 - 07/07/18
Sprint 14	09/07/18 - 21/07/18

Tabla 6.3 Cronograma de iteraciones

Vale aclarar algunas consideraciones que se desprenden de la tabla anterior:

- Todos los *sprints* comenzaron los lunes y finalizaron los viernes. Esto fue planificado teniendo en cuenta los siguientes aspectos:

    - Las reuniones con Ascentis fueron llevadas a cabo los viernes. En éstas se validaba el avance con el *product owner* y se recibían sus comentarios considerando lo realizado y evaluando el cumplimiento de los objetivos del *sprint* (se aprovecha esta instancia como ceremonia *sprint review*).

    - Las ceremonias *sprint retrospective* y *sprint planning* se realizaron durante los fines de semana, momento en el que todos los integrantes tenían tiempo disponible para reunirse.

Las retrospectivas consistían de una reunión presencial del equipo donde cada integrante exponía las acciones que se debían continuar haciendo y dejar o comenzar a hacer. Luego se discutía cuales eran las acciones más importantes para el crecimiento del equipo y éstas debían ser llevadas a cabo en el próximo *sprint*. En cada reunión también se evaluaba si las acciones del encuentro anterior habían dado resultados positivos o no.

A modo de ejemplo, en el transcurso del proyecto se dió que en varios *sprints* no se cumplió con todas las historias de usuario planificadas, por lo que en una instancia retrospectiva se decidió como acción a ejecutar el desglose de subtareas para cada historia de usuario a la hora de planificar el trabajo, con el objetivo de generar estimaciones más precisas y de esta forma ajustar lo planificado con lo real.

Por otro lado, las ceremonias de planificación se llevaban a cabo con el objetivo de organizar el trabajo a realizarse en el siguiente *sprint*. Estas generalmente duraban entre 2 y 3 horas aproximadamente, donde se tomaban las historias de usuario ya priorizadas desde el *product backlog* siempre teniendo en cuenta la velocidad del equipo hasta el momento.

- Las reuniones con el tutor fueron programadas para los lunes. Se le informaba que se hizo en el *sprint* anterior, los comentarios del *product owner* y lo que se pretende hacer en el *sprint* entrante. Este último podría ser modificado en base a la reunión con el tutor.

- No se planificó ni ejecutó un *sprint* en el período 12/02/18 - 17/02/18. Esta semana corresponde a la semana de carnaval, en donde los integrantes del equipo se tomaron vacaciones y acordaron no trabajar en el proyecto durante este período.

- Luego de finalizado el *sprint 14* el equipo se enfocó exclusivamente en la documentación. Aquí se unificaron los documentos generados a lo largo de todo el proyecto, llevando a cabo una formalización y extensión de éstos en el presente documento.

6.1.7 Seguimiento y evaluación de iteraciones

Con el objetivo de medir la productividad y el desvío durante las distintas iteraciones, se tomaron métricas que permitieron evaluar objetivamente el estado del proyecto. Éstas sirvieron como base para las distintas autoevaluaciones llevadas a cabo en las ceremonias de retrospectiva.

Incremento del product backlog

La siguiente gráfica muestra el crecimiento del *product backlog* a lo largo del proyecto.

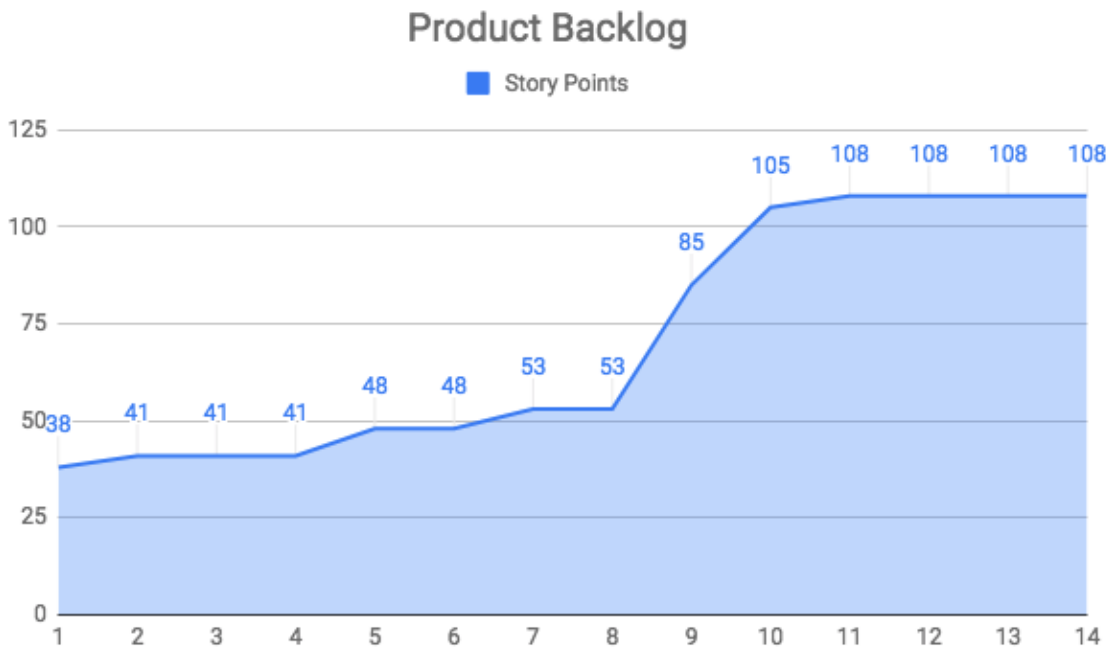


Ilustración 6.6 Evolución del product backlog

Se puede ver que el crecimiento en los primeros *sprints* es muy bajo, consecuencia de que los requerimientos solicitados por Ascentis eran muy estables y no tuvieron grandes cambios. A medida que el equipo avanzaba con el

desarrollo e iba incorporando conocimientos sobre las tecnologías a utilizar, se reconoció la subestimación de algunas historias de usuario, por lo que se estimaron nuevamente en base a la experiencia adquirida.

La gráfica muestra un incremento importante en los *sprints* 9 y 10. En la primera revisión en la Universidad ORT Uruguay, los revisores consideraron que lo solicitado por el cliente representaba un alcance bajo para un proyecto de duración anual integrado por tres estudiantes avanzados de Ingeniería de Sistemas. Esta consideración fue acertada ya que lo solicitado por el cliente fue en su mayoría implementado y presentado al final del *sprint 8* en la primera versión de Feedback Assistant (ver sección 6.3.2 Plan de *releases*). Como se explica en el capítulo 3 Ingeniería de requerimientos, para revertir esta situación el equipo presentó al cliente una lista de funcionalidades adicionales luego de entregado el MVP que agregan valor al producto, los cuales fueron discutidos, validados y priorizados junto con el cliente. Todas las nuevas historias de usuarios fueron incorporadas en el *product backlog* a partir del *sprint 9*.

Evolución del proceso

La siguiente gráfica muestra la relación entre los *story points* planificados y realizados de cada *sprint*, así como la velocidad promedio del equipo a lo largo de las iteraciones.

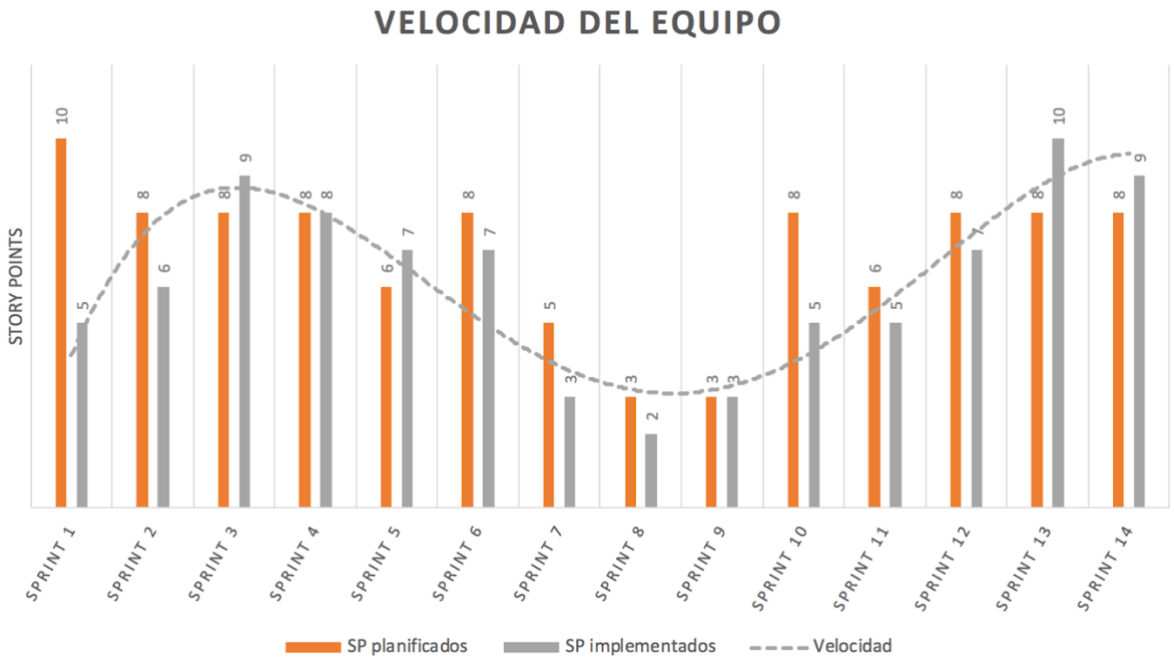


Ilustración 6.7 Comparación story points planificados vs. realizados

De esta comparación se pueden destacar varios puntos:

- En los primeros *sprints*, la cantidad de *story points* implementados fueron siempre menores a los planificados. Esto se debió a que las planificaciones no eran tan certeras respecto al trabajo real, situación que se fue ajustando paulatinamente durante los próximos *sprints*. En estas primeras fases se hizo foco en la integración con Google G-Suite, integración que terminó siendo mucho más compleja que la esperada (ver capítulo 5. Análisis tecnológico).

- En los *sprints 4, 5* y *6* se puede puede observar una mayor precisión en las estimaciones, pasando de un desvío sobre lo planificado del 30% en los primeros tres *sprints* a uno del 9% en estas iteraciones. Una de las decisiones que ayudaron a revertir esta situación fue la de desglosar las

historias de usuario en subtareas para generar estimaciones más precisas (acción que surge luego de una retrospectiva).

- Se observa un descenso de los *story points* planificados en los *sprints* 7 y 8. Esto se debió a que se concentró el esfuerzo en ejecutar una prueba de regresión (ver sección 7.3.5.4 Pruebas de Regresión) para identificar y resolver defectos y realizar mejoras en el sistema para presentar un MVP estable a fines de Abril (ver sección 6.3 Gestión de alcance).

- La baja planificación del *sprint* 9 corresponde principalmente al refactoreo de la arquitectura, en donde se desacoplaron responsabilidades para facilitar la futura integración con nuevos proveedores de procesamiento de lenguaje natural. También se resolvieron defectos que no pudieron ser corregidos en el *sprint* anterior.

- En el *sprint 10* el equipo se comprometió en la integración con IBM Watson para el análisis de lenguaje natural, la cual fue más compleja de lo esperado. En la retrospectiva asociada a este *sprint* surge como acción la ejecución de *spikes* (si fuese necesario) al final de cada *sprint* para mejorar la planificación del próximo. Cabe destacar que éstos se venían realizando al inicio de cada *sprint*. En esta iteración también se elaboró el informe de avance del proyecto para la Universidad ORT Uruguay y el equipo decidió comprometerse con menos trabajo.

- En los últimos tres *sprints* el equipo logra cumplir con lo planificado y en particular en el *sprint 13* supera lo planeado. Esto coincide con el aumento del esfuerzo en las etapas finales del proyecto (ver sección 6.4 Gestión de esfuerzo).

Los mayores desvíos se atribuyeron a que el equipo tendió a subestimar las historias de usuario, estimando incorrectamente historias que llevaron más tiempo de lo planeado. Por ejemplo, las curvas de aprendizaje para la integración con G-Suite e IBM Watson fueron más pronunciadas de lo que se esperaba, debiéndose dedicar más tiempo a lo planeado en actividades de investigación.

Otro artefacto del *framework Scrum* utilizado fue el *burndown-chart* [47], el cual permite ver a simple vista la evolución de lo realizado a lo largo del proyecto, así como también el trabajo que queda por hacer. Este artefacto permite verificar si el equipo está avanzando correctamente, y en caso de existir un desvío, muestra la relación que tiene respecto a lo planificado.

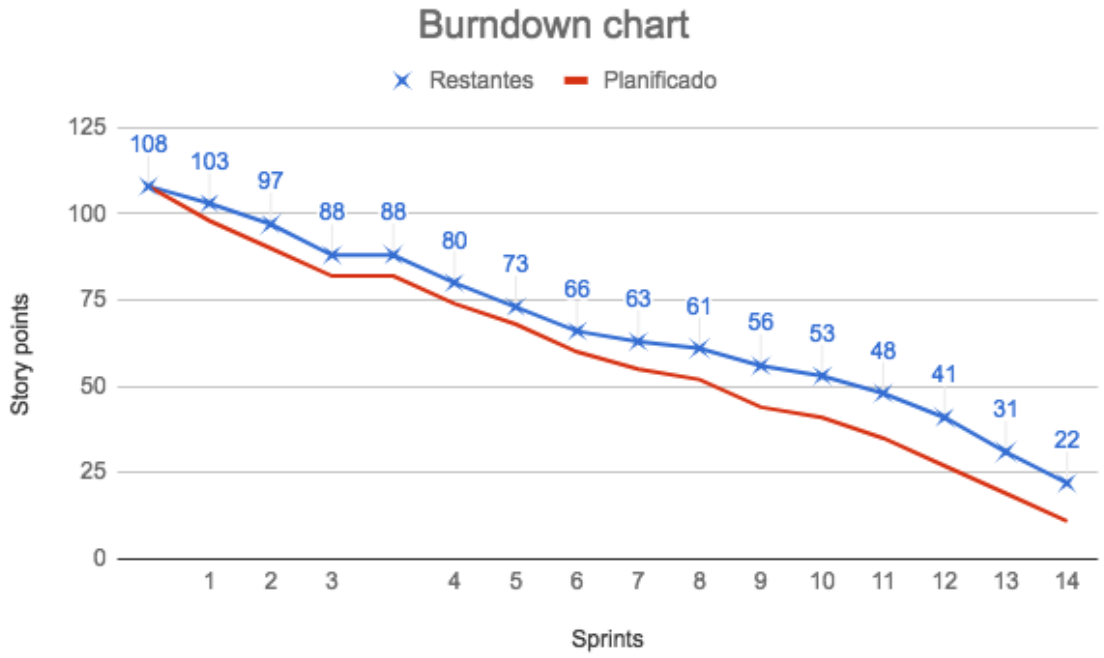


Ilustración 6.8 Evolución del proyecto / Burndown chart

Como se puede ver, los desvíos más pronunciados se dieron en los *sprints* 1 y 2, correspondientes a la integración con G-Suite, y en el *sprint* 9, correspondiente a la integración con IBM Watson. Se puede ver también que quedan pendientes por hacer 22 *story points* los cuales corresponden a las funcionalidades descritas en la sección 9.5 Próximos Pasos.

El *product backlog* tiene incluído una considerable cantidad de *stretch goals*, estos son requerimientos adicionales que se implementarían en caso de completar los imprescindibles antes de tiempo, y no formaban parte del objetivo comprometido con el cliente. Todas las historias de usuario no implementadas son un subconjunto de *stretch goals*. Este subconjunto contiene las historias de usuario con menor prioridad.

6.2 Hitos del proyecto

En la presente sección se detallan los principales sucesos ocurridos a lo largo del proyecto. El objetivo es ubicar en el tiempo los distintos eventos de importancia que involucraron al equipo tanto con la Universidad ORT Uruguay como con Ascentis.

En la siguiente ilustración se presenta una línea de tiempo donde se pueden apreciar los distintos hitos del proyecto con la fecha exacta y descripción del evento.

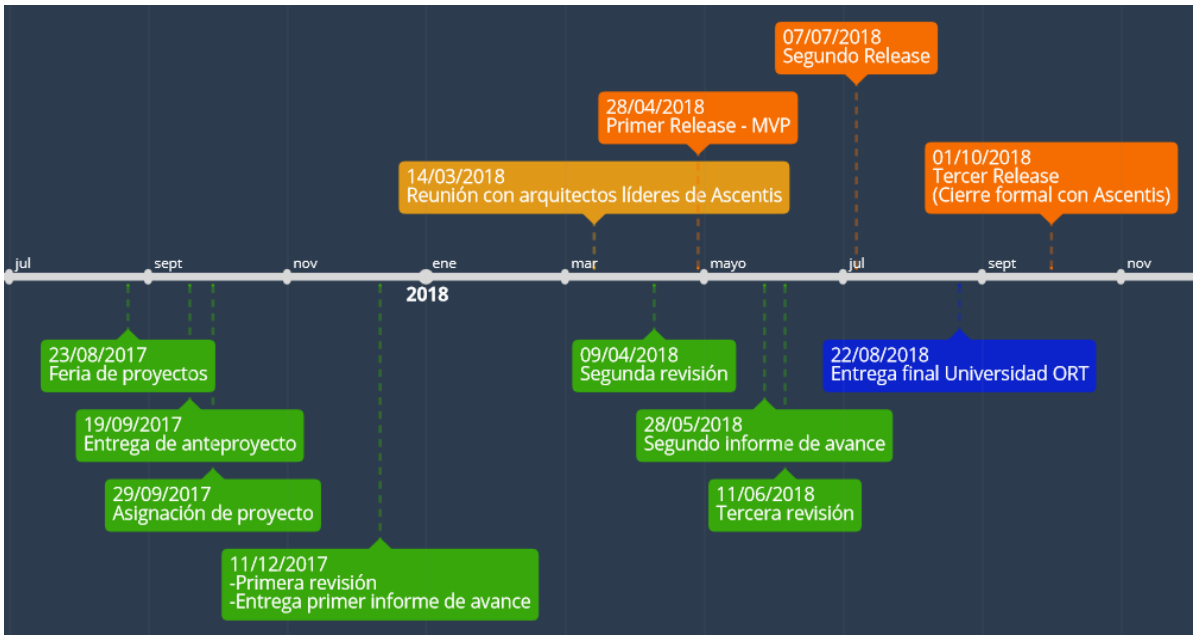


Ilustración 6.9 Ubicación temporal de los hitos del proyecto

A continuación se expone para cada hito una breve descripción y la importancia del mismo:

- 23 de Agosto 2017 - **Feria de proyectos**

  Esta instancia se realizó en la Universidad ORT Uruguay donde se introdujo al equipo en los distintos proyectos disponibles. Aquí se dió el primer encuentro con Ascentis, donde éste comentó sobre los dos proyectos propuestos a la Universidad ORT Uruguay. Feedback Assistant fue el que más llamó la atención a los integrantes del equipo debido al alto grado de innovación y uso de tecnologías relacionados a la inteligencia artificial.

- 19 de Setiembre 2017 - **Entrega de anteproyecto**

  Luego del primer encuentro realizado en la feria de proyectos el equipo se reunió con Ascentis en las nuevas oficinas que estaban abriendo en Uruguay con el fin de obtener más información acerca del proyecto. Se aprovecha esta instancia para argumentar al cliente los motivos por los cuales el equipo se cree capaz de llevar a cabo el proyecto junto con las motivaciones que impulsaron a su elección. Se presentó de forma escrita una carta para que el representante de Ascentis Uruguay firme y luego fue entregada a la Universidad ORT Uruguay para manifestar el interés por parte de la empresa en trabajar con los integrantes del equipo.

- 29 de Setiembre 2017 - **Asignación de proyecto**

  A fines de setiembre del 2017 se lleva a cabo la asignación formal de Feedback Assistant al equipo. A partir de este momento se comienzan a definir los distintos objetivos detallados en la sección 1.2 Objetivos. Con el soporte del tutor se comienza a trabajar en las primeras fases del proyecto.

- 11 de Diciembre 2017 - **Primera revisión e informe de avance**

  La primera revisión fue de suma utilidad para identificar desvíos que se dieron en el primer trimestre de proyecto. Para profundizar en el contenido de esta instancia ver Anexo 11.5 Revisiones ORTsf.

  En esta misma fecha se llevó a cabo la entrega del primer informe de avance el cual consistió de entregar a Universidad ORT Uruguay la presentación expuesta en la primera revisión.

- 14 de Marzo 2018 - **Reunión con arquitectos líderes de Ascentis**

  Se aprovechó la visita de los referentes tecnológicos de la empresa a Uruguay para validar la arquitectura final (ver sección 4.5 Validación de arquitectura) y hacer una demo del trabajo realizado hasta el momento. Esta instancia permitió también al equipo medir los niveles de satisfacción por parte del cliente.

- 09 de Abril 2018 - **Segunda revisión**

  En esta instancia surgieron comentarios muy interesantes y constructivos por parte del revisor asociados tanto al producto como al proceso. En este momento el equipo se encontraba a tres semanas del primer *release* asociado al MVP. Se aprovecha este evento para realizar una demo luego de exponer las diapositivas. El revisor presenta sugerencias sobre cómo mostrar el producto debido a la complejidad que implica mostrar la funcionalidad de un sistema sin interfaz de usuario. Para conocer más acerca de esta instancia ver Anexo 11.5 Revisiones ORTsf.

- 28 de Abril 2018 - **Primer *release* (MVP)**

  Esta corresponde a la primer entrega formal al cliente conteniendo las funcionalidades del MVP (ver sección 6.3.2. Plan de *releases*). Se hizo entrega del código fuente y se realizó una demo al cliente tomando en cuenta las sugerencias de la segunda revisión respecto a cómo mostrar el producto. Se midieron nuevamente los niveles de satisfacción y se logró apreciar un alto grado de interés por parte de Ascentis durante el evento.

- 28 de Mayo 2018 - **Segundo informe de avance**

  Este evento corresponde a la entrega formal de un documento a la Universidad ORT Uruguay con el objetivo de informar el estado del proyecto mediante datos sobre la situación actual. Se indicaron también los resultados de las actividades realizadas así como también el pronóstico hasta el final del proyecto.

- 11 de Junio 2018 - **Tercera revisión**

  Se llega a la última instancia de revisión con una sólida presentación considerando las sugerencias propuestas en las dos primeras revisiones. Igualmente surgen comentarios constructivos como por ejemplo hacer mayor énfasis en el aseguramiento de la calidad volcado al proceso y no tanto al producto. Luego de realizada la demo surge nuevamente el problema de cómo mostrar el producto considerando la ausencia de interfaz gráfica. Aquí la revisora sugiere aprovechar la

funcionalidad de reportes para mostrar los resultados de los flujos internos de la aplicación.

- 07 de Julio 2018 - **Segundo *release***

  El equipo hace entrega del código fuente asociado al segundo *release* del producto. Este contiene algunas de las funcionalidades que el equipo presentó al cliente como valor adicional al MVP. Para obtener mayor información sobre el contenido de esta entrega ver sección 6.3.2 Plan de *releases*. Fue importante la medición de la satisfacción por parte del cliente debido a que el próximo *release* se llevaría a cabo luego de entregar el presente documento. Esta medición logró manifestar la elevada satisfacción de Ascentis respecto al trabajo realizado en esta entrega, sobre todo por el cumplimiento con la funcionalidad asociada al *bot* de llamadas telefónicas.

- 22 de Agosto 2018 - **Entrega final Universidad ORT Uruguay**

  Este hito refiere a la entrega formal del presente documento en Universidad ORT Uruguay.

- 01 de Octubre 2018 - **Tercer *release***

  Este evento refiere a la última entrega formal del producto hacia Ascentis. En esta instancia se dará una clausura formal del proyecto haciendo entrega tanto del código fuente como de documentos de instalación, configuración, arquitectura y los casos de prueba funcionales. También se llevará a cabo una demo final mostrando al cliente todas las funcionalidades agregadas. Se fija para esta fecha debido a que se decide hacer énfasis en la creación del presente documento y luego en la preparación de la defensa final.

6.3 Gestión de alcance

Esta sección explica cómo se gestionó el alcance del proyecto, identificando qué se incluye y qué no en cada una de las entregas formales con el cliente.

6.3.1 Marco de trabajo

El alcance del producto fue dividido en tres grandes hitos, cada uno identificado con un número de versión. Se consideró que gestionar el alcance era sumamente importante ya que permite llevar un control sobre el trabajo que se debía hacer, definiéndose metas a corto y mediano plazo y tomándose como base para la planificación de los *sprints*. Asimismo, la gestión de alcance permitió darle una idea al cliente acerca de qué y cuándo iba a recibirlo, otorgándole una visión a futuro del proyecto.

Al final de cada *release* se realiza una entrega formal al cliente conteniendo el producto de *software* construido. Asimismo se presenta una demo mostrando las funcionalidades del sistema que se agregaron en esa entrega.

En la entrega final del producto, además del *software* funcionando se entregará un documento para la correcta instalación y configuración del sistema (ver Anexo 11.11 Manual de instalación y configuración), la planilla de casos de prueba funcionales así como también el documento de arquitectura.

6.3.2 Plan de *releases*

Hito	Fecha	Funcionalidades
V1	28/04/2018 *feature-driven*	• RF01 • RF02 • RF03 • RF04 • RF05 • RF06 • RF08
V2	07/07/2018 *date-driven*	• RF07 • RF09 • RF10 • RF11 • RF12 • RF13 • RF14 • RF15
V3	01/10/2018 *date-driven*	• RF16 • RF17 • RF18 • RF19

Tabla 6.4 Plan de *releases*

Para la primera versión se decidió llevar a cabo el desarrollo del MVP, con el fin de validar el buen desempeño de las funcionalidades más imprescindibles.

En las siguientes versiones del producto se agregaron funcionalidades negociadas con el cliente, que agreguen valor con el fin de complementar las funcionalidades mínimas del producto.

Para la entrega del MVP se acordó trabajar en modalidad *feature-driven,* en donde el alcance estaba medido en términos de qué funcionalidades estaban incluidas, generándose un compromiso de cubrir la totalidad de las funcionalidades imprescindibles requeridas por el cliente. Sin embargo, para las siguientes dos entregas se acordó cambiar la metodología a *date-drive*, donde éstas contienen todas las funcionalidades para la fecha pactada.

Al igual que el *product backlog*, el plan de versiones fue dinámico, cambiando a lo largo del proyecto a medida que se iban adquiriendo conocimientos sobre las funcionalidades pendientes. Un ejemplo de esto son los requerimientos RF12 y

RF13 correspondientes a la solicitud y recepción de *feedback* mediante llamada telefónica, los cuales inicialmente fueron asignados a la tercer entrega, pero fueron re-planificados para la segunda debido a que el equipo completó los requerimientos del segundo *release* antes de tiempo, lo que dio la oportunidad de incluir otras funcionalidades no planificadas.

6.4 Gestión de esfuerzo

Esta sección tiene como objetivo mostrar la dedicación del equipo a lo largo del proyecto. Esta gestión permitió llevar un seguimiento sobre la inversión de horas de trabajo de cada integrante del equipo y determinar si se estaba cumpliendo con lo comprometido.

6.4.1 Elección de herramientas

Se decidió utilizar la herramienta Toggl [48] para el registro del esfuerzo. Esta decisión se debió a que uno de los integrantes contaba con experiencia previa sumamente positiva de la herramienta. Toggl ofrece no solo registrar el tiempo trabajado, sino también agrupar este esfuerzo en distintas áreas, permitiendo determinar qué tareas fueron más demandantes en dedicación. Otras fortalezas de la herramienta son su facilidad de uso, cuenta con una aplicación móvil y es gratuita.

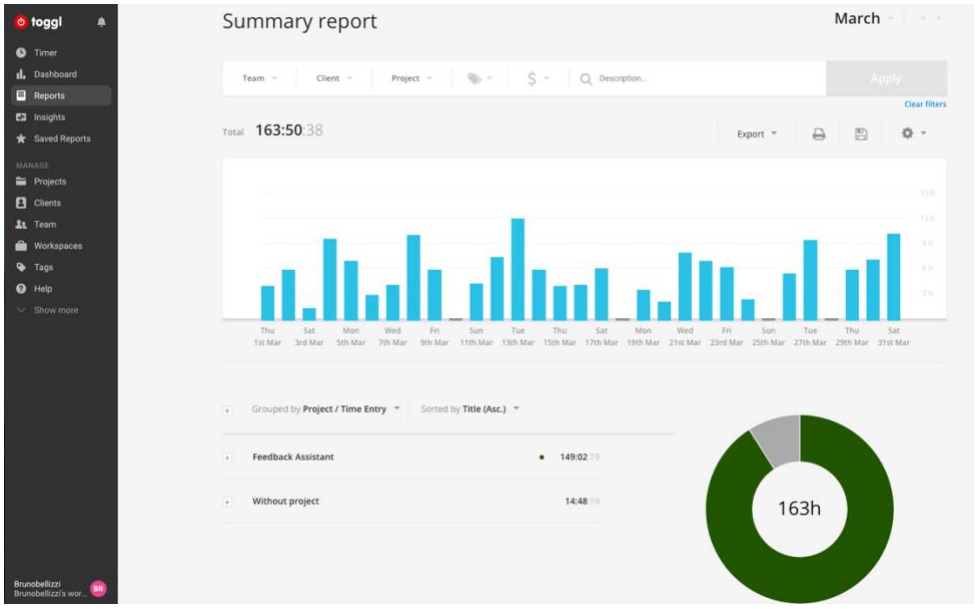


Ilustración 6.10 Herramienta Toggl

6.4.2Análisis de resultados

La siguiente gráfica muestra la distribución del esfuerzo del equipo entre las distintas áreas:

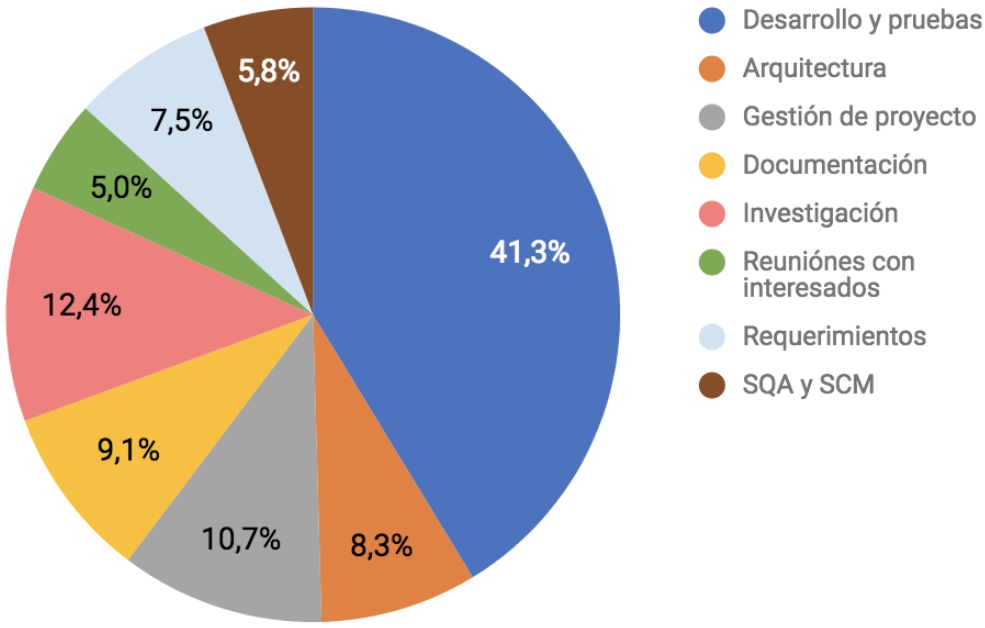


Ilustración 6.11 Distribución de esfuerzo por área

De esta distribución se pueden observar algunos aspectos que caracterizaron al proyecto:

- El mayor esfuerzo fue dedicado al desarrollo de la solución de *software*. Esto indica que la metodología de trabajo permitió dedicarle la mayor cantidad de tiempo al desarrollo del producto. El esfuerzo empleado en la corrección de defectos se encuentra también dentro del área Desarrollo y pruebas.

- La investigación de tecnologías representó un porcentaje alto comparado con el tiempo de desarrollo. Esto sucedió debido a que la solución propuesta implicó la capacitación sobre un nuevo *framework* de desarrollo (*Spring Framework*) y la integración con múltiples servicios externos. La mayoría de las historias de usuario implementadas requirieron una investigación previa sobre alguno de los servicios

externos, provocando que la investigación de tecnologías sea una actividad constante a lo largo del proyecto. A pesar de la alta dedicación involucrada, esto produjo grandes beneficios ya que permitió agregar a la solución funcionalidades innovadoras con alto impacto positivo.

- El enfoque arquitectónico que utilizó el equipo llevó a dedicarle un tiempo considerable al diseño de la arquitectura. La misma sufrió reiterados cambios durante el proyecto, como por ejemplo los sugeridos en la validación de la arquitectura con Ascentis (ver sección 4.5 Validación de arquitectura) .

- La alta dedicación a la documentación fue motivada por generar un documento final con altos niveles de calidad respecto a las expectativas de la Universidad ORT Uruguay. Esto corresponde al objetivo académico descrito en la sección 1.2.2 Objetivos académicos, referente a la aprobación del proyecto como requisito final de carrera.

Otra métrica útil para evaluar el esfuerzo del equipo es la comparación de las horas dedicadas vs planificadas por *sprint*. Como se mencionó anteriormente, el objetivo acordado eran 30hs de trabajo por integrante en cada *sprint*.

La siguiente gráfica muestra el esfuerzo real en cantidad de horas dedicadas por *sprint*:

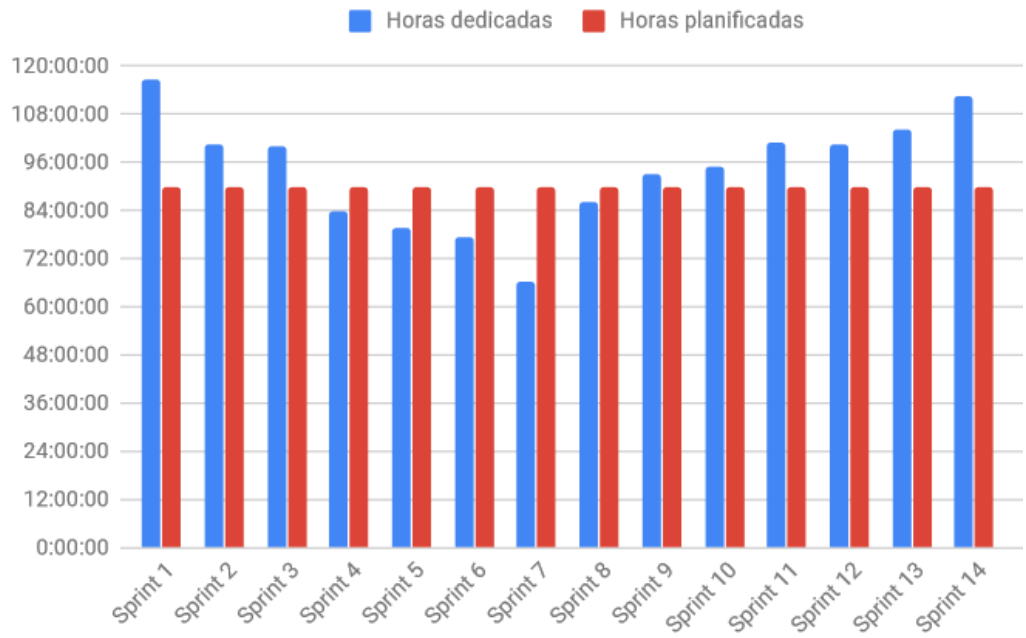


Ilustración 6.12 Distribución de esfuerzo por sprint

Esta gráfica corresponde a los datos registrados en Toggl. Cada *sprint* contiene las horas dedicadas en las distintas áreas del proyecto (desarrollo, gestión, documentación, etc) así como también las horas planificadas.

Se dedicaron 94 horas en promedio por *sprint*, lo que significa que el equipo trabajó por encima de lo acordado. Si se comparan las horas dedicadas respecto a las planificadas se puede observar que en total el equipo dedicó un 4,5% por encima de lo planificado, ocasionado principalmente porque las estimaciones realizadas al inicio del proyecto no fueron tan precisas y exigieron una mayor dedicación, así como también por las actividades de investigación que demandaron un mayor esfuerzo de lo esperado.

El equipo realizó un esfuerzo total de 1560 horas reales de trabajo a lo largo de todo el proyecto incluyendo las primeras fases donde no se contaba con *sprints*.

A continuación se muestra la ilustración referente a la distribución de la dedicación por cada integrante en el proyecto.

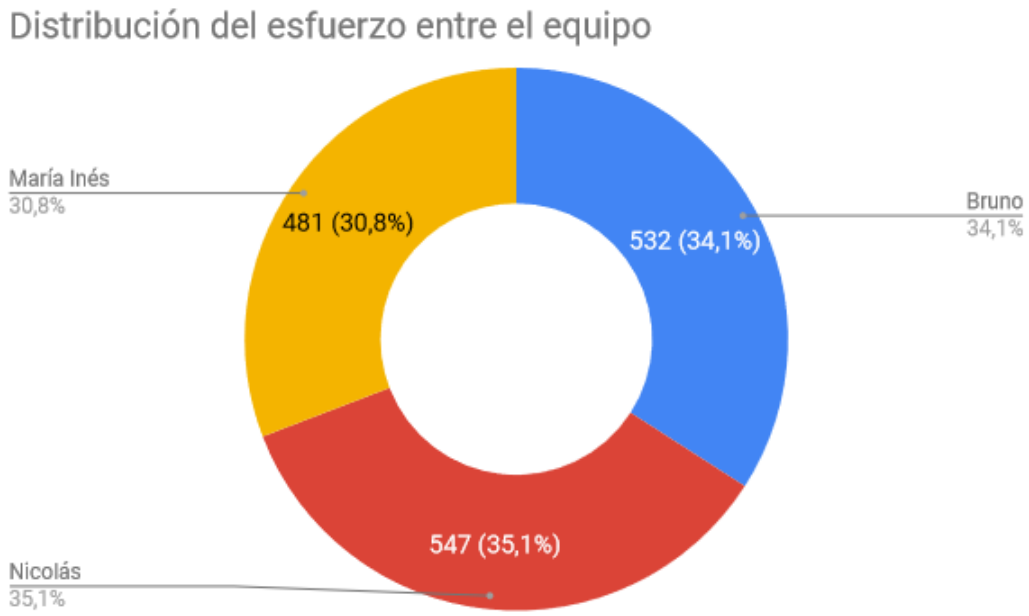


Ilustración 6.13 Distribución del esfuerzo entre el equipo

6.5 Gestión de riesgos

En esta sección se detalla cómo se llevó a cabo la gestión de riesgos. Debido a que se trata de un proyecto de gran porte, se consideró fundamental realizar distintas actividades para favorecer el cumplimiento de los objetivos.

Un riesgo es un evento puntual, que si ocurre, genera un impacto negativo respecto a las metas establecidas. Para realizar esta gestión se identificaron los riesgos, se llevó a cabo un estudio cualitativo de los mismos, se definieron planes de respuesta y contingencia para cada uno y se realizó un seguimiento a lo largo del proyecto.

6.5.1 Identificación y categorización

Desde un inicio y a medida que el equipo fue entendiendo con más profundidad el problema, fueron surgiendo varios factores que se consideraron como una posible amenaza.

Luego de identificar los riesgos, estos fueron categorizados en tres grupos: tecnológicos, del proyecto y de gestión. A continuación se presentan los riesgos identificados:

Riesgos tecnológicos

ID	Riesgo
R01	Servicio externo deja de existir
R02	Cambios significativos en la solución debido a la modificación de la API expuesta por algún proveedor.
R03	Desconocimiento de las tecnologías a utilizar
R04	Cliente no provee la API necesaria para la integración
R05	El análisis del sentimiento del feedback es impreciso

Tabla 6.5 Riesgos tecnológicos

Riesgos del proyecto

ID	Riesgo
R06	El cliente abandona el proyecto
R07	Cambios bruscos en los requerimientos
R08	No se interiorizan los conceptos de negocio involucrados
R09	Uso del idioma inglés complejiza la comunicación con Ascentis.

Tabla 6.6 Riesgos del proyecto

Riesgos de equipo y gestión

ID	Riesgo
R10	Integrante/s no cumplen con las horas acordadas de trabajo
R11	Problemas de gestión debido a la inexperiencia en proyectos de tal alcance
R12	Integrante decide abandonar el equipo
R13	Retraso en entregas debido a estimaciones incorrecta

Tabla 6.7 Riesgos de equipo y gestión

6.5.2 Análisis cualitativo

Para poder medir los riesgos identificados y evaluar su posible impacto, se realizó un análisis cualitativo de los mismos.

A cada riesgo se le asignó una probabilidad que hace referencia a la posibilidad de que el riesgo se materialice.

Probabilidad	Descripción
0	No probable
0.2	Poco probable
0.4	Probable
0.6	Muy probable
0.8	Altamente probable
1	Se convierte en problema

Tabla 6.8 Probabilidad de riesgos

Luego se le asignó un impacto que hace referencia al efecto negativo sobre el proyecto en caso de materializarse. Éste se midió utilizando una escala del 1 al 5:

Impacto	Descripción
1	Marginal
2	Poca importancia
3	Importante
4	Crítico
5	Catastrófico

Tabla 6.9 Impacto de riesgos

Por último, se calculó la magnitud del riesgo multiplicando el impacto por la probabilidad. Este valor permite priorizar los riesgos y saber a cuáles darle mayor importancia.

	1	2	3	4	5
0	0	0	0	0	0
0.2	0.2	0.4	0.6	0.8	1
0.4	0.4	0.8	1.2	1.6	2
0.6	0.6	1.2	1.8	2.4	3
0.8	0.8	1.6	2.4	3.2	4
1	1	2	3	4	5

Ilustración 6.14 Matriz de magnitud

6.5.3 Planes de respuesta y contingencia

Los planes de respuesta son estrategias que realizó el equipo para disminuir la probabilidad de ocurrencia y/o impacto de cada riesgo, mientras que los planes de contingencia buscan reducir el impacto de los riesgos una vez materializados.

La aceptación de un riesgo es una forma de respuesta pasiva, en la cual se busca responder a los riesgos en la medida que aparezcan. Se aceptaron riesgos en los casos donde la probabilidad de ocurrencia fuera muy baja, o el costo de aplicar un plan de respuesta fuese mayor al de corregir el problema ocasionado.

La siguiente tabla muestra los planes de respuesta y contingencia generados para los riesgos comentados:

Riesgo	**Descripción**	**Plan de respuesta**	**Plan de contingencia**
R01	Servicio externo deja de existir	Aceptar riesgo	Migrar a otro servicio
R02	Cambios significativos en la solución debido a la modificación de la API expuesta por algún proveedor.	Diseñar la arquitectura de manera tal que el impacto del cambio sobre un sector del sistema no afecte al resto de la solución	Incrementar esfuerzo para implementar los cambios
R03	Desconocimiento de las tecnologías a utilizar	- Realizar pruebas de concepto en etapas tempranas del desarrollo - Consultar con expertos de Ascentis y/o Universidad ORT Uruguay	Cambiar de tecnología

R04	Cliente no provee la API necesaria para la integración	Motivar al cliente con las demos de avance para lograr consolidar la totalidad del proyecto con éxito (esto involucra el esfuerzo de Ascentis en proveer la API de TMS)	Mockear las dependencias de terceros para no depender de los mismos
R05	El análisis del sentimiento del feedback es impreciso	Aceptar riesgo	Integrarse con múltiples servicios y promediar los resultados
R06	El cliente abandona el proyecto	- Proponer funcionalidades atractivas adicionales al MVP para atraer su interés. - Mantenerlo informado permanentemente mostrando los avances del proyecto	- Acudir a ORTsf y al tutor para que indiquen los próximos pasos. - Continuar con el proyecto sin la participación del cliente
R07	Cambios importantes en los requerimientos	- Reuniones quincenales con Ascentis para mantener comunicación continua y que no hayan sorpresas - Realizar una validación formal de requerimientos y solicitar aprobación escrita por el cliente.	- Si fuera necesario se incrementa el esfuerzo - Evaluar nuevamente el alcance con el cliente - Si fuera necesario se rediseña la solución
R08	No se interiorizan los conceptos de negocio involucrados	- Aprovechar las reuniones semanales al inicio del proyecto y luego quincenales para despejar dudas de negocio.	- Solicitar al cliente sesiones de transferencia de conocimiento en cuanto al negocio - Contactar con otros expertos en RRHH
R09	Uso del idioma inglés complejiza la comunicación con Ascentis.	- Se complementan los integrantes para la comunicación con Ascentis.	Solicitar a Ascentis que las reuniones se realicen en español con los recursos humanos provistos de Uruguay
R10	Integrante/s no cumplen con las horas acordadas de trabajo	- Definir compromisos individuales a corto plazo - Comprometerse a compensar las horas perdidas en otro *sprint*	Repartir las tareas del integrante indispuesto entre los demás integrantes
R11	Problemas de gestión debido a la inexperiencia en proyectos de tal alcance	- Aplicar los conocimientos de Ingeniería de Software - Realizar una investigación sobre gestión de proyectos en base a los documentos provistos por ORTsf - Realizar un cronograma con las actividades de gestión de proyectos a desarrollar y guiarse por el mismo	Realizar nuevamente actividades de planificación teniendo en cuenta los problemas sufridos previamente.

		- Aprovechar las tutorías lo máximo posible	
R12	Integrante decide abandonar el equipo	Aprovechar las ceremonias de retrospectiva para hablar de cómo se sienten los integrantes del equipo. En caso de incomodidad por parte de alguno buscar alternativas para revertir esta situación.	Aumentar el compromiso de los restantes miembros del equipo para cumplir con lo planificado.
R13	Retraso en entregas debido a estimaciones incorrectas	- Manejar historias de usuario granulares - Dedicar el tiempo que sea necesario en la ceremonia de planificación para generar un buen entendimiento de las tareas - Realizar investigaciones previo a la ceremonia de planificación	- Aumentar el esfuerzo para compensar la falla de estimación - Solicitar cambio de fecha para la entrega con el cliente

Tabla 6.10 Planes de respuesta y contingencia

6.5.4 Evolución y seguimiento

Debido a que la magnitud de cada riesgo varió en el tiempo, se acordó realizar una reevaluación de los mismos cada cuatro *sprints* (dos meses), considerándose éste un tiempo razonable respecto a la posible variación de éstos.

Riesgos Tecnológicos

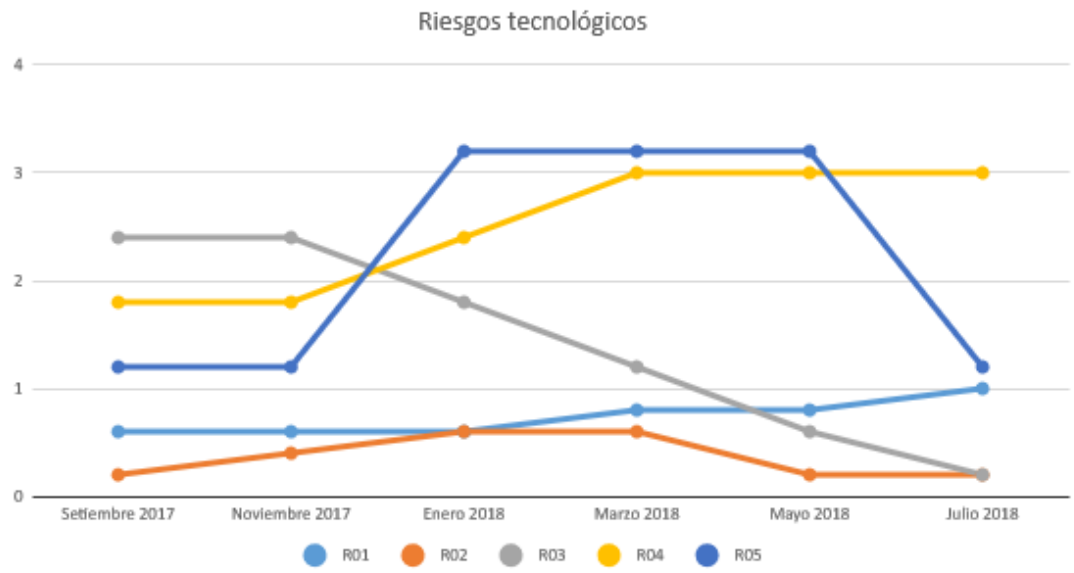


Ilustración 6.15 Riesgos tecnológicos

El **R01** referente a que algún proveedor deje de brindar el servicio consumido es significativamente bajo en probabilidad debido a que se trabaja con proveedores como Google, IBM y Microsoft, los cuales cuentan con productos extremadamente estables. Como se puede observar en la ilustración con el paso del tiempo esto genera un impacto mayor, ya que a medida que se avanza en el proyecto la pérdida de un servicio al cual ya se integró es una gran amenaza.

El riesgo **R02** asociado a una modificación significativa en la solución provocada por el cambio en alguna API provista por algún proveedor, al igual que el **R01** cuenta con una muy baja probabilidad de ocurrencia. Igualmente el impacto aumenta desde el último trimestre de 2017 hasta enero del 2018 manteniéndose constante hasta la evaluación de riesgos realizada en mayo del 2018, donde aquí la magnitud desciende ya que se contaba con una arquitectura consolidada,

validada por Ascentis y con gran foco en la modificabilidad (ver capítulo 4. Descripción de la solución de software) por lo que un cambio en una integración no sería una tarea compleja.

El riesgo **R03** referente al desconocimiento de las tecnologías a utilizar, fue disminuyendo a medida que el equipo se familiarizaba con las mismas, lo cual era esperable. Para mitigar este riesgo en etapas tempranas del proyecto fue de suma importancia la prueba de concepto realizada.

Un riesgo de suma importancia debido a su materialización en Marzo del 2018 fue el **R04** referente a que el cliente no provea la API necesaria para la integración con TMS. Esto se debe a que el cliente brindó la especificación de la API pero nunca la implementó y el equipo ya se encontraba en situación de desarrollar la integración. Siguiendo el plan de contingencia, el equipo decide simular el comportamiento de TMS mediante un *mock* para poder realizar las pruebas necesarias y no interrumpir con el desarrollo. Para obtener mayor información acerca de la especificación provista por Ascentis ver Anexo 11.13 Especificación de la API asociada a TMS.

Otro riesgo materializado fue el **R05** relacionado a un impreciso análisis del sentimiento del *feedback*. En enero del 2018 al realizar la prueba de concepto se identificó que algunas respuestas del servicio de Google Natural Language API no reflejaban el sentimiento real del *feedback*. Si bien el equipo y el *product owner* concordaron en la importancia de conseguir resultados precisos en el análisis, se tomó la decisión en conjunto de finalizar con el desarrollo del MVP previo a la ejecución del plan de contingencia asociado a este riesgo. Luego de entregado el MVP a Ascentis, se ejecuta el plan de contingencia y en el seguimiento de riesgos llevado a cabo en Julio del 2018 se aprecia cómo desciende la magnitud del mismo. En el capítulo 5 Principales desafíos tecnológicos se profundiza sobre la ejecución de este plan de contingencia.

Riesgos de proyecto

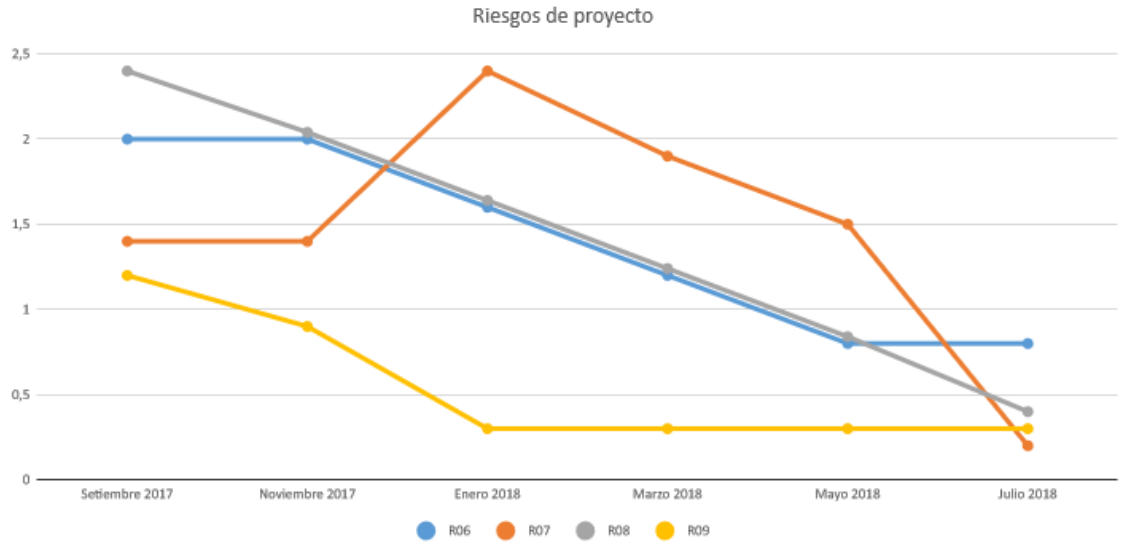


Ilustración 6.16 Riesgos de proyecto

El riesgo **R06** asociado al abandono del proyecto por parte del cliente fue disminuyendo en el tiempo. Si bien el impacto se mantuvo constantemente alto, la probabilidad de ocurrencia fue cada vez menor. Esto se debió a que el cliente cada vez mostró mayor interés en el proyecto.

En cuanto al riesgo **R07** relacionado a cambios en los requerimientos que lleven a un impacto negativo sobre la planificación, se puede observar un aumento notorio en Enero del 2018. Esto ocurre debido a que los resultados de la prueba de concepto introdujeron cambios no previstos, cambiando por ejemplo los requerimientos asociados al acceso de calendarios repercutiendo de esta forma sobre la planificación. Luego de esta etapa los requerimientos no tendieron a cambiar, para suerte del equipo la interacción constante con Ascentis fue de gran ayuda y éste no presentó significativos cambios en los requerimientos.

El riesgo asociado a la no interiorización de conceptos de negocio involucrados en el proyecto (**R08**) comienza con una magnitud elevada debido al desconocimiento por parte de los integrantes del equipo en esta área. Gracias a la interacción constante con Ascentis y el conocimiento puesto a disposición por su parte, el equipo logra comprender los objetivos del cliente en cuanto a negocio y las dificultades que se presentan, por lo que este riesgo desciende continuamente a lo largo del tiempo.

El riesgo **R09** hace referencia a la dificultad en la comunicación con el cliente debido a la interacción en inglés. En los primeros meses este fue un gran desafío para el equipo ya que en las videoconferencias se perdía información importante debido a que el cliente hablaba muy rápido. Pero este riesgo disminuyó rápidamente debido a que estas reuniones se daban con bastante frecuencia y ésto permitió que el equipo gane experiencia y mayor confianza. Además una habilidad del equipo que vale destacar fue la complementación que se dió entre los integrantes ya que algunos tienen mayor experiencia con este idioma que otros.

Riesgos de equipo y gestión

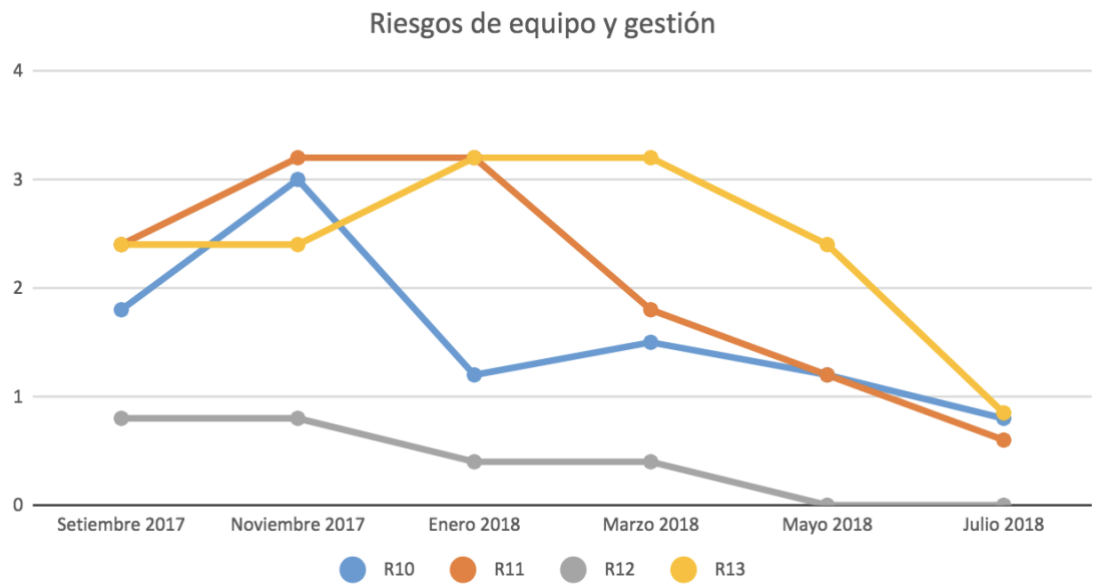


Ilustración 6.17 Riesgos de equipo y gestión

El riesgo **R10** refiere a que los miembros del equipo no cumplan con las horas establecidas de trabajo. Éste riesgo se materializó en Noviembre del 2017 debido a que otros compromisos académicos desviaron al equipo del esfuerzo esperado. Luego de la primer revisión en Diciembre del 2017 el equipo aumentó notoriamente el esfuerzo de manera de recuperar el tiempo perdido. En la evaluación de riesgos correspondiente al mes de Marzo del 2018 se observa un aumento en el impacto de este riesgo. Esto se debió a que el proyecto contaba con

un atraso en cuanto a *story points* pendientes y se acercaba la entrega de la primera versión al cliente, por lo que el impacto de este riesgo subió ya que era necesario que todos los integrantes cumplieran con lo planificado para recuperar el atraso. A partir de este momento el equipo se comprometió a cumplir con mayores objetivos a corto plazo y este riesgo disminuye continuamente a lo largo del tiempo (ver Ilustración 6.12 Distribución de esfuerzo por *sprint*).

El riesgo **R11** asociado a problemas de gestión debido a la inexperiencia en proyectos de este calibre, comenzó con una magnitud muy alta debido a que fue difícil organizar el trabajo y distribuir los tiempos entre las distintas actividades a realizar. El enfoque ágil utilizado permitió aprender sobre la marcha y así organizar el trabajo cada vez de mejor manera a lo largo del proyecto, disminuyendo los desvíos de lo planificado así como la probabilidad de ocurrencia de este riesgo.

A diferencia de los anteriores, la magnitud del riesgo **R12** que refiere a que uno de los integrantes abandone el grupo, siempre se mantuvo baja. Ésta llegó a cero en las últimas etapas, ya que el equipo consideró que la probabilidad de que esto suceda era nula. Una forma de mitigar este riesgo fue dedicar un tiempo en las retrospectivas para hablar de cómo se sentía cada uno respecto al proyecto.

Por último se analiza el riesgo **R13** referente al retraso en entregas debido a estimaciones incorrectas. La magnitud de dicho riesgo se mantuvo relativamente alta a lo largo de todo el proyecto descendiendo en el último período. Las reiteradas subestimaciones sobre las historias de usuario complejas provocaron un aumento en la magnitud. Esto pudo revertirse mediante el énfasis en historias de usuario más granulares y en mayor dedicación a los *spikes* para llegar a las planificaciones con menor incertidumbre sobre las próximas tareas a realizar.

6.6 Conclusiones y lecciones aprendidas

Observando los resultados obtenidos se puede concluir que la metodología aplicada junto con sus variaciones resultó exitosa, permitiendo al equipo trabajar de forma organizada, guiándose por objetivos a corto y mediano plazo, que posibilitaron la implementación de todos los requerimientos solicitados por el cliente como también *stretch goals* adicionales.

El registro de todas las actividades realizadas en las diferentes herramientas de gestión permitió obtener métricas que reflejaron la salud del proyecto, logrando identificar desvíos en cuanto a lo planificado y sirviendo como base para la toma de decisiones. Permitió verificar si se iba a poder cumplir o no con el alcance comprometido y reorganizar las prioridades del proyecto en conjunto con Ascentis.

La asignación de roles permitió mejorar aún más la organización del equipo, donde cada integrante se sintió cómodo con su rol y se hizo responsable de los documentos y actividades de sus áreas. La comodidad de cada uno fue expresada en distintas instancias de reunión internas del equipo donde cada uno expresaba su opinión referente a ésto.

Se destaca la importancia de la gestión de riesgos, que permitió tomar una actitud proactiva frente a los problemas que pudieran surgir y los planes de respuesta permitieron que estos problemas no retrasen el avance del proyecto.

Una de las lecciones más importantes para el equipo fue la importancia de los *spikes* sobre las distintas tecnologías para estimar las historias de usuario. En los primeros *sprints* los *spikes* se realizaban al inicio de cada uno, y generalmente ocurría que se habían subestimado las historias de usuario relacionadas, provocando *sprints* incompletos por mala estimación. La solución para esto fue mover los *spikes* para el final del *sprint* y utilizarlos para la planificación del siguiente. De esta forma se logró planificar mejor ya que se contaba con el conocimiento técnico de lo que se debía implementar antes de comprometerse a su implementación.

Por último, esta forma de trabajo afectó notoriamente de forma positiva la satisfacción del cliente ya que el constante contacto con el mismo lo hizo sentir parte del proyecto y sus aportes llevaron a un producto de mayor valor y calidad de lo que el equipo podría haber implementado sin su soporte.

7 Gestión de calidad

Este capítulo tiene como objetivo presentar las distintas actividades llevadas a cabo y las decisiones tomadas para el aseguramiento de la calidad tanto del producto como del proceso.

En primer lugar se muestran los objetivos en cuanto a la calidad que el equipo buscó cumplir. Luego se detalla el plan de calidad que guió al equipo durante todo el proceso, especificando las actividades realizadas durante las distintas fases del proyecto.

También se exponen las decisiones que se tomaron para cumplir con estas actividades, los procesos involucrados en la gestión de defectos, las revisiones realizadas y las actividades empleadas para la medición y seguimiento de la satisfacción del cliente.

Por último se presentan las conclusiones y lecciones aprendidas relacionadas a la gestión realizada por el equipo para asegurar la calidad del proyecto.

7.1 Objetivos de calidad

Para gestionar la calidad de forma correcta es necesario definir qué se entiende por calidad, tanto para el cliente como para el equipo. Es por esto que se definieron objetivos de calidad que el equipo se propuso cumplir para asegurar la misma. Siguiendo con los principios SMART [49], estos objetivos deben ser específicos, medibles, alcanzables, relevantes y con un tiempo determinado.

7.1.1 Objetivos de calidad del proceso

Los objetivos de calidad del proceso fueron definidos por el equipo y se enfocaron en asegurar la satisfacción del cliente con el trabajo realizado y la eficiencia del equipo durante el desarrollo.

Se definieron los siguientes objetivos:

- **Minimizar el retrabajo**

  En el documento de investigación *Deuda técnica: ¿Cuáles son los límites de la metáfora?* [50] los autores definen a la deuda técnica como *"… el costo futuro que puede tener la omisión de ciertas actividades en el proceso de Ingeniería de Software"*, relacionando esta métrica directamente con el retrabajo. El equipo se propuso que la deuda técnica de cada *release* no podía superar un máximo de 1 día de retrabajo (24 horas). En el Anexo 11.10 Métricas de calidad del código se profundiza sobre la evidencia y su análisis respecto a este objetivo.

- **Conseguir excelentes niveles de satisfacción del cliente**

  La satisfacción del cliente refleja si el equipo está trabajando según lo esperado, y es influenciada por la calidad del proceso de trabajo empleado. Se decidió medir la misma mediante encuestas utilizando la escala de 5 niveles de Likert [51]. Se estableció como objetivo conseguir niveles de satisfacción mayores o iguales a 4.5 en todo momento. La sección 7.3.7 Satisfacción del cliente detalla el contenido y resultados de estas evaluaciones.

- **Minimizar la introducción de defectos**

  Se concuerda que para mantener una eficiencia adecuada durante el desarrollo hay que minimizar la introducción de defectos. El equipo se propuso como objetivo que la cantidad máxima de defectos existentes por *sprint* debía ser siempre menor o igual a 5. Para esto se definieron actividades de *testing* y revisiones de código que se explicarán en las sección 7.3.2. Revisiones y 7.3.5. Pruebas de *software*.

7.1.2 Objetivos de calidad del producto

Al cumplir Ascentis con el rol de *product owner,* los objetivos de calidad asociados al producto fueron definidos según sus consideraciones durante las primeras reuniones. Aquí el cliente definió cuáles eran los aspectos que consideraba fundamentales para que el producto sea de utilidad una vez entregado.

Los objetivos definidos fueron los siguientes:

- **Cumplimiento de la totalidad de los requerimientos funcionales imprescindibles para Ascentis**
  El producto final debe contar con la totalidad de los requerimientos funcionales imprescindibles solicitados por el cliente al principio del proyecto. Estos requerimientos, los cuales conforman el MVP, fueron considerados como requeridos para conseguir la aceptación de Ascentis. (ver sección 3.4 Listado de requerimientos).

- **Cumplimiento de la totalidad de los requerimientos no funcionales**
  Ascentis y el equipo concuerdan en que los requerimientos no funcionales son tan importantes como los funcionales para el éxito del proyecto, por lo que se define el objetivo de cumplir con la totalidad de los requerimientos no funcionales relevados.

- **Proyectos con alta cobertura de pruebas unitarias**
  Durante las primeras reuniones el cliente expresó que las pruebas unitarias, además de brindar seguridad respecto a las modificaciones del código, son una excelente fuente de documentación para desarrolladores que se incorporan a un proyecto. Ascentis considera que para poder continuar agregando valor al producto luego de la entrega final, es importante que las funcionalidades estén acompañadas de buenas pruebas unitarias para la mayor cantidad de casos posibles. Por tanto se definió el

objetivo de alcanzar una cobertura mínima del 75% sobre el código entregado en la versión final.

- **Obtener análisis de sentimiento de feedbacks precisos**
  Para que el producto pueda aportar real valor es necesario que el análisis del sentimiento del *feedback* sea lo más certero posible, por lo que el equipo fija como objetivo lograr obtener análisis precisos respecto a los *feedbacks* procesados. Se recopiló un set de datos de prueba con *feedbacks* obtenidos de distintas fuentes para someterlos al análisis de Feedback Assistant y así poder validar este objetivo. Para medirlo el equipo etiquetó cada *feedback* considerando el sentimiento que más se adecuaba al mismo, luego se analizó el *feedback* mediante el sistema y se compararon los resultados para de esta forma visualizar el desvío del análisis respecto a lo esperado. Se cumple con el objetivo si al menos el 80% de los datos de prueba cumplen con lo etiquetado. Como se muestra en el Anexo 11.6 Comparación de análisis de sentimiento de *feedbacks* se logra un 85% de efectividad en los análisis generados, cumpliendo de esta forma con el objetivo planteado.

7.2 Plan de calidad

Según la Asociación Española para la Calidad "se define Plan de calidad como el documento que especifica qué procedimientos y recursos asociados deben aplicarse, quién debe aplicarlos y cuándo deben aplicarse a un proyecto, producto, proceso o contrato específico." [52]

En las primeras fases del proyecto el equipo definió un Plan de Calidad con el fin de llevar a cabo una adecuada trazabilidad respecto al cumplimiento de los objetivos detallados anteriormente en el presente capítulo.

Para la creación del mismo se realizó una categorización de las distintas fases del proyecto y para cada una se definieron una serie de actividades relacionadas a la calidad. Asociadas a éstas actividades también se indican los productos consumidos, productos generados, roles responsables y participantes en cada una.

En el Anexo 11.8 Plan de Calidad se presenta el plan de calidad definido para el proyecto.

7.3 Aseguramiento de la calidad

Para asegurar la calidad tanto del producto como del proceso, se siguieron un conjunto de actividades, estándares y buenas prácticas que ayudaron a cumplir con los objetivos de calidad establecidos. Estas actividades de soporte son transversales al proceso de desarrollo, realizándose durante todas las fases del proyecto.

7.3.1 Aplicación de estándares

El cumplimiento de estándares ayuda a generar un producto consistente, con código fuente y documentación fácil de leer, entender y mantener. Cumplir tanto con estándares de programación como de documentación es fundamental para que el equipo se complemente trabajando sin ambigüedades en la elaboración de la solución y que Ascentis sea capaz de continuar trabajando sobre el producto final con facilidad.

7.3.1.1 Estándares de programación

Esta sección presenta los estándares de programación utilizados para el desarrollo del *software*. Estos refieren a las convenciones y normas a seguir a la hora escribir nuevo código y apuntan a la mantenibilidad del sistema.

Uso del idioma inglés

Considerando que el inglés es la lengua nativa tanto de Ascentis como de sus clientes y que éste es el idioma dominante en el rubro del *software*, el equipo seleccionó el inglés como idioma de desarrollo. Esto aporta a la legibilidad del código para que los futuros desarrolladores puedan entenderlo más rápidamente.

Estándares de código

Para producir código fuente alineado a los estándares utilizados en la industria, se siguieron los estándares de codificación para Java sugeridos por Oracle, *"Code conventions for the java programming language"* [53]. También se siguieron las buenas prácticas de programación presentadas por Robert C. Martin en su publicación *"Clean Code: A Handbook of Agile Software Craftsmanship"*, el cual fue parte de la bibliografía utilizada en múltiples dictados a lo largo de la carrera [54].

Métricas

Se utilizó la herramienta de código abierto SonarQube [55] para medir el correcto cumplimiento de los estándares utilizados en el código fuente. Esta herramienta permite realizar un análisis sobre el código identificando *code smells,* defectos, vulnerabilidades de seguridad y secciones de código donde no se respetan estándares de codificación.

Ilustración 7.1. Logo SonarQube

SonarQube analiza el código fuente y le asigna una clasificación de calidad de 5 niveles (A, B, C, D, y E) a distintos aspectos de calidad, donde A y E son las mejores y peores clasificaciones respectivamente. A continuación se muestra un análisis realizado sobre los microservicios de Feedback Assistant.

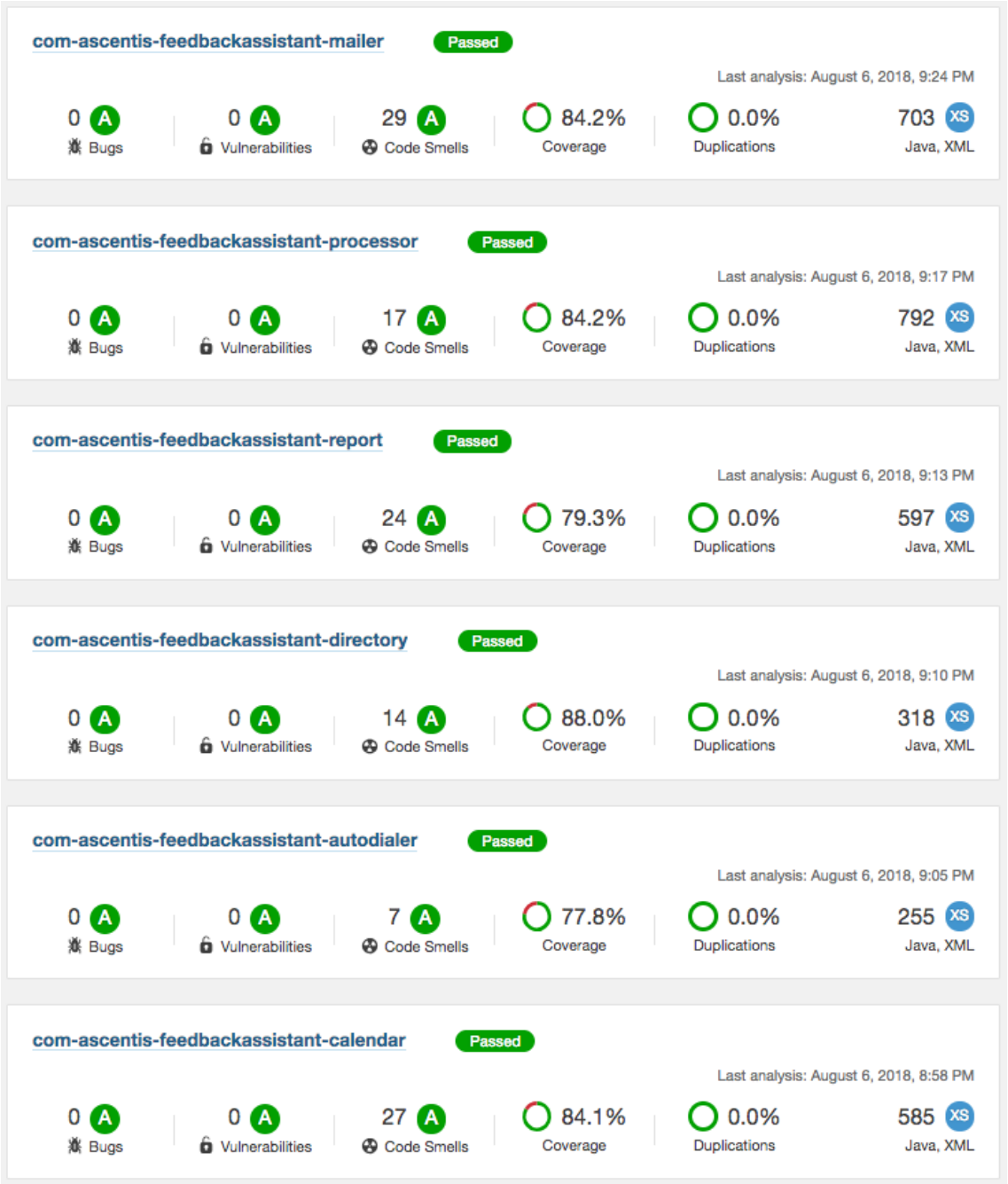


Ilustración 7.2 Análisis de SonarQube

En el Anexo 11.10 Métricas de calidad del código se puede profundizar sobre este análisis.

El equipo se encuentra muy satisfecho de haber alcanzado estos valores ya que tanto para éste como para el cliente es muy importante obtener buenos niveles de calidad.

7.3.1.2 Estándares de documentación

Para la creación del presente documento se siguieron distintas pautas y buenas prácticas asociadas a distintos estándares de documentación expuestos por la Universidad ORT Uruguay. Éstos se encuentran disponibles en el sitio Aulas de la institución. Específicamente para el caso del documento de arquitectura el equipo se basó en la bibliografía utilizada en el dictado Arquitectura de Software.

Normas y estándares	Documento asociado
Documento 302 - FI Normas específicas para la presentación de trabajos finales de Carrera de la Facultad de Ingeniería [56].	Documento final
Documento 303 - Hoja de verificación pautas de presentación de trabajos finales de carreras de Facultad de Ingeniería [57].	Documento final
Documento 304 - Normas para el desarrollo de trabajos finales de carrera [58].	Documento final
Documento 306 - Orientación para títulos, resúmenes o *abstracts* e informes de corrección de trabajos finales de carrera [59].	Documento final
Documento 307 - Pautas generales de formato de trabajos finales [60].	Documento final
Libro “Documenting Software Architectures: Views and Beyond - Second Edition” [61].	Documento de arquitectura
ORTsf Plantilla del Plan de la Calidad [62].	Plan de Calidad

Tabla 7.1 Normas y estándares utilizados para la creación del documento final.

7.3.2 Revisiones

Se realizaron distintas revisiones tanto del código fuente como de los documentos a lo largo del proyecto con el objetivo de detectar problemas o posibles mejoras.

Vale destacar la importancia que tuvieron las tres revisiones formales en la Universidad ORT Uruguay a lo largo del proyecto. Éstas permitieron encontrar desvíos y aplicar acciones correctivas sobre éstos. Para visualizar estos eventos en el tiempo ver sección 6.2 Hitos del proyecto. Para ver lo discutido en cada revisión ver Anexo 11.5 Revisiones ORTsf.

A continuación se detallan las actividades de revisiones realizadas sobre el código y los documentos generados.

7.3.2.1 Revisiones de código fuente

Éstas revisiones se llevaron a cabo mediante la utilización de *pull requests*. Cuando se finalizaba la implementación de alguna funcionalidad o corrección de defecto, el desarrollador a cargo de ésta tarea creaba un *pull request* con los cambios asociados al código, con el fin de llevar esa nueva sección de código a la rama base de desarrollo de Git (*develop*). En el mismo se agregaba a los otros desarrolladores del equipo como revisores para que aprueben o no la incorporación de ese nuevo código a la rama base de desarrollo, pudiendo hacer comentarios sobre el código si fuese necesario.

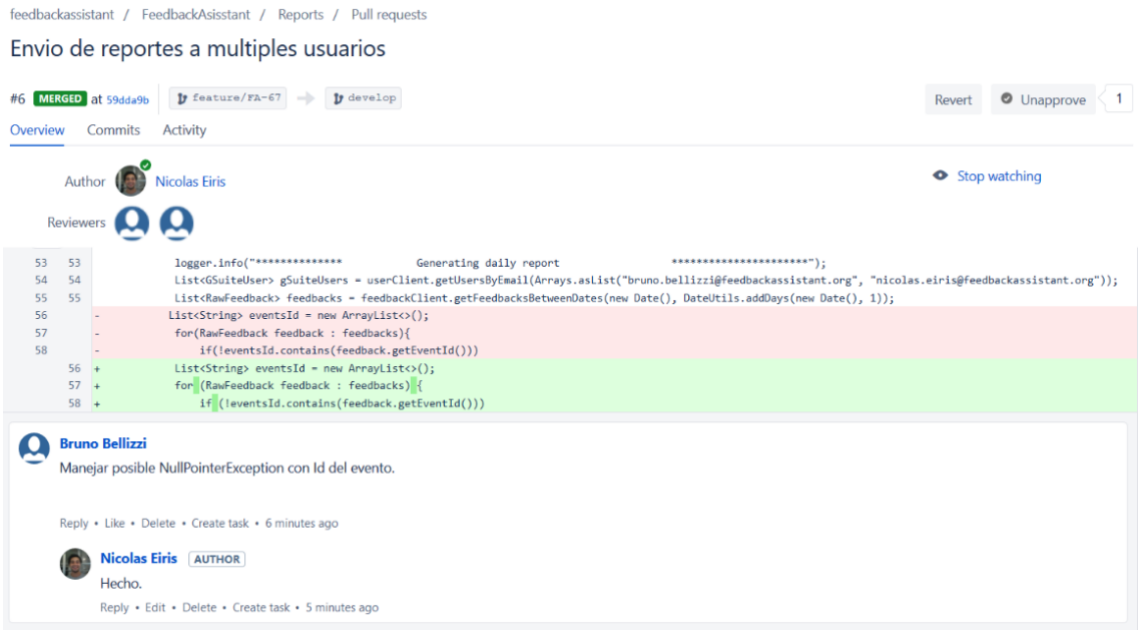


Ilustración 7.3 Ejemplo de Pull Request.

Esto permitió encontrar errores rápidamente y evitar que se conviertan en defectos, ya que cada nuevo cambio del código era revisado por otro integrante. Por otro lado, revisar los cambios de los compañeros ayudó a que todos conocieran el código de la totalidad de la solución.

7.3.2.2 Revisiones de documentos

Se tomó la decisión de generar revisiones de documentos utilizando la herramienta Google Docs. A medida que se finalizaba con la generación o avance de un documento, quien estaba editando notificaba al resto y el primer integrante que se encontraba disponible lo revisaba dejando comentarios si la situación lo ameritaba.

Se utilizó la funcionalidad de sugerencias donde el revisor del documento podía cambiar cierta sección de la documentación como sugerencia sin eliminar la existente. De esta forma se podían comparar los cambios y discutir acerca de cuál alternativa era más conveniente.

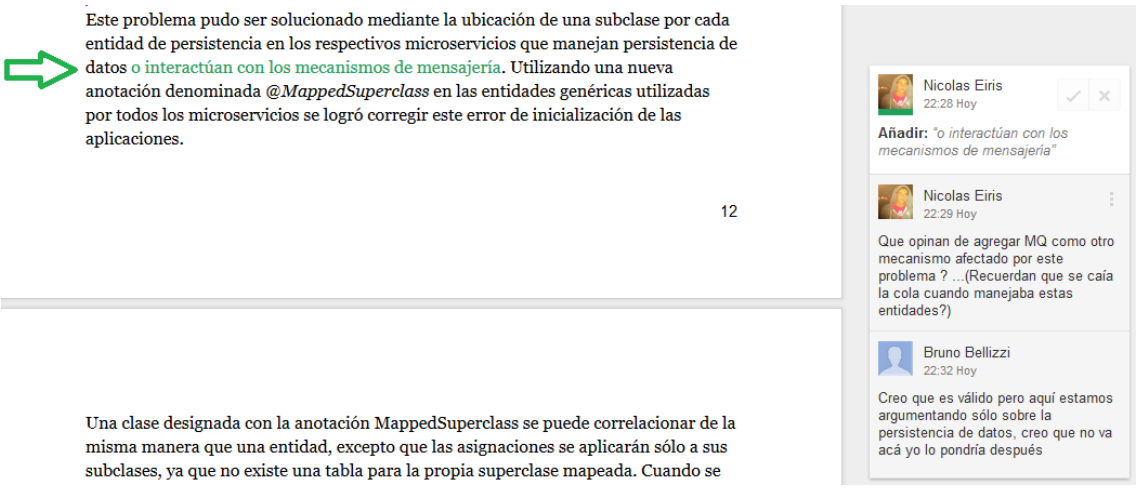


Ilustración 7.4 Ejemplo revisión de documento

7.3.3 Validaciones

Las validaciones del producto se realizaron durante cada presentación con el *product owner*, en donde el cliente validaba si el producto cumplía con sus expectativas o en su defecto explicaría las razones por las cuales no las cumplía (hecho que nunca sucedió). Las validaciones más relevantes se dieron durante la demo de la prueba de concepto (ver sección 3.2 Validación de requerimientos) y las entregas formales (ver sección 6.2 Hitos del proyecto). También se realizó una validación de la arquitectura con los arquitectos de Ascentis, la cual se detalla en la sección 4.5 Validación de arquitectura.

También se hicieron validaciones de seguimiento con el cliente donde se exponían los avances del proyecto mediante una presentación y luego se realizaba una demo del producto, similar a lo ocurrido en las revisiones de ORTsf.

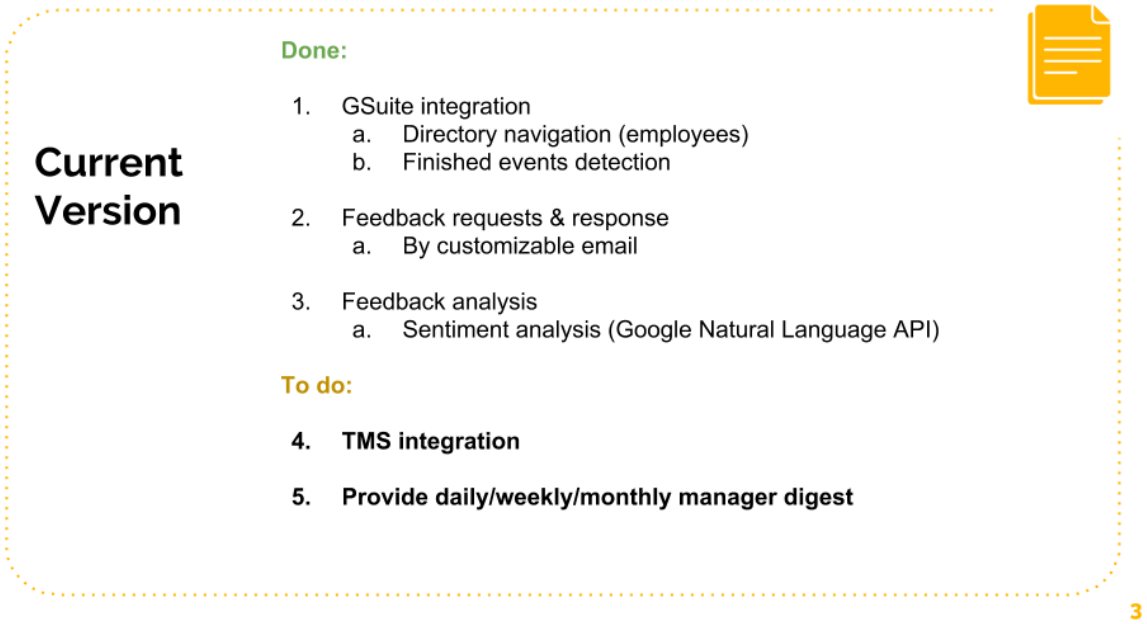


Ilustración 7.5 Diapositiva de presentación en reunión de seguimiento

7.3.4 Verificaciones

Éstas actividades refieren a la verificación de los distintos requerimientos ya sean funcionales o no funcionales. Se llevaron a cabo varias etapas de pruebas las cuales se detallan en la siguiente sección. Éstas permitieron asegurar una correcta verificación sobre el cumplimiento de los distintos requerimientos asociados al proyecto.

7.3.5 Pruebas de *software*

En busca de altos niveles de calidad el equipo tomó la decisión de llevar a cabo distintas actividades de *testing* sobre los distintos componentes de software.

Con esto se quiere expresar que se probó el comportamiento del producto desde cada método de clase aislado hasta el producto en ejecución probando los principales flujos de negocio.

A continuación se detallan los distintos tipos de pruebas realizados sobre la solución desarrollada.

7.3.5.1 Pruebas unitarias

Vincent Massol en su libro JUnit IN ACTION brinda la siguiente definición de prueba unitaria asociada al lenguaje de programación utilizado (Java), “A unit test examines the behavior of a distinct unit of work. Within a Java application, the ‘distinct unit of work’ is often (but not always) a single method. A unit of work is a task that is not directly dependent on the completion of any other task.” [63]

Por tanto estas pruebas refieren a asegurar el correcto funcionamiento del elemento más granular del código fuente, es decir, a cada método de clase.

Estos métodos o funciones de clase son aislados del resto cuando se realiza la prueba unitaria probando únicamente su entrada y salida. Aquí se percibe la aplicación del concepto de prueba de caja negra, donde no se analiza el comportamiento lógico interno del método sino que se ejecuta con una entrada determinada y luego se valida si retorna la salida esperada.

Además de permitir verificar el correcto funcionamiento de cada unidad lógica, estas pruebas sirven como fuente de documentación. Por ejemplo cuando los desarrolladores de Ascentis comiencen a trabajar sobre Feedback Assistant, una forma de comprender los comportamientos del producto es mirando las pruebas unitarias.

A continuación se presenta un ejemplo de una prueba unitaria implementada en uno de los microservicios.

```java
@Test
public void TestGetEventsByIdOk() {
    List<String> ids = Arrays.asList("ID1", "ID2");
    List<CalendarEvent> expected = new ArrayList<>();

    for(String id : ids) {
        CalendarEvent event = new CalendarEvent();
        event.setId(id);
        expected.add(event);
    }

    Mockito.when(eventRepository.findAll(ids)).thenReturn(expected);

    List<CalendarEvent> events = eventService.getEventsById(ids);
    Assert.assertEquals(expected, events);
}
```

Ilustración 7.6 Ejemplo prueba unitaria

Las revisiones de código consistieron no solamente en revisar el código fuente referente a las funcionalidades agregadas o modificadas, sino también revisar las pruebas unitarias asociadas a éstas y verificar que se estén cubriendo los cursos normales y de error.

Se utilizaron dos herramientas para la generación de pruebas unitarias automáticas. Por un lado *JUnit* como *framework* de *testing* unitario para facilitar la programación de estas pruebas. Por otro lado, en cada prueba unitaria se debía simular el comportamiento de otros componentes de los cuales el elemento a probar dependía. Para esto fue necesario contar con algún mecanismo de *mocking*, por lo que se tomó la decisión de utilizar *Mockito* para generar los respectivos *mocks* asociados a las dependencias involucradas.

Ilustración 7.7 Logo JUnit

Ilustración 7.8 Logo Mockito

A continuación se ejemplifica el mecanismo por el cual el equipo simuló el comportamiento de dependencias mediante *mocks*.

```
@RunWith(MockitoJUnitRunner.class)
public class EventServiceImplTest {

    @Mock
    private EventRepository eventRepository;

    @InjectMocks
    private EventServiceImpl eventService;
```

Ilustración 7.9 Ejemplo uso de Mockito.

Estas dos herramientas fueron de suma utilidad para aumentar la productividad del equipo y a no dedicar tanto tiempo a la codificación de la prueba sino a enfocarse en el diseño de los distintos casos a cubrir.

Por último, el equipo tomó la política de que ningún *pull request* era aprobado si el código implementado no estaba acompañado de sus correspondientes pruebas unitarias. Los resultados obtenidos por cada microservicio fueron los siguientes (datos medidos con la herramienta SonarQube):

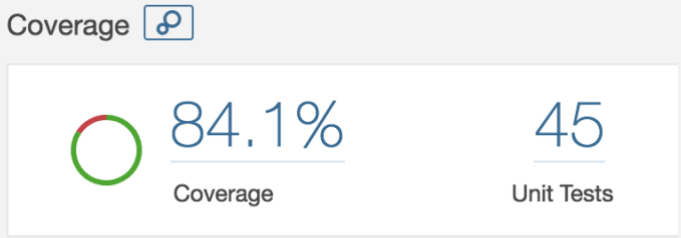
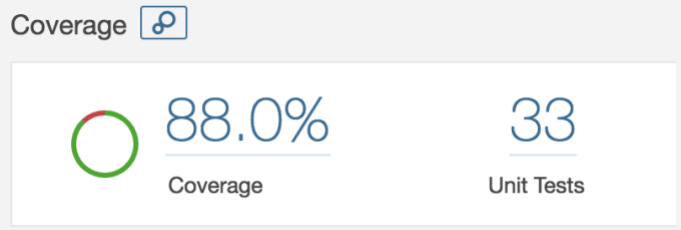
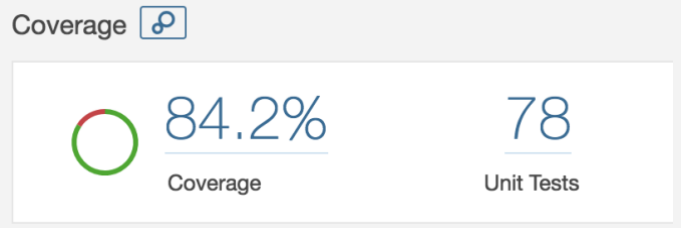
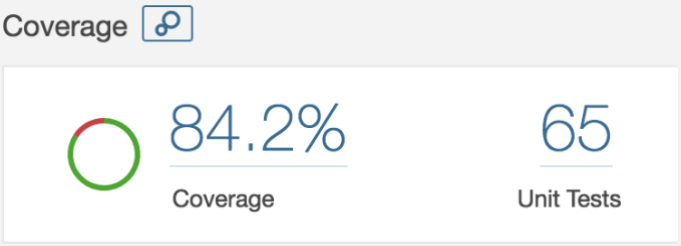
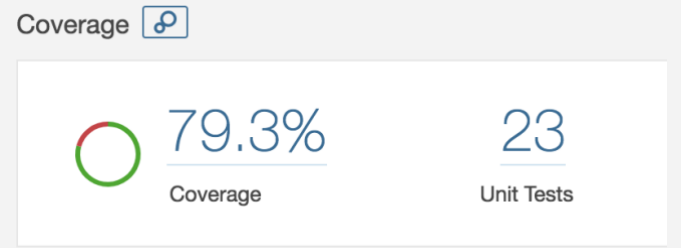
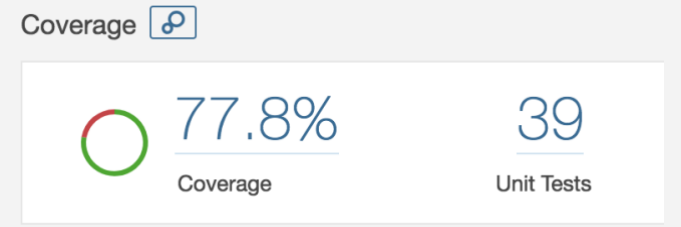
Microservicio	Cobertura y cantidad de pruebas unitarias
calendar	
directory	
mailer	
processor	
report	
autodialer	

Tabla 7.2 Cobertura y cantidad de pruebas unitarias

Como se puede ver en la tabla 7.2 se lograron alcanzar los objetivos establecidos en cuanto al nivel de cobertura de pruebas unitarias. Esto fue producto de que cada *pull request* debía estar acompañado de las pruebas unitarias que verificaran el correcto funcionamiento de los cambios introducidos, provocando así que la generación de pruebas unitarias fuera una actividad constante a lo largo del desarrollo.

Se puede ver que para el alto porcentaje de cobertura en cada microservicio la cantidad de pruebas unitarias no es elevada. Esto se debe a que cada componente posee una única responsabilidad, por lo que no existen tantos casos a cubrir como en una aplicación monolítica.

7.3.5.2 Pruebas de integración

Considerando la cantidad de componentes existentes en la solución (provocado por la aplicación del patrón arquitectónico de microservicios) fue de suma importancia llevar a cabo distintas pruebas que permitan validar el comportamiento de múltiples componentes trabajando en conjunto.

Una vez finalizadas las pruebas unitarias asociadas a una funcionalidad o cambio, el desarrollador a cargo debía realizar estas pruebas para verificar que las comunicaciones entre los componentes involucrados sean correctas.

Sucedió en algunas ocasiones que las unidades lógicas funcionaban correctamente de forma aislada (siendo exitosas las pruebas unitarias), pero luego al probar un flujo de comunicación entre varios componentes no se obtenía el resultado esperado, detectando de esta forma alguna anomalía en la interacción entre distintas estructuras de software debiendo ser revertida por el desarrollador.

7.3.5.3 Pruebas funcionales

Se realizaron pruebas de caja negra utilizando el software en ejecución e interactuando con el mismo. Ésto se realizó para verificar el cumplimiento de las especificaciones requeridas (ver capítulo 3. Ingeniería de Requerimientos) y eliminar los posibles defectos existentes.

Éstas fueron diseñadas con el propósito de cubrir la mayor cantidad de casos posibles que se pudieran dar en la operativa real de la aplicación. El equipo agrupó los casos de prueba según el requerimiento funcional al que pertenecen. A continuación se describen los atributos de cada caso de prueba:

- ID Requerimiento: refiere al identificador de requerimiento al que pertenece la prueba.
- Requerimiento Funcional: breve descripción del requerimiento funcional.
- ID Prueba: identificador de la respectiva prueba.
- Nombre: título descriptivo del propósito de caso de prueba.
- Descripción: descripción detallada del caso de prueba.
- Resultado esperado: el comportamiento esperado del sistema.
- Resultado obtenido: define si se cumplió o no el comportamiento esperado.
- ID Defecto: Identificador del defecto encontrado (puede ser más de uno).

A continuación se presenta una ilustración ejemplificando un caso de prueba.

Id Req.	Requerimiento Funcional	ID Prueba	Nombre	Descripción	Resultado esperado	Resultado obtenido	ID defecto
RF02	Sincronizar eventos con Google Calendar	P4	Sincronización de los eventos de un usuario	El sistema realiza una sincronización con los calendarios de los usuarios en Google Calendar, guardando los nuevos eventos y actualizando los existentes.	Se sincronizan los eventos de los usuarios, agregando los nuevos eventos y modificando los existentes.	OK	FA-50

Ilustración 7.10 Ejemplo de caso de prueba funcional

Para obtener mayor detalle acerca de estas pruebas ver el Anexo 11.9 Planilla de casos de pruebas funcionales.

7.3.5.4 Pruebas de regresión

Previo a cada *release* se llevó a cabo una prueba de regresión utilizando los casos de prueba funcionales con el objetivo de probar el sistema en su totalidad.

Durante el desarrollo de los *sprints* cada integrante agregaba o modificaba funcionalidades probando solamente los sectores de la arquitectura directamente afectados por el cambio (pruebas unitarias, integración y funcionales). De esta forma no se tenía conocimiento si el cambio había afectado negativamente a otros sectores de forma indirecta, de aquí la motivación por llevar a cabo estas pruebas.

Las pruebas de regresión implicaron utilizar la tabla de casos de prueba funcionales y llevarlos a cabo en su totalidad. El encargado de calidad era el responsable de esta actividad y las pruebas a realizarse eran distribuidas equitativamente entre todos los integrantes del equipo. Se prefirió no asignar casos de prueba al integrante que llevó a cabo la implementación de su comportamiento esperado.

Al abordar todos los casos de prueba existentes, ésta actividad le implicaba dedicar múltiples horas de trabajo a cada integrante por lo que no se podían realizar cada períodos cortos de tiempo. Por esto el equipo tomó la decisión de realizarlas durante el penúltimo *sprint* previo a cada *release* para dedicar el último *sprint* a la resolución de defectos. Por tanto se ejecutaron pruebas de regresión en el *sprint 7* previo al primer *release (MVP)* y en el *sprint 12* previo al segundo *release (*ver secciones 6.1.6 Planificación de iteraciones y 6.3.2. Plan de *releases).*

En algunos casos sucedió que al avanzar en el desarrollo de nuevas funcionalidades se estaban introduciendo defectos de manera indirecta sobre otros sectores del sistema. Éstos no fueron identificados en las demás actividades de calidad, logrando ser detectados mediante estas pruebas de regresión. Estas

pruebas fueron de gran utilidad ya que se lograron detectar múltiples defectos que habían pasado desapercibidos durante el desarrollo y otros tipos de pruebas, pudiendo ser corregidos antes de la entrega al cliente (ver sección 7.3.6.3 Métricas).

7.3.6 Gestión de defectos

En esta sección se describe cómo se llevó a cabo la gestión de los defectos que fueron identificados durante el desarrollo y pruebas, presentando el ciclo de vida de un defecto y las métricas resultantes del proceso.

Según el estándar 1044 de la IEEE [64], un defecto es cualquier condición que se aparte de lo esperado, pudiendo lo esperado ser definido por un documento (ej. requerimientos funcionales, diseño de arquitectura, casos de prueba) o por la percepción y experiencia de un individuo (ej. el equipo o el cliente). Para gestionar correctamente estos defectos de forma de poder minimizarlos, se implementó un proceso de gestión de defectos que permitió llevar un correcto seguimiento de los mismos.

7.3.6.1 Identificación y registro

Los defectos fueron identificados y registrados durante el desarrollo y particularmente durante las pruebas de integración y regresión, utilizándose Jira como la herramienta para el registro y seguimiento de los defectos. La utilización de la misma herramienta para gestionar los *sprints* y los defectos facilitó al equipo la organización del trabajo pendiente ya que podía visualizar tanto historias de usuario como defectos en el mismo *board*. También permitía asignar la resolución de ciertos defectos como tareas dentro de un *sprint* en particular, permitiendo planificar cuándo se debían solucionar ciertos defectos y quién debía solucionarlos. Por último, otra funcionalidad de utilidad que facilitó la herramienta fue la definición de un ciclo de vida para los defectos, el cual se detalla en la presente sección.

Cuando un integrante del equipo identificaba un defecto durante alguna de las actividades de desarrollo o testing, el mismo se registraba con las siguientes características:

- **Identificador del defecto:** Código unico que identifica al defecto. Es utilizado para el registro del defecto en los casos de prueba y para identificar la rama de *Git* (ver capítulo 8. Gestión de la configuración).
- **Título:** Breve título descriptivo del defecto.
- **Descripción:** Descripción del defecto conteniendo los pasos para reproducirlo, el resultado esperado y el resultado obtenido.
- **Estado:** Indica el estado en el cual se encuentra el defecto. Los posibles estados son: Por hacer, En curso, En revisión, Testing, Listo
- **Prioridad:** Categoriza el evento según cuán crítico es para el funcionamiento del sistema. Existen 5 posibles categorías: Muy alto, Alto, Medio, Muy bajo y Bajo. El equipo definió que ningún defecto con prioridad mayor o igual a Alto podía incluirse en un *release*.
- **Fecha (de creación y de resolución):** Fechas en las que se registró y solucionó el defecto.
- **Sprint:** Indica el *sprint* en donde se identificó el defecto.
- **Reportado por:** Integrante que identificó y registró el defecto en cuestión.
- **Responsable:** Integrante responsable de solucionar el defecto. Se priorizó que quien introdujo el defecto en el sistema fue el responsable de corregirlo ya que contaba con mayor conocimiento del problema.
- **Adjuntos:** Cualquier tipo de archivo que sea de utilidad para entender o resolver el defecto.

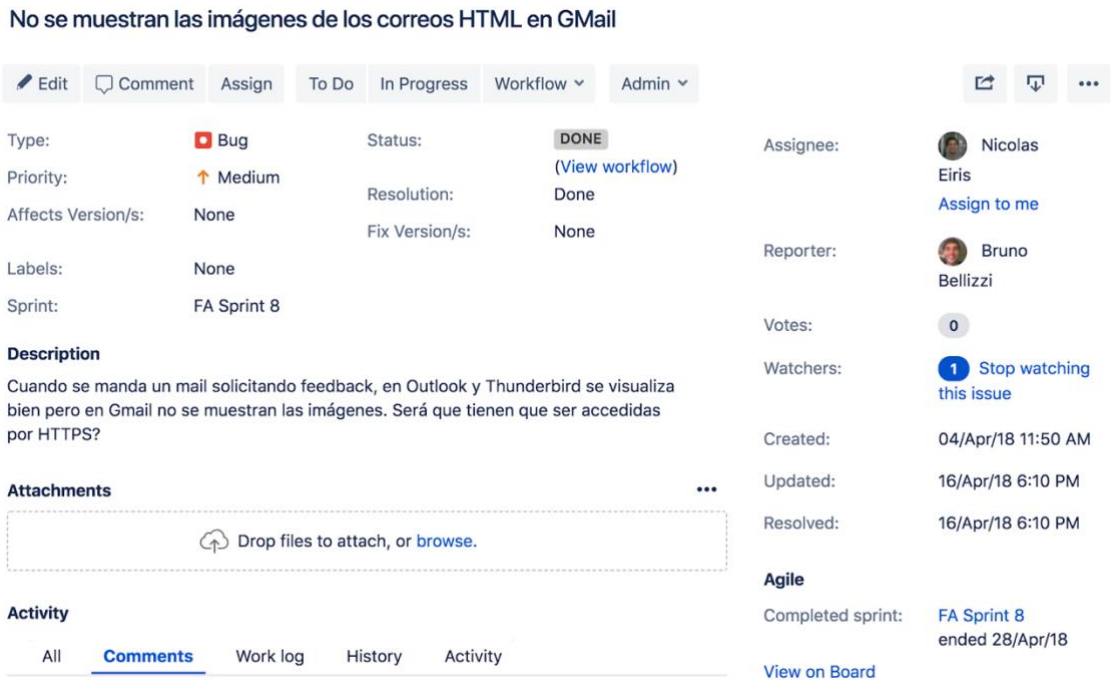


Ilustración 7.11 Ejemplo de defecto registrado en JIRA

7.3.6.2 Ciclo de vida de un defecto

A continuación se detalla el ciclo de vida utilizado por el equipo para gestionar los defectos identificados a lo largo del proyecto.

El ciclo de vida comienza cuando un integrante encontraba un defecto y lo registraba en la herramienta, donde debía asignar como responsable a quien había estado trabajando en la funcionalidad relacionada al defecto identificado. En caso de que quien reporta sea distinto del responsable, Jira le enviaba un correo electrónico informando que se le había asignado un nuevo defecto para resolver. En ese momento el defecto se encontraba en estado "Por hacer" (*To Do*) a la espera de que el responsable comience su corrección.

Cuando el responsable comenzaba a trabajar en la corrección del defecto, el mismo cambiaba a estado "En curso" (*In Progress*). Luego, cuando se finalizaba su corrección y se superaban todas sus pruebas unitarias, el responsable realizaba un *pull request* al resto del equipo para realizar la revisión de código correspondiente (ver sección 7.3.2.1 Revisiones de código fuente) y pasaba su

estado a “Revisión de código” (*Code Review*). El responsable daba por terminado el desarrollo y podía comenzar a trabajar en otra tarea.

Una vez aprobado el *pull request* el defecto pasaba al estado de “*Testing*”, en el que se mantenía hasta que se superaban las pruebas funcionales necesarias para verificar que el defecto fue correctamente corregido. En caso contrario, el defecto volvía al estado “Por hacer” y el proceso volvía al estado inicial.

A continuación se muestra un diagrama que representa el ciclo de vida utilizado.

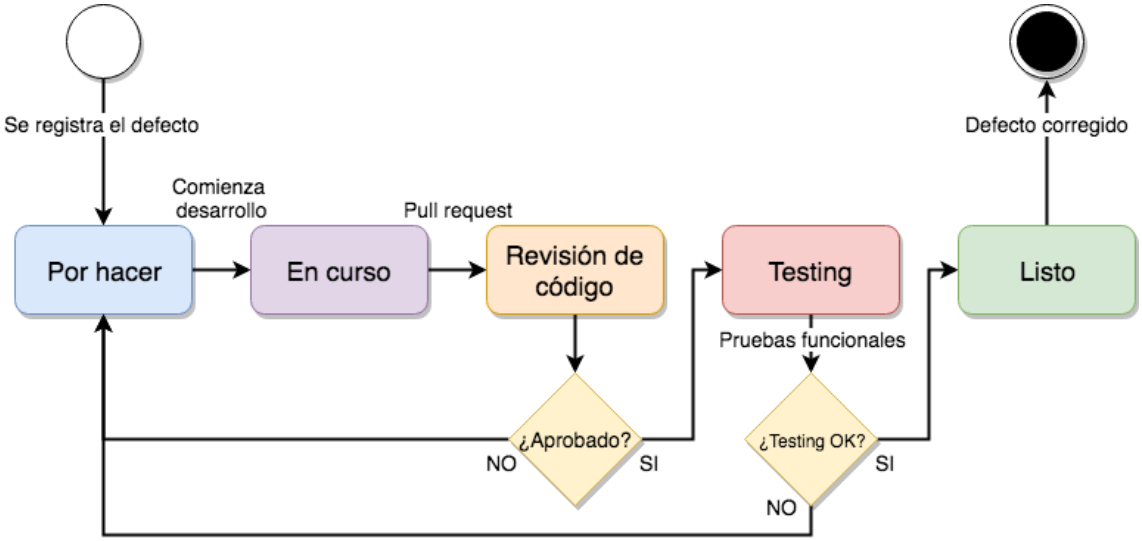


Ilustración 7.12 Ciclo de vida de un defecto

7.3.6.3 Métricas

A lo largo del proyecto se fueron recolectando datos específicos tanto del proceso como del producto, en busca de tomar decisiones justificadas acorde a datos reales y poder cumplir con los objetivos establecidos. En esta sección se resumen las métricas utilizadas y se explican sus resultados.

Métricas del proceso

Éstas permitieron realizar correcciones en cada *sprint* en cuanto al proceso de trabajo con el fin obtener niveles de calidad deseados y cumplir con los tiempos establecidos. Llevar una correcta trazabilidad en cuanto a la calidad del proceso es vital ya que esta se vé directamente relacionada con la calidad del producto.

En cuanto a la gestión de proyecto se realizaron varios controles:

- Desvío de *story points* realizados vs. *story points* planificados. Ver sección 6.1.7 Seguimiento y evaluación de iteraciones.

- Evolución del product backlog presentando la cantidad de trabajo restante vs. lo planificado en *story points*. Ver sección 6.1.7 Seguimiento y evaluación de iteraciones.

- Distribución de esfuerzo por mes. Ver sección 6.4 Gestión de esfuerzo.

Se llevó a cabo una inspección cuantitativa asociada al objetivo de permitir una cantidad máxima de 5 defectos acumulados por *sprint*. Con esto se hace referencia a la totalidad de defectos abiertos al momento de cerrar un *sprint*, contando los identificados en ese *sprint* más los que no han podido ser resueltos previamente.

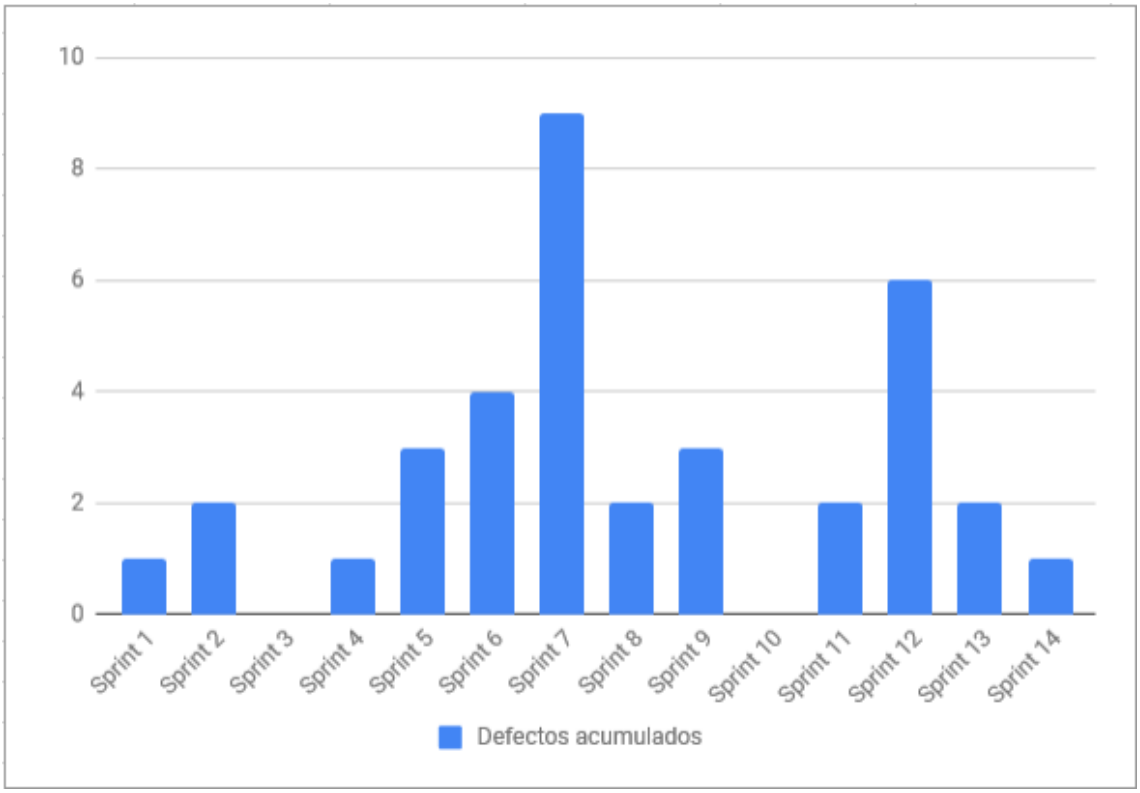


Ilustración 7.13 Acumulación de defectos en cada sprint.

Como se puede observar en la ilustración, se cumple con el objetivo de no superar los 5 defectos acumulados por *sprint* en la mayoría de ellos, obteniendo un 86% de efectividad asociado a este objetivo. Particularmente durante los *sprints* 3 y 10 no se identificó ningún defecto y todos los identificados anteriormente habían sido corregidos.

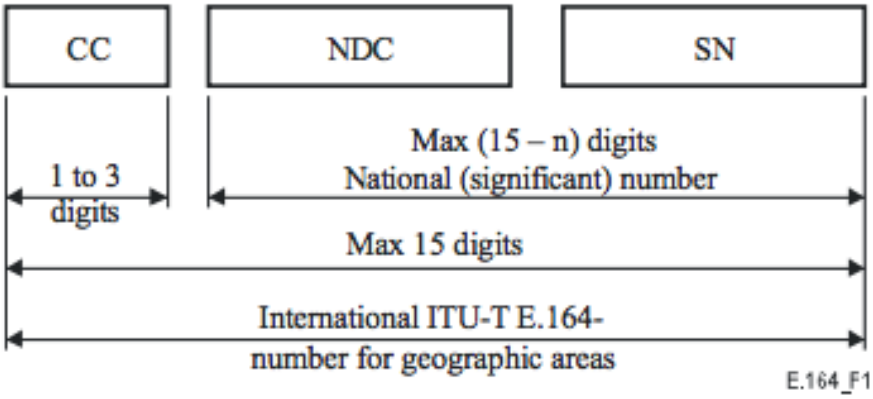
El no cumplimiento de este objetivo se dió solamente en los *sprints* 7 y 12, siendo éstas las iteraciones en donde se llevaron a cabo las pruebas de regresión. Como se menciona en la sección 7.3.5.4. Pruebas de regresión, éstas se realizaron en el penúltimo *sprint* previo a cada *release*. En este gráfico se observa la efectividad de las mismas en cuanto a la detección de defectos.

En la gráfica se puede ver que en el último *sprint* existe un único defecto encontrado y aún no resuelto. Este defecto corresponde al caso de prueba P51 que se muestra a continuación.

Id Req.	Requerimiento Funcional	ID Prueba	Nombre	Descripción	Resultado esperado	Resultado obtenido	ID defecto
RF12	Solicitar feedback mediante llamada telefónica al finalizar un evento	P51	Número de teléfono no existe	Si el número de teléfono registrado del usuario no es un número válido, se debe ignorar la llamada y dejar un registro en el log	Se cancela la llamada y se deja registro en el log	No se valída	FA-171

Ilustración 7.14 Caso de prueba P51

Aquí se está probando que el sistema sea capaz de validar un número telefónico antes de realizar una llamada telefónica. Esto no es trivial, ya que la validación debe funcionar para cualquier número telefónico independientemente del país al que pertenezca. Para validar esto habría que implementar un algoritmo que valide que un número respete el estándar internacional E.164 [65] de la ITU (International Telecommunication Union) la cual especifica cómo los números de teléfono son distribuidos en el mundo.



CC Country Code for geographic area
NDC National Destination Code
SN Subscriber Number
n Number of digits in the country code

NOTE – National and international prefixes are not part of the international ITU-T E.164-number for geographic areas.

Ilustración 7.15 ITU-T E.164 - Estructura de números para áreas geográficas

El mayor desafío es que tanto el CC (código de país) como el NDC (código de área) son códigos de largo variable que deben corresponder a cierta combinación de códigos válidos, ya que cada país tiene su propio CC y determinados NDC. Considerando el costo-beneficio que implicaría corregir este defecto, el mismo aún se encuentra pendiente de corrección. La probabilidad de que ocurra este escenario es muy baja y en caso de ocurrir no tendría mayor impacto sobre el sistema (sólo generaría una llamada fallida y trazabilidad de error en el log) por lo que se tomó la decisión de mantenerlo pendiente e informarle al cliente.

Métricas del producto

Una vez relevados los requerimientos no funcionales y detectados los atributos de calidad a favorecer, fue de suma importancia definir el mecanismo por el cual éstos atributos eran medidos para asegurar el favorecimiento de los mismos.

Esta sección tiene como propósito exponer al lector las métricas asociadas a los principales atributos de calidad. En particular se evaluaron los atributos Modificabilidad y Portabilidad al ser estos los más importantes para el proyecto en cuestión.

- **Modificabilidad**

Uno de los atributos a los cuales se hizo mayor énfasis refiere a la facilidad de cambio. Por esto se decidió aplicar el patrón arquitectónico de microservicios y se ejecutaron múltiples tácticas para realizar un correcto favorecimiento del mismo.

Igualmente se debió definir un mecanismo para medir el favorecimiento de la modificabilidad. Para esto se utilizó la herramienta *SonarQube* presentada previamente en el presente capítulo, la cual provee métricas asociadas a este atributo de calidad.

Ilustración 7.16 Métrica modificabilidad microservicio calendar.

Esta gráfica muestra la relación entre la cantidad de líneas de código y la deuda técnica de cada clase del microservicio analizado. La herramienta utiliza una escala denominada *SQALE Rating* [66] (detallada en el anexo), de esta forma se categoriza cada clase según su mantenibilidad. Como se puede apreciar en la ilustración planteada a modo de ejemplo, el 100% de las clases del microservicio *calendar* se establecen con la mejor categorización posible dando a conocer el alto favorecimiento de este atributo de calidad. Para ver el resultado de otros microservicios, ver Anexo 11.10 Métricas de calidad del código.

- **Portabilidad**

Por otro lado se debió llevar a cabo una medición del favorecimiento de la portabilidad en la arquitectura del sistema. Para medir este atributo, se generó una tabla analizando el *stack* de tecnologías utilizadas respecto a las posibles variaciones de entornos de ejecución.

Tecnología	Microsoft Windows	Linux	MacOS
Java	✓	✓	✓
PostgreSQL	✓	✓	✓
Apache Tomcat	✓	✓	✓
Apache ActiveMQ	✓	✓	✓

Tabla 7.3 Métrica de portabilidad.

Como se puede apreciar en la tabla el equipo es capaz de favorecer notoriamente el atributo de calidad de portabilidad mediante el *stack* de tecnologías utilizado.

Se apuntó en todo momento a generar una solución que permita a Ascentis poder comercializar este producto integrado a su *suite* de productos sin restricciones en cuanto al ambiente de ejecución. Esta métrica permite asegurar el alto grado de flexibilidad por parte de Feedback Assistant en cuanto al sistema operativo donde el mismo se ejecuta. Esto permite que Feedback Assistant sea una opción posible para todos los clientes de Ascentis sin restricciones de infraestructura.

7.3.7 Satisfacción del cliente

Al tratarse de un proyecto para un cliente es de suma importancia lograr cumplir con las expectativas del mismo y si es posible superarlas. Luego de cada *release* el equipo envió una encuesta a los asistentes de la reunión con el objetivo de obtener *feedback* y así lograr medir la satisfacción del mismo.

Para esto se utilizaron afirmaciones asociadas a distintos aspectos del proyecto y se le brindó la posibilidad de seleccionar una respuesta utilizando la escala de Likert de cinco niveles para transmitir su opinión respecto a la afirmación presentada. Siendo 1 la opción asociada al total desacuerdo y 5 al total acuerdo con la afirmación planteada.

Las afirmaciones expuestas en la encuesta final fueron las siguientes:

- El nivel de satisfacción global es muy alto.
- El equipo siempre se encontró disponible.
- Se notó al equipo altamente involucrado con el proyecto.
- El equipo propuso ideas innovadoras.
- Se entregó el producto funcional que Ascentis esperaba.
- Se entregó un producto con altos niveles de calidad.
- Los entregables se realizaron dentro de las fechas pactadas.

En el Anexo 11.12 Encuestas de satisfacción del cliente se presentan las respuestas obtenidas por parte de los integrantes de Ascentis involucrados en el proyecto durante ambas entregas.

Los resultados demostraron que el objetivo relacionado a la satisfacción del cliente ha podido ser cumplido en ambas demos realizadas. Para la primera versión entregada se obtuvo una satisfacción promedio de 4.5 sobre 5, mientras que en la segunda se obtuvo un promedio de 4.8 sobre 5. Esto es sumamente positivo ya que además de superar el objetivo planteado se ve cómo la satisfacción del cliente aumentó mientras se avanzaba en el proyecto.

Podemos concluir entonces que el cliente se encuentra sumamente satisfecho respecto al trabajo realizado por el equipo en ambas entregas. La tercer y última entrega está planificada para el primero de octubre y se apuntará a aumentar estos niveles de satisfacción.

7.4 Conclusiones y lecciones aprendidas

Debido al contexto académico y el trato profesional con el cliente, el equipo considera que la gestión de la calidad fue clave para lograr cumplir con los objetivos expuestos en este capítulo. La realización de todas las actividades definidas en el plan de calidad favorecieron de forma notoria tanto al proceso como al producto final.

Se destaca la importancia de las revisiones para el aseguramiento de la calidad y en particular las revisiones de código, que permitieron no solo identificar errores antes de que se conviertan en defectos, sino también permitir conocer lo realizado por los otros integrantes y de esta forma homogeneizar el trabajo. El hecho de que todos los desarrolladores debieran aprobar todos los cambios en el código aportó también una sensación de seguridad al introducir modificaciones y fortaleció el sentimiento de equipo, en donde los defectos eran responsabilidad de todos y no de quien los introducía.

A su vez fueron de gran utilidad las pruebas de regresión, considerando que al probar exhaustivamente el sistema se logró detectar una gran cantidad de defectos que habían pasado por alto y que de lo contrario hubiesen sido incluidos en las entregas al cliente, lo que hubiera sido una gran amenaza para la calidad.

Una decisión muy importante fue la utilización de la herramienta analítica *SonarQube* que además de validar, exigir el cumplimiento de los estándares de programación y proveer distintas métricas, permitió conocer una gran cantidad de buenas prácticas sobre el lenguaje Java que eran desconocidas antes del proyecto.

Por último se quieren destacar las actividades realizadas para la medición de la satisfacción del cliente. Si bien todos los objetivos de calidad fueron cumplidos, el objetivo referente a conseguir excelentes niveles de satisfacción por parte del cliente fue considerado el más trascendente. Al validar que esto se estaba cumpliendo motivó al equipo a trabajar con mayor dedicación para superar aún más las expectativas de Ascentis.

8 Gestión de la configuración

En este capítulo se darán a conocer las actividades que permitieron llevar a cabo la gestión de la configuración tanto del *software* como de la documentación generada.

Se detallarán las herramientas utilizadas, las estrategias para el manejo de versiones, los mecanismos para gestionar las dependencias del software así como también los elementos de la configuración presentes en la solución.

Estas actividades fueron claves para lograr trabajar sobre los mismos artefactos en simultáneo y poder llevar un control sobre los avances que se realizaban a lo largo del tiempo.

Cabe destacar que todas las estrategias y mecanismos aplicados en estas actividades fueron transversales a las distintas fases del proyecto logrando ser un gran soporte para la calidad tanto del producto como del proceso.

8.1 Identificación de los elementos de la configuración

El equipo identificó dos tipos de elementos de la configuración del *software* (ECS); *software* y documentos. Éstos artefactos sufrieron varios cambios en el tiempo, lo que provocó la necesidad de llevar a cabo un control y registro de su evolución.

En la siguiente tabla se presentan para cada tipo de elemento los distintos artefactos gestionados.

Tipo	Artefacto
Software	• Código fuente • APIs de terceros • Bibliotecas de terceros
Documentos	• Encuestas realizadas a Ascentis • *Product backlog* • Plan de calidad • Planilla de casos de prueba funcionales • Documento de arquitectura • Plan de riesgos • Plan de *releases* • Requerimientos funcionales y no funcionales • Informes de avance • Documento final

Tabla 8.1 Artefactos de la configuración identificados

8.2 Elección de herramientas

Teniendo en cuenta la categorización definida en la sección anterior, se procede a exponer las herramientas utilizadas sobre los distintos artefactos para poder llevar una correcta gestión sobre los mismos.

8.2.1 Documentos

Para la gestión de los distintos artefactos de este tipo es necesario que la herramienta a utilizar cuente con las siguientes características:

- Ser colaborativa permitiendo la edición de documentos en simultáneo por múltiples usuarios y poder visualizar los cambios realizados en tiempo real.

- Que el espacio de almacenamiento no sea una limitante considerando la cantidad de documentos que se manejan.

- Permitir alojar sugerencias o comentarios asociados al contenido de un documento sin necesidad de sobreescribir lo existente.

- Contar con un historial de cambios.

- Manejar una estructura de directorios y archivos.

- Facilidad para compartirlo con otros individuos considerando que muchos de estos artefactos serían compartidos por ejemplo con el tutor o Ascentis.

Se optó por utilizar la herramienta Google Drive para gestionar estos artefactos. Este servicio además de ser gratuito y utilizado por todos los integrantes del equipo tanto en el ambiente laboral como académico, provee variadas funcionalidades que permiten satisfacer todas las necesidades listadas previamente en la presente sección.

Se manejó también la posibilidad de utilizar Confluence [67] como herramienta para los documentos, pero el equipo consideró que no es muy flexible ni intuitiva en cuanto a la creación de tablas y ordenación de la información en las mismas. Teniendo en cuenta que se cuenta con una gran cantidad de tablas a lo largo de los documentos no era conveniente. Además no provee la fácil navegabilidad ni la edición simultánea del contenido de un documento por varios usuarios como permite Google Drive.

8.2.2 *Software*

No hubieron discrepancias a la hora de seleccionar la herramienta para la gestión de los artefactos asociados al *software*. El equipo tomó la decisión de seleccionar una herramienta que ya dominaba, debido a que fue utilizada en múltiples dictados a lo largo de la carrera y a su vez en el ámbito laboral, en busca de invertir horas de investigación en otros aspectos del proyecto y no en la herramienta de gestión del código fuente.

Se utilizó el sistema distribuido Git [68] para el control de versiones del *software* y BitBucket [69] como plataforma de hosting para dicho código fuente. La elección de BitBucket se debe principalmente a dos motivos fundamentales.

En primer lugar, junto con Jira (herramienta para la gestión de proyecto utilizada) pertenecen a la compañia Atlassian. Esto permite una gran compatibilidad entre ambas herramientas pudiendo relacionar directamente un *pull request* con la historia de usuario alojada en el *sprint* activo. Además permite la sincronización automática de eventos, por ejemplo cuando un *pull request* es revisado, aprobado y *mergeado* a la rama base de desarrollo, la historia de usuario en Jira es transferida de forma automática del estado En Desarrollo a QA sin que los desarrolladores tengan que hacer el esfuerzo o inclusive recordar de actualizar las historias de usuario.

En segundo lugar, pero no menos importante, otra motivación que llevó a la utilización de esta herramienta fue la necesidad tanto de Ascentis como del grupo en mantener el código fuente en un ámbito privado, siendo ésta una funcionalidad gratuita provista por BitBucket y no por otras herramientas como por ejemplo GitHub que requiere de un plan no gratuito para contar con repositorios privados.

8.3 Organización de los repositorios

Esta sección tiene como propósito dar a conocer la forma en que se agruparon los artefactos previamente mencionados en los distintos repositorios.

8.3.1 Documentos

A continuación se presenta la disposición de directorios en la herramienta Google Drive utilizados para organizar los distintos documentos generados a lo largo del proyecto.

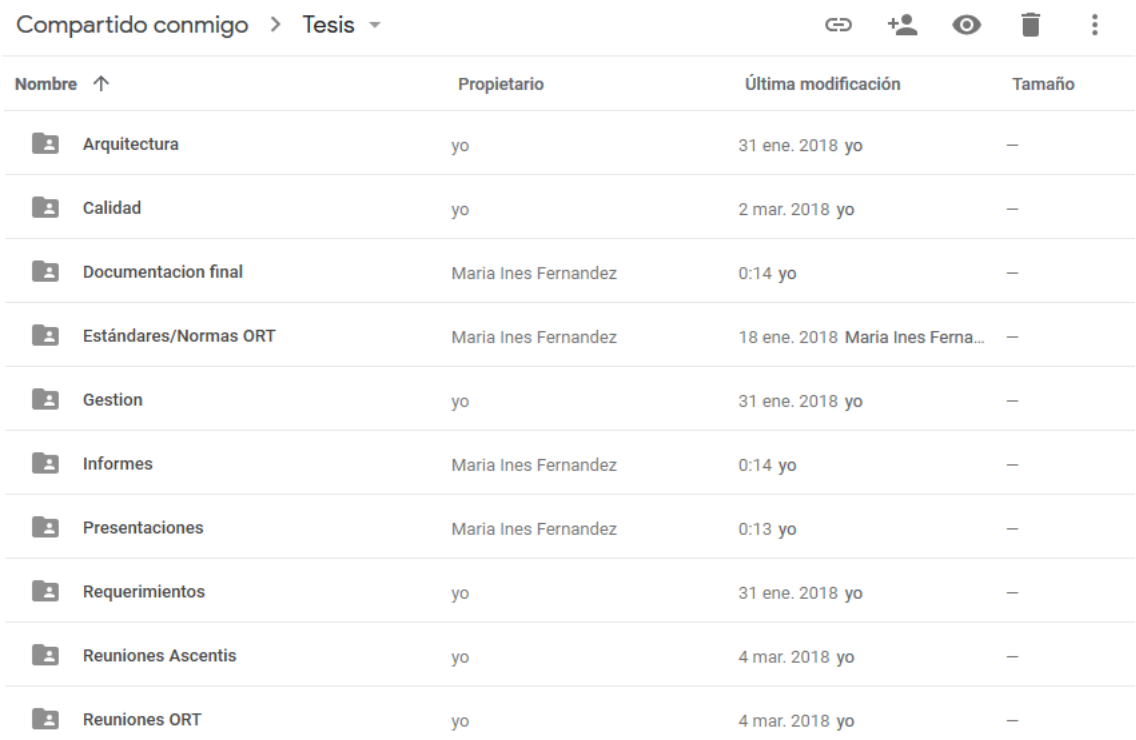


Ilustración 8.1 Repositorio de documentos

- **Arquitectura**: en esta carpeta se encuentra toda la información relacionada a la descripción de la solución de software, conteniendo distintos diagramas, comparaciones entre distintos enfoques arquitectónicos, justificación de distintas decisiones tomadas así como también el documento de arquitectura que se iba modificando a medida que se avanzaba en el desarrollo de Feedback Assistant.

- **Calidad**: aquí se pueden encontrar los documentos asociados a la calidad tanto del producto como del proceso. Se aloja la planilla de casos de prueba funcionales, métricas referentes a los defectos, las definiciones de calidad acordadas con el cliente, la trazabilidad de la satisfacción del cliente en el tiempo y el plan de calidad.

- **Documentación final**: en este directorio se ubicaron los distintos documentos formales que se mapean a cada capítulo del presente documento.

- **Estándares/Normas ORT**: los estándares y normas encontrados en el sitio Aulas que utilizó el equipo a lo largo del proyecto se alojaron en esta carpeta con el fin de no tener que visitar el sitio cada vez que se requería acudir a uno de ellos.

- **Gestión**: en esta carpeta se encuentra la información referente a la gestión de proyecto. Se encuentra la definición de la metodología utilizada, el plan de riesgos, métricas del seguimiento del trabajo realizado, registros de esfuerzo, plan de *releases* y el cronograma del proyecto.

- **Informes**: se alojaron en este directorio los distintos informes de avance generados para la Universidad ORT Uruguay.

- **Presentaciones**: todas las presentaciones creadas tanto para mostrar avances al cliente como revisiones de ORTsf se alojaron en esta carpeta.

- **Requerimientos**: se encuentra toda la información relacionada a la ingeniería de requerimientos como ser la especificación de requerimientos funcionales y no funcionales junto con el documento de validación entregado a Ascentis.

- **Reuniones Ascentis**: se alojaron todas las actas de reunión generadas luego de los encuentros con el cliente y documentos que se manejaban en las mismas.

- **Reuniones ORT**: se alojaron todas las notas de reunión generadas luego de los encuentros con el tutor y las revisiones de ORTsf.

8.3.2 *Software*

Se generaron ocho repositorios independientes a lo largo del proyecto, donde seis de ellos pertenecen a cada microservicio, otro para la biblioteca de clases conteniendo las entidades de negocio y por último uno asociado a la prueba de concepto realizada al inicio del proyecto.

A continuación se presenta a modo de ejemplo una ilustración para mostrar el contenido de uno de los repositorios y luego se detalla el propósito de cada uno.

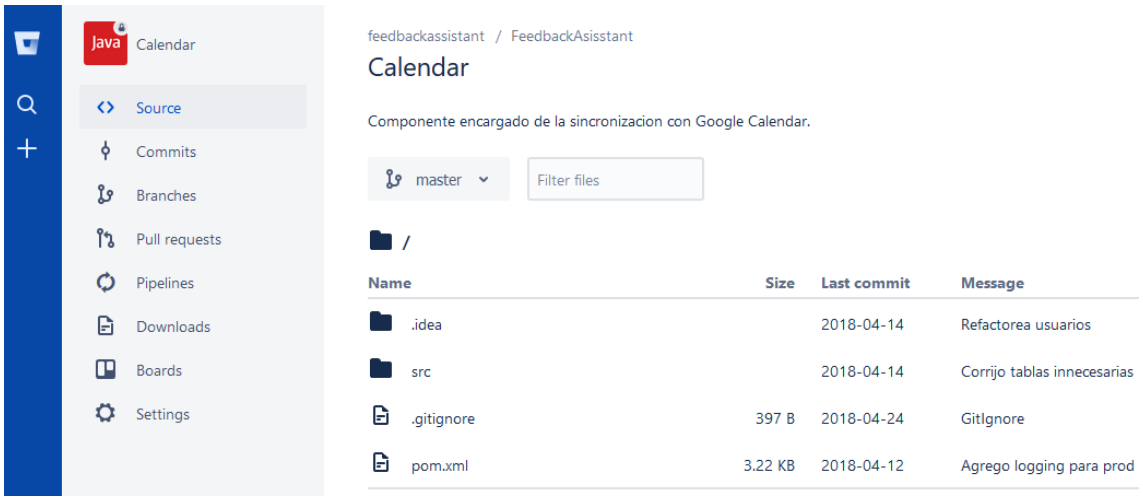


Ilustración 8.2 Repositorio Calendar en BitBucket

- **Calendar**: repositorio que contiene el código fuente y versionado del microservicio *calendar*.

- **Mailer**: repositorio que contiene el código fuente y versionado del microservicio *mailer*.

- **Feedback**: repositorio que contiene el código fuente y versionado del microservicio *feedback*.

- **Autodialer**: repositorio que contiene el código fuente y versionado del microservicio *autodialer*.

- **Directory**: repositorio que contiene el código fuente y versionado del microservicio *directory*.

- **Report**: repositorio que contiene el código fuente y versionado del microservicio *report*.

- **Model**: repositorio que contiene el código fuente y versionado de la biblioteca de clases *model*.

- **GSuite-POC:** repositorio que contiene el código fuente y versionado asociado a la prueba de concepto para la integración con Google G-Suite.

Política de Ramificación

El equipo debió definir los mecanismos de manejo del código fuente teniendo en cuenta el estado en el que se encontraba el mismo, pudiendo ser por ejemplo en estado de desarrollo, testing, listo para liberación o corrección de defecto/s en producción.

Se decidió utilizar el modelo de control de versiones denominado *GitFlow* [70]. Este flujo de trabajo define un modelo de ramificación estricto diseñado en torno a la versión del proyecto. Esto proporciona un marco robusto para la gestión de configuración de proyectos grandes.

A continuación se detalla el uso específico de cada rama dentro del modelo aplicado a Feedback Assistant:

- **master:** contiene las versiones estables del producto, es decir, la última versión que se encuentra en producción o en su defecto que en un futuro se encontrará en producción.

- **develop:** esta contiene las funcionalidades adicionales al último *release* alojado en master. Ésta es la rama base de desarrollo que luego será *mergeada* a *master* a la hora de liberar una nueva versión. En esta se realizaron por ejemplo las pruebas de regresión.

- **feature:** cuando se trabaja sobre una nueva funcionalidad o corrección de defecto se parte de la rama *develop* hacia una de éstas para no interferir con el código estable alojado en la rama base de desarrollo. Cuando se finaliza con el desarrollo en ésta, se crea un *pull request* solicitando *mergear* esta nueva funcionalidad (o en su defecto modificación) hacia la rama *develop*.

- **hotfix:** esta rama se utiliza cuando se detecta un defecto en producción. Ésta permite generar una corrección inmediata del defecto y luego *mergear* este cambio nuevamente a *master* (producción) y a su vez a *develop*.

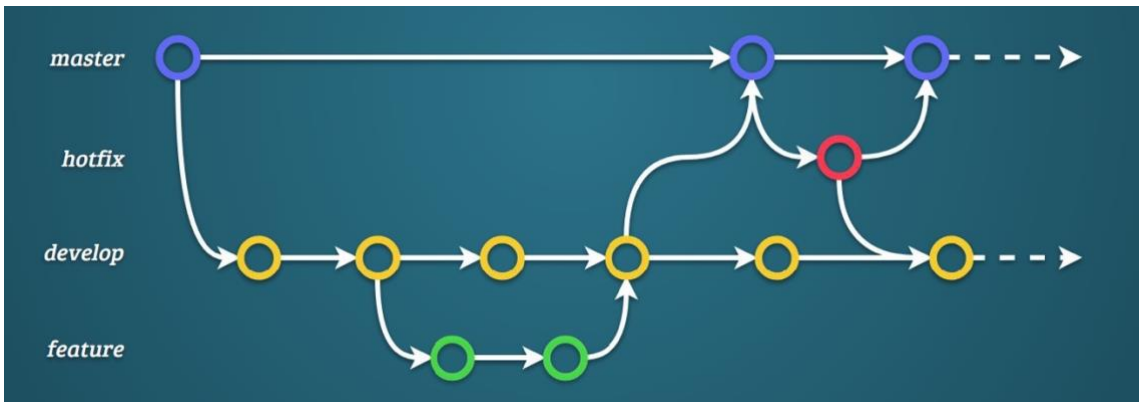


Ilustración 8.3 Modelo de ramificación para el control de versiones

A continuación se presenta una ilustración ejemplificando el uso de este modelo de trabajo en el proyecto Feedback Assistant.

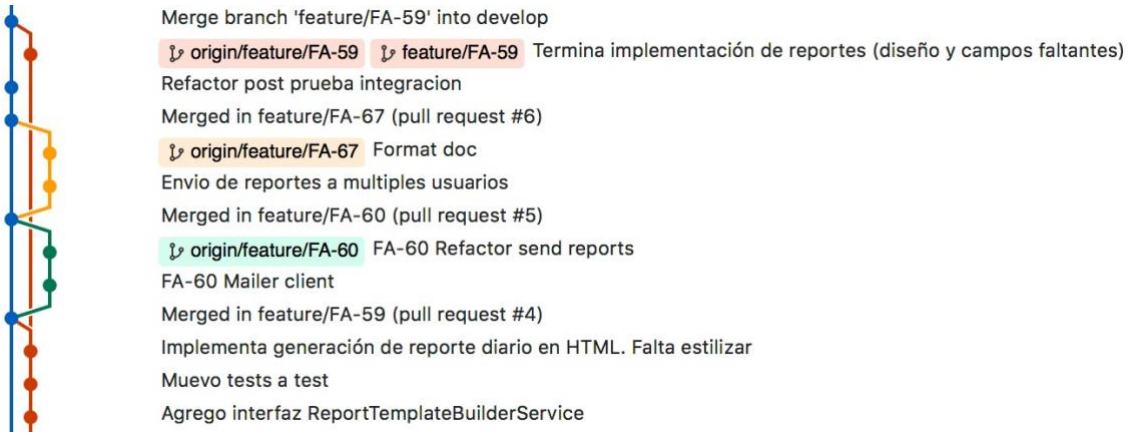


Ilustración 8.4 Flujo de desarrollo con GitFlow

8.4 Gestión de dependencias

Considerando que este es un proyecto que apunta a la integración con múltiples servicios externos, fue sumamente útil llevar a cabo una adecuada gestión de las dependencias.

Como fue mencionado en el capítulo 5. Análisis tecnológico (donde se detalla esta herramienta), se utilizó Apache Maven para la gestión de dependencias. Esta es una herramienta para la creación y gestión de proyectos Java, la cuál ya había sido utilizada por uno de los integrantes del equipo lo cual permitió la transferencia de su conocimiento al resto acerca de su utilización.

Esta herramienta no sólo permitió facilitar de gran manera el manejo de las distintas versiones de los microservicios, sino también mantener las bibliotecas de clase de las cuales los microservicios dependen fuera de éstos, utilizando un repositorio de dependencias local permitiendo sencillamente declarar su utilización en un archivo denominado POM (Project Object Model). Cuando se declarara una dependencia en el POM y no existe en el repositorio local, al ejecutar el comando `mvn install` se descargan las dependencias automáticamente y se alojan en el repositorio local. Esto ahorra una gran cantidad de tiempo y almacenamiento respecto a la inyección manual de las bibliotecas de clase dentro del proyecto.

8.5 Control de cambios

En esta sección se detallan las actividades involucradas en el proceso del control de cambios. Fue de suma importancia definir el flujo desde que algún interesado solicitó un cambio hasta que éste era agregado a la solución.

Se considera como cambio a toda funcionalidad no existente, es decir, no contemplada en el *product backlog,* así como también cualquier cambio en los requerimientos existentes. Vale aclarar que los defectos no se incluyen dentro de éstos siendo manejados por la gestión de defectos la cual se detalla en la sección 7.3.6 Gestión de defectos.

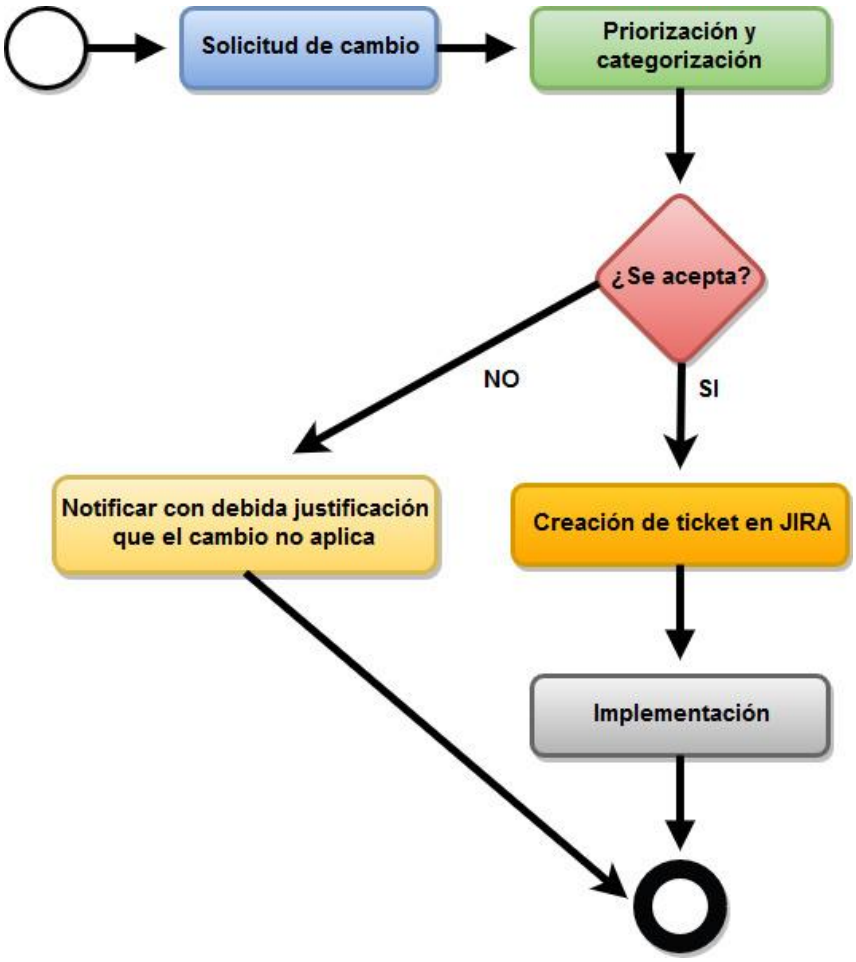


Ilustración 8.5 Flujo de actividades en el control de cambios.

El responsable del correcto cumplimiento de todas estas actividades fue el encargado de SCM. Como se puede observar en la ilustración, el flujo comienza con la solicitud de cambio generado por un interesado, pudiendo ser el cliente o alguno de los integrantes del equipo. Luego el encargado de SCM generaba una priorización y categorización del mismo teniendo en cuenta su magnitud. La siguiente actividad asociada a la aceptación o no del cambio se realizaba en las reuniones presenciales del equipo donde se discutía si era válido o no tomar en cuenta el cambio. Si se llegaba a un consenso de aceptar el cambio, el encargado de SCM procedía a crear el respectivo ticket en Jira agregando el cambio al *product backlog* siendo posteriormente implementado y llegando al final del flujo de este proceso. Si no se aceptaba el cambio, el encargado de SCM debía notificar al respectivo interesado que el cambio no aplicaba justificando con argumentos sólidos.

8.6 Conclusiones y lecciones aprendidas

Las actividades detalladas en la presente sección fueron de gran soporte durante todas las fases del proyecto. Principalmente generaron un gran aumento de la productividad debido a la excelente disposición de los artefactos generando una organización estable del trabajo.

Por otro lado remarcar cómo el uso de GitFlow para el manejo de las ramificaciones logró brindar alta seguridad a la hora de respaldar el código fuente del proyecto.

El uso de Maven logró facilitar enormemente el trabajo a la hora de manejar las distintas dependencias de cada microservicio tanto internamente en el ecosistema como hacia librerías de terceros. Previo a conocer esta herramienta se realizaba la inclusión de librerías manualmente, lo que en este proyecto podría haber sido una gran amenaza ante errores de versiones. Además facilitó enormemente a la hora de generar los artefactos a instalar en el servidor, definiendo el identificador y versión de los mismos en el POM respectivo a cada microservicio o librería.

Los integrantes del equipo pudieron adoptar conocimientos formales respecto a las actividades en el control de cambios. En el ámbito laboral de los integrantes este proceso no es formal por lo que el flujo de actividades no está definido y luego de formalizar este proceso se logró entender el por qué de las actividades y su respectivo orden.

9 Conclusiones

En este capítulo se exponen al lector las conclusiones asociadas a los objetivos establecidos en cuanto al proyecto y lo académico, dando a conocer también la reflexión de cada integrante en cuanto a los objetivos personales.

Adicionalmente se considera conveniente dar a conocer el estado actual en el que se encuentra el proyecto y dejar en claro cuáles serán los próximos pasos que se llevarán a cabo tanto a corto plazo como en un futuro.

9.1 Estado actual

Hoy en día se cuenta con un ambiente de producción estable, donde se encuentran seis microservicios distribuidos ejecutando en la infraestructura provista por Ascentis en el *datacenter* de sus oficinas en Uruguay.

Se siguen realizando reuniones de seguimiento ya que se decidió continuar trabajando en esta innovadora solución para agregarle aún más valor, comprometiéndose a un tercer *release* para el mes de octubre.

9.2 Conclusiones generales

En cuanto al proyecto y sus metas asociadas, no sólo se logró cumplir con la entrega del producto mínimo viable, sino que también el equipo fue capaz de generar ideas altamente novedosas logrando la entrega de una segunda versión con funcionalidades adicionales propuestas en mayor medida por el equipo. Esto demuestra el alto nivel de involucramiento por parte de los integrantes y cómo se buscaron alternativas para maximizar la satisfacción de Ascentis.

El equipo se encuentra orgulloso de haber podido cumplir con el objetivo de mantener altos niveles de satisfacción a lo largo del transcurso del proyecto, teniendo en cuenta que Ascentis es una compañía con una cultura distinta respecto a las empresas uruguayas y con altos niveles de calidad.

También es posible afirmar que se cumplió con el objetivo de llevar a cabo un proceso de ingeniería de software estandarizado, donde se pudo realizar una gestión y medición de los avances de una forma sumamente profesional.

En lo que refiere al producto, el equipo se encuentra realmente complacido con el trabajo realizado, el gran compromiso y dedicación involucrada por parte de todos los integrantes resultaron en un excelente producto de calidad donde existen intenciones de continuar trabajando en el marco del proyecto. Se ha logrado construir una solución adaptable a cualquier tipo de organización sin importar el sector industrial en la que se encuentre, donde se fomenta la comunicación interna en pro de mejorar el desempeño de sus miembros y las relaciones interpersonales. La autonomía de Feedback Assistant permite una fácil integración con los procesos internos y herramientas de cualquier organización, sin la necesidad de capacitación previa hacia los usuarios finales. Además, la arquitectura diseñada brinda la posibilidad de que Feedback Assistant sea totalmente escalable, pudiendo ser utilizado por organizaciones de pequeño, mediano y gran porte.

En lo que concierne a los objetivos académicos, el equipo se encuentra en condiciones de afirmar el cumplimiento de éstos. Se lograron aplicar conocimientos obtenidos en distintos dictados a lo largo de toda la carrera. Al momento de aprender nuevos conceptos en las distintas materias, solía hacerse enfocado en el contexto de ese mismo dictado, por lo que este proyecto final de carrera fue una gran oportunidad para integrar todos los conocimientos en una única instancia profesional. Un ejemplo de esto es cómo la aplicación de conocimientos del área ingeniería de *software* permitieron facilitar y realizar de mejor manera las actividades de arquitectura, diseño y desarrollo de *software*. Si bien aún no se tiene conocimiento del cumplimiento del objetivo asociado a la aprobación del proyecto final de grado como requisito final de carrera, el equipo se encuentra sumamente conforme con el trabajo realizado como se mencionó previamente.

Se distribuyeron las tareas de forma efectiva teniendo en cuenta las fortalezas de cada integrante y logrando compartir conocimiento entre los mismos en todo

momento. Cabe destacar la permanente colaboración y apoyo entre los miembros del equipo.

Cuando se suscitó un problema incapaz de ser resuelto por un único miembro del equipo, los otros miembros siempre se encontraron dispuestos, guiándolo por el camino correcto sin realizar el trabajo por él. Todo esto logra confirmar el cumplimiento del objetivo asociado a la autosuficiencia del equipo a la hora de afrontar este desafiante proyecto.

9.3 Conclusiones personales

Bruno

"No cabe duda alguna que esta instancia de proyecto final fue una de las más desafiantes y provechosas de toda la carrera. Tuve la oportunidad de aplicar los conocimientos técnicos, gerenciales y sociales que fui incorporando a lo largo de estos años en un escenario real donde tanto el cliente como la Universidad ORT Uruguay esperaban lo mejor, hecho que me motivó a cumplir y superar las expectativas de todos los interesados.

Encontré en las tecnologías del procesamiento de lenguaje natural un sinfín de posibilidades que aún no han sido explotadas, y que se limitan únicamente por la creatividad de desarrolladores como nosotros para crear soluciones que cambien nuestra forma de trabajar.

Quiero destacar la buena sincronía que se formó entre los integrantes de mi equipo y que fortaleció aún más la amistad que mantuvimos desde un inicio. Nicolás y María se han destacado como profesionales y amigos, y espero que podamos volver a trabajar juntos en futuros proyectos."

Nicolás

"Luego de haber finalizado este ciclo de proyecto puedo confirmar que fue la experiencia más enriquecedora de la carrera, no sólo en cuanto a conocimientos técnicos, sino también a la autogestión que demandó y la química positiva que se fue generando con el tiempo para sobrellevar los distintos problemas que surgieron.

Creo que la parte humana es muy importante en estos proyectos, ya que las relaciones interpersonales son la base para poder enfrentarse a un proyecto de este tipo. Si el equipo cuenta con gran conocimiento pero no existe una idea de grupo y colaboración los resultados no serán tan beneficios como podrían serlo.

En mi opinión considero que fuimos sumamente profesionales tanto con el cliente como con el tutor y revisores de la Universidad ORT Uruguay. Estoy realmente orgulloso del grupo que se formó, ya que fuimos capaces de complementar los conocimientos entre todos, autoevaluarnos y darnos soporte cuando era necesario.

Además logramos satisfacer al cliente aportando gran valor al nuevo proceso que desea implantar en su mercado objetivo, lo que me genera una enorme satisfacción personal."

María Inés

"Considero que este proyecto ha sido muy enriquecedor tanto en lo académico como en lo personal. La duración del mismo y el hecho de haber tenido un cliente real hicieron que este proyecto adquiera una dimensión profesional. Aprendimos a comunicarnos y a trabajar fluidamente con un cliente del exterior con otra cultura y lenguaje lo que nos generó un gran aprendizaje.

A su vez adquirimos nuevos conocimientos tanto en la gestión de los recursos humanos como en nuevas tecnologías. La gran diferencia fue que todo esto se aprendió fuera del salón de clase y a través de investigación. Esta experiencia se acerca más a un escenario real, en el cual vamos a interactuar en nuestra vida profesional.

Por último me llevo como mayor aprendizaje el trabajo en equipo. Como grupo fuimos capaces de sobrellevar los distintos desafíos a los que nos enfrentamos basándonos en el diálogo constante y el trabajo en conjunto."

9.4 Lecciones aprendidas

Además de la aplicación de los distintos conocimientos obtenidos a lo largo de la carrera, se han podido interiorizar una gran cantidad de nuevos conocimientos.

Una de las grandes enseñanzas que dejó este proyecto fue llevar a la práctica distintos procesos de gestión en un proyecto real, como por ejemplo la planificación a largo plazo, la utilización de procesos de soporte (como ser SCM, gestión de riesgos, etc.) y el trabajo dirigido por un plan de calidad. En los distintos dictados se han tratado todos estos conceptos en proyectos de tiempo reducido y en el ámbito laboral nunca se tuvo la responsabilidad de dirigir la gestión de los proyectos en los que se trabajó. Esto ocasionó que el equipo se enfrentara a distintos problemas de planificación donde se tuvieron que tomar acciones correctivas para sobrellevarlos y así se aprendió a planificar de mejor manera. La segmentación de historias de usuario en subtareas más granulares junto con los *spikes* al final de los *sprint* son algunos de los ejemplos de aprendizajes en la gestión de proyectos.

También se adquirieron grandes conocimientos en cuanto a aspectos arquitectónicos, donde los integrantes aplicaron por primera vez el patrón arquitectónico basado en microservicios desde las primeras fases del proyecto, sin partir de un enfoque monolítico como se realizó en los dictados donde se trataron los microservicios. Además, se debieron manejar múltiples dependencias tanto hacia terceros como dentro del sistema, donde se incursionó sobre la herramienta Maven y la gran facilidad que provee ésta a la hora de crear los proyectos, manejar los versionados y relaciones entre los mismos.

La utilización de servicios de procesamiento de lenguaje natural permitió al equipo entender los conceptos involucrados en esta rama de la inteligencia artificial, específicamente en el análisis del sentimiento de un texto. Previo al proyecto no se tenía conocimiento que para realizar este análisis se debía someter el texto proporcionado a una serie de pasos, generando un árbol sintáctico y luego analizando a partir de éste su estructura para obtener el sentimiento del mismo. A pesar de no implementarlo el equipo se interesó en el proceso que llevan a cabo

estos servicios para entender lo que sucede internamente durante el análisis del sentimiento.

En el dictado de Ingeniería de Software Ágil se trataron las pruebas de concepto como mecanismo de demostración de viabilidad respecto a la puesta en práctica de una teoría, pero ningún integrante del equipo había realizado una antes previo a la realización de Feedback Assistant. Llevando a cabo este mecanismo ágil en el proyecto se entendió que es de gran utilidad para entender determinado problema desconocido, así como también validar si el mecanismo utilizado para la misma es viable dentro de una solución, llevando a la práctica este concepto que únicamente se poseía de forma teórica. Además se aprendió empíricamente que la realización de una prueba de concepto es una vía de mitigación de riesgos tecnológicos como sucedió con la impersonalización de usuarios en la plataforma Google G-Suite.

9.5 Próximos pasos

En la presente sección se procede a detallar los futuros pasos del proyecto, dando a conocer tanto lo que refiere al producto en un corto plazo así como lo que el equipo espera de Feedback Assistant en un futuro.

Como fue mencionado previamente, el equipo se comprometió a continuar trabajando en Feedback Assistant agregando nuevas funcionalidades acordadas con el cliente. Por ésto, el primero de octubre se realizará un tercer *release* el cuál se trabajará con una modalidad *date-driven*. Aquí se dará una clausura formal del proyecto haciendo entrega no sólo del código fuente (mínimo requisito exigido por Ascentis) sino también de manuales de instalación y configuración en conjunto con la planilla de casos de prueba funcionales y documento de arquitectura.

Para trabajar en éste último período fue necesario mantener contacto con el *product owner* con el fin de llevar a cabo una priorización de las funcionalidades. A continuación se listan los requerimientos incluidos para este último *release* ordenados por prioridad:

- **RF18:** Solicitud de *feedback* sobre cierto tema recurrente.
- **RF16:** Sincronizar usuarios de la plataforma Office 365.
- **RF17:** Sincronizar eventos desde la plataforma Office 365.
- **RF19:** Limitar solicitudes de *feedback* a ciertas áreas de la organización.

Más allá de la construcción del presente documento y posterior preparación de la defensa final donde se hará una presentación del proyecto junto con una demo, el equipo se compromete a dar lo máximo de sí mismo utilizando los nuevos aprendizajes adquiridos a lo largo del proyecto, con el fin de lograr cumplir con la mayor cantidad de funcionalidades restantes posibles.

En cuanto a un futuro más lejano, el equipo espera que Feedback Assistant logre cambiar los procesos internos de las organizaciones clientes de Ascentis de manera positiva. A continuación se presenta el proceso que desea implementar el cliente dentro de su mercado.

Ilustración 9.1 Proceso New Performance Tuning para el manejo de talentos.

Ascentis quiere eliminar el concepto de evaluación de desempeño ya sea anual o semestral, apuntando a evaluaciones periódicas a lo largo del año. Para esto es de vital importancia la retroalimentación o *feedback* continuo internamente dentro de la organización. Por eso Feedback Assistant facilita enormemente esta búsqueda de *feedback* continuo y se espera que el paso asociado a "*On-going Feedback"* sea posible cumplirlo con la solución creada en este proyecto.

Se espera que esta solución integrada a la suite de productos para la gestión del capital humano genere una innovación disruptiva en el mercado en el cual compite Ascentis, cambiando por completo el enfoque de evaluación de desempeño a un ámbito mucho más objetivo y certero al utilizado actualmente.

10 Referencias bibliográficas

[1] P. G. Rivero, "¿Cambiará la Inteligencia Artificial el ejercicio del liderazgo?", Madrid, España: APD Asociación Para El Progreso De La Dirección, no 328, pp. 18–19, 2017.

[2] Universidad ORT Uruguay, "Laboratorio ORT Software Factory", [Online]. Available: https://fi.ort.edu.uy/3393/17/inicio.html. Accessed on: Ago. 17, 2018.

[3] Google, "G Suite: Gmail, Documentos, Drive y Calendar para empresas". [Online]. Available: https://gsuite.google.es/intl/es/. Accessed on: Ago. 08, 2018.

[4] "G Suite: Gmail, Documentos, Drive, Calendar y otras aplicaciones para empresas". [En línea]. Disponible en: https://gsuite.google.es/intl/es/. [Accedido: 08-ago-2018].

[5] Salesforce, "Soluciones CRM On Demand". [Online]. Available: https://www.salesforce.com/es/. Accessed on: Ago. 08, 2018.

[6] A. Mejía-Giraldo, M. Bravo-Castillo, y A. Montoya-Serrano, "El factor del talento humano en las organizaciones", Caliz, COL: Universidad de San Buenaventura, 2013. vol. 34, no 1, pp. 2–11.

[7] R. Rutz, "New Trends in Performance Management for Happier, More Engaged Workforce". [Online] Available: https://www.ascentis.com/ascentis-different-than-traditional-reviews/. Accessed on: Ago. 04, 2018.

[8] Ascentis, "Talent Management Software – Ascentis Solutions". [Online] Available: https://www.ascentis.com/solutions/ascentis-talent-management-system/. Accessed on: Ago. 06, 2018.

[9] Google, "Business Collaboration Solutions - G Suite". [Online]. Available: gsuite.google.com/learn-more/google-apps-datasheet.html. Accessed on: Jul. 28, 2018.

[10] Google, "Gmail: correo electrónico y almacenamiento gratuitos de Google". [Online]. Available: http://www.google.com/gmail/about/. Accessed on: Ago. 11, 2018.

[11] Google, "Google Calendar: programación para empresas y calendario online". [Online]. Available: https://gsuite.google.es/intl/es/products/calendar/. Accessed on: Ago. 11, 2018.

[12] A. C. Vásquez y A. M. Huayna, “Procesamiento de lenguaje natural”, Lima, Perú: UNMSM, 2009. vol. 6, no 2, pp. 45–54. [Online] Available: http://revistasinvestigacion.unmsm.edu.pe/index.php/sistem/article/view/5923. Accessed on: Ago. 13, 2018.

[13] Screencast, “TMS”. [Online] Available: https://www.screencast.com/users/Ascentis.Discovery/playlists/New%20Playlist(7)?mediaSetId=af1e2b0e-b165-4250-8952-df7525782179. Accessed on: Aug. 19, 2018.

[14] Buzachis Aris, “Microservices vs Monolithic architectures”. [Online] Available: https://blog.buzachis-aris.com/2014/12/microservices-vs-monolithic-architectures/. Accessed on: Ago. 01, 2018

[15] P. Clements, “Documenting Software Architectures: Views and Beyond”, 2 ed. Boston, MA, USA: Addison-Wesley, 2010.

[16] C. Larman y B. Moros Valle, “UML y patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado”. Upper Saddle River, NJ, USA: Prentice Hall, 2002, p. 201.

[17] R. C. Martin, “Clean Architecture: A Craftsman's Guide to Software Structure and Design”, Upper Saddle River, NJ, USA: Prentice Hall.

[18] SourceMaking, “Chain of Responsibility”. [Online] Available: https://sourcemaking.com/design_patterns/chain_of_responsibility. Accessed on: Ago. 05, 2018.

[19] Ira R. Forman y Nate Forman, “Java Reflection in Action”. Greenwitch, CT, USA: Manning Publications, 2004, p. 26.

[20] H. Valdecantos, “Principios y patrones de diseño de software en torno al patrón compuesto Modelo Vista Controlador para una arquitectura de aplicaciones interactivas.”. Tucuman, ARG: UNT, 2010. p. 45.

[21] IBM, “ACID Properties”. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.4.0/product-overview/acid.html. Accessed on: Ago. 06, 2018.

[22] Pivotal, “5. The IoC container”. [Online]. Available: https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/beans.html. Accessed on: Ago. 05, 2018.

[23] Pivotal, “1. Introduction to Spring Framework”. [Online]. Available: https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/overview.html. Accessed on: Ago. 05, 2018.

[24] The Apache Software Foundation, “Apache ActiveMQ”. [Online]. Available: https://activemq.apache.org/. Accessed on: Ago. 05, 2018.

[25] The Apache Software Foundation, “Apache Maven”. [Online]. Available: https://maven.apache.org/. Accessed on: Ago. 05, 2018.

[26] The Apache Software Foundation, “Apache Tomcat” [Online]. Available: http://tomcat.apache.org/. Accessed on: Ago. 05-, 2018.

[27] Google, “Seguridad de G Suite”. [Online] Available: https://gsuite.google.com/security/. Accessed: Ago. 05, 2018.

[28] Google, “G Suite Pricing Plans”. [Online] Available: https://gsuite.google.com/pricing.html. Accessed on: Ago. 04, 2018.

[29] Google, “Cloud Identity and Access Management Documentation”. [Online] Available: https://cloud.google.com/iam/docs/service-accounts. Accessed on: Ago. 05, 2018.

[30] OAuth, “OAuth 2.0”. [Online] Available: https://oauth.net/2/. Accessed on: Ago. 05, 2018.

[31] Microsoft, “Text Analytics API”. [Online]. Available: https://azure.microsoft.com/en-us/services/cognitive-services/text-analytics/. Accessed on: Ago. 06, 2018.

[32] IBM, “Watson Natural Language Understanding”. [Online]. Available: https://www.ibm.com/watson/services/natural-language-understanding/. Accessed on: Ago. 06, 2018.

[33] The Apache Software Foundation, “Apache Maven”. [Online]. Available: https://maven.apache.org/what-is-maven.html. Accessed on: Ago. 06, 2018.

[34] Oracle, “Java Persistence API”. [Online]. Available: http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html. Accessed on: Ago. 06, 2018.

[35] Red Hat, “Everything data”. [Online]. Available: http://hibernate.org/. Accessed on: Ago. 06, 2018.

[36] Oracle, “Entities - Java EE 6”. [Online]. Available: https://docs.oracle.com/javaee/6/tutorial/doc/bnbqa.html. Accessed on: Ago. 06, 2018.

[37] Oracle, “MappedSuperclass (Java EE 5 SDK)”. [Online]. Available: https://docs.oracle.com/javaee/5/api/javax/persistence/MappedSuperclass.html. Accessed on: Ago. 06, 2018.

[38] Twilio, “What is Cloud Communications”. [Online] Available: https://www.twilio.com/what-is-cloud-communications. Accessed on: Ago. 06, 2018.

[39] Twilio, “Built with Twilio - Customer Stories”. [Online]. Available: https://customers.twilio.com/. Accessed on: Ago. 12, 2018.

[40] Scrum.org, “What is Scrum?”. [Online]. Available: https://www.scrum.org/resources/what-is-scrum. Accessed on: Ago. 05, 2018.

[41] Scrum.org, “The Scrum Framework Poster”. [Online]. Available: https://www.scrum.org/resources/scrum-framework-poster. Accessed on: Ago. 12, 2018.

[42] Atlassian, “Organize anything, with anyone”. [Online]. Available: https://trello.com. Accessed on: Ago. 05, 2018.

[43] LeanKit, “What is Kanban?”. [Online]. Available: https://leankit.com/learn/kanban/what-is-kanban/. Accessed on: Ago. 05, 2018.

[44] Atlassian, “Jira | Software de seguimiento de proyectos e incidencias”. [Online]. Available: https://es.atlassian.com/software/jira. Accessed on: Ago. 05, 2018.

[45] Atlassian, “Software Development and Collaboration Tools”. [Online]. Available: https://www.atlassian.com/. Accessed on: Ago. 05, 2018.

[46] J. W. Grenning, “Planning Poker or How to avoid analysis paralysis while release planning”, Hawthorn Woods, IL, USA: Wingman Software, 2002. [Online]. Available: https://wingman-sw.com/papers/PlanningPoker-v1.1.pdf

[47] International Scrum Institute, “Scrum Burndown Chart”. [Online]. Available: https://www.scrum-institute.org/Burndown_Chart.php. Accessed on: Ago. 05, 2018.

[48] Toggl, “Free Time Tracking Software”. [Online]. Available: https://toggl.com/. Accessed on: Ago. 05, 2018.

[49] HR Virginia, "Writing S.M.A.R.T. Goals". [Online] Available: http://www.hr.virginia.edu/uploads/documents/media/Writing_SMART_Goals.pdf Accessed on: Ago. 18, 2018.

[50] S. Matalonga, A. Villar, C. Nacimiento, “Deuda técnica: ¿Cuáles son los límites de la metáfora?”, Montevideo, UY: Universidad ORT Uruguay, 2013. [Online]. Available: https://www.ort.edu.uy/fi/pdf/documento10fi.pdf.

[51] Likert, R. “A Technique for the Measurement of Attitudes”. New York, NY, USA: R. S. Woodworth, 1932. Asociación Española Para La Calidad, “Plan de calidad”. [Online]. Available: https://www.aec.es/web/guest/centro-conocimiento/plan-de-calidad. Accessed on: Ago. 12, 2018.

[52] “AEC - Plan de calidad”. [En línea]. Disponible en: https://www.aec.es/web/guest/centro-conocimiento/plan-de-calidad. [Accedido: 12-ago-2018].

[53] Oracle, “Code conventions for the Java programming language.” [Online]. Available: http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html. Accessed on: Ago. 12, 2018.

[54] R. C. Martin, “Clean Code: A Handbook of Agile Software Craftsmanship”. Boston, MA, USA: Prentice Hall, 2008.

[55] SonarQube, “Continuous Inspection”. [Online]. Available: https://www.sonarqube.org/. Accessed on: Ago. 12, 2018.

[56] Universidad ORT Uruguay, "Normas específicas para la presentación de trabajos finales de carrera". [Online]. Available: http://fi.ort.edu.uy/innovaportal/file/30964/3/documento-302-escuela-de-tecnologia.pdf. Accessed on: Ago. 12, 2018.

[57] Universidad ORT Uruguay, "Hoja de verificación pautas de presentación de trabajos finales de carrera". [Online]. Available: http://www.ort.edu.uy/facs/pdf/documento303carrerascortasfacs.pdf. Accessed on: Ago. 12, 2018.

[58] Universidad ORT Uruguay, "Normas para el desarrollo de trabajos finales de carrera". [Online]. Available: http://www.ort.edu.uy/varios/pdf/documento304.pdf. Accessed on: Ago. 12, 2018.

[59] Universidad ORT Uruguay, "Orientación para títulos, resúmenes o abstracts e informes de corrección de trabajos finales de carrera". [Online]. Available: http://www.ort.edu.uy/varios/pdf/documento306.pdf. Accessed on: Ago. 12, 2018.

[60] Universidad ORT Uruguay, "Pautas generales de formato de trabajos finales". [Online]. Available: http://fa.ort.edu.uy/innovaportal/file/39750/1/pautas-generales-de-formato-de-trabajos-finales.pdf. Accessed on: Ago. 12, 2018.

[61] P. Clements, "Documenting Software Architectures: Views and Beyond", 2 ed. Boston, MA, USA: Addison-Wesley, 2010.

[62] Universidad ORT Uruguay, "ORTsf Plantilla del Plan de la Calidad". [Online]. Available: https://aulas.ort.edu.uy/pluginfile.php/89022/mod_folder/content/0/SQA/PL-SQA-02-Plan_de_la_Calidad-V1.0.doc. Accessed on: Ago. 12, 2018.

[63] V. Massol y T. Husted, "JUnit in action". Greenwich, CT, USA: Manning, 2003.

[64] IEEE Standards Association, "IEEE standard classification for software anomalies". New York, NY, USA: Institute of Electrical and Electronics Engineers, 2002.

[65] ITU-T. "The International Public Telecommunication Numbering Plan". Ginebra, Suiza: ITU-T, 2010.

[66] SonarQube, "Metric Definitions". [Online]. Available: https://docs.sonarqube.org/display/SONAR/Metric+Definitions#MetricDefinitions-Maintainability. Accessed on: Ago. 18, 2018.

[67] Atlassian, "Confluence - Team Collaboration Software". [Online]. Available: https://www.atlassian.com/software/confluence. Accessed on: Ago. 18, 2018.

[68] Git, "Git SCM". [Online]. Available: https://git-scm.com/. Accessed on: Ago. 18, 2018.

[69] Atlassian, "Bitbucket | The Git solution for professional teams". [Online]. Available: https://bitbucket.org/product. Accessed on: Ago. 13, 2018.

[70] Atlassian, "Gitflow Workflow". [Online]. Available: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow. Accessed on: Ago. 13, 2018.

[71] SonarSource, "Java Static Code Analysis Source". [Online]. Available: https://rules.sonarsource.com/java. Accessed on: Ago. 08, 2018.

[72] R. C. Martin, “Clean Architecture: A Craftsman's Guide to Software Structure and Design”, Boston, MA, USA: Prentice Hall, 2017.

[73] Apache, “Apache Tomcat” [Online]. Available: http://tomcat.apache.org/. Accessed on: Ago. 05, 2018.

[74] Microsoft Azure, “Text Analytics API”. [Online]. Available: https://azure.microsoft.com/en-us/services/cognitive-services/text-analytics/. Accessed on: Ago. 06, 2018.

[75] IBM, “Watson Natural Language Understanding”. [Online]. Available: https://www.ibm.com/watson/services/natural-language-understanding/. Accessed on: Ago. 06, 2018.

11 Anexos

11.1 Investigación del producto TMS

Se realizó una etapa de investigación del producto a integrarse con el fin de entender a qué apunta el cliente y de esta forma lograr aportar el mayor valor posible al mismo.

Ascentis Talent Management ofrece un conjunto de potentes herramientas que incluyen administración de rendimiento, aprendizaje y planificación de compensación. El cliente envió al equipo el *link* asociado a un *webinar* donde se exponen las principales funcionalidades y los principales objetivos de negocio. A continuación se muestran algunas capturas de este *webinar* con las principales ideas del sistema.

Ilustración 11.1.1 Introducción al *webinar* de TMS

Con este sistema de gestión del talento, los gerentes pueden identificar fácilmente qué hace que sus mejores empleados sean tan exitosos y aprovechar esas habilidades para ayudar a otros, enfocarse en los empleados de nivel medio para ayudarlos a acelerar sus capacidades y competencias, mientras se toman medidas sobre las brechas críticas que están impidiendo que sus trabajadores menos performantes logren su potencial.

Ilustración 11.1.2 Proceso *New Performance Tuning* para el manejo de talentos.

Performance Tuning es el nuevo proceso de gestión de talentos que desea implantar Ascentis en su mercado objetivo. Este nuevo enfoque implica una interacción periódica entre los empleados y asesores de talento, donde se definen objetivos a corto y largo plazo. Luego mediante el *feedback* continuo se obtienen métricas acerca de estos objetivos (aquí es cuando entra en juego la funcionalidad de *Feedback Assistant)*. El *feedback* continuo permite ajustar el desempeño si es necesario para luego realizar una instancia formal de revisión de la *performance*. Esto permitirá crecer a los empleados y expondrá a los que se destaquen para lograr llevar la organización al siguiente nivel. Además se buscan eliminar las evaluaciones anuales realizadas sin datos objetivos y certeros en los cuales volcarse.

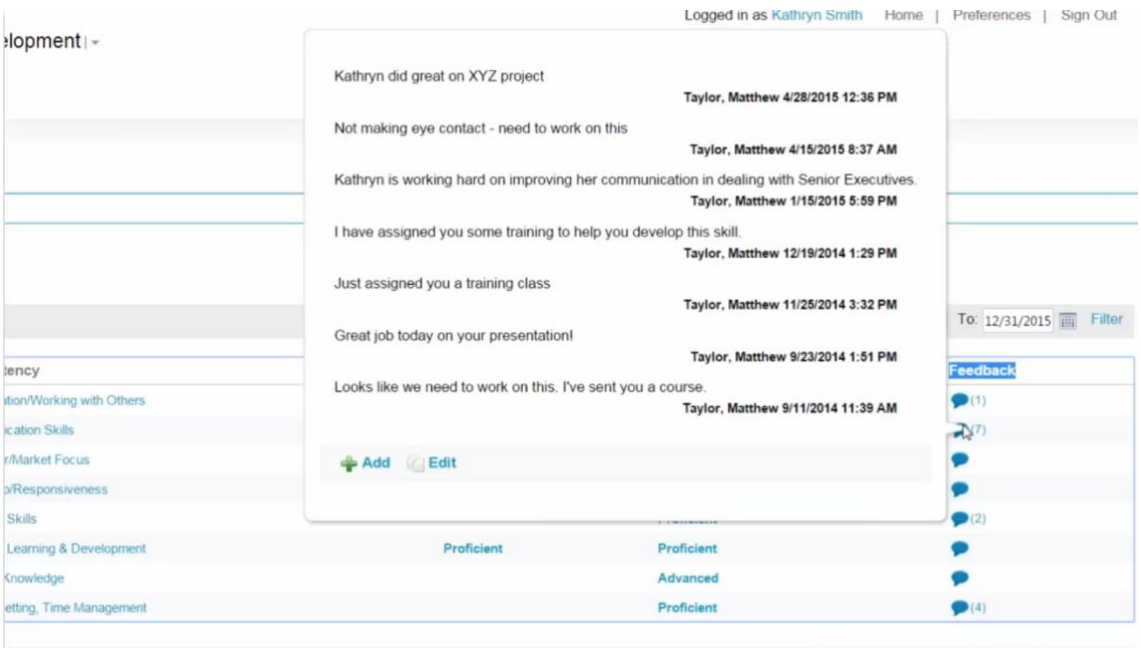


Ilustración 11.1.3 Listado de *feedbacks* para un objetivo seleccionado.

A su vez, los empleados tienen la capacidad de actualizar sus logros y calificarse a sí mismos u otros dando *feedback* en la sección de administración de talentos, brindando oportunidades continuas para influir positivamente en sus revisiones. En estas ilustraciones vemos la sección del sistema TMS en donde el usuario visualiza todos los *feedbacks* asociados a un objetivo en particular y por quién fue provisto con la posibilidad de agregar un nuevo *feedback* si se desea. Como se puede observar la herramienta no posee una interfaz de usuario amigable y además el usuario debe autenticarse al sistema y seguir una serie de pasos para lograr proveer *feedback* ya sea a sí mismo u otra persona, lo que da la pauta de que la experiencia de usuario tampoco es muy favorecida. Por esto creemos que Feedback Assistant será un gran complemento a este sistema.

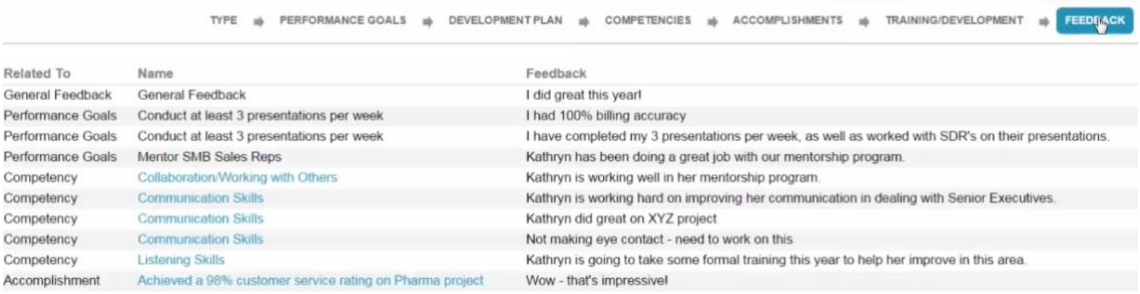


Ilustración 11.1.4 Listado de *feedbacks* asociados a objetivos en TMS.

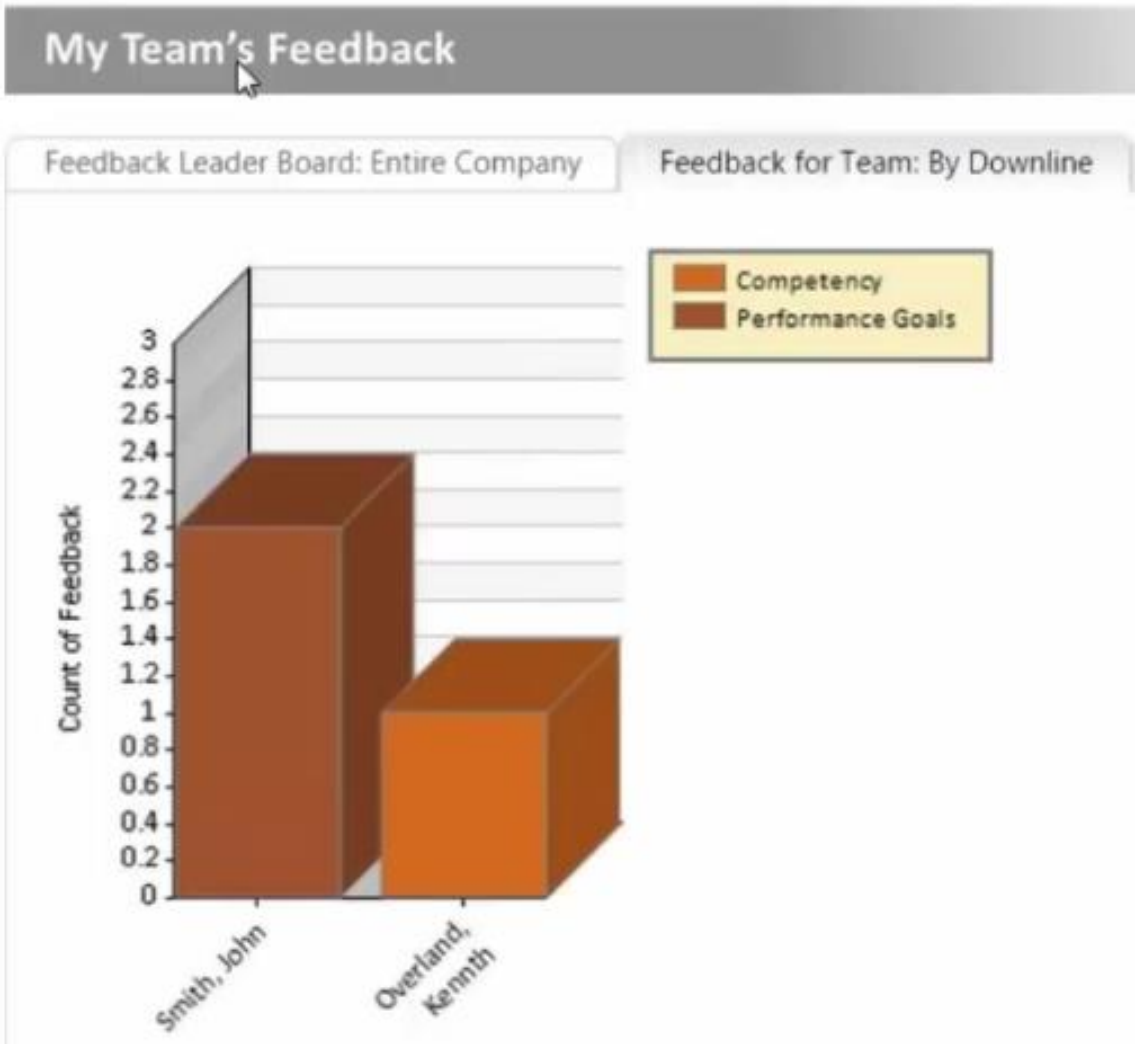


Ilustración 11.1.5 Gráfico cantidad de *feedback* por persona TMS.

En la herramienta, los asesores de talento (o *managers*) son capaces de visualizar mediante distintos gráficos la cantidad de *feedbacks* provistos para cada individuo del equipo o en su defecto múltiples equipos que éste lidere. Esto permite tener información cuantitativa acerca de la retroalimentación continua que fluye dentro de la organización. Aquí se hace tangible la necesidad de Ascentis en qué Feedback Assistant exija el mínimo esfuerzo posible al usuario para proveer *feedback*. Cuanto menos invasiva y más intuitiva sea la herramienta a desarrollar, se espera que los valores de éstos gráficos asociados a la cantidad de *feedback* provisto en TMS sean mayores a los actuales.

En las siguientes ilustraciones se presentan las principales funcionalidades volcadas a los asesores de talento así como también a los empleados. Esto permitió al equipo entender en profundidad lo que se busca con esta herramienta y la información que se maneja para evolucionar el desempeño de los individuos de la organización, es decir, generar una mejora continua en el activo más importante de una organización, el capital humano.

Ilustración 11.1.6 Funcionalidades del asesor de talento en TMS.

Ilustración 11.1.7 Funcionalidades del empleado en TMS.

11.2 Documento de validación de requerimientos

A continuación se presenta el documento entregado al cliente el cual contiene los requerimientos relevados con el fin de que éste lo apruebe y así poder comenzar con las etapas de desarrollo teniendo el *product backlog* validado.

Feedback Assistant Requirements

Functional Requirements

1) **Integration with Google Directory API**
   The system should be able to synchronize with the Google G-Suite platform in order to use the employees data under the organization interacting with Google Directory API.

2) **Integration with Google Calendar API**
   The system should be able to synchronize with the Google Calendar API in order to trigger actions when an event of a specific user finishes.

3) **Ask for feedback by sending email.**
   By synchronizing with the Google Calendar API, the system should detect when an event finishes, sending an email to all the attendees of the event asking for feedback.

4) **Receive feedback by email**
   The system should be able to receive an email with feedback, that should be parsed to a common internal format, so that it can be processed and then stored.

5) **Processing feedback**

Processing feedback consists of a group of steps which can be configurable on runtime.

The steps are:

a. **Google Natural Language API analysis**

Use Google Natural Language API service to evaluate the general sentiment of the feedback. The sentiment will be categorized into one of five categories:

- Very positive
- Positive
- Neutral
- Negative
- Very negative

b. **Notify the TMS system**

The system will send the processed feedbacks to the TMS application by invoking a rest API provided by Ascentis.

c. **Persist feedback**

At last, the system will persist the processed feedback in a database so that it can be used in the future for reporting purposes.

6) **Reports**

The system should generate daily, weekly and monthly digest in order to send it to configurable managers accounts via email.

The report should contain, for each feedback processed:

- Provider name, surname and email
- Event name
- Content
- Sentiment

7) **Frontend dashboard**

   The system should count with a web application in order to visualize the activity (feedback processed, average qualification of a employee, reports, etc.)

8) **Search engine**

   In the web application, the user should be able to search for a feedback by different criteria: date, entities, language, ranking, user, content, etc.

Non Functional Requirements

1) **Ability to execute on any environment (Portability)**

   The system should be able to execute on any possible environment in order to adapt to the client's infrastructure, regardless the operative system or hardware involved.

2) **Put up with new third-party services (Extensibility & Maintenance)**

   The system should have an architecture and design which allows an easy addition of new third party services integration. In future stages of the project, the system should be able to communicate with different third-party services simultaneously, retrieving feedback from the different platforms.

3) **Easily change the processing steps of the feedback (Maintenance & Availability)**

   Maintenance is one of the most important attributes to favor in this project. Effort of changing a step on the feedback processing should be simple. Having a configurable list of steps allows the developer to change the processing of the feedback on runtime which not only favors maintenance but also availability.

4) **Almost zero effort by the user (Usability)**

   One of the most important things to consider is how to communicate with the end user. The system should be able to ask for feedback in the least annoying way for the user, considering that the user will not be forced to send feedback. It is crucial to minimize the effort of the end user when providing feedback.

5) **Effective error management (Availability & Auditability)**

   The system should have a detailed and organized management of the errors, being able to know when, where, why and how an error occurred. In addition, it should alert the user when an error occurs and the cause of it.

6) **Ability to manage large quantity of users and events. (Scalability)**

   Feedback Assistant will be used by companies with a number of employees between 100 and 500.

7) **Sentiment analysis independent of the feedback language (Usability)**

   The system should be able to analyze the sentiment, regardless the language.

11.3 Presentación de prueba de concepto al cliente

En este anexo se expone la presentación realizada al cliente el día 2 de Febrero del 2018. El objetivo fue dar a conocer la prueba de concepto realizada con el fin de validar la viabilidad de ciertos aspectos de la solución, así como también confirmar que lo que se iba a desarrollar era lo que Ascentis realmente esperaba.

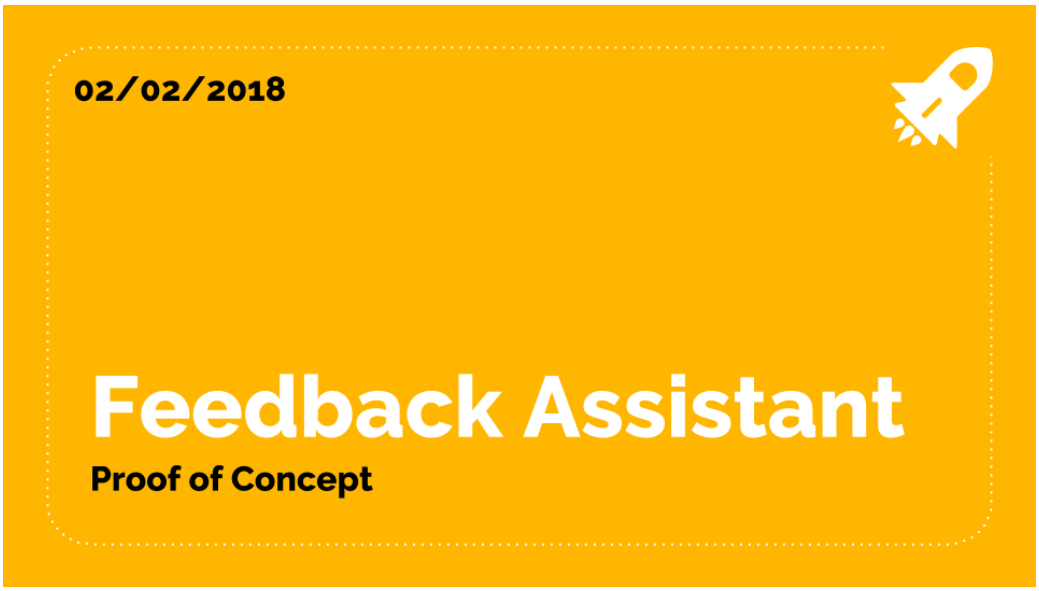


Ilustración 11.3.1 Diapositiva 1 prueba de concepto

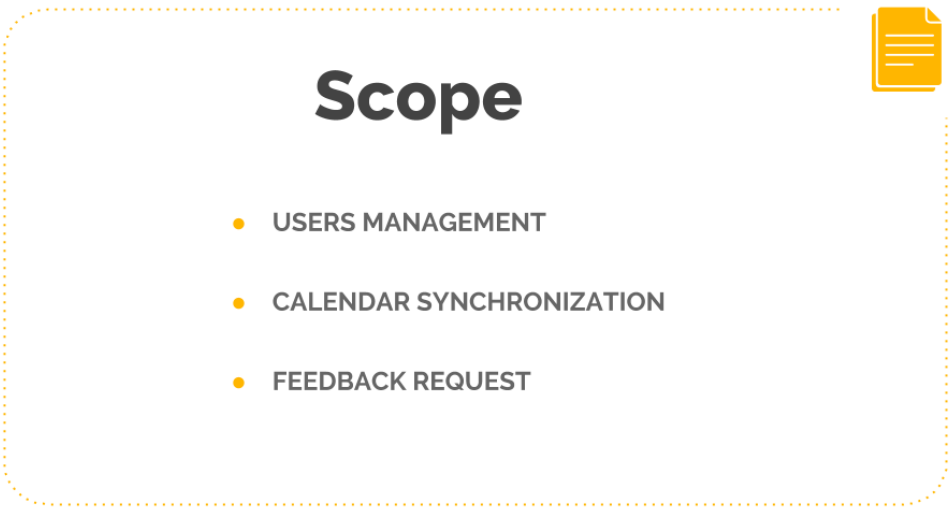


Ilustración 11.3.2 Diapositiva 2 prueba de concepto

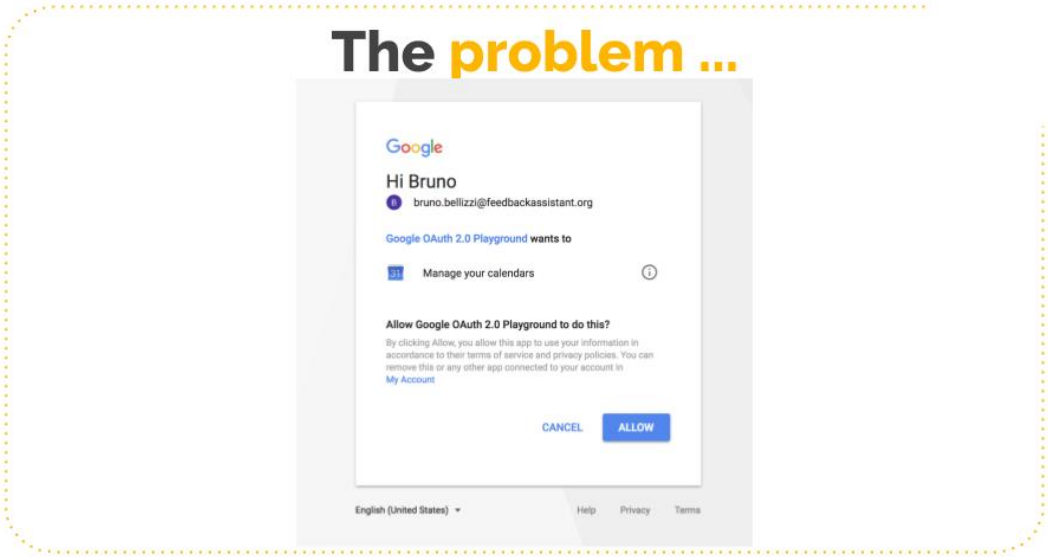


Ilustración 11.3.3 Diapositiva 3 prueba de concepto

En esta diapositiva se explica al cliente el gran problema tecnológico asociado a la restricción de no contar con interfaz de usuario y a su vez tener que solicitar permiso a los usuarios para acceder a sus datos laborales. Para conocer más acerca de este problema tecnológico ver sección 5.2 Principales desafíos tecnológicos.

Ilustración 11.3.4 Diapositiva 4 prueba de concepto

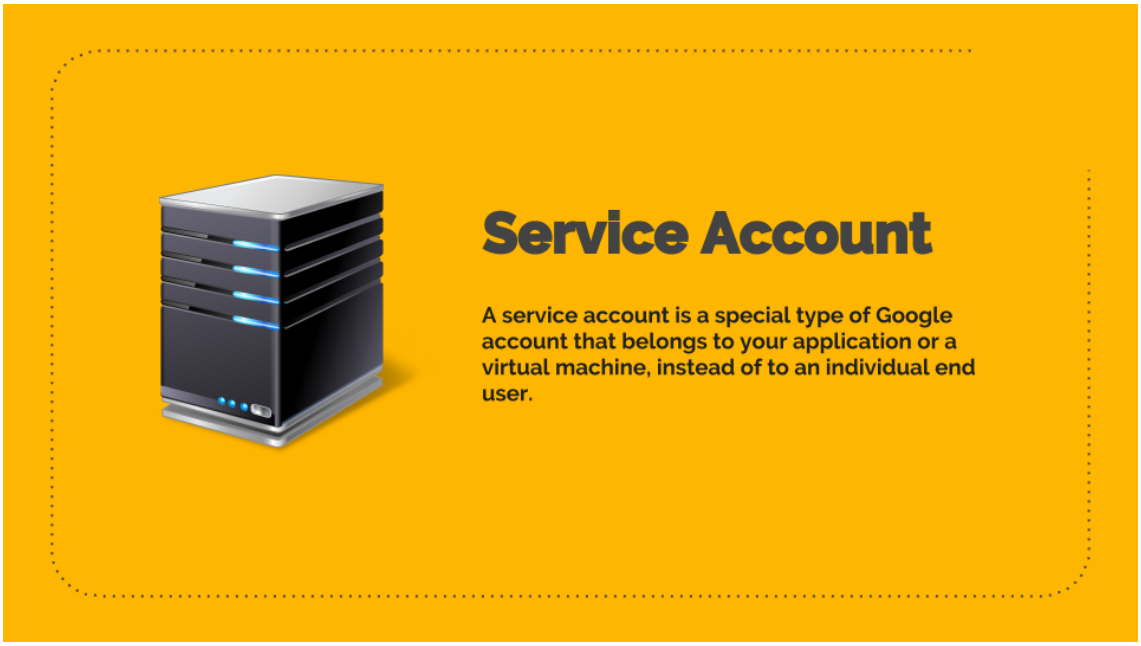


Ilustración 11.3.5 Diapositiva 5 prueba de concepto

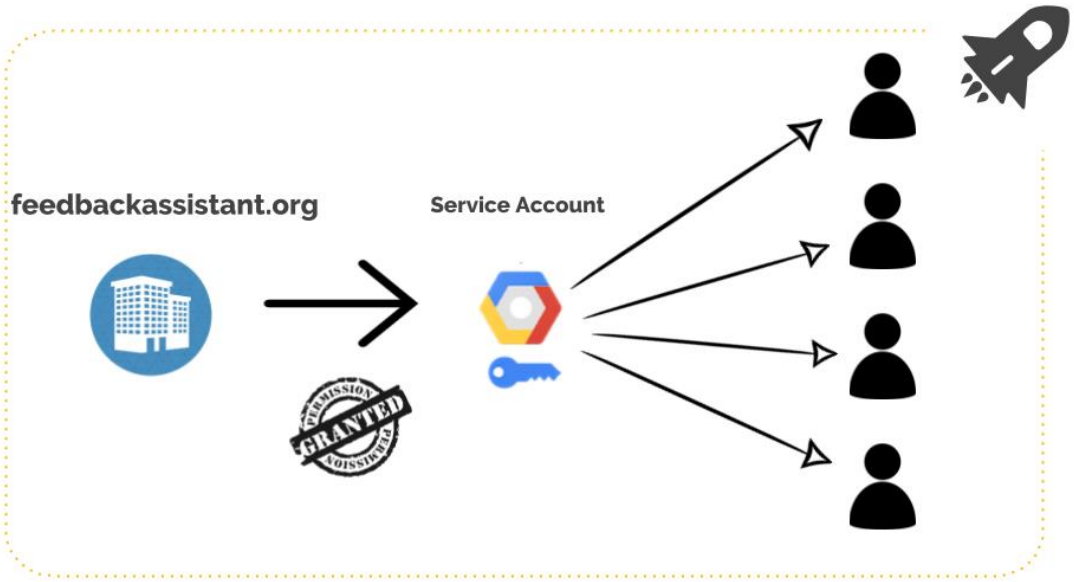


Ilustración 11.3.6 Diapositiva 6 prueba de concepto

En este momento se da a conocer el mecanismo por el cual el equipo resuelve el problema de acceso a los datos de los usuarios sin su previo consentimiento manual.

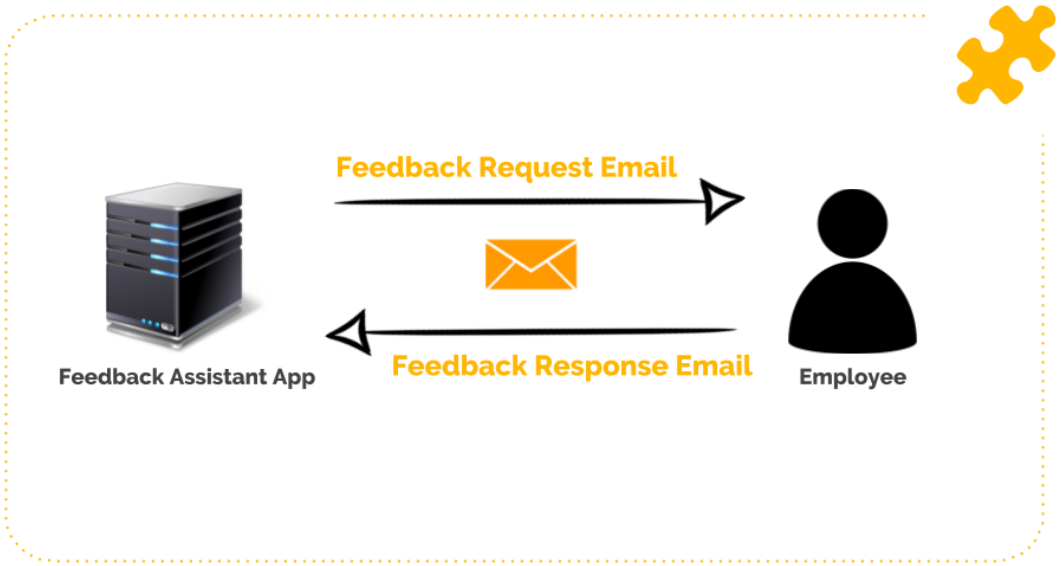


Ilustración 11.3.7 Diapositiva 7 prueba de concepto

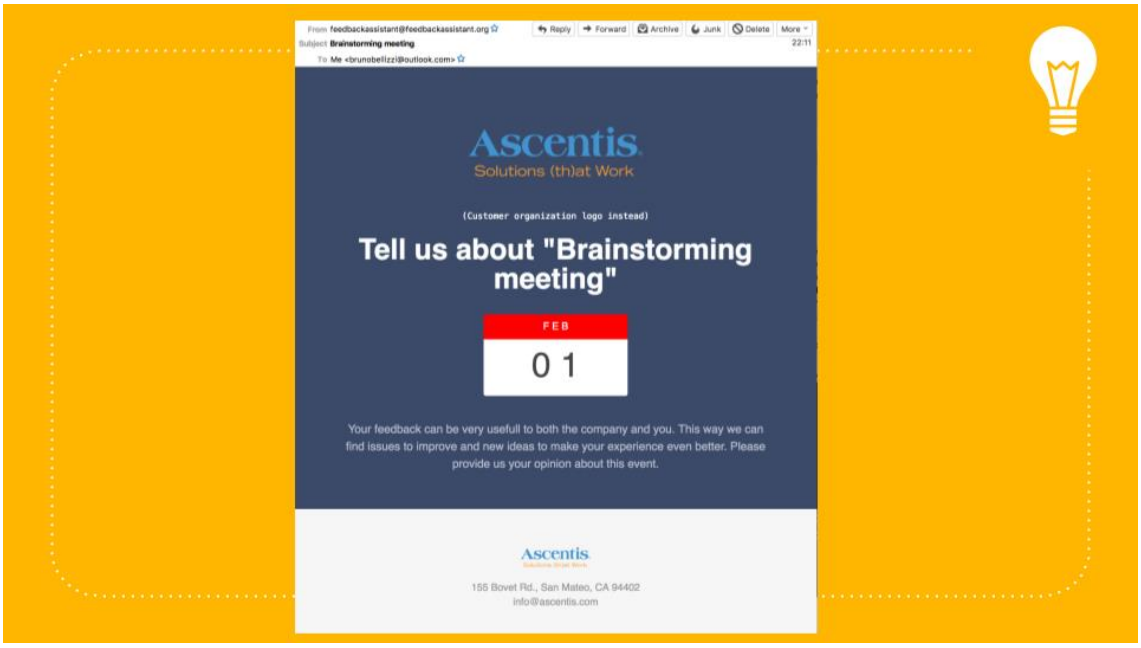


Ilustración 11.3.8 Diapositiva 8 prueba de concepto

Se presenta el correo electrónico diseñado por el equipo para la solicitud de *feedback*, se deja en claro que más allá del diseño generado, el mismo es totalmente customizable para que se adapte fácilmente a todos los clientes de Ascentis.

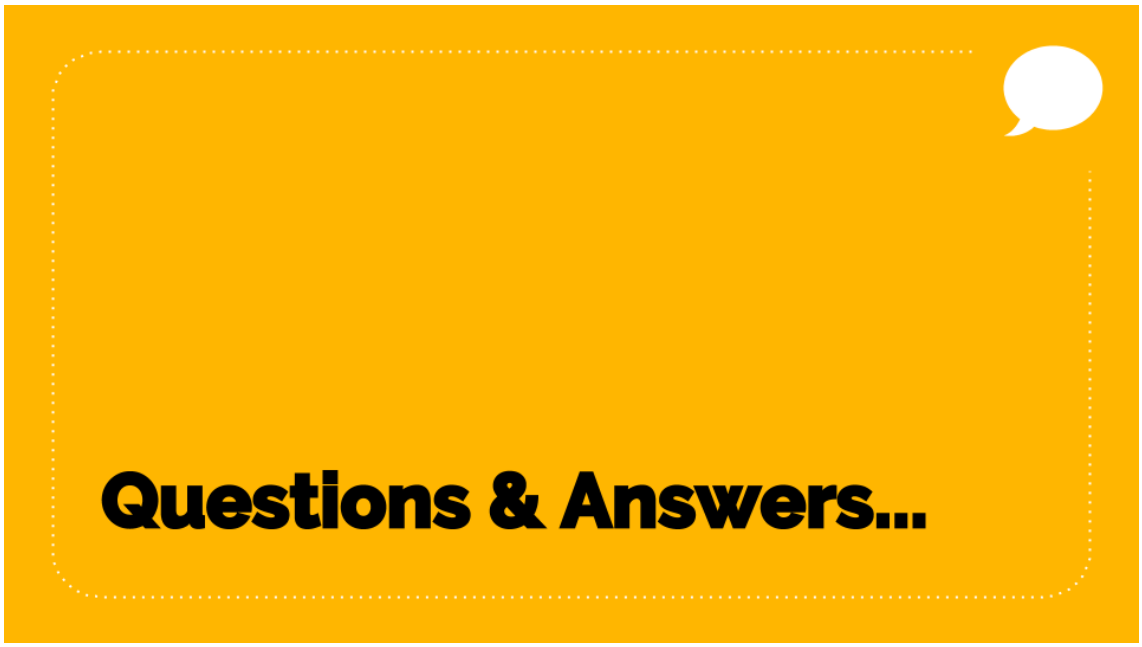


Ilustración 11.3.9 Diapositiva 9 prueba de concepto

Ilustración 11.3.10 Diapositiva 10 prueba de concepto

11.4 Presentación de nuevas funcionalidades

A continuación se muestra la presentación realizada el 19 de Abril del 2018 en una reunión de seguimiento con Ascentis. El objetivo fue mostrar la arquitectura actual al momento, mostrar avances y proponer nuevos requerimientos para desarrollar luego de entregado el MVP.

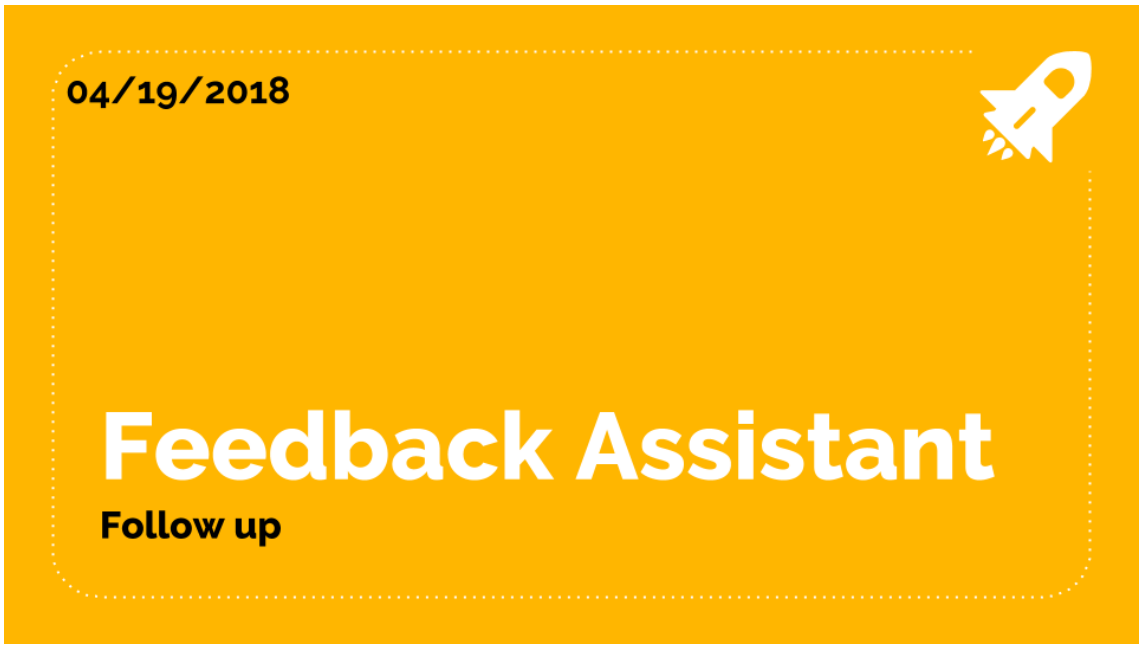


Ilustración 11.4.1 Diapositiva 1 de reunión de seguimiento 19/04/2018.

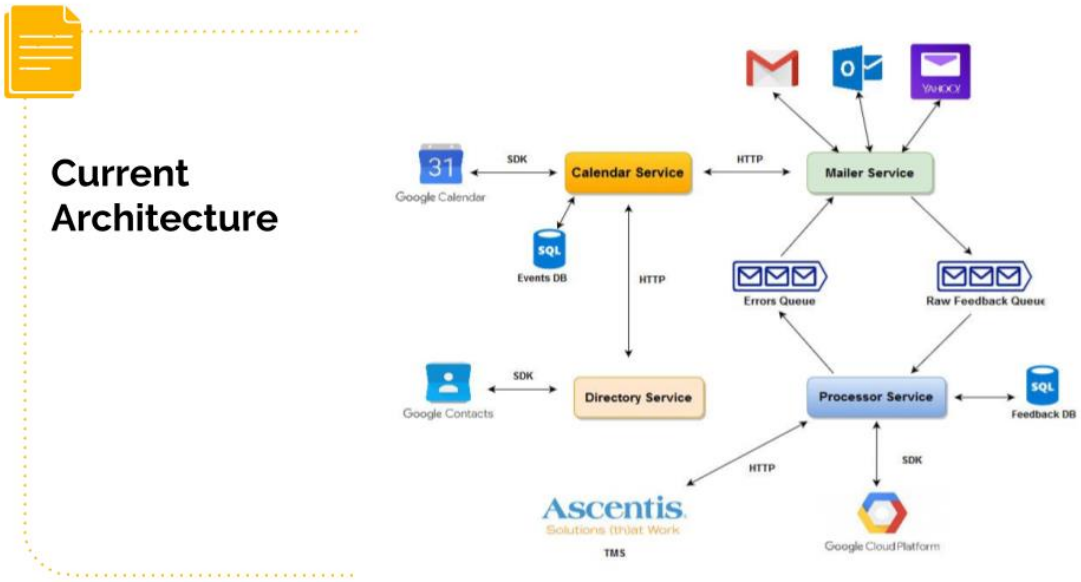


Ilustración 11.4.2 Diapositiva 2 de reunión de seguimiento 19/04/2018

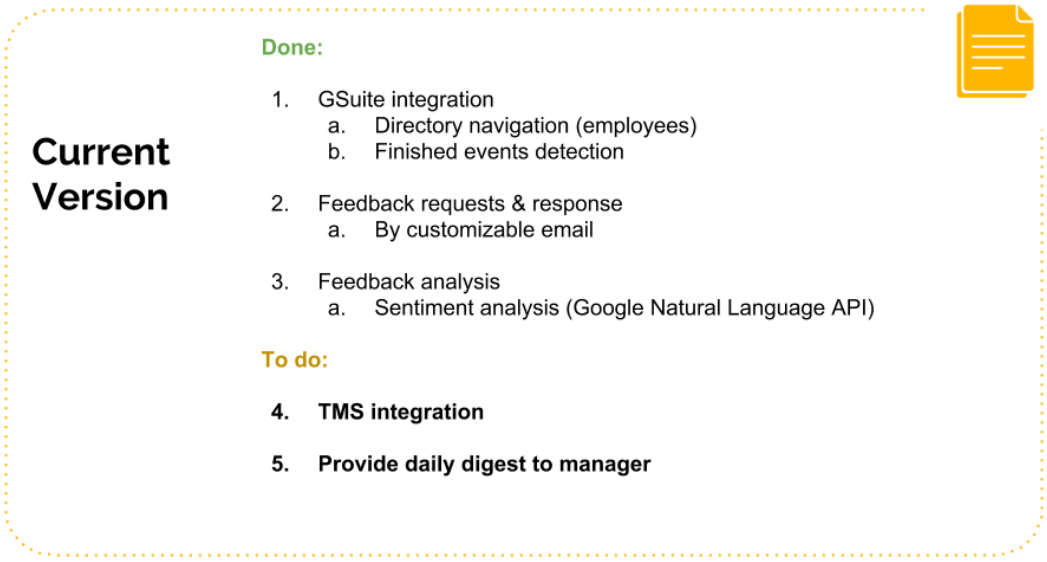


Ilustración 11.4.3 Diapositiva 3 de reunión de seguimiento 19/04/2018

11.5 Revisiones ORTsf

Primera revisión
Revisor: Pablo Hernández

Comentarios:

- Las horas de trabajo dedicadas son muy pocas.
- El problema no es presentado claramente.
- El alcance del producto es corto.
- No se muestra una planificación clara.
- Definir qué es un evento.
- No se muestran fechas de cuándo se comienza a codificar.
- Los roles no están definidos.
- No se menciona el ciclo de vida del producto.
- ¿Cómo se van a medir los objetivos que se plantean?
- ¿Cómo se validan los requerimientos?

Segunda revisión
Revisor: Marcelo Cagnani

Comentarios:

- Es más claro presentar la metodología antes de los requerimientos.
- En vez de jefes, usar término asesores de desempeño o talento.
- Explicar mejor que nuestro producto va a ser una parte más del producto TMS.
- La gráfica de distribución del tiempo queda confusa ya que hay muchas categorías, suprimir algunas.
- Sección SQA
- Agregar valores de testing
- Tener en cuenta la usabilidad
- Cúal fue el criterio utilizado para la selección de tecnologías.

Tercera revisión
Revisora: Amalia Álvarez

Comentarios:

- Agregar ppt de lecciones aprendidas.
- Definir qué es lo que le vamos a entregar al cliente (casos de prueba, manuales, etc).
- Falta restricción asociada a que todo tiene que ser en inglés.
- Explicar con más detalle cómo influye el product owner en el producto, no queda claro.
- La escala de colores de las gráficas de satisfacción al cliente son confusas.

11.6 Comparación de análisis de sentimiento de *feedbacks*

El propósito de este anexo es dar a conocer el alto grado de precisión en el análisis del sentimiento del *feedback* respecto a la integración con un único servicio, mediante el mecanismo de utilizar múltiples servicios de PLN y luego generando el promedio entre los resultados obtenidos para disminuir el posible margen de error en el análisis.

A continuación se presenta la categorización que realizó el equipo para etiquetar cada *feedback* según su sentimiento.

Etiqueta del *feedback*	**Rango de porcentaje de análisis de sentimiento (%)**
Muy positivo	81 - 100
Positivo	61 - 80
Neutral	41 - 60
Negativo	21 - 40
Muy Negativo	0 - 20

Tabla 11.6.1 Categorización del sentimiento del feedback

En la siguiente tabla se presenta el sentimiento obtenido por cada servicio junto con el promedio de los resultados. En la columna Rango esperado se encuentra el fondo coloreado que expresa si la columna Promedio de análisis coincide con el rango de sentimiento esperado por el equipo o no. Cuando aparece en verde implica que el promedio obtenido coincide con lo esperado siendo rojo cuando no coincide.

Feedback	Google Natural Language(%)	Microsoft Azure(%)	IBM Watson (%)	Promedio (%)	Rango esperado
“Great work! The meeting was very productive. We were able to understand the problem and think in possible solutions.”	84	86	90	**87**	**Muy positivo**
“You always provide great service to both fellow employees and the public. They always walk away with a smile.”[ref]	75	96	91	**87**	**Muy positivo**
“Bill, you exceeded your production goal by 20% last week. Great job, that’s really going to help us meet our overall plant production and financial goals. How did you do it?”[ref]	65	78	99	**81**	**Muy positivo**
“Your presentation today was so engaging. I especially liked how you spoke up clearly and made a point of making eye contact with each person in the room. I know you’ve said in the past that talking in front of groups makes you nervous, but today you came across as in command. It was a great example of your persistence and strength.”[ref]	70	90	71	77	**Positivo**
“Kathryn is working hard on improving her communication in dealing with Senior Executives”	75	79	50	**68**	**Positivo**
“I can see the team decided to find the right way to solve the problem. Keep going in this direction!”	55	86	73	**72**	**Positivo**
“Your review skills are good, but try to limit your	50	84	89	**75**	**Neutral**

conversation and let your employees have a chance to speak. It is supposed to be a tool to open up communication."					
"Your work was good as regular. Keep working and communicate things better to folks."	49	60	58	**55**	**Neutral**
"Nancy, at the meeting this morning I noticed you getting defensive when your data was challenged during your presentation. When Amy asked a question about your calculations, you were short with her and told her she needs to trust that you know how to do your job". When you responded to her that way, she shut down for the rest of the meeting and seemed angry. You really need her support, and I'm wondering if you'll have it now." [ref]	50	22	37	**37**	**Negativo**
"Not making eye contacto - need to work on this"	30	18	31	**26**	**Negativo**
"The meeting was too short. We were able to discuss only the first three point from the agenda. I suggest another meeting in the next days."	50	21	26	**33**	**Negativo**
"I think the work done by the company wasn't enough. We need to make a change! We can't continue like this."	45	50	12	**36**	**Muy negativo**
"I think the meeting was very long. We were all tired and it was impossible to solve the problem. I suggest another meeting next week."	32	6	15	**18**	**Muy negativo**

“I am not happy with today's meeting. People are not taking this instance seriously and this is causing a problem! There are clients waiting for our response! We have to change the attitude immediately.”	30	4	24	**19**	**Muy negativo**

Tabla 11.6.2 Comparación de análisis de sentimientos de feedbacks

11.7 Despliegue ideal

A continuación se presenta un diagrama mostrando la distribución ideal de los artefactos de software en distintos contenedores. Como fue mencionado en la sección 4.3.3 Vista de alocación no se realizó una instalación de este tipo por motivos de simplificar el despliegue en etapas de desarrollo, debido al poco tiempo con el que se contó. Una alternativa posible hubiese sido desplegar los microservicios como se muestra a continuación en distintos contenedores Docker, favoreciendo así la disponibilidad.

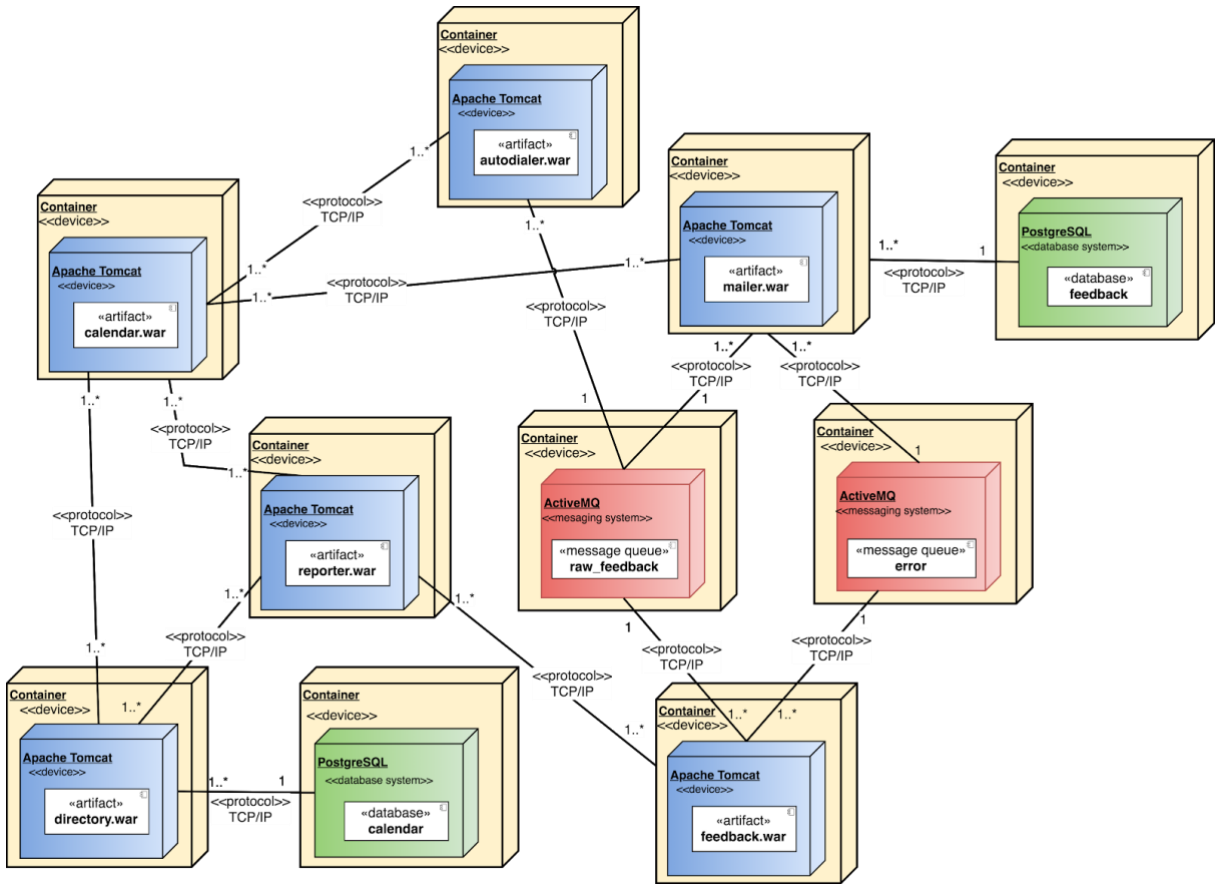


Ilustración 11.7.1 Diagrama de despliegue ideal

Se puede observar el ecosistema de microservicios donde cada pieza de software se aloja en infraestructura independiente. La multiplicidad en las relaciones dentro del diagrama refieren a una escalabilidad horizontal por parte de cada servicio en distintos servidores. Es conveniente replicar los nodos en distintos contenedores para favorecer una mayor disponibilidad y escalabilidad.

11.8 Plan de Calidad

A continuación se presenta el plan de calidad elaborado por el equipo durante las primeras fases del proyecto. Para la generación de este documento se tomó como referencia el plan de calidad expuesto por Universidad ORT Uruguay en el sitio Aulas.

Nombre fase	Actividad	Descripción	Producto Resultado	Producto/s Consumido/s	Rol Responsable	Roles Participantes
Investigación	Estudio de las herramientas tecnológicas	- Analizar ventajas y desventajas de usar *Java EE* o *Spring Framework*. - Investigar las APIs y servicios a integrarse.	- Pruebas de concepto - Documento de investigación tecnológica.	- Documentos tecnológicos. - Información y tutoriales sobre determinadas tecnologías.	Arquitecto de Software	Equipo de proyecto.
	Estudio del negocio y sus restricciones.	- Incursionar en conocimientos asociados al sector de recursos humanos. - Investigar los productos de software ya existentes ofrecidos por la empresa.	Documento de investigación del negocio.	- Documentos provistos por Ascentis. - Información acerca de la gestión de recursos humanos. - Reuniones con Ascentis.	Ingeniero en requerimientos.	Equipo de proyecto.
Ingeniería de requerimientos	Identificación de requerimientos.	Lograr captar las funcionalidades con las que debe contar el producto.	Documento con listado de requerimientos	- Videollamadas con Ascentis. - Documento de especificación del sistema entregado por Ascentis al inicio del proyecto.	Ingeniero de requerimientos.	Cliente, Equipo de proyecto.
	Instancias de brainstorming.	Realizar instancias de brainstorming entre los integrantes del equipo donde puedan surgir discusiones constructivas y nuevas ideas.	Listado de posibles requerimientos.	Reuniones de equipo.	Ingeniero de requerimientos.	Equipo de proyecto.
	Especificación de requerimientos	Generar un documento en donde volcar formalmente los requerimientos y restricciones relevadas.	Especificación de requerimientos funcionales (*Product Backlog*) y no funcionales.	Listado de requerimientos validados por Ascentis.	Ingeniero de Requerimientos	Equipo de proyecto.

	Validación de requerimientos	Obtener la aprobación por parte de Ascentis en cuanto al ESRE y *Product Backlog.*	- ESRE validado. - *Product Backlog* validado.	- ESRE - Product Backlog	Ingeniero de Requerimientos	Cliente, Equipo de proyecto.
	Priorización de requerimientos	Ordenar los requerimientos según el valor que aporta a Ascentis.	- Product Backlog priorizado. - ESRE con requerimientos ordenados por prioridad.	- ESRE validado. - *Product Backlog* validado.	Ingeniero de Requerimientos	Cliente, Equipo de proyecto.
Diseño y Arquitectura	Análisis de requerimientos no funcionales.	Analizar hacia qué aspectos de la calidad se inclina cada requerimiento no funcional contenido en el ESRE.	Atributos de calidad a favorecer y sus tácticas asociadas.	Requerimientos no funcionales (alojados en ESRE) .	Arquitecto de Software	Equipo de proyecto y Cliente.
	Definición de patrones de diseño y arquitectónicos a utilizar.	Buscar alternativas de soluciones a problemas identificados utilizando patrones arquitectónicos y de diseño.	Documento de arquitectura.	- Documentos de patrones de diseño y arquitectónicos. - Atributos de calidad a favorecer.	Arquitecto de software.	Equipo de proyecto.
	Presentar posibles soluciones arquitectónicas.	Considerando los requerimientos, atributos de calidad a favorecer y sus tácticas asociadas generar posibles arquitecturas de alto nivel.	Diagramas de alto nivel de las distintas soluciones.	- ESRE. - Atributos de calidad y tácticas. - Documentos de patrones tanto arquitectónicos como de diseño.	Arquitecto de software.	Equipo de proyecto.
	Elección de solución arquitectónica más favorable para la realidad del proyecto.	Seleccionar la arquitectura más adecuada para la realidad del proyecto y evidenciar su justificación con sólidos argumentos.	Justificación detallada de la elección de solución arquitectónica.	- Diagramas de alto nivel de las posibles arquitecturas. - ESRE. - Atributos de calidad a favorecer.	Arquitecto de Software.	Equipo de proyecto y Cliente.
	Definición de la solución arquitectónica.	Generar el documento formal asociado a la arquitectura del sistema.	Documento de arquitectura	- Libro "Documenting Software Architectures: Views and Beyond" - ESRE. - Atributos de calidad y tácticas. - Justificación de la elección de la solución a ejecutar.	Arquitecto de Software.	Líder de arquitectura

	Revisión de la arquitectura	Una vez generado el documento de arquitectura los integrantes del equipo llevan a cabo una revisión del mismo.	Documento de arquitectura revisado	Documento de arquitectura	Líder de SQA	Equipo de proyecto.
	Validación de la arquitectura	Una vez revisado el documento de arquitectura Ascentis aprueba el mismo o propone los cambios necesarios para su aprobación.	Documento de arquitectura validado.	Documento de arquitectura revisado.	Arquitecto de software.	Cliente, Equipo de proyecto.
Construcción del software	Planificación del desarrollo.	Corresponse a la ceremonia *Sprint Planning* la cuál se realiza cada dos semanas en conjunto con el *Product Owner.*	*Sprint Backlog*	- *Product Backlog* - *Sprint planning*	Equipo de proyecto.	Cliente, Equipo de proyecto.
	Codificación de una iteración	Desarrollo de *software* llevado a cabo en el *sprint.*	- Código fuente. - Artefactos generados.	- ESRE. - *Sprint Backlog.* - Documento de Arquitectura	Equipo de proyecto.	Equipo de proyecto.
	Revisión del código	A medida que se desarrolla se crean *Pull Requests* los cuales luego son revisados por otro integrante del equipo.	Código fuente revisado.	- Estándares de codificación. - Documento de arquitectura.	Equipo de proyecto.	Equipo de proyecto.
	Pruebas unitarias	Implementación de pruebas unitarias a medida que se desarrolla.	Código fuente de pruebas unitarias.	- *Sprint Backlog* - ESRE	Responsable de SQA	Equipo de proyecto.
Testing	Diseño de casos de prueba	Diseñar casos de prueba para las funcionalidades mas importantes del sistema.	Documento de casos de prueba	Product Backlog.	Responsable de SQA	Equipo de proyecto.
	Validación de documento de casos de prueba	Validar con Ascentis los casos de prueba para los flujos principales del sistema.	Documento de casos de prueba validado.	Documento de casos de prueba.	Responsable de SQA.	Equipo de proyecto, Cliente.
	Ejecución de pruebas y registración de resultados.	Ejecutar las pruebas diseñadas y registrar resultados.	Documento de pruebas ejecutadas con su resultado esperado y obtenido.	- Documento de casos de prueba. - Artefactos de *software.*	Responsable de SQA.	Equipo de proyecto.
Procesos de Gestión (Paralelos a las fases)	Planificación de actividades	Organización de las actividades teniendo en cuenta los	Cronograma de actividades	- Hitos del proyecto. - Plan de calidad.	Gerente de proyecto.	Gerente de proyecto, Responsable de SQA.

		eventos de mayor importancia durante el proyecto.				
	Gestión de riesgos	Trazabilidad en el tiempo acerca de los principales riesgos existentes	Plan de riesgos	- ESRE. - Documento de investigación tecnológica. - Documento de investigación del negocio.	Gerente de proyecto.	Equipo de proyecto.
	Diseño del Plan de calidad	Identificar las fases del proyecto y sus actividades asociadas.	Plan de la calidad	- Lista de entregables. - Documentos de calidad existentes	Responsable de SQA	Equipo de proyecto.
	Planificación de las versiones.	Realizar el plan de versiones donde se detallan las fechas donde se entregarán versiones al cliente y qué será entregado. La primer versión es el MVP negociado con el cliente.	Plan de versiones.	Necesidades del cliente, alcance, producto mínimo viable y recursos humanos disponibles.	Gerente de proyecto.	Equipo de proyecto.
	Seguimiento de satisfacción del cliente	Realizar encuestas luego de cada demo con el cliente solicitando feedback respecto a distintos aspectos.	Métricas de satisfacción del cliente	- Demo. - Encuestas. - *Feedback*.	Gerente de proyecto.	Equipo de proyecto, Cliente.

Tabla 11.8.1 Plan de calidad

11.9 Planilla de casos de pruebas funcionales

Id Req	Requerimiento Funcional	ID Prueba	Nombre	Descripción	Condiciones	Resultado esperado	Resultado obtenido	ID defecto
RF01	Sincronizar usuarios de Google GSuite	P1	Sincronización de los usuarios de una organización	El sistema debe ser capaz de obtener todos los usuarios dentro de una organización de GSuite		Se obtienen los usuarios de una organización	**OK**	
		P2	Falta o errores en la configuración	Se realiza una sincronización con una configuración incorrecta (por ejemplo, falta el certificado o el nombre de la organización se escribió incorrectamente)		Se deja registro en el log	**OK**	
		P3	Indisponibilidad del servicio de GSuite	Si el servicio de GSuite se encuentra indisponible (tanto por problemas de comunicación como por interrupción del servicio de Google) la sincronización se cancela y se reintenta en el futuro.	Pérdida de acceso a internet o API de Google Calendar no disponible	Se cancela la sincronización actual y se deja registro en el log	**OK**	
RF02	Sincronizar eventos con Google Calendar	P4	Sincronización de los eventos de un usuario	El sistema realiza una sincronización con los calendarios de los usuarios en Google Calendar, guardando los nuevos eventos y actualizando los existentes.	Condiciones normales	Se sincronizan los eventos de los usuarios, guardando los nuevos eventos y modificando los existentes.	**OK**	FA-50
		P5	Sincronización período	La sincronización de eventos se realiza de forma periódica en un intervalo de tiempo configurable	Condiciones normales	La sincronización de eventos ocurre de forma automática, en el intervalo de tiempo configurado	**OK**	

		P6	Falta o errores en la configuración	Se realiza una sincronización con una configuración incorrecta (por ejemplo, falta el certificado o el nombre de la organización se escribió incorrectamente)		Se deja registro en el log	**OK**	
		P7	Mutua exclusión de las sincronizaciones de eventos	Si hay una sincronización de eventos en curso y se dispara una nueva, se deberá cancelar la nueva sincronización para evitar la perdida de integridad en los datos (como eventos duplicados). Además se evita sobrecargar al sistema con actualizaciones idénticas en paralelo	Una sincronización de eventos demora más que el intervalo de tiempo configurado	Se cancela la nueva sincronización, mientras que la primera continúa sin interrupciones	**OK**	
		P8	Indisponibilidad del servicio de Google Calendar	Si la API de Google Calendar se encuentra indisponible (tanto por problemas de comunicación como por interrupción del servicio de Google) la sincronización se cancela.	Pérdida de acceso a internet o API de Google Calendar no disponible	Se cancela la sincronización de la tarea actual y se deja registro en el log	**OK**	
		P9	Evento compartido por varios usuarios	Los eventos se persisten como entidades independientes. Si durante una sincronización aparece un evento compartido por dos o más usuarios, el evento se persiste una única vez sin duplicados. El evento tiene la referencia de todos sus participantes	Eventos compartidos por dos o más usuarios	El evento se persiste una sola vez, sin duplicarse y con las referencias de todos sus participantes	**OK**	FA-52

RF03	Solicitar feedback mediante correo elctrónico al terminar un evento	P10	Solicitud de feedback exitósa	Al identificar la finalización de un evento el sistema envía un correo electrónico a todos los asistentes del evento que pertenezcan a la organización solicitando feedback.	Condiciones normales	Se envía un correo electrónico a todos los participantes del evento que pertenezcan a la organización	**OK**	FA-51 FA-114
		P11	Filtro de participantes no pertenecientes a la organización	Un evento podría tener participantes no pertenecientes a la organización, como clientes o proveedores. A estos participantes no se les debe solicitar feedback. Se considera que un participante partenece a la organización si posee un correo electrónico de la organización (ej: @ascentis.com)	Finalización de un evento con por lo menos un participante no empleado de la organización	Se solicita feedbacks únicamente a los participantes empleados de la organización.	**OK**	
		P12	Indisponibilidad del servicio de envío de emails	En caso de no poder enviar un correo electrónico solicitando feedback (indisponibilidad del microservicio encargado de enviar mails, perdida de conexión a internet, etc.) el evento debe quedar en estado pendiente hasta que se solucione el problema,y se reintentará solicitar feedback automaticamente	Pérdida de acceso a internet o servicio de envío de emails indisponible	Se suspende la solicitud de feedback mediante correo electrónico hasta que se reestablezca el servicio de envío de mails	**OK**	FA-70
		P13	Información relacionada al evento en la solicitud	La solicitud de feedback deberá indicar de qué evento se trata,	Condiciones normales	La solicitud de feedback contiene los datos	**OK**	FA-84

				conteniendo en el mail el nombre del evento y la fecha de realización de forma que el usuario pueda reconocer rápidamente sobre qué evento se le está solicitando feedback		correctos del evento solicitado		
		P14	Falta archivo del template del mail	En caso de que no se encuentre el archivo HTML del template del mail se debe cancelar el envío y registrar el error		Se cancela el envío y se guarda registro en el log	**OK**	
RF04	Recibir feedback por correo electrónico	P15	Recepción de feedback por correo electrónico	El sistema recibe los feedbacks enviados por los usuarios vía correo electrónico. Debe ser capáz de parsear el mail, descartando el contenido no relevante (por ej. el texto correspondiente a la solicitud) y guardando únicamente el contenido real del feedback.	Condiciones normales	Se construye el feedback recibido, identificando al usuario que lo generó y el evento asociado.	**OK**	FA-95 FA-117
		P16	Inserción de feedbacks recibidos en cola de mensajes	Una vez recibido y parseado un feedback, el mismo se inserta en una cola de mensajes para que otro componente del sistema lleve a cabo su procesamiento.	Condiciones normales	Luego de parsear el feedback recibido, el mismo se encola en la cola de mensajes raw_feedback.	**OK**	
		P17	Recepción de feedback duplicado	Es posible que un usuario responda dos o más veces a una misma solicitud de feedback. El sistema debe soportar esto y registrar feedbacks independientes por cada respuesta	Se reciben dos o más correos electrónicos respondiendo la misma solicitud de feedback (mismo evento)	Se registra cada feedback de forma independiente, pudiendo coexistir varios feedbacks del mismo usuario para el mismo evento	**OK**	FA-85

		P18	Revisión periódica de nuevos feedbacks	El sistema revisa periódicamente la casilla de correo electrónico en busca de nuevos feedbacks recibidos cada un intervalo de tiempo configurable	Condiciones normales	Se revisan y descargan los nuevos feedbacks recibidos durante un intervalo de tiempo predefinido	**OK**	
		P19	Indisponibilidad del servidor de correo electrónico	En caso de no poder acceder al correo electrónico para descargar los feedbacks recibidos (servidor de correo inaccesible), el sistema deberá suspender la recepción de feedbacks hasta que se reestablezca el servicio (reintentando en en el futuro, dejando un log con información del suceso	Pérdida de acceso a internet o servidor de correo electrónico indisponible	Se suspende la recepción de feedbacks y se deja registro en los logs.	**OK**	
		P20	Indisponibilidad de la cola de mensajes	En caso de no poder insertar el feedback recibido en la cola de mensajes (por problemas en la red o en la misma cola de mensajes) el feedback quedará en estado pendiente (se mantendrá el correo electrónico como no leído) hasta que se reestablezca el acceso a la cola de mensajes	Pérdida de acceso a la cola de mensajes, por problemas de comunicación o de la propia cola de mensajes	Se ignora el feedback, que será reprocesado en la próxima iteración de la tarea periódica.	**OK**	
RF05	Delegación de autoridad a cuenta de servicio de Gsuite.	P21	Acceso a los datos de un usuario sin su previo consentimiento	El sistema debe ser capaz de acceder a los datos del GSuite de un usuario, como sus eventos o calendario, sin requerir su	Condiciones normales	Se logra acceder a los datos de los usuarios registrados en GSuite sin consentimiento manual.	**OK**	

				explícito consentimiento. Para esto un administrador de GSuite deberá habilitar permisos a Feedback Assistant para que pueda acceder a toda la información. Esto libera al usuario final de acciones innecesarias				
		P22	Intento de acceder a los datos de un usuario sin haber otorgado permisos previamente.	Si el administrador de GSuite no habilita a Feedback Assistant para acceder a la información de sus usuarios, las solicitudes al GSuite deberán suspenderse hasta la próxima iteración de la tarea periódica dejando registro en el log.	No se habilita a Feedback Assistant a consultar información sobre los usuarios de la organización de GSuite	Se suspenden todas las solicitudes a las APIs de Google hasta la próxima iteración, dejando registro en el log del suceso	**OK**	FA-116
RF06	Analizar el sentimiento de un feedback con Google Natural Language API	P23	Análisis de sentimiento de un feedback válido	Se solicita al servicio de análisis de sentimiento un análisis sobre el contenido del feedback	Condiciones normales	Se logra analizar el sentimiento de un feedback recibido, persistiendo sentimiento del mensaje.	**OK**	FA-87
		P24	Análisis de sentimiento de un feedback vacío	Si un feedback no tiene contenido se deberá descartar al mismo.	Análsis de un feedback con poca o nula información	Se descarta y se registra en el log.	**OK**	
		P25	Indisponibilidad del servicio de análisis de sentimiento	En caso de que el servicio de análisis de sentimiento no se encuentre disponible (por errores de comunicación, credenciales o problemas del própio servicio), se deberá notificar el evento al usuario y asignarle un sentimiento 0 (neutral)	Indisponiblidad del servicio de análisis de sentimiento, por problemas de comunicación o del servicio en si	Se envía un correo electrónico al usuario indicando que su feedback no pudo ser procesado y se deja registro en log.	**OK**	

RF07	Notificar al TMS luego que el feedback es procesado	P26	Notificación al servicio de recepción de feedbacks de TMS	Luego de procesar un feedback, el sistema envía al sistema del cliente (TMS) el feedback recibido	Condiciones normales	Se invoca al mock de la API de TMS.	**OK**	
		P27	Indisponiblidad del servicio de recepción de feedbacks de TMS	En el caso de no poder notificar al sistema del cliente el nuevo feedback procesado, tanto por problemas de comunicación como por problemas del propio servicio, se deberá suspender la notificación hasta que se reestablezca el servicio	Indisponiblidad del servicio de recepción de feedbacks del cliente, por problemas de comunicación o del servicio en si	Se suspende la comunicación al mock.	**OK**	
RF08	Generar reportes diarios	P28	Envío de reportes diarios a los asesores de talento	Se debe enviar un correo electrónico a los asesores de talento conteniendo un reporte con los feedbacks procesados el día anterior		Se envía el reporte diario correspondiente con la información del día.	**OK**	FA-115 FA-118 FA-119
		P29	Envío periódico	El envío del reporte debe ocurrir cada un intervalo de tiempo configurable (en milisegundos)		Se envía el reporte en cada intervalo de tiempo configurado	**OK**	
		P30	Sin destinatarios configurados	Si no hay destinatarios (correos electrónicos de asesores de talento) configurados se debe cancelar el envío y dejar un registro en el log. No se debe generar el reporte		Se registra el suceso y se cancela el envío	**OK**	FA-120
		P31	Múltiples destinatarios configurados	Si existen varios destinatarios configurados (concatenados por coma) se debe enviar		Se envía un único correo con copia a todos los destinatarios	**OK**	

				ún solo correo con copia a todos los destinatarios				
		P32	No hay feedbacks recibidos en el día	Si no hay ningun feedback recibido en el día se debe cancelar el envío de este reporte		Se cancela el envío del reporte hasta la siguiente iteración y se deja registro en el log	**OK**	
		P33	Servicio indisponible	Alguno de los microservicios necesarios no se encuentra disponible (por problemas de comunicación o problemas del propio servicio). Se debe cancelar el envío del reporte y registrar el suceso en el log		Se cancela el envío del reporte hasta la siguiente iteración y se deja registro en el log	**OK**	
RF09	Generar reportes semanales	P34	Envío de reportes semanales a los asesores de talento	Se debe enviar un correo electrónico a los asesores de talento conteniendo un resumen de los feedbacks recibidos la semana anterior con la tendencia del sentimiento global y por usuario + cantidad de feedbacks recibidos por cada tipo (Negativo, Postiivo, Neutral, etc.)		Se envía el reporte semanal correspondiente con la información de la semana anterior	**OK**	FA-144
		P35	Envío periódico	IDEM Reporte Diario		IDEM Reporte Diario	**OK**	
		P36	Sin destinatarios configurados	IDEM Reporte Diario		IDEM Reporte Diario	**OK**	
		P37	Múltiples destinatarios configurados	IDEM Reporte Diario		IDEM Reporte Diario	**OK**	
		P38	No hay feedbacks recibidos en la semana anterior	IDEM Reporte Diario		IDEM Reporte Diario	**OK**	
		P39	Servicio indisponible	IDEM Reporte Diario		IDEM Reporte Diario	**OK**	

RF10	Generar reportes mensuales	P40	Envío de reportes mensuales a los asesores de talento	Se debe enviar un correo electrónico a los asesores de talento conteniendo un resumen de los feedbacks recibidos el mes anterior con la tendencia del sentimiento global y por usuario + cantidad de feedbacks recibidos por cada tipo (Negativo, Positivo , etc.)		Se envía el reporte mensuale correspondiente con la información del mes anterior	**OK**	
		P41	Envío periódico	IDEM Reporte Diario		IDEM Reporte Diario	**OK**	
		P42	Sin destinatarios configurados	IDEM Reporte Diario		IDEM Reporte Diario	**OK**	
		P43	Múltiples destinatarios configurados	IDEM Reporte Diario		IDEM Reporte Diario	**OK**	
		P44	No hay feedbacks recibidos en el mes anterior	IDEM Reporte Diario		IDEM Reporte Diario	**OK**	
		P45	Servicio indisponible	IDEM Reporte Diario		IDEM Reporte Diario	**OK**	
RF11	Feedback instantáneo	P46	Recepción de feedback instantáneo	El usuario provee feedback seleccionando una de las opciones disponibles en el email de solicitud. Luego es redirigido a una página que le agradece por proveer feedback y le informa de su correcta recepción		Se recibe el feedback, se le asigna la calificación correspondiente y se le agradece por su tiempo	**OK**	FA-150 FA-151
		P47	Falta archivo del template HTML de la respuesta	En caso de que no se encuentre el archivo HTML del template de la respuesta se debe mostrar la página de agradecimiento predefinida		Se muestra la página de agradecimiento predefinida y se guarda el feedback	**OK**	
		P48	Múltiples feedbacks instantáneos	Si un usuario provee dos o más feedbacks		Se reciben y procesan todos los feedbacks	**OK**	

				instantáneos sobre un mismo evento, se deben considerar como feedbacks independientes y aceptar cada uno		instantáneos (no se descartan)		
RF12	Solicitar feedback mediante llamada telefónica al finalizar un evento	P49	Solicitud de feedback por llamada telefónica	El usuario recibe una llamada telefónica con un mensaje configurable y un "beep" al final indicando que se comienza a grabar. El usuario da su feedback oralmente y espera 4 segundos, luego el bot le agradece con otro mensaje configurable y termina la llamada		Se recibe la llamada, se da feedback, se espera 4 segundos y se recibe la respuesta de agradecimiento por parte del bot.	**OK**	FA-169 FA-170
		P50	Servicio Twilio no disponible	Si el servicio de Twilio no se encuentra disponible (por problemas de comunicación o del propio servicio) se debe cancelar la llamada y dejar un registro en el log		No se realiza la llamada y se guarda un registro informando el suceso	**OK**	
		P51	Número de teléfono no existe	Si el número de teléfono registrado del usuario no es un número válido, se debe ignorar la llamada y dejar un registro en el log		Se cancela la llamada y se deja registro en el log	No se valída	FA-171
RF13	Recibir feedback mediante llamada telefónica	P52	Recepción de feedback por llamada telefónica	El usuario recibe una llamada telefónica con un mensaje configurable y un "beep" al final indicando que se comienza a grabar. El usuario da su feedback oralmente y espera 4		Se recibe la llamada, se da feedback y se recibe la respuesta. El feedback es recibido y encolado para posterior procesamiento	**OK**	FA-186

				segundos, luego el bot le agradece con otro mensaje configurable y termina la llamada				
		P53	Llamada rechazada	El usuario recibe la llamada telefónica pero la rechaza en lugar de contestar		Se identifica que la llamada fue cancelada y no se guarda el feedback	**OK**	
		P54	Llamada terminada antes de la grabación	Se corta la llamada (por problemas de comunicaciones o el usuario cuelga antes de tiempo) antes de que se pida proveer feedback (esto es, antes del "beep")		Se identifica que la llamada fue interrumpida y no se guarda el feedback	**OK**	
		P55	Llamada terminada durante la grabación	Se corta la llamada (por problemas de comunicaciones o el usuario cuelga antes de tiempo) mientras se está dando feedback (esto es, luego del "beep" y antes del mensaje de agradecimiento)		Se guarda el feedback grabado hasta el momento de la interrupción como un feedback válido	**OK**	
RF14	Análisis de sentimiento con IBM Watson - Natural Language Understanding	P56	Análisis de sentimiento de un feedback válido, promediando con el resto de los proveedores	Se solicita al servicio de análisis de sentimiento un análisis sobre el contenido del feedback. El resultado es luego promediado con el resto de los proveedores para determinar la clasificación final del feedback	Condiciones normales	Se logra analizar el sentimiento de un feedback recibido, se promedia con el resto de los servicios y se persiste el resultado	**OK**	
		P57	Análisis de sentimiento de un feedback vacío	Si un feedback no tiene contenido se deberá descartar al mismo.	Análsis de un feedback con poca o nula información	Se descarta y se registra en el log.	**OK**	
		P58	Indisponibilidad del servicio de análisis de sentimiento	En caso de que el servicio de análisis de sentimiento no	Indisponiblidad del servicio de análisis de sentimiento,	Se envía un correo electrónico al usuario indicando	**OK**	

				se encuentre disponible (por errores de comunicación, credenciales o problemas del própio servicio), se deberá notificar el evento al usuario y asignarle un sentimiento 0 (neutral)	por problemas de comunicación o del servicio en si	que su feedback no pudo ser procesado y se deja registro en log		
RF15	Análsis de sentimiento con Microsoft Azure - Cognitive Services	P59	Análisis de sentimiento de un feedback válido	Se solicita al servicio de análisis de sentimiento un análisis sobre el contenido del feedback	Condiciones normales	Se logra analizar el sentimiento de un feedback recibido, se promedia con el resto de los servicios y se persiste el resultado	**OK**	
		P60	Análisis de sentimiento de un feedback vacío	Si un feedback no tiene contenido se deberá descartar al mismo.	Análsis de un feedback con poca o nula información	Se descarta y se registra en el log.	**OK**	
		P61	Indisponibilidad del servicio de análisis de sentimiento	En caso de que el servicio de análisis de sentimiento no se encuentre disponible (por errores de comunicación, credenciales o problemas del própio servicio), se deberá notificar el evento al usuario y asignarle un sentimiento 0 (neutral)	Indisponiblidad del servicio de análisis de sentimiento, por problemas de comunicación o del servicio en si	Se envía un correo electrónico al usuario indicando que su feedback no pudo ser procesado y se deja registro en log	**OK**	

Tabla 11.9.1 Casos de prueba funcionales

11.10 Métricas de calidad del código

Todas las métricas presentadas en el presente anexo fueron extraídas utilizando la herramienta SonarQube. La misma realiza un análisis estático del código fuente identificando bugs, vulnerabilidades, cobertura de pruebas unitarias, código duplicado y *code smells* en más de 20 lenguajes de programación, incluyendo Java entre ellos. Para esto examina el código fuente del proyecto y verifica que se cumplan más de 491 reglas de calidad (por ejemplo, cantidad de parámetros de un método, definición inutilizada de variable, etc.) de forma de poder comprobar que se cumplan con los mejores estándares y buenas prácticas de Java. La lista completa de reglas puede encontrarse en el sitio de SonarSource [71].

Este análisis tuvo dos objetivos:

- **Determinar el estado actual de la calidad del código**
  Se busca determinar de forma objetiva cuál es el estado actual del código en términos de calidad.

- **Determinar cuáles microservicios deben mejorarse primero**
  Una manera de seguir mejorando la calidad del proyecto es identificar los componentes con mayor riesgo de afectar la calidad antes de que se conviertan en una amenaza, aplicando una actitud proactiva frente a los problemas que puedan ocurrir. Se busca responder la pregunta "¿Qué debo mejorar primero?"

Luego de realizar el análisis se expondrán las conclusiones resultantes del mismo.

El análisis fue realizado sobre todos los microservicios:

- *Calendar*
- *Directory*
- *Mailer*
- *Feedback*
- *Report*
- *Autodialer*

En donde para cada uno se evaluaron las siguientes métricas:

Métrica	Descricpión
Confiabilidad	Mide la confiabilidad del sistema en relación a la cantidad de bugs identificados por SonarQube. Los bugs son clasificados en 4 categorías según su severidad, las cuales buscan responder la pregunta "¿Qué es lo peor que puede ocurrir que ocasione que la aplicación quede indisponible o se pierda integridad de información, y cuál es la probabilidad de que eso suceda?". Estas categorías son: • `Minor` = desperfecto de calidad que puede impactar levemente la productividad del desarrollador (líneas de código demasiado largas, sentencias "switch" deberían contener al menos 3 casos, convenciones de nombres, no seguimiento de estándares, etc.) • `Major` = desperfecto de calidad que puede impactar significativamente la productividad del desarrollador (secciones grandes de código no cubierto por pruebas, bloques de código duplicados, parámetros no utilizados, ciclos en dependencias, etc.) • `Critical` = tanto bugs con baja probabilidad de impactar el comportamiento de la aplicación en producción como fallas de seguridad (NullPointerException, bloques try-catch vacíos, inyecciones SQL, etc.). Estos bugs deben ser revisados inmediatamente.

	• `Blocker` = bugs con alta probabilidad de impactar en el comportamiento de la aplicación en producción (memory leaks, conexiones JDBC no cerradas, sockets no cerrados, etc.). Estos bugs deben ser corregidos inmediatamente. Se asigna luego una calificación a cada clase del microservicio según la siguiente escala: • `A = 0 bugs` • `B = al menos 1 Minor bug` • `C = al menos 1 Major bug` • `D = al menos 1 Critical bug` • `E = al menos 1 Blocker bug` Finalmente la calificación del microservicio es igual a la calificación de la peor clase, es decir, aquella que contenga el bug más severo.
Seguridad	Analiza que el código cumpla con determinadas reglas de seguridad identificando vulnerabilidades que puedan ser explotadas por un usuario malintencionado. Estas reglas verifican que el código cumpla con los tres mayores estándares de seguridad: CWE, SANS Top 25 y OWASP Top 10. Al igual que los bugs, las vulnerabilidades son clasificadas en 4 categorías (Minor, Major, Critical y Blocker) y buscan responder la pregunta "¿Cuál es el mayor daño que pueda ocasionar la explotación de la vulnerabilidad y cual es la probabilidad de que un usuario malintencionado la explote?" Se asigna luego una calificación a cada clase del microservicio según la siguiente escala:

	• A = 0 vulnerabilidades • B = al menos 1 vulnerabilidad Minor • C = al menos 1 vulnerabilidad Major • D = al menos 1 vulnerabilidad Critical • E = al menos 1 vulnerabilidad Blocker La calificación del microservicio es igual a la calificación de la peor clase, es decir, aquella que contenga la vulnerabilidad más severa
Mantenibilidad	Califica la mantenibilidad del proyecto según su deuda técnica. Por deuda técnica se entiende al esfuerzo necesario para corregir todos los *code smells* presentes en el código. El valor de esta métrica está relacionado con el Ratio de Deuda Técnica (TDR), de donde: TDR = Esfuerzo deuda técnica / Esfuerzo total de desarrollo Se asigna luego una calificación al microservicio según su TDR: • A = TDR menor o igual a 5% • B = TDR entre 6% y 10% • C = TDR entre 11% y 20% • D = TDR entre 21% y 50% • E = TDR mayor a 50% Esta métrica es también denominada SQALE Rating[ref]
Duplicaciones de código	Mide el porcentaje de bloques de código duplicados en todo el sistema. Para el lenguaje Java se considera que un bloque está duplicado si existen al menos 10[ref] líneas sucesivas

	de código duplicado en todo el microservicio. Los comentarios, las diferencias de indentación y los valores constantes (números mágicos y strings literales) son ignorados mientras se buscan duplicaciones.

Cobertura de pruebas	Mide el porcentaje del código fuente que ha sido cubierto por al menos una prueba unitaria. Debido a las variaciones en los flujos de ejecución por las estructuras de control (if, for, while, etc.), puede ocurrir que todas las líneas de un método estén cubiertas por pruebas unitarias pero no todos sus posibles flujos hayan sido probados, por lo que se tiene en cuenta también si se probaron todas las posibles expresiones condicionales de estas estructuras de control.
	
	`Cobertura = (CT + CF + LC)/(2*B + L)`
	
	Donde:
	
	`CT = condiciones evaluadas en true al menos una vez`
	`CF = condiciones evaluadas en false al menos una vez`
	`LC = cantidad de líneas cubiertas`
	
	`B = cantidad total de condiciones`
	`L =  cantidad total de líneas`

Tabla 11.10.1 Métricas evaluadas

A continuación se presentan los resultados obtenidos luego de realizar el análisis:

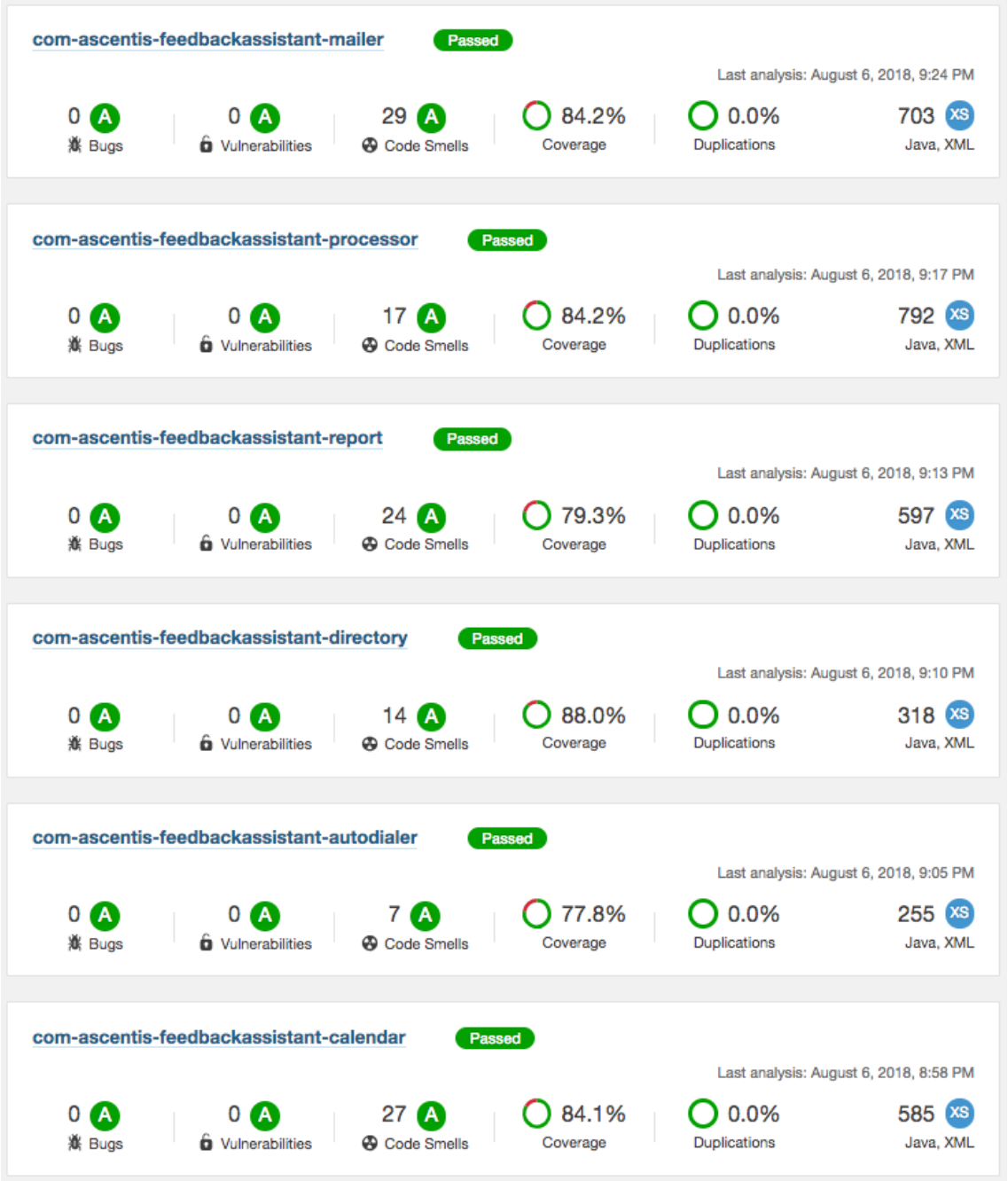


Ilustración 11.10.1 Resultado de análisis con SonarQube

La siguiente tabla permite visualizar los resultados obtenidos de una forma más clara.

Microservicio	Confiabilidad	Seguridad	Mantenibilidad	Duplicaciones de código	Cobertura de pruebas	Deuda técnica
Calendar	A	A	A	0%	84,1%	3h
Directory	A	A	A	0%	88%	1h
Mailer	A	A	A	0%	84,2%	5h
Feedback	A	A	A	0%	84,2%	2h
Report	A	A	A	0%	79,3%	4h
Autodialer	A	A	A	0%	77,8%	47min
						15h 47min

Tabla 11.10.2 Métricas de calidad del código

Se puede apreciar que la totalidad de los microservicios obtuvieron resultados excelentes para las distintas métricas analizadas. Estos resultados no fueron casualidad, sino que fueron producto de un proceso de mejora continua a lo largo de todo el desarrollo en donde el equipo iba corrigiendo los distintos problemas a medida que la herramienta los identificaba.

Es relevante mencionar que la arquitectura utilizada fue realmente beneficiosa para obtener estos resultados, ya que resultó mucho más sencillo mantener y mejorar componentes granulares que un único componente con la totalidad del código de la solución.

Se presentan a continuación distintas gráficas generadas con la herramienta utilizada, que permiten visualizar los resultados del análisis bajo distintas perspectivas. SonarQube utiliza gráficas de burbuja; las gráficas de burbuja además de representar medidas en una escala (como las gráficas de dispersión) permiten también representar una tercer medida utilizando el tamaño de la burbuja.

Perspectiva de riesgo

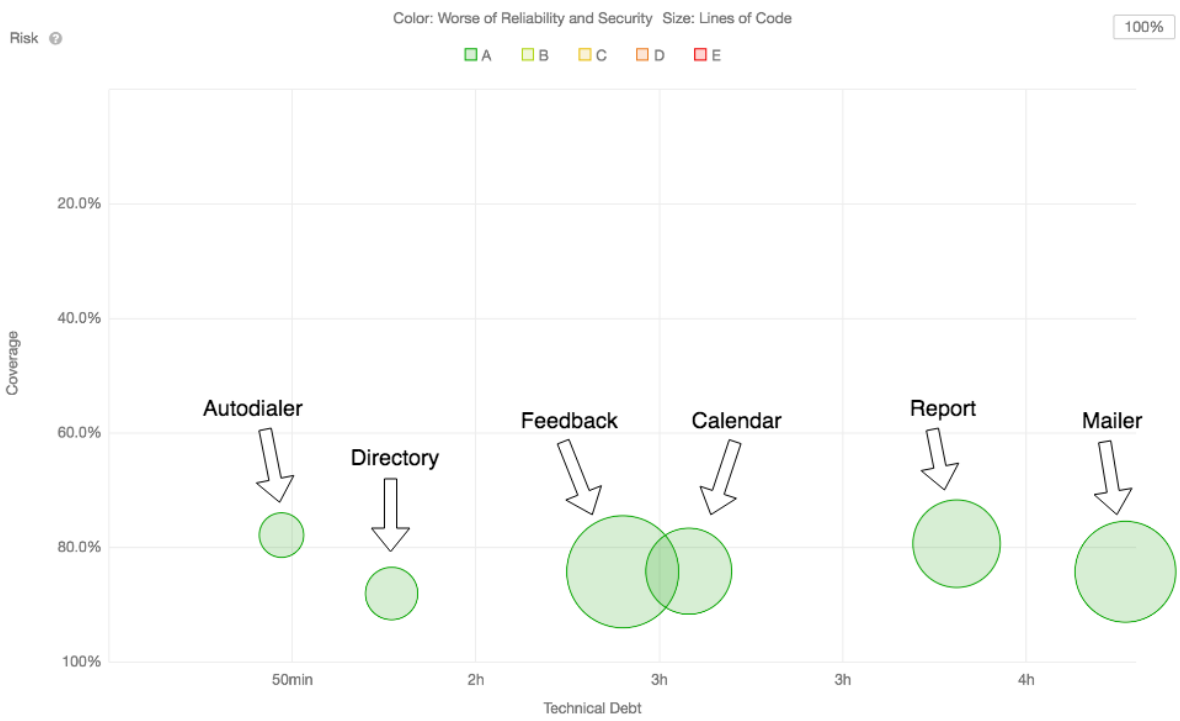


Ilustración 11.10.2 Perspectiva de riesgo

Esta perspectiva permite identificar rápidamente riesgos dentro de cada microservicio. Cualquier color distinto de verde indica un riesgo inmediato: bugs o vulnerabilidades que deben ser inmediatamente examinados. Una burbuja en la esquina superior derecha significa que la salud a largo plazo del microservicio puede estar en riesgo. El eje vertical representa la cobertura de pruebas, mientras que el horizontal representa la deuda técnica total del microservicio (en horas). El tamaño de cada burbuja representa la cantidad total de líneas de código del microservicio.

El microservicio que más destaca es Mailer al estar ubicado a la derecha del gráfico. Esto indica que es el microservicio con mayor deuda técnica de la solución, por lo que debería ser el primero en donde aplicar un refactoreo para disminuir el riesgo de que se convierta en un problema.

Por otro lado la totalidad de las burbujas son verdes, lo que indica que no se encontraron bugs ni vulnerabilidades durante el análisis.

Perspectiva de cobertura

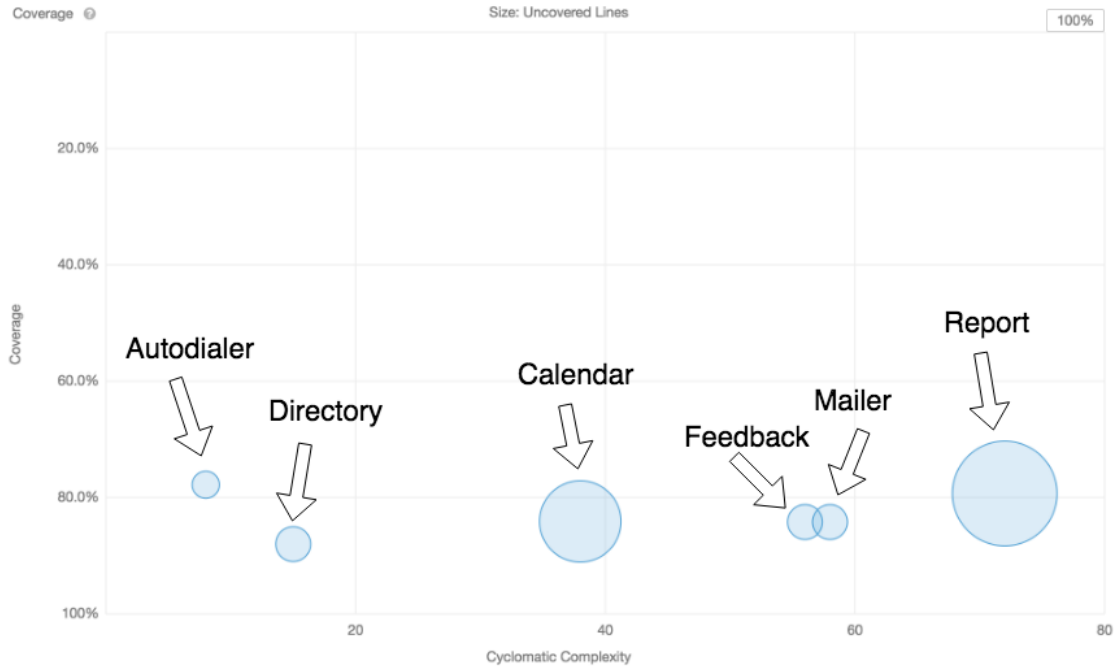


Ilustración 11.10.3 Perspectiva de cobertura

La perspectiva de cobertura permite identificar los riesgos resultantes por la falta de pruebas unitarias en microservicios complejos. En los ejes vertical y horizontal se representan el porcentaje de cobertura de pruebas y la complejidad ciclomática, mientras que el tamaño de las burbujas representan la cantidad de líneas de código no cubiertas por pruebas unitarias. Burbujas pequeñas en la parte inferior del gráfico son mejores.

Se puede apreciar que todos los microservicios se ubican en la zona inferior del gráfico, lo que indica una buena cobertura de pruebas, pero el microservicio Report se destaca al tener la mayor complejidad ciclomática y una mayor cantidad de líneas de código sin probar que el promedio. Esto indica que el microservicio Report debería ser el primero al que se le debería aumentar la cobertura de pruebas unitarias, buscando reducir la cantidad de líneas de código no cubiertas.

Perspectiva de mantenibilidad

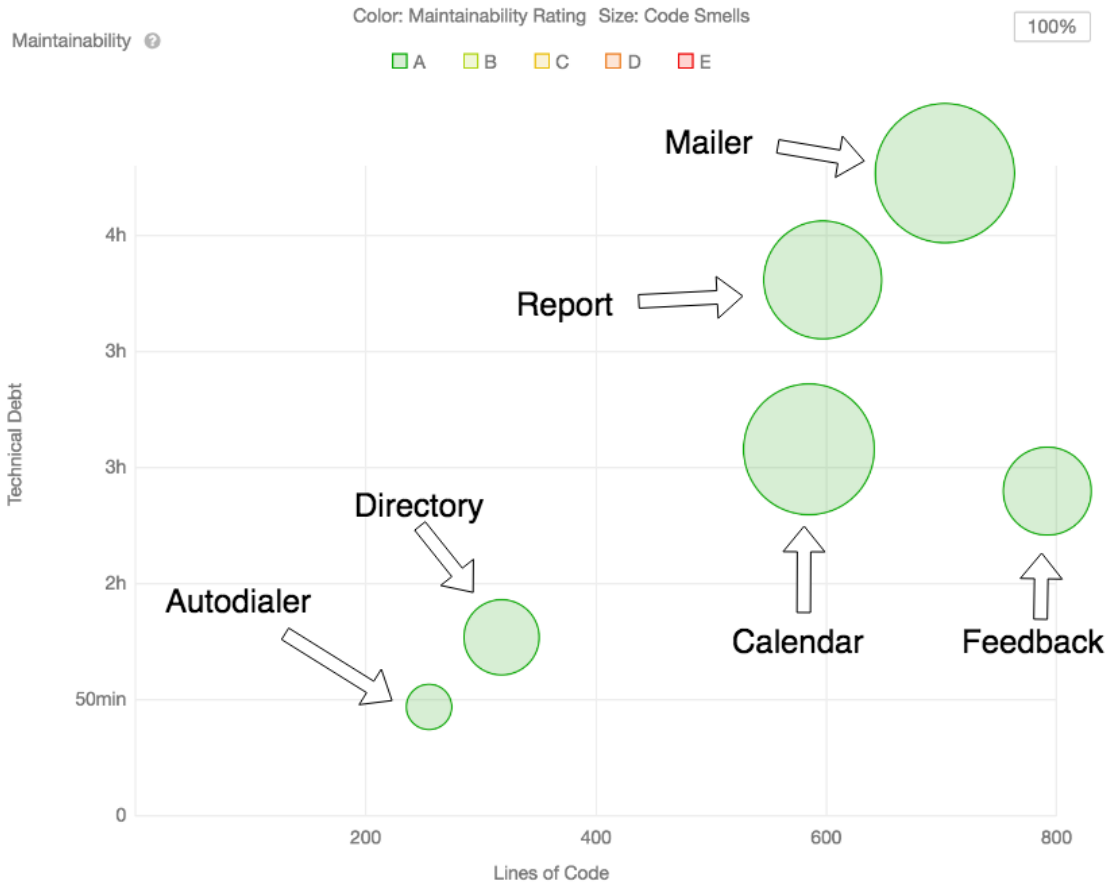


Ilustración 11.10.4 Perspectiva de mantenibilidad

Con la perspectiva de mantenibilidad se pueden ver los riesgos relacionados con los *code smells* y la deuda técnica, que podrían afectar negativamente a la mantenibilidad a largo plazo. Mientras más cercano es el color de la burbuja al rojo, mayor es la relación entre la deuda técnica y el tiempo total de desarrollo del microservicio. El tamaño de cada burbuja indica la cantidad de *code smells* encontrados, y su posición vertical indica la deuda técnica (en horas) del microservicio (es decir, el tiempo necesario para corregir todos sus *code smells*). El eje horizontal indica el tamaño del microservicio en cantidad de líneas de código. Burbujas verdes pequeñas cercanas al eje horizontal son mejores.

Coincide que los microservicios con mayor deuda técnica (los que se ubican más arriba en el gráfico) son Mailer y Report, lo que significa que son los

microservicios con mayor riesgo de presentar problemas de mantenibilidad en un futuro. Esto sugiere que son primeros microservicios en los que se debería dedicar esfuerzo para mejorar la mantenibilidad de la solución.

También se puede ver que todos los microservicios se presentan en verde, lo que denota que la deuda técnica de cada uno es mínima comparado con el tamaño de los proyectos.

Conclusiones

Del análisis realizado se puede concluir que se obtuvieron excelentes niveles de calidad para todas las métricas calculadas. El seguimiento de estas métricas a lo largo del proyecto permitieron implementar un plan de mejora continua que se enfocó que conseguir en todo momento altos niveles de calidad del código, lo cual fue considerado imprescindible al trabajar para un cliente real y por tratarse de una instancia académica que demanda excelentes resultados en todas sus dimensiones.

Además se reconoce que la gestión de la calidad no es una actividad puntual, sino un proceso transversal a las fases del proyecto, por lo que se analizó también dónde se encuentran las oportunidades de mejora en la solución construida. Se concluye del análisis que las mayores oportunidades de mejora se encuentran en los microservicios Mailer y Report, en donde se podría trabajar para aumentar aún más su mantenibilidad y confiabilidad realizando refactoreos que disminuyan los *code smells* identificados y construyendo pruebas unitarias que prueben las pocas secciones aún no cubiertas.

11.11 Manual de instalación y configuración

This document explains how to put Feedback Assistant up and running. Before starting the installation, there are some prerequisites that need to be complied.

Prerequisites:

- Java 8 or greater (https://www.java.com/en/download/help/download_options.xml)
- Apache Maven 3.5.x (https://maven.apache.org/install.html)
- Apache Tomcat 9.0.x (https://tomcat.apache.org/tomcat-9.0-doc/setup.html)
- Apache ActiveMQ 5.12.x (http://activemq.apache.org/getting-started.html)
    - After installing navigate to http://localhost:8161/admin/queues.jsp
    - Create two message queues called:
        - "raw_feedback_queue"
        - "error_feedback_queue
- PostgreSQL Server 10 (https://www.postgresql.org/download/windows/)
    - After installing create two empty databases called:
        - "calendar-db"
        - "feedback-db"

Feedback Assistant will have to access employee and managers data, so it is necessary to create a service account under Google G-Suite domain and grant certain permissions to it in order to synch with users data. This must be done by Google G-Suite Administrator/s user/s.

Link to guide for creating Feedback Assistant service account : https://support.google.com/a/answer/7378726?hl=en
After creating the service account, you must grant permission to it in order to enable Feedback Assistant to access users calendars and contacts. Follow the steps detailed on this link (https://developers.google.com/admin-sdk/directory/v1/guides/delegation#delegate_domain-wide_authority_to_your_service_account) and add the following URL's to the permissions:

- https://mail.google.com/
- https://www.googleapis.com/auth/admin.directory.user
- https://www.googleapis.com/auth/calendar

Download the .json private key generated after delegating domain wide authority, this will be referenced later in directory, calendar and feedback microservices.

Now is the moment to download source code.

- Open Git Shell.
- Run the following commands
    - git clone https://nicoeiris@bitbucket.org/feedbackassistant/mailer.git
    - git clone https://nicoeiris@bitbucket.org/feedbackassistant/calendar.git
    - git clone https://nicoeiris@bitbucket.org/feedbackassistant/directory.git
    - git clone https://nicoeiris@bitbucket.org/feedbackassistant/processor.git
    - git clone https://nicoeiris@bitbucket.org/feedbackassistant/autodialer.git
    - git clone https://nicoeiris@bitbucket.org/feedbackassistant/report.git
    - git clone https://nicoeiris@bitbucket.org/feedbackassistant/model.git

Feedback Assistant configuration

Now it is time to configure each microservice:

1. **Calendar**

- Paste the credential .json file downloaded previously from Google G-Suite on "…/src/main/resources/" directory.

- Open with a text editor and change values according to custom settings. "…/src/main/resources/application.properties".

```
server.context-path=/calendar

spring.application.name=Calendar

feedbackassistant.gsuite.appname=Feedback Assistant
feedbackassistant.gsuite.credential=Feedback Assistant-0571840f465f.json
feedbackassistant.gsuite.adminaccount=bruno.bellizzi@feedbackassistant.org
feedbackassistant.gsuite.domain=feedbackassistant.org

feedbackassistant.gsuite.synchevents.interval=60000
feedbackassistant.gsuite.feedback.request.interval=90000
feedbackassistant.gsuite.synchusers.interval=120000

feedbackassistant.mailer.url=http://localhost:8080/mailer/calendar
feedbackassistant.directory.url=http://localhost:8080/directory/users
feedbackassistant.autodialer.url=http://localhost:8080/autodialer/dial

spring.datasource.url=jdbc:postgresql://localhost:5432/calendar-db
spring.datasource.username=postgres
spring.datasource.password=root
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.hibernate.ddl-auto=update
```

Ilustración 11.11.1 Archivo configuración Calendar

- "feebackassistant.gsuite.synchevents.interval" refers to the period of time that the system will wait to synch events from Google G-suite in milliseconds.

- “feebackassistant.gsuite.feedback.request.interval” refers to the period of time that the system will wait to request mailer microservice to ask users for feedback in milliseconds.

- “feebackassistant.gsuite.synchusers.interval” refers to the period of time that the system will wait to request directory microservice synch users in milliseconds.

2. Directory

- Paste the credential .json file downloaded previously from Google G-Suite on “.../src/main/resources/” directory.

- Open with a text editor and change values according to custom settings. “.../src/main/resources/application.properties”.

```
server.context-path=/directory

feedbackassistant.gsuite.appname=Feedback Assistant
feedbackassistant.gsuite.credential=Feedback Assistant-0571840f465f.json
feedbackassistant.gsuite.adminaccount=bruno.bellizzi@feedbackassistant.org
feedbackassistant.gsuite.domain=feedbackassistant.org
```

Ilustración 11.11.2 Archivo configuración Directory

3. Mailer

- Customize feedback request and feedback error templates HTML´s. To customize them navigate to "…/src/main/resources/".
  - feedback-request.html
  - feedback-error.html

- Open using a text editor and change values according to custom settings. "…/src/main/resources/application.properties".

```
server.context-path=/mailer

feedbackassistant.mailer.sender.host=smtp.gmail.com
feedbackassistant.mailer.sender.port=587
feedbackassistant.mailer.receiver.protocol=imaps
feedbackassistant.mailer.receiver.host=imap.gmail.com
feedbackassistant.mailer.receiver.folder=INBOX
feedbackassistant.mailer.receiver.interval=10000
feedbackassistant.mailer.username=feedbackassistant@feedbackassistant.org
feedbackassistant.mailer.password=feedbackassistant
feedbackassistant.mailer.template.request=feedback-request.html
feedbackassistant.mailer.template.error=feedback-error.html
feedbackassistant.mail.uri=imaps://feedbackassistant%40feedbackassistant.org:feedbackassistant@imap.gmail.com/INBOX
feedbackassistant.mail.quickFeedbackUrl=http://localhost:8080/processor/feedback/quickFeedback

feedbackassistant.messaging.rawfeedback.queue=raw_feedback_queue
feedbackassistant.messaging.errorFeedback.queue=error_feedback_queue
feedbackassistant.messaging.broker.url=tcp://localhost:61616
feedbackassistant.messaging.broker.username=admin
feedbackassistant.messaging.broker.password=admin

spring.activemq.user=admin
spring.activemq.password=admin
spring.activemq.broker-url=tcp://localhost:61616

service.cache.timeout=5
service.cache.time-unit=MINUTES
service.cache.max-size=100

feedbackassistant.calendar.url=http://localhost:8080/calendar/events
```

Ilustración 11.11.3 Archivo configuración Mailer.

"feedbackassistant.mail.quickfeedbackurl" corresponds to the http resource assigned to receive instant feedback.

4. Autodialer

To setup the autodialer you need to sign up on Twilio and enable call service. They will send you an account ID, security token and the number of the dialer bot. (https://www.twilio.com/voice)

- Open using a text editor and change values according to custom settings. "…/src/main/resources/application.properties".

```
server.context-path=/autodialer

feedbackassistant.twilio.accountSsid=ACe3515281470a5d5658f0d7097e45045d
feedbackassistant.twilio.authToken=d980a76fc3a228e0e7adeeef225317ed
feedbackassistant.twilio.number=+1 669-238-2121

feedbackassistant.twilio.twiml.feedbackRequest=
http://179.27.65.138/autodialer/twiml/feedbackRequest?eventId={eventId}&userId={userId}&name={name}
feedbackassistant.twilio.twiml.feedbackRequest.message=
Hello. Please provide some feedback about {event} event, leaving a message after the beep.
feedbackassistant.twilio.twiml.feedbackResponse=http://179.27.65.138/autodialer/twiml/feedbackResponse
feedbackassistant.twilio.twiml.feedbackResponse.message=Thank you, your feedback have been received

feedbackassistant.messaging.rawfeedback.queue=raw_feedback_queue
feedbackassistant.messaging.broker.url=tcp://localhost:61616
feedbackassistant.messaging.broker.username=admin
feedbackassistant.messaging.broker.password=admin

spring.activemq.user=admin
spring.activemq.password=admin
spring.activemq.broker-url=tcp://localhost:61616
```

Ilustración 11.11.4 Archivo configuración Autodialer

- "feedbackassistant.twilio.accountSsid" set here the accountID Twilio have sent you after registering the call service.

- "feedbackassistant.twilio.authToken" set here the security token Twilio have sent you after registering the call service.

- "feedbackassistant.twilio.number" set here the dialer bot numer Twilio have sent you after registering the call service.

5. Processor

To configure the processor you need to sign up on IBM Watson and Microsoft Azure (Google is already covered) and enable respective natural language services. They will send you an account ID, security token and the number of the dialer bot.

- IBM Watson https://www.ibm.com/watson/services/natural-language-understanding/

- Microsoft Azure https://azure.microsoft.com/es-es/services/cognitive-services/text-analytics/

- Customize instant feedback response template HTML. To customize it navigate to ".../src/main/resources/quickfeedback-response".

```
server.context-path=/processor
feedbackassistant.gsuite.appname=Feedback Assistant
feedbackassistant.gsuite.credential=Feedback Assistant-0571840f465f.json
feedbackassistant.gsuite.domain=feedbackassistant.org

feedbackassistant.messaging.rawfeedback.queue=raw_feedback_queue
feedbackassistant.messaging.errorFeedback.queue=error_feedback_queue
feedbackassistant.messaging.broker.url=tcp://localhost:61616?jms.useAsyncSend=true
feedbackassistant.messaging.broker.username=admin
feedbackassistant.messaging.broker.password=admin

feedbackassistant.processor.step_count=3
feedbackassistant.processor.step_1=com.ascentis.feedbackassistant.processor.service.SentimentAnalysisStep
feedbackassistant.processor.step_2=com.ascentis.feedbackassistant.processor.service.TmsStep
feedbackassistant.processor.step_3=com.ascentis.feedbackassistant.processor.service.PersistenceStep

feedbackassistant.quickfeedback.responsetemplate=quickfeedback-response.html

spring.datasource.url=jdbc:postgresql://localhost:5432/feedback-db
spring.datasource.username=postgres
spring.datasource.password=root
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.hibernate.ddl-auto=update
service.cache.timeout=5
service.cache.time-unit=MINUTES
service.cache.max-size=100

azure.natural.language.api.key=0cab563883d743d1b049375e8fac163a
azure.natural.language.api.host=https://eastus.api.cognitive.microsoft.com
azure.natural.language.api.path=/text/analytics/v2.0/sentiment

watson.natural.language.api.path=https://gateway.watsonplatform.net/natural-language-understanding/api
watson.natural.language.api.username=ac2bf48c-c1e2-4133-b0b0-6104617fee49
watson.natural.language.api.password=o0Fob67cl0lT
watson.natural.language.api.version=2018-03-19
```

Ilustración 11.11.5 Archivo configuración Feedback.

- “feedbackassistant.processor.step.count” corresponds to the number of steps involved in the feedback processing flow. This enables to change them without compiling the solution.

- “feedbackassistant.processor.step_1” corresponds to the bean namespace of the first step of the processing flow.

- “feedbackassistant.processor.step_2” corresponds to the bean namespace of the first step of the processing flow.

- “feedbackassistant.processor.step_1” corresponds to the bean namespace of the first step of the processing flow.

- “feedbackassistant.quickfeedback.responsetemplate” corresponds to the response retrieved to the user after sending quick feedback.(thumbs up & down).

- “azure.natural.language.api.key” corresponds to the string key provided by Microsoft after registering the natural language service.

- “azure.natural.language.api.host” corresponds to the service endpoint resource.

- “azure.natural.language.api.key” corresponds to the specified service URI used to analyse text sentiment.

- “watson.natural.language.api.path” corresponds to the natural language service endpoint resource.

- “watson.natural.language.api.username” corresponds to the username of the user who registered the service on IBM.

- “watson.natural.language.api.password” corresponds to the password of the user who registered the service on IBM.

- “watson.natural.language.api.version” corresponds to the version of the service to use.

6. Report

Customize reports templates JRXML. To customize them navigate to “.../src/main/resources/”.

- daily_digest.jrxml.
- weekly_digest.jrxml.
- monthly_digest.jrxml.

```
server.context-path=/report

feedbackassistant.calendar.url=http://localhost:8080/calendar/events
feedbackassistant.directory.url=http://localhost:8080/directory/users/byEmail
feedbackassistant.processor.url=http://localhost:8080/processor/feedback
feedbackassistant.mailer.url=http://localhost:8080/mailer/send
feedbackassistant.reports.daily.interval=86400000
feedbackassistant.reports.weekly.interval=86400000
feedbackassistant.reports.monthly.interval=86400000
feedbackassistant.reports.daily.template=daily-report.jrxml
feedbackassistant.reports.weekly.template=weekly-report.jrxml
feedbackassistant.reports.monthly.template=monthly-report.jrxml
feedbackassistant.reports.managers=
bruno.bellizzi@feedbackassistant.org,nicolas.eiris@feedbackassistant.org
```

Ilustración 11.11.6 Archivo configuración Report

- “feedbackassistant.reports.daily.interval” corresponds to the period of time in which daily digests will be sent to managers.

- “feedbackassistant.reports.weekly.interval” corresponds to the period of time in which weekly digests will be sent to managers.

- “feedbackassistant.reports.monthly.interval” corresponds to the period of time in which monthly digests will be sent to managers.

- “feedbackassistant.reports.managers” corresponds to the email of the managers to whom you want to send the generated reports. If there are more than one, they must be separated by a coma.

Feedback Assistant installation

After configuring each microservice by editing configuration files, you must generate the different artifacts to make the deploy on the Apache Tomcat servers.

1. Navigate to Model class library directory.
2. Open command prompt on this directory.
3. Execute mvn install.

4. Navigate to Calendar microservice.
5. Open command prompt on this directory.
6. Execute mvn install.

7. Navigate to Directory microservice.
8. Open command prompt on this directory.
9. Execute mvn install.

10. Navigate to Mailer microservice.
11. Open command prompt on this directory.
12. Execute mvn install.

13. Navigate to Autodialer microservice.
14. Open command prompt on this directory.
15. Execute mvn install.

16. Navigate to Feedback microservice
17. Open command prompt on this directory
18. Execute mvn install.

19. Navigate to Report microservice.
20. Open command prompt on this directory.
21. Execute mvn install.

Once the artifacts are generated successfully, you have to navigate to Feedback Assistant directory on Maven local repository "C:/Users/[user]/.m2/repository/com/ascentis/feedbackassistant".

Copy the .war file inside each folder and paste it on /webapps folder of the corresponding Apache Tomcat server.

Feedback Assistant execution

Now you are able to start the different applications, first check you have executed the different services listed before such as PostgreSQL and Apache ActiveMQ.

After that check, enter /bin folder of Apache Tomcat and execute startup.bat (or startup.sh) file.

11.12 Encuestas de satisfacción del cliente

1. Satisfacción del cliente - MVP

A continuación se presentan los resultados de la encuesta realizada al cliente luego de la demo donde se expuso el MVP. Ésta encuesta fue realizada mediante Google Forms y consistió en cuatro preguntas de múltiple opción que refieren al producto y el equipo.

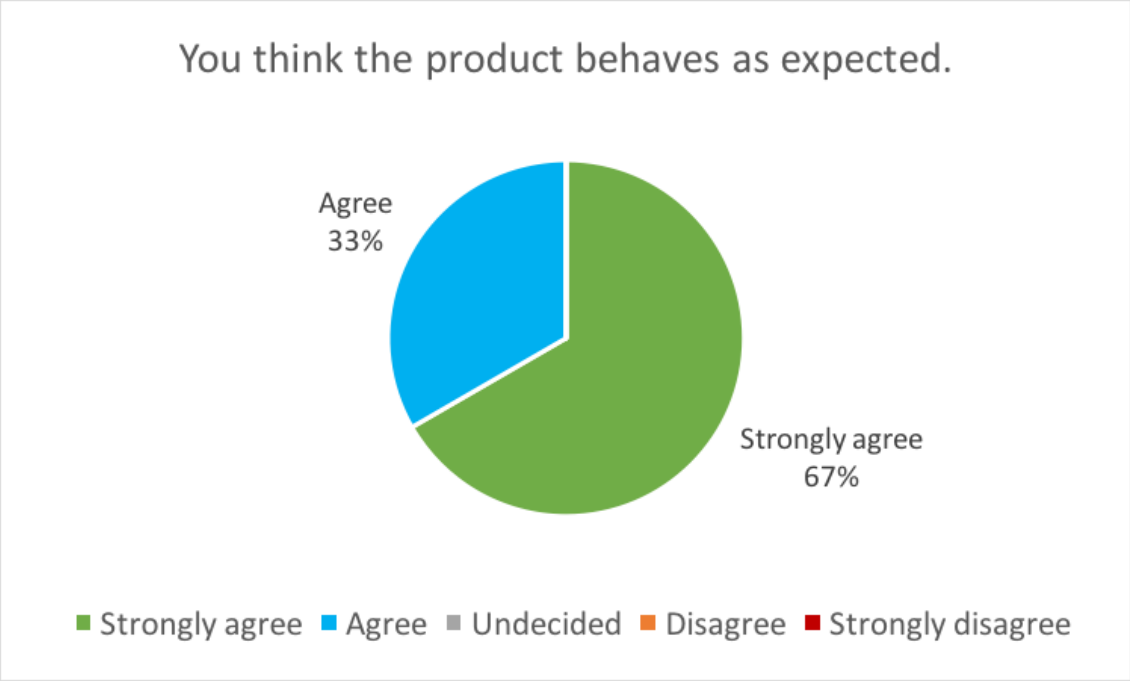


Ilustración 11.12.1 Afirmación 1 encuesta de satisfacción *release 1*

La primer pregunta hace referencia a si el producto se comporta como esperado, se puede observar que dos de los participantes están muy de acuerdo y uno de acuerdo.

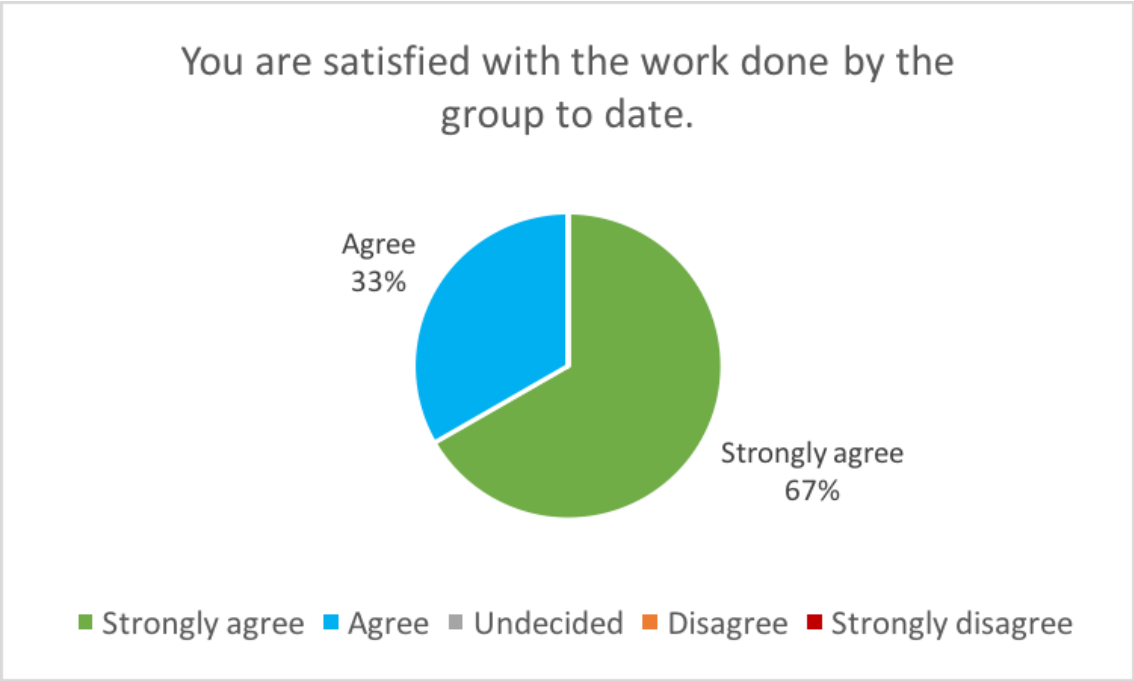


Ilustración 11.12.2 Afirmación 2 encuesta de satisfacción *release 1*

La segunda pregunta refiere a si el cliente está satisfecho con el trabajo realizado por el grupo hasta el momento. También se observan resultados positivos, donde dos personas contestan que están muy de acuerdo y una de acuerdo.

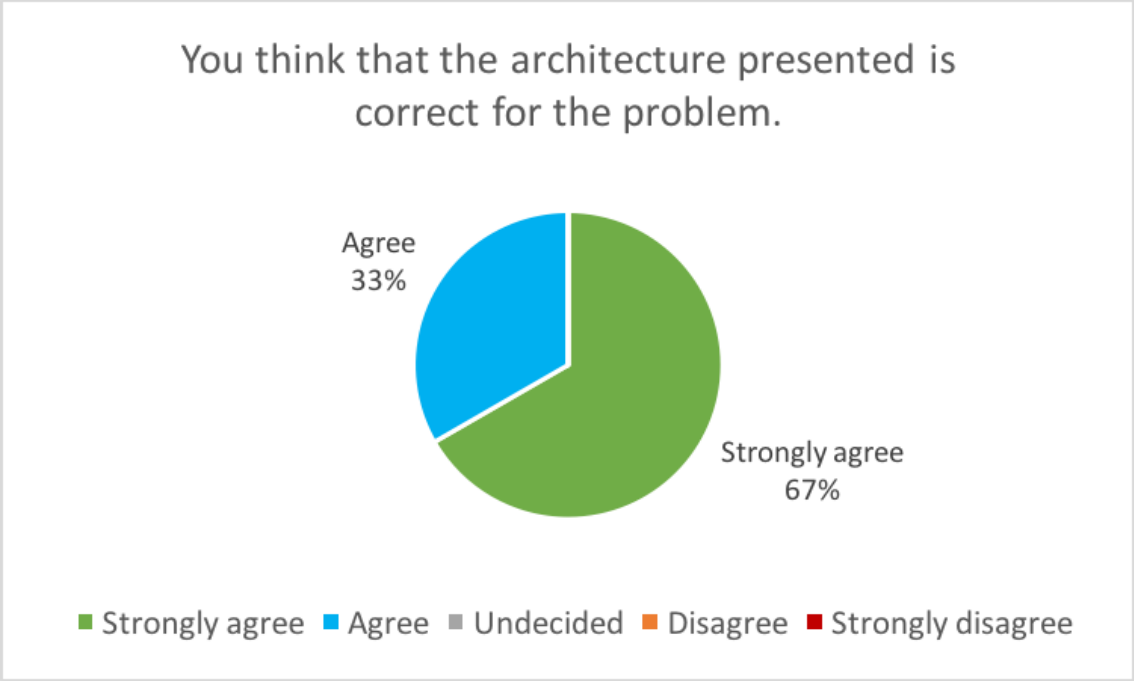


Ilustración 11.12.3 Afirmación 3 encuesta de satisfacción *release 1*

La tercer pregunta hace referencia a la arquitectura presentada para el problema y si la misma es adecuada. Las respuestas muestran que el cliente cree que la misma es correcta ya que dos de los participantes están muy de acuerdo y uno de acuerdo.

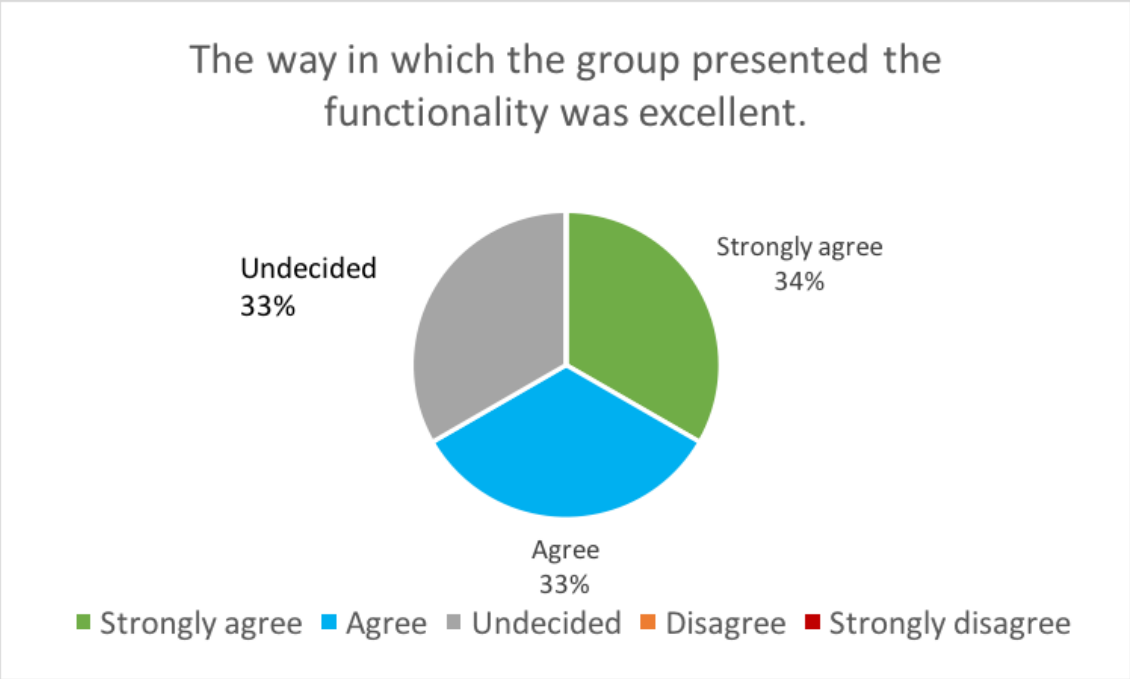


Ilustración 11.12.4 Afirmación 4 encuesta de satisfacción *release 1*

Por último se pregunta si la forma en que la funcionalidad fue presentada fue excelente. Se puede observar que la gráfica muestra una satisfacción indecisa, esto refiere a que en esta demo no se contaba aún con el ambiente de producción y el equipo debió mostrar el producto utilizando el IDE referente al ambiente de desarrollo.

La satisfacción promedio de Ascentis en esta encuesta fue de un 90% (4,5/5). Esto refleja que el cliente está muy satisfecho con el MVP y con el trabajo realizado por el grupo, cumpliendo por el momento con el objetivo de contar con altos niveles de satisfacción por parte del cliente.

2. Satisfacción final del cliente

A continuación se presentan los resultados de la encuesta final realizada al cliente para evaluar su satisfacción. Ésta fue enviada luego de realizar la entrega formal y su respectiva demo del segundo *release*.

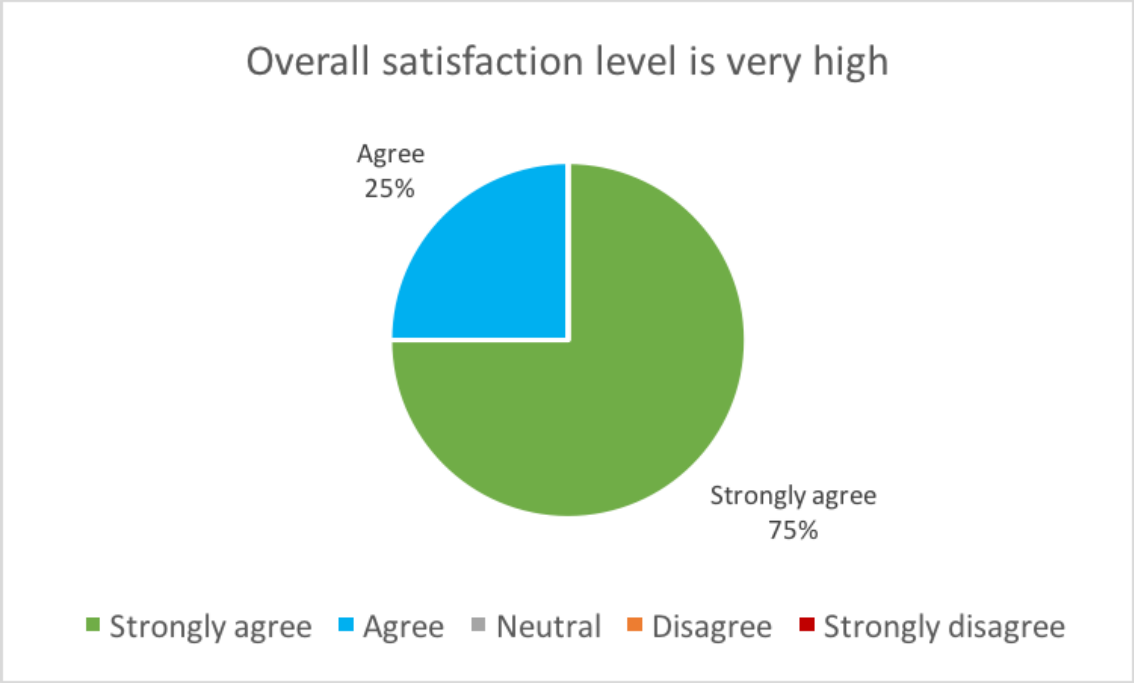


Ilustración 11.12.5 Afirmación 1 encuesta final de satisfacción

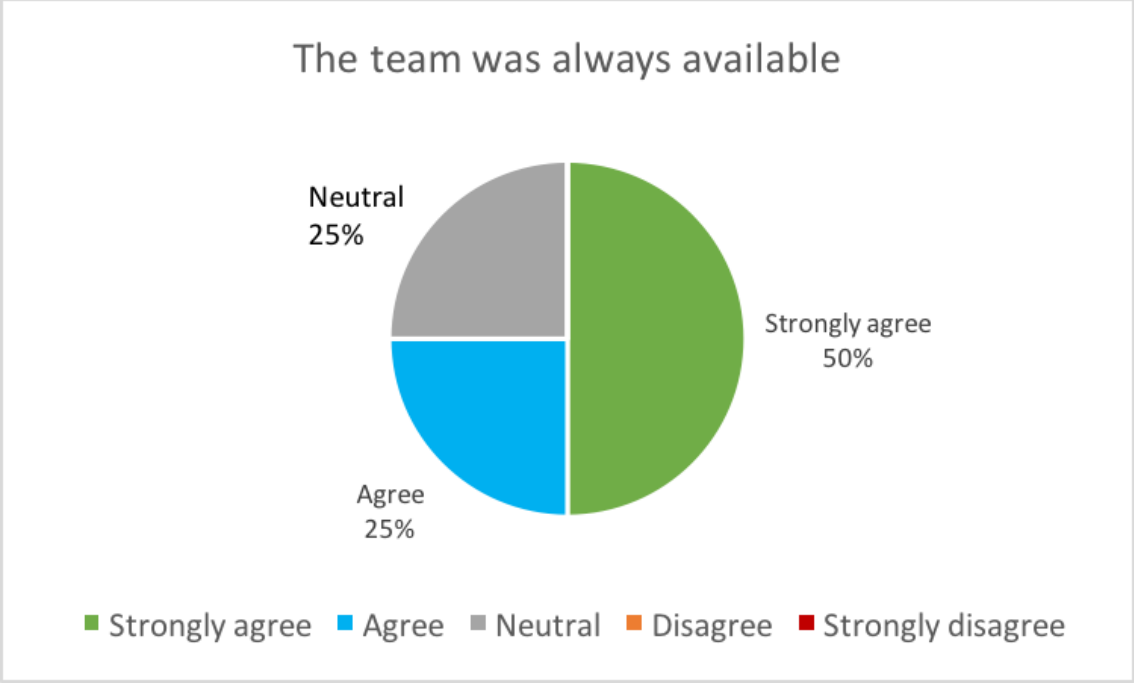


Ilustración 11.12.6 Afirmación 2 encuesta final de satisfacción

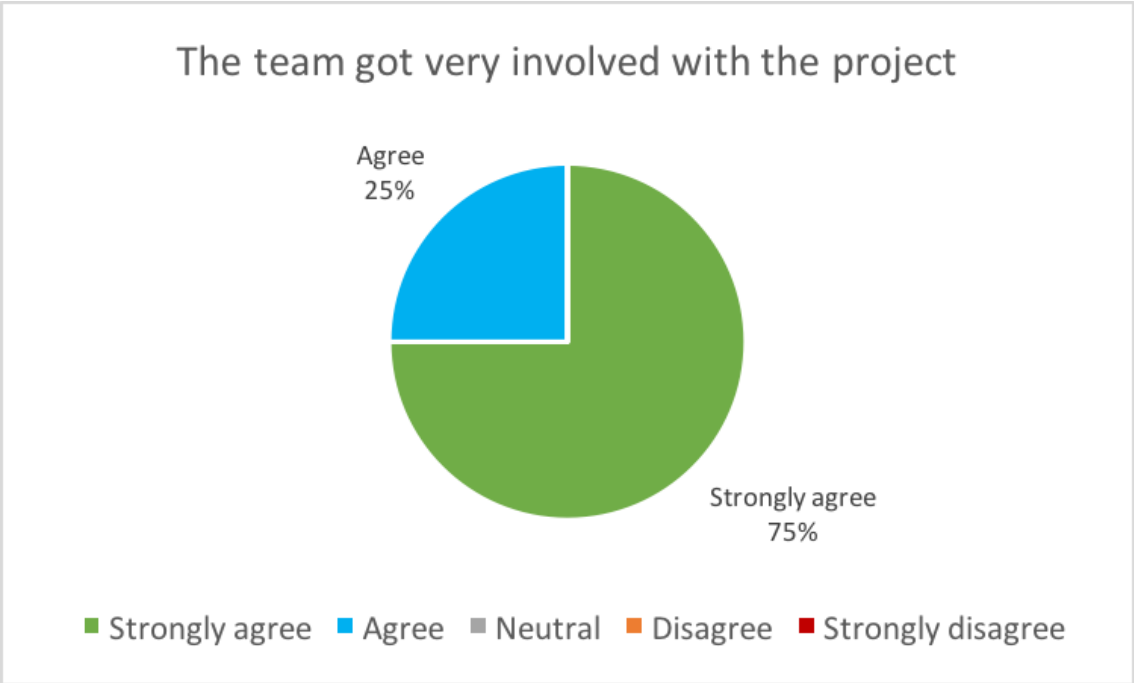


Ilustración 11.12.7 Afirmación 3 encuesta final de satisfacción

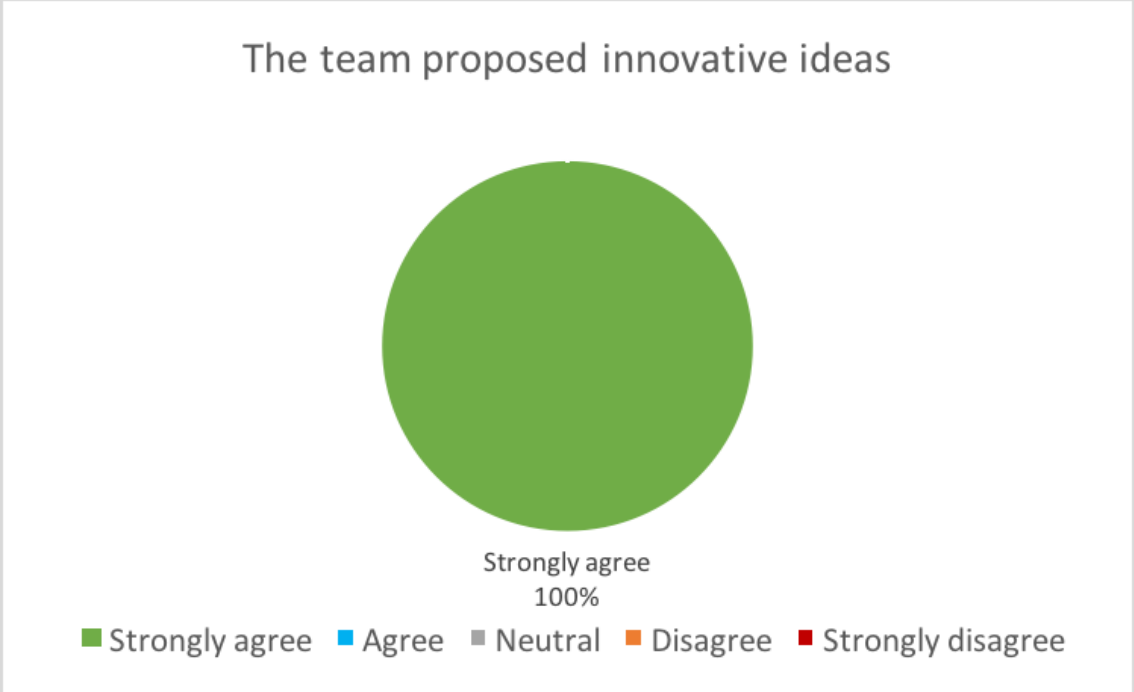


Ilustración 11.12.8 Afirmación 4 encuesta final de satisfacción.

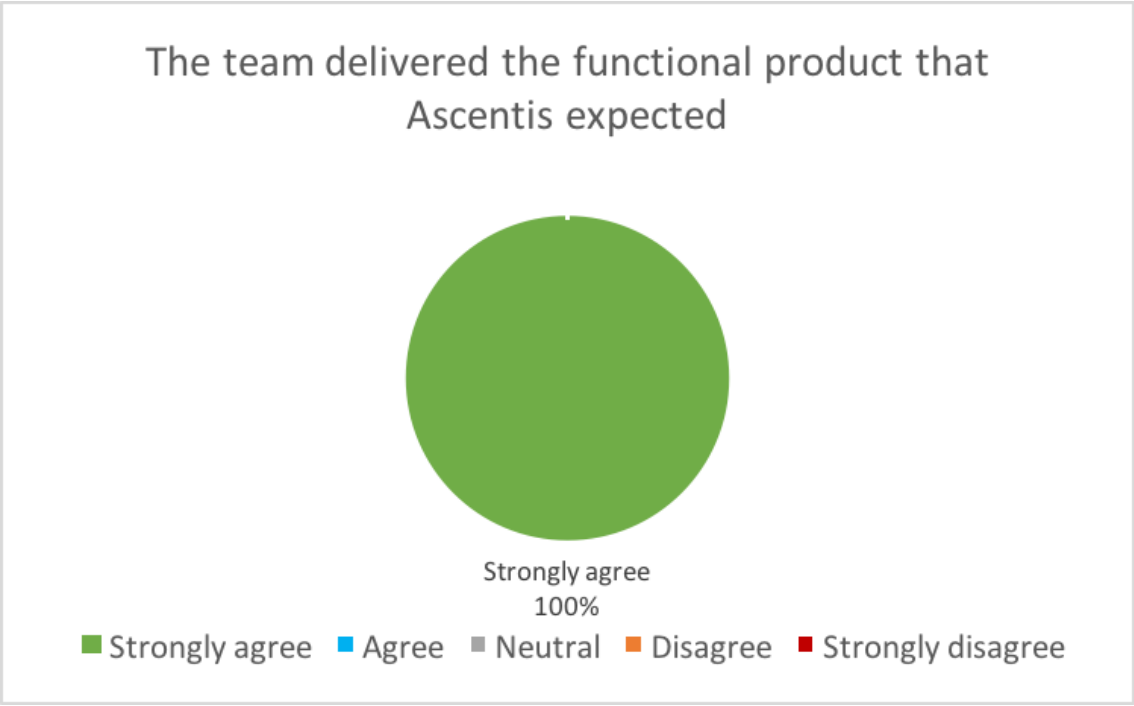


Ilustración 11.12.9 Afirmación 5 encuesta final de satisfacción

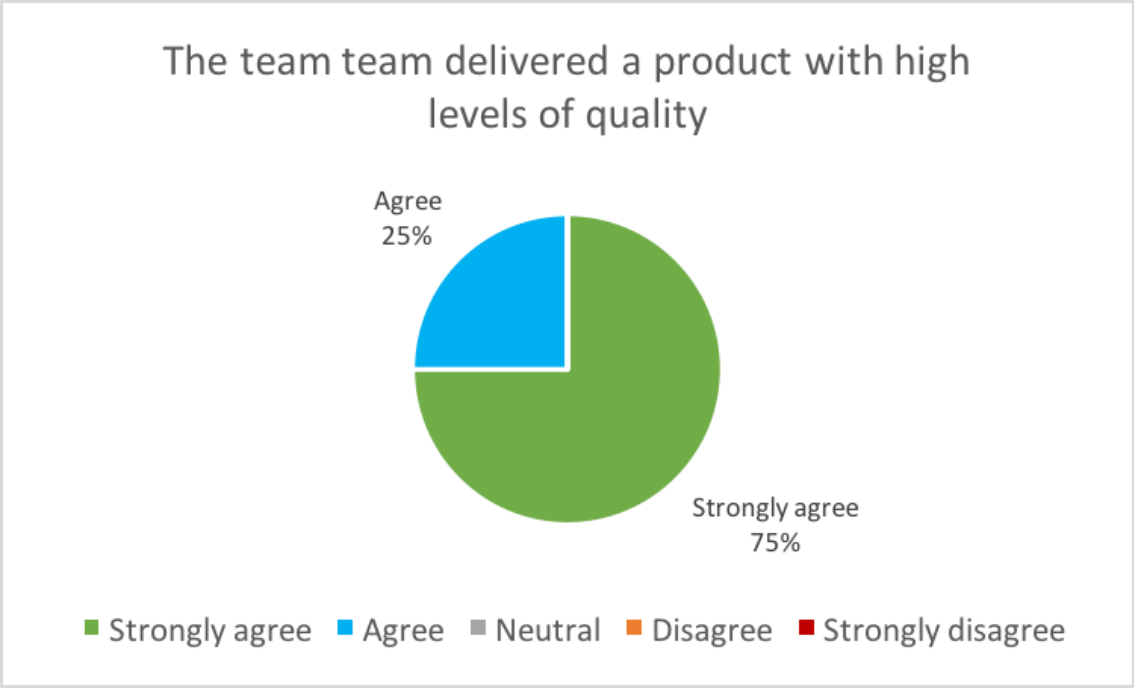


Ilustración 11.12.10 Afirmación 6 encuesta final de satisfacción.

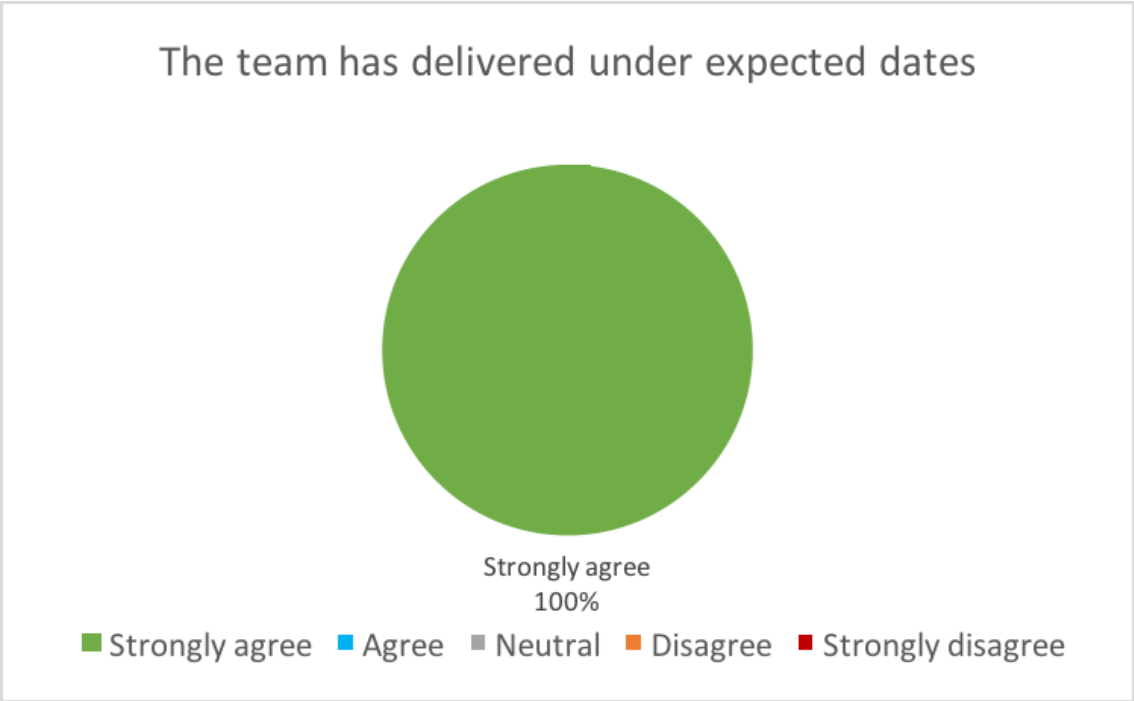


Ilustración 11.12.11 Afirmación 7 encuesta final de satisfacción

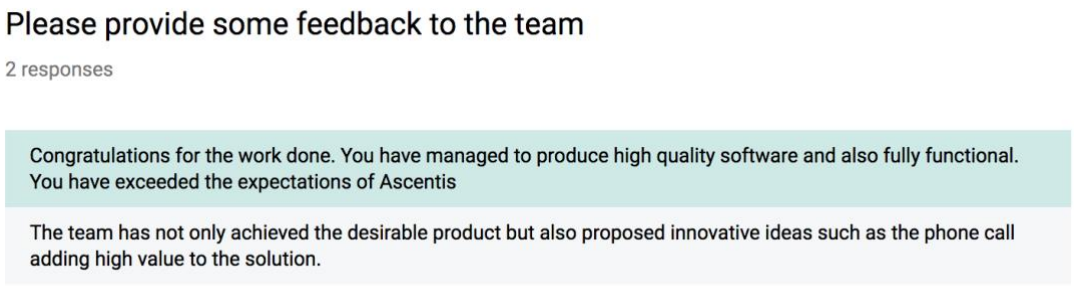


Ilustración 11.12.12 Comentarios encuesta final de satisfacción

Los resultados de la encuesta reflejan claramente el alto nivel de satisfacción por parte del cliente. El promedio de satisfacción es de un 95% (4,75/5), se logró alcanzar uno de los objetivos del proyecto asociado a cumplir y superar las expectativas del cliente. Vale observar también los comentarios opcionales, donde se felicita al equipo por el trabajo realizado, por cómo se lograron cumplir los niveles de calidad deseados por Ascentis así como también la propuesta de nuevas ideas innovadoras.

11.13 Especificación de la API asociada a TMS

Como fue mencionado en la sección 6.5.4 Evolución y seguimiento, el riesgo tecnológico asociado a la no liberación de la API de TMS por parte del cliente para la integración llego al punto de materializarse. Igualmente, el mismo envió una especificación de la misma donde se detallan las operaciones disponibles junto con sus parámetros y modelo asociado. Ésto permitió al equipo ejecutar el plan de contingencia del riesgo correspondiente y así continuar con el desarrollo simulando el comportamiento referente a la integración.

A continuación se muestra el contenido del documento de especificación de la API:

```
module FeedbackService
 module FeedbackSwagger
   extend ActiveSupport::Concern
   included do
     swagger_controller :feedback, "Feedback"

     swagger_api :create do
       summary "Create new Feedback"
       param :path , :customer_instance , :string   , :required , "Customer Name"
       param :body , :body              , :Feedback , :required , "Feedback to Create"
     end

     swagger_model :Feedback do
       " warning! The fields available on this model change depending on feedback type!
         Supported feedback_type are:
         (G)eneral ******* For first stages only use this one. *******
         (A)ctivity
         (C)ompetency
         (Go)al
         (Ac)complishment
         (O)verallRating"
       property :feedback_id    , :integer , :optional , "The feedback's ID(All Feedback)"
       property   :feedback_type    ,   :string      ,   :optional   ,   "The   type   of
feedback(G,A,C,Go,Ac,O)"
       property :create_date    , :string  , :optional , "Created Date"
       property :create_user    , :guid    , :optional , "User providing feedback"
       property :feedback       , :string  , :optional , "The feedback content"
       property :rating         , :deciaml , :optional , "The rating generated"
       property :session_id     , :integer , :optional , "Feedback Session ID (A)"
       property :user_evaluation_id   , :integer , :optional , "User evaluation"
     end
   end
 end
end
```
