Desarrollo de aplicación web para la gestión de proyectos ágiles en desarrollo de software

Kahyberth Stiven Gonzalez Sayas
Kevin Andres Ordoñez Villegas

Universidad del Valle
Facultad de ingeniería
Escuela de ingeniería de sistemas y computación
Tuluá-Valle
2024

Desarrollo de aplicación web para la gestión de proyectos ágiles en desarrollo de software

Kahyberth Stiven Gonzalez Sayas
Código 202060121
kahyberth.gonzalez@correounivalle.edu.co
Kevin Andres Ordoñez Villegas
Código 202060301
ordonez.kevin@correounivalle.edu.co

Director
Ign. Héctor Fabio Ocampo Arbeláez
hector.ocampo@correounivalle.edu.co

Universidad del Valle
Facultad de ingeniería
Escuela de ingeniería de sistemas y computación
Tuluá, Valle del Cauca
2024

Tabla de Contenido

Resumen

En el entorno actual del desarrollo de software, la implementación efectiva de metodologías ágiles resulta fundamental para el éxito de los proyectos. Sin embargo, muchas de las herramientas existentes presentan limitaciones en sus versiones gratuitas, lo que afecta especialmente a estudiantes y desarrolladores independientes con recursos limitados. En respuesta a esta problemática, se desarrolló una aplicación web de gestión de proyectos ágil, enfocada en ofrecer una solución gratuita, accesible y funcional.

La plataforma permitió la creación y gestión de tableros Kanban, la generación de informes del backlog, la integración de roles como Product Owner, Scrum Master y Team Member, y la creación de equipos. Además, incluyó funcionalidades como chat integrado, Planning Poker y una interfaz responsive adaptable a distintos dispositivos. El sistema fue diseñado bajo principios de usabilidad y escalabilidad, y se validó mediante pruebas funcionales. Esta herramienta demostró ser una alternativa eficaz frente a soluciones comerciales, permitiendo a equipos ágiles mejorar su productividad y coordinación sin restricciones de acceso ni costo.

Adicionalmente, la aplicación podría representar una herramienta valiosa en el entorno académico, especialmente en programas como Ingeniería de Sistemas, donde su uso puede ser integrado en cursos relacionados con el desarrollo de software, metodologías ágiles o trabajo colaborativo.

Palabras claves: Agile software development, Agile project management, Scrum (Software development), Software development management, Open-source software.

1. Introducción

En el contexto del desarrollo de software, las metodologías ágiles emergieron como una respuesta a la necesidad de flexibilidad, colaboración y entrega continua de valor. Estas metodologías, entre las que destacan Scrum, Kanban y Extreme Programming (XP), permitieron transformar la forma en que los equipos abordan la planificación, ejecución y entrega de proyectos tecnológicos. Según Schwaber y Sutherland [5], el desarrollo ágil promueve un marco de trabajo iterativo e incremental donde la adaptabilidad al cambio y la comunicación constante entre los miembros del equipo son pilares fundamentales.

Pese a su popularidad, la adopción de herramientas que permiten gestionar proyectos ágiles de forma efectiva ha enfrentado una limitación importante: muchas de las plataformas más reconocidas, como Jira, Trello o Asana, restringen funcionalidades clave en sus versiones gratuitas. Estas restricciones impactan especialmente a estudiantes universitarios, desarrolladores independientes y equipos de bajo presupuesto, quienes no siempre cuentan con acceso a soluciones premium. Tal como se expuso en este trabajo, estas herramientas imponen límites en cuanto a número de usuarios, integraciones, reportes, capacidad de personalización e incluso restricciones en el volumen de notificaciones diarias.

Con base en esta problemática, se desarrolló una aplicación web orientada a la gestión de proyectos ágiles, con un enfoque en la accesibilidad, escalabilidad y usabilidad. Esta plataforma fue concebida para ofrecer de manera gratuita un conjunto completo de funcionalidades necesarias para el seguimiento ágil de proyectos, integrando tableros Kanban, visualizaciones del estado del backlog y métricas del proyecto, gestión de equipos y roles (como Scrum Master, Product Owner, Desarrollador, entre otros), así como herramientas colaborativas como un módulo de chat en tiempo real y Planning Poker.

El diseño e implementación de la aplicación se apoyó en los principios del diseño centrado en el usuario (UCD), asegurando que la interfaz fuera intuitiva, responsiva y coherente con las expectativas de los usuarios finales. Además, la arquitectura del sistema se basó en microservicios, lo que permitió segmentar la lógica de negocio y mejorar la mantenibilidad, escalabilidad y despliegue de cada módulo.

Durante el proceso de levantamiento de requerimientos, se aplicaron encuestas a potenciales usuarios del sistema, entre ellos estudiantes universitarios y miembros de equipos de desarrollo, con el fin de identificar qué funcionalidades eran más valoradas en una herramienta de gestión ágil. Entre estas se destacaron la visualización clara del progreso mediante tableros, la facilidad de uso y la priorización de tareas. La implementación de estas características respondió directamente a los resultados obtenidos, alineando las funcionalidades del sistema con las necesidades reales del usuario (ver capítulo 5).

Asimismo, se adoptó un enfoque iterativo en el diseño de la interfaz, lo que permitió incorporar mejoras progresivas basadas en la retroalimentación de los usuarios, fortaleciendo así la experiencia final del sistema.

Cabe destacar que esta solución no solo fue pensada para su aplicación en contextos profesionales o de desarrollo independiente, sino también como una herramienta de apoyo para el ámbito académico. Dado que en programas como Ingeniería de Sistemas se incluyen asignaturas enfocadas en el desarrollo de software, la plataforma desarrollada puede ser utilizada en cursos prácticos donde los estudiantes deban gestionar proyectos en equipo. De este modo, se convierte en un recurso que complementa el proceso formativo, facilitando la adopción de buenas prácticas desde los primeros semestres de la carrera.

Esta propuesta se fundamentó en un marco teórico sólido. Según Bertalanffy [3], los equipos de trabajo pueden entenderse como sistemas abiertos y adaptativos donde las interacciones entre individuos, procesos y tecnología determinan el rendimiento global. De igual forma, estudios como los de Juhola [9] y Rönn [8] han evidenciado que, ante la falta de soluciones comerciales adecuadas, el desarrollo de herramientas personalizadas puede representar una alternativa eficaz, siempre y cuando se alineen con las necesidades del usuario final.

El desarrollo de esta aplicación respondió a una necesidad real en entornos académicos y profesionales: contar con una plataforma gratuita, funcional y adaptable para gestionar proyectos ágiles. El proyecto combinó un enfoque técnico riguroso con una comprensión profunda del contexto del usuario, lo que dio como resultado una herramienta de valor tanto para desarrolladores como para estudiantes universitarios.

2. Formulación del problema .

2.1. Descripción del problema

En el ámbito del desarrollo de software, las herramientas de gestión de proyectos como Jira, Asana y Trello facilitan que los equipos de desarrollo de software colaboren, supervisen y planifiquen sus proyectos. Sin embargo, muchas de estas herramientas solo ofrecen funciones que garantizan eficiencia que se encuentran en el modo de pago. las versiones gratuitas tienen limitaciones significativas en términos de capacidad y funcionalidades, Estas limitaciones pueden afectar principalmente a estudiantes y desarrolladores independientes con poco presupuesto. [1]

Los planes de Jira pueden tomarse como ejemplo para ilustrar estas limitaciones. La versión gratuita de Jira permite obtener datos básicos sobre el trabajo en equipo registrado en cualquier producto o proyecto, así como vistas de backlog, lista, tablero, cronograma, calendario y resumen. Esta versión gratuita, sin embargo, presenta limitaciones importantes, como un límite de hasta 10 usuarios y un máximo de 100 correos electrónicos por día. Las notificaciones se suspenden hasta el día siguiente una vez alcanzado este límite. Por otro lado, los planes de pago de Jira, a partir de 7 dólares por usuario, proporcionan funcionalidades adicionales, como roles y permisos de usuario avanzados, un número ilimitado de usuarios y herramientas para la planificación y gestión de dependencias entre varios equipos. [2]

Para estudiantes y desarrolladores independientes, no solo el costo de las versiones pagas de herramientas de gestión de proyectos representa un desafío, sino también las limitaciones impuestas por las versiones gratuitas. Los estudiantes, siendo el caso más común, suelen enfrentarse a restricciones financieras que dificultan el acceso a herramientas completas y efectivas.

Por lo tanto, es crucial contar con una solución gratuita que no imponga limitaciones severas en el tamaño del equipo o en las funcionalidades disponibles. Una herramienta de gestión de proyectos accesible y completa permitiría a los estudiantes gestionar sus proyectos de manera más eficiente, colaborar efectivamente con sus compañeros y adaptarse rápidamente a los cambios. Esta necesidad es particularmente relevante en un entorno académico, donde los recursos suelen ser limitados y la capacidad de utilizar herramientas eficaces puede marcar una diferencia significativa en el éxito de los proyectos.

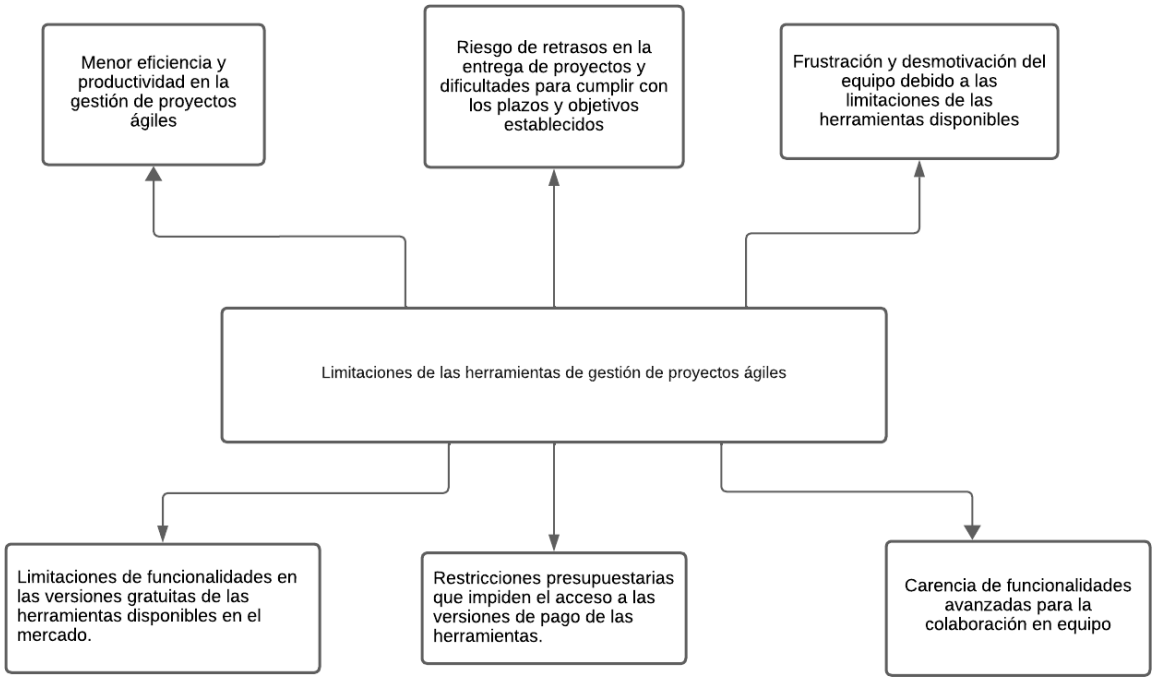

Figura 1: Árbol de problemas

2.2. Definición del problema

¿Cómo se puede abordar las limitaciones funcionales que proporcionan los software de gestión de proyectos ágiles a través de una aplicación web y de libre acceso?

3. Marco de Referencia

3.1. Marco teórico

Teoría de Sistemas: Esta teoría, propuesta por Bertalanffy (1968), postula que los sistemas son entidades complejas compuestas por componentes interrelacionados que funcionan juntos para lograr un objetivo común. Von Bertalanffy sostiene que los sistemas exhiben propiedades emergentes que no pueden explicarse únicamente por el análisis de sus partes individuales, enfatizando así la importancia de un enfoque holístico para comprender su funcionamiento[3]. En el contexto de la gestión de proyectos ágiles, esta teoría proporciona un marco para entender la dinámica de los equipos ágiles como sistemas adaptativos complejos, donde las interacciones entre individuos, procesos y tecnología afectan el rendimiento general del proyecto. Los equipos ágiles pueden considerarse sistemas abiertos que interactúan con su entorno, respondiendo de manera flexible a los cambios en los requisitos del proyecto y adaptándose continuamente para alcanzar sus objetivos.

Fundamentos de la Gestión de Proyectos Ágiles: La esencia de la gestión de proyectos ágiles se basa en la flexibilidad, colaboración y entrega continua de valor al cliente, influenciada principalmente por el Manifiesto Ágil. Según el Manifiesto Ágil, se prioriza Ïndividuos e interacciones sobre procesos y herramientas", y Respuesta ante el cambio sobre seguir un plan". Estos principios se implementan en metodologías como Scrum, que estructuran el trabajo de manera iterativa e incremental a través de roles definidos, eventos y artefactos específicos [4].

Herramientas y Tecnologías en la Gestión de Proyectos Ágiles: En la gestión de proyectos ágiles, las herramientas y tecnologías desempeñan un papel crucial a la hora de facilitar la colaboración, la comunicación y la coordinación entre los miembros del equipo. Según Schwaber y Sutherland (2017), las herramientas de gestión de proyectos ágiles específicas incluyen gestión de tareas, tableros, Kanban virtuales, herramientas de seguimiento de defectos y sistemas de integración continua. Con estas herramientas, los equipos ágiles pueden mantener una comprensión clara del trabajo en progreso, priorizar las tareas de manera efectiva y responder rápidamente a los cambios en los requisitos del proyecto. Además de las herramientas de gestión de proyectos ágiles, es importante considerar tecnologías que puedan mejorar la colaboración y la eficiencia del equipo. Por ejemplo, las plataformas de comunicación en tiempo real como Slack o Microsoft Teams permiten la comunicación instantánea entre los miembros del equipo, independientemente de su ubicación geográfica. Además, herramientas de videoconferencia como Zoom o Google Meet permiten reuniones virtuales efectivas, que son particularmente útiles para equipos geográficamente dispersos [5].

Diseño de interfaces centradas en el usuario: En la gestión ágil de proyectos, el diseño de interfaz de usuario orientado a la interfaz de usuario es una parte clave de la

usabilidad y la satisfacción del cliente. El diseño centrado en el usuario se basa en la comprensión de las necesidades, comportamientos y expectativas del usuario final para crear interfaces de usuario que sean intuitivas, eficientes y agradables de usar [6]. En el contexto de la gestión ágil de proyectos, esto significa incorporar prácticas de diseño iterativas donde los usuarios reciben comentarios continuos durante todo el proceso de desarrollo para refinar y mejorar la interfaz de usuario. Los métodos ágiles como Scrum fomentan una estrecha colaboración entre los equipos de desarrollo y los usuarios finales, lo que facilita la creación de interfaces de usuario que satisfagan las necesidades reales de los usuarios.

Tendencias Futuras en la Gestión de Proyectos Ágiles: En el panorama actual de la gestión de proyectos ágiles, es importante considerar las tendencias futuras que darán forma a la gestión de proyectos. Según Schwaber y Sutherland (2017), una tendencia emergente es la integración de la inteligencia artificial (IA) y el aprendizaje automático (Maching Learning) en prácticas ágiles. Estas técnicas tienen el potencial de mejorar la planificación de proyectos, identificar riesgos potenciales y optimizar la asignación de recursos mediante el análisis de grandes cantidades de datos históricos y en tiempo real. Además, se espera que la gestión de proyectos ágiles evolucione para adaptarse a la creciente demanda de proyectos más grandes y complejos, esto puede incluir enfoques híbridos que combinan prácticas ágiles con elementos de enfoques más tradicionales, como el modelo en cascada, para gestionar proyectos que requieren mayor estructura y control [7].

Aplicación de Conceptos en el Contexto del Proyecto: En el contexto de nuestro proyecto de desarrollo de una aplicación para la gestión de proyectos agiles, es importante hacer una conexión directa entre los principios ágiles, las herramientas y tecnologías mencionadas en el marco teórico, y el diseño y desarrollo de nuestra aplicación. La metodología ágil de Scrum con un enfoque iterativo e incremental sirve como modelo central para la estructuración de la aplicación, lo que permite la entrega continua de funcionalidades y una adaptación flexible a los requisitos cambiantes del proyecto. Además, las herramientas específicas de gestión de proyectos ágiles, como los tableros Kanban virtuales, se implementarán en la aplicación para mejorar la visualización y el seguimiento de las tareas del equipo. Este vínculo directo entre los conceptos teóricos y la práctica de desarrollo de aplicaciones garantiza que nuestra solución sea relevante y efectiva para las necesidades de los equipos.

3.2. Estado del Arte

En esta sección se analizarán estudios y desarrollos realizados con el objetivo de evaluar las ventajas y limitaciones de las herramientas que hoy en día permiten trabajar de manera flexible y organizada. Se abordarán desde la personalización de software realizada por Rönn para una empresa, que buscaba adaptar mejor las herramientas a su estilo de trabajo, hasta la comparativa de aplicaciones open-source ampliamente utilizadas. El propósito es identificar las similitudes y extraer conclusiones tanto de los autores como de la comparativa, en relación con los desafíos que enfrentan algunos grupos de desarrollo pequeños, no necesariamente empresas, al intentar adaptar metodologías ágiles a sus proyectos.

Herramienta para la gestión de proyectos personalizada

Es un hecho que actualmente existen numerosas aplicaciones para la gestión de metodologías ágiles, y muchas de ellas ofrecen una gran variedad de herramientas y funcionalidades que ayudan a un desarrollador a tener control, colaboración, mejora continua y flexibilidad en el proyecto que se está llevando a cabo. Sin embargo, esto a su vez puede llegar a ser un problema para algunas empresas de desarrollo o equipos pequeños, ya que la manera de trabajar no es la misma para todos. Un estudio realizado por Ellinor Rönn indica que, a pesar de haber muchas aplicaciones para la gestión de metodologías ágiles, no todas cumplen con los requerimientos de trabajo que las empresas necesitan. Esto provoca que algunas empresas trabajen con otras aplicaciones para compensar la falta de adaptación de algunas. Por ejemplo, se nos dice que hay aplicaciones que son demasiado simples o que tienen tantas funciones que se vuelven complejas de usar, o bien otras que obligan a seguir un estándar que solo permite un método específico de manera estricta, lo cual se debe evitar.

Una empresa que se ha encontrado con este problema es Codemill, donde actualmente hay cinco equipos de desarrollo que utilizan diferentes métodos. Los equipos utilizan principalmente Trello para realizar el seguimiento de sus proyectos, ya que se menciona que es la herramienta que mejor se adapta a sus necesidades; sin embargo, no están satisfechos con ninguna de las herramientas que han probado, ya que no se adaptan completamente. Ante esta situación, Ellinor Rönn, optó por desarrollar una aplicación personalizada que mejorara la adaptación a sus necesidades. El desarrollo de la aplicación implicó un enfoque iterativo, en el que se incorporaban retroalimentaciones constantes del equipo de Codemill para asegurar que la herramienta cumpliera con sus expectativas. El análisis del objetivo que se planteó Rönn concluye que, aunque existen herramientas comerciales que pueden ser adaptadas para gestionar proyectos ágiles, el desarrollo de una solución personalizada puede ofrecer beneficios significativos en términos de alineación con las necesidades específicas de una empresa. Sin embargo, esto también requiere una inversión considerable en tiempo y recursos. En resumen, el desarrollo de una solución personalizada, como la llevada a cabo por Codemill, demuestra que aunque las herramientas comerciales ofre-

cen flexibilidad, la personalización puede ser crucial para la alineación con los procesos específicos de la empresa, lo que subraya la importancia de un análisis cuidadoso de las necesidades y los recursos disponibles**.** Además, se menciona que futuras investigaciones deberían enfocarse en el análisis comparativo de costo-beneficio de personalizar herramientas existentes versus desarrollar nuevas desde cero, así como estudiar cómo las aplicaciones personalizadas pueden escalar a medida que la empresa crece y mantener la flexibilidad sin comprometer la usabilidad.[8]

Desarrollo ágil personalizado para el desarrollo de software integrado

En 2010, Tomi Juhola realizó un desarrollo en el que abordó la creciente popularidad del desarrollo ágil de software desde la publicación del Manifiesto Ágil en 2001. A pesar de este auge, Juhola señala que persiste una creencia generalizada de que los métodos ágiles no son adecuados para el desarrollo de software integrado, crítico o en tiempo real. Sin embargo, destaca que existen numerosos estudios que contradicen esta creencia, demostrando lo contrario.

Juhola decidió desarrollar una aplicación de desarrollo personalizada debido a la percepción de que los métodos ágiles tradicionales no proporcionan un control suficiente, carecen de una disciplina rigurosa y aplican prácticas de ingeniería menos estrictas. Su objetivo fue crear un proceso ágil, ligero, pero disciplinado, que pudiera aplicarse en proyectos de software integrado con equipos de entre 10 y 500 personas. Este proceso se basó en una combinación de características de Extreme Programming, Scrum y el Modelo Ágil.

Para ello, Juhola desarrolló un proceso ágil para una empresa de formación y consultoría. La empresa, que tenía un cliente importante en Finlandia, necesitaba implementar métodos ágiles, pero no estaba segura de cuál sería el más adecuado para su entorno ni de cómo podría adoptarlo. El proyecto comenzó con un estudio de los métodos ágiles más populares y la revisión de estudios de caso. El análisis consistió en evaluar diferentes aspectos de estos métodos y su aplicación en la organización objetiva. Juhola caracterizó cada método según sus estereotipos: XP era considerado muy radical, FDD estaba orientado a las características, Scrum era visto solo como un marco, y DSDM estaba enfocado en el software empresarial y financiero.

En el estudio, se menciona que se realizó una versión modificada de FDD, tomando en cuenta las retroalimentaciones necesarias para adaptarlo de la mejor manera posible. Aunque el proceso de desarrollo enfrentó dificultades debido a la falta de claridad inicial, Juhola subraya que la adopción ágil es un proceso complejo que requiere más que documentación; es esencial una orientación adecuada y la implementación práctica de las prácticas diarias. Además, destaca que crear una versión modificada, como en el caso de FDD, les enseñó lecciones valiosas:

- Ningún proceso es útil en el papel

- Ningún proceso puede realizarse sin involucrar a las personas
- Una entrega más rápida aumenta las posibilidades de éxito

Finalmente, Juhola concluye que, aunque lograron desarrollar un proceso sólido y estaban satisfechos con los resultados, se dieron cuenta de que, una vez finalizado el desarrollo, el proceso no fue utilizado. Esto les llevó a reflexionar sobre la importancia de que un proceso de desarrollo de software debe tener valor práctico y no ser puramente teórico. [9]

Limitada adopción de metodologías ágiles en otros sectores

En 2014, Aljaž Stare, Investigador de la Universidad de Liubliana, Eslovenia, llevó a cabo un estudio significativo que fue publicado en la prestigiosa revista Procedia- Social and Behavioral Sciences. Este estudio aborda un tema crítico en la gestión de proyectos: La limitada adopción de metodologías ágiles en sectores fuera del desarrollo de software, como la ingeniería, investigación y desarrollo. El estudio incluyó un análisis detallado de varias empresas seleccionadas para el caso de estudio, con el objetivo de evaluar no solo la adopción de técnicas ágiles, sino también el impacto concreto de cada técnica en el éxito de los proyectos. Stare utilizó un enfoque mixto que combinó cuestionarios y análisis de regresión para identificar la presencia y efectividad de estas técnicas en el desarrollo de productos. El estudio concluye que los resultados revelaron que, si bien las empresas del estudio aplicaban técnicas ágiles en sus proyectos de desarrollo de productos, solo algunas técnicas específicas, como los sprints y los equipos autoorganizados, demostraron ser especialmente efectivas en términos de contribuir al éxito del proyecto. Estos hallazgos subrayan la importancia de seleccionar y adaptar cuidadosamente las técnicas ágiles según el contexto del proyecto, y sugieren que la metodología ágil tiene un potencial significativo para ser aplicada en una gama más amplia de industrias. [10].

Comparación de aplicaciones open-source

Las aplicaciones open-source han ganado popularidad debido a su flexibilidad, personalización y la comunidad que las respalda. Sin embargo, estas aplicaciones presentan una variedad de características y limitaciones que pueden influir en su elección, desentendiendo de las necesidades específicas de un equipo o empresa. Esto está muy arraigado a lo que mencionaba Rönn en su investigación, donde se nos menciona que las empresas no se adaptan completamente a una herramienta y hace el uso de otras para complementar y así realizar un balance. [8]

Tabla 1: Cuadro comparativo de aplicaciones de gestión de metodologías ágiles open source.

Aplicaciones	Ventajas	Desventajas
Taiga	▪ Interfaz simple ▪ Sin publicidad ▪ Elaboración de informes ▪ Tablero Kanban ▪ Gratuito ▪ Colaboración en tiempo real ▪ Seguimiento de las tareas	▪ Sin seguimiento del tiempo ▪ Sin controlador de versiones ▪ Sin integración de chat colaborativo ▪ Sin estadísticas ni informes de análisis ▪ Los usuarios solo pueden tener un rol dentro del proyecto
OpenProject	▪ Flexibilidad ▪ Tablero Kanban ▪ Estructura jerárquica ▪ Compatibilidad con Markdown ▪ Sin publicidad ▪ Sincronización en la nube ▪ Tablero Scrum ▪ Gratis	▪ Herramientas complejas de usar ▪ Interfaz poco amigable ▪ No es completamente gratuito ▪ No contiene todas las funciones liberadas ▪ Límite de proyectos ▪ Dificultad en el uso móvil ▪ Curva de aprendizaje

Aplicaciones	Ventajas	Desventajas
Redmine	▪ Fácil de usar ▪ Sistema de tickets ▪ Diagrama de Gantt ▪ Gratis ▪ Control de versiones ▪ Tablero Kanban ▪ Colaboración en tiempo real ▪ Centrado en la privacidad	▪ Interfaz anticuada ▪ No tiene integración con chat colaborativo ▪ Versión gratuita limitada en cuanto a soporte

En la Tabla 1, se presenta un cuadro comparativo que analiza las principales ventajas y desventajas de tres aplicaciones open-source ampliamente utilizadas, Taiga, OpenProject y Redmine. Este análisis permite observar como cada herramienta aborda aspectos clave como la interfaz de usuario, la capacidad de integración y funcionalidades específicas que ofrecen cada una.

A pesar de las ventajas que cada aplicación puede ofrecer, es evidente que ninguna de ellas proporciona una solución completamente holística. Mientras algunas aplicaciones se destacan por su simplicidad y facilidad de uso, otras ofrecen un conjunto más amplio de funcionalidades a expensas de una curva de aprendizaje más pronunciada o de limitaciones en su uso. Esta comparativa subraya la importancia de evaluar detenidamente las necesidades del equipo antes de seleccionar una herramienta para la gestión de metodologías ágiles.

3.3. Antecedentes

Los autores: Maria Ilaria Lunesu, Roberto Tonelli, Lodovica Marchesi y Michele Marchesi de la Universidad de Cagliari, Italia. Tuvieron como objetivo principal de estudio presentar un enfoque para el manejo de riesgos en el desarrollo de software con metodologías ágiles, utilizando la simulación del proceso de software. Los estudios fueron realizados en simulaciones de tres proyectos de código abierto utilizando la herramienta JIRA con el fin de analizar el impacto de ciertos factores de riesgo en el proceso de desarrollo de software. Los resultados mostraron que la herramienta de simulación podía aproximar de manera efectiva el progreso de un proyecto real y también podía usarse para evaluar cuantitativamente el impacto. Se encontró un error de estimación para desarrollar una función podría aumentar un 30 %, mientras que una asignación aleatoria de problemas podría aumentar la duración del proyecto un 60 %. El estudio concluye que utilizar simulación del proceso de software puede ser una herramienta útil para la evaluación del riesgo en el desarrollo de software con metodologías ágiles. Además, se destaca la modelización del dato del proyecto y del proceso, así como la necesidad de validar las herramientas de simulación de datos reales [11].

Danijela Ciric y otros autores de la Universidad de Novi Sad en Serbia. Realizaron como objetivo estudio proporcionar una visión coherente de las estrategias para introducir la gestión de proyectos ágiles en entornos de gestión de proyectos tradicionales y, a través de la investigación empírica, mostrar cuáles son las razones para la introducción de la gestión de proyectos ágiles y los desafíos de su aplicación en proyectos de desarrollo de software. El estudio tuvo como conclusión que la implementación de la gestión de proyectos ágiles en entornos de gestión de proyectos tradicionales presenta desafíos y oportunidades. Además, se menciona el hecho de la importancia de adaptar cuidadosamente el enfoque ágil a la naturaleza específica de cada organización y seleccionar las prácticas ágiles que sean adaptables a los requerimientos. [12].

Un estudio realizado por Jana Pócsová, Dagmar Bednárová, Gabriela Bogdanovská y Andrea Mojžišová en la Universidad Técnica de Košice tenía como objetivo explorar cómo la implementación de metodologías ágiles en el ámbito educativo puede ser una herramienta eficiente para mejorar tanto las habilidades técnicas como las habilidades blandas de los estudiantes. El estudio se llevó a cabo en el curso de Matemáticas 1, y se centró en la generación Z, un grupo de jóvenes que han crecido en una era digital, pero que a menudo carecen de habilidades analíticas y blandas. Para el estudio, se utilizó el marco de trabajo SCRUM, una "metodología ágil"popular en el desarrollo de software. Se buscó observar cómo la implementación de SCRUM, con sus roles específicos, artefactos y ceremonias, podía rediseñar la dinámica de la clase. El estudio concluyó que el uso de SCRUM en el grupo experimental de estudiantes durante el año académico 2019/2020 no solo aumentó la eficiencia del proceso educativo, sino que también mejoró las habilidades de los estudiantes para la práctica profesional. [13]

Proyecto	Objetivo	Fecha de Publicación	Autores	Características
Assessing the Risk of Software Development in Agile Methodologies Using Simulation	El objetivo principal del estudio es presentar un enfoque para el manejo de riesgos en el desarrollo de software con metodologías ágiles, utilizando la simulación del proceso de software.	27 de septiembre del 2021	Maria Ilaria Lunesu, Roberto Tonelli, Lodovica Marchesi y Michele Marchesi	Los estudios fueron realizados en simulaciones de tres proyectos de código abierto utilizando la herramienta JIRA con el fin de analizar el impacto de ciertos factores de riesgo en el proceso de desarrollo de software.
Agile vs. Traditional Approach in Project Management: Strategies, Challenges and Reasons to Introduce Agile	El objetivo del estudio fue proporcionar una visión coherente de las estrategias para introducir la gestión de proyectos ágiles en entornos de gestión de proyectos tradicionales	2019	Danijela Ciric y otros autores de la Universidad de Novi Sad en Serbia	Se concluye que la implementación de la gestión de proyectos ágiles en entornos de gestión de proyectos tradicionales presenta desafíos y oportunidades. Se destaca el hecho de la importancia de adaptar cuidadosamente el enfoque ágil a la naturaleza específica de cada organización y seleccionar las prácticas ágiles a seleccionar.
Implementation of Agile Methodologies in an Engineering Course	Explorar cómo la implementación de metodologías ágiles en el ámbito educativo puede mejorar tanto las habilidades técnicas como las habilidades blandas de los estudiantes.	17 de noviembre del 2020	Jana Pócsová, Dagmar Bednárová, Gabriela Bogdanovská y Andrea Mojžišová	Se observó que el uso de SCRUM aumentó la eficiencia del proceso educativo y mejoró las habilidades de los estudiantes para la práctica profesional.

Tabla 2: Comparación de antecedentes

3.4. Marco Conceptual

Los siguientes conceptos son fundamentales para comprender la gestión de proyectos ágiles:

1. **Metodologías Ágiles:**
   Conjunto de prácticas que promueven el desarrollo iterativo, colaboración constante y adaptación al cambio. Se centran en entregar valor de forma continua al cliente.

2. **Manifiesto Ágil:**
   Documento que se centra en los 4 valores y 12 principios del desarrollo de software con metodologías ágiles.

3. **Principios Ágiles:**
   Conjunto de directrices fundamentales que guían la metodología ágil de desarrollo de software.

4. **Iteración:**
   Ciclo de trabajo que implica el desarrollo de una porción funcional del software o el producto, seguido de una revisión y evaluación de los resultados.

5. **Entrega Incremental:**
   Implica la entrega de un producto en pequeñas partes funcionales y completas, en lugar de esperar hasta el final del proyecto para entregar un producto completo.

6. **Retroalimentación Continua:**
   Proceso por el cual los comentarios y evaluación sobre el trabajo realizado se obtienen y aplican de manera constante durante todo el ciclo de desarrollo de un proyecto.

7. **Colaboración:**
   Trabajo coordinado entre los miembros del equipo con el fin de alcanzar el mismo objetivo.

8. **Mejora Continua:**
   Proceso fundamental que busca elevar la calidad, eficiencia y efectividad a través de cambios incrementales.

9. **Adaptabilidad:**
   Es la capacidad de ajustarse y responder a los cambios que surjan mediante el proceso.

10. **Autoorganización:**
    Es la capacidad del equipo para estructurarse y gestionar su trabajo de manera autónoma y eficaz.

11. **Valor para el Cliente:**
    Es un conjunto de beneficios y ventajas que un cliente percibe al adquirir un producto o servicio.

12. **Proyecto Ágil:**
    Un proyecto ágil se caracteriza por el uso de ciclos iterativos, colaboración continua y entregas frecuentes de valor. Su flexibilidad permite adaptarse a cambios en los requisitos o prioridades del cliente.

13. **Scrum:**
    Marco de trabajo ágil que organiza el desarrollo en sprints, con roles, eventos y artefactos definidos para facilitar la entrega continua de valor.

14. **Scrum Master:**
    Se asegura de que el equipo siga los principios y prácticas de Scrum, eliminando obstáculos y facilitando un ambiente productivo.

15. **Product Owner:**
    Representa a los interesados y clientes, define y prioriza el backlog del producto.

16. **Equipo de Desarrollo:**
    Responsable de convertir los elementos del backlog del producto en incrementos de trabajo potencialmente entregables.

17. **Stakeholder:**
    Personas, grupos o instituciones que tienen un interés directo o indirecto en el proyecto y pueden influir en su éxito o verse afectados por sus resultados.

18. **Desarrollador:**
    Profesional que diseña, crea, prueba y mantiene programas informáticos, asegurando su funcionamiento eficiente y su adaptación a las necesidades del usuario.

19. **Tester:**
    Encargado de evaluar y verificar la calidad de software mediante la ejecución de pruebas para identificar defectos, errores o fallos.

20. **Sprint:**
    Período de tiempo fijo durante el cual se desarrolla un incremento de trabajo potencialmente entregable.

21. **Sprint Planning:**
    Reunión al inicio del sprint para definir el trabajo que se realizará y cómo se logrará.

22. **Daily Scrum:**
    Reunión diaria breve para sincronizar al equipo y resolver obstáculos.

23. **Sprint Review:**
    Reunión al final del sprint para demostrar el trabajo completado y recibir retroalimentación.

24. **Sprint Retrospective:**
    Reunión al final del sprint para reflexionar sobre el desempeño y buscar mejoras.

25. **Product Backlog:**
    Lista priorizada de funcionalidades y requisitos necesarios para el producto.

26. **Sprint Backlog:**
    Subconjunto del Product Backlog que el equipo se compromete a desarrollar en un sprint.

27. **Resumen del Backlog:**
    Visión general que presenta el estado actual del backlog, con elementos organizados por prioridad y estado.

28. **Incremento:**
    Resultado del trabajo completado durante un sprint que se suma al valor entregado anteriormente.

29. **Sprint Goal:**
    Objetivo principal que el equipo se compromete a alcanzar durante un sprint.

30. **Definition of Done:**
    Conjunto de criterios que define cuándo un elemento del backlog o incremento está completado.

31. **Burndown Chart:**
    Herramienta visual que muestra el trabajo pendiente frente al tiempo restante durante un sprint.

32. **Kanban:**
    Método visual de gestión del flujo de trabajo que utiliza tarjetas y columnas para representar el estado de las tareas. Promueve la eficiencia limitando el trabajo en curso y maximizando el flujo continuo.

33. **Historia de Usuario:**
    Breve descripción de una funcionalidad o requerimiento desde la perspectiva del usuario final. Suelen redactarse así: *Como [rol] quiero [funcionalidad] para [beneficio].*

34. **Épica:**
    Gran historia de usuario que representa un conjunto de funcionalidades amplias. Se divide en historias más pequeñas y manejables.

35. **Tarea:**
    Unidad de trabajo concreta derivada de una historia de usuario, que describe una acción específica.

36. **Puntos de Historia:**
    Unidad de medida relativa usada para estimar el esfuerzo requerido para completar una historia de usuario.

4. Alcance del Proyecto

4.1. Declaración del alcance

La plataforma ofrece funcionalidades sin costo alguno, donde se da prioridad a proporcionar un conjunto de herramientas, sin restricciones que pueden aumentar la eficiencia y productividad de un equipo de manera completamente gratuita.

Las funcionalidades serán las siguientes:

- Funcionalidad de tableros Kanban
- Integración con correo electrónico para notificar actualizaciones importantes
- Generación de informes basados en el proyecto, los informes serán los siguientes:
  - Burndown Charts
  - Resumen del backlog:
    - Épicas
    - Historias de usuario
    - Tareas
- Implementar roles internos relacionados con el proyecto:
  - Product Owner
  - Scrum Master
  - StakeHolder
  - Team Member
    - Developer
    - Tester
- Implementar un modulo de tareas que permita priorizar, organizar y asignar historias de usuario.
- Desarrollar un diseño responsive, adaptable a diferentes dispositivos.
- Permitir la creación de historias de usuario y épicas con facilidad.
- Implementar una vista que permita al usuario ver sus asignaciones del backlog.
- Agregar la opción para crear proyectos, generar sprints asociados y asignar miembros al proyecto.

Restricciones:

- Aplicación móvil

- Otras metodologías ágiles
- Multi lenguaje
- Open source

4.2. Objetivos

Árbol de objetivos

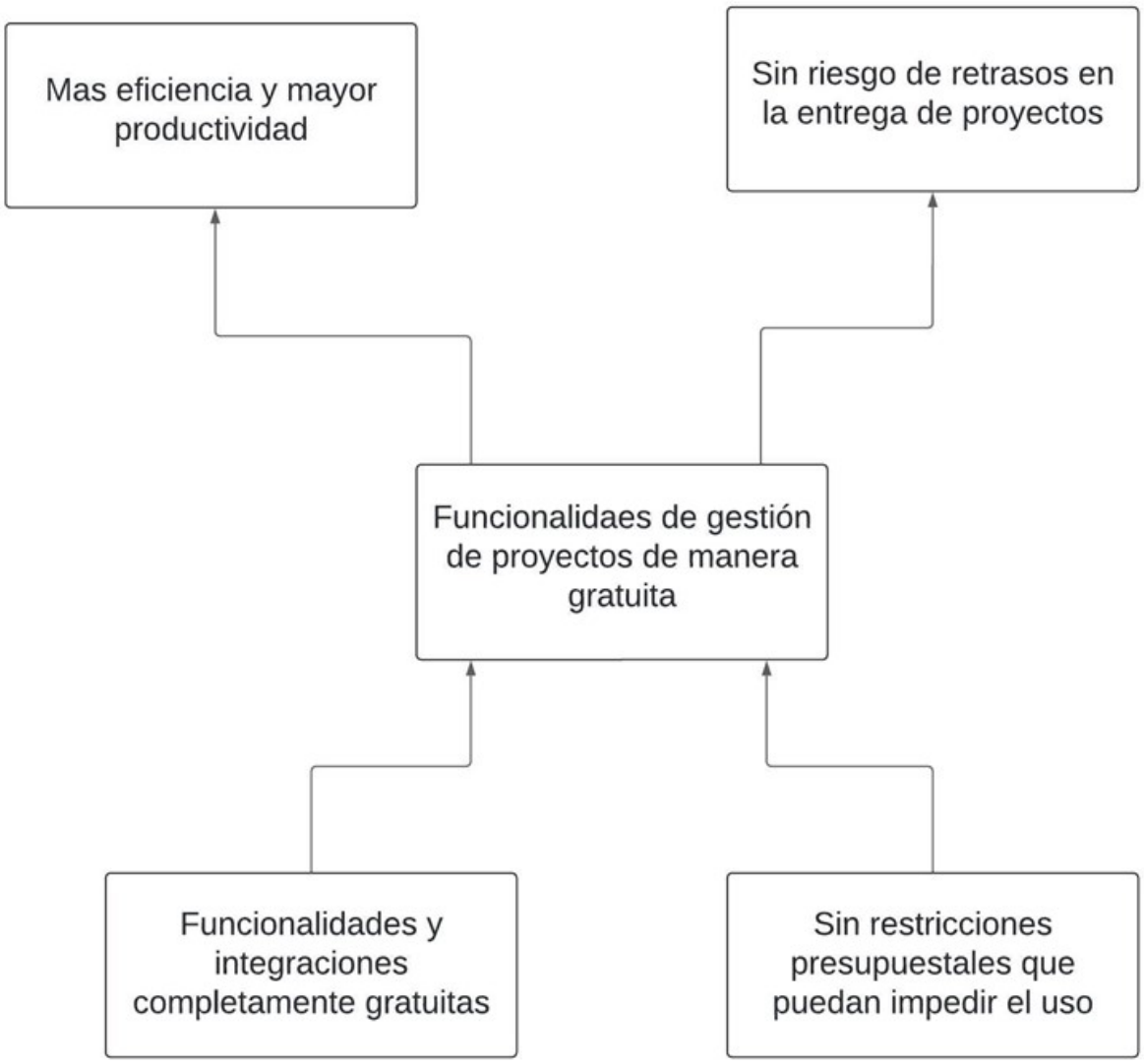

Figura 2: Árbol de objetivos

4.2.1. Objetivo general

Desarrollar un sistema de gestión de proyectos de software en un entorno web que incluya funcionalidades como planificación de tareas y seguimiento del progreso, además de proporcionar una herramienta para la colaboración y la productividad de los equipos de desarrollo.

4.2.2. Objetivos específicos

- Documentar los requisitos funcionales y no funcionales del sistema de gestión de proyectos de software utilizando encuestas como medio de recopilación de información.
- Diseñar interfaces de usuario intuitivas y eficientes, aplicando principios de diseño centrado en el usuario (UCD) y desarrollando prototipos interactivos que permitan la validación con los usuarios finales.
- Diseñar un modelo entidad-relación que describa detalladamente la estructura de la base de datos y su interacción con el sistema.
- Implementar una aplicación web que cumpla con los requisitos y las historias de usuario definidas, utilizando tecnologías de desarrollo web, asegurando la integración y cohesión de todos los módulos del sistema.
- Realizar pruebas funcionales utilizando frameworks de pruebas para asegurar que cada componente de la aplicación funcione correctamente y que el sistema cumpla con los requisitos especificados.

4.3. Entregables

Objetivos específicos	**Entregables**
Requerimientos e historias de usuario	▪ Documento detallado de los requerimientos. ▪ Historias de usuario bien definidas con criterios de aceptación.
Diseños UI de la aplicación	▪ Prototipo de la interfaz de usuario. ▪ Mockups y wireframes detallados.
Código fuente de la aplicación	▪ Repositorio de código fuente con documentación. ▪ Instrucciones para la instalación y despliegue de la aplicación.
Reporte de las pruebas	▪ Resultados de las pruebas unitarias, de integración y de usuario. ▪ Documentación de los casos de pruebas y sus resultados.

Tabla 3: Entregables del proyecto

5. Levantamiento de Requerimientos

Esta sección presenta los requerimientos del sistema organizados conforme a la norma IEEE 830, dividiéndolos en funcionales y no funcionales. La información fue obtenida a través de una encuesta dirigida a potenciales usuarios del sistema, lo que permitió identificar y priorizar las funcionalidades más valoradas.

5.1. Metodología de Recolección de Información

Para obtener un conjunto de requerimientos claros y alineados con las necesidades del usuario, se emplearon diversas técnicas de recopilación de información, siendo la principal una encuesta dirigida a potenciales usuarios del sistema. La encuesta fue diseñada con el fin de abordar diversas características del sistema, desde su facilidad de uso hasta su capacidad de integración con herramientas externas.

Cada pregunta se formuló utilizando una escala del 1 al 5, donde 1 representa una baja importancia y 5 una alta importancia. Este enfoque permitió cuantificar las respuestas de manera efectiva, facilitando el análisis y la comparación de prioridades. Además, se incluyó una pregunta abierta para profundizar en las expectativas de los usuarios.

5.2. Encuesta Preliminar para Potenciales Usuarios

A continuación se presentan los resultados obtenidos a partir de la encuesta, donde se calculó un promedio para cada pregunta con el fin de determinar la importancia y prioridad de las características solicitadas.

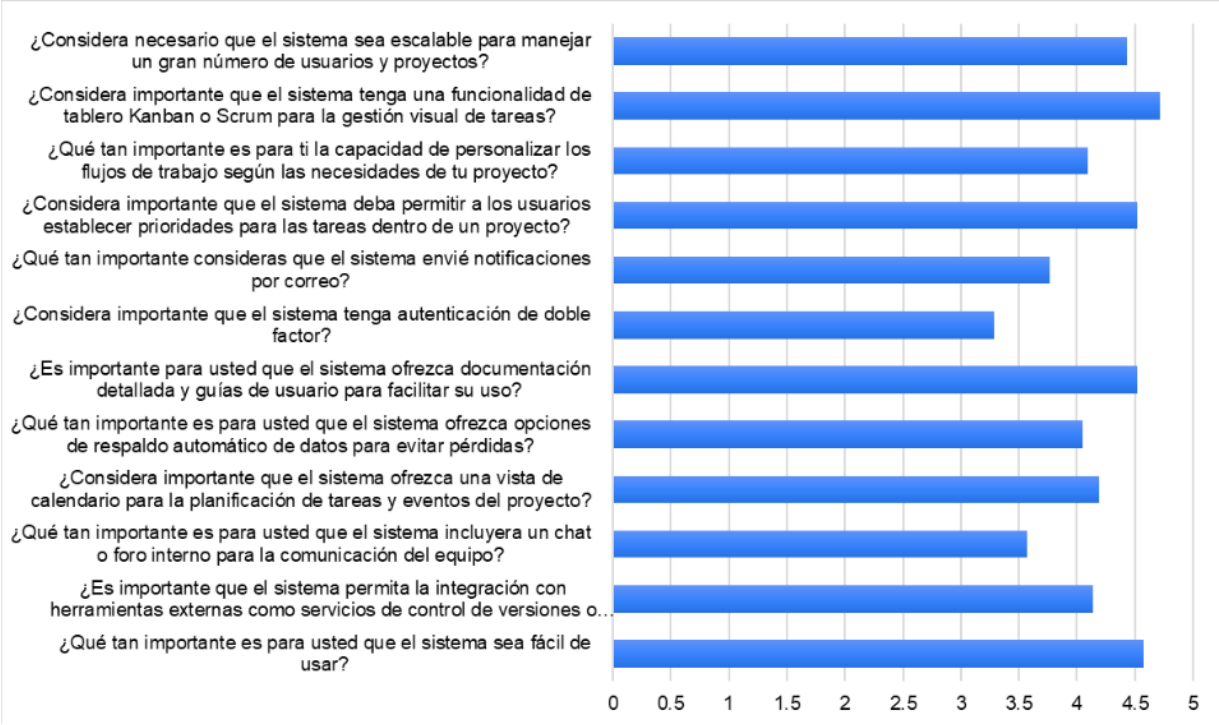

Figura 3: Resultados de la encuesta preliminar

Figura 4: Gráfica de priorización de funcionalidades

5.3. Conclusión General

Los resultados revelan que los usuarios priorizan tres aspectos fundamentales: la facilidad de uso, la gestión visual de tareas mediante tableros Kanban o Scrum y la comunicación interna efectiva mediante chat o foros.

Aunque las integraciones con herramientas externas y la escalabilidad son valoradas, no representan barreras críticas para la mayoría. Esta información orienta el desarrollo hacia una estrategia centrada en la experiencia del usuario.

———-

5.4. Requisitos Comunes de Interfaces

Interfaces de usuario

La interfaz gráfica del sistema es clara, intuitiva y accesible. Cuenta con una distribución lógica de los elementos, uso de iconografía clara, botones accesibles y navegación sencilla. El sistema se adapta automáticamente a distintos tamaños de pantalla (responsive), garantizando una experiencia consistente en computadores de escritorio, tablets y dispositivos móviles.

Interfaces de hardware

No se requieren requisitos especiales de hardware. El sistema puede ejecutarse en cualquier equipo que disponga de un navegador web moderno y conexión a internet.

Interfaces de software

El sistema se conecta con servicios de backend mediante APIs RESTful. Utiliza bases de datos relacionales para el almacenamiento (PostgreSQL), y se integra con servicios externos como proveedores de correo electrónico para notificaciones.

Interfaces de comunicación

Toda la comunicación entre el cliente y el servidor se realiza mediante HTTPS para garantizar la seguridad. El sistema no necesita comunicarse con sistemas de terceros, lo cual simplifica su arquitectura y reduce riesgos de seguridad.

5.5. Requisitos Funcionales

Tabla 4: RF-01: Gestión de proyectos

Número de requerimiento	RF-01
Nombre del requerimiento	Gestión de proyectos
Tipo	Requerimiento funcional
Fuente	Encuesta a usuarios
Prioridad	Alta

Tabla 5: RF-02: Tableros Kanban/Scrum

Número de requerimiento	RF-02
Nombre del requerimiento	Visualización y gestión de tareas con tableros
Tipo	Requerimiento funcional
Fuente	Encuesta a usuarios
Prioridad	Alta

Tabla 6: RF-03: Notificaciones por correo

Número de requerimiento	RF-03
Nombre del requerimiento	Notificación automática por correo electrónico
Tipo	Requerimiento funcional
Fuente	Encuesta a usuarios
Prioridad	Media

Tabla 7: RF-04: Informes de estado del proyecto

Número de requerimiento	RF-04
Nombre del requerimiento	Generación de reportes (burndown, backlog)
Tipo	Requisito
Fuente	Encuesta a usuarios
Prioridad	Media

Tabla 8: RF-05: Gestión de roles

Número de requerimiento	RF-05
Nombre del requerimiento	Asignación de roles con permisos específicos
Tipo	Requisito
Fuente	Encuesta a usuarios
Prioridad	Alta

Tabla 9: RF-06: Prioridad y asignación de tareas

Número de requerimiento	RF-06
Nombre del requerimiento	Priorización y distribución de tareas
Tipo	Requisito
Fuente	Encuesta a usuarios
Prioridad	Alta

Tabla 10: RF-07: Interfaz responsive

Número de requerimiento	RF-07
Nombre del requerimiento	Adaptabilidad a dispositivos móviles
Tipo	Requisito
Fuente	Encuesta a usuarios
Prioridad	Media

Tabla 11: RF-08: Chat en tiempo real

Número de requerimiento	RF-08
Nombre del requerimiento	Módulo de comunicación interna en tiempo real
Tipo	Requisito
Fuente	Encuesta a usuarios
Prioridad	Alta

Tabla 12: RF-09: Estimación colaborativa

Número de requerimiento	RF-09
Nombre del requerimiento	Planning Poker y estimación participativa
Tipo	Requisito
Fuente	Buenas prácticas ágiles
Prioridad	Media

Tabla 13: RF-10: Gestión de cuentas

Número de requerimiento	RF-10
Nombre del requerimiento	Registro y administración de usuarios
Tipo	Requisito
Fuente	Necesidad funcional básica
Prioridad	Alta

Tabla 14: RF-11: Inicio y cierre de sesión

Número de requerimiento	RF-11
Nombre del requerimiento	Acceso seguro al sistema
Tipo	Requisito
Fuente	Buenas prácticas de seguridad
Prioridad	Alta

Tabla 15: RF-12: Búsqueda avanzada

Número de requerimiento	RF-12
Nombre del requerimiento	Localización de elementos mediante filtros
Tipo	Requisito
Fuente	Encuesta a usuarios
Prioridad	Alta

Tabla 16: RF-13: Comentarios en incidencias

Número de requerimiento	RF-13
Nombre del requerimiento	Espacio de comunicación contextual
Tipo	Requisito
Fuente	Encuesta a usuarios
Prioridad	Media

5.5.1. Requisitos Funcionales - Detallados

RF-01: Gestión de proyectos
El sistema debe permitir a los usuarios crear y gestionar múltiples proyectos. Para organizar el trabajo, se deben ofrecer herramientas como: la creación de tareas, la asignación de roles, la administración del backlog y la invitación de miembros a los equipos.

RF-02: Tableros Kanban/Scrum
Los usuarios podrán visualizar y organizar tareas utilizando tableros personalizados con metodologías ágiles (Kanban o Scrum). Los estados de las tareas serán: Por hacer (To-do), En progreso (In progress), Finalizado (Done), En revisión (Review, opcional) y Cerrado (Closed, opcional).

RF-03: Notificaciones por correo
El sistema enviará notificaciones automáticas por correo electrónico al invitar usuarios a proyectos, asignar tareas o realizar cambios relevantes.

RF-04: Informes de estado del proyecto
Se generarán automáticamente informes visuales como: gráfico de tareas completadas y

pendientes por día (task completion), burndown chart, y gráfico de progreso por épica. Además, cada usuario podrá consultar estadísticas individuales como número total de tareas, tareas finalizadas y tareas pendientes.

RF-05: Gestión de roles
El sistema permitirá asignar distintos roles a los usuarios (Scrum Master, Product Owner, Stakeholder, Team Member), controlando el acceso y las acciones permitidas según el perfil.

RF-06: Prioridad y asignación de tareas
Las tareas podrán clasificarse por prioridad (alta, media, baja) y asignarse a uno o varios miembros del equipo.

RF-07: Interfaz responsive
La aplicación será completamente adaptable a diferentes dispositivos, asegurando funcionalidad y legibilidad en computadoras, tablets y móviles.

RF-08: Chat en tiempo real
El sistema integrará un módulo de mensajería interna para facilitar la comunicación entre miembros del equipo en tiempo real.

RF-09: Estimación colaborativa
El sistema implementará herramientas de estimación como Planning Poker para asignar esfuerzo de tareas de forma consensuada entre los miembros del equipo.

RF-10: Gestión de cuentas
Los usuarios podrán registrarse, iniciar sesión, cerrar sesión y editar su información personal. El proceso de registro será sencillo y seguro.

RF-11: Inicio y cierre de sesión
Se implementará autenticación segura para controlar el acceso de los usuarios y proteger la confidencialidad de sus datos.

RF-12: Búsqueda avanzada
Los usuarios podrán buscar elementos del sistema mediante filtros por nombre, categoría, estado, prioridad u otros criterios relevantes.

RF-13: Comentarios en incidencias
Cada incidencia o tarea tendrá un espacio dedicado para comentarios, permitiendo registrar observaciones, aclaraciones y discusiones asociadas.

5.6. Requisitos No Funcionales

Tabla 17: RNF-01: Accesibilidad

Número de requerimiento	RNF-01
Nombre del requerimiento	Accesibilidad desde navegadores modernos
Tipo	Requerimiento no funcional
Fuente	Buenas prácticas técnicas
Prioridad	Alta

Tabla 18: RNF-02: Usabilidad e interfaz intuitiva

Número de requerimiento	RNF-02
Nombre del requerimiento	Interfaz centrada en el usuario
Tipo	Requerimiento no funcional
Fuente	Principios de diseño UX y encuesta a usuarios
Prioridad	Alta

Tabla 19: RNF-03: Seguridad de la información

Número de requerimiento	RNF-03
Nombre del requerimiento	Seguridad y control de acceso por roles
Tipo	Requerimiento no funcional
Fuente	Requisitos legales y de privacidad
Prioridad	Alta

Tabla 20: RNF-04: Escalabilidad del sistema

Número de requerimiento	RNF-04
Nombre del requerimiento	Soporte a múltiples usuarios y proyectos simultáneos
Tipo	Requerimiento no funcional
Fuente	Requerimientos de arquitectura
Prioridad	Media

Tabla 21: RNF-05: Rendimiento eficiente

Número de requerimiento	RNF-05
Nombre del requerimiento	Fluidez en operaciones con gran volumen de datos
Tipo	Requerimiento no funcional
Fuente	Buenas prácticas de desarrollo
Prioridad	Alta

Tabla 22: RNF-06: Plan de pruebas completo

Número de requerimiento	RNF-06
Nombre del requerimiento	Testing unitario, integración y rendimiento
Tipo	Requerimiento no funcional
Fuente	Buenas prácticas de aseguramiento de calidad (QA)
Prioridad	Alta

Tabla 23: RNF-07: Mantenibilidad del código

Número de requerimiento	RNF-07
Nombre del requerimiento	Código basado en buenas prácticas de programación
Tipo	Requerimiento no funcional
Fuente	Principios de ingeniería de software
Prioridad	Media

5.7. Requisitos No Funcionales - Detallados

RNF-01: Accesibilidad
La plataforma debe ser accesible desde cualquier navegador web moderno, asegurando la disponibilidad sin requerimientos técnicos específicos.

RNF-02: Usabilidad
La interfaz será fácil de usar, basada en principios UX que reduzcan la curva de aprendizaje.

RNF-03: Seguridad
Se protegerán los datos mediante cifrado, control de acceso por roles y buenas prácticas de seguridad.

RNF-04: Escalabilidad
La arquitectura permitirá un aumento en el número de usuarios y proyectos sin afectar el rendimiento.

RNF-05: Rendimiento
El sistema ofrecerá tiempos de respuesta adecuados incluso al manejar grandes volúmenes de datos y múltiples usuarios activos.

RNF-06: Pruebas de calidad
Se ejecutarán pruebas unitarias, de integración, rendimiento y usabilidad para garantizar la calidad del sistema.

RNF-07: Mantenibilidad

El código fuente seguirá estándares de programación, será modular y estará documentado para facilitar futuras mejoras.

6. Prototipo y Diseño de Interfaces de Usuario

El diseño de la interfaz de usuario es fundamental para garantizar una experiencia intuitiva, eficiente y coherente. Esta sección presenta el proceso de diseño seguido, los wireframes iniciales y los mockups finales que representan visualmente la estructura y funcionalidad de la aplicación.

6.1. Proceso de Diseño

El diseño se llevó a cabo mediante un enfoque iterativo, centrado en la usabilidad y la experiencia del usuario. Se utilizaron herramientas como Figma y Sora para la elaboración de prototipos. Durante este proceso, se evaluaron diferentes opciones de distribución de contenido, accesibilidad y estilo visual.

6.2. Wireframes

Los wireframes muestran la estructura inicial de la aplicación, enfocándose en la organización de componentes y navegación sin elementos visuales detallados. Sirven como base para validar la lógica y flujo de la interfaz. A continuación se muestran los wireframes de la aplicación:

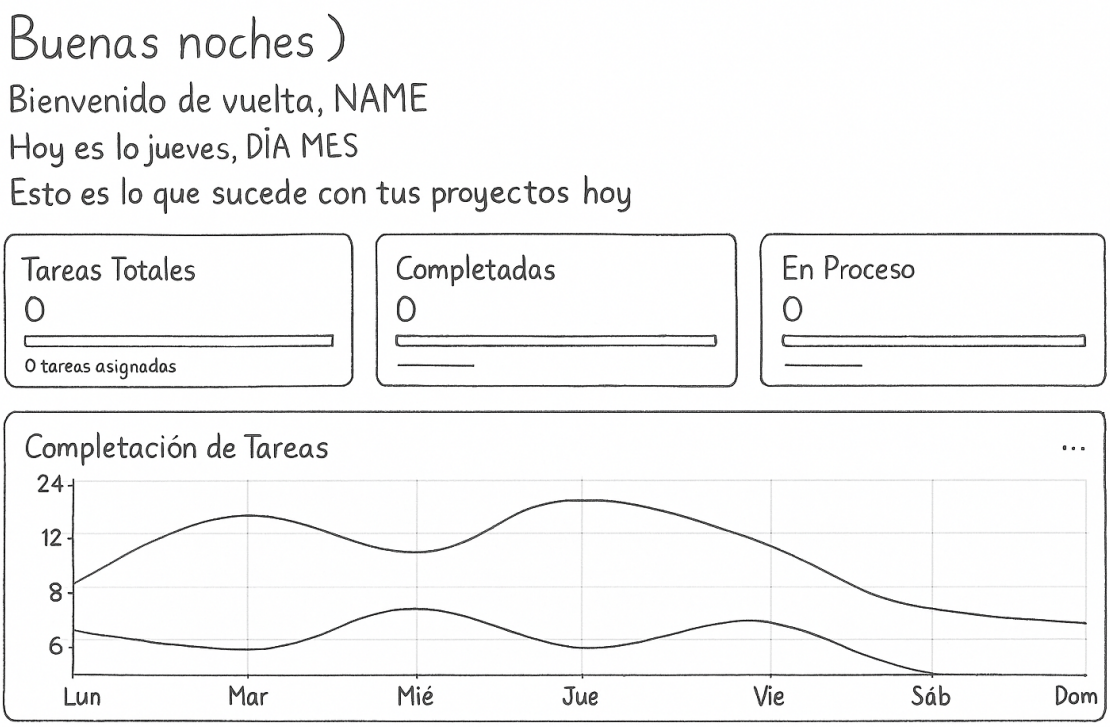

Figura 5: Wireframe del panel principal (*Dashboard*) con accesos rápidos a estadísticas, tareas y proyectos activos.

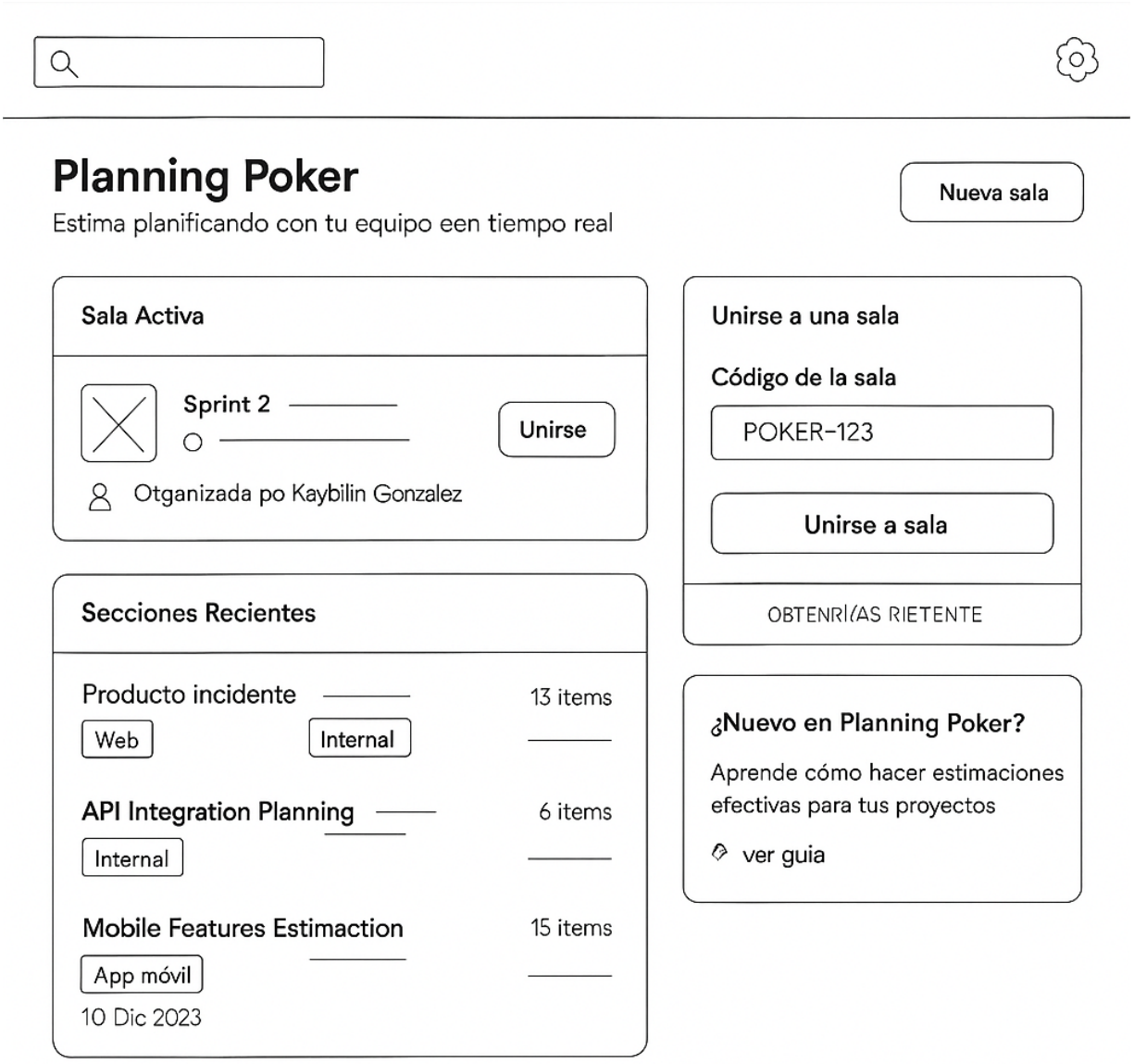

Figura 6: Wireframe de la pantalla inicial del módulo *Planning Poker*, donde los usuarios pueden crear o unirse a una sala.

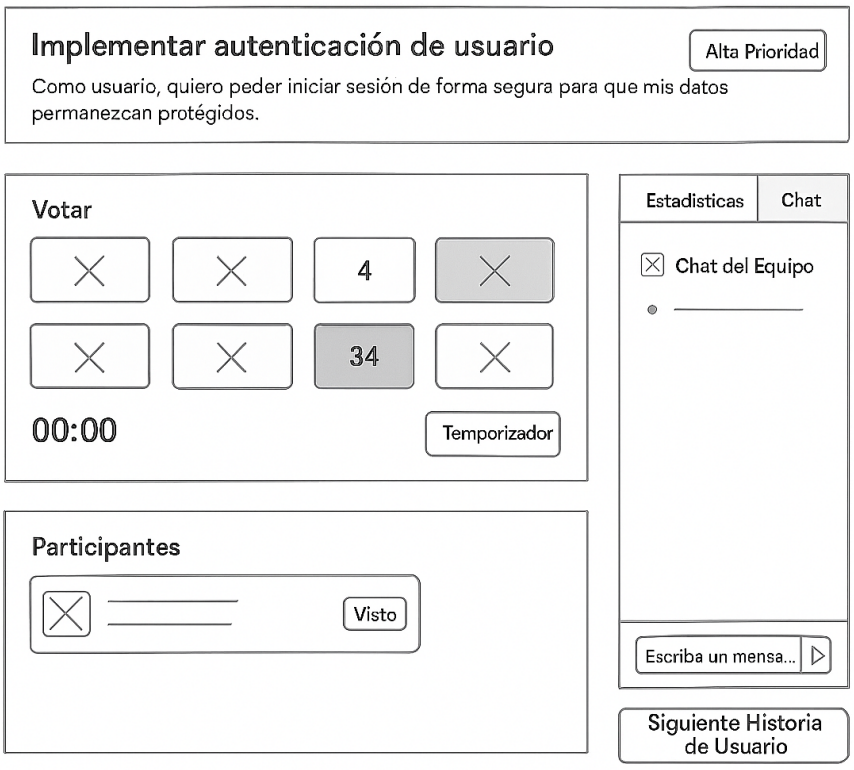

Figura 7: Wireframe de una sala activa de *Planning Poker*, mostrando a los participantes y las cartas seleccionadas.

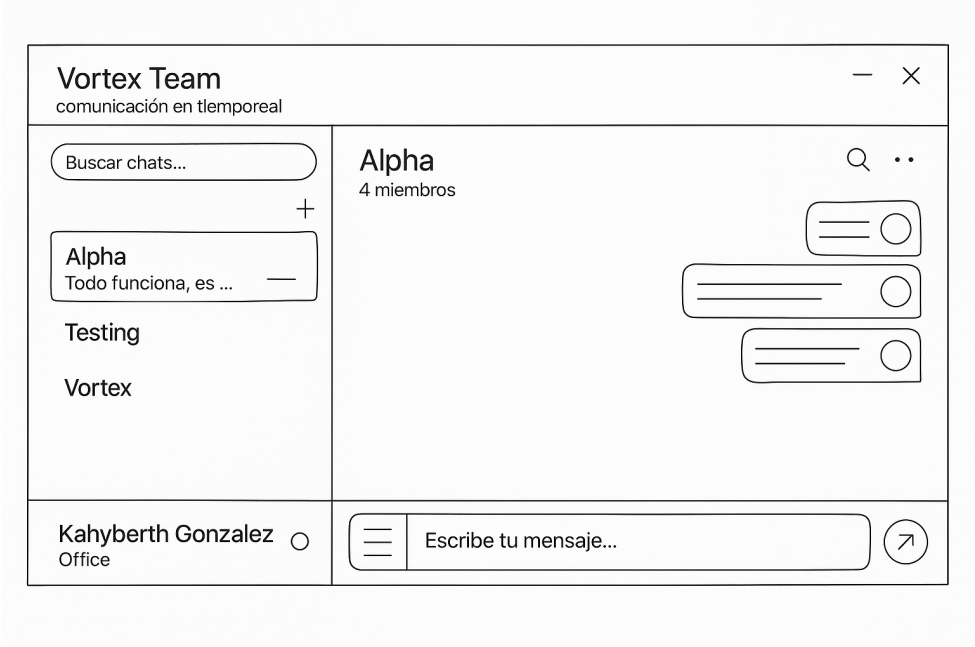

Figura 8: Wireframe de la sección de chat dentro de un equipo.

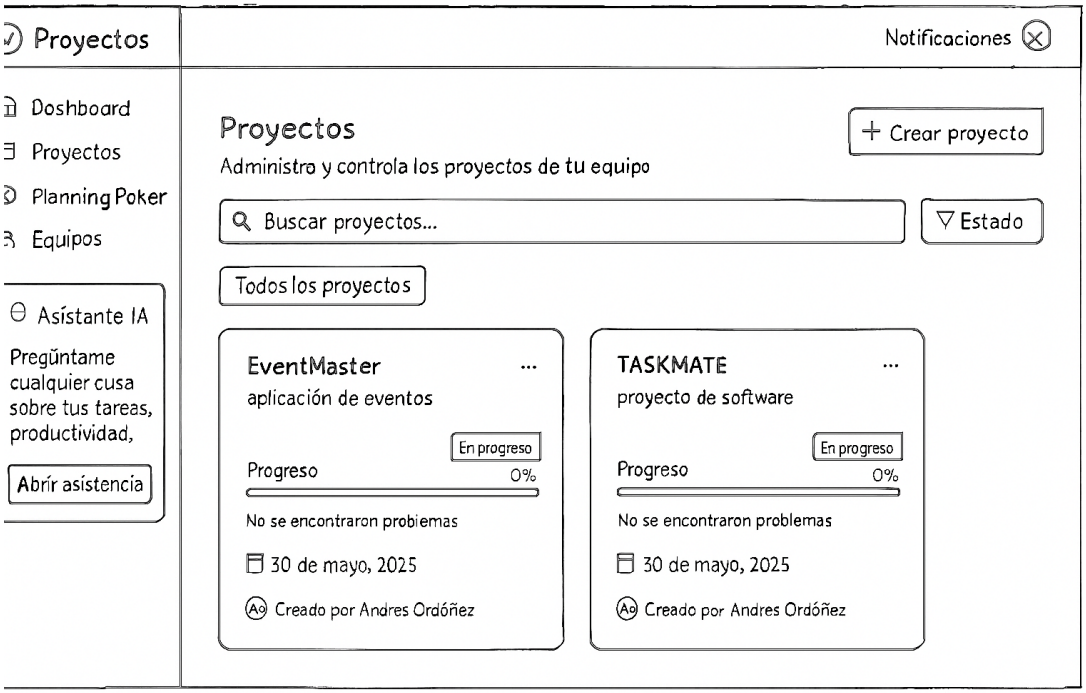

Figura 9: Wireframe de la pagina de proyectos, donde se muestran cada uno de los proyectos a los que pertenece el usuario.

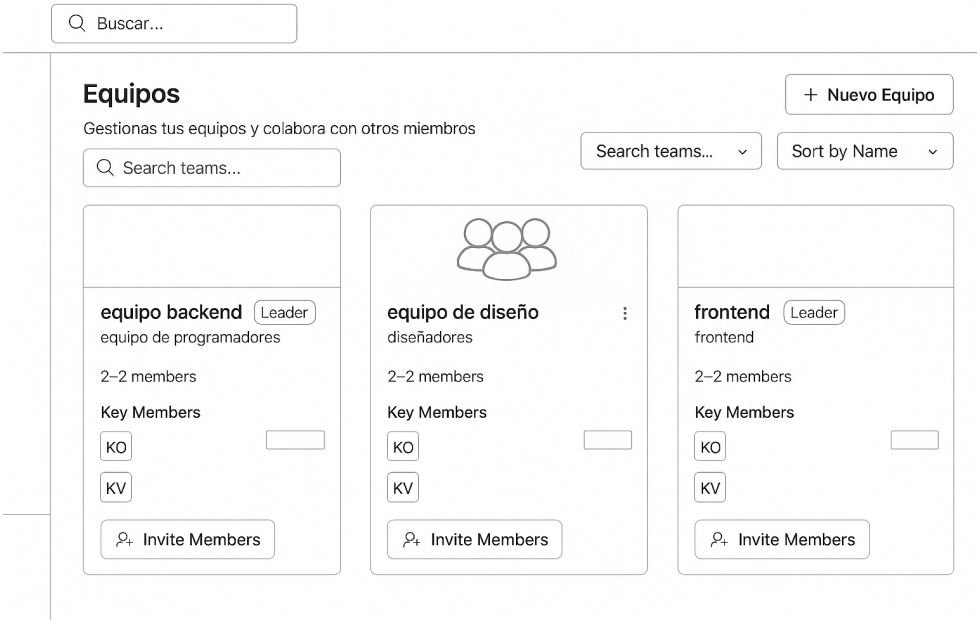

Figura 10: Wireframe de la sección de equipos, donde se listan los integrantes y su información de contacto.

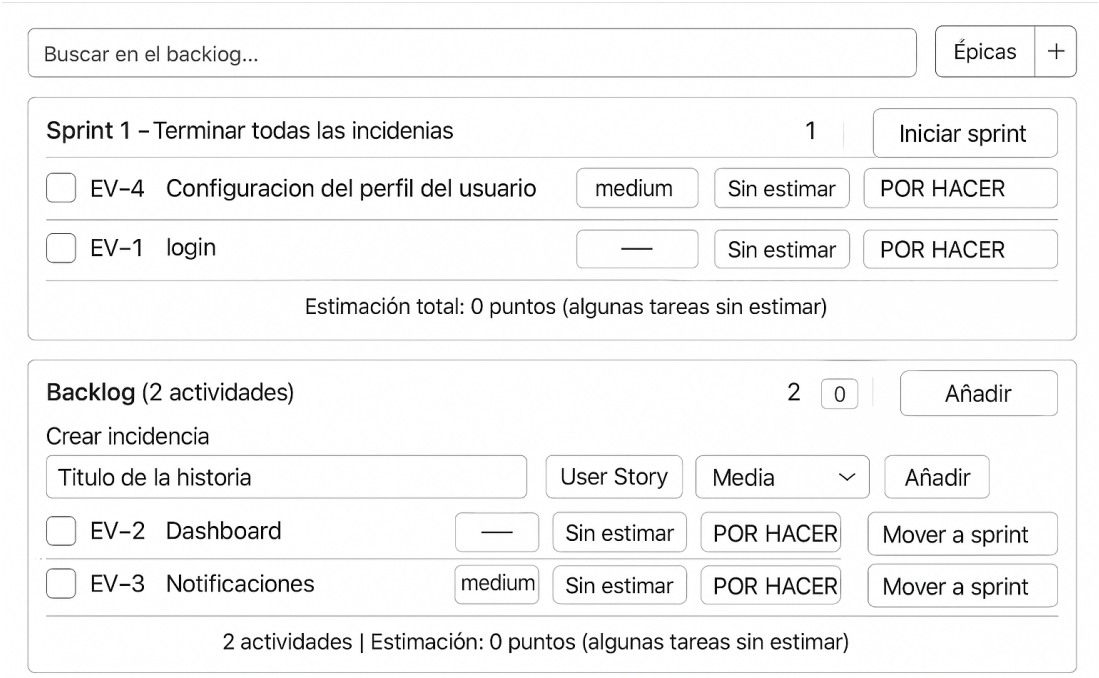

Figura 11: Wireframe de la sección de backlog, donde se encuentra tanto el product backlog como el sprint de un proyecto.

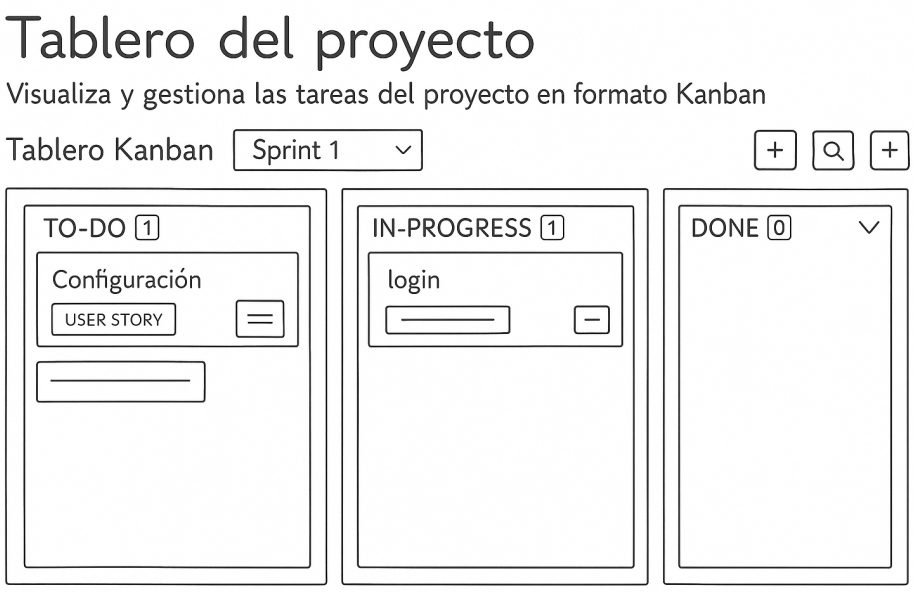

Figura 12: Wireframe del tablero kanban de un proyecto, donde se visualiza mejor el estado de las issues del sprint.

6.3. Evolución del Diseño

Durante el desarrollo, el diseño de la interfaz pasó por varias iteraciones importantes. Este proceso de evolución fue clave para llegar a una experiencia de usuario más pulida y funcional.

6.3.1. Comparación con diseños iniciales

Dashboard Principal: El dashboard experimentó los cambios más significativos durante el desarrollo. Las primeras versiones tenían un enfoque muy diferente en cuanto a la organización y presentación de la información.

En las figuras 13 y 14 se puede ver cómo los diseños iniciales se caracterizaban por tener alta densidad de información, múltiples gráficos y métricas concentrados en poco espacio y secciones innecesarias que ocupaban espacio que podía ser mejor aprovechado para visualizar las gráficas de manera más clara.

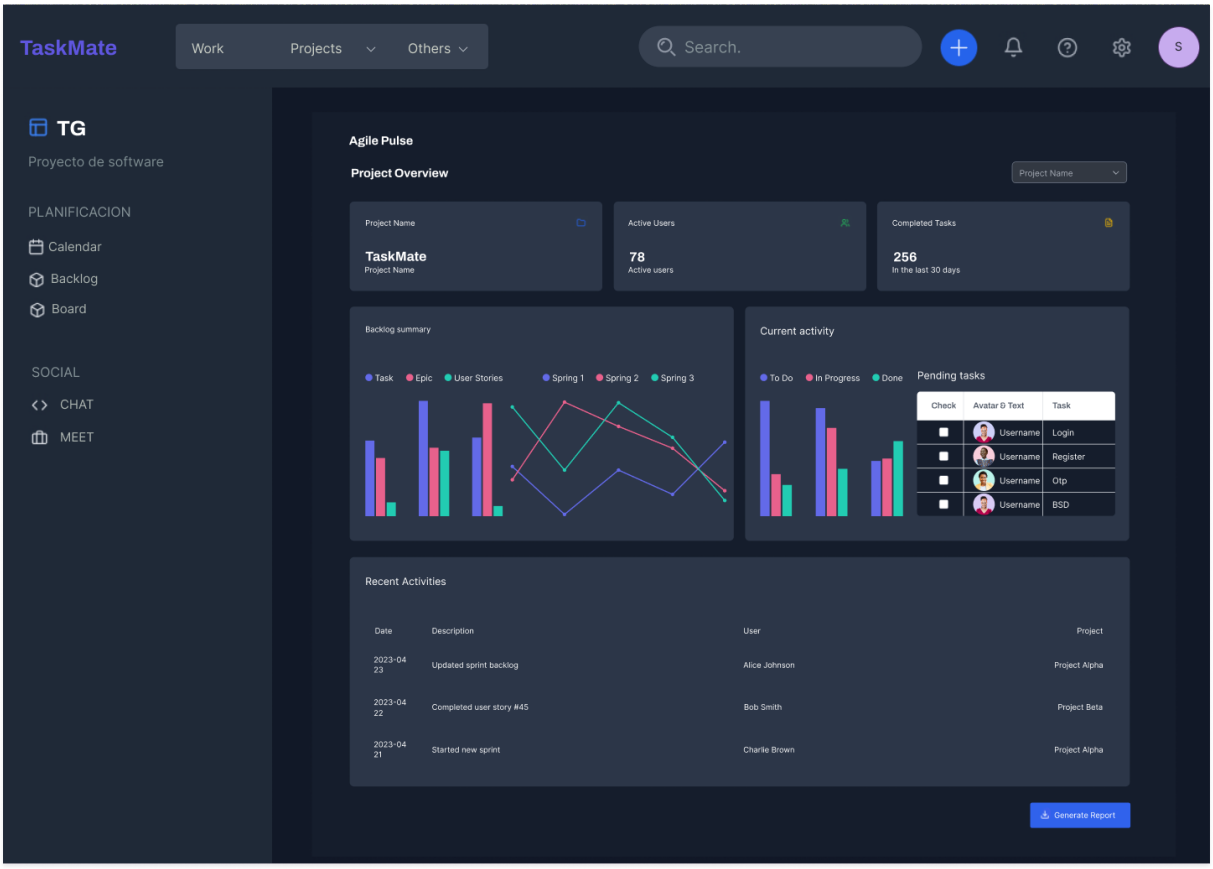

Figura 13: Primera versión del dashboard con tema oscuro

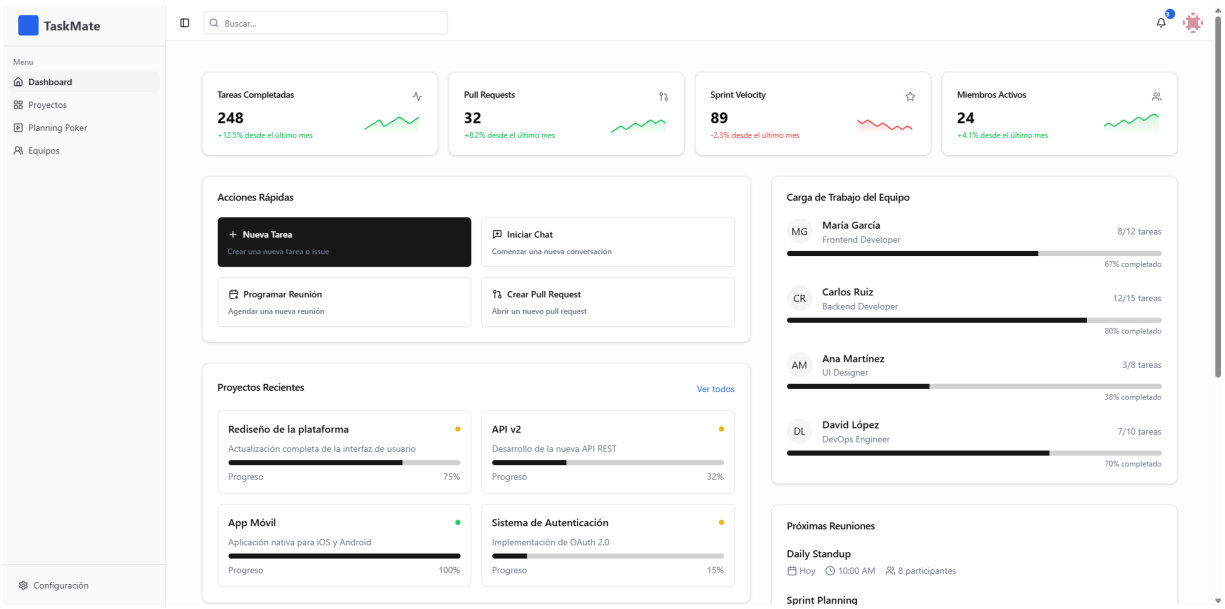

Figura 14: Segunda iteración del dashboard

La versión final implementa cambios significativos; se hizo una mejor distribución del espacio pasando a tener una interfaz más limpia, simplificando los elementos visuales con más espacio en blanco y un enfoque en la usabilidad priorizando la facilidad de uso sobre mostrar mucha información simultáneamente. Además, los primeros diseños no se implementaron porque requerían funcionalidades más complejas que probablemente no alcanzábamos a completar en el tiempo disponible. El diseño actual cumple mejor con los objetivos del proyecto, resultando en un dashboard más intuitivo que permite a los usuarios encontrar rápidamente lo que necesitan.

Figura 15: Dashboard final con diseño simplificado

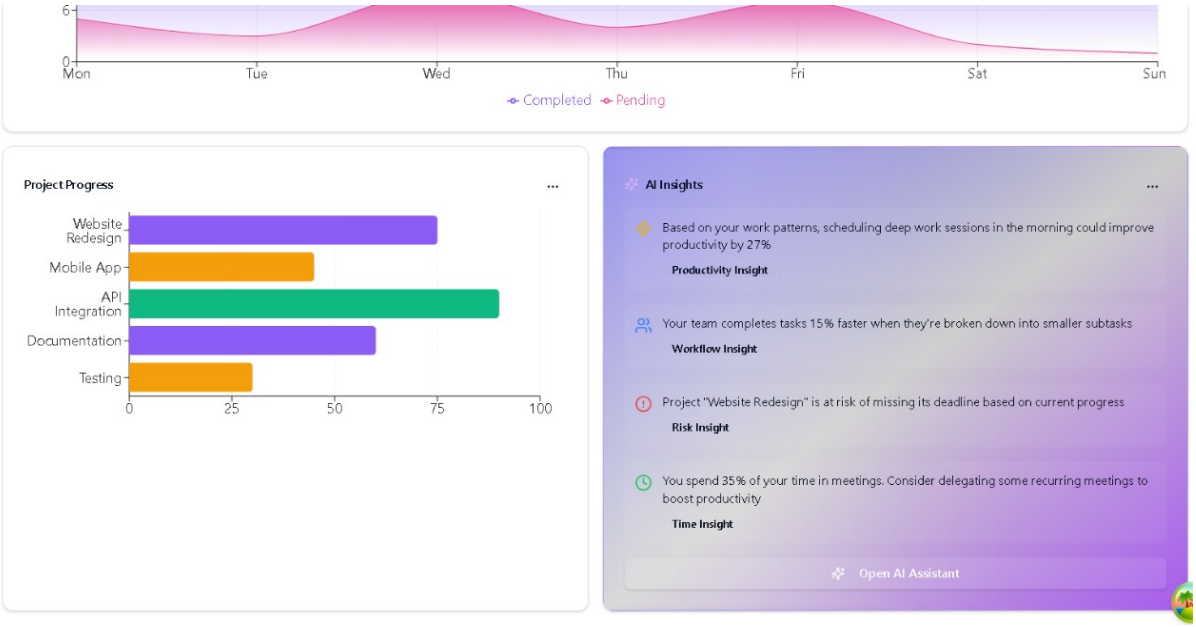

Figura 16: Vista ampliada de los elementos principales del dashboard actual

Planning Poker: La sala de planning poker también tuvo una evolución significativa en su diseño, pasando de una interfaz básica centrada únicamente en la votación hacia un diseño más completo y con funcionalidades ampliadas.

En la versión inicial (Figura 17), la interfaz se caracterizaba por tener una funcionalidad básica centrada únicamente en la votación, con un panel lateral de issues limitado a funciones básicas de votación y un área central simple que mostraba las cartas de participantes junto con el botón de para revelar los votos. Esta versión carecía de funcionalidades complementarias para el seguimiento de sesiones y presentaba una interfaz minimalista sin herramientas de colaboración adicionales.

La versión final (Figura 18) incorpora grandes mejoras en la experiencia de usuario, presentando una interfaz más limpia y moderna con mejor organización visual, una distribución más equilibrada de los elementos en pantalla y mayor claridad en la navegación y controles.

Entre las funcionalidades ampliadas se destaca la incorporación de un historial de votos que permite revisar las votaciones anteriores de la sesión, un nuevo panel de participantes que muestra información detallada de los miembros del equipo conectados, y un chat integrado con funcionalidad de comunicación en tiempo real para facilitar las discusiones durante la sesión. Además, se añadieron controles de tiempo con indicadores visuales y botones más intuitivos para gestionar las sesiones de votación.

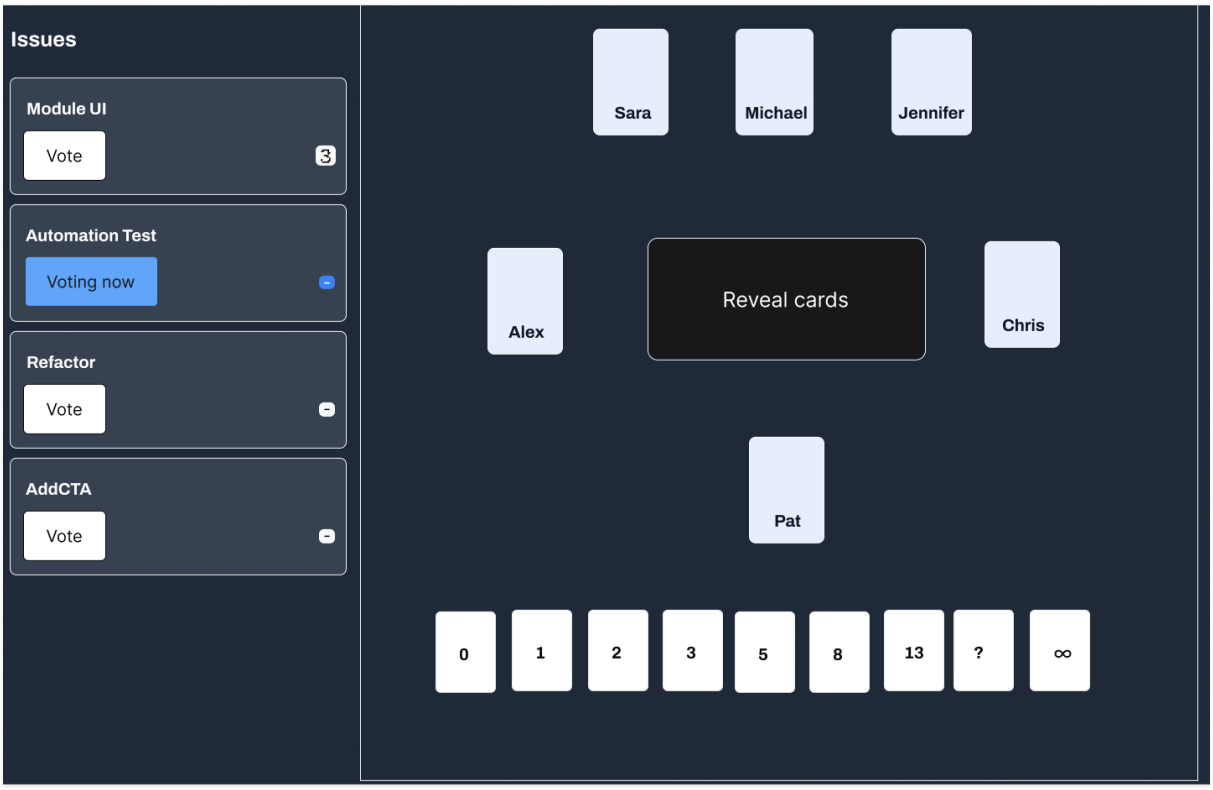

Figura 17: Primera versión de la sala de planning poker

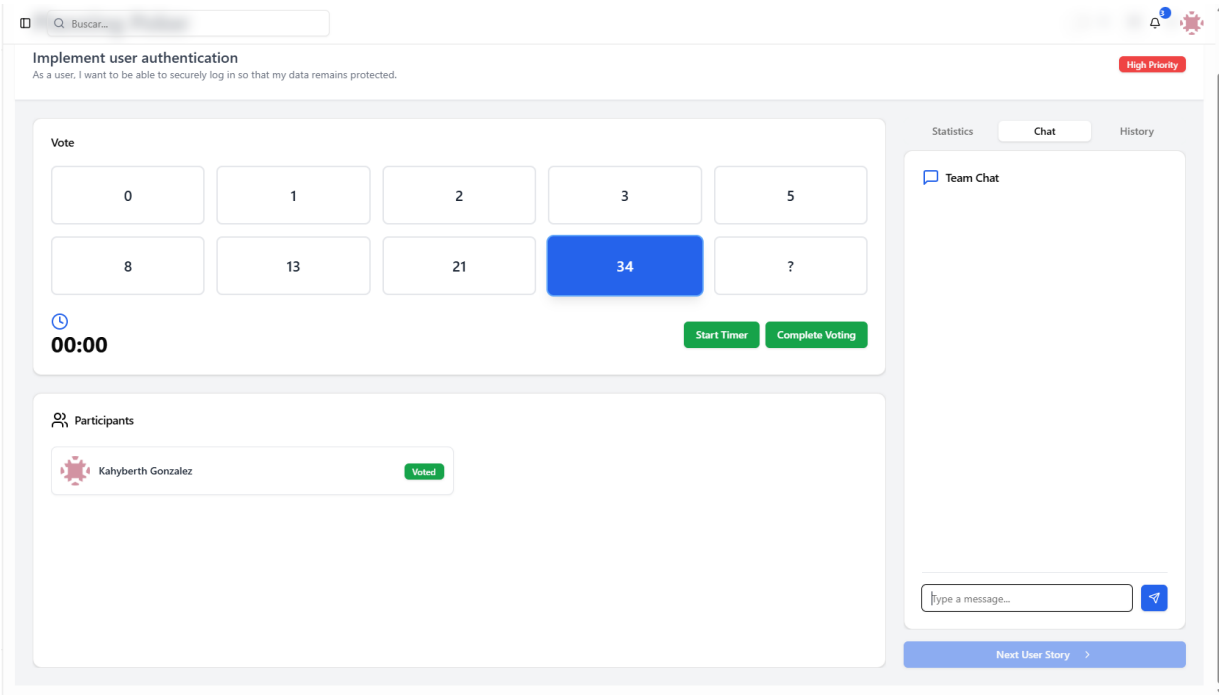

Figura 18: Versión final de la sala de planning poker con diseño mejorado

Landing Page: La página de inicio también experimentó una transformación significativa, ya que representa el primer punto de contacto con los nuevos usuarios y es crucial para generar una primera impresión positiva que invite a explorar la aplicación.

La versión inicial (Figura 19) presentaba un diseño funcional pero básico, con elementos distribuidos de manera simple y una presentación estática de las características de la aplicación. Aunque cumplía con el propósito informativo, carecía del impacto visual necesario para captar efectivamente la atención de los visitantes y transmitir la propuesta de valor de manera atractiva.

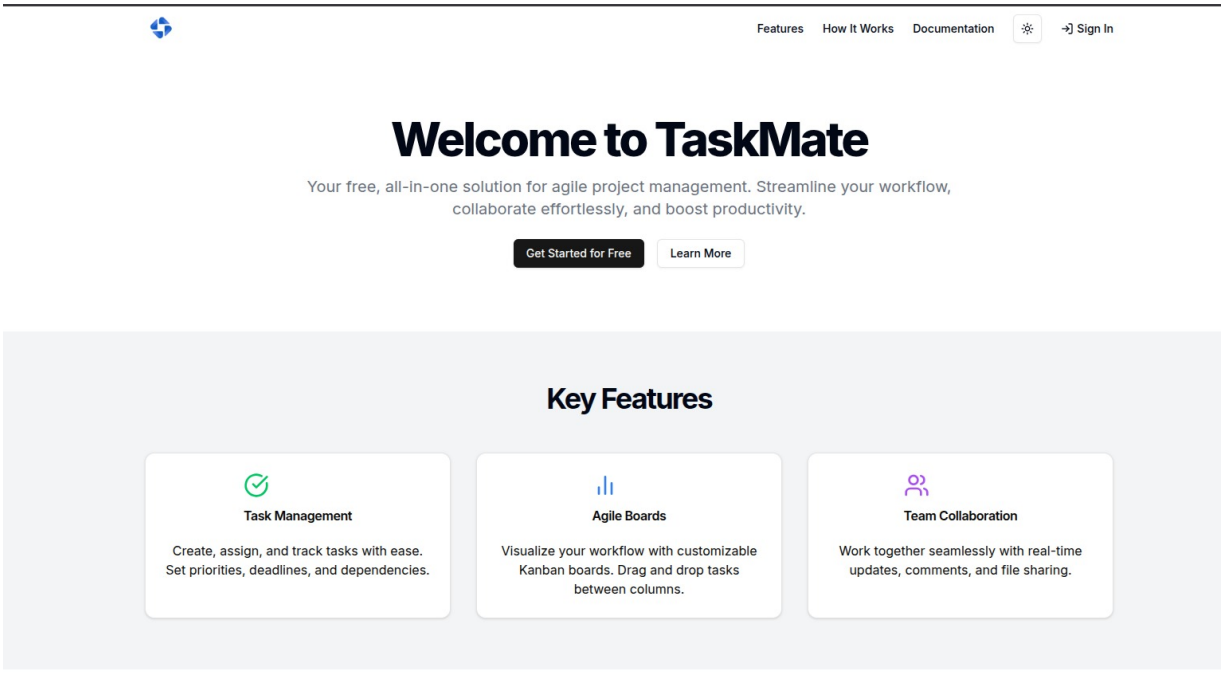

Figura 19: Versión inicial de la landing page

La versión final (Figura 20) presenta un diseño mucho más atractivo y profesional, con colores llamativos, efectos visuales elegantes y una mejor distribución de los elementos. Las características principales se muestran de forma más clara en las tarjetas con íconos que se entienden de inmediato, y simplificamos el proceso para que registrarse sea más sencillo. Con estos cambios buscamos crear una primera impresión realmente buena que transmita la calidad de nuestra aplicación y logre que más visitantes se animen a registrarse.

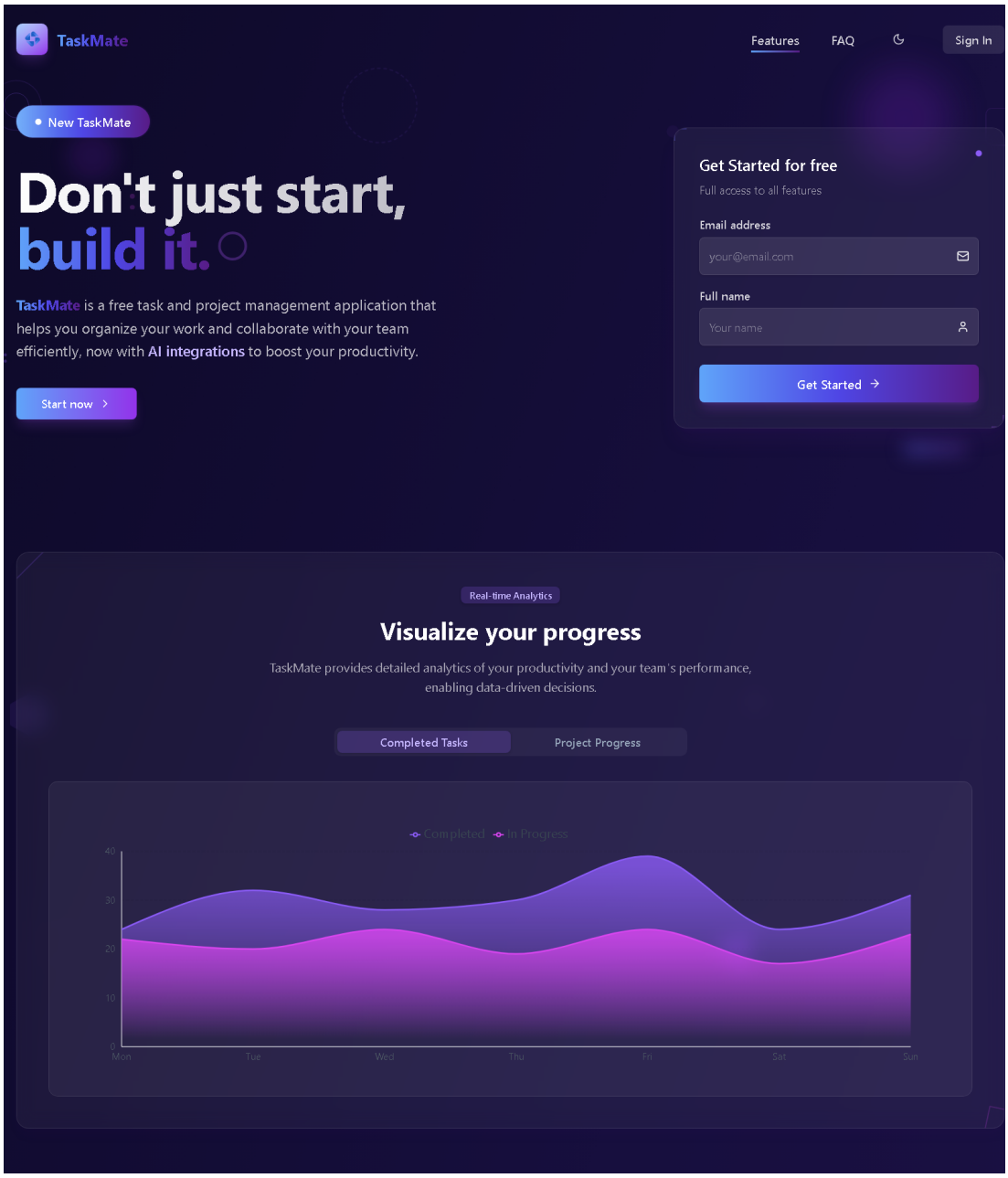

Figura 20: Versión final de la landing page con diseño mejorado

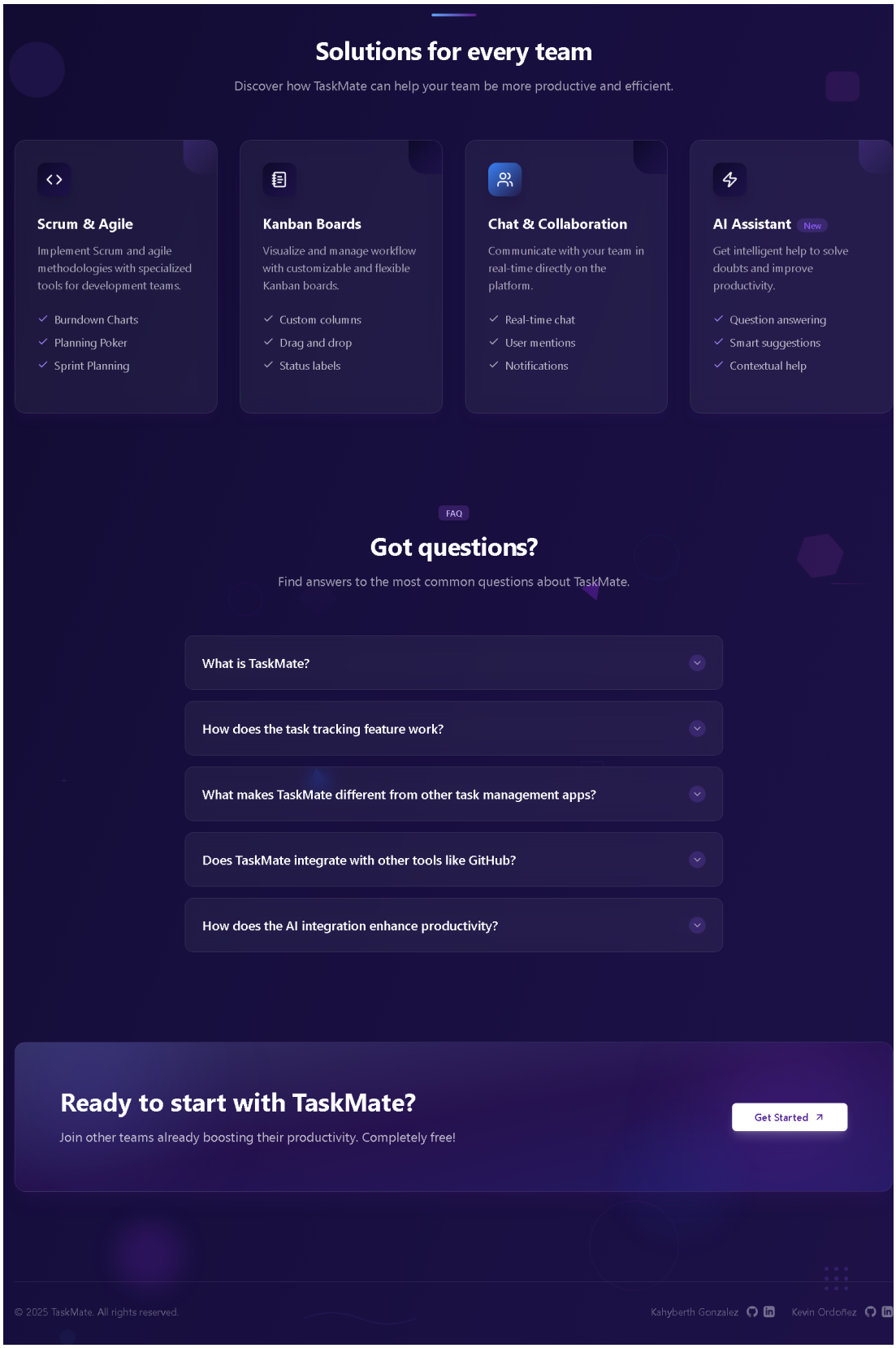

Figura 21: Continuación de la landing page final

Cabe destacar que las otras interfaces del sistema no necesitaron modificaciones importantes, conservando sus diseños originales con pequeños refinamientos.

Los demás diseños antiguos se encuentran disponibles en el Anexo 12.

6.4. Diseños finales de la interfaz

Las siguientes imágenes corresponden a los demás diseños definitivos con todos los elementos gráficos implementados.

Figura 22: Logo de la aplicación.

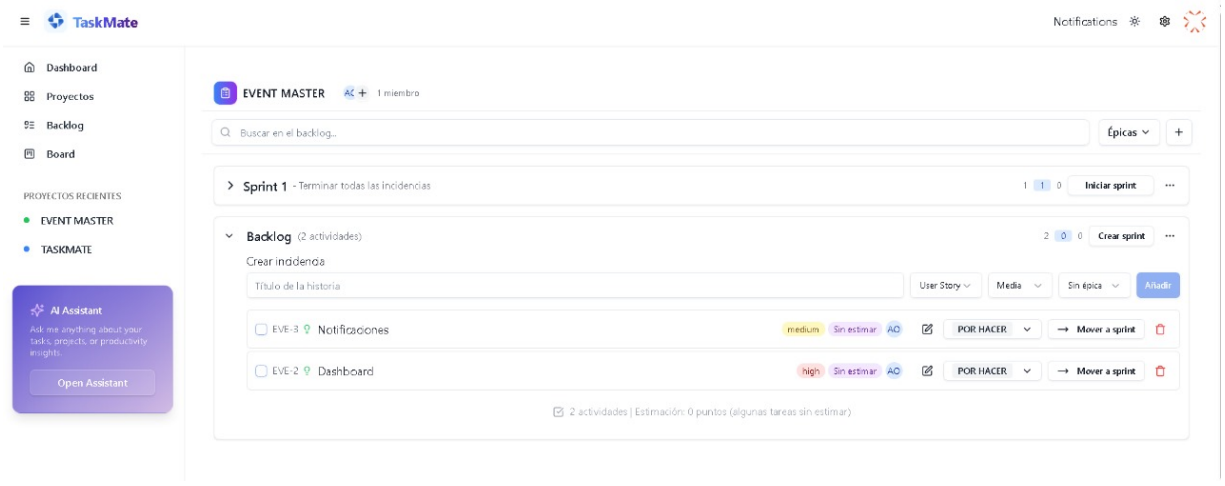

Figura 23: Diseño final de la sección *Product Backlog*, donde se muestran las issues del product backlog y el sprint del proyecto.

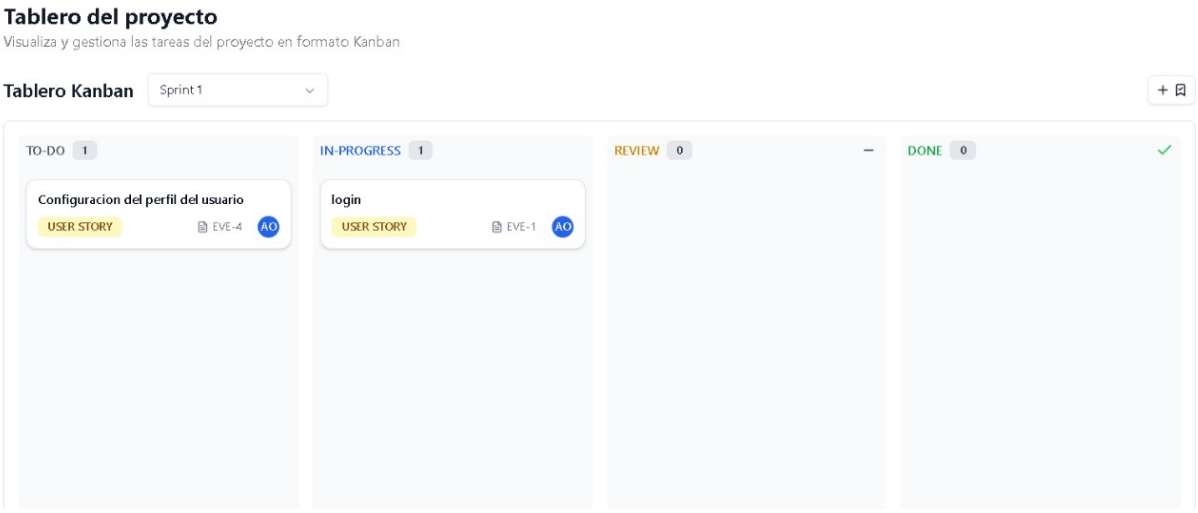

Figura 24: Diseño final de un *tablero Kanban* que permite visualizar las issues de forma más clara y gráfica.

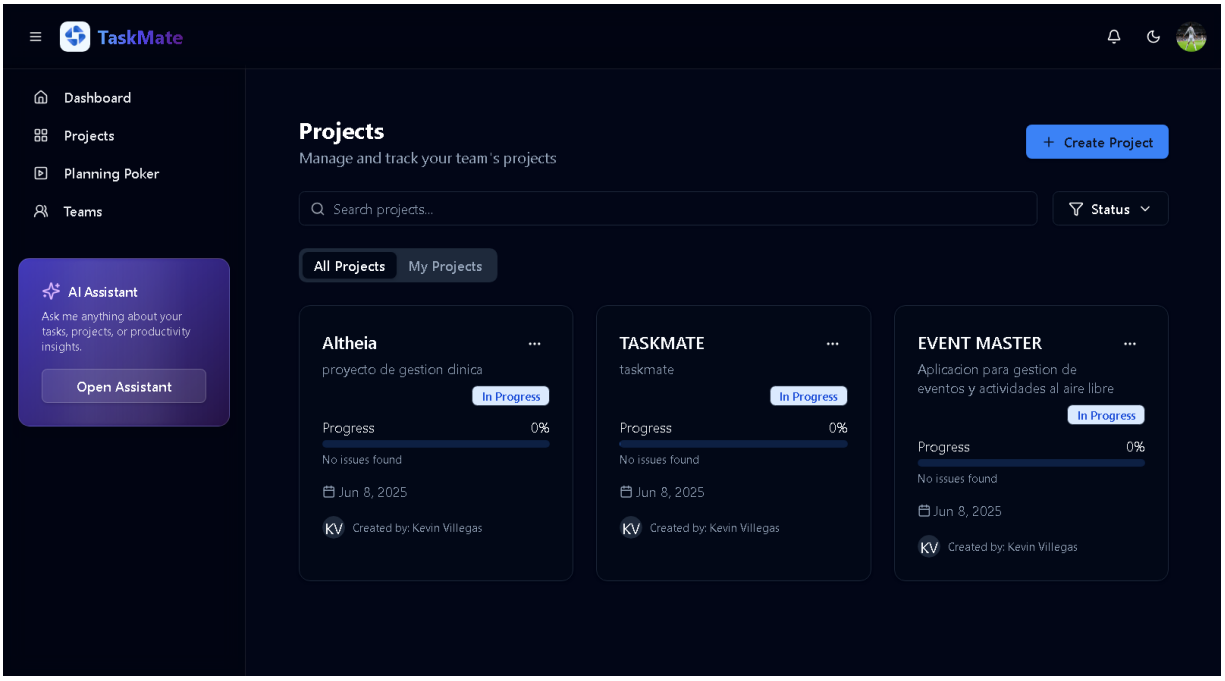

Figura 25: Diseño final de pagina de *proyectos* en modo oscuro.

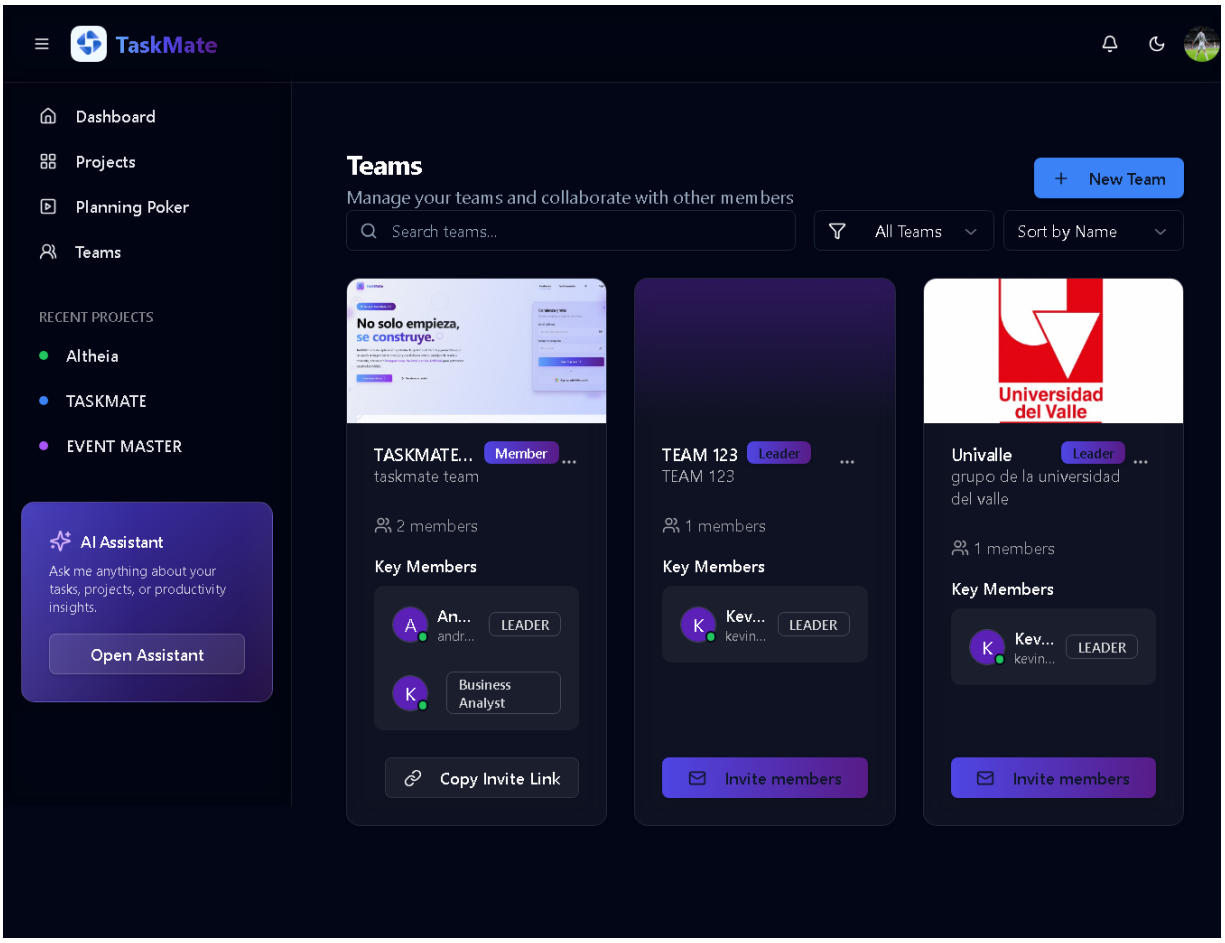

Figura 26: Diseño final de pagina de *proyectos* en modo oscuro.

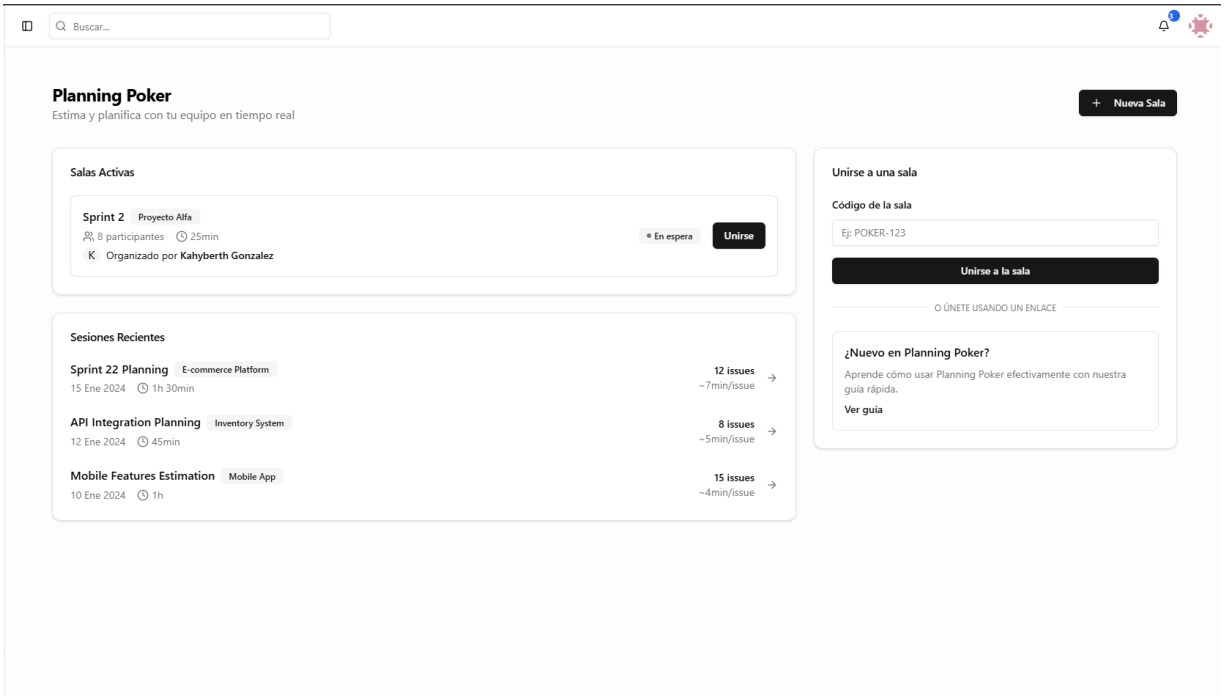

Figura 27: Diseño final del panel de inicio del módulo *Planning Poker*, con accesos rápidos a salas creadas.

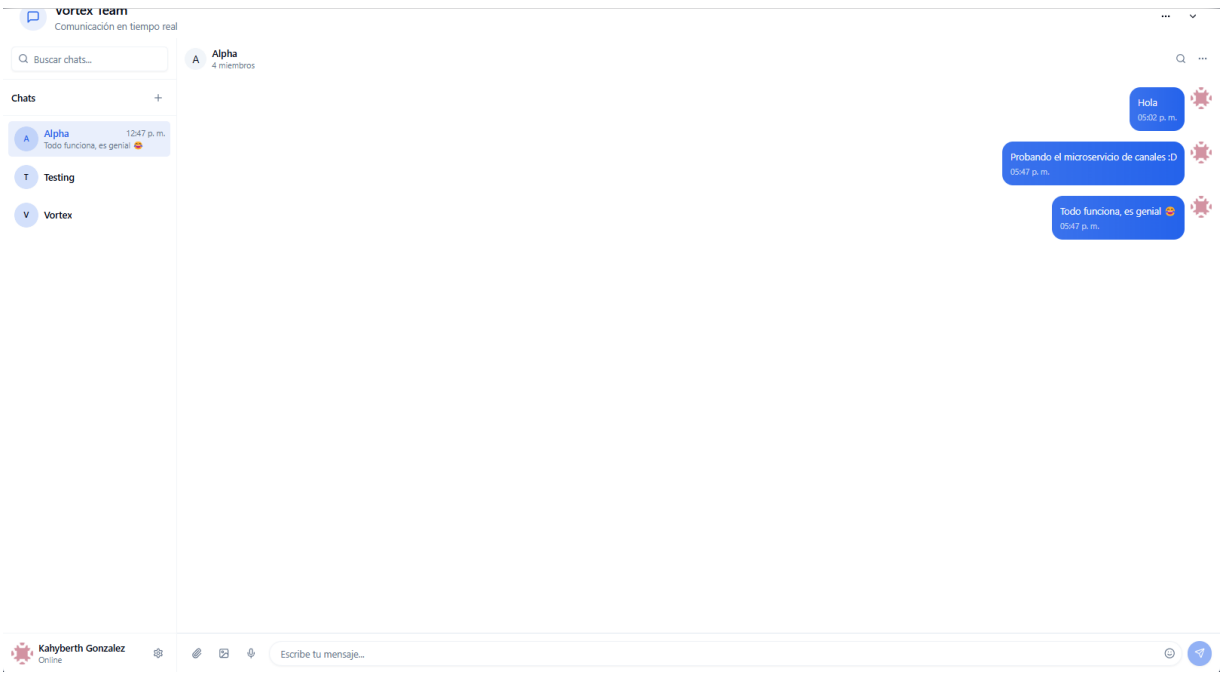

Figura 28: Diseño final del chat dentro de un equipo.

6.5. Principios de Diseño Aplicados

Durante el desarrollo de nuestro sistema, aplicamos una serie de principios fundamentales que nos ayudó a tomar la mejor decisión a la hora de hacer cada diseño. Como ya se mencionó anteriormente, durante el desarrollo, realizamos algunas modificaciones de los primeros diseños, con el objetivo de asegurar la coherencia, escalabilidad y eficiencia del sistema a largo plazo.

- **Consistencia**: Se desarrolló un sistema de componentes reutilizables y patrones visuales unificados, lo cual permitió mantener coherencia visual en toda la aplicación y reducir bastante el tiempo de desarrollo.
- **Usabilidad**: Se priorizó un flujo de navegación intuitivo centrado en las necesidades reales de los usuarios, ya que se identificó que la experiencia del usuario sería determinante para el éxito de la aplicación. Además, evitamos hacer un diseño demasiado radical o experimental, ya que los usuarios están más familiarizados con ciertos patrones comunes presentes en aplicaciones similares. Esta decisión tiene como fin reducir la curva de aprendizaje y garantizar una interacción más fluida y natural con la plataforma.
- **Escalabilidad**: Se diseñaron interfaces adaptables capaces de ajustarse a distintas resoluciones y dispositivos. Esto asegura que la experiencia de uso se mantenga consistente y funcional, sin importar el entorno desde el cual se acceda al sistema.

6.6. Evaluación y Mejoras Implementadas

Durante el desarrollo realizamos algunas pruebas que nos permitieron identificar y corregir aspectos importantes de la experiencia de usuario.

- **Reorganización de la navegación**: Ajustamos la estructura del menú basándonos en cómo los usuarios realmente buscaban las funciones, reorganizando el acceso a herramientas como Planning Poker para que fuera más intuitivo según los flujos de trabajo naturales.
- **Optimización responsive**: Durante las pruebas en diferentes dispositivos descubrimos inconsistencias en pantallas más pequeñas. Tuvimos que ajustar varios breakpoints y reorganizar componentes para lograr una experiencia consistente en todos los tamaños de pantalla.
- **Mejora del feedback visual**: Implementamos animaciones de carga para acciones como crear issues, indicadores más claros para el estado de las votaciones, y transiciones más fluidas entre vistas para que la aplicación se sintiera más responsiva.

Estas iteraciones nos permitieron consolidar una experiencia que no solo cumple técnicamente, sino que resulta natural e intuitiva para el uso diario.

7. Bases de Datos y Diagramas Entidad-Relación

7.1. Arquitectura de Bases de Datos por Microservicio

Cada microservicio del sistema cuenta con su propia base de datos independiente, respetando el principio de separación de responsabilidades y reduciendo el acoplamiento entre dominios. Esta decisión arquitectónica favorece la escalabilidad, mejora la seguridad y facilita el mantenimiento del sistema.

Se ha utilizado **PostgreSQL** como sistema gestor de bases de datos relacional en todos los microservicios, debido a su robustez, compatibilidad con arquitecturas distribuidas y el soporte activo de la comunidad. Como proveedor de infraestructura, se empleó principalmente **Neon**, una plataforma de base de datos en la nube optimizada para PostgreSQL, que permite escalabilidad automática, respaldos instantáneos y baja latencia.

7.2. Diseño de Bases de Datos por microservicio

- **Autenticación y Gestión de Equipos:**

  Esta base de datos almacena tanto la información de los usuarios como de los equipos a los que pertenecen. Incluye tablas para credenciales, perfiles, relaciones usuario-equipo y control de accesos.A continuación se describen las tablas involucradas:

  - `User`: Contiene la información principal del usuario, incluyendo nombre, correo, contraseña, idioma, empresa, disponibilidad, estado de cuenta y fechas de creación y última actividad.
  - `Profile`: Información extendida del usuario como foto de perfil, biografía, habilidades, ubicación, zona horaria y enlaces sociales. Está vinculada a la tabla `User` mediante la clave foránea `user_id`.
  - `Role`: Representa los distintos roles del sistema (por ejemplo: administrador, miembro, invitado). Contiene el nombre del rol y sus fechas de creación y modificación.
  - `UsersRole`: Tabla de relación entre usuarios y roles. Permite asignar múltiples roles a un usuario con su respectiva fecha de incorporación.
  - `Team`: Contiene los equipos de trabajo registrados en el sistema. Incluye el nombre del equipo, descripción, imagen y el identificador del usuario líder (`leaderId`).
  - `UsersTeam`: Tabla de relación entre usuarios y equipos. Registra en qué equipo participa cada usuario y cuál es su rol dentro del mismo (`roleInTeam`).

  El diagrama entidad-relación de este módulo se muestra en la Figura 29 y su modelo relacional en la Figura 30.

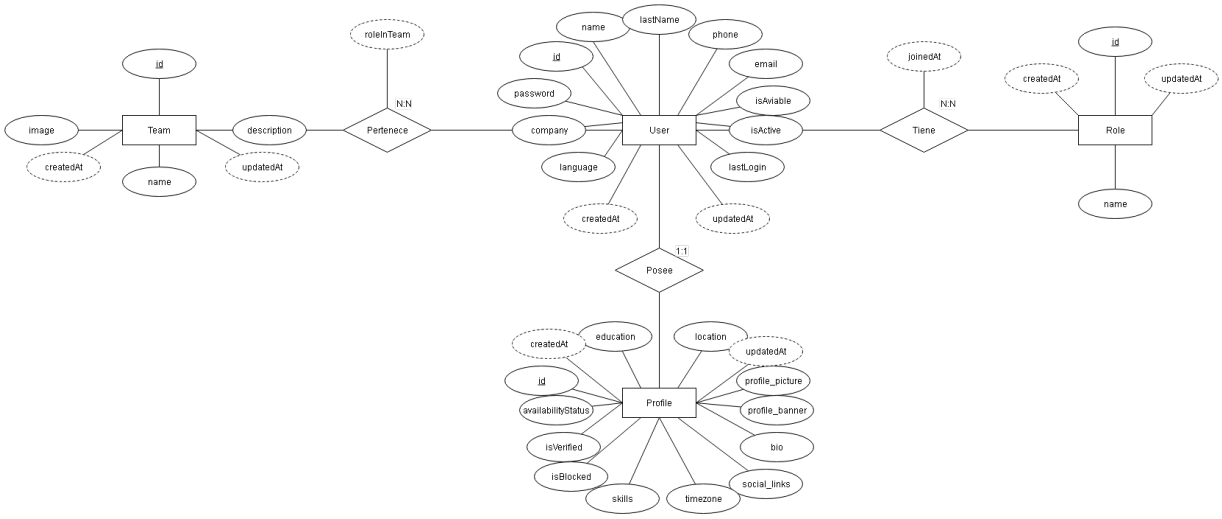

Figura 29: Diagrama MER del microservicio de autenticación y gestión de equipos

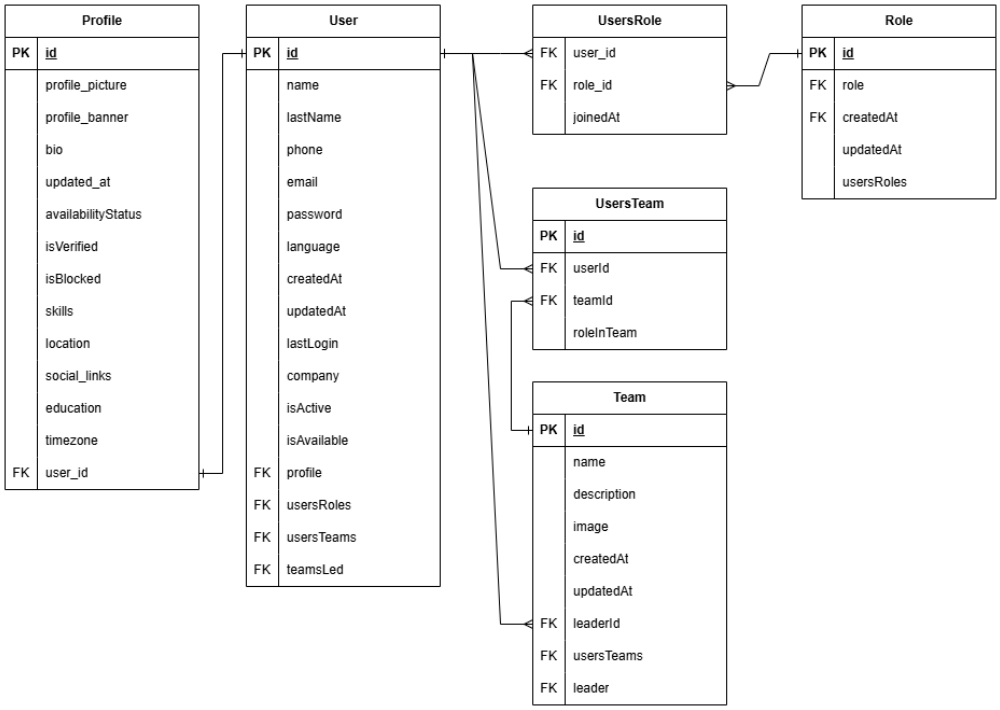

Figura 30: Modelo relacional del microservicio de autenticación

- **Gestión de Proyectos y Sprints:**

  El microservicio `Projects` cuenta con una base de datos estructurada para representar la organización jerárquica de un proyecto ágil, permitiendo la trazabilidad entre proyectos, usuarios, historias, tareas y ciclos de trabajo. A continuación, se describen las tablas que componen el esquema relacional:

- `Projects`: Almacena la información general de cada proyecto, incluyendo su nombre, descripción, estado, fechas de inicio y finalización, así como la prioridad del proyecto.
- `Project_Users`: Tabla de relación que vincula a los usuarios con los proyectos en los que participan. Se especifica el rol que cumple cada usuario dentro del proyecto y su estado de participación.
- `Epics`: Representa agrupaciones de funcionalidades extensas o grandes objetivos dentro de un proyecto. Están vinculadas a un proyecto específico y pueden contener múltiples historias de usuario.
- `User_Stories`: Contiene el detalle de cada historia de usuario, la cual describe una funcionalidad específica a implementar. Incluye campos como título, descripción, prioridad, estado y tiempo estimado de desarrollo.
- `Backlogs`: Define el backlog del producto o del sprint, organizando las historias de usuario en listas priorizadas para su desarrollo.
- `Backlog_User_Stories`: Relación entre el backlog y las historias de usuario asignadas a él. Esta tabla permite que una historia esté asociada a un backlog determinado y controla su orden y prioridad.
- `Sprints`: Representa los ciclos de desarrollo iterativos dentro de un proyecto. Contiene información como fechas de inicio y finalización, objetivo del sprint, estado y número de sprint.
- `Sprint_User_Stories`: Registro que enlaza las historias de usuario asignadas a un sprint específico. También guarda el estado de avance de cada historia dentro del sprint.
- `Sprint_Users`: Registra qué usuarios están asignados a un sprint determinado y su rol dentro del mismo (por ejemplo, Scrum Master, Developer, Product Owner).
- `Tasks`: Representa las tareas técnicas que se derivan de una historia de usuario. Incluye detalles como título, descripción, estado, tiempo estimado, prioridad y la historia de usuario a la que pertenecen.
- `Task_Assignees`: Tabla intermedia que vincula las tareas con los usuarios responsables de ejecutarlas, permitiendo múltiples asignaciones.
- `Comments`: Contiene los comentarios realizados por los usuarios sobre las historias de usuario, tareas u otros elementos del proyecto. Incluye el contenido del comentario, autor, fecha y tipo de entidad comentada.
- `Attachments`: Archivos adjuntos subidos por los usuarios que están vinculados a historias de usuario, tareas o comentarios. Esta tabla registra el nombre del archivo, tipo, peso y ruta de almacenamiento.
- `Tags`: Permite etiquetar historias de usuario o tareas con palabras clave para facilitar su categorización y filtrado.

- `Entity_Tags`: Tabla de relación que asocia etiquetas con entidades del sistema (principalmente tareas e historias de usuario).

El modelo entidad-relación correspondiente se muestra en la Figura 31.

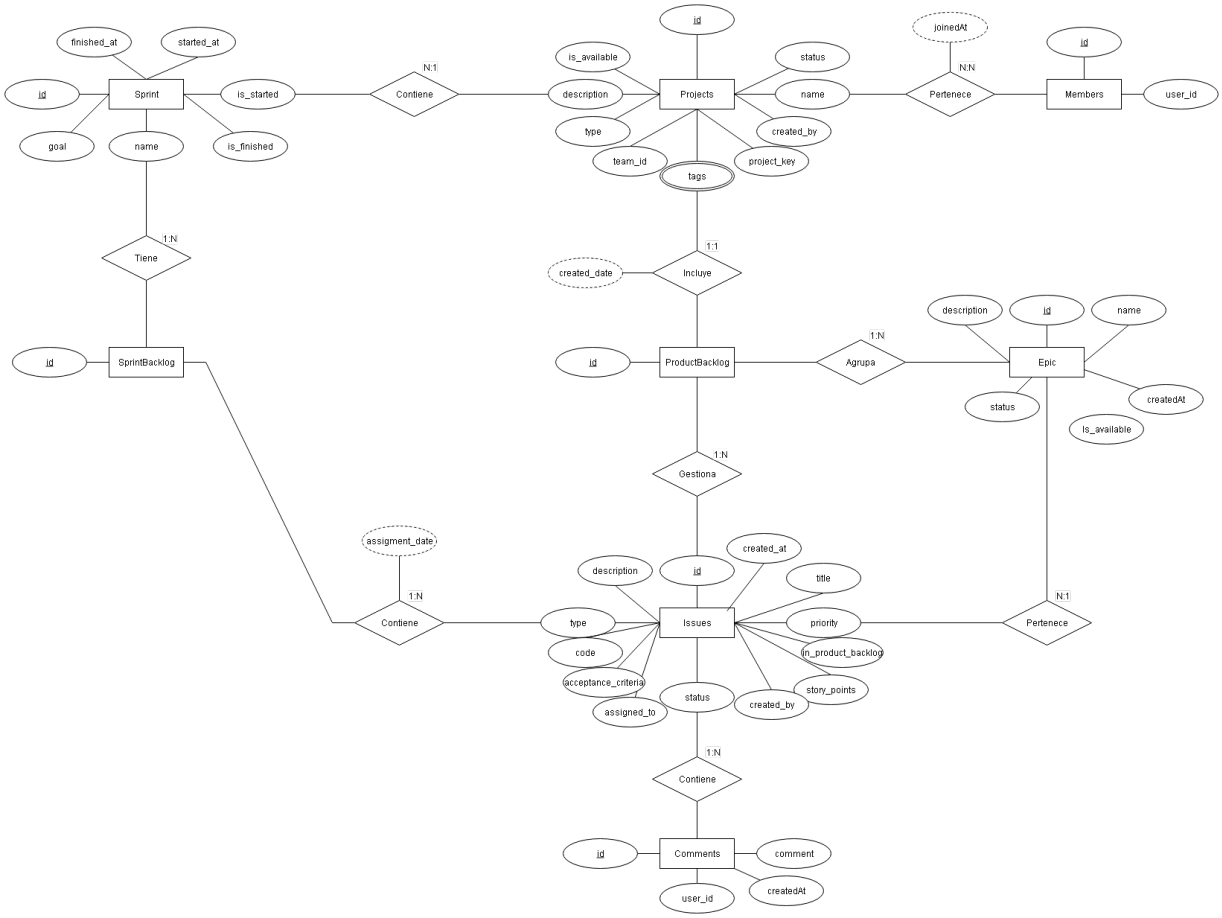

Figura 31: Diagrama MER del microservicio de proyectos

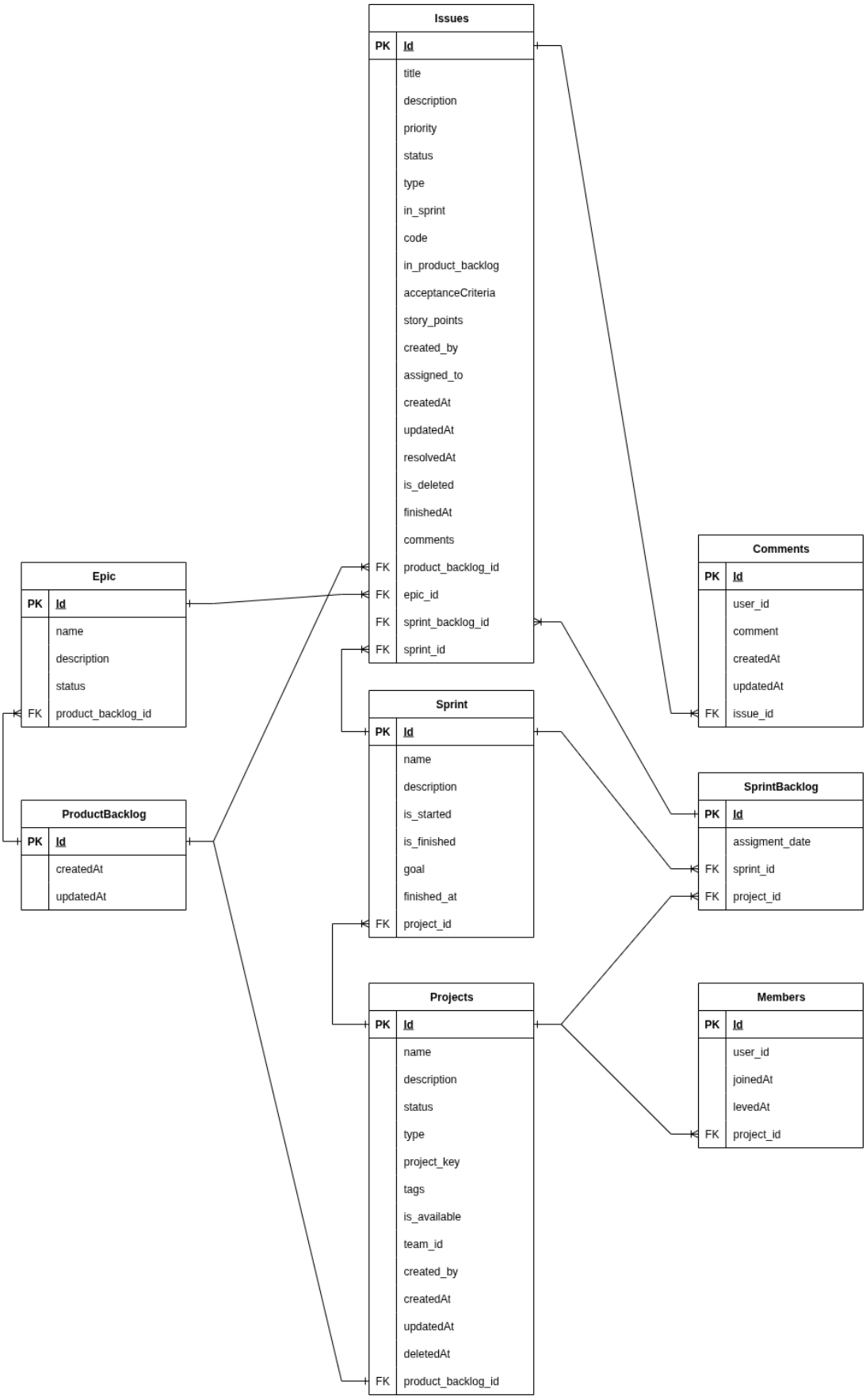

Figura 32: Modelo relacional del microservicio de proyectos

- **Comunicación (Chats y Canales):**

  La base de datos del microservicio de comunicación está diseñada para permitir la interacción entre usuarios mediante canales estructurados, ya sea en espacios grupales o privados. Las tablas principales que componen este esquema son:

  - `Channel`: Representa un canal general de comunicación. Puede ser público o privado, y permite organizar las conversaciones por contexto (proyecto, equipo, tema específico, etc.).
  - `Chat`: Subdivisión lógica dentro de un canal. Cada chat agrupa mensajes relacionados a una conversación específica. Está vinculado a un canal y puede tener participantes definidos.
  - `Message`: Contiene los mensajes enviados dentro de un chat. Cada registro almacena el contenido del mensaje, la hora de envío, el autor, y referencias al chat al que pertenece.
  - `Chat_Participants`: Tabla que define los usuarios que participan en un chat determinado. Permite gestionar la visibilidad y acceso a los mensajes.
  - `Channel_Participants`: Tabla intermedia que vincula usuarios con canales, definiendo su rol (ej. creador, miembro) y controlando los permisos de entrada a los chats dentro del canal.

  Ver diagrama en la Figura 33.

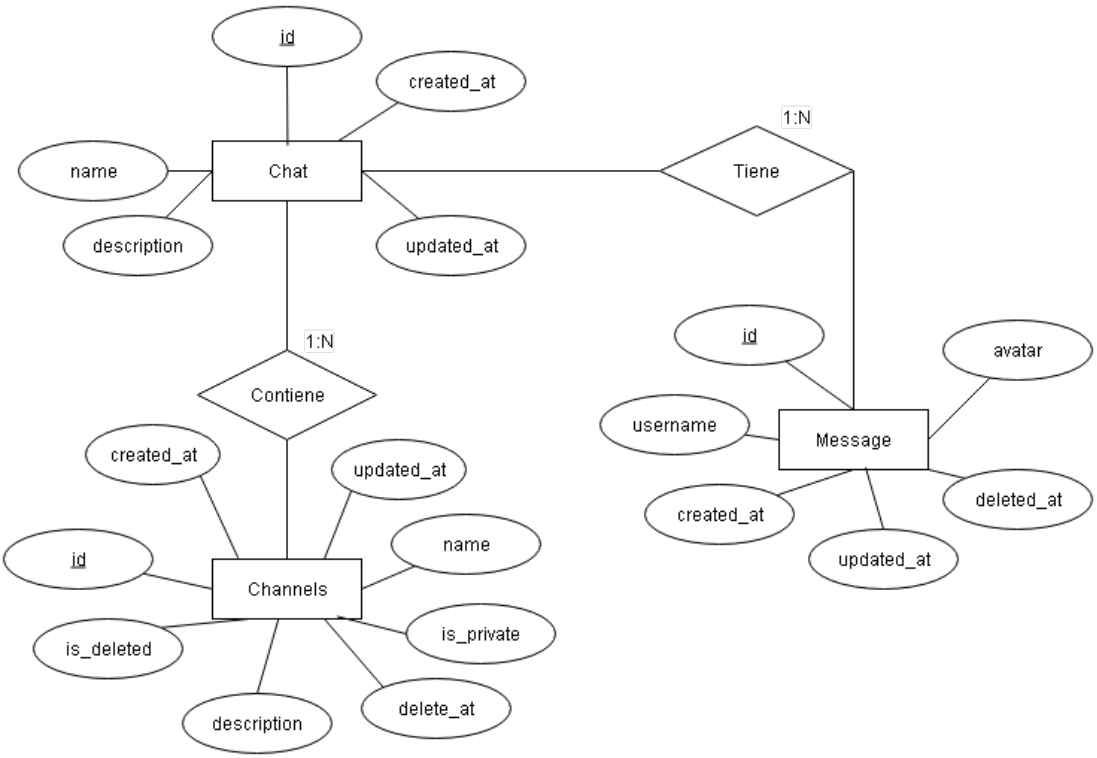

Figura 33: Diagrama MER del microservicio de comunicación

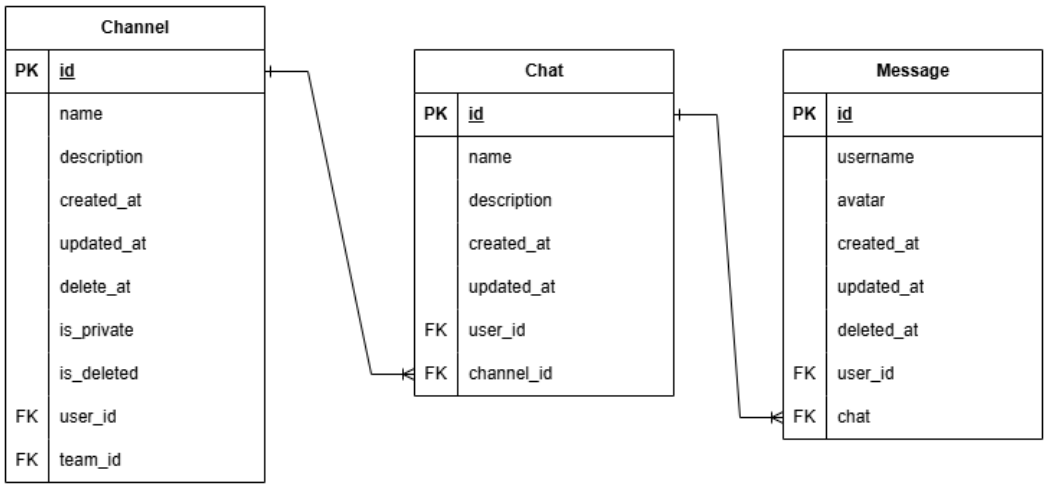

Figura 34: Modelo relacional del microservicio de comunicación

- **Planning Poker:**

  Este módulo facilita la estimación colaborativa de tareas. La base de datos incluye:

  - `Session`: Sesión de planificación con escala usada, proyecto, creador.
  - `Vote`: Registro de votos por historia de usuario y usuario.
  - `Chat`: Chat dentro de la sesión.
  - `Decks`: Mazos de cartas posibles para estimación.
  - `History`: Historial cronológico de votaciones.
  - `Join_Session`: Relación usuario-sesión.

  Diagrama en la Figura 35.

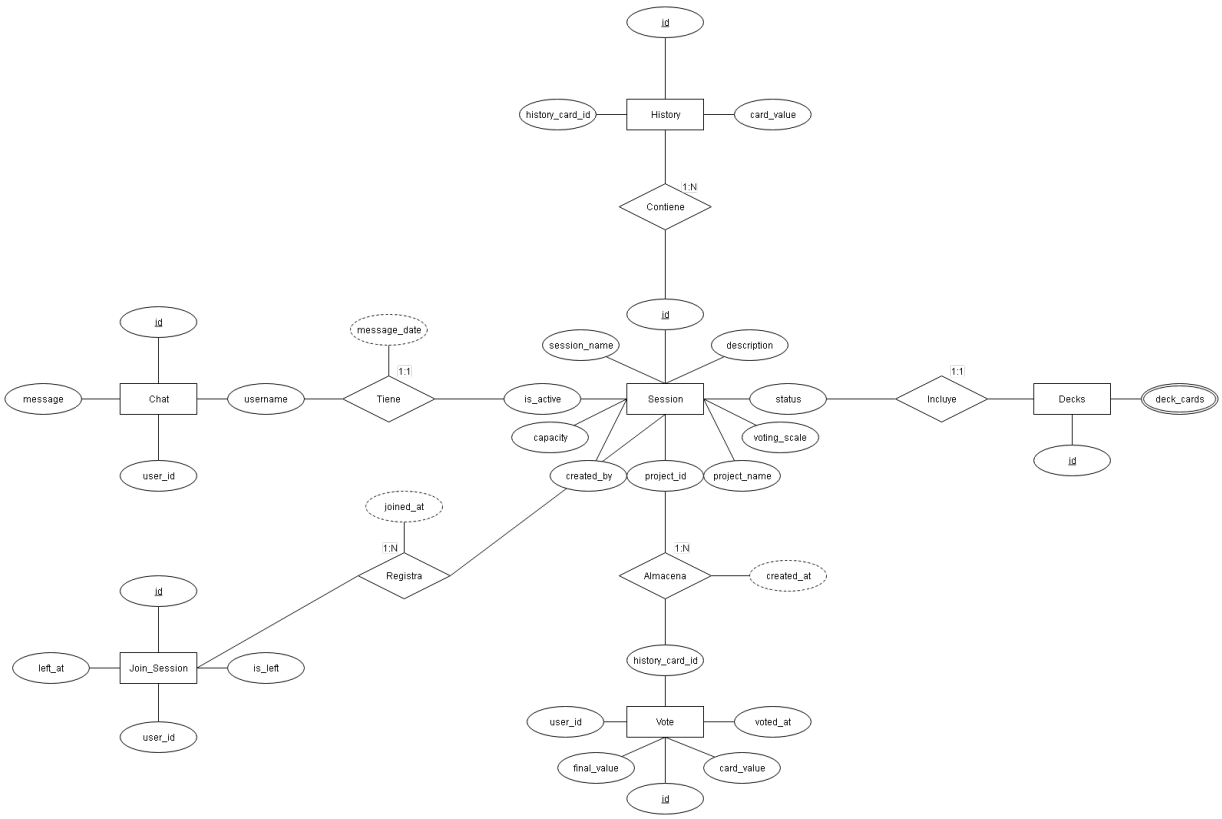

Figura 35: Diagrama MER del microservicio de Planning Poker

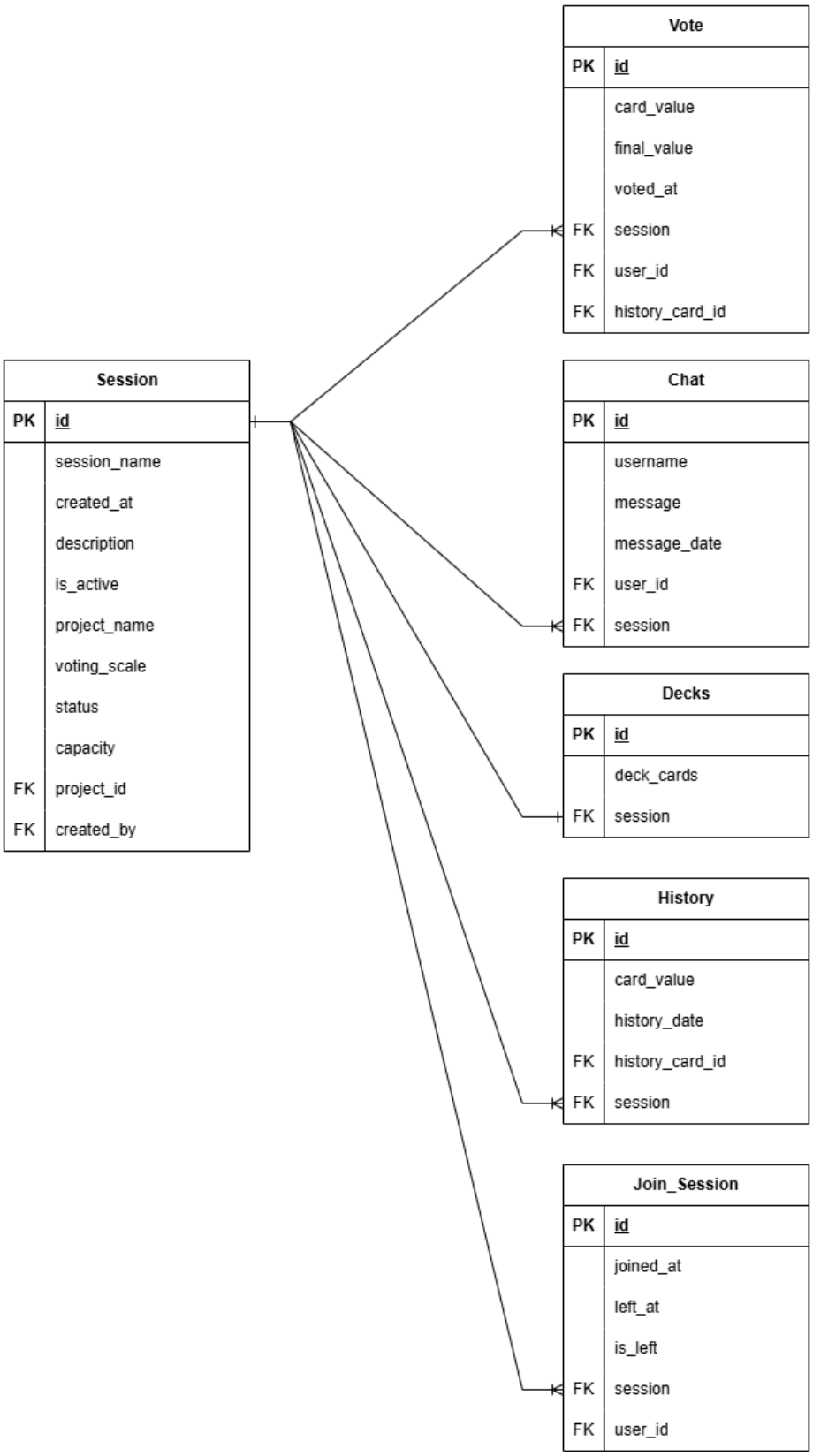

Figura 36: Modelo relacional del microservicio de Planning Poker

8. Implementación

8.1. Tecnologías Utilizadas

El sistema se desarrolló utilizando un stack tecnológico moderno que garantiza escalabilidad y mantenibilidad. A continuación, se describen las principales tecnologías empleadas tanto en el frontend como en el backend, y el propósito específico de cada una dentro del desarrollo de la aplicación.

Frontend

- **React con Vite**: Solución inicialmente planteada con Next.js, pero migrada a React + Vite al identificar que el renderizado predominante sería del lado del cliente, mejorando así el rendimiento.
- **TypeScript**: Implementado como superset de JavaScript para proporcionar tipado estático, mejorando la detección temprana de errores.
- **Tailwind CSS**: Framework CSS utilitario que agilizó el desarrollo de las interfaces mediante clases predefinidas, reduciendo la necesidad de escribir CSS personalizado y manteniendo un diseño consistente.
- **React Query**: Biblioteca implementada para el manejo eficiente del estado del servidor y caché de datos. Se integró de manera selectiva en componentes específicos que requerían sincronización de datos en tiempo real.
- **Mantine UI y Radix UI**: Bibliotecas de componentes accesibles y personalizables que proporcionaron una base sólida para la interfaz de usuario, acelerando el desarrollo de componentes complejos como modales, tabs y menús desplegables.
- **DND Kit**: Librería para crear componentes con funcionalidad de arrastrar y soltar, especialmente utilizada en la funcionalidad de tableros Kanban para la gestión de tareas.
- **Axios**: Cliente HTTP empleado para realizar peticiones a los microservicios, ofreciendo una API clara y la capacidad de interceptar respuestas para manejar errores de forma centralizada.
- **Recharts**: Biblioteca de gráficos basada en componentes React utilizada en el dashboard para visualizar datos de proyectos.

Backend

- **NestJS**: Framework de Node.js utilizado para construir la arquitectura de microservicios, proporcionando una estructura modular, escalable y basada en decoradores que facilitó el desarrollo y mantenimiento del código.

- **NATS.io**: Sistema de mensajería de alto rendimiento implementado para la comunicación asincrónica entre microservicios, permitiendo operaciones distribuidas eficientes y tolerancia a fallos.
- **Docker**: Tecnología de contenedores empleada para encapsular cada microservicio con sus dependencias, garantizando un despliegue consistente y aislado en diferentes entornos.
- **Nodemailer**: Se integró esta biblioteca para el envío automatizado de notificaciones por correo electrónico, utilizada en funcionalidades como registro de usuarios, notificaciones de invitación a equipos y a proyectos.
- **TypeORM**: ORM (Object-Relational Mapping) seleccionado para facilitar las operaciones CRUD y permitir un enfoque orientado a objetos en la interacción con las bases de datos.
- **PostgreSQL**: Sistema de bases de datos relacional usado en cada microservicio para guardar y gestionar la información de manera segura y eficiente.
- **WebSockets**: Protocolo de comunicación bidireccional implementado para establecer conexiones persistentes de baja latencia entre cliente y servidor. Se usó en los microservicios de autenticación, planning poker y canales para habilitar funcionalidades en tiempo real como sincronización de estados de usuario, votaciones colaborativas y mensajería instantánea.
- **Jest**: Framework de pruebas utilizado para implementar pruebas unitarias y de integración automatizadas, garantizando la calidad del código y facilitando la detección temprana de errores.
- **Swagger**: Herramienta implementada para la documentación interactiva de las APIs, facilitando la comprensión y prueba de los endpoints disponibles en cada microservicio.
- **Postman**: Plataforma utilizada durante el desarrollo para probar, documentar y automatizar las pruebas de las APIs, asegurando su correcto funcionamiento antes de la integración.

8.2. Arquitectura del Sistema

La arquitectura del sistema se basa en un enfoque de microservicios, cada uno con una responsabilidad específica. En la Figura 37 se muestra gráficamente como funciona la arquitectura de manera general.

A continuación, se detalla cada microservicio con sus responsabilidades y funcionalidades claves.

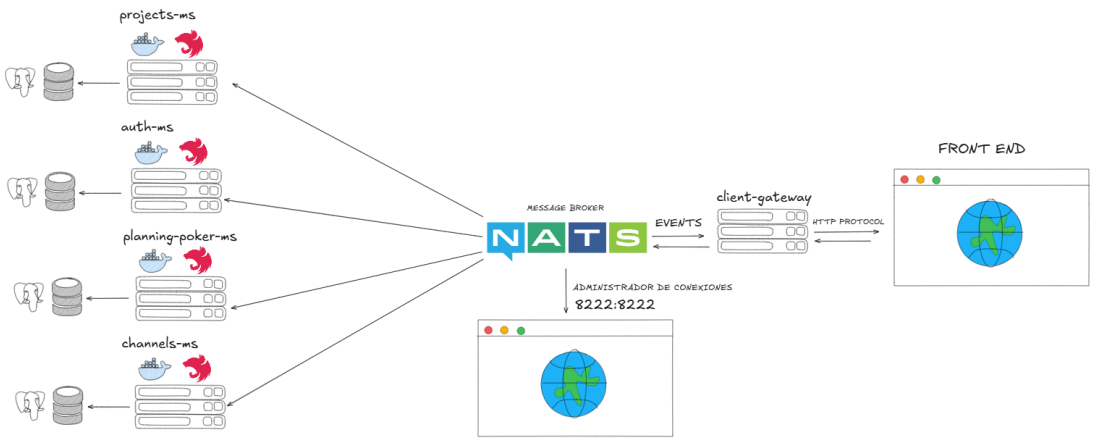

Figura 37: Esquema general de la arquitectura de la aplicación

8.2.1. *Client Gateway*

El Client Gateway es el punto central de entrada que gestiona y enruta las peticiones hacia los microservicios correspondientes, garantizando seguridad, control de acceso y balanceo de carga distribuido. Este componente arquitectónico se encarga de recibir las solicitudes del cliente, aplicar políticas transversales como validación de tokens JWT, para posteriormente distribuirlas al microservicio correspondiente mediante un sistema de prefijos definidos. La implementación está hecha en Nest.js como framework base y Nats.io como sistema de mensajería asíncrona, permitiendo que el gateway construya diferentes módulos y grupos de prefijos (*prefix groups*) para el enrutamiento correcto de peticiones.

A continuación, se detallan los grupos de prefijos que representan a cada microservicio, con sus funcionalidades específicas implementadas:

- **api/backlog**: Gestiona el product backlog, incluyendo la administración de historias de usuario, tareas pendientes y elementos del sprint, tambien incluye funcionalidades de organización de elementos del backlog por prioridad o estado.
- **api/issues**: Manejo de incidencias, incluye la creación, asignación y clasificación por prioridad, tambien maneja la creación de comentarios dentro de las incidencias.
- **api/channels**: Administración de canales de comunicación o colaboración entre equipos, integrando funcionalidades como mensajería o notificaciones.
- **api/epics**: Se encarga de toda la gestión de las épicas para la organización o agrupación de incidencias relacionadas bajo un objetivo común.
- **api/projects**: Gestión de proyectos, incluyendo su creación, configuración, miembros asociados y progreso general.

- **api/sprints**: Administración de los ciclos de trabajo (sprints), con creación de sprints, asignación de incidencias al sprint y obtención de las mismas, también hace registro de los cambios para luego sacar métricas de rendimiento.
- **api/teams**: Módulo para crear y gestionar equipos, asignar roles, visualización de equipos por usuario.
- **api/auth**: Maneja autenticación y autorización de usuarios mediante JWT, incluyendo registro, login, recuperación de cuenta y validación de tokens.
- **api/auth**: Maneja autenticación y autorización de usuarios mediante JWT, incluyendo el registro y verificación de usuarios, inicio de sesión, la protección de rutas y validación de tokens para mantener sesiones activas.
- **api/poker**: Funcionalidad de estimación de tareas mediante la técnica de "Planning Poker", promoviendo la participación del equipo en la planificación. incluye la integración con incidencias del backlog.
- **etc.**: El gateway puede extenderse con más módulos a medida que se agregan nuevos microservicios a la arquitectura.

8.2.2. Microservicio: *Auth-Ms*

El microservicio de autenticación (`auth-ms`) es responsable de gestionar el registro, inicio de sesión y manejo de sesiones dentro de la plataforma, así como la administración completa de equipos. Implementa un esquema de autenticación basado en tokens JWT con mecanismo de refresh token, garantizando la seguridad de sesiones mediante validación criptográfica y expiración temporal controlada, ademas proporciona operaciones de búsqueda y recuperación de perfiles de usuario. La funcionalidad de logout implementa invalidación explícita de tokens a través de limpieza de cookies del lado cliente.

Endpoints de autenticación:

- **POST /auth/register**: Registra un nuevo usuario en el sistema.
- **POST /auth/login**: Autentica al usuario, devuelve y establece cookies `accessToken` y `refreshToken`.
- **POST /auth/logout**: Limpia las cookies del cliente, cerrando la sesión.
- **GET /auth/verify-token**: Verifica que el token de acceso enviado es válido.
- **GET /auth/profile/:id**: Devuelve la información del perfil del usuario con el ID dado.
- **GET /auth/find/:email**: Busca y retorna un usuario por su email.
- **GET /auth/find/user/:id**: Busca y retorna un usuario por su ID.

- **POST /auth/find/:token**: Retorna al usuario vinculado al token recibido.
- **POST /auth/refresh-token**: Genera un nuevo `accessToken` usando un `refreshToken` válido.

Funcionalidades de Gestión de Equipos:
Como parte de una arquitectura modular, el módulo de Teams se encuentra integrado dentro del mismo microservicio de autenticación, lo que permite una mejor gestión de usuarios y sus relaciones con los equipos. Este diseño facilita la validación de permisos y reduce la latencia en operaciones críticas relacionadas con la membresía y los roles. Las integraciones internas del módulo incluyen acceso directo a la información de usuarios para procesos de validación y comunicación con el servicio de correo electrónico para automatizar el envío de invitaciones a nuevos miembros del equipo.

Endpoints de gestión de equipos:

- **POST /teams/create-team**: Crea un nuevo equipo con el usuario autenticado como líder.
- **GET /teams/get-all-teams**: Lista todos los equipos (paginado).
- **GET /teams/get-team/:id**: Obtiene detalles de un equipo específico.
- **PATCH /teams/update-team**: Actualiza información del equipo (nombre, descripción, imagen).
- **DELETE /teams/delete-team/:teamId**: Elimina un equipo completo y sus asociaciones.
- **POST /teams/invite-user/:teamId**: Invita un usuario al equipo mediante correo electrónico.
- **POST /teams/leave-team/:teamId**: Permite a un miembro abandonar voluntariamente un equipo.
- **POST /teams/transfer-ownership/:teamId**: Transfiere el liderazgo a otro miembro del equipo.
- **GET /teams/get-team-by-user/:userId**: Obtiene todos los equipos asociados a un usuario.
- **GET /teams/get-members-by-team/:teamId**: Lista todos los miembros de un equipo específico.
- **POST /teams/expel-member/:teamId**: Permite al líder expulsar a un miembro del equipo.
- **POST /teams/generate-invite-link/:teamId**: Genera un enlace de invitación con token JWT.

- **POST /teams/accept-invite**: Procesa la aceptación de una invitación a un equipo.
- **POST /teams/validate-invite-link**: Valida un token de invitación antes de aceptarlo.

A continuación se explica paso a paso algunos de los flujos principales en la gestión de equipos:

1. Creación de Equipo

1. El usuario autenticado envía una solicitud de creación de equipo al client-gateway.
2. El auth-ms procesa la solicitud y realiza las siguientes operaciones en una transacción atómica:
   - Crea el registro del nuevo equipo en la base de datos.
   - Establece al usuario creador como líder del equipo (`LEADER`).
   - Registra la relación usuario-equipo en la tabla de asociación.
3. Simultáneamente, el auth-ms envía un mensaje a través de NATS al microservicio de canales (channels-ms).
4. El channels-ms recibe la solicitud y crea automáticamente un canal general asociado al nuevo equipo.
5. Una vez completadas todas las operaciones, se confirma la transacción distribuida.
6. El resultado se propaga de vuelta al client-gateway y finalmente al usuario.

El proceso de creación de equipos se entiende mejor en la Figura 38, donde se muestra el flujo completo desde la autenticación del usuario hasta el establecimiento de la relación usuario-equipo.

2. Invitación de Miembros

1. El líder del equipo envía una solicitud de invitación a través del client-gateway, especificando el correo electrónico del usuario a invitar.
2. El client-gateway redirige la petición al microservicio de autenticación (auth-ms) mediante NATS.
3. El auth-ms procesa la solicitud y verifica que el solicitante sea el líder del equipo, luego genera un token JWT firmado con los datos de invitación.
4. El auth-ms utiliza su servicio de correo electrónico integrado para enviar una invitación formal al usuario con un enlace con el token (válido por 24 horas).

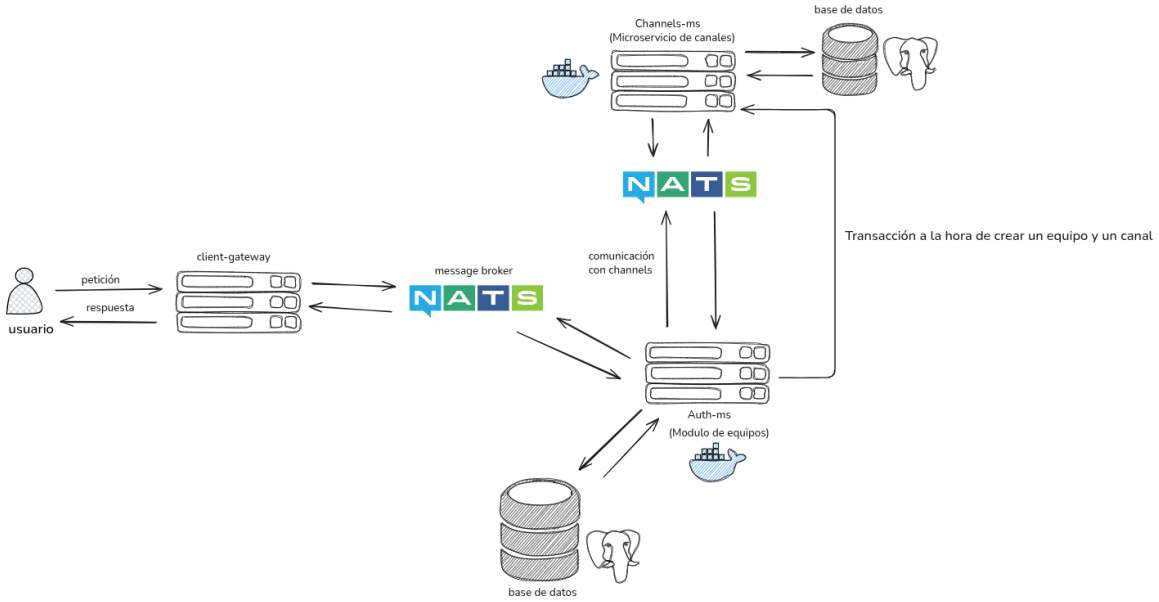

Figura 38: Flujo de creación de equipos

5. Finalmente, El usuario acepta la invitación, se valida el token y se establece la relación. Se envía un enlace con el token por correo electrónico (válido por 24 horas).

En la Figura 39 se explica mejor gráficamente este flujo.

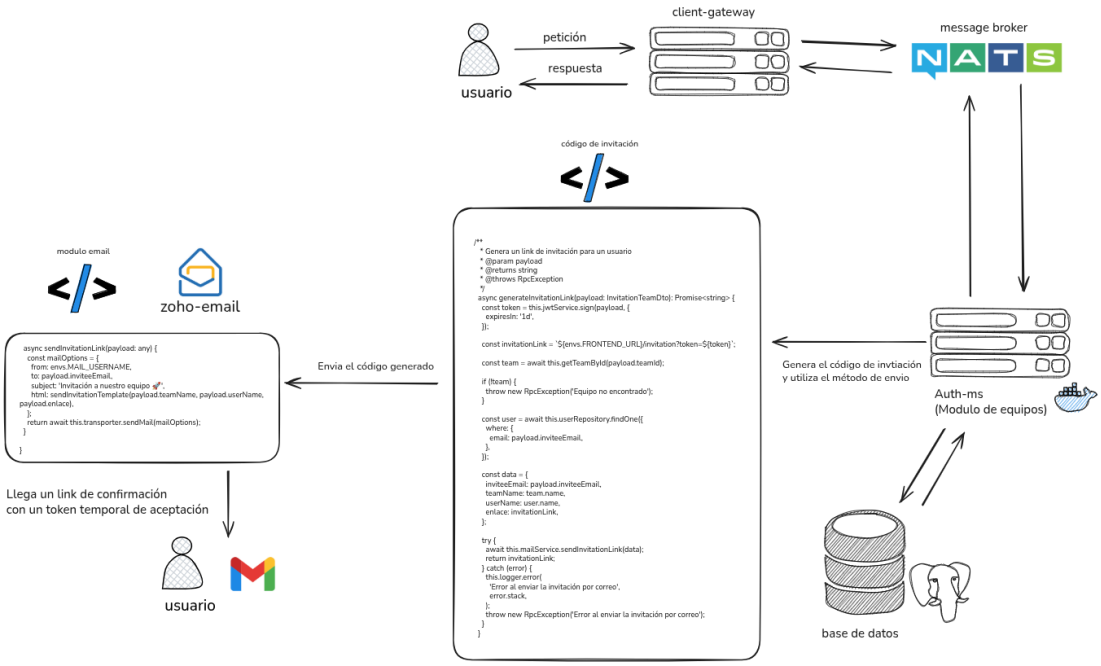

Figura 39: Flujo de invitación a equipos

8.2.3. Microservicio: *Poker-Ms*

El "Poker-ms.[es] un microservicio encargado de gestionar sesiones de estimación ágil. Esto incluye creación de salas, unión de participantes, votación a tiempo real, cálculo de estimaciones finales, historial de votos dentro de una sesión y chats incluidos para discutir las votaciones si es necesario.

Endpoints principales del microservicio:

- **POST /poker/create-session**: Crea una nueva sesión de Planning Poker.
- **GET /poker/all-sessions**: Obtiene todas las sesiones activas.
- **POST /poker/validate-session**: Valida si un usuario puede unirse a una sesión.
- **POST /poker/join-session**: Permite a un usuario unirse a una sesión existente.
- **POST /poker/join-session-code**: Unión a sesión mediante código.
- **POST /poker/magic-link**: Acceso mediante enlace mágico.
- **POST /poker/ai-estimation**: Obtiene una estimación automatizada para una historia de usuario.
- **GET /poker/session-details**: Obtiene detalles de una sesión específica.
- **GET /poker/stories**: Endpoint de ejemplo con historias de usuario.

Interacción en Tiempo Real mediante WebSockets

En este microservicio se usan WebSockets de una manera robusta. Esto se debe a que en las sesiones de Planning Poker se requiere de comunicación e interacción en tiempo real constante y compleja. Esta implementación debe soportar múltiples capas de sincronización simultánea. Esto incluye la sincronización de temporizadores compartidos entre todos los participantes, garantizando que todos los usuarios visualicen el mismo contador de tiempo de manera coherente, así como la visualización instantánea de los votos de otros usuarios, ver los usuarios que están dentro de la sala y la implementación de un chat que permite la comunicación instantánea durante las sesiones de estimación.

A continuación se describen los eventos principales que utilizan la comunicación WebSocket para gestionar las funcionalidades de la sesión de Planning Poker:

- **Eventos principales:**
  - `join-room`: Unirse a una sala de votación
  - `submit-vote`: Enviar una votación
  - `complete-voting`: Finalizar la votación actual
  - `next-story`: Pasar a la siguiente historia

- `send-message`: Enviar mensaje al chat
- `start-timer`: Iniciar temporizador
- `end-session`: Finalizar la sesión

- **Flujo de una sesion de Planning Poker:**

  1. Los participantes se unen a la sala mediante `join-room`
  2. El líder inicia la sesión (`start-session`)
  3. Los participantes votan (`submit-vote`)
  4. Cuando todos han votado, se calcula el promedio (`complete-voting`). si el promedio no es un valor de la serie fibonacci, el lider de la sala puede volver a iniciar la votación para tener un consenso.
  5. El líder puede avanzar a la siguiente historia (`next-story`)
  6. Finalmente se cierra la sesión (`end-session`)

  En la Figura 40 se ilustra este flujo de comunicación por WebSockets en las sesiones de Planning Poker.

- **Características:**

  Algunas características que tienen las salas de planning poker son:

  - Temporizador configurable para votaciones
  - Chat en tiempo real entre participantes
  - Modificación manual del líder para forzar consenso
  - Historial completo de votaciones
  - Validación de números Fibonacci para votaciones
  - Sugerencias utilizando Inteligencia Artificial

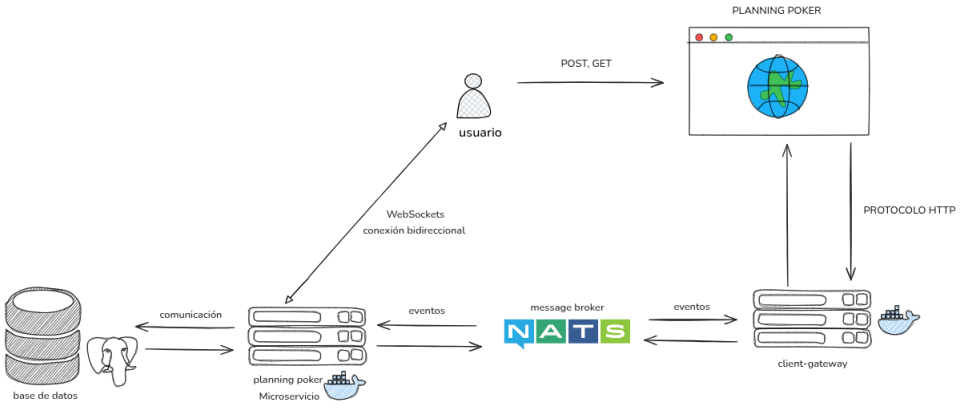

Figura 40: Flujo de comunicación WebSocket en Planning Poker

En la Figura 41 se ilustra las conexiones que se hacen con el microservicio de projects-ms y auth-ms, estas conexiones se hacen cuando se quiere crear una sala para obtener las incidencias que se quieren agregar para ser votadas.

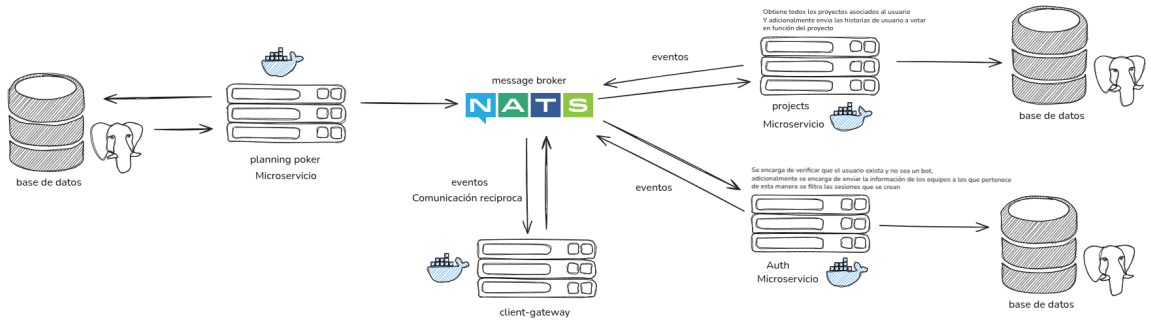

Figura 41: Flujo de comunicación del Poker-ms con otros microservicios a la hora de crear una sala.

8.2.4. Microservicio: *Projects-Ms*

El microservicio de Proyectos (`projects-ms`) es el núcleo central para la gestión del ciclo de vida completo de los proyectos dentro de la plataforma. implementa una arquitectura modular que integra diversas entidades fundamentales como proyectos, product backlogs, sprint backlogs, épicas e incidencias, proporcionando una solución integral para la metodología ágil de desarrollo.

Funcionalidades Principales:

Este microservicio implementa un conjunto completo de operaciones para administrar proyectos de desarrollo de software, facilitando la planificación, seguimiento y gestión de tareas durante todo el ciclo de desarrollo. Su diseño modular permite una clara separación de responsabilidades entre sus diferentes componentes.

Integración con otros microservicios:

- Interactúa con `auth-ms` para verificación de permisos de usuarios y equipos
- Coordina con `poker-ms` para integrar las estimaciones realizadas
- Utiliza `mail-service` para el envío de invitaciones y notificaciones relacionadas con proyectos

Estructura Modular

El microservicio se organiza en cinco módulos principales que trabajan de manera coordinada para proporcionar una gestión completa de proyectos ágiles:

- **Módulo de Proyectos:** Gestiona la creación, eliminación, configuración y la asignación de miembros.
- **Módulo de Product Backlog:** Administra el repositorio de incidencias, permitiendo priorizar y gestionar las tareas pendientes para el desarrollo del producto.
- **Módulo de Sprint Backlog:** Coordina la planificación y ejecución de sprints, facilitando la selección de historias del product backlog y el seguimiento de su progreso.
- **Módulo de Issues:** Proporciona funcionalidades para la creación, actualización y seguimiento de incidencias, incluyendo la gestión de comentarios y categorización.
- **Módulo de Métricas:** Recopila y procesa datos sobre el rendimiento del proyecto, generando métricas relevantes para la toma de decisiones y la mejora continua.

Endpoints principales del microservicio:

- **Gestión de Proyectos:**
  - **POST /projects/create**: Crea un nuevo proyecto con su configuración inicial.
  - **PATCH /projects/update/:id**: Actualiza la información y configuración de un proyecto existente.
  - **GET /projects/find/:id**: Obtiene los detalles de un proyecto específico.
  - **GET /projects/findAllByUser?userId={userId}**: Obtiene todos los proyectos asociados a un usuario específico.
  - **POST /projects/invite-member**: Procesa la invitación de un nuevo miembro a un proyecto.
  - **GET /projects/members/:projectId**: Obtiene la lista de miembros de un proyecto con paginación.
  - **GET /projects/team-members/unassigned**: Obtiene los miembros de un equipo que no están asignados al proyecto.
- **Gestión de Product Backlog:**
  - **POST /backlog/add-issue/:id**: Agrega una nueva incidencia al product backlog.
  - **GET /backlog/get-all-issues/:id**: Obtiene todas las incidencias de un backlog con opción de filtrado.
  - **GET /backlog/get-backlog/:id**: Recupera los detalles de un product backlog específico.
  - **GET /backlog/get-backlog-by-project/:id**: Obtiene el product backlog asociado a un proyecto.
  - **POST /backlog/move-issue-to-sprint**: Transfiere una incidencia del product backlog al sprint backlog.

- **POST /backlog/move-issue-to-backlog**: Devuelve una incidencia al product backlog.

- **Gestión de Sprints:**
  - **POST /sprints/create**: Crea un nuevo sprint dentro del proyecto.
  - **POST /sprints/start/:id**: Inicia formalmente un sprint previamente configurado.
  - **POST /sprints/complete/:id**: Finaliza un sprint activo y procesa las métricas asociadas.
  - **GET /sprints/project/:projectId**: Obtiene todos los sprints asociados a un proyecto.
  - **GET /sprints/backlog/:sprintId**: Recupera las incidencias asignadas a un sprint específico.

- **Gestión de Issues y Épicas:**
  - **POST /issues/create**: Crea una nueva incidencia.
  - **PATCH /issues/update/:id**: Actualiza la información de una incidencia existente.
  - **GET /issues/by-user/:userId**: Recupera las incidencias asignadas a un usuario específico.
  - **GET /issues/comments/:issueId**: Obtiene los comentarios asociados a una incidencia.
  - **POST /issues/add-comment/:issueId**: Agrega un nuevo comentario a una incidencia.
  - **POST /epics/create**: Crea una nueva épica en el proyecto.
  - **GET /issues/by-epic/:epicId**: Obtiene todas las issues asociadas a una épica.
  - **GET /epics/by-backlog/:backlogId**: Obtiene las épicas asociadas a un product backlog específico.

Flujos Clave del Microservicio

El microservicio Projects-Ms implementa varios flujos de trabajo que permiten la gestión ágil de proyectos siguiendo la metodología Scrum. A continuación, se detallan los principales flujos con sus características e interacciones:

Gestión del Ciclo de Vida Completo del Sprint: El ciclo de vida de un sprint es uno de los procesos fundamentales de la metodología ágil implementada en el microservicio, abarcando desde la inicialización hasta la transición entre iteraciones. El proceso se fundamenta en transacciones atómicas que garantizan consistencia de datos y trazabilidad

completa de operaciones.

Cada sprint tiene asociado un tablero de sprint backlog, donde se organizan las incidencias en tres estados principales (Por hacer, En progreso, Terminado).

La inicialización del sprint, ilustrada en la Figura 42,incluye la configuración inicial y la preparación del tablero de sprint backlog.

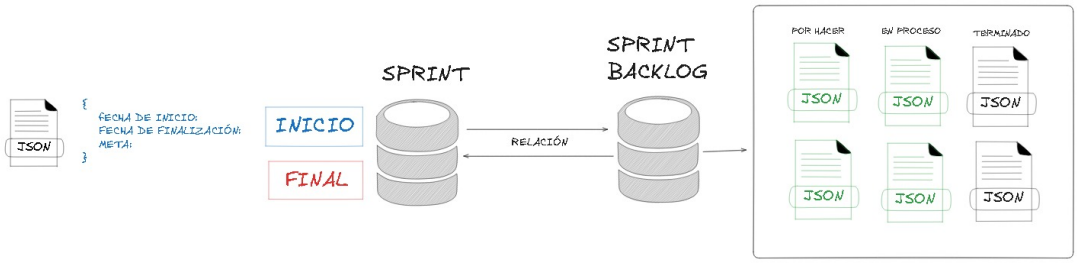

Figura 42: Flujo de inicialización del sprint

La finalización de un sprint y la inicialización del siguiente es uno de los procesos clave en la metodología ágil. Como se observa en la Figura 43, cuando se finaliza un sprint las incidencias que no se completaron se envían automáticamente al nuevo sprint.

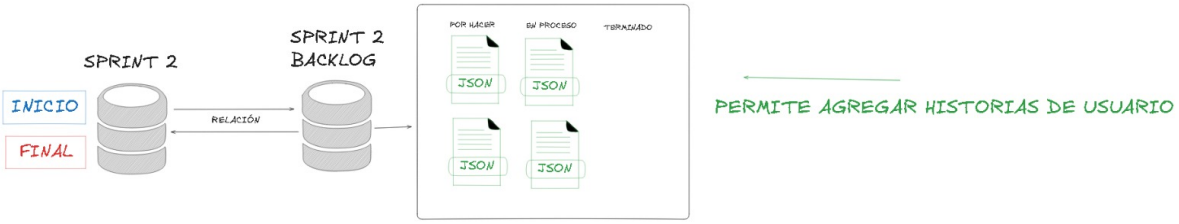

Figura 43: Finalización y transición de incidencias restantes entre sprints

Gestión Bidireccional de Incidencias en Sprint Backlog: El microservicio implementa procesos de transferencia bidireccional de incidencias entre product backlog y sprint backlog durante las fases de planificación y ejecución del sprint. Como se ilustra en las Figuras 44 y 45, ambos flujos operan mediante solicitudes JSON transmitidas a través del client-gateway al microservicio de proyectos, donde se actualizan las relaciones correspondientes en la base de datos, confirmando la operación con código de respuesta 200.

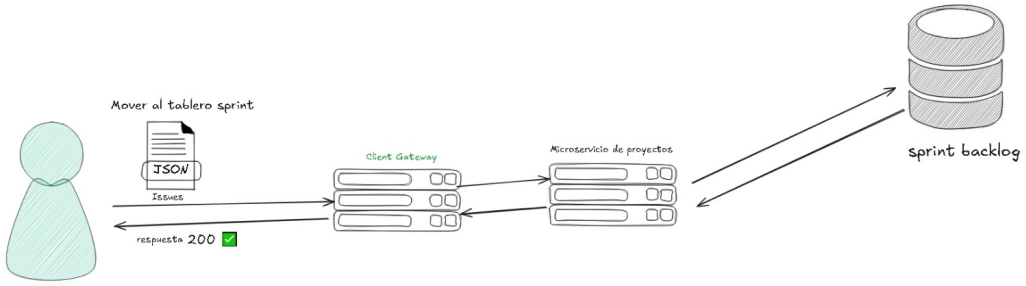

Figura 44: Flujo de movimiento de incidencias al sprint backlog

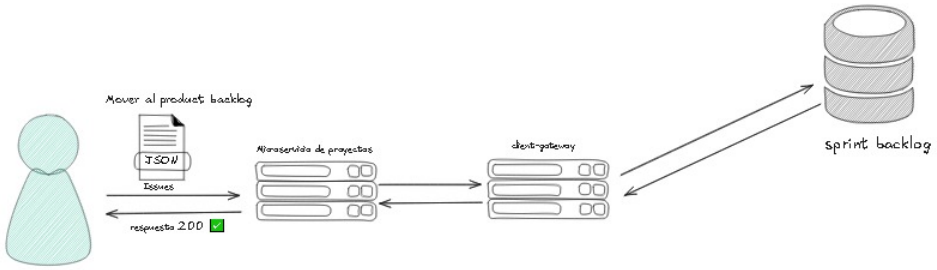

Figura 45: Flujo de movimiento de incidencias al product backlog

8.2.5. Microservicio: *Channels-Ms*

El microservicio de canales (`channels-ms`) es responsable de gestionar todos los aspectos relacionados con la comunicación entre miembros de equipos dentro de la plataforma, implementando una estructura jerárquica de canales de comunicación con capacidades de mensajería en tiempo real. Este componente proporciona una infraestructura robusta para la colaboración, permitiendo la creación de canales generales y subcanales especializados, cada uno con su propio chat asociado.

Endpoints principales del microservicio:

- **POST /channels/create-channel**: Crea un nuevo canal asociado a un equipo específico, con su chat correspondiente.
- **POST /channels/create-child-channel**: Crea un subcanal dentro de un canal existente, permitiendo organizar conversaciones por temas específicos.
- **GET /channels/load-channels**: Obtiene todos los canales disponibles para un equipo determinado, incluyendo información sobre los miembros activos.

- **GET /channels/load-messages/:channel_id**: Recupera el historial de mensajes para un canal específico.

Se realizó una implementación de WebSockets mediante Socket.IO para la comunicación en tiempo real entre usuarios conectados.A continuación se muestran algunos de los eventos que se emiten mediante estos Sockets.

Eventos de WebSocket:

- **join**: Permite a un usuario unirse a un canal específico, cargando automáticamente el historial de mensajes.
- **message**: Procesa y distribuye nuevos mensajes a todos los participantes conectados en un canal.
- **participants**: Actualiza y notifica el estado de conexión de los usuarios dentro de un canal.
- **leave**: Gestiona la salida de un usuario de un canal, actualizando el estado de presencia.
- **loadMessages**: Solicita la carga del historial completo de mensajes de un canal.

A continuación se describen los flujos principales de este microservicio:

1. Creación de Canal

El proceso de creación de canales es fundamental para establecer espacios de comunicación dentro de los equipos:

1. Un usuario autenticado envía una solicitud de creación de canal al client-gateway, especificando el equipo al que estará asociado.
2. El client-gateway redirige la petición al microservicio de canales utilizando NATS.
3. El channels-ms procesa la solicitud y realiza las siguientes operaciones en una transacción atómica:
    - Crea el registro del nuevo canal en la base de datos.
    - Inicializa un chat asociado al canal creado.
    - Establece las relaciones entre canal, equipo y usuario creador.
4. Si la solicitud incluye un canal padre (para crear un subcanal), se verifica su existencia y se establece la relación jerárquica correspondiente.
5. Una vez completadas todas las operaciones, se confirma la transacción y se devuelve la información del canal creado.

2. Comunicación en Tiempo Real

La funcionalidad de mensajería instantánea se implementa mediante un sofisticado sistema de WebSockets:

1. Cuando un usuario se une a un canal, emite un evento `join` con el identificador del canal.
2. El sistema inicializa los mapas de participantes para el canal si es necesario y añade al usuario a la sala correspondiente.
3. Se recupera y envía automáticamente el historial de mensajes al cliente recién conectado.
4. El sistema notifica a todos los participantes sobre el nuevo usuario conectado.
5. Cuando un usuario envía un mensaje, este se persiste en la base de datos y se distribuye en tiempo real a todos los usuarios conectados al canal.
6. El sistema mantiene registro de usuarios conectados y desconectados, actualizando el estado de presencia para todos los participantes.

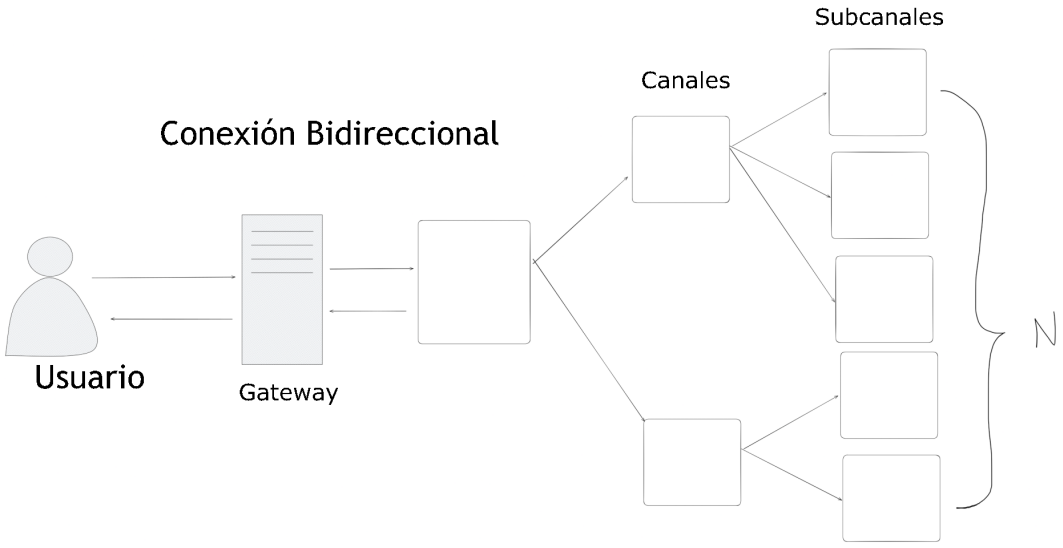

Figura 46: Flujo de comunicación del usuario con los canales

3. Integración con el Servicio de Equipos: El microservicio de canales se integra con el servicio de equipos para proporcionar una experiencia coherente. Al solicitar la lista de canales para un equipo, el microservicio realiza consultas al servicio de autenticación para obtener información detallada sobre los miembros del equipo y sus perfiles. Esta información se combina con los datos de canales para construir una respuesta completa que incluye metadatos sobre participantes. La respuesta permite a la interfaz de usuario mostrar tanto la estructura de canales como el estado de presencia de los usuarios.

8.3. Documentación de la API (Swagger)

Para facilitar el consumo y comprensión de los servicios REST, se generó documentación automática utilizando Swagger. Esta herramienta permite explorar los endpoints de la API, visualizar los parámetros requeridos, los posibles códigos de respuesta, así como realizar pruebas directamente desde el navegador.

Swagger permite validar rápidamente los servicios implementados, especialmente durante el desarrollo y pruebas de integración. Además, mejora la comunicación entre el backend y el frontend, al proveer una fuente centralizada de referencia para los endpoints disponibles.

A continuación, se muestran algunas capturas específicas de distintos endpoints:

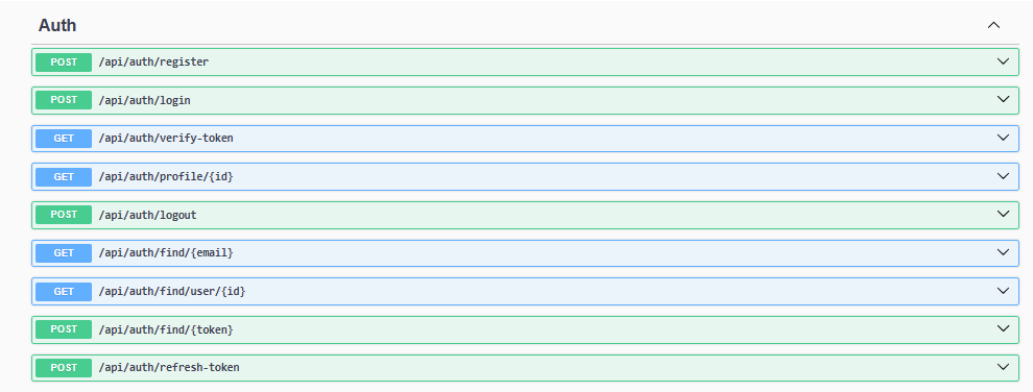

Figura 47: Endpoints del microservicio de autenticación

Figura 48: Endpoints para el microservicio de plannning poker

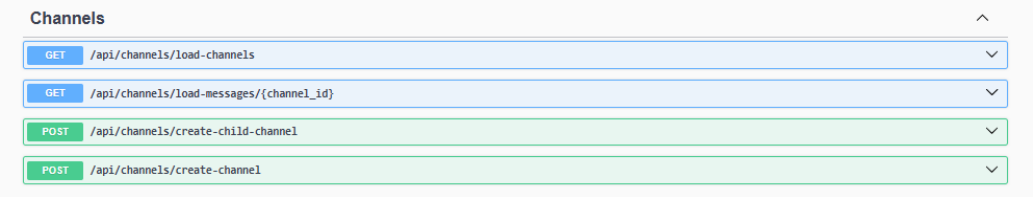

Figura 49: Endpoints del microservicio de chat

Figura 50: Endpoints para el modulo de equipos dentro del microservicio auth-ms

8.4. Proceso de Desarrollo

El desarrollo se hizo bajo el marco de trabajo SCRUM con sprints de dos semanas.Durante todo el proceso se usó Jira para la gestión y asignación de historias de usuario, lo que permitió mantener un seguimiento claro del progreso de cada funcionalidad. Las historias se definieron siguiendo las mejores prácticas ágiles y se asignaron según las fortalezas de los miembros del equipo y la carga de trabajo de cada sprint. Para el control de versiones se implementó un flujo de trabajo flexible con Git, donde se creaban ramas separadas cuando se trabajaba en la misma funcionalidad o cuando los cambios podrían afectar el trabajo del compañero, pero se trabajó directamente en la rama ”main”para modificaciones que cada uno podía hacer de forma independiente sin interferir con lo que estaba desarrollando el otro. La documentación completa de las historias de usuario implementadas se encuentra disponible en el Anexo 12.

Fase 1: Configuración Inicial y diseño de UI

- Configuración de repositorios dentro de una organización en GitHub para mas centralización.
- Configuración inicial de microservicios con NestJS.
- Diseño de wireframes y mockups utilizando Figma y Sora.
- Diseño del logo de la aplicación. (ver Figura 22)
- Configurar el Message Broker (NATS.io) con Docker para comunicación de los microservicios.
- Diseño de esquemas y diagramas MER de cada uno de los microservicios.

Fase 2: Implementación del Frontend

- Desarrollo de componentes reutilizables para la interfaz de usuario.
- Creación de diversos componentes y hooks utilizando Shadcn.
- Implementación de AuthProvider para la verificación, autenticación y cierre de sesión del usuario.
- Implementación de websockets y SocketProvider para la comunicación en tiempo real entre cliente y servidor.
- Implementación completa de la funcionalidad de los módulos del frontend, siguiendo el siguiente orden:
    1. Diseño e implementación de la landing page.
    2. Desarrollo del módulo de autenticación.
    3. Implementación del módulo de Planning Poker.
    4. Creación del módulo de gestión de equipos.
    5. Desarrollo del módulo de chat.
    6. Implementación del módulo de gestión de proyectos.
    7. Implementación de tableros Kanban interactivos (ver Figura 24)
    8. Desarrollo del módulo de analíticas.
- Integración con servicios backend mediante Axios

Fase 3: Pruebas

- Se implementaron pruebas unitarias en cada microservicio para verificar que todos los módulos funcionaran correctamente de manera individual.
- Se realizaron pruebas de integración entre microservicios para validar la comunicación y el funcionamiento conjunto del sistema.
- Se ejecutaron pruebas con usuarios finales, donde se solicitó a varias personas que probaran la aplicación y se aplicó una encuesta para recopilar sus opiniones y evaluar la experiencia de usuario.

Fase 4: Despliegue

- Configuración inicial de contenedores Docker para cada microservicio.
- Primer intento de despliegue utilizando Azure Kubernetes Service (AKS) con orquestación completa.
- Migración a Azure Container Apps (Web Containers) como solución alternativa más directa.
- Configuración de variables de entorno y conexiones entre servicios en el nuevo entorno.
- Implementación exitosa en producción con todos los microservicios funcionando correctamente.

8.5. Despliegue

El proceso de despliegue presentó varios desafíos que requirieron adaptaciones en la estrategia inicial. Se comenzó configurando contenedores Docker individuales para cada microservicio y posteriormente con el despliegue en Azure Kubernetes Service (AKS), que inicialmente parecía la opción más robusta para la orquestación de servicios.

Sin embargo, durante la implementación en AKS surgieron problemas de autorización que impedían el acceso correcto a los endpoints de los microservicios. A pesar de varios intentos de configuración, no se logró resolver estos errores de autorización. Estos inconvenientes hizo evaluar alternativas, considerando inicialmente una migración a Google Cloud Platform donde se realizaron pruebas exitosas. No obstante, al analizar las opciones disponibles, se identificó que Azure ofrecía una mejor capa gratuita para estudiantes, por lo que se optó por permanecer con dicha plataforma.

Se decidió implementar Azure Container Apps (Web Containers), una alternativa más directa y menos compleja que AKS para las necesidades específicas del proyecto. Asimismo, se implementaron Azure Pipelines para automatizar el proceso de integración y entrega

continuas (CI/CD), lo que incluyó la construcción de imágenes Docker y el despliegue automático hacia los Container Apps. Además, el frontend de la aplicación fue desplegado en Cloudflare Pages, plataforma seleccionada por ser gratuita y ofrecer múltiples beneficios, como la integración sencilla con flujos de CI/CD. Esta configuración permitió establecer un flujo de trabajo eficiente y facilitar el proceso de despliegue.

Configuración de Producción

La implementación final integra las siguientes configuraciones:

Cloudflare Pages: Despliegue automático desde Git con CDN global, optimización de recursos.

Azure Container Apps: Variables de entorno centralizadas, networking privado entre microservicios, escalado automático basado en carga, y balanceamiento de carga automático.

Azure Pipelines: Trigger automático por cada push, construcción de imágenes Docker, despliegue automático tras validación exitosa, y verificación del estado de los servicios.

En la Figura 51 se muestra la forma en que se comunican los distintos contenedores de la aplicación desplegada en la plataforma Azure.

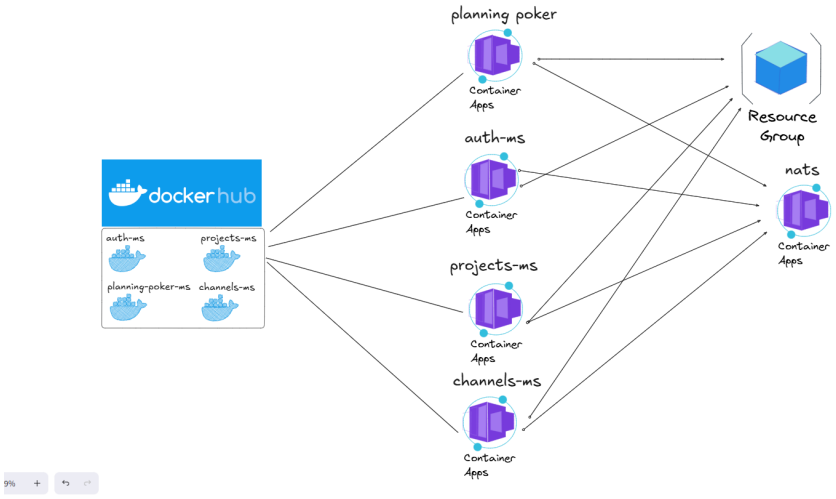

Figura 51: Comunicación de los app containers en Azure

8.6. Desafíos Técnicos

Durante el desarrollo se presentaron varios retos:

- Fue necesario aprender algunas tecnologías desde cero.
- **Migración de Next.js a React + Vite**: Inicialmente se trabajó con Next.js, pero se identificaron problemas de rendimiento en el renderizado, debido a que este framework utiliza por defecto el renderizado del lado del servidor (SSR) y esto es un problema en una aplicación como TaskMate en la que hay mucha interactividad y constante comunicación entre el backend. Despues se decidió usar React que usa CSR (Client Side Rendering) por defecto.
- Tal como se indicó anteriormente, se presentó dificultad al seleccionar la mejor alternativa para el despliegue del backend. Aunque Azure se consideró como la opción más adecuada, se encontraron inconvenientes al utilizar Azure Kubernetes Service (AKS), ya que el despliegue inicial generaba errores de autorización al probar los endpoints. Finalmente, se realizó el despliegue utilizando Azure Container Apps, solución que no presentó dichos inconvenientes.

La fase de implementación y despliegue validó varias decisiones técnicas fundamentales y proporcionó aprendizajes valiosos. El proyecto demostró la importancia de la flexibilidad en las decisiones tecnológicas. Los principales desafíos estuvieron relacionados con la sincronización en tiempo real mediante WebSockets y la complejidad inicial del despliegue. Sin embargo, cada obstáculo se resolvió satisfactoriamente mediante investigación, experimentación y adaptación de la estrategia inicial.

9. Pruebas Funcionales

La validación exhaustiva de sistemas distribuidos basados en microservicios requiere un enfoque metodológico riguroso que garantice tanto la funcionalidad individual de cada componente como la integridad de las comunicaciones inter-servicios. En el contexto de este proyecto, se implementó una estrategia de *testing* multicapa que abarca desde pruebas unitarias granulares hasta validaciones de integración, con el objetivo de asegurar la calidad, confiabilidad y mantenibilidad del sistema desarrollado.

La arquitectura del sistema está compuesta por cuatro microservicios especializados: **Authentication & Teams (Auth), Projects Management, Planning Poker y Channels**, cada uno diseñado siguiendo los principios de responsabilidad única y desacoplamiento. Adicionalmente, el **Client Gateway** actúa como punto de entrada unificado y orquesta las comunicaciones entre el frontend y los microservicios mediante el protocolo NATS.

9.0.1. Estrategia de Testing Implementada

- **Cobertura integral:** Cada microservicio cuenta con pruebas unitarias que validan la lógica de negocio individual, y pruebas de integración que verifican las interacciones con bases de datos, servicios externos y otros microservicios.
- **Aislamiento de componentes:** Mediante el uso de *mocks* y *stubs*, las pruebas unitarias aseguran que cada función o método se comporte correctamente independientemente de sus dependencias externas.
- **Validación de contratos:** Las pruebas de integración garantizan que los contratos de comunicación entre microservicios se mantengan estables y funcionales a lo largo del tiempo.

9.0.2. Herramientas y Framework de Testing

Para la implementación de las pruebas se utilizó **Jest**, el framework de testing oficial recomendado por NestJS, el cual proporciona:

- Capacidad de *mocking*.
- Reportes detallados de cobertura de código.
- Soporte nativo para pruebas asíncronas y manejo de promesas.
- Integración con TypeScript y los decoradores propios de NestJS.

La configuración de testing incluye bases de datos de prueba aisladas, servicios simulados para dependencias externas.

9.1. Microservicio: Authentication & Teams

El microservicio **Authentication & Teams** constituye el núcleo de seguridad del sistema. Es responsable de la gestión de usuarios, autenticación mediante JWT, autorización basada en roles y administración de equipos de trabajo. Su arquitectura incluye entidades con relaciones *many-to-many*, así como servicios para el manejo de tokens, encriptación de contraseñas y envío de notificaciones por correo electrónico.

Prueba Unitaria: Método `createUser`

Esta prueba unitaria valida el correcto funcionamiento del método `createUser` dentro del microservicio **Authentication & Teams**. Dicha función es responsable de registrar un nuevo usuario, crear su perfil asociado, asignarle un rol por defecto y enviarle un correo de bienvenida.

- **Objetivo:** Verificar que el método cree correctamente un usuario, su perfil, y que se notifique al correo indicado.
- **Tipo de prueba:** Unitaria.
- **Herramientas utilizadas:** `Jest`, `TypeORM`, `bcrypt`, y `nestjs/testing`.

Flujo validado en la prueba:

1. Simulación del proceso de encriptación de contraseña utilizando un *mock* de `bcrypt.hash`.
2. Creación y guardado del usuario en la base de datos a través de un repositorio simulado.
3. Creación de un perfil asociado al usuario.
4. Obtención del rol por defecto (User) desde la tabla de roles y creación de la relación en `UsersRole`.
5. Envío de un correo de bienvenida mediante el servicio `Mail`, también simulado.
6. Retorno de los datos básicos del usuario creado.

Código de la prueba:

```
describe('createUser', () => {
  it('should create a user and return basic info', async () => {
    (bcrypt.hash as jest.MockedFunction<typeof bcrypt.hash>) = jest.fn().mockResolvedValue('hashed' as never);

    jest.spyOn(userRepo, 'create').mockReturnValue(mockUser as any);
    jest.spyOn(userRepo, 'save').mockResolvedValueOnce(mockUser as any);
    jest.spyOn(profileRepo, 'create').mockReturnValue({ ...mockUser, userId: mockUser.id } as any);
    jest.spyOn(profileRepo, 'save').mockResolvedValueOnce({} as any);
    jest.spyOn(roleRepo, 'findOne').mockResolvedValueOnce({ id: 'role-id', role: RoleEnum.User } as any);
    jest.spyOn(userRoleRepo, 'create').mockReturnValue({} as any);
    jest.spyOn(userRoleRepo, 'save').mockResolvedValueOnce({} as any);

    const dto = {
      name: 'Test',
      lastName: 'User',
      email: 'test@test.com',
      password: '123456',
      company: 'TestCorp',
    };

    const result = await service.createUser(dto as any);

    expect(result.user).toBeDefined();
    expect(result.user.email).toBe(dto.email);
    expect(mockMailService.sendGreetingsToUser).toHaveBeenCalled();
  });
});
```

Figura 52: Código del método de prueba createUser

Prueba Unitaria: Método `login`

La función `login` permite autenticar a un usuario en el sistema y retornar un par de tokens JWT: uno de acceso y otro de refresco. Esta prueba unitaria valida que el proceso de inicio de sesión funcione correctamente cuando las credenciales proporcionadas son válidas.

- **Objetivo:** Verificar que se autentique al usuario y se generen los tokens correctamente.
- **Tipo de prueba:** Unitaria.
- **Herramientas utilizadas:** `Jest`, `bcrypt`, `JwtService`.

Flujo validado en la prueba:

1. Se mockea la contraseña almacenada utilizando `bcrypt.hash`.
2. Se simula una búsqueda en la base de datos del usuario mediante `userRepo.findOne`.
3. Se simula la comparación entre la contraseña almacenada y la ingresada mediante `bcrypt.compare`.
4. Se llama al servicio de autenticación con el DTO correspondiente.
5. Se validan los tokens retornados y la presencia de datos en la respuesta.

Código de la prueba:

```
describe('login', () => {
  it('should login and return tokens', async () => {
    const dto = { email: mockUser.email, password: '123456' };
    const hashed = await bcrypt.hash(dto.password, 10);

    jest.spyOn(userRepo, 'findOne').mockResolvedValueOnce({ ...mockUser, password: hashed } as any);

    (bcrypt.compare as jest.MockedFunction<typeof bcrypt.compare>) = jest.fn().mockResolvedValue(true as never);

    const result = await service.login(dto);

    expect(result.data).toBeDefined();
    expect(result.accessToken).toBe('mockToken');
    expect(result.refreshToken).toBe('mockToken');
  });
});
```

Figura 53: Prueba unitaria del método `login` en el microservicio de autenticación.

Resultados esperados:

- La respuesta incluye los tokens `accessToken` y `refreshToken`.
- La propiedad `data` del resultado está definida.
- Las funciones `hash` y `compare` se comportan según lo esperado.

Prueba Unitaria: Método `updateProfile`

El método `updateProfile` permite modificar los datos personales de un usuario y su perfil. Esta prueba unitaria asegura que tanto los campos del usuario como los del perfil se actualicen correctamente en la base de datos.

- **Objetivo:** Verificar que la actualización del perfil y datos del usuario se realice exitosamente.
- **Tipo de prueba:** Unitaria.
- **Herramientas utilizadas:** `Jest`, `TypeORM`.

Flujo validado en la prueba:

1. Se define un DTO con nuevos valores para actualizar el perfil.
2. Se simula la búsqueda del usuario por ID mediante `userRepo.findOne`.
3. Se simula el guardado del nuevo estado del usuario y perfil.
4. Se ejecuta el método `updateProfile` con los datos simulados.
5. Se verifica que el mensaje de éxito se retorne correctamente.

Código de la prueba:

```
describe('updateProfile', () => {
  it('should update profile and user fields', async () => {
    const updateDto = {
      userId: 'user-id',
      bio: 'New bio',
      company: 'UpdatedCorp',
      phone: '123456789',
    };

    const profile = { bio: '', updatedAt: new Date() };
    const user = { ...mockUser, profile };

    jest.spyOn(userRepo, 'findOne').mockResolvedValueOnce(user as any);
    jest.spyOn(userRepo, 'save').mockResolvedValueOnce(user as any);
    jest.spyOn(profileRepo, 'save').mockResolvedValueOnce(profile as any);

    const result = await service.updateProfile(updateDto as any);

    expect(result.message).toBe('Profile updated successfully');
  });
});
```

Figura 54: Prueba unitaria del método `updateProfile` en el microservicio de autenticación.

Resultados esperados:

- El mensaje retornado debe indicar que el perfil se actualizó con éxito.
- Se debe invocar el guardado tanto del usuario como del perfil.
- Los valores actualizados deben coincidir con los proporcionados en el DTO.

Pruebas Unitarias del Módulo Teams

El módulo **Teams**, integrado en el microservicio **Authentication & Teams**, gestiona la creación de equipos, asignación de roles, y la invitación de usuarios a equipos específicos. Las pruebas unitarias desarrolladas garantizan el correcto funcionamiento de estas funcionalidades críticas, simulando repositorios, servicios de mensajería y dependencias externas como el correo electrónico.

Herramientas utilizadas:

- `Jest` para pruebas unitarias.
- `@nestjs/testing` para inyección de dependencias.
- Repositorios simulados de `TypeORM`.
- Servicio `NATS` simulado con `ClientProxy`.
- Servicio `Mail` simulado.

Casos de prueba cubiertos:

- **createTeam:**
  - Rechaza la operación si el líder especificado no existe.
  - Crea un equipo y asigna al líder como miembro con el rol `LEADER`.
  - Emite un evento al canal `channel.create.channel` para crear el canal del equipo.
- **inviteUserToTeam:**
  - Rechaza la invitación si el equipo no existe.
  - Valida que solo el líder pueda invitar usuarios.
  - Verifica que el usuario no sea ya miembro del equipo.
  - Registra correctamente la invitación como una nueva relación `UsersTeam`.

Resultados esperados:

- Validación completa de roles y pertenencia a equipos.
- Integridad en la creación de relaciones entre usuarios y equipos.
- Aislamiento total de dependencias externas mediante mocks.
- Correcto disparo de eventos hacia otros microservicios mediante NATS.

Resultados Generales de las Pruebas Unitarias del Microservicio Auth y Teams

Las pruebas unitarias desarrolladas para el microservicio **Authentication & Teams** abarcan casos críticos relacionados con el registro de usuarios, autenticación (login) y actualización de perfiles. Estas pruebas buscan asegurar la robustez y el correcto comportamiento de los servicios clave en escenarios aislados y controlados mediante el uso de *mocks* y *spies*.

En la siguiente imagen se muestra la ejecución de los test utilizando el framework `Jest`. Puede observarse que se han definido y ejecutado exitosamente un total de 9 pruebas distribuidas en 2 suites de test, sin errores ni fallos:

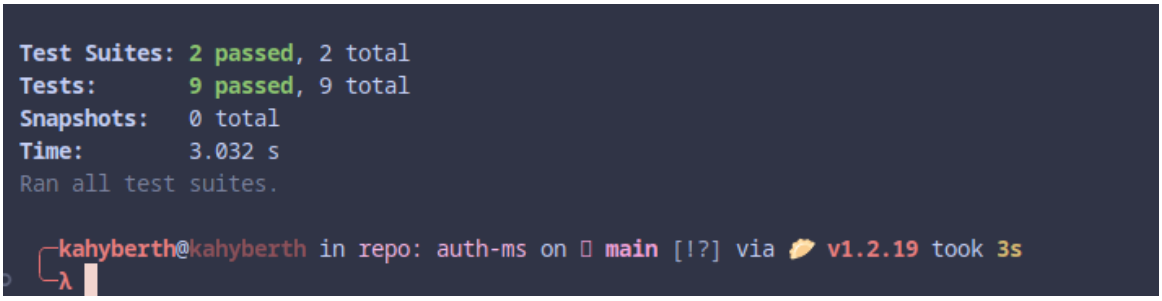

Figura 55: Resultado global de las pruebas unitarias ejecutadas en el microservicio Auth utilizando Jest.

Resumen de la salida:

- **Test Suites:** Se ejecutaron 2 suites de pruebas, ambas exitosas.
- **Tests Totales:** 9 pruebas unitarias pasaron correctamente.
- **Snapshots:** No se utilizaron pruebas de tipo snapshot.
- **Tiempo de ejecución:** El tiempo total fue de aproximadamente 3 segundos.

Pruebas End-to-End del Microservicio Authentication & Teams

Las pruebas end-to-end (E2E) desarrolladas para el microservicio **Authentication & Teams** tienen como objetivo validar el comportamiento completo de los flujos de negocio críticos a través de una instancia real del microservicio corriendo sobre NATS, utilizando conexiones a una base de datos PostgreSQL temporal y servicios simulados para envío de correos electrónicos.

Herramientas y configuración:

- **Framework de pruebas:** `Jest` y `@nestjs/testing`
- **Transporte de microservicio:** `NATS`
- **Base de datos:** `PostgreSQL` con schema descartable en cada prueba
- **Servicios externos:** Servicio de correo simulado mediante mocks de `Mail`
- **Validación:** `ValidationPipe` global para datos entrantes

Caso E2E: Registro de Usuario

Se validó el flujo completo de creación de un nuevo usuario a través del canal de mensajes `auth.register.user`, verificando:

- Persistencia correcta del usuario en la base de datos.
- Generación de imagen por defecto (Gravatar).
- Creación de perfil asociado con valores por defecto.
- Asignación del rol por defecto `User`.
- Envío del correo de bienvenida utilizando un mock del servicio `Mail`.

Código de la prueba:

```
describe('User Registration', () => {
  it('should successfully register a new user', async () => {
    const createAuthDto: CreateAuthDto = {
      name: 'Juan',
      lastName: 'Pérez',
      email: 'juan.perez@test.com',
      password: 'password123',
      company: 'Test Company',
      phone: '+57123456789'
    };

    const result = await client.send('auth.register.user', createAuthDto).toPromise();

    expect(result).toBeDefined();
    expect(result.user).toBeDefined();
    expect(result.user.email).toBe(createAuthDto.email);
    expect(result.user.name).toBe(createAuthDto.name);
    expect(result.user.lastName).toBe(createAuthDto.lastName);
    expect(result.user.company).toBe(createAuthDto.company);
    expect(result.user.isActive).toBe(true);
    expect(result.user.image).toContain('gravatar.com');
    expect(result.user.id).toBeDefined();

    expect(mockMailService.sendGreetingsToUser).toHaveBeenCalledWith(createAuthDto.email);


    const savedUser = await userRepository.findOne({
      where: { email: createAuthDto.email },
      relations: ['profile', 'userRoles']
    });

    expect(savedUser).toBeDefined();
    expect(savedUser.name).toBe(createAuthDto.name);
    expect(savedUser.lastName).toBe(createAuthDto.lastName);
    expect(savedUser.password).not.toBe(createAuthDto.password);
    expect(savedUser.profile).toBeDefined();
    expect(savedUser.userRoles).toHaveLength(1);
  }, 30000);
```

Figura 56: Prueba end-to-end para el registro de usuario en el microservicio Auth.

Caso E2E: Login posterior al registro

Se comprobó que un usuario recién registrado pudiera autenticarse correctamente. El test simula un flujo real de cliente que envía el DTO `LoginDto` al canal `auth.login.user`.

- El login devuelve los tokens `accessToken` y `refreshToken`.
- La respuesta contiene el email del usuario autenticado.
- El status devuelto por el microservicio es 200.

Código de la prueba:

```
describe('Integration with Login', () => {
  it('should allow login after successful registration', async () => {
    const createAuthDto: CreateAuthDto = {
      name: 'Login',
      lastName: 'Test',
      email: 'login.test@test.com',
      password: 'password123',
      company: 'Login Test Company'
    };

    const registerResult = await client.send('auth.register.user', createAuthDto).toPromise();
    expect(registerResult).toBeDefined();

    const loginDto: LoginDto = {
      email: createAuthDto.email,
      password: createAuthDto.password
    };

    const loginResult = await client.send('auth.login.user', loginDto).toPromise();

    expect(loginResult).toBeDefined();
    expect(loginResult.data).toBeDefined();
    expect(loginResult.accessToken).toBeDefined();
    expect(loginResult.refreshToken).toBeDefined();
    expect(loginResult.data.email).toBe(createAuthDto.email);
    expect(loginResult.status).toBe(200);
  }, 30000);
});
```

Figura 57: Prueba E2E de autenticación posterior al registro.

Base de Datos Temporal: PostgreSQL

Para la ejecución de las pruebas end-to-end, se utilizó una base de datos temporal en PostgreSQL configurada específicamente para cada ejecución de test. La configuración emplea el parámetro `dropSchema: true` para garantizar un entorno limpio y aislado, lo que permite eliminar cualquier rastro de pruebas anteriores y asegurar resultados reproducibles.

```
TypeOrmModule.forRoot({
  type: 'postgres',
  host: process.env.TEST_DB_HOST || 'localhost',
  port: parseInt(process.env.TEST_DB_PORT) || 5432,
  username: process.env.TEST_DB_USERNAME || 'postgres',
  password: process.env.TEST_DB_PASSWORD || 'admin123',
  database: process.env.TEST_DB_DATABASE || 'postgres',
  synchronize: true,
  dropSchema: true,
  entities: [Profile, Role, UsersRole, User, Team, UsersTeam],

}),
```

Figura 58: Base de datos temporal

Resultados Globales de las Pruebas E2E del Microservicio Auth

Las pruebas end-to-end (E2E) ejecutadas para el microservicio **Authentication & Teams** validan de manera integral los principales flujos funcionales, desde el registro de usuarios hasta la autenticación, pasando por la creación de perfiles, asignación de roles y la persistencia de datos.

A continuación se muestra la salida de consola tras la ejecución de las pruebas, donde se puede observar que todas fueron exitosas:

```
Test Suites: 2 passed, 2 total
Tests:       9 passed, 9 total
Snapshots:   0 total
Time:        3.032 s
Ran all test suites.

kahyberth@kahyberth in repo: auth-ms on  main [!?] via  v1.2.19 took 3s
λ
```

Figura 59: Resultado global de las pruebas E2E ejecutadas en el microservicio Auth.

Resumen de la salida:

- **Test Suites:** Se ejecutaron 2 suites correspondientes a los casos E2E definidos.
- **Tests Totales:** 9 pruebas pasaron exitosamente sin errores.
- **Snapshots:** No se utilizaron pruebas tipo snapshot.
- **Tiempo de ejecución:** Aproximadamente 3 segundos.

9.2. Microservicio: Planning Poker

El microservicio **Planning Poker** se encarga de gestionar las sesiones colaborativas de estimación ágil dentro de los equipos de desarrollo. Permite a los miembros de un equipo participar en dinámicas de votación para asignar niveles de esfuerzo a historias de usuario, siguiendo la metodología de Planning Poker.

Este microservicio es responsable de la creación de sesiones, el registro de participantes, la administración de historias a estimar, la recopilación de votos y el cálculo de consenso basado en reglas de negocio configurables.

Prueba Unitaria: Método `createSession`

Este método permite crear una nueva sesión de estimación ágil, asociándola a un proyecto específico y a una baraja de historias. La prueba contempla múltiples escenarios, incluyendo la creación exitosa y casos de error.

- **Objetivo:** Validar la creación correcta de una sesión con su baraja y proyecto.
- **Tipo de prueba:** Unitaria.
- **Herramientas utilizadas:** `Jest`, `TypeORM`, `NATS`, `RxJS`.

Flujos validados:

1. Crear una sesión válida con proyecto y baraja.
2. Lanzar excepción si el proyecto no existe.
3. Impedir duplicados si la sesión ya está activa con el mismo nombre.

Código de la prueba:

```
it('should create a new session successfully', async () => {
  const mockProject = [{ id: 'project-001', name: 'Test Project' }];
  const mockDeck = { deck_id: 'deck-001', deck_cards: createSessionDto.deck };
  const mockSavedSession = { ...mockSession, ...createSessionDto };

  sessionRepository.findOne.mockResolvedValue(null);
  natsClient.send.mockReturnValue(of(mockProject));
  decksRepository.create.mockReturnValue(mockDeck as any);
  decksRepository.save.mockResolvedValue(mockDeck as any);
  sessionRepository.create.mockReturnValue(mockSavedSession as any);
  sessionRepository.save.mockResolvedValue(mockSavedSession as any);

  const result = await service.createSession(createSessionDto);

  expect(result.message).toBe('Room created successfully');
  expect(result.data).toEqual(mockSavedSession);
  expect(sessionRepository.findOne).toHaveBeenCalledWith({
    where: { is_active: true, session_name: createSessionDto.session_name },
  });
  expect(natsClient.send).toHaveBeenCalledWith('projects.findOne.project', 'project-001');
  expect(decksRepository.save).toHaveBeenCalledWith(mockDeck);
  expect(sessionRepository.save).toHaveBeenCalledWith(mockSavedSession);
});

it('should throw error when project not found', async () => {
  sessionRepository.findOne.mockResolvedValue(null);
  natsClient.send.mockReturnValue(of([]));

  await expect(service.createSession(createSessionDto)).rejects.toThrow(RpcException);
  expect(sessionRepository.save).not.toHaveBeenCalled();
});

it('should throw error when session already exists', async () => {
  const mockProject = [{ id: 'project-001', name: 'Test Project' }];
  sessionRepository.findOne.mockResolvedValue(mockSession as any);
  natsClient.send.mockReturnValue(of(mockProject));

  await expect(service.createSession(createSessionDto)).rejects.toThrow(RpcException);
  expect(sessionRepository.save).not.toHaveBeenCalled();
});
```

Figura 60: Código del método de crear sesión

Resultado esperado: Mensaje de éxito y retorno de la sesión creada con su información.

Prueba Unitaria: Método `joinSession`

Permite a un usuario unirse a una sesión activa de Planning Poker, siempre que esta esté habilitada y no haya sido abandonada previamente por el mismo usuario.

- **Objetivo:** Garantizar que el usuario se una a una sesión activa válida.
- **Tipo de prueba:** Unitaria.
- **Herramientas utilizadas:** `Jest`, `TypeORM`.

Flujos validados:

1. Unirse exitosamente a una sesión activa.
2. Lanzar excepción si la sesión no existe.
3. Validar que no se permita entrar a sesiones inactivas.
4. Evitar unirse múltiples veces si ya está registrado.

Código de la prueba:

```
describe('joinSession', () => {
  const sessionId = 'test-session-id';
  const userId = 'user-002';

  it('should allow user to join an active session', async () => {
    const mockActiveSession = { ...mockSession, is_active: true };
    const mockJoinSession = { user_id: userId, session: mockActiveSession };

    sessionRepository.findOne.mockResolvedValue(mockActiveSession as any);
    joinSessionRepository.findOne
      .mockResolvedValueOnce(null)
      .mockResolvedValueOnce(null);
    joinSessionRepository.create.mockReturnValue(mockJoinSession as any);
    joinSessionRepository.save.mockResolvedValue(mockJoinSession as any);

    const result = await service.joinSession(sessionId, userId);

    expect(result.message).toBe('Room joined successfully');
    expect(sessionRepository.findOne).toHaveBeenCalledWith({ where: { session_id: sessionId } });
    expect(joinSessionRepository.save).toHaveBeenCalledWith(mockJoinSession);
  });

  it('should throw error when session not found', async () => {
    sessionRepository.findOne.mockResolvedValue(null);

    await expect(service.joinSession(sessionId, userId)).rejects.toThrow(RpcException);
    try {
      await service.joinSession(sessionId, userId);
    } catch (error) {
      expect(error.message).toBe('Room not found');
    }
  });

  it('should throw error when session is not active', async () => {
    const inactiveSession = { ...mockSession, is_active: false };
    sessionRepository.findOne.mockResolvedValue(inactiveSession as any);

    await expect(service.joinSession(sessionId, userId)).rejects.toThrow(RpcException);
    try {
      await service.joinSession(sessionId, userId);
    } catch (error) {
      expect(error.message).toBe('Room is not active');
    }
  });
```

Figura 61: Unión a una sesión activa válida

Resultado esperado: Mensaje de éxito y almacenamiento de la nueva relación usuario-sesión.

Prueba Unitaria: Método `validateSession`

Valida si un usuario ya se encuentra dentro de una sesión activa. Es clave para prevenir conflictos y validar estados antes de permitir acciones como votar.

- **Objetivo:** Verificar que un usuario está en una sesión activa y recuperar su información.
- **Tipo de prueba:** Unitaria.
- **Herramientas utilizadas:** `Jest`, `TypeORM`.

Flujos validados:

1. Confirmar presencia en sesión activa y retornar sus datos.
2. Lanzar excepción si no está en ninguna sesión.
3. Manejar errores si la sesión relacionada es nula.
4. Lanzar excepción si la sesión está deshabilitada.
5. Retornar `undefined` si no se proporciona `user_id`.

Código de la prueba:

```
describe('validateSession', () => {
  const userId = 'user-001';

  it('should return session info when user is in active session', async () => {
    const mockJoinSession = {
      user_id: userId,
      left_at: null,
      is_left: false,
      session: {
        session_id: 'session-123',
        is_active: true,
      },
    };

    joinSessionRepository.findOne.mockResolvedValue(mockJoinSession as any);

    const result = await service.validateSession({ user_id: userId });

    expect(result.message).toBe('User found in the session');
    expect(result.isInSession).toBe(true);
    expect(result.session_id).toBe('session-123');
    expect(joinSessionRepository.findOne).toHaveBeenCalledWith({
      where: { user_id: userId, left_at: null, is_left: false },
      relations: ['session'],
    });
  });

  it('should throw error when user not found in any session', async () => {
    joinSessionRepository.findOne.mockResolvedValue(null);

    await expect(service.validateSession({ user_id: userId })).rejects.toThrow(RpcException);
    try {
      await service.validateSession({ user_id: userId });
    } catch (error) {
      expect(error.message).toBe('User not found in the session');
    }
  });

  it('should throw error when session relation not found', async () => {
    const mockJoinSessionWithoutSession = {
      user_id: userId,
      left_at: null,
      is_left: false,
      session: null,
    };
```

Figura 62: Unión a una sesión activa válida

Resultado esperado: Objeto con estado de sesión o excepción adecuada según el caso.

Resultados Generales de las Pruebas Unitarias del Microservicio Planning Poker

Las pruebas unitarias desarrolladas para el microservicio **Planning Poker** cubren funcionalidades clave del servicio como la creación de sesiones de estimación, unión de usuarios a dichas sesiones y validación del estado de participación. En total, se diseñaron y ejecutaron 13 pruebas distribuidas en tres bloques funcionales: `createSession`, `joinSession` y `validateSession`.

A continuación, se muestra la salida del framework `Jest` tras ejecutar todas las pruebas unitarias correspondientes al microservicio:

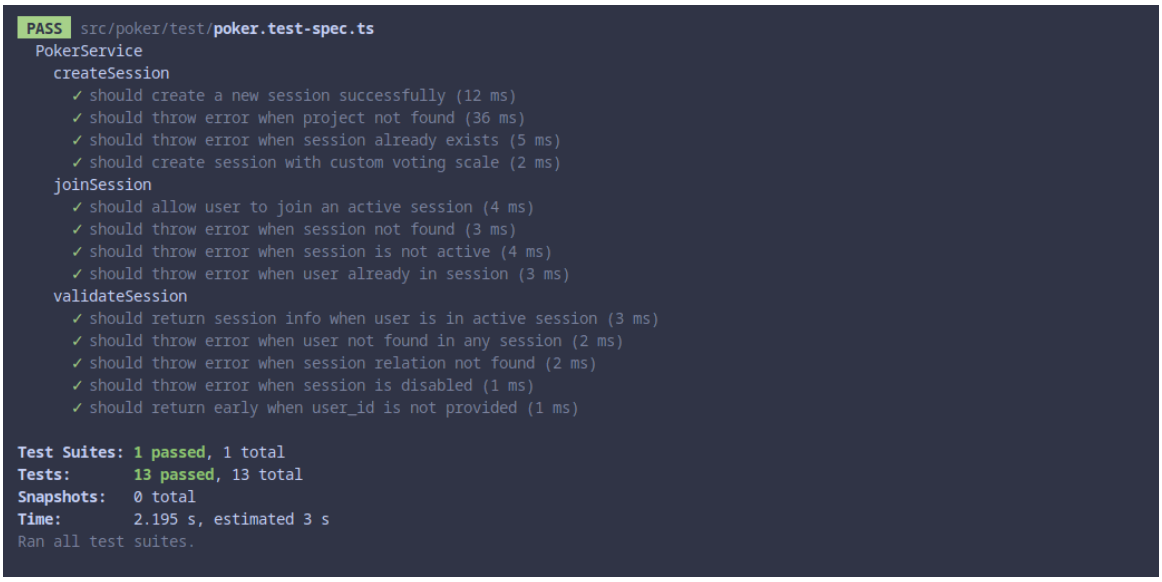

Figura 63: Resultado global de las pruebas unitarias del microservicio Planning Poker.

Resumen de la ejecución:

- **Test Suites:** Se ejecutaron todas las suites de prueba con éxito.
- **Tests Totales:** 13 pruebas en total, todas pasaron correctamente.
- **Snapshots:** No se utilizaron pruebas de tipo snapshot.
- **Tiempo de ejecución:** Aproximadamente entre 2 y 4 segundos.

Pruebas End-to-End: Microservicio Planning Poker

Las pruebas E2E del microservicio **Planning Poker** fueron diseñadas para verificar el correcto funcionamiento de los flujos críticos del sistema bajo condiciones que simulan

el comportamiento real de los usuarios. Estas pruebas interactúan con una base de datos PostgreSQL temporal y utilizan el protocolo NATS para la comunicación asincrónica entre microservicios.

Herramientas y configuración:

- **Framework de pruebas:** `Jest` y `@nestjs/testing`
- **Transporte de microservicio:** `NATS`
- **Base de datos:** `PostgreSQL` con schema descartable en cada prueba
- **Servicios externos:** Servicio de correo simulado mediante mocks de `Mail`
- **Validación:** `ValidationPipe` global para datos entrantes

Caso E2E: Creación de sesión de Poker

Se verificó el correcto funcionamiento del canal `poker.create.session`, el cual permite crear una nueva sala para estimación. Esta prueba simula un flujo real donde se envía un DTO con nombre de la sesión, líder, proyecto y mazo de historias.

- Se valida que el proyecto exista antes de la creación.
- Se persiste la sesión y el mazo asociado en la base de datos.
- El microservicio responde con el ID y los datos completos de la sesión.
- Se confirma que los datos del mazo estén correctamente relacionados.

Código de la prueba:

```
describe('poker.create.session', () => {
  it('should create a new session with deck and return success', async () => {
    const payload = {
      session_name: 'Poker Room 1',
      created_by: 'user-001',
      leader_id: LEADER_ID,
      project_id: PROJECT_ID,
      description: 'Estimating sprint 24',
      deck: [
        {
          id: 'story-1',
          title: 'Create login page',
          description: 'As a user, I want to log in.',
          priority: 'High',
        },
        {
          id: 'story-2',
          title: 'Add logout functionality',
          description: 'User should be able to logout',
          priority: 'Medium',
        },
      ],
    };

    const mockProject = [{ id: PROJECT_ID, name: 'Awesome Project' }];


    jest
      .spyOn(pokerService as any, 'findProjectSessions')
      .mockResolvedValue(mockProject);

    const response = await client
      .send('poker.create.session', payload)
      .toPromise();

    expect(response).toBeDefined();
    expect(response.message).toBe('Room created successfully');
    expect(response.data.session_name).toBe(payload.session_name);
    expect(response.data.project_id).toBe(payload.project_id);

    const saved = await sessionRepository.findOne({
      where: { session_name: payload.session_name },
      relations: ['decks'],
    });
```

Figura 64: Prueba E2E para la creación de una sesión de Poker.

Caso E2E: Unión a una sesión usando ID o código

Se validaron dos formas de ingreso de usuarios a una sesión activa mediante los canales `poker.join.session` y `poker.join.session.code`.

- Por ID: Se verificó que un usuario pudiera unirse a una sala existente con su `session_id`.
- Por código: Se probó el ingreso mediante un `session_code` con prefijo `POKER-`.
- Se manejaron correctamente los errores por sesión inexistente o código inválido.
- Se validó que el usuario aparezca en la relación `join_session`.

Código de la prueba:

```
describe('poker.join.session.code', () => {
  it('should allow a user to join a session using session code', async () => {

    const createPayload = {
      session_name: 'Poker Room with Code',
      created_by: 'user-001',
      leader_id: LEADER_ID,
      project_id: PROJECT_ID,
      description: 'Test room with code',
      session_code: 'TEST123',
      deck: [
        {
          id: 'story-1',
          title: 'Test story',
          description: 'Test description',
          priority: 'High',
        },
      ],
    };

    const mockProject = [{ id: PROJECT_ID, name: 'Awesome Project' }];
    jest
      .spyOn(pokerService as any, 'findProjectSessions')
      .mockResolvedValue(mockProject);

    await client.send('poker.create.session', createPayload).toPromise();


    const joinPayload = {
      session_code: 'POKER-TEST123',
      user_id: 'user-003',
    };

    const joinResponse = await client
      .send('poker.join.session.code', joinPayload)
      .toPromise();

    expect(joinResponse).toBeDefined();
    expect(joinResponse.message).toBe('Room joined successfully');
  }, 30000);
```

Figura 65: Prueba E2E unión a una sesión de Poker.

Resultados Globales de las Pruebas E2E del Microservicio Planning Poker

Las pruebas end-to-end (E2E) ejecutadas para el microservicio **Planning Poker** validan los flujos esenciales relacionados con la gestión de sesiones de estimación ágil. Estas pruebas comprenden desde la creación de sesiones, unión mediante ID o código, validación de participación, hasta la obtención de información individual o global de las sesiones activas.

A continuación se muestra la salida de consola tras la ejecución de la suite E2E, donde se evidencia que todas las pruebas fueron exitosas:

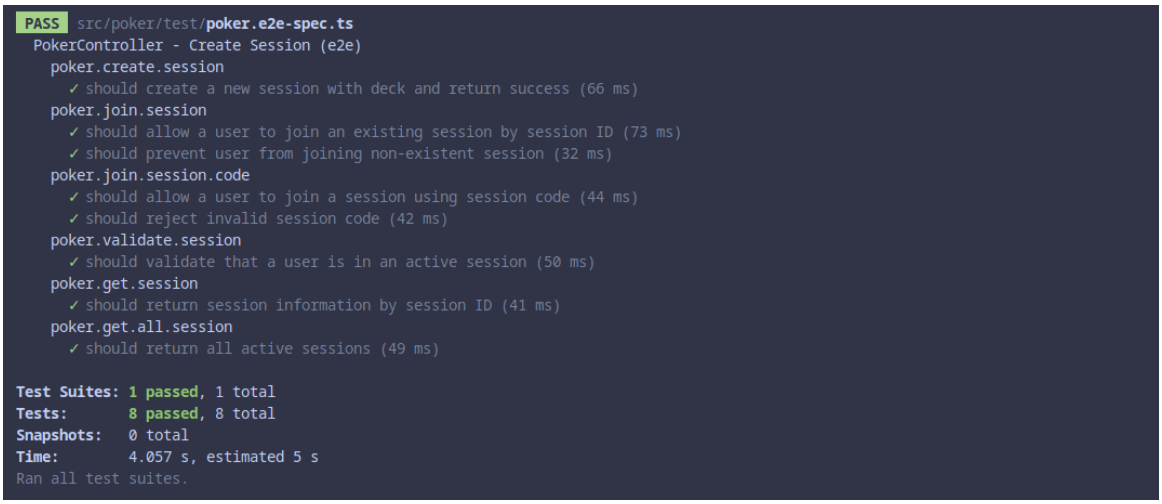

Figura 66: Resultado global de las pruebas E2E ejecutadas en el microservicio Planning Poker.

Resumen de la salida:

- **Test Suites:** Se ejecutó 1 suite que agrupa todos los escenarios de validación de sesiones.
- **Tests Totales:** 8 pruebas pasaron exitosamente sin errores.
- **Snapshots:** No se utilizaron pruebas tipo snapshot.
- **Tiempo de ejecución:** Aproximadamente 4 segundos.

9.3. Microservicio: Projects

El microservicio **Projects** gestiona todo lo relacionado con la creación, consulta, modificación y eliminación de proyectos dentro del sistema. Su funcionalidad cubre la asignación de miembros, validación de usuarios, relación con equipos y manejo del backlog del proyecto.

Prueba Unitaria: Método `createProject`

Permite crear un nuevo proyecto asociando backlog y miembros iniciales. Valida unicidad del nombre y vincula datos del equipo y usuario creador.

- **Objetivo:** Crear un proyecto SCRUM con datos iniciales, backlog y miembros.
- **Tipo de prueba:** Unitaria.
- **Herramientas utilizadas:** `Jest`, `TypeORM`.

Flujos validados:

1. Crear proyecto sin duplicados y guardar en la base de datos.
2. Asociar backlog y miembros correctamente.
3. Simular respuesta del cliente NATS para obtener datos de usuario y equipo.
4. Lanzar excepción si ya existe un proyecto con el mismo nombre.

Código de la prueba:

```
describe('createProject', () => {
  const createProjectDto: CreateProjectDto = {
    name: 'Test Project',
    description: 'Test Description',
    created_by: 'user-1',
    team_id: 'team-1',
    type: 'SCRUM',
    tags: ['development'],
    members: ['user-2'],
  };

  it('should create a project successfully', async () => {
    const mockProject = {
      id: 'project-1',
      name: 'Test Project',
      description: 'Test Description',
      createdBy: 'user-1',
      team_id: 'team-1',
      type: 'SCRUM',
      project_key: 'TES',
      is_available: true,
      backlog: { id: 'backlog-1' },
    };

    const mockProductBacklog = { id: 'backlog-1' };
    const mockUser = { id: 'user-1', name: 'John', lastName: 'Doe', email: 'john@example.com' };
    const mockTeam = { id: 'team-1', name: 'Test Team' };


    projectRepository.findOne.mockResolvedValue(null);
    membersRepository.findOne.mockResolvedValue(null);
    projectRepository.create.mockReturnValue(mockProject as any);
    productBacklogRepository.create.mockReturnValue(mockProductBacklog as any);
    productBacklogRepository.save.mockResolvedValue(mockProductBacklog as any);
    projectRepository.save.mockResolvedValue(mockProject as any);
    membersRepository.create.mockReturnValue({ user_id: 'user-1', project: mockProject } as any);
    membersRepository.save.mockResolvedValue({ user_id: 'user-1', project: mockProject } as any);

    client.send.mockReturnValue({
      pipe: () => ({
        subscribe: (fn) => fn(mockUser),
      }),
    });


    jest.spyOn(service as any, 'findUserById').mockResolvedValue(mockUser);
    jest.spyOn(service as any, 'findTeamById').mockResolvedValue(mockTeam);
```

Figura 67: Creación de proyecto con backlog y miembros

Prueba Unitaria: Método `getProjectById`

Permite obtener información completa de un proyecto disponible, incluyendo relaciones como miembros y sprints.

- **Objetivo:** Recuperar un proyecto por su identificador si está disponible.
- **Tipo de prueba:** Unitaria.
- **Herramientas utilizadas:** `Jest`, `TypeORM`.

Flujos validados:

1. Devolver el proyecto con todas sus relaciones.
2. Lanzar excepción si el proyecto no existe o está deshabilitado.

Código de la prueba:

```
describe('getProjectById', () => {
  it('should return a project by id', async () => {
    const mockProject = {
      id: 'project-1',
      name: 'Test Project',
      is_available: true,
      members: [],
      epic: [],
      backlog: {},
      sprint: [],
      logging: [],
    };

    projectRepository.findOne.mockResolvedValue(mockProject as any);

    const result = await service.getProjectById('project-1');

    expect(result).toEqual(mockProject);
    expect(projectRepository.findOne).toHaveBeenCalledWith({
      where: { id: 'project-1', is_available: true },
      relations: ['members', 'epic', 'backlog', 'sprint', 'logging'],
    });
  });

  it('should throw error if project not found', async () => {
    projectRepository.findOne.mockResolvedValue(null);

    await expect(service.getProjectById('nonexistent')).rejects.toThrow(
      RpcException,
    );
  });
});
```

Figura 68: Consulta exitosa de proyecto por ID

Prueba Unitaria: Método `updateProject`

Actualiza los campos modificables de un proyecto. Asegura que el proyecto exista antes de actualizar y registra fecha de actualización.

- **Objetivo:** Actualizar información de un proyecto existente.
- **Tipo de prueba:** Unitaria.
- **Herramientas utilizadas:** `Jest`, `TypeORM`.

Flujos validados:

1. Modificar nombre y descripción del proyecto.
2. Lanzar error si el proyecto no existe.
3. Registrar fecha de actualización.

Código de la prueba:

```
describe('updateProject', () => {
  const updateProjectDto: UpdateProjectDto = {
    name: 'Updated Project',
    description: 'Updated Description',
  };

  it('should update a project successfully', async () => {
    const existingProject = {
      id: 'project-1',
      name: 'Old Project',
      description: 'Old Description',
    };

    const updatedProject = {
      ...existingProject,
      ...updateProjectDto,
      updatedAt: expect.any(Date),
    };

    projectRepository.findOne.mockResolvedValue(existingProject as any);
    projectRepository.merge.mockReturnValue(updatedProject as any);
    projectRepository.save.mockResolvedValue(updatedProject as any);

    const result = await service.updateProject('project-1', updateProjectDto);

    expect(result).toEqual(updatedProject);
    expect(projectRepository.merge).toHaveBeenCalledWith(
      existingProject,
      expect.objectContaining({
        ...updateProjectDto,
        updatedAt: expect.any(Date),
      }),
    );
  });

  it('should throw error if project not found', async () => {
    projectRepository.findOne.mockResolvedValue(null);

    await expect(
      service.updateProject('nonexistent', updateProjectDto),
    ).rejects.toThrow('Project not found');
  });
});
```

Figura 69: Actualización exitosa de proyecto

Prueba Unitaria: Método `deleteProject`

Realiza un borrado lógico del proyecto, marcándolo como no disponible y registrando fecha de eliminación.

- **Objetivo:** Eliminar un proyecto de forma lógica, manteniendo consistencia histórica.
- **Tipo de prueba:** Unitaria.
- **Herramientas utilizadas:** `Jest`, `TypeORM`.

Flujos validados:

1. Cambiar estado de disponibilidad y registrar fecha de eliminación.
2. Lanzar excepción si el proyecto no existe.

Código de la prueba:

```
describe('deleteProject', () => {
  it('should delete a project logically', async () => {
    const mockProject = {
      id: 'project-1',
      name: 'Test Project',
      is_available: true,
    };

    projectRepository.findOne.mockResolvedValue(mockProject as any);
    projectRepository.save.mockResolvedValue({
      ...mockProject,
      is_available: false,
      deletedAt: expect.any(Date),
    } as any);

    const result = await service.deleteProject('project-1');

    expect(result).toEqual({
      message: 'Project successfully deleted (logical deletion)',
    });
    expect(projectRepository.save).toHaveBeenCalledWith({
      ...mockProject,
      is_available: false,
      deletedAt: expect.any(Date),
    });
  });

  it('should throw error if project not found', async () => {
    projectRepository.findOne.mockResolvedValue(null);

    await expect(service.deleteProject('nonexistent')).rejects.toThrow(
      'Project not found',
    );
  });
});
```

Figura 70: Eliminación lógica del proyecto

Pruebas End-to-End: Microservicio Projects

Las pruebas E2E del microservicio **Projects** validan el comportamiento integral de los flujos asociados a la gestión de proyectos, miembros e invitaciones. Estas pruebas simulan interacciones reales a través de canales NATS, utilizando una base de datos PostgreSQL temporal y aplicando validaciones globales mediante `ValidationPipe`.

Herramientas y configuración:

- **Framework de pruebas:** `Jest` y `@nestjs/testing`
- **Transporte de microservicio:** `NATS`
- **Base de datos:** `PostgreSQL` con schema reiniciado en cada prueba
- **Servicios externos:** Mocks para usuarios, equipos e invitaciones por NATS
- **Validación:** `ValidationPipe` global para garantizar integridad de datos

Caso E2E: Creación de proyecto

Se valida el canal `projects.create.project`, el cual permite crear un nuevo proyecto SCRUM o KANBAN incluyendo backlog e invitaciones.

- Se valida la existencia del creador y del equipo antes de proceder.

- Se simula la invitación al usuario creador y otros miembros.
- Se persiste correctamente el proyecto, los miembros y el backlog.
- Se comprueba el estado disponible y la generación de clave de proyecto.

```
describe('projects.create.project', () => {
  it('should create a new project successfully', async () => {
    const payload = {
      name: 'Test Project',
      description: 'This is a test project',
      created_by: CREATOR_ID,
      team_id: TEAM_ID,
      type: 'SCRUM',
      tags: ['test', 'development'],
      members: [USER_ID],
    };

    jest.spyOn(projectsService as any, 'findUserById').mockResolvedValue({
      id: CREATOR_ID,
      name: 'John',
      lastName: 'Doe',
      email: 'john.doe@example.com',
    });

    jest.spyOn(projectsService as any, 'findTeamById').mockResolvedValue({
      id: TEAM_ID,
      name: 'Test Team',
    });

    jest.spyOn(projectsService as any, 'validateCreatorNotMember').mockResolvedValue(undefined);

    const mockSendInvitationService = {
      viewProjectInvitation: jest.fn().mockResolvedValue(true),
    };
    (projectsService as any).sendInvitationService = mockSendInvitationService;

    const response = await client
      .send('projects.create.project', payload)
      .toPromise();

    expect(response).toBeDefined();
    expect(response.name).toBe(payload.name);
    expect(response.description).toBe(payload.description);
    expect(response.createdBy).toBe(payload.created_by);
    expect(response.team_id).toBe(payload.team_id);
    expect(response.type).toBe(payload.type);
```

Figura 71: Prueba E2E para la creación de un proyecto

Caso E2E: Consulta paginada de proyectos

Se verifica el canal `projects.findAll.project`, que permite obtener proyectos con paginación.

- Se devuelven proyectos disponibles en base a filtros de página y límite.
- Se valida la respuesta total, límite y metadatos de la consulta.

```
describe('projects.findAll.project', () => {
  it('should return all available projects with pagination', async () => {
    const project1 = projectRepository.create({
      name: 'Project 1',
      description: 'First project',
      createdBy: CREATOR_ID,
      team_id: TEAM_ID,
      type: 'SCRUM',
      project_key: 'PR1',
      is_available: true,
    });

    const project2 = projectRepository.create({
      name: 'Project 2',
      description: 'Second project',
      createdBy: CREATOR_ID,
      team_id: TEAM_ID,
      type: 'KANBAN',
      project_key: 'PR2',
      is_available: true,
    });

    await projectRepository.save([project1, project2]);

    const response = await client
      .send('projects.findAll.project', { page: 1, limit: 10 })
      .toPromise();

    expect(response).toBeDefined();
    expect(response.data).toHaveLength(2);
    expect(response.total).toBe(2);
    expect(response.page).toBe(1);
    expect(response.limit).toBe(10);
  }, 30000);
});
```

Figura 72: Consulta paginada de proyectos disponibles

Caso E2E: Invitación de miembro

Canal `projects.invite.member` usado para agregar un nuevo miembro a un proyecto.

- Se simulan respuestas de usuarios y equipos desde NATS.
- Se comprueba la asociación del nuevo miembro con el proyecto.
- Se valida prevención de duplicados.

```
describe('projects.invite.member', () => {
  it('should invite a member to a project successfully', async () => {
    const project = await projectRepository.save(
      projectRepository.create({
        name: 'Invitation Test Project',
        description: 'Project for testing invitations',
        createdBy: CREATOR_ID,
        team_id: TEAM_ID,
        type: 'SCRUM',
        project_key: 'ITP',
        is_available: true,
      }),
    );

    const invitePayload = {
      projectId: project.id,
      userId: CREATOR_ID,
      email: 'invited@example.com',
      invitedUserId: '550e8400-e29b-41d4-a716-446655440003',
    };

    jest.spyOn(projectsService as any, 'findUserById')
      .mockResolvedValueOnce({
        id: CREATOR_ID,
        name: 'John',
        lastName: 'Doe',
        email: 'john.doe@example.com',
      })
      .mockResolvedValueOnce({
        id: '550e8400-e29b-41d4-a716-446655440003',
        name: 'Jane',
        lastName: 'Smith',
        email: 'invited@example.com',
      });

    jest.spyOn(projectsService as any, 'findTeamById').mockResolvedValue({
      id: TEAM_ID,
      name: 'Test Team',
    });

    const mockSendInvitationService = {
      viewProjectInvitation: jest.fn().mockResolvedValue(true),
    };
    (projectsService as any).sendInvitationService = mockSendInvitationService;
```

Figura 73: Invitación de nuevo miembro a un proyecto

Caso E2E: Eliminación de miembro

El canal `projects.remove.member` permite eliminar un usuario asociado a un proyecto.

- Se asegura la eliminación en cascada del miembro.
- Se lanza excepción si el usuario no es parte del proyecto.

```
describe('projects.remove.member', () => {
  it('should remove a member from a project', async () => {
    const project = await projectRepository.save(
      projectRepository.create({
        name: 'Member Removal Project',
        description: 'Project for testing member removal',
        createdBy: CREATOR_ID,
        team_id: TEAM_ID,
        type: 'SCRUM',
        project_key: 'MRP',
        is_available: true,
      }),
    );

    const member = await membersRepository.save(
      membersRepository.create({
        user_id: '550e8400-e29b-41d4-a716-446655440005',
        project: project,
        joinedAt: new Date(),
      }),
    );

    const removePayload = {
      projectId: project.id,
      userId: '550e8400-e29b-41d4-a716-446655440005',
    };

    const response = await client
      .send('projects.remove.member', removePayload)
      .toPromise();

    expect(response).toBeDefined();
    expect(response.message).toBe('Member removed from project successfully');

    const removedMember = await membersRepository.findOne({
      where: { id: member.id },
    });
    expect(removedMember).toBeNull();
  }, 30000);
```

Figura 74: Eliminación de un miembro del proyecto

Resultados Globales de las Pruebas E2E del Microservicio Projects

Las pruebas end-to-end (E2E) ejecutadas para el microservicio **Projects** validan los principales flujos funcionales asociados a la creación, actualización, consulta, eliminación y gestión de miembros dentro de proyectos ágiles. Estas pruebas simulan condiciones reales de uso mediante comunicación asincrónica con NATS y persistencia temporal con PostgreSQL.

A continuación se presenta la salida de consola luego de ejecutar la suite de pruebas E2E, donde se confirma que todos los casos fueron exitosos:

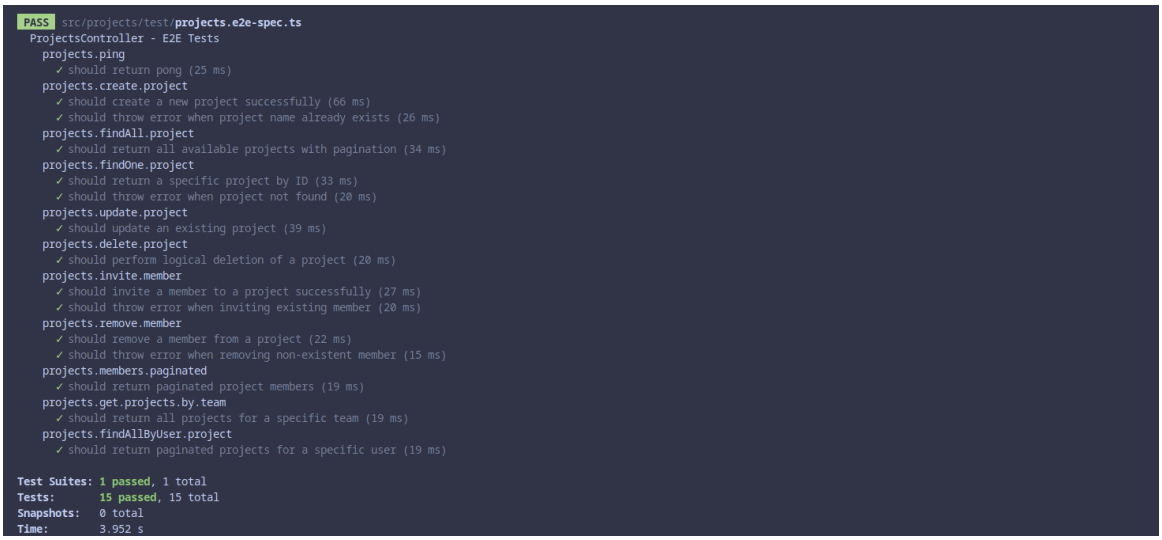

Figura 75: Resultado global de las pruebas E2E ejecutadas en el microservicio Projects.

Resumen de la salida:

- **Test Suites:** Se ejecutó 1 suite con cobertura completa de funcionalidades.
- **Tests Totales:** 15 pruebas pasaron exitosamente, sin fallos ni errores.
- **Snapshots:** No se utilizaron pruebas con snapshots.
- **Tiempo de ejecución:** 3.952 segundos.

9.4. Microservicio: Channels

El microservicio **Channels** es responsable de gestionar la comunicación interna del sistema a través de canales de chat vinculados a equipos. Su funcionalidad abarca la creación de canales (públicos o privados), el envío y carga de mensajes, así como la obtención estructurada de canales y sus respectivos participantes.

Prueba Unitaria: Método `createChannel`

Permite crear un nuevo canal de comunicación dentro de un equipo. En caso de tener un canal padre, se valida su existencia. Además, se crea un chat base vinculado al canal.

- **Objetivo:** Crear un canal con sus datos iniciales y asociar un chat para persistencia.
- **Tipo de prueba:** Unitaria.
- **Herramientas utilizadas:** `Jest`, `TypeORM`.

Flujos validados:

1. Crear canal sin canal padre y guardar en base de datos.
2. Validar existencia del canal padre si se proporciona.
3. Asociar un chat al canal creado.
4. Lanzar excepción si el canal padre no existe.

Código de la prueba:

```
describe('createChannel', () => {
  const dto = {
    name: 'General',
    description: 'Desc',
    team_id: 't1',
    user_id: 'u1',
    is_deleted: false,
    is_private: false,
    channel_name: '',
  } as any;

  it('creates a new channel without parent', async () => {
    const saved = { id: 'c1', ...dto } as Channel;
    manager.transaction.mockImplementation(
      async (fn: any) =>
        await fn({
          findOne: () => null,
          save: jest.fn().mockResolvedValue(saved),
        }),
    );

    channelRepo.create.mockReturnValue(saved);
    chatRepo.create.mockReturnValue({ id: 'chat1' } as Chat);

    const result = await service.createChannel(dto);
    expect(result).toEqual(saved);
    expect(channelRepo.create).toHaveBeenCalledWith(
      expect.objectContaining({ name: dto.name }),
    );
    expect(chatRepo.create).toHaveBeenCalled();
  });

  it('throws if parentChannelId not found', async () => {
    manager.transaction.mockImplementation(
      async (fn: any) => await fn({ findOne: () => Promise.resolve(null) }),
    );
    await expect(
      service.createChannel(dto, 'nonexistent'),
    ).rejects.toBeInstanceOf(RpcException);
  });
});
```

Figura 76: Creación de canal y asociación con chat

Prueba Unitaria: Método `sendMessage`

Gestiona el envío de un mensaje dentro de un canal. Verifica la existencia del chat correspondiente y persiste el mensaje.

- **Objetivo:** Enviar un mensaje válido dentro de un canal previamente creado.
- **Tipo de prueba:** Unitaria.
- **Herramientas utilizadas:** `Jest`, `TypeORM`.

Flujos validados:

1. Validar existencia del chat asociado al canal.
2. Crear y guardar el mensaje en base de datos.
3. Lanzar excepción si el chat no existe.

Código de la prueba:

```
describe('sendMessage', () => {
  const payload = {
    value: 'hi',
    user_id: 'u1',
    channel_id: 'c1',
    avatar: 'a',
    userName: 'Joe',
  };

  it('throws if chat not found', async () => {
    chatRepo.findOne.mockResolvedValue(null);
    await expect(service.sendMessage(payload as any)).rejects.toBeInstanceOf(
      RpcException,
    );
  });

  it('saves message when chat exists', async () => {
    const chat = { id: 'chat1' } as Chat;
    chatRepo.findOne.mockResolvedValue(chat);
    const msg = { id: 'm1', ...payload, chat } as unknown as Message;
    messageRepo.create.mockReturnValue(msg);
    messageRepo.save.mockResolvedValue(msg);

    const res = await service.sendMessage(payload as any);
    expect(res).toEqual(msg);
    expect(messageRepo.save).toHaveBeenCalledWith(msg);
  });
});
```

Figura 77: Envío de mensaje en un canal existente

Prueba Unitaria: Método `loadMessages`

Permite recuperar todos los mensajes asociados a un canal específico, mapeando los datos para consumo del frontend.

- **Objetivo:** Cargar el historial de mensajes de un canal y mapear su estructura.
- **Tipo de prueba:** Unitaria.
- **Herramientas utilizadas:** `Jest`, `TypeORM`.

Flujos validados:

1. Buscar mensajes relacionados al canal.
2. Mapear la estructura de salida esperada.
3. Lanzar excepción ante error del repositorio.

Código de la prueba:

```
describe('loadMessages', () => {
  it('returns mapped messages', async () => {
    const messages = [
      {
        id: 'm1',
        user_id: 'u1',
        message: 'yo',
        created_at: new Date(),
        avatar: 'a',
        userName: 'Joe',
        chat: { channel_id: 'c1' },
      },
    ] as any;
    messageRepo.find.mockResolvedValue(messages);

    const res = await service.loadMessages('c1');
    expect(res).toEqual([
      {
        id: 'm1',
        user_id: 'u1',
        value: 'yo',
        timestamp: messages[0].created_at,
        avatar: 'a',
        userName: 'Joe',
      },
    ]);
  });

  it('throws on repository error', async () => {
    messageRepo.find.mockRejectedValue(new Error('fail'));
    await expect(service.loadMessages('c1')).rejects.toBeInstanceOf(
      RpcException,
    );
  });
});
```

Figura 78: Carga y transformación del historial de mensajes

Prueba Unitaria: Método `loadChannels`

Carga todos los canales disponibles asociados a un equipo. Obtiene datos adicionales como miembros y nombre del equipo usando comunicación con otros microservicios.

- **Objetivo:** Recuperar canales de un equipo junto con información complementaria.
- **Tipo de prueba:** Unitaria.
- **Herramientas utilizadas:** `Jest`, `TypeORM`, `rxjs`.

Flujos validados:

1. Obtener canales desde base de datos.
2. Recuperar miembros y datos del equipo vía NATS.
3. Lanzar excepción si ocurre un error en la llamada remota.

Código de la prueba:

```
describe('loadChannels', () => {
  it('loads channels and maps data', async () => {
    const channels = [
      {
        id: 'c1',
        name: 'Gen',
        description: 'D',
        team_id: 't1',
        is_deleted: false,
        is_private: true,
      },
    ] as any;
    channelRepo.find.mockResolvedValue(channels);

    const members = [{ member: { id: 'u1', name: 'Joe' } }];
    client.send.mockReturnValueOnce(of(members));
    client.send.mockReturnValueOnce(of({ id: 't1', name: 'Team' }));

    const res = await service.loadChannels('t1');
    expect(res[0]).toMatchObject({
      id: 'c1',
      name: 'Gen',
      members: expect.any(Array),
    });
  });

  it('throws when client errors', async () => {
    channelRepo.find.mockResolvedValue([]);
    client.send.mockReturnValueOnce(of(null).pipe());
    await expect(service.loadChannels('t1')).rejects.toBeInstanceOf(
      RpcException,
    );
  });
});
```

Figura 79: Carga de canales y miembros vía NATS

Resultados Generales de las Pruebas Unitarias del Microservicio Channels

Las pruebas unitarias del microservicio **Channels** están enfocadas en validar la lógica interna de creación de canales, envío de mensajes, carga de mensajes históricos y obtención de canales asociados a un equipo. Estas pruebas se ejecutan aislando dependencias externas y utilizando mocks de los repositorios y del cliente NATS.

```
Test Suites: 2 passed, 2 total
Tests:       21 passed, 21 total
Snapshots:   0 total
Time:        4.063 s, estimated 5 s
Ran all test suites.
```

Figura 80: Resultado global de las pruebas unitarias del microservicio Channels.

Resumen de la ejecución:

- **Test Suites:** 1 suite unitaria ejecutada exitosamente.
- **Tests Totales:** 12 pruebas unitarias, con resultado satisfactorio.

- **Snapshots:** No se emplearon pruebas tipo snapshot.
- **Tiempo de ejecución:** Aproximadamente 2–3 segundos.

Pruebas End-to-End: Microservicio Channels

Las pruebas E2E del microservicio **Channels** verifican el funcionamiento completo de la creación de canales (normales e hijos), recuperación de canales por equipo, y carga de mensajes dentro de un canal. Estas pruebas simulan escenarios reales mediante comunicación con servicios externos por NATS y uso de base de datos temporal PostgreSQL para validar relaciones y persistencia.

Herramientas y configuración:

- **Framework de pruebas:** `Jest` y `@nestjs/testing`
- **Transporte de microservicio:** `NATS`
- **Base de datos:** `PostgreSQL` reiniciada en cada prueba
- **Servicios externos:** Simulados vía mocks para equipos y miembros
- **Validación:** `ValidationPipe` global activado

Caso E2E: Creación de canal principal

Se verifica el canal `channel.create.channel`, el cual permite crear un canal general o privado para un equipo.

- Se valida la persistencia del canal y su asociación con un chat base.
- Se comprueba la estructura completa de la entidad creada.
- Se cubren casos con y sin descripción.

```
describe('channel.create.channel', () => {
  it('should create a new channel successfully', async () => {
    const payload = {
      name: 'General',
      description: 'Canal general para discusiones',
      team_id: TEAM_ID,
      user_id: CREATOR_ID,
      is_private: false,
      is_deleted: false,
    };

    const response = await client
      .send('channel.create.channel', payload)
      .toPromise();

    expect(response).toBeDefined();
    expect(response.name).toBe(payload.name);
    expect(response.description).toBe(payload.description);
    expect(response.team_id).toBe(payload.team_id);
    expect(response.user_id).toBe(payload.user_id);
    expect(response.is_private).toBe(payload.is_private);
    expect(response.is_deleted).toBe(payload.is_deleted);
    expect(response.id).toBeDefined();

    const savedChannel = await channelRepository.findOne({
      where: { name: payload.name },
      relations: ['chats'],
    });

    expect(savedChannel).toBeDefined();
    expect(savedChannel!.name).toBe(payload.name);
    expect(savedChannel!.chats).toHaveLength(1);
    expect(savedChannel!.chats[0].name).toBe(payload.name);
  }, 30000);

  it('should create a private channel successfully', async () => {
    const payload = {
      name: 'Private Channel',
      description: 'Canal privado para el equipo',
      team_id: TEAM_ID,
      user_id: CREATOR_ID,
      is_private: true,
      is_deleted: false,
    };
```

Figura 81: Prueba E2E para la creación de un canal principal

Caso E2E: Creación de canal hijo

Valida el canal `channel.create.child.channel`, encargado de asociar un nuevo canal como hijo de uno existente.

- Se valida la existencia del canal padre.
- Se persiste el canal hijo con su relación jerárquica y su propio chat.
- Se lanza error si el canal padre no existe.

```
describe('channel.create.child.channel', () => {
  let parentChannel: Channel;

  beforeEach(async () => {
    parentChannel = await channelRepository.save(
      channelRepository.create({
        name: 'Parent Channel',
        description: 'Canal padre',
        team_id: TEAM_ID,
        user_id: CREATOR_ID,
        is_private: false,
        is_deleted: false,
      }),
    );

    await chatRepository.save(
      chatRepository.create({
        name: parentChannel.name,
        description: parentChannel.description,
        user_id: parentChannel.user_id,
        channel_id: parentChannel.id,
        channel: parentChannel,
      }),
    );
  });

  it('should create a child channel successfully', async () => {
    const payload = {
      name: 'Child Channel',
      description: 'Canal hijo',
      team_id: TEAM_ID,
      user_id: CREATOR_ID,
      is_private: false,
      is_deleted: false,
      parentChannelId: parentChannel.id,
      channel_name: 'Custom Chat Name',
    };
```

Figura 82: Prueba E2E para la creación de un canal hijo

Caso E2E: Carga de mensajes del canal

Prueba el canal `channel.load.messages`, que retorna los mensajes almacenados en un canal específico.

- Se verifica que los mensajes retornados contengan metainformación del usuario.
- Se valida el orden y la estructura de los mensajes.
- Se cubren escenarios vacíos o con canal inexistente.

```
describe('channel.load.channels', () => {
  beforeEach(async () => {
    const channel1 = await channelRepository.save(
      channelRepository.create({
        name: 'General',
        description: 'Canal general',
        team_id: TEAM_ID,
        user_id: CREATOR_ID,
        is_private: false,
        is_deleted: false,
      }),
    );

    const channel2 = await channelRepository.save(
      channelRepository.create({
        name: 'Random',
        description: 'Canal aleatorio',
        team_id: TEAM_ID,
        user_id: CREATOR_ID,
        is_private: true,
        is_deleted: false,
      }),
    );

    await chatRepository.save([
      chatRepository.create({
        name: channel1.name,
        description: channel1.description,
        user_id: channel1.user_id,
        channel_id: channel1.id,
        channel: channel1,
      }),
      chatRepository.create({
        name: channel2.name,
        description: channel2.description,
        user_id: channel2.user_id,
        channel_id: channel2.id,
        channel: channel2,
      }),
    ]);
  });
```

Figura 83: Prueba E2E para la carga de mensajes en un canal

Resultados Globales de las Pruebas E2E del Microservicio Channels

Las pruebas end-to-end (E2E) desarrolladas para el microservicio **Channels** cubren funcionalidades esenciales como la creación de canales principales e hijos, carga de canales por equipo y recuperación de mensajes. Estas pruebas aseguran la correcta persistencia en base de datos, el uso de relaciones entre entidades y la interacción con servicios externos a través de NATS.

A continuación se presenta la salida de consola tras ejecutar la suite de pruebas E2E del microservicio:

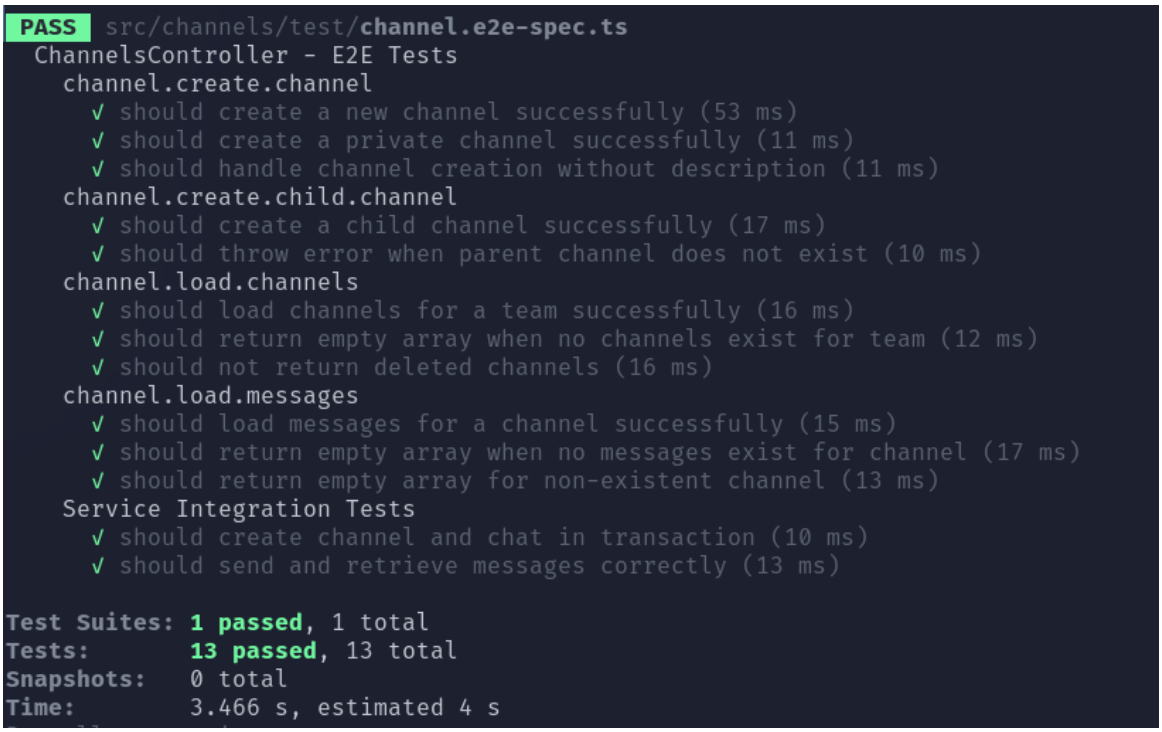

Figura 84: Resultado global de las pruebas E2E ejecutadas en el microservicio Channels.

Resumen de la salida:

- **Test Suites:** 1 suite E2E ejecutada exitosamente.
- **Tests Totales:** 13 pruebas pasaron correctamente, sin errores ni fallos.
- **Snapshots:** No se emplearon pruebas tipo snapshot.
- **Tiempo de ejecución:** 3.466 segundos.

10. Encuesta a Usuarios

Con el propósito de evaluar la experiencia de usuario y la percepción general sobre la aplicación desarrollada para la gestión de proyectos ágiles, se aplicó una encuesta en línea a un grupo de participantes con perfil académico y profesional en el área de TI. Esta sección presenta el diseño del cuestionario, el público al que fue dirigido, el proceso de recolección de información, así como los principales resultados obtenidos.

10.1. Diseño de la Encuesta, Público Objetivo y Recolección de Información

La encuesta fue diseñada como un cuestionario estructurado que combinó preguntas cerradas de tipo escala Likert, casillas de verificación y preguntas abiertas, con el objetivo de obtener retroalimentación detallada sobre diversos aspectos de la aplicación, incluyendo usabilidad, funcionalidades, experiencia general del usuario y comparaciones con otras herramientas similares.

El formulario fue dividido en secciones que abordaban los siguientes temas:

- Facilidad de uso y experiencia de usuario.
- Funcionalidades más valoradas.
- Dificultades encontradas durante el uso.
- Comparación con herramientas similares de gestión ágil.
- Percepción de efectividad en funcionalidades clave como el dashboard y el Planning Poker.
- Recomendaciones y mejoras sugeridas.

Público objetivo: La encuesta estuvo dirigida a personas con experiencia o interés en la gestión de proyectos ágiles y el uso de herramientas tecnológicas en contextos colaborativos. En particular, el público objetivo incluyó:

- Estudiantes universitarios de octavo semestre en adelante en carreras relacionadas con Ingeniería de Sistemas o afines.
- Profesores con conocimientos en metodologías ágiles o gestión de proyectos ágiles.
- Profesionales en el área de TI, incluyendo ingenieros de software y tecnólogos en desarrollo de software.

Proceso de recolección de información: La recolección de datos se realizó mediante un formulario en línea, el cual fue difundido de manera voluntaria a través de canales de comunicación informales y comunidades tecnológicas. Las principales vías utilizadas fueron:

- Recomendación directa (voz a voz).
- Grupos de Discord especializados en desarrollo de software y metodologías ágiles.
- Grupos de WhatsApp y compañeros relacionados en el mundo TI.

El formulario estuvo disponible durante un período de varios días, y se recopilaron un total de 15 respuestas válidas. Se garantizó el anonimato de los participantes y se informó que los resultados serían utilizados únicamente con fines académicos.

10.2. Resultados de la Encuesta

Facilidad de uso y satisfacción general

La mayoría de los usuarios consideró que la aplicación fue fácil de usar. El 33.3 % la calificó con 4 y el 40 % con 2, mientras que solo un 6.7 % indicó que le resultó muy difícil (Figura 85).

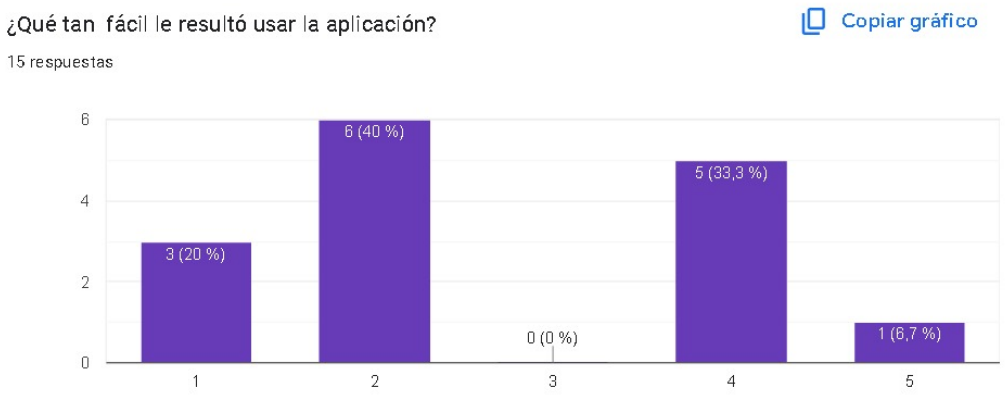

Figura 85: Facilidad de uso percibida por los usuarios

Respecto a la satisfacción general, el 60 % de los usuarios se mostró muy satisfecho con la experiencia de uso y el 40 % satisfecho (Figura 86).

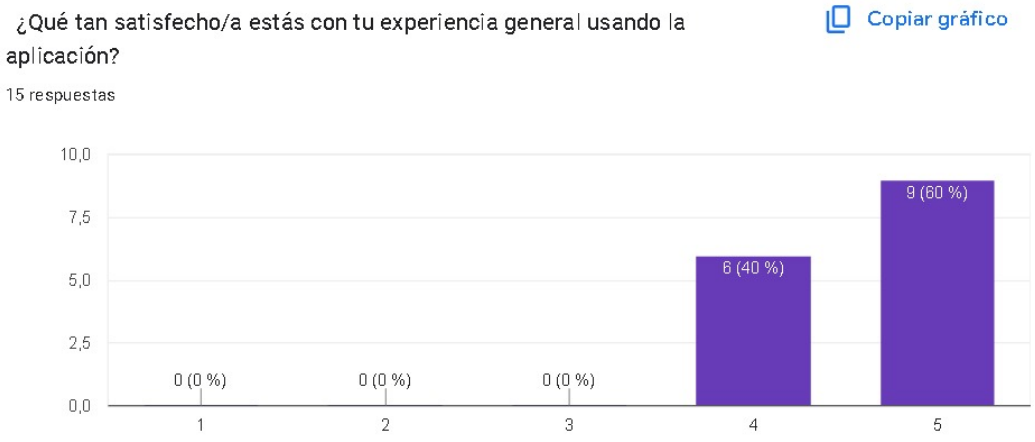

Figura 86: Satisfacción general de los usuarios

Aspectos más valorados

Las funcionalidades más destacadas por los usuarios fueron el chat integrado (73.3 %), la facilidad para crear equipos (66.7 %), la funcionalidad de Planning Poker y la interfaz visual (60 % cada una), seguidas por las sugerencias automáticas con IA (Figura 87).

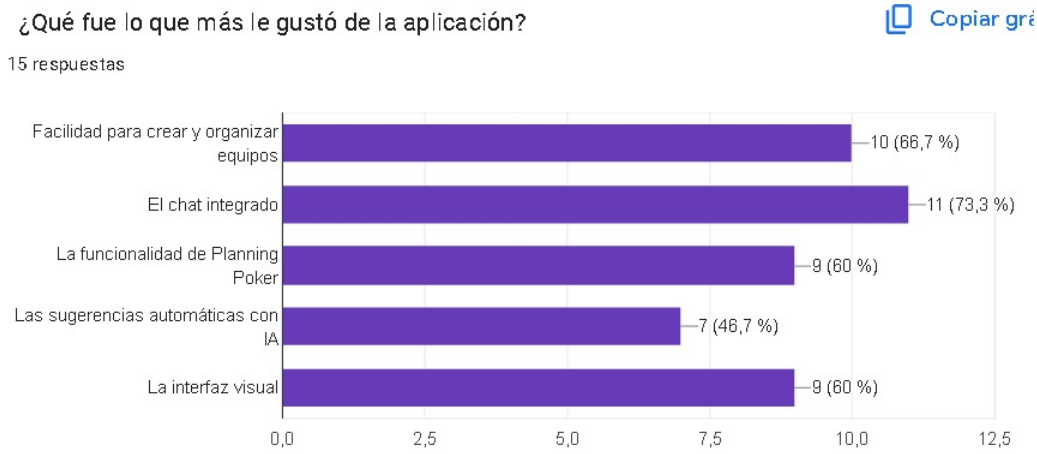

Figura 87: Aspectos más valorados por los usuarios

Dificultades encontradas

La principal dificultad fue no encontrar ciertas funciones fácilmente (75 %). También se registraron casos de dificultad para entender el Planning Poker (25 %) y un caso aislado para cada una de las siguientes situaciones: lentitud, problemas al unirse a un equipo, y dudas con las sugerencias de IA (Figura 88).

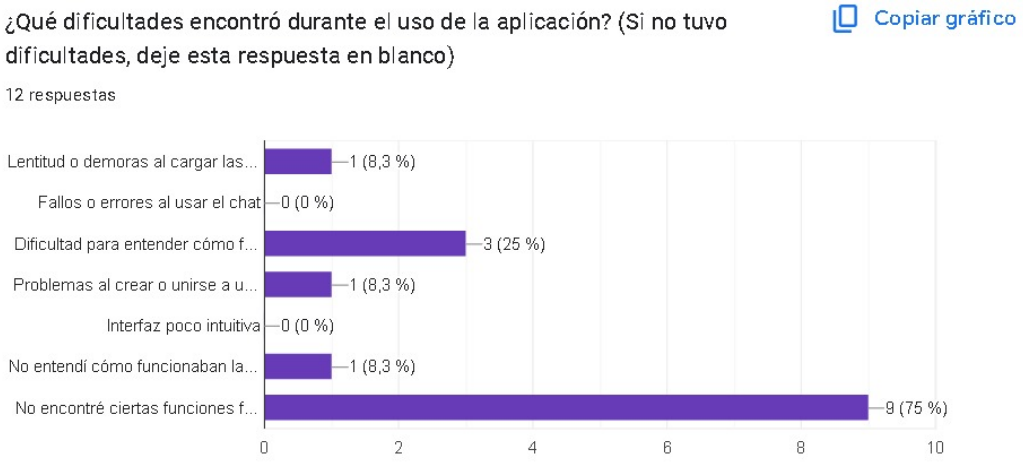

Figura 88: Principales dificultades encontradas durante el uso

Comparación con otras herramientas

Frente a herramientas similares, los usuarios valoraron especialmente el chat sin depender de otras apps (80 %), la inclusión nativa de Planning Poker, las sugerencias con IA y la facilidad de uso (Figura 89).

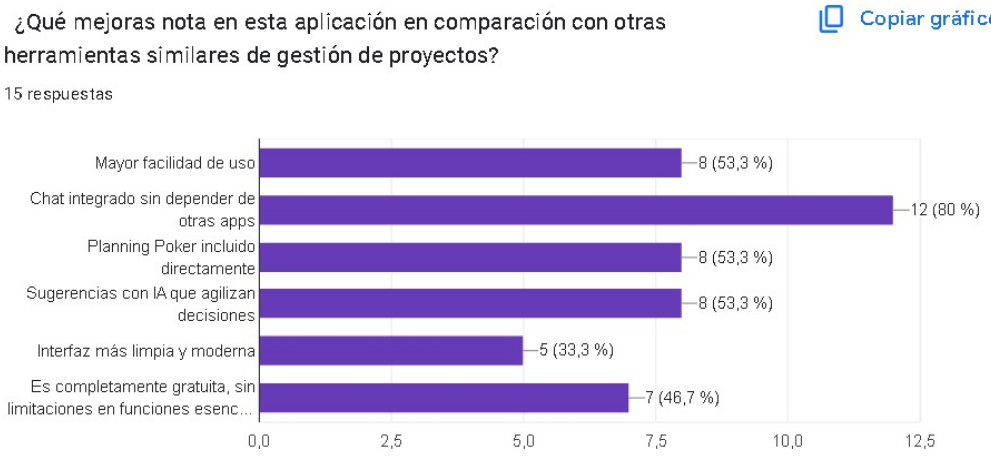

Figura 89: Mejoras en comparación con otras herramientas

Funcionalidades deseadas

En cuanto a nuevas funcionalidades, la más solicitada fue la integración con GitHub (86.7 %), seguida por videollamadas internas (46.7 %) e integración con Slack (40 %) (Figura 90).

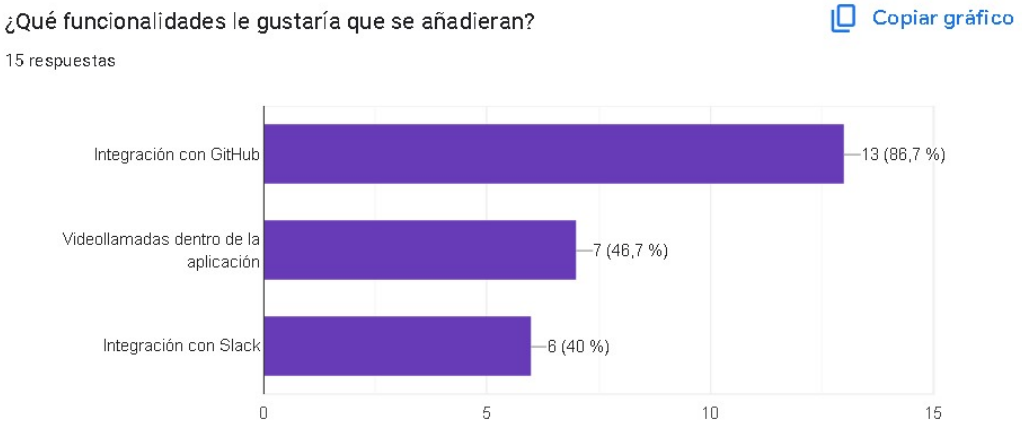

Figura 90: Funcionalidades que los usuarios desearían añadir

Percepción de efectividad

Las gráficas y métricas del dashboard fueron bien recibidas, con un 93.3 % de los usuarios calificándolas con 4 o 5 puntos (Figura 91). De igual forma, el Planning Poker fue valorado como efectivo por el 100 % de los usuarios, con calificaciones de 4 o 5 (Figura 92).

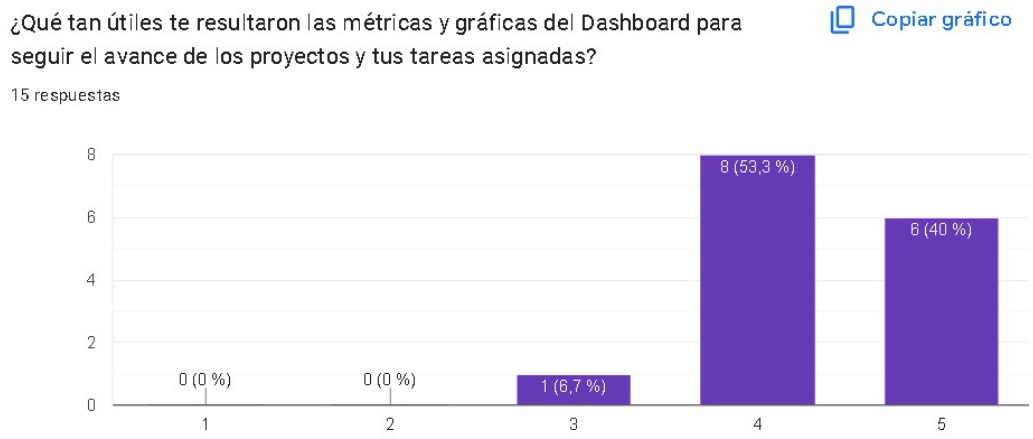

Figura 91: Efectividad del Dashboard

Figura 92: Efectividad del Planning Poker

Intención de recomendación

Finalmente, el 93.3 % de los encuestados indicó que recomendaría la aplicación a otros, lo que representa un fuerte indicador de aceptación (Figura 93).

Figura 93: Intención de recomendar la aplicación

10.3. Conclusiones de la Encuesta

Los resultados obtenidos reflejan una percepción positiva por parte de los usuarios respecto a la aplicación desarrollada. La mayoría consideró que la plataforma es fácil de usar y expresó un alto nivel de satisfacción general.

Se identificaron como fortalezas principales el chat integrado, la funcionalidad de Planning Poker, la creación de equipos y la interfaz visual. Asimismo, se señalaron posibles mejoras, principalmente en la localización de funciones y en la integración con otras herramientas como GitHub y Slack.

Las funcionalidades clave, como el Dashboard y el Planning Poker, fueron valoradas como efectivas, y la alta intención de recomendación indica un buen nivel de aceptación por parte del público objetivo.

11. Conclusiones

Este proyecto ha tenido un gran impacto en el desarrollo profesional del equipo, principalmente por ser la primera aplicación funcional que se logró desarrollar y que, además, tiene el potencial de generar un impacto real. A lo largo del proceso, se mantuvo un constante aprendizaje, ya que inicialmente no se tenía familiaridad con muchas de las herramientas utilizadas, como los frameworks empleados o la arquitectura basada en microservicios. También se aprendió mucho enfrentando los errores y desafíos que se presentaron durante el desarrollo, lo que permitió mejorar las habilidades técnicas y adaptarse mejor a nuevas situaciones.

Uno de los mayores logros fue fortalecer las habilidades técnicas, especialmente en la implementación de arquitecturas con microservicios. Esta decisión no solo brinda flexibilidad al sistema, sino que sienta las bases para un crecimiento sostenible a largo plazo. La arquitectura elegida facilita futuras integraciones, que por cuestiones de tiempo no se pudieron implementar, como la programación de videollamadas o la integración con GitHub para un mejor seguimiento de incidencias relacionadas con commits y ramas de desarrollo. Estas funcionalidades son especialmente valiosas dentro de metodologías ágiles y marcos como Scrum, donde se realizan varias reuniones y el seguimiento del trabajo y la comunicación son aspectos fundamentales.

La aplicación desarrollada se perfila como una alternativa viable para ser utilizada en la institución, sobre todo en cursos como desarrollo de software. Ofrecer a los profesores y estudiantes una herramienta gratuita, con funcionalidades clave y sin depender de múltiples plataformas, puede facilitar tanto la enseñanza como la organización de proyectos. Esta solución aporta valor al centralizar herramientas de planificación, colaboración y seguimiento en un solo entorno.

Además, la integración con inteligencia artificial en la aplicación representa un avance importante. Esta funcionalidad permite analizar datos, ofrecer sugerencias personalizadas y resolver dudas frecuentes que puedan tener los usuarios. El impacto de esto es significativo, ya que puede ahorrar mucho tiempo y mejorar la eficiencia del trabajo en equipo dentro de los proyectos.

Como parte final del proyecto, se aplicó una encuesta a usuarios reales para evaluar la percepción general sobre la aplicación, en la cual se evidenció una recepción mayoritariamente positiva. Los usuarios destacaron especialmente la facilidad de uso de la interfaz, la utilidad del chat integrado y la efectividad del módulo de Planning Poker. También se valoró el hecho de contar con sugerencias inteligentes mediante IA y el acceso a todas las funcionalidades sin restricciones. Sin embargo, también se recibieron algunas observaciones sobre pequeños detalles a mejorar, como una pequeña curva de aprendizaje inicial en algunas secciones.

Con estos aportes se puede evidenciar que la herramienta cumple con los objetivos plan-

teados y valida muchas de las decisiones tomadas durante el desarrollo. Además, esta información servirá como base para seguir mejorando la herramienta en futuras versiones y asegurar que evolucione en función de las necesidades reales de quienes la utilizan.

Trabajos Futuros

Existen varias opciones de mejora y expansión del sistema que podrían abordarse en el futuro. Entre ellas se destacan:

- **Sistema de notificaciones personalizable:** Agregar notificaciones dentro de la aplicación y permitir a los usuarios configurar qué tipos de notificaciones desean recibir (vía correo, en la app, push, etc.).
- **Integración con GitHub:** Implementar la sincronización de issues, commits y pull requests con los elementos del backlog y sprints. Esta integración incluiría la trazabilidad automática entre tareas del proyecto y cambios en el código.
- **Integración de videollamadas:** Implementar dentro de la aplicación una sección donde se pueda crear y planificar videollamadas sin necesidad de recurrir a herramientas externas.
- **Aplicación móvil:** Desarrollar una versión móvil nativa o híbrida para Android e iOS que permita la gestión de proyectos desde dispositivos móviles.
- **Proveer herramientas a los docentes:** Promover el uso de la aplicación dentro de la universidad para que la institución gestione un presupuesto destinado a mejorar la app, especialmente en la infraestructura en la nube que requiere una buena inversión para poder ser usada a gran escala. De esta manera, se podrá brindar a los profesores del programa de Ingeniería de Sistemas una herramienta que facilite la enseñanza de cursos como "Desarrollo de software".
- **Sistema de configuración avanzada de usuario:** Permitir personalizar la experiencia dentro de la app, como idioma, accesos rápidos, o privacidad del perfil.
- **Acceso a perfiles públicos de usuarios:** Incorporar la posibilidad de consultar el perfil de otros usuarios dentro de la aplicación, ya sea al dar clic sobre su avatar en las secciones de equipo o mediante enlaces directos. Esto permitiría visualizar información relevante de cada miembro del proyecto.

Estas mejoras permitirán que el sistema evolucione hacia una plataforma más robusta y adaptable a diversos contextos educativos y profesionales.

12. Anexos

En esta sección se presentan enlaces, capturas y recursos visuales que documentan el desarrollo de la aplicación. Debido a la cantidad de imágenes generadas a lo largo del proyecto, se incluyen únicamente algunas representativas, mientras que el resto pueden consultarse en los enlaces provistos.

Capturas del proceso de desarrollo ágil: Se incluyen algunas imágenes que evidencian el uso de historias de usuario, planificación de sprints, tableros Kanban y tareas desarrolladas durante cada iteración.

Enlace a todas las capturas: `https://docs.google.com/document/d/1mtxqorRAxBq9egMzNXSj5LmHKqJOLA7y/edit?tab=t.0`

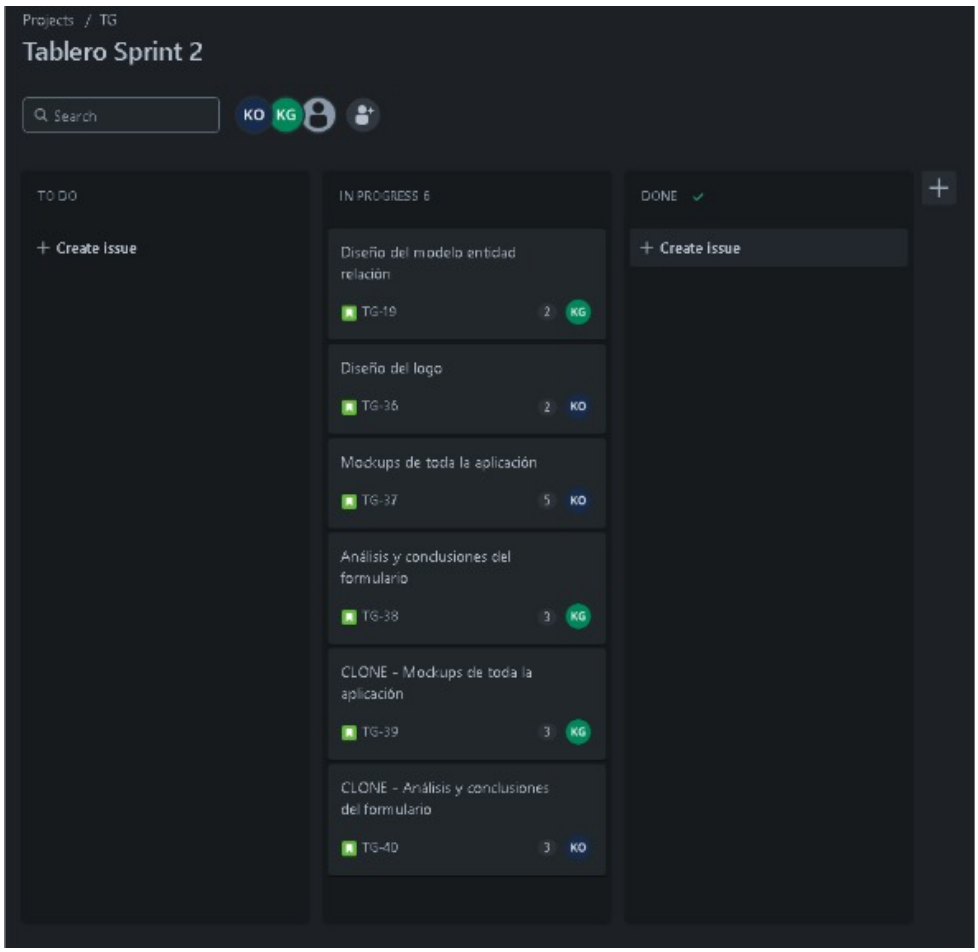

Figura 94: Captura de sprint 2

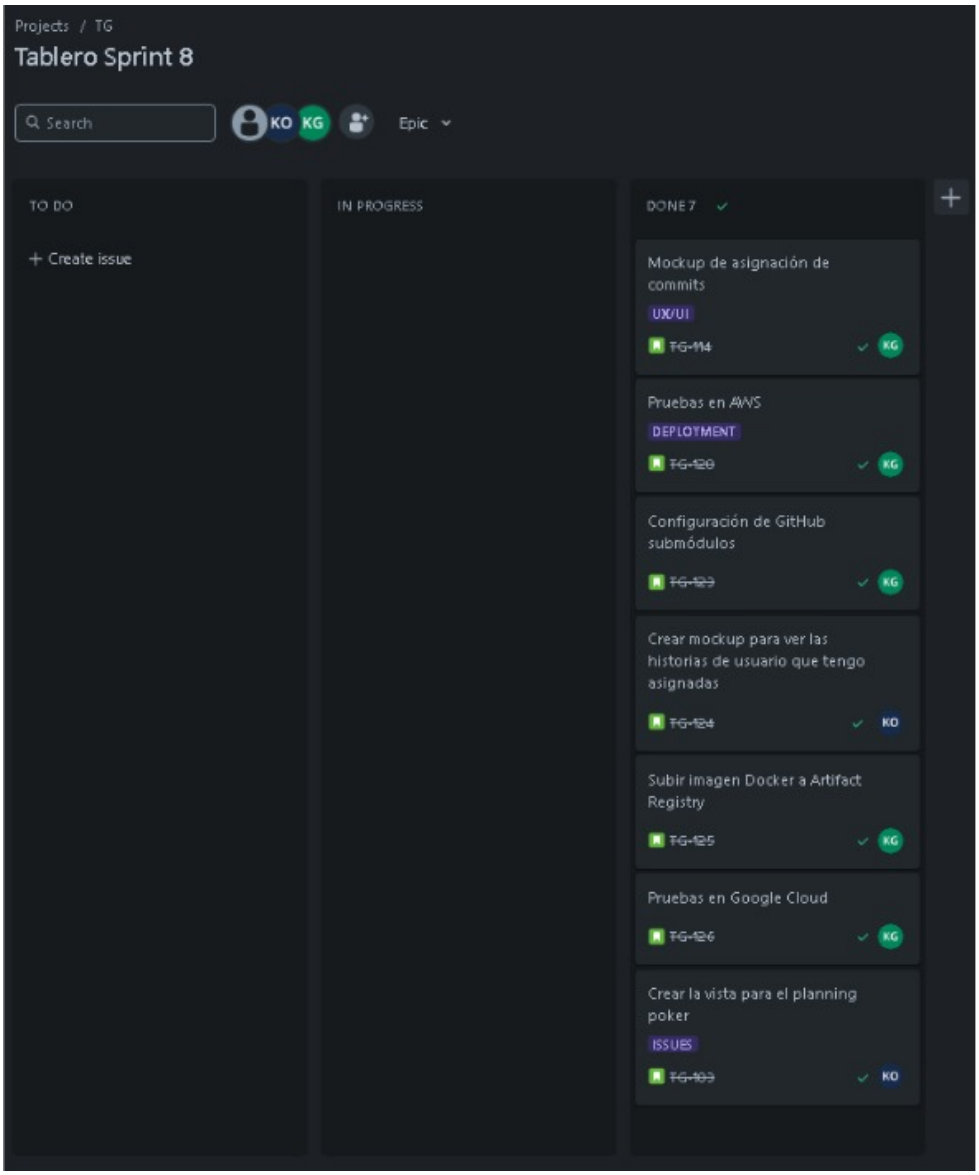

Figura 95: Captura de sprint 8

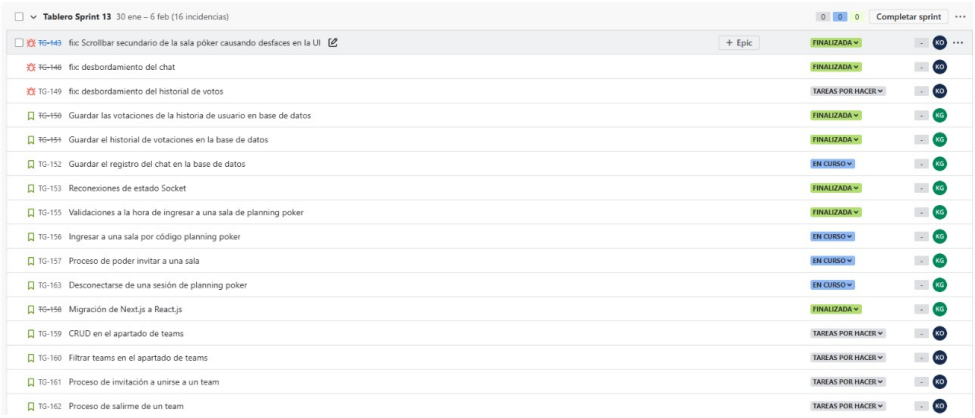

Figura 96: Captura de sprint 13

Diseños de interfaz y evolución visual: Se muestran algunos de los diseños iniciales de la aplicación, incluyendo mockups, wireframes y pantallas completas de los módulos principales.

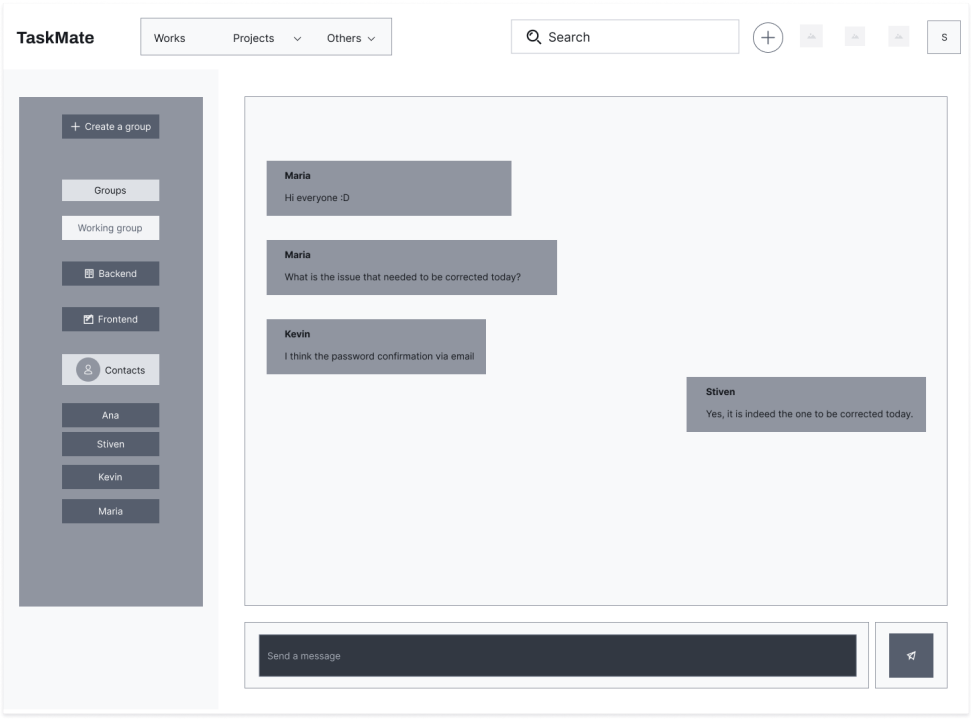

Figura 97: Primer wireframe de chat

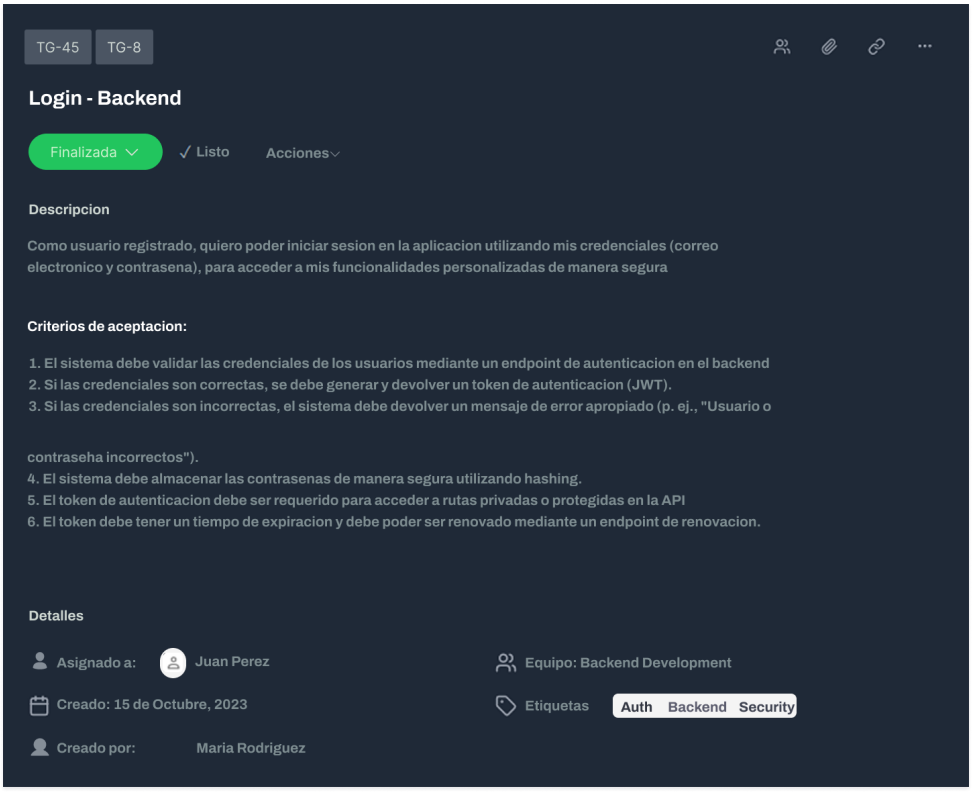

Figura 98: primer vista de detalles de una incidencia

Enlace a recurso completo: `https://www.figma.com/design/HXDQNNKZDK5WVICw0t7Ua8/TaskMate?m=auto&t=Avv2hiqzdqEJjgfw-1`

- **Enlace a la encuesta final completa:** `https://forms.gle/wwxu5ETwdop25xAc7`
- **Enlace a la aplicación:** `https://taskmate.pages.dev/`
- **Repositorio frontend:** `https://github.com/Kahyberth/taskmate`
- **Repositorio client-gateway:** `https://github.com/Kahyberth/client-gateway`
- **Repositorio microservicio projects:** `https://github.com/Kahyberth/projects-ms`
- **Repositorio microservicio auth-ms:** `https://github.com/Kahyberth/auth-ms`
- **Repositorio microservicio planning-poker:** `https://github.com/Kahyberth/planning-poker-ms`
- **Repositorio microservicio channels:** `https://github.com/Kahyberth/channels`

13. Referencias

[1] P. Letelier, C. Penadés, J. Canós, and E. Sánchez, "Metodologías Ágiles en el Desarrollo de Software," de Valencia, Valencia, 2009.

[2] Jira, Planes y precios. Atlassian. [En línea]. Disponible en: https://www.atlassian.com/software/jira/pricing.

[3] E. Equipo editorial, "Teoría de Sistemas - Concepto, principios, autor y enfoque." Accessed: May 01, 2024. [Online]. Available: `https://concepto.de/teoria-de-sistemas/`

[4] A. Cockburn, J. Highsmith, and R. Jeffries, "Manifiesto por el Desarrollo Ágil de Software," *Agilemanifesto.Org*.

[5] K. Schwaber and J. Sutherland, "The Scrum Guide: The Definitive The Rules of the Game," *Scrum.Org and ScrumInc*, no. November, 2017.

[6] "Ventajas y desventajas, características de Jira Software." Accessed: May 26, 2024. [Online]. Available: `https://blog.ganttpro.com/es/caracteristicas-de-jira-software/`

[7] "Alternativas a Jira: 7 herramientas de proyectos alternativas - IONOS España." Accessed: May 26, 2024. [Online]. Available: `https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/alternativas-a-jira/`

[8] E. R. Rönn Ellinor, "Designing a Project Management Application for Agile Software Development," 2016.

[9] T. Juhola, "Customized agile development process for embedded software development." [Online]. Available: https://www.researchgate.net/publication/236960701

[10] A. Stare, "Agile Project Management in Product Development Projects," *Procedia Soc Behav Sci*, vol. 119, pp. 295–304, Mar. 2014, doi: 10.1016/j.sbspro.2014.03.034.

[11] M. I. Lunesu, R. Tonelli, L. Marchesi, and M. Marchesi, "Assessing the risk of software development in agile methodologies using simulation," *IEEE Access*, vol. 9, 2021, doi: 10.1109/ACCESS.2021.3115941.

[12] D. Ciric, B. Lalic, D. Gracanin, N. Tasic, M. Delic, and N. Medic, "Agile vs. Traditional approach in project management: Strategies, challenges and reasons to introduce agile," in *Procedia Manufacturing*, Elsevier B.V., 2019, pp. 1407–1414. doi: 10.1016/j.promfg.2020.01.314.

[13] J. Pócsová, D. Bednárová, G. Bogdanovská, and A. Mojžišová, "Implementation of agile methodologies in an engineering course," Educ Sci (Basel), vol. 10, no. 11, pp. 1–19, Nov. 2020, doi: 10.3390/educsci10110333.

[14] N. Figuerola, "Kanban, Su Uso en el Desarrollo de Software," *J Pers*, 2004.

[15] T. E. I. EL MINISTERIO DE CIENCIA, "Niveles De Madurez Tecnológica (Trl) Y De Manufactura (Mrl)," *Miniciencias*, no. 601, 2021.

[16] M. Georgakalou and K. Koutsikos, "Project management: Lean vs. Agile methodology," *International Conference on Business and Economics - Hellenic Open University*, vol. 1, no. 1, 2023, doi: 10.12681/icbe-hou.5312.