



Facultad de Ingeniería

“Desarrollo de un sistema web para la gestión de proyectos tecnológicos”

Avance de Proyecto Final 2

**Carrera de Ingeniería de Software
Desarrollo Full Stack (35187)**

DOCENTE

JAIME JAVIER DE LA TORRE VASQUEZ

INTEGRANTES

Anghelina Jhuliet de la Victoria Alva Encinas (**U22233667**)

Diego Fernando Napa Alvarez (**U22234939**)

Jorge Elias Castañeda Bardales (**U22213144**)

Josue Abrham Ayala Valladolid (**U22235461**)

Leonardo Manuel Justo Jurado (**U22203888**)

Lima, Perú - 2025

ÍNDICE

| | |
|---|-----------|
| 1. Introducción..... | 3 |
| 1.1. Descripción de proyecto..... | 3 |
| 1.2. Planteamiento del problema..... | 4 |
| 1.3. Objetivos..... | 5 |
| 1.3.1. Objetivo general..... | 5 |
| 1.3.2. Objetivos específicos..... | 5 |
| 1.4. Justificación..... | 5 |
| 1.5. Alcances y limitaciones..... | 5 |
| 1.5.1. Alcances..... | 5 |
| 1.5.2. Limitaciones..... | 6 |
| 2. Marco teórico..... | 6 |
| 2.1. Antecedentes bibliográficos..... | 6 |
| 2.2. Bases teóricas..... | 9 |
| 3. Diseño de la solución..... | 12 |
| 3.1. Requerimientos funcionales y no funcionales..... | 12 |
| 3.1.1. Requerimientos funcionales..... | 12 |
| 3.1.2. Requerimientos no funcionales..... | 12 |
| 3.2. Casos de uso y actores..... | 13 |
| 3.2.1. Actores..... | 13 |
| a. Administrador de Proyecto..... | 13 |
| c. Miembro del Equipo / Colaborador..... | 13 |
| d. Observador / Stakeholder..... | 14 |
| f. Analista / Reporteador..... | 14 |
| 3.2.2. Casos de uso..... | 15 |
| 3.3. Diagramas de casos de uso..... | 18 |
| 4. Avance de desarrollo frontend..... | 20 |
| 4.1. Descripción técnica..... | 20 |
| 4.2. Wireframes..... | 22 |
| Figura 10..... | 23 |
| Apartado de inicio de sesión de Kuska elaborado en Figma..... | 23 |
| Figura 11..... | 23 |
| Apartado del dashboard de Kuska elaborado en Figma..... | 23 |
| 4.3. Páginas implementadas..... | 24 |
| 4.4. Diseño responsive..... | 27 |
| 5. Avance de desarrollo backend..... | 29 |
| 5.1. Arquitectura y enfoque de desarrollo..... | 29 |
| 5.2. Stack tecnológico y justificación..... | 30 |
| 5.3. Componentes de infraestructura y herramientas..... | 30 |
| 5.4. Estructura modular del sistema..... | 31 |

| | |
|--|-----------|
| 5.5. Patrones arquitectónicos implementados..... | 31 |
| 6. Conclusiones preliminares..... | 32 |
| 7. Bibliografía..... | 33 |

1. Introducción

1.1. Descripción de proyecto

El proyecto Kuska se concibe como una plataforma web integral orientada a la gestión de proyectos y la colaboración en equipo. Su principal objetivo es proporcionar a organizaciones y grupos de trabajo un espacio unificado donde sea posible planificar, organizar y supervisar proyectos de manera estructurada, al mismo tiempo que se fomenta una comunicación eficiente entre los integrantes.

El sistema permitirá la creación de proyectos personalizados, en los que se podrán establecer fases de trabajo, asignar responsables, definir prioridades y dar seguimiento al progreso mediante tableros y reportes dinámicos. Con estas herramientas, se busca que los usuarios tengan un panorama claro del estado de sus proyectos y puedan tomar decisiones informadas.

Un aspecto central del sistema es la gestión de equipos por proyecto. Cada proyecto contará con un espacio propio en el que se podrán conformar grupos de trabajo diversos, definir roles específicos y asignar permisos diferenciados según las responsabilidades de cada integrante. Asimismo, cada equipo dispondrá de un canal de comunicación interna en tiempo real, diseñado para facilitar el intercambio de ideas, el envío de archivos y la coordinación de actividades de manera ágil y organizada.

En suma, Kuska se plantea como una solución tecnológica que integra en un mismo espacio la planificación de proyectos, la gestión de equipos, la comunicación en tiempo real y el análisis de resultados. Con ello, se espera ofrecer una herramienta robusta, adaptable y escalable que responda a las exigencias actuales de los entornos de trabajo colaborativos, contribuyendo al logro de objetivos de manera más ordenada y eficiente.

1.2. Planteamiento del problema

En la actualidad, los equipos de trabajo enfrentan el desafío de coordinar actividades, tareas y responsabilidades en entornos cada vez más dinámicos y exigentes. Aunque existen múltiples herramientas digitales orientadas a la gestión de proyectos o a la comunicación interna, estas suelen estar fragmentadas, lo que genera dispersión de la información, duplicación de esfuerzos y dificultades para mantener un control eficiente de los avances.

Una problemática recurrente es que la planificación y el seguimiento de proyectos no siempre están integrados con los canales de comunicación del equipo, lo cual ocasiona retrasos en la toma de decisiones, pérdida de datos importantes y disminución en la productividad general. Adicionalmente, muchos sistemas no ofrecen un espacio centralizado donde sea posible evaluar de forma simultánea el progreso de las tareas, el desempeño individual y colectivo, así como el cumplimiento de objetivos.

Esta situación se hace aún más evidente en contextos educativos y académicos, donde los estudiantes universitarios suelen trabajar en proyectos grupales. La falta de plataformas que combinen gestión estructurada de proyectos y comunicación en tiempo real dificulta la organización de equipos estudiantiles, generando descoordinación, tareas incompletas y dificultades para evidenciar avances de manera clara. Como consecuencia, tanto en entornos profesionales como académicos, se observa la necesidad de contar con una herramienta que permita centralizar la planificación, la colaboración y la evaluación en un solo espacio digital.

1.3. Objetivos

1.3.1. Objetivo general

Desarrollar una plataforma web integral denominada **Kuska**, que permita la gestión eficiente de proyectos y equipos de trabajo mediante la centralización de la planificación, la comunicación en tiempo real y el análisis de resultados

1.3.2. Objetivos específicos

- Diseñar un sistema de gestión de proyectos que permita la creación, personalización y seguimiento de proyectos.
- Implementar un módulo de gestión de equipos, que contemple la creación de grupos por proyecto, la asignación de roles y permisos diferenciados.
- Integrar un sistema de comunicación interna en tiempo real, que facilite el intercambio de mensajes, archivos y notificaciones.
- Incorporar herramientas de análisis y métricas de desempeño, orientadas a evaluar el progreso individual y colectivo.
- Garantizar la usabilidad y accesibilidad de la plataforma, mediante un diseño responsive y una interfaz intuitiva que favorezca la experiencia del usuario en distintos dispositivos y contextos.

1.4. Justificación

1.5. Alcances y limitaciones

1.5.1. Alcances

- Creación y personalización de proyectos con fases, prioridades y responsables.
- Visualización del progreso mediante tableros dinámicos y reportes.
- Conformación de equipos con roles y permisos diferenciados.

- Espacios propios para cada proyecto, garantizando organización y privacidad.
- Comunicación en tiempo real con mensajería interna, notificaciones y compartición de archivos.
- Herramientas de análisis y métricas para evaluar desempeño individual y colectivo.
- Interfaz intuitiva y responsive, accesible desde navegadores web sin necesidad de instalación.
- Centralización de toda la información de planificación, ejecución y seguimiento en un solo lugar.

1.5.2. Limitaciones

- No incluye funcionalidades de contabilidad, facturación o gestión financiera avanzada.
- No integra videoconferencias ni inteligencia artificial en su primera versión.
- Requiere conexión a internet, sin modo offline ni sincronización automática.
- Limitaciones en almacenamiento, usuarios simultáneos y proyectos según la infraestructura.
- Orientado principalmente a equipos pequeños o medianos..
- Seguridad limitada a funciones básicas, sin autenticación de dos factores ni cifrado avanzado.
- Su adopción dependerá de la disposición de los usuarios y de la calidad de la conexión a internet.
- La escalabilidad estará condicionada por los recursos técnicos y humanos disponibles.

2. Marco teórico

2.1. Antecedentes bibliográficos

En el campo de la gestión de proyectos tecnológicos, las metodologías ágiles han cobrado gran relevancia debido a su capacidad para adaptarse a entornos

cambiantes, mejorar la comunicación entre los equipos y optimizar los resultados en la entrega de productos o servicios. Como consecuencia, diversas investigaciones han explorado el diseño y desarrollo de aplicaciones web que faciliten la planificación, el control y el seguimiento de proyectos, basándose en enfoques ágiles como Scrum y Kanban.

A continuación, se presentan dos estudios que constituyen antecedentes importantes para el desarrollo de un sistema web de gestión de proyectos tecnológicos.

El primer antecedente corresponde al trabajo de Francia Del Busto (2023), titulado **“Aplicación web para la gestión de proyectos de software usando Scrum”**. En este estudio se plantea el diseño de una plataforma web orientada a apoyar la gestión de proyectos bajo el marco de trabajo de Scrum. La investigación identifica que muchas empresas presentan dificultades para cumplir con plazos y presupuestos, y plantea que la aplicación de metodologías ágiles puede reducir estas limitaciones. La solución propuesta contempla funcionalidades como la gestión del backlog, la planificación de sprints, el seguimiento de tareas y el control de roles específicos (Scrum Master, Product Owner y equipo de desarrollo). Asimismo, el autor destaca que Scrum permite iteraciones cortas, retroalimentación constante y una comunicación más eficiente entre los miembros del equipo. No obstante, se señalan algunas limitaciones, como la falta de integración con herramientas externas y la necesidad de mayor análisis sobre la escalabilidad del sistema, lo que deja espacio para el perfeccionamiento de futuras aplicaciones similares.

Figura 1.

Portada de la tesis "Aplicación web para la gestión de proyectos de software usando Scrum".



UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS

FACULTAD DE INGENIERÍA

PROGRAMA ACADÉMICO DE INGENIERÍA DE SOFTWARE

Aplicación web para la gestión de proyectos de software usando Scrum

TESIS

Para optar el título profesional de Ingeniero de Software

AUTORES

Francia Del Busto, Pablo Josué (0009-0003-9803-0418)

Ayme Tamara, Rodson Vladimir (0009-0004-0364-540X)

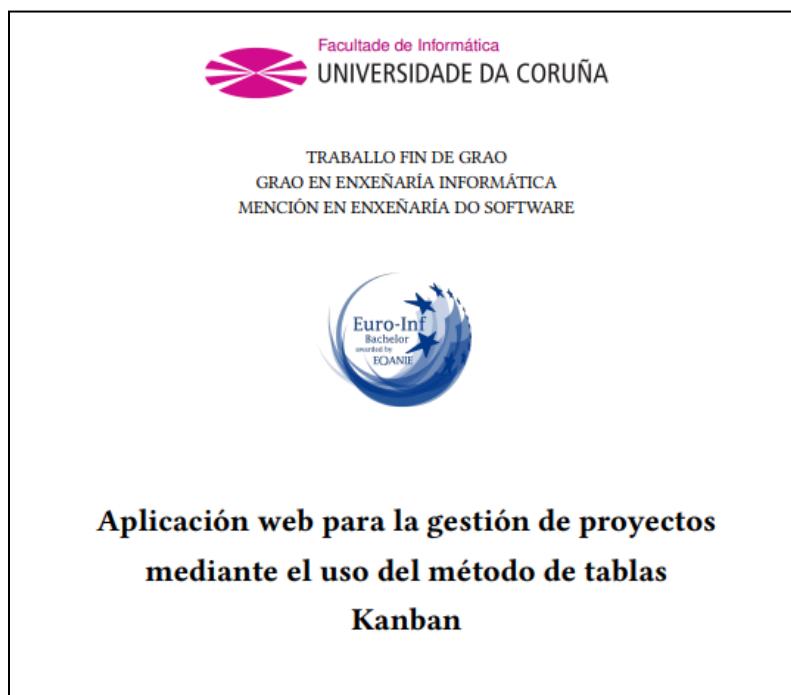
Nota. Adaptado de *Aplicación web para la gestión de proyectos de software usando Scrum* de F. Del Busto (2023), Repositorio Académico de la UPC.
[https://repositorioacademico.upc.edu.pe/bitstream/handle/10757/669753/](https://repositorioacademico.upc.edu.pe/bitstream/handle/10757/669753/Francia_BP.pdf?sequence=1&isAllowed=y)
Francia_BP.pdf?sequence=1&isAllowed=y

Por otro lado, el trabajo de Fernández Torreira (2024), denominado “**Aplicación web para la gestión de proyectos mediante el uso del método de tablas Kanban**”, propone un sistema basado en la visualización de tareas a través de tableros Kanban. La investigación destaca que este método permite organizar el flujo de trabajo en columnas que representan distintos estados, limitando el trabajo en curso y facilitando la detección de cuellos de botella. La aplicación desarrollada incorpora funciones como la creación y personalización de tableros, el arrastre y asignación de tareas, la gestión de usuarios y la incorporación de métricas visuales sobre el rendimiento del equipo. A diferencia de Scrum, el enfoque Kanban se centra en un flujo continuo y flexible, lo cual resulta especialmente útil para proyectos que requieren adaptaciones frecuentes sin planificaciones rígidas. Sin embargo, el autor reconoce limitaciones, como la falta

de integración con metodologías híbridas y la ausencia de análisis predictivos avanzados.

Figura 2.

Portada de la tesis "Aplicación web para la gestión de proyectos mediante el uso del método de tablas Kanban".



Nota. Adaptado de *Aplicación web para la gestión de proyectos mediante el uso del método de tablas Kanban* de D. Fernández Torreira (2024), Repositorio de la Universidad da Coruña. <https://ruc.udc.es/rest/api/core/bitstreams/709e2bc2-6cdf-4bf4-90ec-0999981f7639/content>

2.2. Bases teóricas

Node

Node.js es un entorno de ejecución de JavaScript del lado del servidor que se caracteriza por su modelo asíncrono, event-driven y de I/O no bloqueante, lo que le permite manejar múltiples conexiones concurrentes con eficiencia, siendo especialmente útil en aplicaciones web escalables y en tiempo real (Chéc &

Nowak, 2019). Además, estudios recientes señalan la necesidad de mejorar la seguridad de aplicaciones en Node.js debido a vulnerabilidades como la *prototype pollution*, lo que resalta la importancia de verificaciones estáticas y dinámicas en su uso (Bogner & Merkel, 2022). En proyectos como Kuska, Node.js resulta ideal como backend para gestionar API, autenticación y comunicación en tiempo real.

React

React es una biblioteca JavaScript basada en componentes reutilizables y en el uso del Virtual DOM, el cual optimiza el rendimiento de las interfaces al actualizar únicamente los elementos necesarios en lugar de renderizar toda la página (Chéć & Nowak, 2019). Esta aproximación mejora la interactividad en aplicaciones dinámicas. Con la introducción de Hooks como useState y useEffect, React simplifica la gestión de estado y ciclo de vida en componentes funcionales, mejorando la legibilidad y mantenibilidad del código (Goli, 2019). Sin embargo, estudios también advierten que aplicaciones a gran escala enfrentan retos de rendimiento, como re-renders innecesarios o sincronización de datos, recomendando técnicas como lazy loading y memoización para optimizar su desempeño (Gu, 2025).

TypeScript

TypeScript es un superset de JavaScript que incorpora tipado estático, interfaces y comprobaciones en tiempo de compilación, lo que contribuye a la detección temprana de errores y a la mejora en la mantenibilidad del software.

Investigaciones han demostrado que los proyectos desarrollados con TypeScript en GitHub presentan mejor calidad de código y menos complejidad cognitiva en comparación con los de JavaScript puro (Bogner & Merkel, 2022). Asimismo, existen motores de análisis estático diseñados para TypeScript que permiten evaluar métricas de calidad y mantener la fiabilidad del código (Göktürk & Koç, 2022). En el caso de Kuska, TypeScript brinda mayor seguridad y escalabilidad al definir contratos claros de datos entre frontend y backend.

Tailwind

Tailwind CSS es un framework de CSS con enfoque utility-first, que ofrece clases pequeñas reutilizables (por ejemplo p-4, text-center, bg-blue-500) para aplicar estilos directamente en el marcado, reduciendo la necesidad de escribir CSS personalizado. Su flexibilidad permite construir interfaces responsivas de forma rápida, aunque estudios sobre la usabilidad de su documentación indican que los usuarios principiantes pueden experimentar cierta dificultad debido a la amplitud de clases y a la organización de la información (Nopriani & Muhammad, 2024). Aun así, investigaciones comparativas muestran que Tailwind ofrece ventajas frente a otros frameworks similares, como mayor velocidad de desarrollo y una comunidad activa que lo respalda (S., Usha Sree & Mohan, 2024).

Git y Github

El control de versiones es esencial en el desarrollo colaborativo de software. Git, como sistema de control de versiones distribuido, permite a los desarrolladores mantener un historial completo de cambios, trabajar de forma paralela en ramas y fusionar aportes sin perder información (Seref & Tanrıover, 2016). GitHub complementa a Git al ofrecer un entorno en la nube que facilita la colaboración mediante *pull requests*, gestión de incidencias y control de acceso (Molnar & Motogna, 2020). En Kuska, Git y GitHub aseguran un flujo de trabajo ordenado y colaborativo que favorece la trazabilidad del proyecto.

Figma

Figma es una herramienta de diseño colaborativo en la nube que permite crear interfaces de usuario, prototipos interactivos y sistemas de diseño, facilitando la comunicación entre diseñadores y desarrolladores. Su principal ventaja es el trabajo en tiempo real, lo cual promueve la sincronización entre miembros del equipo y reduce las inconsistencias en la implementación de interfaces (Nopriani & Muhammad, 2024). En Kuska, Figma funciona como la base visual que se traduce en componentes de React estilizados con Tailwind CSS.

3. Diseño de la solución

3.1. Requerimientos funcionales y no funcionales

3.1.1. Requerimientos funcionales

| Item | Descripción |
|------|--|
| RF1 | El sistema debe permitir la creación de proyectos personalizados con fases, responsables, prioridades y fechas límite. |
| RF2 | El sistema debe permitir la visualización del progreso de los proyectos mediante tableros dinámicos. |
| RF3 | El sistema debe permitir la conformación de equipos por proyecto, con roles y permisos diferenciados para cada integrante. |
| RF4 | El sistema debe proporcionar espacios propios para cada proyecto, garantizando la organización y privacidad de la información. |
| RF5 | El sistema debe integrar un canal de comunicación interna en tiempo real que permita el intercambio de mensajes y archivos. |
| RF6 | El sistema debe centralizar toda la información de planificación, ejecución y seguimiento en un solo espacio digital. |

3.1.2. Requerimientos no funcionales

| Item | Descripción |
|------|--|
| RNF1 | El sistema debe ser accesible únicamente con conexión a internet, sin modo offline. |
| RNF2 | El sistema debe garantizar tiempos de respuesta adecuados para la carga de proyectos y visualización de tableros (menores a 3 segundos en promedio bajo carga normal). |
| RNF3 | El sistema debe ser responsive, adaptándose a diferentes dispositivos (PC, laptop, tablet y smartphone). |
| RNF4 | El sistema debe garantizar un nivel básico de seguridad, incluyendo autenticación de usuarios y permisos diferenciados. |

| | |
|------|---|
| RNF5 | El sistema no debe incluir funcionalidades de contabilidad, facturación ni gestión financiera. |
| RNF6 | El sistema debe contar con un límite en el número de usuarios simultáneos y proyectos activos, de acuerdo con la capacidad de la infraestructura inicial. |

3.2. Casos de uso y actores

3.2.1. Actores

a. Administrador de Proyecto

- **Descripción:** Usuario con privilegios máximos dentro de un proyecto específico
- **Responsabilidades:**
 - Crear y configurar proyectos
 - Definir fases de trabajo
 - Establecer prioridades del proyecto
 - Gestionar la estructura general del proyecto
 - Configurar tableros y reportes
 - Asignar y revocar permisos a otros usuarios

b. Gestor de Equipo / Líder de Equipo

- **Descripción:** Usuario responsable de coordinar un equipo dentro de un proyecto
- **Responsabilidades:**
 - Conformar grupos de trabajo
 - Asignar roles específicos a miembros
 - Distribuir tareas entre colaboradores
 - Supervisar el progreso del equipo
 - Coordinar actividades del equipo
 - Gestionar la comunicación interna del equipo

c. Miembro del Equipo / Colaborador

- **Descripción:** Usuario que participa activamente en uno o más proyectos
- **Responsabilidades:**
 - Ejecutar tareas asignadas
 - Actualizar el estado de sus actividades
 - Comunicarse con otros miembros del equipo
 - Compartir archivos e información
 - Consultar el estado del proyecto
 - Reportar avances

d. Observador / Stakeholder

- **Descripción:** Usuario con acceso de solo lectura o limitado al proyecto
- **Responsabilidades:**
 - Visualizar el progreso del proyecto
 - Consultar reportes y métricas
 - Monitorear el desempeño general
 - Acceder a información relevante sin capacidad de modificación

e. Administrador del Sistema

- **Descripción:** Usuario con privilegios técnicos sobre toda la plataforma
- **Responsabilidades:**
 - Gestionar la plataforma a nivel global
 - Administrar usuarios y organizaciones
 - Configurar parámetros del sistema
 - Supervisar el funcionamiento técnico
 - Gestionar seguridad y accesos

f. Analista / Reporteador

- **Descripción:** Usuario enfocado en el análisis de métricas y desempeño

- **Responsabilidades:**

- Generar reportes de progreso
- Analizar métricas de desempeño individual y colectivo
- Evaluar indicadores del proyecto
- Crear dashboards personalizados
- Proporcionar insights para toma de decisiones

3.2.2. Casos de uso

- a. CU-01: Gestionar usuarios y organizaciones — Actor: Administrador del Sistema. Resumen: Crear, editar, eliminar y auditar usuarios y organizaciones; resultado: usuarios y organizaciones administradas y logs registrados.
- b. CU-02: Configurar parámetros del sistema — Actor: Administrador del Sistema. Resumen: Ajustar cuotas, integraciones y personalización global; resultado: parámetros aplicados y sistema configurado.
- c. CU-03: Administrar seguridad — Actor: Administrador del Sistema. Resumen: Gestionar autenticación, políticas, logs y respaldos; resultado: políticas de seguridad y auditoría activas.
- d. CU-04: Crear proyecto — Actor: Administrador de Proyecto. Resumen: Inicializar un proyecto con plantilla, fechas, visibilidad y configuración base; resultado: espacio de proyecto creado y dashboard inicial.
- e. CU-05: Configurar fases de trabajo — Actor: Administrador de Proyecto. Resumen: Definir etapas, dependencias y criterios de aceptación; resultado: workflow del proyecto establecido.

- f. CU-06: Establecer prioridades — Actor: Administrador de Proyecto. Resumen: Definir niveles de prioridad y reglas de escalamiento; resultado: prioridades configuradas disponibles en tareas.
- g. CU-07: Gestionar tableros — Actor: Administrador de Proyecto. Resumen: Crear y personalizar tableros (Kanban, Scrum, etc.) y automatizaciones; resultado: tableros operativos para visualización y gestión.
- h. CU-08: Conformar grupos de trabajo — Actor: Líder de Equipo. Resumen: Crear equipos, invitar miembros y activar canales; resultado: equipos operativos y canales de comunicación.
- i. CU-09: Asignar roles y permisos — Actor: Administrador de Proyecto / Líder. Resumen: Asignar y personalizar roles con permisos y vigencias; resultado: control de acceso por rol aplicado.
- j. CU-10: Distribuir tareas — Actor: Líder de Equipo. Resumen: Asignar responsables, prioridades, estimaciones y dependencias; resultado: tareas asignadas y notificaciones enviadas.
- k. CU-11: Coordinar actividades — Actor: Líder de Equipo. Resumen: Programar reuniones, hitos y balancear carga; resultado: actividades sincronizadas y calendario actualizado.
- l. CU-12: Ejecutar tareas asignadas — Actor: Miembro del Equipo. Resumen: Trabajar en tareas, registrar tiempo, subir entregables y marcar completadas; resultado: progreso y entregables registrados.

- m. CU-13: Actualizar estado de tareas — Actor: Miembro del Equipo.
Resumen: Cambiar estados según workflow, registrar historial y notificar; resultado: estados y auditoría de cambios actualizados.
- n. CU-14: Comunicarse en tiempo real — Actor: Miembro / Líder.
Resumen: Chat, hilos, compartir archivos y llamadas; resultado: comunicación instantánea y mensajes persistentes.
- o. CU-15: Compartir archivos — Actor: Miembro del Equipo.
Resumen: Subir, versionar y otorgar permisos a documentos; resultado: repositorio de archivos con control de versiones y permisos.
- p. CU-16: Generar reportes de progreso — Actor: Analista.
Resumen: Crear reportes personalizados (PDF/Excel), programar envíos; resultado: reportes generados y distribuidos.
- q. CU-17: Analizar métricas de desempeño — Actor: Analista.
Resumen: Examinar productividad, calidad y cumplimiento; resultado: insights, alertas y métricas configurables.
- r. CU-18: Crear dashboards — Actor: Analista. Resumen: Diseñar paneles con widgets y permisos de acceso; resultado: dashboards interactivos para stakeholders.
- s. CU-19: Evaluar indicadores — Actor: Analista. Resumen: Calcular KPIs, estado (verde/amarillo/rojo) y recomendaciones; resultado: evaluaciones documentadas y planes de acción.
- t. CU-20: Visualizar progreso — Actor: Observador / Stakeholder.
Resumen: Consultar dashboard en solo lectura y timeline;

resultado: visión del estado del proyecto sin modificaciones.

- u. CU-21: Consultar reportes — Actor: Observador / Stakeholder.
Resumen: Acceder y descargar reportes disponibles; resultado: stakeholders informados con reportes exportables.
- v. CU-22: Monitorear desempeño — Actor: Observador / Stakeholder. Resumen: Supervisar KPIs en tiempo real, recibir alertas y comparar periodos; resultado: monitoreo continuo y alertas configuradas.

3.3. Diagramas de casos de uso

Figura 3.

Diagrama de la administración del sistema con los siguientes casos de uso: CU-01, CU-02, CU-03.

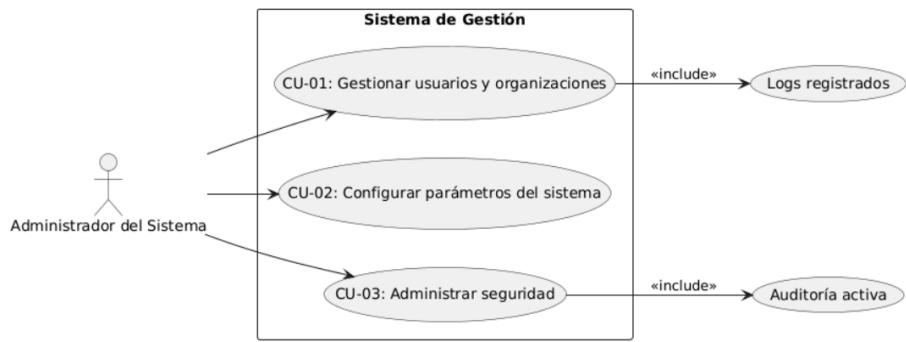


Figura 4.

Diagrama de administración del proyecto con los siguientes casos de uso: CU-04, CU-05, CU-06, CU-07, CU-09.

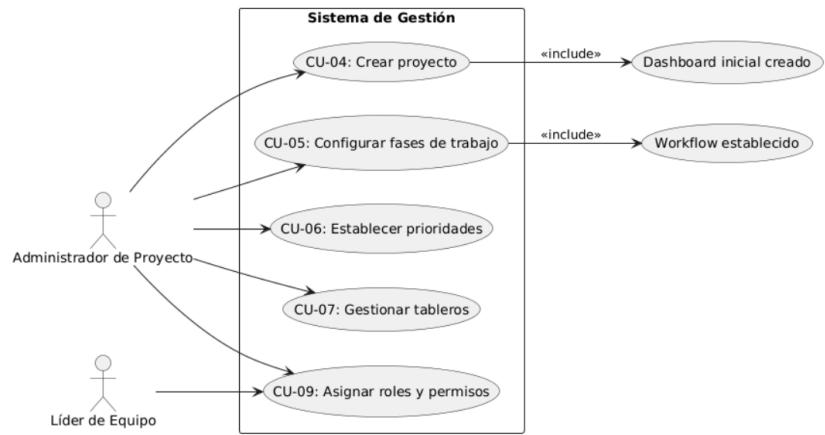


Figura 5.

Diagrama de la gestión de equipos con los siguientes casos de uso:
CU-08, CU-09, CU-10, CU-11.

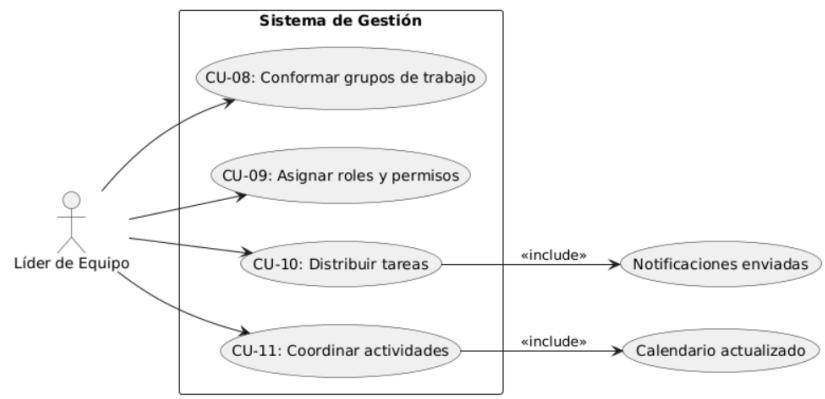


Figura 6.

Diagrama de la ejecución operativa con los siguientes casos de uso:
CU-15, CU-12, CU-13, CU-14

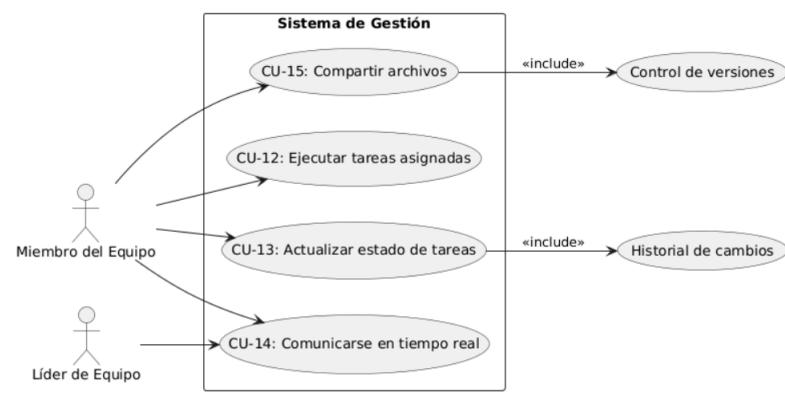


Figura 7.

Diagrama de los análisis y reportes con los siguientes casos de uso:
CU-16, CU-17, CU-18, CU-19.

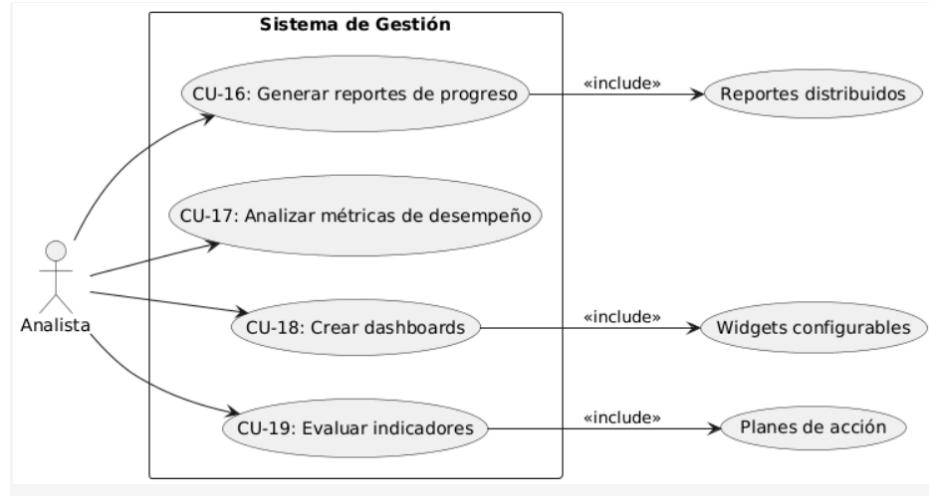
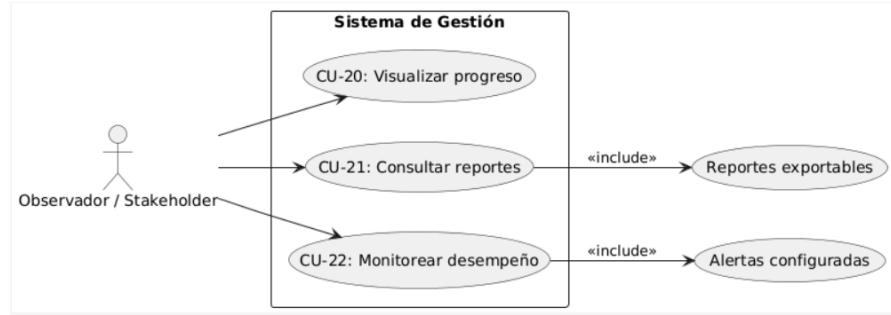


Figura 8.

Diagrama del seguimiento y monitoreo con los siguientes casos de uso: CU-20, CU-21, CU-22.



4. Avance de desarrollo frontend

4.1. Descripción técnica

El **frontend** del sistema **Kuska** se encuentra diseñado bajo un enfoque moderno y escalable que responde a las necesidades de usabilidad, rendimiento y mantenibilidad de aplicaciones web de mediana y gran envergadura. Para su construcción se emplea el ecosistema **React** como biblioteca principal para la creación de interfaces de usuario, complementado con **Vite** como herramienta de construcción y empaquetado, lo que garantiza tiempos de compilación reducidos, recarga en caliente y un entorno de desarrollo altamente optimizado. Asimismo, se hace uso de **TypeScript**, lo que permite disponer de tipado estático, detección temprana de errores y una mayor robustez en la definición de los modelos de datos y contratos del sistema.

En cuanto a la **organización del código fuente**, se adopta el principio de **Screaming Architecture**, lo cual significa que la estructura del proyecto refleja de manera explícita los **dominios de negocio** y módulos funcionales del sistema, tales como: *Gestión de Plataforma, Gestión de Proyectos, Gestión de Equipos, Ejecución y Colaboración, Análisis y Reportes, y Monitoreo*. Bajo esta organización, cada dominio incluye sus propias páginas, componentes de interfaz, hooks, servicios y modelos, lo que fomenta la cohesión interna y minimiza el acoplamiento entre módulos.

La capa de presentación se desarrolla siguiendo un esquema **component-based**, en el que cada módulo está compuesto por elementos reutilizables y encapsulados, favoreciendo la modularidad y la reutilización de código. La gestión de estado se resuelve principalmente mediante **hooks de React**, mientras que la sincronización de datos con el servidor se gestiona a través de librerías modernas de **data fetching** que implementan almacenamiento en caché, actualizaciones optimistas y estrategias de invalidación de consultas. Esto asegura que la información desplegada en la interfaz se mantenga coherente y actualizada en todo momento.

Desde el punto de vista estético y de experiencia de usuario, se emplea **Tailwind CSS** para la definición de estilos, priorizando un diseño **responsivo** y adaptado a distintos dispositivos, con un enfoque *mobile-first*. A nivel de navegación, se implementa **React Router**, lo cual permite la definición de rutas por cada dominio funcional y habilita el *lazy loading* de componentes para optimizar el rendimiento.

En relación con los aspectos de **calidad del software**, se incorporan herramientas de análisis estático, formateo automático y pruebas unitarias para asegurar la corrección del código. Asimismo, se promueve la accesibilidad (*all y*) mediante buenas prácticas de semántica HTML, etiquetas ARIA y control del enfoque de navegación por teclado.

En suma, el frontend de Kuska se caracteriza por:

- Un **stack tecnológico robusto** (React + Vite + TypeScript).
- Una **arquitectura orientada al negocio** (Screaming Architecture) que organiza el sistema en función de sus dominios funcionales.
- Un **diseño modular y escalable** que facilita el mantenimiento y la extensión futura.
- Un enfoque en la **eficiencia y la usabilidad**, garantizando tiempos de carga rápidos, comunicación fluida con el backend y una experiencia de usuario consistente.

De esta forma, la interfaz de usuario de Kuska se convierte en un componente esencial que no solo ofrece acceso a las funcionalidades del sistema, sino que también refleja la organización interna del proyecto y respalda los principios de ingeniería de software aplicada en el contexto académico y profesional.

4.2. Wireframes

Figura 9.
Wireframe del apartado del landing page elaborado en Figma

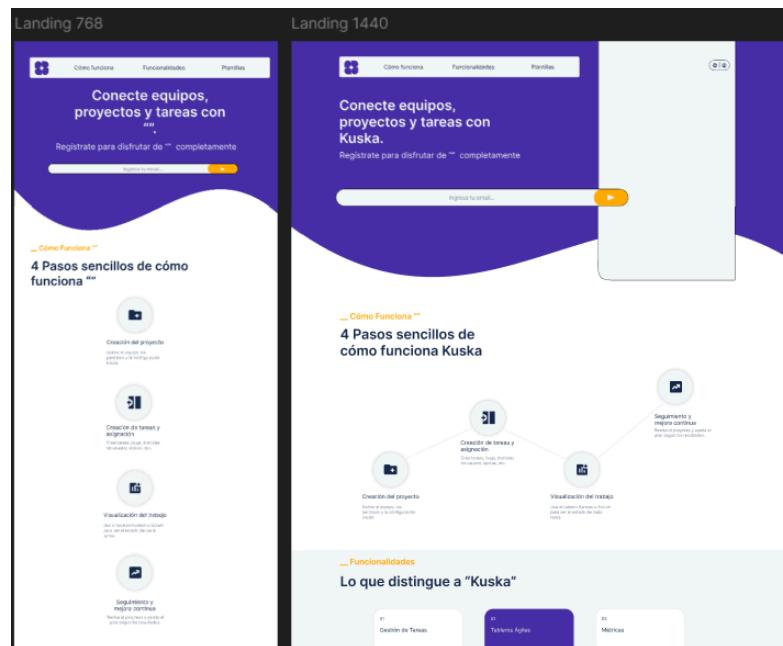


Figura 10.

Apartado de inicio de sesión de Kuska elaborado en Figma

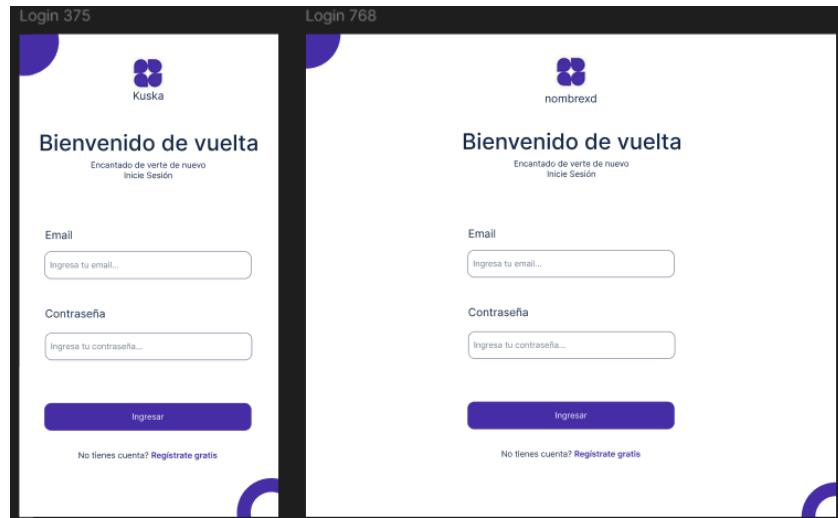


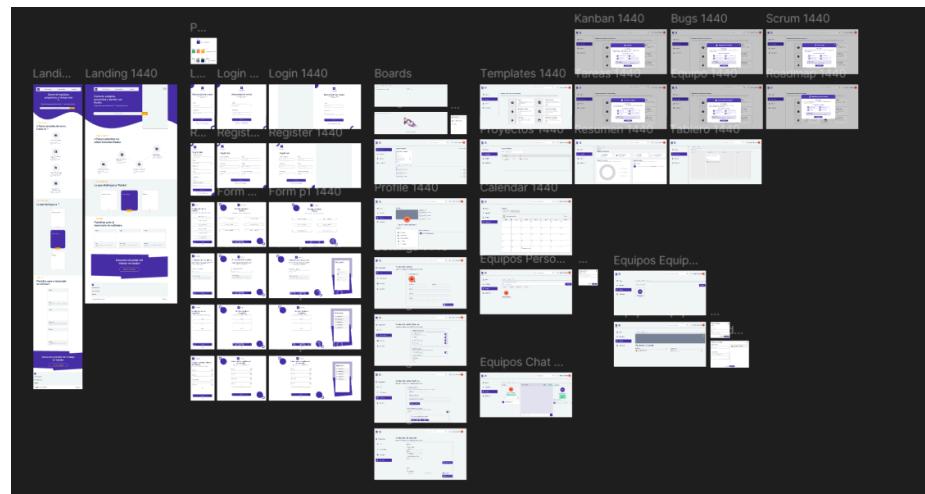
Figura 11.

Apartado del dashboard de Kuska elaborado en Figma

The image displays a screenshot of the Kuska dashboard. At the top, there's a navigation bar with tabs 'En trabajo', 'Asignado a mí', and 'Tareas'. A dropdown menu is open over the 'Asignado a mí' tab, showing options like 'Mi perfil', 'Configuraciones', and 'Salir'. The main content area has a dark background. On the left, there's a sidebar with sections for 'Proyectos' (with a 'Punto 1' card), 'Equipos', and 'Calendario'. The central part of the dashboard features a 'Boards' section with a board titled 'Tareas - Plataforma de Desarrollo' and a 'Assigned to me' section showing a placeholder image of a person. At the bottom, there's a 'Main P 1440' section containing a search bar, user info 'Jorge C. Bautista', and a 'Todos los proyectos' link. The bottom right corner shows three circular progress indicators labeled 'Creado ayer', 'Creado hoy', and 'Creado ayer'.

Figura 12.

Vista general de todo el prototipo de la aplicación elaborado en Figma.



4.3. Páginas implementadas

Todas las vistas pueden ser visualizadas ingresando al siguiente enlace

kuska.onrender.com

a. Vista de Landing Page

- Ruta: /
- Descripción: Página pública principal con Navbar, Hero (CTA), secciones HowWorks, Functionalities, Templates y un CTA final + footer. Diseñada para presentar la propuesta de valor y captar emails para registro.

b. Vista de inicio de sesión (Login)

- Ruta: /auth/login
- Descripción: Formulario de inicio de sesión con inputs para email y contraseña, diseño responsive con lado decorativo morado. Implementa autocompletado del email si se pasa como query param (?email=usuario@gmail.com). El submit actualmente registra en consola (placeholder para lógica de autenticación).

c. Vista de registro (Register)

- Ruta: /auth/register
- Descripción: Formulario de registro con campos (nombre, email, nacimiento, celular, contraseña y confirmación), UI consistente con el login; enlace para ir a iniciar sesión que ahora apunta a /auth/login. Submit muestra datos en consola (placeholder).

d. Pasos de registro (multi-step)

- Ruta: /auth/work-type
- Descripción: Paso para seleccionar tipo de trabajo del usuario (parte del flow de onboarding).
- Ruta: /auth/project-name
- Descripción: Paso para ingresar el nombre del proyecto durante el onboarding.
- Ruta: /auth/work-needs
- Descripción: Paso para detallar necesidades de trabajo del equipo/proyecto.
- Ruta: /auth/work-tracking
- Descripción: Paso final del onboarding que recopila información y redirige al dashboard.

e. Dashboard (layout principal)

- Ruta: /dashboard
- Descripción: Layout principal con Sidebar y Navbar. Contiene rutas anidadas con Outlet para las siguientes vistas internas:
 - /dashboard (index) -> For You (resumen personal)

- /dashboard/for-you -> ForYou: Resumen personalizado con estadísticas, proyectos recientes, pestañas y actividades del día.
- /dashboard/projects -> Projects: Lista de proyectos recientes/propios (usa ProjectList con tarjetas, búsqueda y filtros).
- /dashboard/projects/:projectId -> Project Summary: Página de resumen de un proyecto con pestañas (Resumen y Tablero). Incluye status cards, chart, actividad reciente y un Board (Kanban).
- /dashboard/templates -> Templates: Galería/gestión de plantillas (placeholder con estructura lista).
- /dashboard/teams -> Teams: Gestión de equipos con pestañas (Personas, Equipos, Chat).
- /dashboard/teams/equipo/:teamId -> TeamDetail: Vista detallada de un equipo con miembros, links fijados y acciones (modales para agregar/invitar).
- /dashboard/calendar -> Calendar: Vista tipo calendario para eventos y planificación.
- /dashboard/profile/:id -> Profile: Perfil de usuario (detalle de usuario).

f. Configuración (independiente, layout con rutas)

- Ruta: /configuration
- Descripción: Layout de configuración que agrupa subsecciones:
 - /configuration (index) y /configuration/profile -> Profile: Ajustes de perfil.

- /configuration/appearance -> Appearance: Opciones de apariencia/tema.
- /configuration/notifications -> Notifications: Preferencias de notificaciones.
- /configuration/security -> Security: Ajustes de seguridad (contraseñas, MFA, sesiones).

4.4. Diseño responsivo

- a. Landing page diseño responsive:

Figura 12.
Apartado de Hero de nuestra landing page

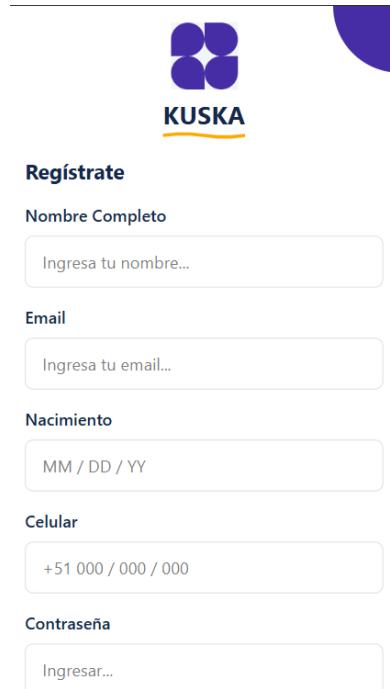


Figura 14.
Apartado de inicio de sesión de Kuska



The image shows the login page for Kuska. At the top center is the Kuska logo, which consists of three overlapping purple circles above the word "KUSKA". Below the logo is a purple header bar with the text "Bienvenido de vuelta" in white. Underneath the header, there is a message "Encantado de verte de nuevo" and a link "Inicie Sesión". The main form area has two input fields: "Email" and "Contraseña", both with placeholder text "Ingresa tu email..." and "Ingresa tu contraseña...". Below these fields is a purple button labeled "Ingresar". At the bottom of the form, there is a link "No tienes cuenta? Regístrate gratis". The background features large, semi-transparent purple curved shapes on the sides.

Figura 15.
Apartado de registro de Kuska



The image shows the registration page for Kuska. At the top center is the Kuska logo. Below it is a purple header bar with the text "Regístrate" in white. The main form area contains several input fields: "Nombre Completo" with placeholder "Ingresa tu nombre...", "Email" with placeholder "Ingresa tu email...", "Nacimiento" with placeholder "MM / DD / YY", "Celular" with placeholder "+51 000 / 000 / 000", and "Contraseña" with placeholder "Ingresar...". The background features large, semi-transparent purple curved shapes on the sides.

Figura 16.
Apartado de dashboard de Kuska



5. Avance de desarrollo backend

5.1. Arquitectura y enfoque de desarrollo

El desarrollo del backend del sistema Kuska se ha estructurado bajo los principios de Domain-Driven Design (DDD), organizando el código en módulos que reflejan los dominios de negocio identificados en el proyecto. Esta aproximación arquitectónica garantiza:

- Separación clara de responsabilidades entre dominios
- Mantenibilidad y escalabilidad del código
- Alineación directa entre la implementación técnica y los conceptos de negocio
- Facilidad para la evolución ordenada del sistema

5.2. Stack tecnológico y justificación

La implementación del servidor utiliza las siguientes tecnologías, seleccionadas por sus ventajas técnicas y ecológica de desarrollo:

- Node.js con Express.js: Elegido por su modelo de E/S no bloqueante y capacidad para manejar múltiples solicitudes concurrentes de manera eficiente, esencial para una plataforma que gestiona numerosas interacciones de usuarios (Chec & Nowak, 2019)
- TypeScript: Implementado para proporcionar tipado estático al ecosistema JavaScript, permitiendo detección temprana de errores, mejor mantenibilidad del código y facilitando la colaboración en equipos de desarrollo (Bogner & Merkel, 2022)
- PostgreSQL: Seleccionado como sistema de base de datos relacional por su robustez, soporte ACID, capacidad para manejar volúmenes significativos de datos y flexibilidad para modelar relaciones complejas entre entidades

5.3. Componentes de infraestructura y herramientas

Para garantizar la calidad, seguridad y documentación del sistema, se han integrado las siguientes herramientas:

- Prisma ORM: Utilizado como capa de abstracción de base de datos, proporcionando type safety, migraciones automatizadas y una API intuitiva para operaciones de persistencia
- Zod: Implementado para validación de esquemas y tipos en tiempo de ejecución, complementando el tipado estático de TypeScript
- Bcrypt: Empleado para el hash seguro de contraseñas, garantizando la protección de credenciales de usuario

- JWT (JSON Web Tokens): Utilizado para la gestión de autenticación y autorización, proporcionando un mecanismo stateless para la gestión de sesiones
- Swagger: Integrado para la documentación automática de la API RESTful, facilitando el consumo de los endpoints por parte del frontend y otros clientes potenciales

5.4. Estructura modular del sistema

El backend se ha organizado en módulos específicos que corresponden a los subdominios del negocio:

- Módulo de Proyectos: Gestiona la creación, configuración y ciclo de vida completo de proyectos, incluyendo fases, prioridades y responsables.
- Módulo de Tareas: Administra el flujo completo de tareas dentro de los proyectos, con soporte para estados, asignación, dependencias y seguimiento del progreso.
- Módulo de Plantillas: Permite la definición y reutilización de plantillas estandarizadas para proyectos y flujos de trabajo.
- Módulo de Gestión de Miembros: Controla la vinculación de usuarios a proyectos, la asignación de roles y la administración de equipos de trabajo.
- Módulo de Permisos: Define políticas de acceso basadas en roles y contextos específicos, garantizando la seguridad y confidencialidad de la información.

5.5. Patrones arquitectónicos implementados

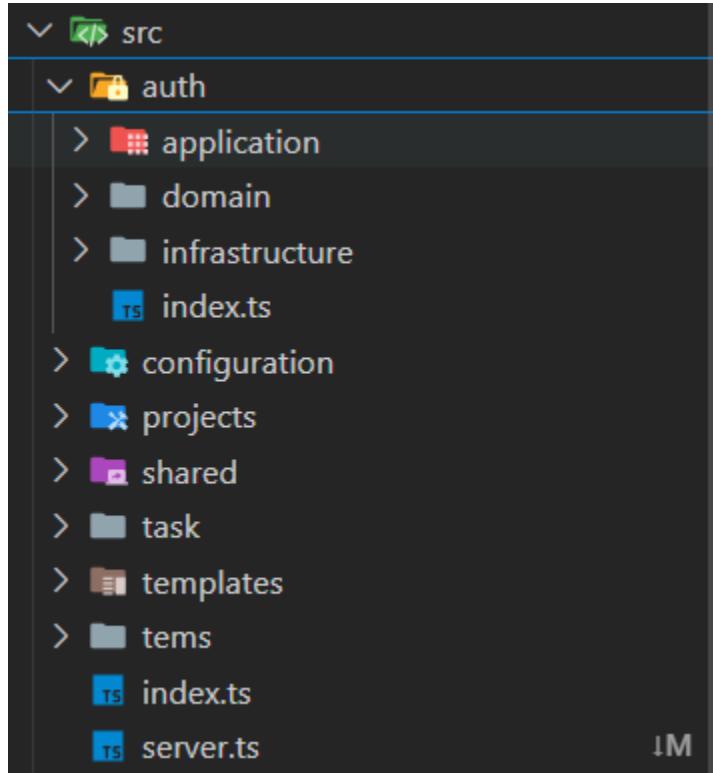
Cada módulo sigue el patrón de arquitectura por capas con separación explícita entre:

- Capa de Dominio: Contiene entidades, objetos de valor, agregados y repositorios que encapsulan la lógica central de negocio
- Capa de Aplicación: Orquesta los casos de uso y define los servicios que exponen la funcionalidad del sistema

- Capa de Infraestructura: Gestiona la persistencia mediante Prisma, la comunicación externa y otros aspectos técnicos

Figura 17

Estructura de las carpetas según DDD



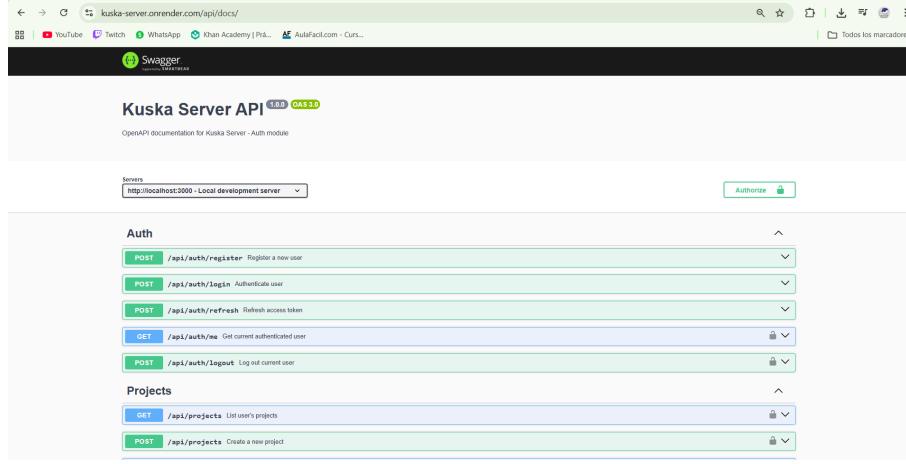
5.6. API y comunicación

La comunicación con el frontend se realiza mediante una API RESTful diseñada bajo estándares HTTP, que incluye:

- Convenciones de nombrado consistentes y predecibles
- Validación de entradas utilizando Zod
- Manejo centralizado de errores
- Autenticación basada en JWT
- Documentación automática con Swagger

Si bien en esta primera versión no se ha implementado comunicación en tiempo real, la arquitectura planteada permite la incorporación futura de WebSockets o Server-Sent Events sin impactar significativamente la estructura existente.

Figura 18
Documentación en Swagger de las apis



6. Despliegue y Puesta en Producción

Este apartado tiene como objetivo demostrar que el sistema es accesible públicamente y funcional en la nube.

6.1. Estrategia de Despliegue en Render

- **Plataforma unificada:** Mencionar que se seleccionó **Render** como proveedor de servicios en la nube (PaaS) debido a su capacidad para alojar tanto servicios web estáticos (Frontend) como servicios web dinámicos (Backend) en un mismo ecosistema.
- **Arquitectura Cloud:** Explicar que la solución se desplegó en dos servicios independientes que se comunican vía HTTP/HTTPS, replicando la arquitectura desacoplada descrita en el diseño de la solución.

6.2. Despliegue del Backend (API)

- Configuración del Servicio:
 - **Entorno:** Node.js con TypeScript.
 - **Build Command:** Detallar el comando de compilación (ej. `npm run build` o `tsc`) para transpilación de TypeScript a JavaScript.
 - **Start Command:** El comando de inicio (ej. `npm start` o `node dist/index.js`)
- **Variables de Entorno (Environment Variables):** Listar (sin revelar secretos) las variables configuradas en el panel de Render:
 - **DATABASE_URL:** Cadena de conexión a PostgreSQL (mencionado en el stack).

- **JWT_SECRET**: Para la firma de tokens.
- **PORT**: Puerto expuesto por Render.
- **CORS_ORIGIN**: La URL del frontend permitida para realizar peticiones.
- **Resultado:** Mencionar la URL pública operativa de la API (visible en tu captura de Swagger): <https://kuska-server.onrender.com>.

6.3. Despliegue del Frontend (Cliente Web)

- Configuración del Servicio:
 - **Tipo de servicio:** Static Site (SPA - Single Page Application).
 - **Stack:** React + Vite.
 - **Build Command:** `npm run build` (que genera la carpeta `dist`).
 - **Publish Directory:** Configurado a la carpeta `dist` o `build`.
- **Manejo de Rutas (Rewrites):** Explicar brevemente que, al ser una SPA con React Router, se configuró en Render una regla de "Rewrite" para que todas las rutas redirijan a `index.html`, evitando errores 404 al recargar páginas internas
- **Resultado:** Mencionar la URL pública operativa del cliente:
<https://kuska.onrender.com>.

6.4. Integración Continua (CI/CD)

- **Automatización:** Destacar que Render se conectó directamente al repositorio de **GitHub** del proyecto.
- **Auto-Deploy:** Explicar que se activó la funcionalidad de despliegue automático ("Auto-Deploy"), de modo que cada **push** a la rama **main** (o **master**) dispara automáticamente una nueva construcción y despliegue del sistema, garantizando que la versión en producción siempre esté actualizada con el código más reciente.

7. Conclusiones

El desarrollo del proyecto **Kuska** ha constituido un ejercicio integral de ingeniería de software que ha permitido validar la viabilidad de unificar la gestión de proyectos y la colaboración en equipo en una sola plataforma. Tras finalizar las etapas de diseño, codificación y despliegue, se puede afirmar que el sistema cumple con su objetivo principal de mitigar la fragmentación de información y la duplicidad de esfuerzos que suelen afectar a los equipos de trabajo modernos .

La solución entregada no solo centraliza la planificación y el seguimiento, sino que establece una base tecnológica sólida para el escalamiento futuro.

Desde la perspectiva del **desarrollo Frontend**, la implementación de una arquitectura basada en componentes utilizando el ecosistema de **React** y **Vite**, complementada con el tipado estático de **TypeScript**, ha resultado en una interfaz de usuario altamente interactiva, modular y mantenible . La fidelidad lograda respecto a los diseños y wireframes elaborados previamente en **Figma** demuestra una correcta traducción de los requisitos visuales a código funcional, garantizando una experiencia de usuario (UX) consistente y un diseño responsive adaptado a múltiples dispositivos gracias al uso de **Tailwind CSS** .

En el ámbito del **Backend**, la adopción de principios de **Domain-Driven Design (DDD)** para estructurar la lógica del servidor ha sido determinante para la organización del código. La separación de módulos por dominios de negocio (Proyectos, Tareas, Miembros) asegura que el sistema sea escalable y fácil de auditar . La elección de un stack tecnológico robusto compuesto por **Node.js**, **Express** y **PostgreSQL**, gestionado a través de **Prisma ORM**, ha permitido construir una API RESTful segura, eficiente y bien documentada mediante **Swagger**, capaz de manejar la complejidad de las relaciones de datos y la seguridad mediante autenticación JWT .

Un hito crucial para la validación del proyecto fue el **ciclo de despliegue y puesta en producción**. La implementación exitosa de la arquitectura cliente-servidor en la plataforma **Render** demostró la capacidad del sistema para operar en un entorno de nube real. Este paso no solo verificó la correcta comunicación entre el frontend y el backend a través de protocolos HTTP seguros, sino que también validó la configuración de variables de entorno, las políticas de acceso (CORS) y la integración continua del código. El despliegue en Render confirma que Kuska ha trascendido la etapa de prototipo local para convertirse en una aplicación web accesible y funcional globalmente.

En definitiva, **Kuska** se consolida como una herramienta Full Stack moderna que responde eficazmente a las necesidades de gestión tecnológica. La sinergia lograda entre una interfaz intuitiva, un servidor estructurado bajo estándares de industria y una infraestructura de despliegue automatizada, evidencia el cumplimiento de los objetivos generales y específicos planteados al inicio del proyecto . El sistema queda preparado para futuras iteraciones que incluyan funcionalidades avanzadas como la comunicación vía WebSockets, manteniendo una deuda técnica baja y una alta calidad de software.

8. Bibliografía

Bogner, J., & Merkel, M. (2022). *To Type or Not to Type? A Systematic Comparison of the Software Quality of JavaScript and TypeScript Applications on GitHub*. arXiv. <https://doi.org/10.48550/arXiv.2203.11115>

Chęć, D., & Nowak, Z. (2019). *The Performance Analysis of Web Applications Based on Virtual DOM and Reactive User Interfaces*. En *Engineering Software Systems: Research and Praxis* (Vol. 119-134). Springer. https://doi.org/10.1007/978-3-319-99617-2_8

Goli, V. Reddy. (2019). *React's Evolution and the Paradigm Shift of Hooks in Modern Web Development*. Turkish Journal of Computer and Mathematics Education (TURCOMAT), 10(1), 788-791.

Gu, Y. (2025). *Practical Approaches to Developing High-performance Web Applications Based on React*. *Frontiers in Science and Engineering*, 5(2).

Göktürk, M., & Koç, F. (2022). *Code Quality Analysis Engine with Codes Written in TypeScript*. Turkish Journal of Computer and Mathematics Education (TURCOMAT), 1(1), 1-7.

Nopriani, F., & Muhammad, M. A. (2024). *Pengujian Usability Website Dokumentasi Menggunakan System Usability Scale (SUS)*. *METHODIKA: Jurnal Teknik Informatika dan Sistem Informasi*, 10(2), 1-6. <https://doi.org/10.46880/mtk.v10i2.2987>

S., N., Usha Sree, R., & Mohan, P. (2024). *Comparison of Utility-First CSS Framework*. *Journal of Innovation and Technology*, 2024(1). DOI:10.61453/joit.v2024no32

Molnar, A-J., & Motogna, S. (2020). *Longitudinal Evaluation of Open-Source Software Maintainability*. arXiv. <https://doi.org/10.48550/arXiv.2003.00447>

Seref, B., & Tanriover, Ö. (2016). *Software Code Maintainability: A Literature Review*. International Journal of Software Engineering & Applications (IJSEA), 7(3).