Universidad Nacional del Noroeste de la Provincia de Buenos Aires

Título: Refactorización y fortalecimiento arquitectónico de un sistema logístico en producción

Carrera: Licenciatura en Sistemas

Práctica Profesional Supervisada

Estudiante: Julián Andrés Zabala

Tutor Docente: Mariana Adó

Tutor de Empresa/Institución/Organización: Juan José Pérez/KipBip

Fecha de presentación: 03/05/2026

Introducción

La presente Práctica Profesional Supervisada (PPS) se realizó en la empresa KipBip, dedicada a brindar servicios tecnológicos, orientados a la gestión logística y de transporte. Su actividad principal consiste en ofrecer soluciones integrales que permiten optimizar la planificación, el seguimiento y la ejecución de operaciones logísticas.

El presente trabajo se basará sobre el software creado en la empresa, una herramienta diseñada para pequeñas y medianas empresas con flotas entre 1 y 50 vehículos. Este sistema, compuesto por una plataforma web (KipBip Web) y una aplicación móvil (KipBip App), facilita la planificación, gestión y monitoreo de rutas en tiempo real, contribuyendo a reducir costos operativos y mejorar la eficiencia del transporte.

La finalidad de esta PPS fue consolidar competencias técnicas y profesionales mediante la participación activa en el área de sistemas de la empresa, contribuyendo directamente a la evolución y transformación de un producto tecnológico en un entorno productivo real.

Las actividades se elaboraron en el equipo de desarrollo de software y se enfocaron en la transformación de la arquitectura de KipBip, que hasta entonces operaba sin backend y dependía únicamente de Cloud Firestore [1]. La PPS acompañó esta transición incorporando un backend estructurado en capas, mecanismos de seguridad y un modelo de datos relacional, con el fin de mejorar la escalabilidad, consistencia e integridad del sistema.

Empresa

KipBip tiene su origen en el desarrollo de un software que lleva el mismo nombre, concebido a partir de la identificación de una necesidad específica dentro del sector logístico. A partir de este proyecto tecnológico inicial, la organización se constituye formalmente con el propósito de brindar soluciones orientadas a la gestión de

logística y transporte. De este modo, la empresa se estructura en torno a un producto diseñado para optimizar procesos operativos y responder a las demandas crecientes del mercado. La compañía tiene una modalidad de trabajo completamente virtual, manteniendo su estructura operativa distribuida de forma remota.

El software KipBip surgió inicialmente como una herramienta complementaria a sistemas externos, con el objetivo de brindar una gestión básica de logística y seguimiento de rutas para pequeñas y medianas empresas.

Este origen es relevante, ya que el proyecto fue diseñado con un alcance funcional limitado, sin anticipar las transformaciones que luego experimentarían tanto el producto como la organización. Con el tiempo, y a partir de nuevas necesidades operativas y estratégicas, se definió la evolución del software KipBip hacia un producto independiente, escalable y autónomo, marcando así el inicio del proceso de independencia tecnológica del software respecto de soluciones externas.

En este contexto de transición y redefinición del sistema, el área de desarrollo inició una etapa de reestructuración técnica, orientada a ampliar capacidades, mejorar la escalabilidad y asegurar la integridad de los datos. La práctica profesional se desarrolló dentro de este proceso, aportando al fortalecimiento de la base tecnológica necesaria para sostener la nueva visión del producto.

Objetivos

Objetivo general

Este objetivo se enfoca en resolver las problemáticas detectadas en el flujo actual de la información, garantizando su consistencia, mejorando el rendimiento y aplicando mecanismos de seguridad; que permitan independizar el software KipBip optimizando la gestión de datos y procesos mediante la migración del flujo de datos desde Cloud Firestore [1] hacia SQL Server [2], bajo una arquitectura en capas que cumpla con criterios de escalabilidad, seguridad y mantenibilidad.

Objetivos específicos

- Analizar las limitaciones de la infraestructura actual sustentada por una base de datos NoSQL (Cloud Firestore [1]), identificando necesidades funcionales y oportunidades de mejora.
- Evaluar alternativas de bases de datos y justificar la elección de SQL Server [2] como solución relacional más adecuada.
- Diseñar modelos para la base de datos relacional, definiendo tablas, relaciones y reglas de negocio acordes a los requerimientos del sistema.
- Ejecutar el proceso de migración y carga inicial de datos entre Cloud Firestore [1] y SQL Server [2], realizando pruebas de integridad y los ajustes necesarios.
- Analizar, diseñar e implementar la sincronización entre la plataforma web y la aplicación móvil garantizando la coherencia de la información.
- Desarrollar una arquitectura backend modular y buenas prácticas para garantizar la estabilidad, escalabilidad y seguridad, basada en los principios SOLID [3], que facilite la integración de nuevos servicios y el mantenimiento.
- Implementar validaciones, reglas de negocio y pruebas funcionales en el backend de KipBip.
- Aplicar estrategias de seguridad mediante autenticación y autorización con JSON Web Token (JWT) [4] y middlewares, contemplando distintos perfiles de usuario y niveles de acceso.
- Desarrollar pruebas unitarias e integrales que verifiquen el correcto funcionamiento de cada componente del sistema.

Plan de Trabajo y Carga Horaria

El plan de trabajo se desarrolló a lo largo de diez semanas, cumpliendo con los requerimientos establecidos por la institución y alcanzando una carga total de 300 horas de actividad profesional. La jornada laboral se organizó de lunes a viernes,

con una dedicación de 6 horas diarias (30 horas semanales), orientadas exclusivamente al desarrollo de las tareas asignadas dentro del proyecto.
Las actividades se organizaron conforme al diagrama de Gantt:

CRONOGRAMA DE TAREAS

Diagrama de Gantt de la distribución en el tiempo de las actividades a realizar en la PPS y entrevistas con el/la Tutor/a Docente para informar el grado de avance del Plan de Trabajo

N°	ACTIVIDADES	TIEMPO DE DURACIÓN (SEMANAS)									
		1	2	3	4	5	6	7	8	9	10
1	Relevamiento de datos en Firebase (1.1)	■									
2	Análisis del backend (2.1)	■									
3	Diseño esquema relacional - tablas para SQL (1.2)		■								
4	Definición de capas en backend (2.2)		■								
5	Refactorización del backend inicial (2.3)			■							
6	Ajustes de refactorización en el backend (2.4)			■							
7	Implementación de tablas, relaciones y reglas de negocio en SQL server (1.3)			*	■						
8	Poblado inicial y pruebas integridad de datos (1.4)				■						
9	Pruebas unitarias backend (2.5)				■						
10	Análisis y definición de reglas de sincronismo entre Firebase y SQL (3.1) (depende de 2.5 y 1.4)					■					
11	Implementación de reglas de sincronización (3.2)						■				
12	Pruebas sincronización entre ambas bases de datos (Firebase y SQL) (3.3)						■				
13	Ajustes sobre sincronización en las bases de datos (3.4)						■				
14	Analisis de integración de validaciones en el backend (4.1) (depende de 2.5 y 1.4)					*	*	■			
15	Implementación de validaciones de datos y reglas de negocio (4.2)							■			
16	Pruebas de validaciones (4.3)							■			
17	Ajustes validaciones (4.4)							■			
18	Diseño de seguridad en el backend (5.1) (depende de 2.5 y 1.4)					*	*	*	■		
19	Implementación de método de autenticación y seguridad JWT (5.2)									■	
20	Implementación middleware de seguridad (5.3)									■	
21	Pruebas de seguridad (5.4)										■
	* Son las semanas que se puede atrasar la tarea										

Fig. 1. Diagrama de Gantt (elaboración propia).

La ejecución del plan se desarrolló según lo programado, sin registrarse desviaciones de tiempo significativas respecto de la planificación original.
Asimismo, una vez finalizado el período formal correspondiente a la PPS, se continuó colaborando con el proyecto, acompañando la evolución de sus requerimientos y el desarrollo de nuevas funcionalidades, lo cual permitió consolidar y profundizar los avances alcanzados durante la práctica.

Metodología de trabajo

Para el desarrollo del trabajo se optó por una metodología ágil, orientada a maximizar la adaptabilidad, la retroalimentación continua y la entrega incremental de resultados. Este enfoque flexible se adoptó debido a que la incorporación de las nuevas funcionalidades requerían flexibilidad y capacidad de respuesta ante

posibles cambios. Aspecto que resultó esencial para un proyecto en evolución, asegurando un desarrollo ordenado, controlado y alineado con las necesidades reales del sistema KipBip.

La metodología se estructuró en iteraciones semanales, cada una con objetivos concretos y medibles. A continuación, se detalla la dinámica de trabajo aplicada a lo largo del proceso:

Planificación semanal

Al inicio de cada semana se realizó una planificación breve, en la cual se definieron los objetivos específicos a alcanzar durante esa iteración. Estos objetivos se integraron con el cronograma general del proyecto.

Priorización de tareas

Las tareas se priorizaron siguiendo criterios de:

- Criticidad técnica: elementos esenciales como la migración de la base de datos, la arquitectura en capas, validaciones y la seguridad del backend.
- Dependencias entre tareas: actividades que debían completarse antes de avanzar con otras.
- Impacto directo: actividades que generarían incidencia directa en el usuario final, puntualmente la integración con KipBip App y los mecanismos de sincronización entre las bases de datos.

Esta priorización permitió avanzar de forma ordenada, reduciendo riesgos y evitando la necesidad de reestructuraciones posteriores.

Comunicación continua y retroalimentación

La interacción con el tutor de la empresa se mantuvo mediante revisiones semanales, en las cuales se presentaron los avances, se evaluaron las decisiones técnicas y se ajustó el enfoque cuando fue necesario. Estas instancias de retroalimentación fluida permitieron:

- Detectar mejoras potenciales en etapas tempranas.

- Alinear el desarrollo con las necesidades reales de la empresa.
- Validar de manera continua el rumbo del proyecto.

Iteraciones con entregas parciales

Cada semana finalizó con una entrega parcial del incremento construido o una demo funcional, dependiendo del tipo de tarea. Estas entregas incluyeron:

- Modelos y diagramas actualizados.
- Endpoints implementados.
- Configuraciones de seguridad.
- Pruebas unitarias y sus resultados.
- Ajustes en la arquitectura o en el modelo de datos.

Este proceso facilitó la detección temprana de errores, inconsistencias o problemas de integridad, y permitió ajustar la planificación sin comprometer los plazos generales.

Ajuste y mejora continua

Al cierre de cada iteración se evaluó:

- El grado de cumplimiento de los objetivos.
- Los obstáculos surgidos.
- Las mejoras a incorporar para la siguiente semana.

Esto permitió adoptar una dinámica de mejora continua, fundamental para un proyecto de migración y rediseño arquitectónico donde la aparición de nuevos requerimientos es lo habitual.

Descripción de la Práctica Profesional

La PPS tuvo como propósito principal el fortalecimiento integral del backend de la plataforma KipBip, acompañando el proceso de evolución tecnológica del proyecto y resolviendo dificultades derivadas de su crecimiento funcional. Para ello, se abordó

la migración desde una base de datos no relacional como Cloud Firestore [1] hacia un sistema relacional, en este caso, SQL Server [2] el diseño e implementación de una arquitectura en capas, alineada con los principios SOLID [3]; la sincronización entre sistemas (Web y App) y la incorporación de mecanismos de seguridad basados en autenticación mediante JSON Web Token [4].

Tareas realizadas

El desarrollo se ejecutó durante un período de diez semanas, organizado en cuatro etapas sucesivas: (1) se llevó a cabo el análisis del estado inicial, el diagnóstico técnico y la selección de herramientas; (2) se diseñó el modelo de datos y la arquitectura del backend; (3) se realizó la integración de sistemas, la validación de datos y las pruebas correspondientes; y (4) se implementa la capa de seguridad y los mecanismos de autenticación.

Análisis del estado inicial, diagnóstico técnico y selección de herramientas

La primera etapa (semanas 1 y 2) se centró en comprender exhaustivamente el funcionamiento actual del sistema, sus limitaciones y las necesidades emergentes del proyecto. Durante este período se realizó un análisis detallado de las características de la base de datos Cloud Firestore [1], utilizada originalmente por KipBip, y se identificaron sus principales restricciones frente a las nuevas demandas del producto. Entre los problemas detectados se destacaron:

- Dificultades para modelar adecuadamente relaciones complejas entre entidades.
- Falta de mecanismos transaccionales nativos.
- Riesgos de inconsistencia debido a la lógica distribuida en el frontend.
- Ausencia de control centralizado sobre la integridad de los datos.

La evolución del software hacia un sistema de gestión logística más amplio exige mayores niveles de consistencia, escalabilidad e integración, lo que justifica la

evaluación de bases de datos relacionales. Para ello se llevó a cabo un análisis comparativo entre PostgreSQL [5], MariaDB [6] y SQL Server [2], considerando rendimiento, soporte, disponibilidad de recursos humanos capacitados, integración con el entorno empresarial y cumplimiento de propiedades ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad) [7]. SQL Server [2] se posicionó como la opción más sólida (debido al personal capacitado) para garantizar un modelo de datos estable y escalable.

Asimismo, se efectuó un relevamiento estructural del modelo en Cloud Firestore [1], analizando colecciones, subcolecciones y reglas implícitas.

Finalmente, se determinó que la lógica de validación, reglas de negocio y acceso a los datos residían en el frontend, lo que generaba riesgos en términos de seguridad, duplicación de procesos y dificultades para el mantenimiento. Este diagnóstico reforzó la necesidad de implementar un backend seguro, eficiente, escalable y estable que centralice la lógica de negocio. Para ello se seleccionaron las tecnologías a utilizar en el backend, en las cuales se priorizaron los conocimientos de la mayoría del equipo, por este motivo se optó por Node.js [8] con TypeScript [9] y el framework Express [10] debido a su popularidad, simplicidad y flexibilidad.

Diseño de los modelos para la base de datos y la arquitectura del backend

La segunda etapa (semanas 3 y 4) se orientó al diseño y puesta en marcha de una arquitectura backend basada en capas, con el propósito de mejorar la mantenibilidad del código y establecer una estructura clara de responsabilidades. Para ello se emplearon los principios SOLID [3] y lineamientos de buenas prácticas en ingeniería de software.

Durante este proceso se han definido los siguientes componentes:

- Capa de rutas y controladores, destinada a la recepción y gestión de las solicitudes del cliente.
- Capa de servicios, donde se unifica la lógica de negocio, evitando su

dispersión y facilitando futuras modificaciones.

- Capa de repositorios, responsable del acceso a los datos y la interacción con Cloud Firestore [1] y SQL Server [2].
- Middlewares iniciales, destinados al manejo de excepciones, recepción de solicitudes y estandarización de respuestas.

El diseño permite un desacoplamiento efectivo entre los distintos módulos del sistema, sentando las bases para un backend más estable y escalable.

En paralelo, el relevamiento de datos en Cloud Firestore [1] permitió establecer las bases para el diseño del modelo entidad-relación. Definiendo entidades, atributos y relaciones. Posteriormente se ha transformado en el modelo lógico relacional con criterios de normalización y finalmente al modelo de tablas en el cual se establecieron las claves primarias y foráneas, restricciones de integridad y reglas de negocio a nivel de base de datos. También se crearon el modelo de tablas y un conjunto de datos de prueba que permitió validar el comportamiento del mismo y detectar ajustes necesarios antes de su integración en la base de datos final en SQL Server [2].

Finalmente, se ejecutaron tareas de refactorización del código con buenas prácticas, orientadas a mejorar la calidad, legibilidad y rendimiento del backend.

Integración de sistemas, validación de datos y pruebas

La tercera etapa (semanas 5, 6 y 7) se abordó la integración entre los distintos sistemas de información que conviven en el ecosistema KipBip: la aplicación móvil (que continúa operando sobre Cloud Firestore [1]) y la plataforma web (migrada a SQL Server [2]). Lo cual implicó diseñar mecanismos de sincronización que han garantizado la coherencia de los datos entre ambos modelos.

Para ello se definieron un conjunto de reglas de determinación de datos maestros y esclavos en cada base, así como los flujos de sincronización bidireccionales. Se desarrollaron procesos de actualización bidireccionales, que aseguran la consistencia de los datos, sin generar duplicidad o desactualización.

En materia de validación, se llevó a cabo un análisis comparativo de librerías

orientadas al aseguramiento de integridad en el backend. Finalmente se seleccionó class-validator [11], dado su nivel de adopción, madurez y facilidad de integración con los Data Transfer Objects (DTOs) [12]. También se desarrollaron DTOs [12] para cada tipo de solicitud y respuesta, estableciendo controles estrictos de estructura, formato y reglas de negocio antes de procesar los datos.

Durante esta etapa se realizaron pruebas funcionales y unitarias mediante herramientas como Postman [13], lo que permitió evaluar el comportamiento del sistema ante distintos escenarios, detectar inconsistencias y corregir errores. Estas pruebas fueron fundamentales para consolidar la comunicación entre las bases de datos y asegurar el correcto funcionamiento del backend.

Implementación de la capa de seguridad y autenticación

La última etapa (semanas 8, 9 y 10) del trabajo se centró en la implementación de la seguridad, capaz de proteger el acceso a los datos y asegurar que los usuarios interactúen con el sistema conforme a sus permisos.

Para ello se profundizó el desarrollo de los middlewares destinados a interceptar las solicitudes entrantes, validar cabeceras, gestionar errores y estandarizar respuestas. También se implementó un mecanismo de autenticación basado en JWT [4], que permitió verificar la identidad del usuario sin necesidad de consultas constantes a la base de datos, optimizar la eficiencia y mantener un nivel adecuado de protección.

También, se definieron roles y permisos específicos para los distintos perfiles de usuario, que garantiza a cada actor poder acceder únicamente a las funcionalidades correspondientes a su nivel de autorización. Este enfoque contribuye a reforzar la seguridad del sistema y prevenir accesos indebidos al mismo.

Finalmente, se ejecutaron pruebas de seguridad y validación, destinadas a comprobar la eficacia de los mecanismos implementados y detectar posibles vulnerabilidades. Estas pruebas permitieron realizar ajustes finales en los procesos de autenticación y autorización antes de su integración definitiva.

Arquitectura y Tecnologías implementadas

La arquitectura del sistema KipBip quedó compuesta por tres componentes principales:

- Frontend Web (KipBip Web).
- Backend.
- Aplicación móvil (KipBip App).

El backend actúa como núcleo del sistema, centralizando la lógica de negocio, la validación de datos y la sincronización entre dos bases de datos heterogéneas: SQL Server [2] y Cloud Firestore [1]. Mientras que el backend opera como intermediario, la aplicación móvil continúa interactuando directamente con Cloud Firestore [1], por lo que la capa de sincronización resulta esencial para garantizar la consistencia global del modelo de datos.

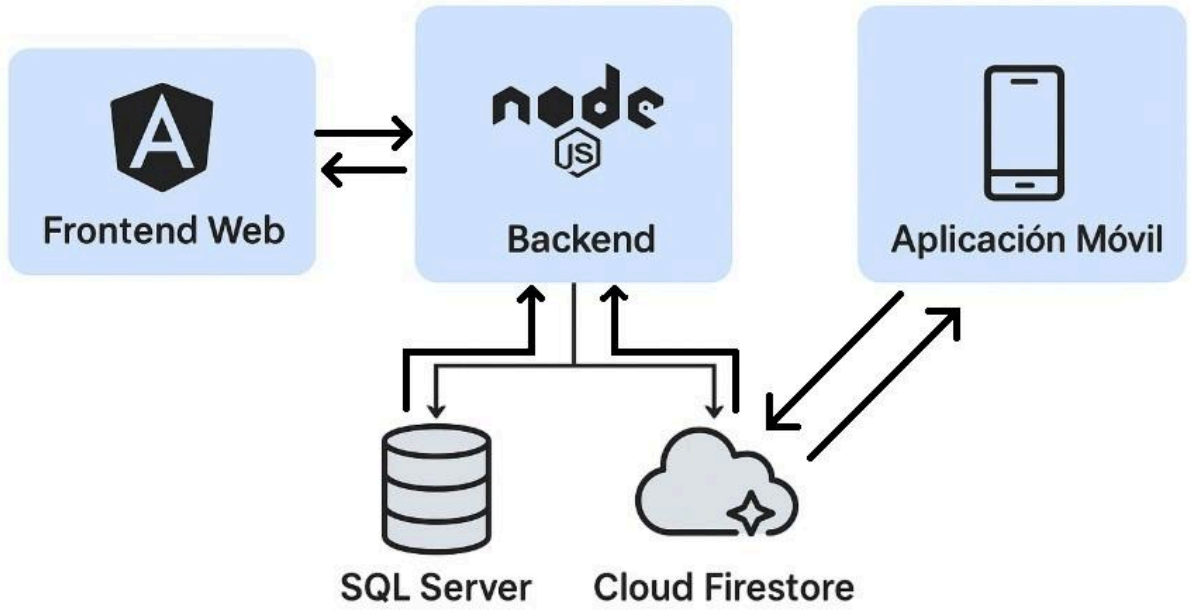

Fig. 2. Diagrama de arquitectura de KipBip (elaboración propia).

Frontend Web

El frontend se implementó con Angular 17 [14], aplicando un enfoque moderno basado en:

- Angular CLI [15] para automatización y estandarización, permitiendo generar artefactos del proyecto, gestionar builds optimizados y asegurar prácticas uniformes en todo el desarrollo.
- Signals y Effects para manejo reactivo de los estados.
- Interceptores para gestión de tokens, control de errores y estandarización de peticiones HTTP.
- Lazy Loading y Angular Router [16] para modularización y rendimiento, posibilitando una carga progresiva de componentes y rutas, disminuyendo el peso inicial de la aplicación y favoreciendo una arquitectura más escalable.
- Bootstrap 5 [17] y Angular Material [18] como frameworks de diseño. El lenguaje Sass (Syntactically Awesome Stylesheets) para estilos modulares y escalables.
- HERE Maps [19] para renderizado de mapas, geolocalización y consumo de APIs de ruteo.

Este cliente se comunica exclusivamente con el backend mediante servicios HTTPS, lo que permite mantener un acoplamiento bajo y garantizar la consistencia de la lógica del negocio en un único punto.

Backend

El backend se desarrolló en Node.js (v20.10.0) [8] utilizando TypeScript [9], lo cual permite trabajar con tipado fuerte, reduciendo errores y mejorando la mantenibilidad. La arquitectura interna sigue un modelo en capas, organizado de la siguiente manera:

- Middleware: autenticación con JWT [4], validación de permisos según roles, control de acceso a rutas, manejo de errores y estandarización de respuestas.
- Router: definición de endpoints y organización modular por dominio.
- Controller: recepción de solicitudes, mapeo de DTOs [12] y orquestación del flujo hacia la lógica del negocio.
- Service: implementación de la lógica de negocio, validación adicional

mediante reglas internas y coordinación con los repositorios.

- Repository: acceso a datos, consultas SQL parametrizadas y operaciones con Cloud Firestore [1].

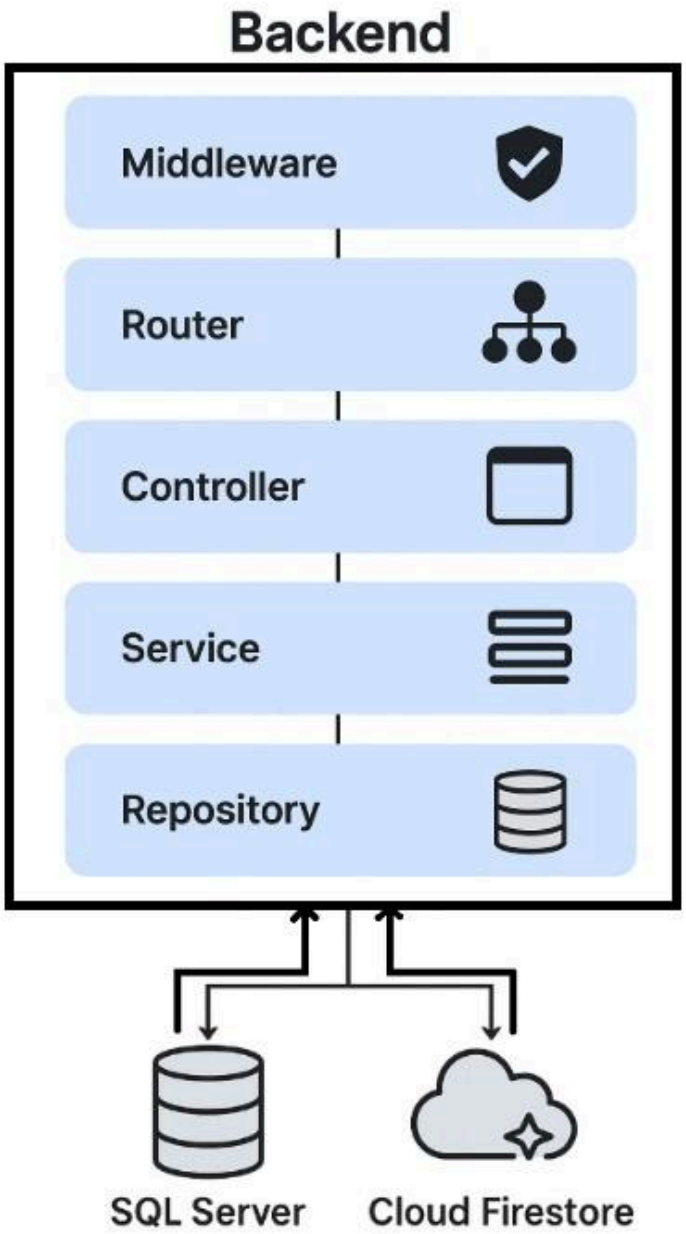

Fig. 3. Diagrama de arquitectura en capas de backend (elaboración propia).

Se integró class-validator [11] para asegurar que todas las solicitudes entrantes cumplieran con las reglas definidas en los DTO [12], fortaleciendo la integridad de

los datos antes de ingresar al flujo de negocio.

Aplicación móvil

La aplicación está desarrollada en Kotlin [20] siguiendo un patrón MVVM (Model-View-ViewModel) [21], adaptado a Jetpack Compose en Android [22], y se complementa con un enfoque funcional-reactivo mediante el uso de estados observables y funciones de intención.

Otras tareas complementarias

- Estandarización de convenciones: se establecieron un conjunto de convenciones de nomenclatura y organización del código, con el objetivo de mejorar la legibilidad, mantenibilidad y escalabilidad del sistema. Se utilizaron nombres descriptivos y consistentes para clases, componentes, servicios y demás estructuras, adoptando sufijos específicos según el tipo de elemento.
- Control de calidad y versión:

  Para el control de versiones y la gestión colaborativa del código se utilizó Git [23] como sistema de control distribuido, en conjunto con GitHub [24] como plataforma de repositorio remoto.

  Esta combinación permitió mantener un registro detallado de los cambios, garantizando la trazabilidad del código y facilitando el trabajo en paralelo con otros miembros del equipo. Se definió una estructura de ramas basada en buenas prácticas, aplicando un flujo con ramas principales (master), de desarrollo (develop) y ramas específicas para nuevas funcionalidades o correcciones.

  Cada cambio fue acompañado por mensajes de commit descriptivos, que documentan el propósito y alcance de la

modificación, asegurando un historial claro y auditable. Las revisiones de código fueron realizadas mediante pull requests en GitHub [24], donde el tutor de la empresa validaba las propuestas antes de su integración al entorno principal. Este proceso permitió mejorar la calidad del código, detectar errores tempranamente y mantener la coherencia con los lineamientos técnicos definidos para el proyecto. Adicionalmente, GitHub [24] se utilizó como medio de control de calidad mediante:

- Seguimiento de incidencias con issues.
- Uso de branches temporales para pruebas y validaciones.

Principales dificultades

Durante el desarrollo se presentaron diversas dificultades de naturaleza técnica. Estas situaciones influyen en los tiempos de avance, en los procesos de toma de decisiones y en la dinámica general del proyecto. Las mismas se detallan a continuación:

Curva de aprendizaje asociada a las tecnologías utilizadas

Uno de los desafíos iniciales estuvo vinculado al dominio de las tecnologías seleccionadas para el proyecto. El backend implementado en Node.js [8] con TypeScript [9] y Express [10], herramientas con las cuales no contaba con experiencia práctica previa. Si bien los conceptos habían sido abordados en la formación académica, su aplicación en un entorno real implicó un proceso de aprendizaje, especialmente en aspectos como:

- Uso de Express [10] y patrones de organización en Node.js [8].
- Sintaxis, tipado y buenas prácticas en TypeScript [9].

Falta de experiencia previa con bases de datos NoSQL

Otra dificultad importante surgió del trabajo con Cloud Firestore [1], una tecnología basada en documentos y colecciones que difiere de forma notable de los modelos relacionales tradicionales.

Si bien la formación académica brinda una introducción teórica y práctica básica a los modelos NoSQL, no contaba con experiencia aplicándolos en un entorno de producción. Esto derivó en desafíos relacionados con:

- Comprender la estructura y relaciones implícitas entre colecciones.
- Detectar redundancias y datos anidados sin normalización.

Futuros desafíos

De cara a la evolución del sistema, se identifican varios desafíos tecnológicos y organizativos que permitirán continuar fortaleciendo la escalabilidad, confiabilidad y capacidad de integración de la plataforma. Entre las líneas de trabajo propuestas se destacan:

Contenerización mediante Docker y despliegue con Kubernetes

Para mejorar la portabilidad de los entornos, facilitar el escalado y permitir despliegues más controlados, se prevé:

- Contenerizar el backend, frontend y base de datos con Docker [25] permitiendo estandarizar y aislar el entorno de ejecución del sistema, garantizando portabilidad entre ambientes, mayor consistencia en los despliegues y reducción de errores derivados de configuraciones locales.
- Migrar la ejecución a un orquestador como Kubernetes [26] permitiendo gestionar el despliegue, escalado y recuperación automática del sistema completo contenerizado, mejorando la disponibilidad, la administración operativa y la capacidad de respuesta ante incrementos de demanda.

- Implementar autoescalado, políticas de alta disponibilidad y monitoreo nativo.

Este enfoque permitiría romper gradualmente el monolito en componentes más independientes sin requerir una reescritura inmediata [27].

Incorporación de testing automatizado

Uno de los desafíos clave será mejorar la calidad del sistema mediante un esquema formal de pruebas automatizadas, que complemente y reemplace gradualmente a las pruebas manuales.

Se prevé incorporar:

- Tests unitarios con Jest [28] o Mocha [29].
- Tests de integración con Supertest [30] u otras herramientas para API REST.
- Tests end-to-end (E2E) [31] para probar el flujo completo del sistema.
- Ejecución automática de pruebas dentro del pipeline CI/CD.

Esto permitirá detectar regresiones de forma temprana, proteger funcionalidades críticas, elevar la confiabilidad del backend y asegurar que la plataforma pueda evolucionar sin comprometer su estabilidad.

Mejora de la documentación técnica

En futuras fases, será conveniente unificar y estandarizar la documentación del backend mediante herramientas basadas en OpenAPI/Swagger [32], lo que permitirá:

- Documentación navegable y centralizada.
- Descripciones formales de endpoints y modelos.
- Mayor claridad para frontend, QA y nuevos desarrolladores.

Esto representa una línea de evolución natural a medida que crece la complejidad del sistema.

Incorporación de pipelines CI/CD

La adopción de una estrategia de Integración Continua y Despliegue Continuo

permitirá automatizar pruebas, validaciones y despliegues, reduciendo errores humanos, acelerando la entrega de nuevas funcionalidades y garantizando mayor estabilidad durante las actualizaciones del sistema.

Conclusiones

A lo largo de las diez semanas de la PPS se llevaron a cabo actividades vinculadas a distintas etapas del ciclo de desarrollo de software, abarcando análisis, diseño, desarrollo, validación e implementación. Estas actividades permitieron aplicar de manera integrada los conocimientos adquiridos durante la formación académica, comprobándose la capacidad para abordar problemáticas reales del ámbito profesional y para desenvolverse dentro de un entorno productivo.

En este proceso se alcanzaron los objetivos planteados en el plan de trabajo, logrando la construcción de una base de datos relacional acorde a las necesidades actuales del producto y la implementación de un backend organizado, seguro y coherente con la nueva arquitectura tecnológica requerida por KipBip. Asimismo, se comprobó para KipBip Web la importancia de contar con un modelo de datos relacional y una arquitectura en capas para garantizar escalabilidad, mantenibilidad y consistencia en el manejo de la información.

Durante la práctica se desarrollaron competencias blandas y técnicas, entre las cuales se destacan:

- Trabajo en equipo interdisciplinario.
- Comunicación efectiva con perfiles técnicos y no técnicos.
- Capacidad de análisis crítico.
- Resolución de problemas.
- Responsabilidad en la toma de decisiones.
- El análisis de limitaciones del sistema, flujos de información y tecnologías existentes.
- El diseño y aplicación de arquitecturas en capas siguiendo buenas prácticas de ingeniería de software.

- La gestión, normalización y estructuración de bases de datos relacionales.
- La validación de datos mediante enfoques basados en DTOs [12] y reglas de negocio.
- La migración y sincronización entre sistemas heterogéneos (Cloud Firestore [1] y SQL Server [2]).
- La implementación de mecanismos de seguridad basados en JWT [6] y middlewares especializados.
- La ejecución de pruebas funcionales asegurando la calidad.

La práctica también permitió transitar de manera integral algunas etapas de la Ingeniería de Software, incluyendo la planificación, el relevamiento y análisis de requerimientos funcionales y no funcionales, el modelado y diseño de la solución, el desarrollo del código, las pruebas o testing y el mantenimiento de la misma, comprendiendo la importancia de cada fase dentro del ciclo de vida del software.
Desde el punto de vista funcional, se trabajó en la interpretación de necesidades, la traducción de procesos en soluciones técnicas y la validación conjunta con los diferentes individuos, garantizando responder efectivamente a los objetivos. En cuanto a la metodología de trabajo, debido al enfoque flexible,
me permitieron fortalecer competencias de organización, gestión del tiempo y adaptación al cambio.
Por otro lado, se superaron dificultades relevantes, como la falta de experiencia previa en tecnologías clave (Angular [14], Node.js [8] y Cloud Firestore [1]), la necesidad de interpretar un sistema existente sin documentación formal y la adaptación a un entorno empresarial con dinámicas exigentes y cambios constantes. Estas dificultades se resolvieron mediante investigación, trabajo colaborativo, revisión de buenas prácticas y construcción progresiva de soluciones.
En términos institucionales, el trabajo realizado aportó valor directo al proyecto, fortaleciendo su arquitectura técnica, mejorando la seguridad del sistema, rendimiento, costos, coherencia de datos y estableciendo bases sólidas para futuras funcionalidades. Al mismo tiempo, el proceso contribuyó significativamente a la

formación profesional, permitiendo consolidar competencias reales de análisis, diseño, programación, documentación y toma de decisiones en contextos productivos.

Bibliografía

[1] “Firestore”. Google Cloud. Accedido el 15 de enero de 2026. [En línea]. Disponible: https://cloud.google.com/products/firestore?hl=es

[2] “Microsoft SQL Server”. Microsoft – AI, Cloud, Produktivität, Computing, Gaming und Apps. Accedido el 15 de enero de 2026. [En línea]. Disponible: https://www.microsoft.com/es-es/sql-server

[3] “SOLID: Los 5 principios que te ayudarán a desarrollar software de calidad”. Profile Software Services. Accedido el 15 de enero de 2026. [En línea]. Disponible: https://profile.es/blog/principios-solid-desarrollo-software-calidad/

[4] “JSON Web Tokens - jwt.io”. JSON Web Tokens - jwt.io. Accedido el 15 de enero de 2026. [En línea]. Disponible: https://www.jwt.io/

[5] “PostgreSQL”. PostgreSQL. Accedido el 12 de febrero de 2026. [En línea]. Disponible: https://www.postgresql.org/

[6] “MariaDB Foundation - MariaDB.org”. MariaDB.org. Accedido el 12 de febrero de 2026. [En línea]. Disponible: https://mariadb.org/

[7] B. Digital. “ACID Properties In DBMS: A Guide to ACID Transactions”. Yugabyte. Accedido el 12 de febrero de 2026. [En línea]. Disponible: https://www.yugabyte.com/key-concepts/acid-properties/

[8] “Node.js — Run JavaScript Everywhere”. Node.js — Run JavaScript Everywhere. Accedido el 15 de enero de 2026. [En línea]. Disponible: https://nodejs.org/es

[9] “The starting point for learning TypeScript”. TypeScript: JavaScript With Syntax For Types. Accedido el 31 de enero de 2026. [En línea]. Disponible: https://www.typescriptlang.org/docs/

[10] “Express routing”. Express - Node.js web application framework. Accedido el 31 de enero de 2026. [En línea]. Disponible: https://expressjs.com/en/guide/routing.html

[11] “GitHub - typestack/class-validator: Decorator-based property validation for classes.” GitHub. Accedido el 15 de enero de 2026. [En línea]. Disponible: https://github.com/typestack/class-validator#readme

[12] “Data Transfer Object | Architectural Patterns”. Home | Architectural Patterns. Accedido el 1 de marzo de 2026. [En línea]. Disponible: https://architectural-patterns.net/data-transfer-object

[13] “Postman: The World's Leading API Platform | Sign Up for Free”. Postman: The World's Leading API Platform | Sign Up for Free. Accedido el 12 de febrero de 2026. [En línea]. Disponible: https://www.postman.com/

[14] “Angular”. Home • Angular. Accedido el 15 de enero de 2026. [En línea]. Disponible: https://v17.angular.io/docs

[15] “Angular CLI • overview • angular”. Home • Angular. Accedido el 31 de enero de 2026. [En línea]. Disponible: https://angular.dev/tools/cli

[16] “Routing • overview • angular”. Home • Angular. Accedido el 31 de enero de 2026. [En línea]. Disponible: https://angular.dev/guide/routing

[17] “Get started with Bootstrap”. Bootstrap · The most popular HTML, CSS, and JS library in the world. Accedido el 31 de enero de 2026. [En línea]. Disponible: https://getbootstrap.com/docs/5.3/getting-started/introduction/

[18] Angular. “Angular Material”. Angular Material. Accedido el 31 de enero de 2026. [En línea]. Disponible: https://material.angular.dev/guides

[19] “HERE Technologies Documentation | HERE Docs”. HERE Technologies | The world's #1 location platform. Accedido el 15 de enero de 2026. [En línea]. Disponible: https://www.here.com/docs/

[20] “Kotlin Docs | Kotlin”. Kotlin Help. Accedido el 12 de febrero de 2026. [En línea]. Disponible: https://kotlinlang.org/docs/home.html

[21] “Modelo-Vista-Modelo de vista - .NET”. Microsoft Learn: Build skills that open doors in your career. Accedido el 12 de febrero de 2026. [En línea]. Disponible: https://learn.microsoft.com/es-es/dotnet/architecture/maui/mvvm

[22] “Instructivo de Android Compose | Jetpack Compose | Android Developers”. Android Developers. Accedido el 12 de febrero de 2026. [En línea]. Disponible: https://developer.android.com/develop/ui/compose/tutorial?hl=es-419

[23] “Git - git Documentation”. Git. Accedido el 15 de enero de 2026. [En línea]. Disponible: https://git-scm.com/docs/git

[24] “GitHub.com Documentación de ayuda”. GitHub Docs. Accedido el 15 de enero de 2026. [En línea]. Disponible: https://docs.github.com/es

[25] Docker Inc. “Home”. Docker Documentation. Accedido el 13 de febrero de 2026.

[En línea]. Disponible: https://docs.docker.com/

[26] “Production-Grade Container Orchestration”. Kubernetes. Accedido el 13 de febrero de 2026. [En línea]. Disponible: https://kubernetes.io/

[27] Accedido el 15 de enero de 2026. [En línea]. Disponible: https://docs.aws.amazon.com/es_es/prescriptive-guidance/latest/modernization-decomposing-monoliths/modernization-decomposing-monoliths.pdf

[28] “Getting Started · Jest”. Jest · Delightful JavaScript Testing. Accedido el 13 de febrero de 2026. [En línea]. Disponible: https://jestjs.io/docs/next/getting-started

[29] “Mocha”. Mocha. Accedido el 13 de febrero de 2026. [En línea]. Disponible: https://mochajs.org/

[30] “GitHub - forwardemail/supertest: Super-agent driven library for testing node.js HTTP servers using a fluent API. Maintained for @forwardemail, @ladjs, @spamscanner, @breejs, @cabinjs, and @lassjs.” GitHub. Accedido el 13 de febrero de 2026. [En línea]. Disponible: https://github.com/forwardemail/supertest#readme

[31] P. Powell y I. Smalley. “End-to-End Testing Best Practices | IBM”. IBM. Accedido el 13 de febrero de 2026. [En línea]. Disponible: https://www.ibm.com/think/insights/end-to-end-testing-best-practices

Agradecimientos

Quiero agradecer en primer lugar a mi familia, Oscar, Claudia, Karen, Denise, y Simon por estar presentes en mi vida, acompañarme en cada detalle, ser mi inspiración y motivación. También, agradecer a mi pareja que me brindó su apoyo incondicional en esta última etapa de la carrera.

Por otro lado, agradecer a la Universidad Nacional del Noroeste de la Provincia de Buenos Aires (UNNOBA) y a todo el personal involucrado (profesores, personal interno, compañeros, administrativos, etc.) por el acompañamiento durante toda la carrera.

Extiendo mi agradecimiento a mi docente tutora Mariana Adó por su acompañamiento y compromiso durante todo el proceso de la PPS. Su apoyo fue fundamental para integrar el aprendizaje académico con la experiencia laboral.

Agradezco a la empresa KipBip por hacer posible este trabajo y brindarme la oportunidad de formar parte de un equipo de desarrollo de software.
Asimismo, agradezco a mis compañeros de equipo Juan y Augusto por su compromiso y el excelente clima laboral. Particularmente agradezco a Juan por ser un gran líder y por su predisposición durante todo el proceso de este trabajo.

Finalmente agradezco a toda mi familia y amigos que siempre estuvieron presentes de una u otra forma. Gracias por el apoyo que me brindaron.