



UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

Desarrollo de Back-End para aplicaciones de gestión de estacionamientos de Viña del Mar

Benjamín Ramírez Ibarra

`benjamin.ramirezi@usm.cl`

Carlos Alten López
Profesor Guía

Resumen: El siguiente texto detalla el desarrollo de un Back-End modular y escalable para la gestión de estacionamientos, utilizando diversas tecnologías. Incluye información sobre la arquitectura del sistema, estrategias de pruebas aplicadas con Postman, y procesos de resolución de bugs y optimización. Se destacan logros como la integración efectiva de los componentes y la preparación para futuras expansiones, como la incorporación de sistemas de IA para monitoreo y pagos automatizados, con el fin de solucionar el problema de la falta de un sistema para la gestión de estacionamientos que se ve afectada debido a la creciente presencia de vehículos que aflige a la Municipalidad de Viña del Mar, generando conflictos y malas experiencias para tanto los residentes como para los turistas. También se abordan los desafíos enfrentados, decisiones tomadas durante el desarrollo, y lecciones aprendidas que guiarán proyectos futuros.

Palabras Clave: Aplicación, Estacionamientos, Viña del mar, Back-End.



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Desarrollo de Back-End para aplicaciones de gestión de estacionamientos de Viña del Mar

Nombre del candidato(a): Benjamin Eduardo Ramirez Ibarra

Carrera / Grado: Ingenieria en Informatica

Campus: Viña del Mar ; Departamento: Electrónica e Informática (Elinf)

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Carlos Alten L., en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (**marcar una opción**):

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 7 de octubre de 2026

Firma:

Estudiante o Candidato(a):

Fecha: 05-10-2026

Firma:

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

1 Introducción

1.1 Contexto y antecedentes

La ciudad de Viña del Mar, conocida como uno de los principales destinos turísticos de Chile, enfrenta un desafío creciente relacionado con la gestión de estacionamientos. La elevada afluencia de vehículos, tanto de residentes como de visitantes, ha generado una saturación en los espacios disponibles para aparcar, especialmente en zonas céntricas y durante temporadas de alta demanda turística. Este problema no solo afecta la experiencia de los conductores, sino que también impacta negativamente en la movilidad urbana, contribuyendo a la congestión vehicular y al aumento de la contaminación ambiental.

Actualmente, la gestión de los estacionamientos en la ciudad carece de una infraestructura tecnológica unificada que permita a los usuarios acceder a información en tiempo real sobre la disponibilidad de espacios. Este vacío genera una serie de ineficiencias que incluyen la pérdida de tiempo en la búsqueda de estacionamientos, mayores costos operativos para los conductores y dificultades administrativas para las autoridades municipales al intentar regular y fiscalizar el uso de estos espacios.

Por otro lado, las ciudades modernas están adoptando cada vez más soluciones tecnológicas para abordar estos problemas, desarrollando sistemas inteligentes que integran aplicaciones móviles, sensores, cámaras y sistemas de bases de datos para gestionar los estacionamientos de manera eficiente. Sin embargo, en el caso de Viña del Mar, la implementación de una solución de este tipo sigue siendo limitada o inexistente, lo que representa una oportunidad significativa para innovar y mejorar la experiencia de los usuarios, optimizando la administración de recursos y reduciendo el impacto en la movilidad.

Con este contexto en mente, el presente proyecto propone un sistema por capas (Front-end, Back-end, Base de Datos) donde en el presente documento se aborda el diseño y desarrollo de un sistema Back-End robusto que soporte una solución integral para la gestión de estacionamientos en Viña del Mar. Este sistema permitirá a los conductores acceder, a través de una aplicación móvil, a información en tiempo real sobre la disponibilidad de estacionamientos, calcular rutas hacia estos espacios y gestionar su estadía mediante un temporizador que facilitará el cálculo del cobro al finalizar el uso.

1.2 Definición del problema.

En el contexto de la gestión de estacionamientos públicos en Viña del Mar, se han identificado los siguientes problemas que afectan tanto la eficiencia del sistema como la experiencia de los usuarios finales:

Falta de información en tiempo real:

Actualmente, los conductores no cuentan con un sistema que les permita conocer la disponibilidad de espacios libres en tiempo real, lo que los obliga a buscar estacionamientos de manera aleatoria, aumentando el tiempo y los costos asociados.

Ausencia de un sistema unificado:

No existe un sistema centralizado que permita a la municipalidad administrar y regular los estacionamientos públicos y privados, lo que genera desorganización y dificulta la toma de decisiones basadas en datos.

Congestión vehicular:

La búsqueda prolongada de estacionamientos contribuye a la congestión del tráfico, especialmente en horas punta, afectando tanto a residentes como a turistas.

Impacto ambiental:

El fenómeno conocido como “cruising for parking” incrementa las emisiones de gases de efecto invernadero debido al consumo innecesario de combustible durante la búsqueda de estacionamientos, aumentando la huella de carbono de la ciudad.

Falta de regulación de tarifas:

No existe un sistema eficiente para implementar tarifas diferenciadas según la residencia de los usuarios o la demanda en tiempo real, lo que podría optimizar los ingresos municipales y distribuir mejor el uso de los espacios.

Inadecuada experiencia del usuario:

Los conductores enfrentan una experiencia poco intuitiva y frustrante debido a la falta de tecnología que facilite la localización, reserva y pago de los estacionamientos.

Ineficiencia en la administración:

La municipalidad carece de herramientas avanzadas para analizar estadísticas sobre el uso de los estacionamientos, lo que limita la capacidad de planificar mejoras en infraestructura o implementar soluciones más eficientes.

Dependencia de procesos manuales:

La ausencia de sistemas automatizados, como cámaras con análisis inteligente para monitorear la ocupación, prolonga los tiempos de respuesta e incrementa la carga operativa de la gestión del sistema.

Riesgos de seguridad en los pagos:

Los métodos de pago actuales no garantizan una experiencia segura y eficiente, lo que puede desincentivar el uso del sistema y exponer a los usuarios a riesgos innecesarios.

Falta de conectividad con otras plataformas:

La inexistencia de integraciones con sistemas de mapas o herramientas de movilidad limita el acceso a funcionalidades clave, como la navegación directa hacia los estacionamientos disponibles.

1.3 Breve descripción sobre la propuesta de solución

Como solución al problema identificado, se propone el desarrollo de un sistema integral que incluye una aplicación móvil para conductores y una plataforma web para la administración municipal. La aplicación móvil estará diseñada para que residentes y turistas puedan:

1. Consultar en tiempo real la disponibilidad de espacios de estacionamiento en diversas zonas de la ciudad.
2. Visualizar información detallada de cada estacionamiento, como ubicación, tarifas y distancia.
3. Utilizar un sistema de navegación integrado que guíe al usuario directamente al estacionamiento seleccionado.
4. Temporizar su estadía en el estacionamiento, facilitando el cálculo automatizado del cobro al momento de la salida.
5. Realizar pagos electrónicos de forma segura y rápida a través de la aplicación.

Por otro lado, se desarrollará una aplicación web destinada a la Municipalidad de Viña del Mar, que funcionará como un panel de control administrativo. Este dashboard permitirá:

1. Monitorear el estado de ocupación de los estacionamientos en tiempo real.
2. Analizar estadísticas detalladas sobre el uso de los estacionamientos, como las horas y días de mayor demanda, con información segmentada por zonas.
3. Implementar y gestionar tarifas dinámicas basadas en la demanda, horarios y características de los usuarios (residentes o turistas).
4. Obtener reportes históricos que faciliten la toma de decisiones para la optimización de los recursos de estacionamiento.

El sistema estará respaldado por un Back-End robusto que actúa como el núcleo central, asegurando el intercambio eficiente de información entre los usuarios, el servidor y la base de datos. Este Back-End será escalable, seguro y estará diseñado para manejar grandes volúmenes de datos en tiempo real, garantizando una experiencia fluida para los usuarios y un control eficaz para las autoridades municipales.

Las funcionalidades del Back-End se centran en el uso de la lógica de negocio y conexión con la base de datos y el funcionamiento interno (login, autenticación, API, seguridad, monitoreo, entre otros).

1.4 Objetivos Generales y Específicos de la Tesina

1.4.1 Objetivo General

Desarrollar un sistema de Back-End eficiente para la gestión de estacionamientos públicos de Viña del Mar, que optimice el almacenamiento, procesamiento y acceso a datos en tiempo real, permitiendo un control efectivo de la disponibilidad y asignación de los espacios.

1.4.2 Objetivos Específicos:

1. **Implementar un sistema de autenticación y autorización seguro** que permita la diferenciación de perfiles de usuario (residente, turista, administrador) y garantice el acceso controlado a las funcionalidades de la plataforma, tanto en la aplicación móvil como en la plataforma web.
2. **Diseñar y documentar una API REST eficiente y segura** para la gestión de datos de los estacionamientos, que permita la consulta y actualización en tiempo real de la disponibilidad de espacios, así como la integración con otros módulos del sistema.
3. **Desarrollar un sistema de cobro dinámico y diferenciado** que permita calcular tarifas según la residencia del usuario y el tiempo de estacionamiento, asegurando una experiencia de pago fluida y justa.
4. **Implementar una estrategia de manejo de errores y monitoreo continuo** del sistema Back-End, para detectar y solucionar posibles fallos o caídas, mejorando la fiabilidad del sistema en tiempo real.
5. **Desarrollar un sistema de visualización de mapas** que permita a los usuarios ver la ubicación exacta de los estacionamientos disponibles, con integración de datos de geolocalización y actualizaciones en tiempo real.
6. **Crear un módulo de registro y gestión de usuarios** que permita a los conductores y administradores crear y actualizar sus perfiles, asegurando un flujo de registro sencillo y seguro.
7. **Implementar un sistema de gestión eficiente de los estacionamientos** que permita a los administradores gestionar la disponibilidad, reservas y pagos de los espacios, con soporte para tarifas diferenciadas según la demanda.
8. **Integrar un sistema de pagos seguro y eficiente** dentro de la aplicación, permitiendo a los usuarios realizar transacciones electrónicas de manera confiable y rápida para el cobro de tarifas de estacionamiento.

1.5 Justificación de proyecto

A nivel técnico, este proyecto es esencial porque resuelve un problema común mediante soluciones avanzadas. El sistema de Back-End es el núcleo de toda la operación: la lógica de calcular tarifas diferenciadas y el estado en tiempo real de los estacionamientos. Por otro lado, los usuarios finales estarán contentos con una experiencia que no solo es relativamente libre de fricción, sino que también se ejecuta óptimamente en una plataforma segura, en crecimiento y rápida. También mejora la administración efectiva de un recurso que siempre es limitado como lo son los estacionamientos.

El uso de tecnologías como el API REST permite que los diversos componentes del sistema se comuniquen, lo que le da, a la aplicación móvil de reservas, la posibilidad de consultar en tiempo real si hay espacios disponibles y a su vez les da a los administradores la posibilidad de ver en esos espacios los indicadores de desempeño (estadísticamente cuanto tiempo se usó, cuál es el estacionamiento más usado, etc.). La elección de MySQL como base de datos relacional posibilita el trabajo con grandes volúmenes de datos estructurados, como es el caso de las transacciones de los usuarios que reservan y cancelan plazas, garantizando integridad y consistencia en las operaciones

1.5.1 Beneficios

El desarrollo del Back-End para la gestión de estacionamientos en Viña del Mar traerá una serie de beneficios clave, tanto para los usuarios finales como para las autoridades municipales, mejorando la eficiencia y sostenibilidad del sistema de estacionamientos en la ciudad.

1. **Eficiencia operativa:** El sistema permitirá gestionar miles de transacciones simultáneamente, lo que optimiza el uso de los estacionamientos disponibles en tiempo real. Esto no solo reducirá la congestión vehicular, sino que también evitará la saturación del sistema, garantizando una experiencia fluida tanto para conductores como para administradores. La automatización de procesos como la asignación de espacios y el cobro de tarifas reducirá la intervención manual, mejorando la productividad y reduciendo los costos operativos.
2. **Acceso a información en tiempo real:** El principal beneficio que se obtiene al tener información a tiempo real es que el usuario podrá saber dónde hay (y a su vez no hay) estacionamientos disponibles logrando así disminuir la congestión. Además, el Back-End proporcionará a la municipalidad datos actualizados sobre la ocupación y demanda de los espacios, facilitando la toma de decisiones informadas sobre la asignación de recursos y la optimización de la infraestructura de estacionamientos.
3. **Escalabilidad:** La solución está diseñada para ser escalable, lo que significa que podrá adaptarse al crecimiento futuro de la ciudad. A medida que aumente el número de vehículos y estacionamientos, el sistema podrá ampliarse sin necesidad de una reingeniería completa. Esta capacidad de adaptación asegurará que el sistema siga siendo efectivo a medida que Viña del Mar crezca, tanto en términos de infraestructura como en la demanda de estacionamientos.
4. **Mejora en la sostenibilidad y reducción de la huella de carbono:** La solución contribuye a la sostenibilidad urbana al reducir la necesidad de "cruising for parking" (búsqueda de estacionamiento), lo que disminuirá las emisiones de gases contaminantes. Al optimizar el uso de los espacios de estacionamiento y reducir el tiempo dedicado a la búsqueda, se contribuirá a una ciudad más verde y a una reducción de la congestión vehicular.
5. **Mayor seguridad y control:** La implementación de mecanismos de autenticación robustos y el uso de tecnologías como cámaras de vigilancia e inteligencia artificial (IA) mejorarán la seguridad tanto para los usuarios como para los administradores. La automatización de la actualización de datos y la identificación de incidentes contribuirán a un entorno más seguro, con una respuesta más rápida ante situaciones de emergencia o irregularidades.
6. **Monitoreo y análisis avanzado:** El sistema ofrecerá herramientas de análisis y monitoreo en tiempo real, permitiendo a las autoridades municipales gestionar de manera eficiente la demanda de estacionamientos, aplicar tarifas dinámicas y prever necesidades futuras. Este análisis avanzado también permitirá detectar patrones de comportamiento, como las épocas de mayor demanda, optimizando así los recursos disponibles.

1.5.2 Innovación

La solución propuesta incorpora varias innovaciones tecnológicas que la hacen única y eficiente para la gestión de estacionamientos en una ciudad dinámica como Viña del Mar:

- **Tarifas dinámicas basadas en demanda y residencia:** El sistema innovará en el aspecto de la gestión de tarifas, utilizando un algoritmo que ajuste las tarifas en función de la demanda en tiempo real y la residencia del usuario. Esta implementación asegurará que los precios se

ajusten automáticamente según la disponibilidad de espacios y la zona en la que se ubique el estacionamiento, creando un modelo más justo y eficiente de cobro.

- **Integración con plataformas de movilidad:** La solución incluirá la integración con otras plataformas de movilidad, como aplicaciones de mapas y navegación, para permitir a los usuarios obtener indicaciones precisas hacia el estacionamiento más cercano disponible. Esta integración no solo facilitará la experiencia del usuario, sino que también mejorará la eficiencia general del sistema de transporte urbano.
- **Accesibilidad y facilidad de uso:** La propuesta incluye la creación de una interfaz sencilla y accesible tanto para usuarios como para administradores, con un diseño intuitivo que facilite la consulta de información y la realización de transacciones. La accesibilidad es clave para garantizar que el sistema sea fácil de usar para todas las personas, independientemente de su nivel de experiencia tecnológica.

1.6 Metodología

Debido a que el tiempo es limitado y acotado, se necesita una metodología ágil para llevar a cabo el proyecto de forma óptima. Por esta razón se eligió Scrum ya que con este podemos tener entregas rápidas llamadas “Sprints” dándonos un feedback temprano del cliente, aparte de tener la opción de un trabajo grupal más eficiente, ya que fomenta la comunicación con reuniones diarias, ayudando a su vez a la priorización de tareas y posibilidad de cambios a medida que se completa el proyecto. [1]

Generando 3 Sprints y reuniones constantes de avances y de priorización de tareas, además de la asignación de roles y un programa para marcar las tareas tanto en proceso como finalizadas a través de la plataforma de “Trello”

1.7 Descripción de la organización del informe

El informe se compone de 5 puntos que van avanzando de manera progresiva la cual ayudara al lector con el entendimiento sobre el desarrollo del proyecto. A continuación, una breve descripción de los puntos a presentar:

- **Marco Teórico:** Donde se verá de manera más técnica todo lo que se usara, porque se usara y donde se usaran, pasando por arquitecturas de software presentar: Metodologías, Framework, entre otros
- **Diseño e Implementación:** En este punto veremos una bitácora de cómo se crea el sistema Back-End y como se implementará, además de mostrar su funcionamiento y rendimiento.
- **Conclusiones:** Cierre de la tesina dando conclusión con el tema y haciendo una breve retrospectiva del proyecto
- **Agradecimientos:** Apartado donde se le dará los respectivos gracias a las personas que me apoyaron en este proceso.
- **Anexos:** Material complementario

2 Marco Teórico

En este punto se presenta de manera técnica las elecciones de las tecnologías elegidas para el desarrollo de la solución propuesta, esto incluye el framework de Springboot en lenguaje JAVA, la selección de la base de datos SQL, la arquitectura, además de describir la tecnología APIRest y la metodología de trabajo SCRUM

2.1 Fundamentos del desarrollo back-end

El Back-End es la parte del sistema que el usuario nunca ve, pero sin la cual ninguna aplicación funcionaría. Mientras el Front-End se encarga de mostrar botones, formularios y páginas, todo lo que ocurre detrás de verificar datos, aplicar reglas y guardar información es responsabilidad del Back-End. Para entenderlo de forma simple: cuando alguien hace clic en "iniciar sesión", lo que valida su contraseña es el trabajo del Back-End. Hay un servidor recibiendo esa solicitud, revisando los datos y devolviendo una respuesta. Ese proceso ocurre en milisegundos y el usuario jamás lo nota, pero es el corazón de cualquier sistema. El diagrama siguiente muestra dónde se ubica el Back-End dentro de un sistema típico:

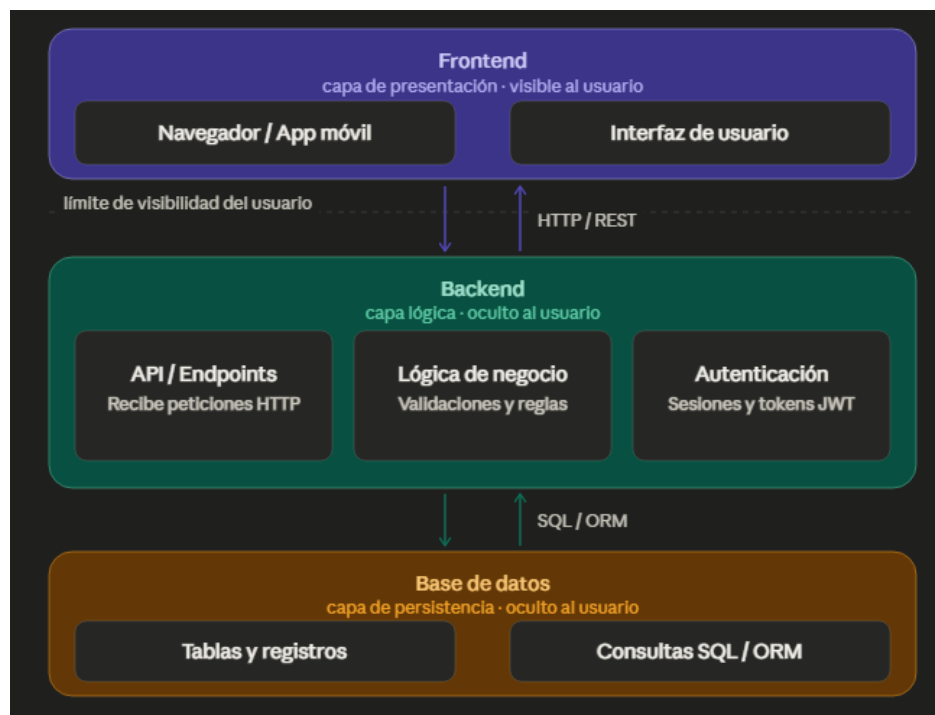


Figura 1. Diagrama Visual de Trabajo de un Sistema

Tomando lo anterior en cuenta las tres responsabilidades principales son:

- Exponer una API que recibe peticiones enviados por el Front-End
- Ejecutar la lógica de negocio (validaciones, cálculos, autenticación)
- Comunicarse con la base de datos para leer o escribir información.

Data Flow in Software

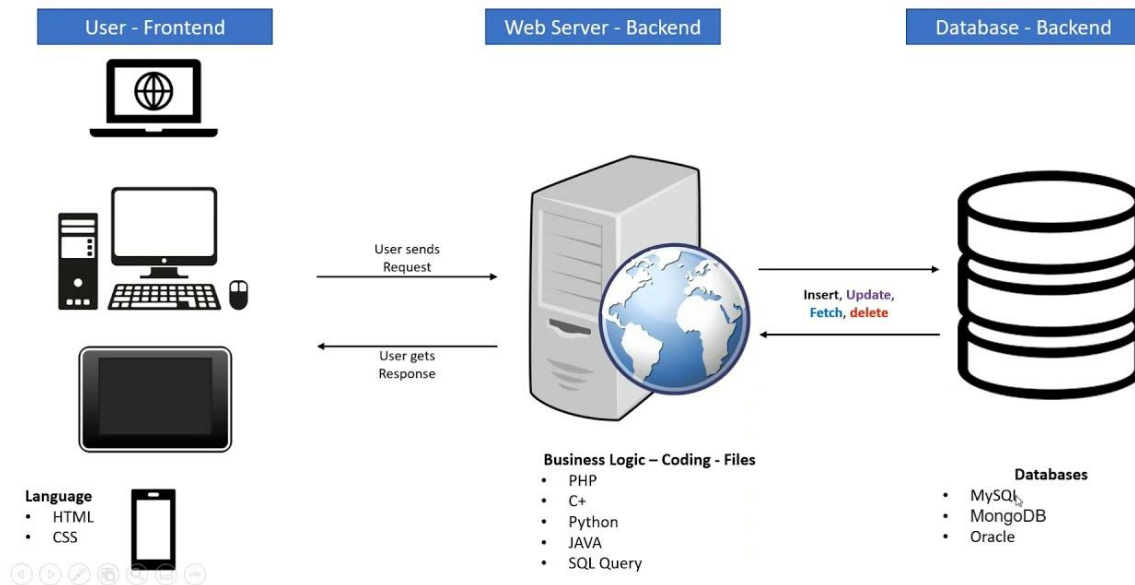


Figura 2. Diagrama Visual de los sistemas

Fuente: "What is Frontend and Backend: Data Flow in Software: What is Software"

2.2 Arquitectura de software

La arquitectura por utilizar se centrará en 3 capas de las cuales son Front-End, Back-End, Base de Datos. Con esta al separar la aplicación en diferentes capas se busca una fácil distribución en las tareas a realizar, ayudando también en su mantenimiento y en la adaptabilidad tecnológica (ya sea de una posible migración en un futuro), este siendo un sistema muy popular en el momento de elegir un Back-End. [2]

- **Capa de Presentación (Front-End):** La capa está conformada por la aplicación móvil y web que será utilizada por los usuarios finales: conductores, administradores, y fiscalizadores. Los usuarios a través de esta capa interactuarán con el sistema para consultar disponibilidad de estacionamientos, hacer reservas, pagos, y recibir reportes. La aplicación móvil y web se comunica con el Back-End a través de APIs REST.
- **Capa de Negocio (Back-End):** El Back-End es la capa que el usuario final no lograra ver, pero es de las más importantes debido a que todo el funcionamiento u orden que manda el usuario a través del Front-End pasara por el Back-End con sus respectivos trabajos mandando solicitudes a la base de datos.
- **Capa de Datos (Base de Datos):** Esta capa es donde de almacena y de administra los datos, mandando las respuestas a las solicitudes del Back-End, agregando o eliminando datos y siendo de vital apoyo para la elaboración de un futuro Dashboard.

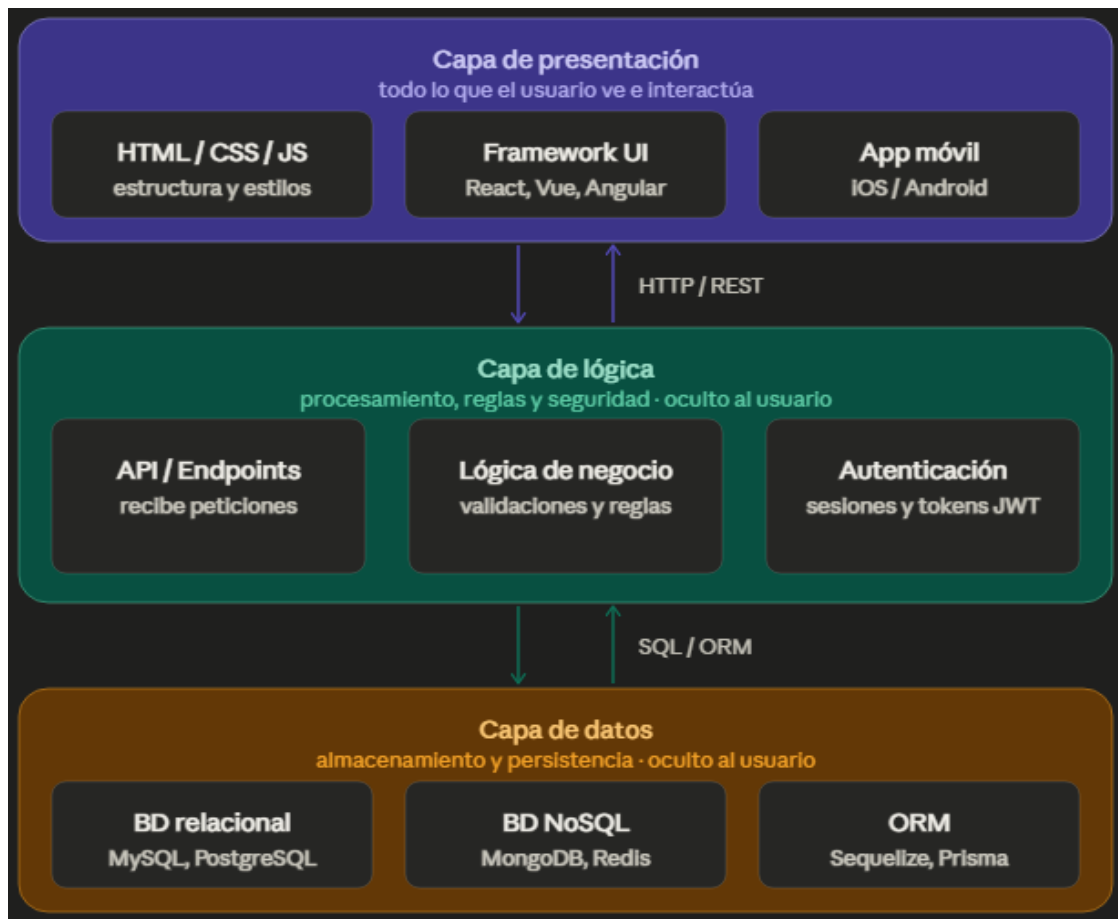


Figura 3. Diagrama Visual resumido de Arquitectura del Software

2.3 Tecnologías y Frameworks utilizados

En este apartado se verán las tecnologías (programas, herramientas, entre otros) y el framework (conceptos, biblioteca, herramientas, lenguaje) a utilizar y el porqué de estos.

2.3.1 Lenguajes de Programación Utilizado

El lenguaje que se usara para el back-end es bastante importante ya que es el pilar del funcionamiento de la aplicación, necesitamos que el lenguaje tenga una opción a escalabilidad a nivel regional o más. Pudiendo soportar un uso de más de miles de personas o hasta cientos de miles.

Tras una investigación y deliberación quedaron como principales opciones PHP, Java o Python. Del cual el vencedor fue Java, con su framework de Spring Boot

- **¿Por Qué Java?**

La decisión de elegir Java como lenguaje para el Back-End se debió a varios factores. El primero fue que el equipo estaba familiarizado con el lenguaje, lo que redujo la curva de aprendizaje y nos permitió avanzar más rápido en el desarrollo. La segunda fue su orientación a objetos, que se adapta bien a la estructura del sistema y facilita el mantenimiento del código a medida que crece el proyecto.

Para la conexión con MySQL se utilizó MySQL Connector/J junto con Spring Data JPA, dependencias muy probadas y con soporte activo dentro del ecosistema Spring Boot. Escoger Spring Boot en concreto facilitó la configuración del servidor, la administración de rutas y la conexión con la base de datos gracias a su sistema de autoconfiguración y su estructura modular, lo cual redujo los tiempos de implementación del Back-End. [3]

Finalmente, su compatibilidad con Flutter en el Front-End mediante una API REST estándar no presentó ninguna restricción, ya que Flutter consume JSON independiente del lenguaje que use el servidor.

- **Comparación con PHP o Python**

Como se dijo anteriormente se necesitaba un lenguaje que al momento de que la aplicación escalara y aumentara el tráfico no diera problemas.

Por mucho que el escribir dentro del lenguaje de Java sea más engorroso, Java se va por delante en los dos aspectos que requiere el programa, mayor rendimiento con baja carga en el sistema, sólido modelo de orientación a objetos y la indiscutible variedad de documentación que sigue creciendo de este lenguaje, que concretan que sea la elección más adecuada para asegurar la estabilidad y crecimiento de la aplicación a largo plazo. [4]

2.3.2 Herramientas utilizadas

1. **API REST:** Esta tecnología es la encargada para la comunicación entre Back-End y Front-End a través del protocolo HTTP:

- Se implementarán API RESTful para garantizar una comunicación fluida entre el Back-End y las aplicaciones móviles y web (Front-End). Estas APIs serán responsables de gestionar solicitudes como la consulta de disponibilidad de estacionamientos, la creación de reservas, el cálculo de tarifas y la confirmación de pagos.
- Las APIs estarán diseñadas con seguridad y escalabilidad en mente, utilizando estándares como JWT (JSON Web Tokens) para autenticación y autorización.

Esta tecnología es bastante utilizada en el mercado actual llegando a que el 90% de los desarrolladores utilizan API's. [5]

2. **Google App Engine:** Es una plataforma como servicio ofrecido por Google Cloud que permite a los desarrolladores crear y desplegar aplicaciones web y Back-End sin tener que gestionar directamente la infraestructura subyacente. [6]

- El Back-End será desplegado en Google App Engine, una plataforma como servicio (PaaS) que proporciona escalabilidad automática, balanceo de carga y un entorno administrado que reduce la necesidad de configuraciones manuales.
- App Engine permite gestionar picos de tráfico al escalar dinámicamente los recursos, asegurando un alto rendimiento incluso en momentos de alta concurrencia.

- Además, facilita la integración con otros servicios de Google Cloud, como Cloud SQL, Cloud Storage, y Cloud Monitoring.

Esta plataforma fue seleccionada frente a otras opciones debido a su escalabilidad nativa, su integración directa con los demás servicios utilizados en el proyecto y su facilidad de despliegue continuo.

3. **MySQL:** Base de datos relacional permite crear, leer, actualizar y eliminar datos estructurados en tablas, usando el lenguaje SQL.

- Se utilizará una base de datos relacional MySQL alojada en Google Cloud SQL, lo que garantiza su integración nativa, alta disponibilidad, respaldo automático y recuperación ante desastres.
- MySQL manejará datos críticos, como usuarios, reservas, historial de transacciones y estadísticas de uso de estacionamientos, asegurando consistencia e integridad de la información.

4. Herramientas de testing

Para validar el correcto funcionamiento del sistema, se realizarán pruebas exhaustivas utilizando:

- **Pruebas unitarias:** Validar el comportamiento de componentes individuales, como servicios o controladores.
- **Pruebas de integración:** Asegurar que las APIs funcionen correctamente al interactuar con la base de datos y otros servicios.
- **Pruebas de estrés:** Evaluar el rendimiento del sistema bajo altas cargas de tráfico.

Herramientas como Postman y JUnit serán utilizadas para automatizar y gestionar estas pruebas.

Postman: Programa para hacer pruebas de las Apis, donde al ingresar los datos logra hacer pruebas de funcionalidad

JUnit: Framework de pruebas unitarias de JAVA, que permite probar las clases y sus métodos.

5. Control de versiones con GitHub

Se utilizará **GitHub** como plataforma para el control de versiones, lo que permitirá:

- Gestión eficiente de ramas para separar los desarrollos del Front-End y Back-End.
- Historial detallado de cambios con posibilidad de revertir a versiones anteriores en caso de errores (plan de roll-back).
- Colaboración fluida entre los equipos de desarrollo mediante *pull requests* y revisiones de código.

6. Integración de mapas con Google Maps API

Para la visualización de ubicaciones de estacionamientos y la navegación en tiempo real, se integrará **Google Maps API** en el sistema. Esta herramienta permitirá:

- Mostrar la ubicación de estacionamientos disponibles en tiempo real.
- Ofrecer rutas optimizadas hacia los espacios seleccionados.

7. Automatización con cámaras potenciadas por IA

Se explorará la implementación de cámaras de vigilancia con capacidades de Inteligencia Artificial para monitorear automáticamente la disponibilidad de espacios de estacionamiento, todo esto en una futura actualización del sistema.

Estas cámaras podrán detectar ocupación en tiempo real, actualizar la base de datos y generar alertas automáticas en caso de incidentes.

8. Herramientas de monitoreo y logging

Para asegurar la estabilidad del sistema, se utilizarán herramientas de Google Cloud como:

- **Cloud Monitoring:** Monitorear métricas de rendimiento, consumo de recursos y tiempos de respuesta.
- **Cloud Logging:** Registrar eventos críticos y errores para facilitar el diagnóstico y solución de problemas.

9. Framework Spring Boot: Framework de JAVA que permite la creación de aplicaciones web y Back-Ends [7]

El desarrollo del Back-End se realizará utilizando **Spring Boot**, que ofrece una estructura robusta y eficiente para crear APIs REST. Este framework destaca por:

- Simplificar la integración con MySQL y herramientas de seguridad.
- Proveer capacidades integradas para pruebas, manejo de excepciones y optimización del rendimiento.

3 Diseño e Implementación

Este capítulo detalla el proceso de diseño y desarrollo del sistema propuesto para la gestión de estacionamientos. Se describe la arquitectura en capas utilizada, los componentes clave y las funcionalidades implementadas en cada módulo de Back-End.

3.1 Diseño de Componentes

A continuación, se detallan los principales aspectos técnicos que estructuran la aplicación, incluyendo la arquitectura, patrones de diseño, la integración de la solución propuesta y enfoques de codificación como lo es el testing

3.1.1 Arquitectura y estructura de los componentes.

Arquitectura

Se optó por una arquitectura en capas debido a su capacidad para separar las responsabilidades del sistema, facilitando el mantenimiento, la escalabilidad y el desarrollo paralelo. La solución consta de tres capas principales:

- **Capa de presentación (Front-End):** Implementada en Flutter, esta capa incluye:
 - Una aplicación móvil para conductores y administradores.
 - Integración con Google Maps API para mostrar ubicaciones de estacionamientos en tiempo real y proporcionar rutas de navegación.
- **Capa lógica (Back-End):** Desarrollada con Spring Boot y alojada en Google App Engine, esta capa es responsable de:
 - Procesar la lógica empresarial para la gestión de usuarios, estacionamientos y pagos.
 - Exponer APIs REST para la comunicación entre el Front-End y el Back-End.
 - Autenticación y autorización de usuarios mediante JWT (JSON Web Tokens).
- **Capa de persistencia de datos:** Utilizando MySQL en Google Cloud SQL, esta capa maneja:
 - Información estructurada sobre usuarios, estacionamientos, reservas y pagos.
 - Garantía de integridad y consistencia de datos mediante relaciones bien definidas entre las tablas.

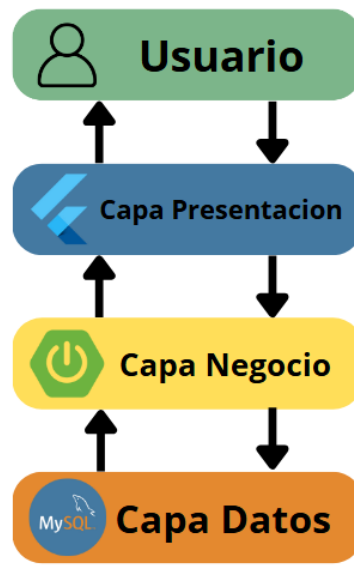


Figura 4. Diagrama Visual resumido de Arquitectura

Este diagrama representa el funcionamiento de la arquitectura utilizada para la aplicación de la solución propuesta, donde el usuario genera las solicitudes en la capa presentación (Front-End) y la capa negocio (Back-End) se encarga de hacer toda la lógica que hay detrás para pedirle los datos a la capa datos.

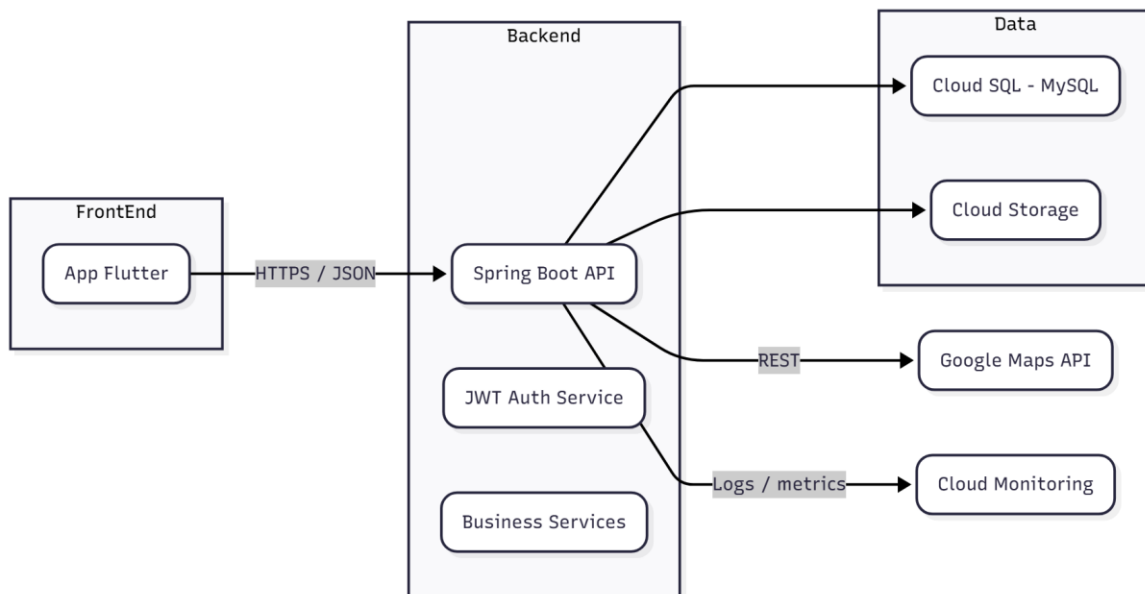


Figura 5. Arquitectura general del sistema.

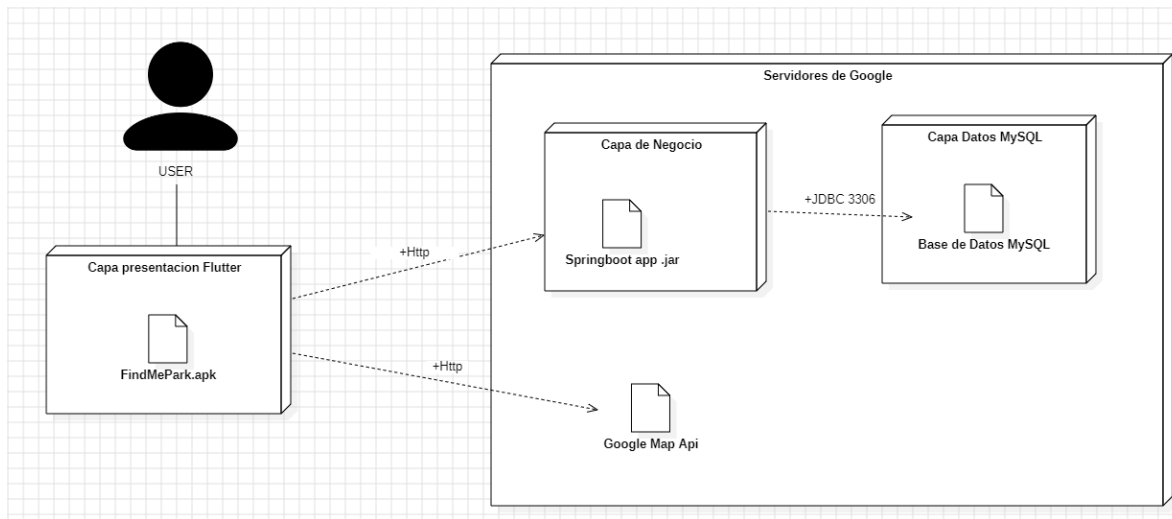


Figura 5.1. Arquitectura del sistema de gestión de estacionamientos.

En la Figura 5 y 5.1 representa el funcionamiento más detallado de la arquitectura utilizada en la aplicación.

El usuario es aquel que interactúa con el Front-End programado en el lenguaje Flutter, esta manda dos solicitudes mediante el protocolo HTTP a los servidores de Google, uno es para la API de Google Maps para mostrar el mapa de la ciudad y el otro al Back-End que es la aplicación de Stringboot en lenguaje Java para comenzar con el funcionamiento general como lo son la autenticación, el despliegue de los estacionamientos guardados en la base de datos, entre otros. El Back-End solicita a la base de datos hecha en MySQL y alojada en el servidor de Google todos los datos necesarios para las funcionalidades anteriormente explicadas.

Estructura de los componentes de software

El sistema está compuesto por los siguientes módulos:

- **Módulo de autenticación y control de acceso:**

Este módulo se dedica a garantizar que solo personas autorizadas puedan entrar a los recursos del sistema.

Para identificar de forma segura a los usuarios tras el inicio de sesión, se vale de un sistema de autenticación basado en tokens JWT. Igualmente, controla el acceso con roles como "conductor" o "administrador", restringiendo o permitiendo funciones según el perfil.

Este módulo es esencial para la seguridad del sistema y evitar accesos sin permiso.

- **Módulo de gestión de usuarios:**

Este módulo unifica todas las funcionalidades atadas al manejo de usuarios.

Facilita ingresar nuevos usuarios, modificar usuarios ya existentes y eliminar algunos usuarios. Los datos se ingresan en Google Cloud SQL (MySQL), y se revisan para que todo esté bien.

Las contraseñas no se almacenan en texto plano ni se cifran, sino que se hashean (proceso que transforma la contraseña en una cadena irreversible de caracteres, imposibilitando recuperar el valor original) mediante BCrypt, una función de derivación lenta con salt (valor aleatorio que se genera y agrega a la contraseña antes de hashearla. Sirve para que dos usuarios con la misma contraseña terminen teniendo hashes distintos) que impide recuperar la contraseña original incluso si la base de datos fuera comprometida

Este módulo es la base para determinar la identidad y cómo se registra cada usuario en el sistema.

- **Módulo de gestión de estacionamientos:**

Este módulo administra todos los aspectos vinculados con los estacionamientos disponibles en la ciudad. Incluyendo el seguimiento en tiempo real de la disponibilidad de cada estacionamiento, la realización de la reserva y el registro de la liberación de espacios tras terminar la estancia. Además, mantiene una sincronización continua con el Front-End para mostrar los cambios de estado de manera instantánea y con la base de datos para mantener la actualización de la información cada vez que se toma o se libera un lugar. Su objetivo principal es maximizar la utilización de los espacios disponibles, permitiendo que los usuarios hallen aparcamiento sin causar atascos innecesarios.

- **Módulo de tarifas y pagos:**

Este módulo se encarga de la lógica detrás de la generación de tarifas, calculando los montos a pagar según el tiempo de uso y el tipo del usuario, ya sea residente o turista. Ha sido diseñado con la visión de integrarse en el futuro con sistemas de pago que permitan a los usuarios realizar transacciones de forma automática, segura y rápida. Esto establece las bases para una funcionalidad esencial: la automatización del cobro, eliminando la necesidad de pagos manuales y ofreciendo una experiencia de usuario más fluida, similar a los sistemas de cobro automático que se encuentran en aplicaciones de entrega o transporte bajo demanda (como Uber o Cabify)

- **Módulo de estadísticas y reportes:**

Este módulo permite a los administradores acceder a la información detallada de la aplicación, pudiendo revisar múltiples aspectos del ámbito de los usos de los estacionamientos, con esta información se pueden generar informes detallados y dashboards, facilitando la toma de decisiones a nivel estratégico y operativo, en caso de planificar alguna mejora en infraestructura

- **Módulo de integración con mapas:**

Este módulo es el encargado de la presentación del mapa en la aplicación y de sus puntos de interés configurados manualmente (los estacionamientos), además es el responsable de la generación de la ruta para guiar al usuario al estacionamiento seleccionado, siendo su motor principal la API de Google Maps.

Estrategia de implementación

- **Desarrollo iterativo:** Cada módulo será desarrollado y validado de forma independiente antes de la integración. Esto permite identificar y resolver problemas en etapas tempranas del desarrollo.
- **Iteraciones del Proyecto:** El proyecto consiste en 3 iteraciones también llamado “Sprints” donde se separó de la siguiente forma:
 - **Sprint 1 (11 sep – 11 oct):**
 - **Objetivo:** Construcción de Front End (Flutter)
 - **Historias de usuario:** Registro y autenticación de usuarios, mapa de estacionamientos, visualización básica de disponibilidad.
 - **Sprint 2 (12 oct-18 nov):**
 - **Objetivo:** Desarrollo del Dashboard administrativo
 - **Historias de usuario:** Generación de estadísticas, gestión de usuarios y visualización de datos con KPIs.
 - **Sprint 3 (19 nov-15 dic):**
 - **Objetivo:** Integración del Back-End y módulo de fiscalización
 - **Historias de usuario:** Conexión a base de datos, manejo de pagos, vista de fiscalizador y cámaras, finalización del MVP.
- **Uso de APIs REST:** Las APIs REST expuestas por el Back-End manejarán todas las interacciones entre el Front-End y la base de datos, garantizando una comunicación eficiente y modular.
- **Pruebas y validación:**
 - Se realizarán pruebas unitarias para validar la lógica interna de cada módulo.
 - Se ejecutarán pruebas de integración para verificar el correcto funcionamiento de las APIs.
 - Pruebas de carga y estrés asegurarán que el sistema maneje altas demandas concurrentes.
- **Despliegue y entorno:**
 - El Back-End se desplegará en Google App Engine, utilizando su escalabilidad automática para manejar picos de tráfico.
 - La base de datos MySQL estará alojada en Google Cloud SQL, con respaldo automático y configuraciones de alta disponibilidad.
 - El código será gestionado en GitHub, con integración continua (CI) mediante Cloud Build.

3.1.2 Uso de buenas prácticas y patrones de diseño.

A continuación, se detallarán los patrones de diseño utilizados en la creación del aplicativo y buenas prácticas realizadas que ayudarán en múltiples aspectos en la codificación de la aplicación Patrones de diseño.

- **Controller MVC (Model-View-Controller):**

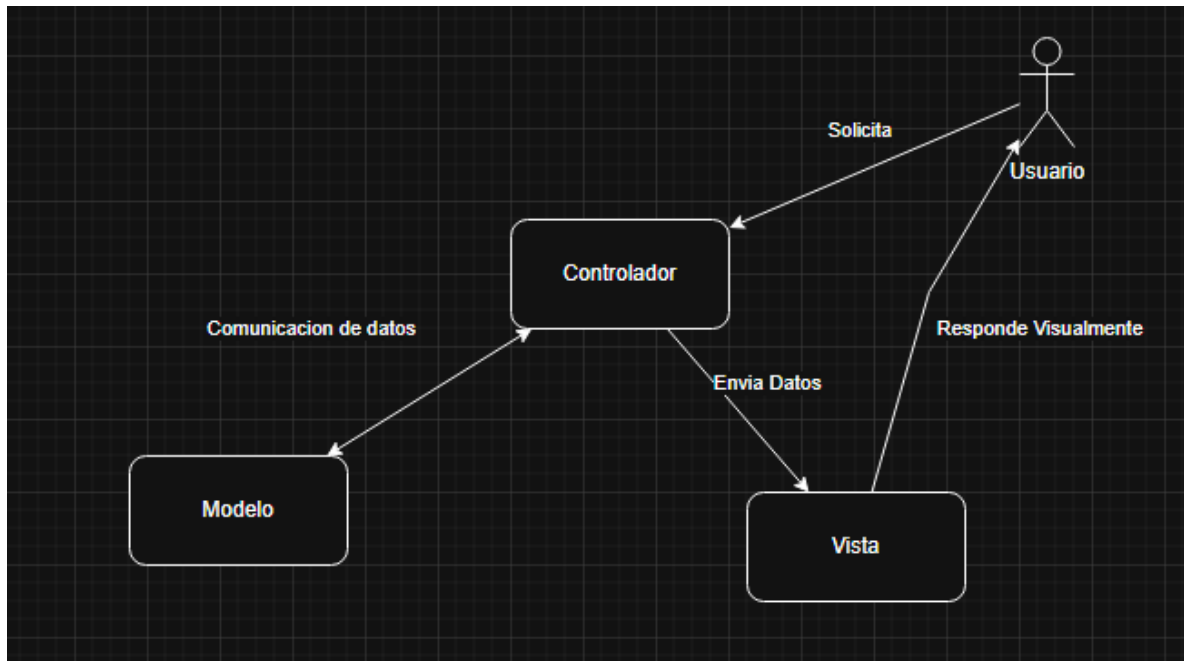


Figura 6. Diagrama de patrón MVC

Descripción: El patrón MVC organiza la aplicación en tres componentes principales: **Modelo**, **Vista**, y **Controlador**. En este caso, el **Controlador** actúa como intermediario entre la solicitud del usuario (desde el Front-End) y los servicios del Back-End que implementan los patrones **Factory** y **Strategy**. [8]

Flujo:

1. El controlador recibe una solicitud HTTP (por ejemplo, para calcular tarifas o consultar disponibilidad).
2. Llama al servicio correspondiente para procesar la lógica empresarial.
3. El servicio utiliza el patrón **Factory** para crear instancias específicas (como diferentes estrategias de cálculo de tarifas según residencia o demanda).
4. El patrón **Strategy** encapsula las diferentes lógicas (por ejemplo, tarifas dinámicas o fijas) y ejecuta la que corresponde.
5. El resultado procesado se envía de regreso al controlador, que lo retorna al Front-End en formato JSON.

Uso: Esto permite una gestión centralizada de las solicitudes, mientras que las implementaciones de lógica empresarial se mantienen flexibles y fáciles de extender.

- **Repository gestionando el acceso a datos**

Descripción: El patrón **Repository** actúa como una capa intermedia entre la lógica empresarial y la base de datos. Simplifica el acceso a datos al encapsular las operaciones CRUD (Crear, Leer, Actualizar, Eliminar).

Flujo:

1. El servicio del Back-End solicita al repositorio realizar una operación específica, como buscar estacionamientos disponibles o registrar un usuario.
2. El repositorio abstrae los detalles de la interacción con la base de datos MySQL, ejecutando consultas predefinidas o procedimientos almacenados.
3. Devuelve los datos procesados al servicio para que este aplique la lógica empresarial necesaria.

Uso: El patrón Repository facilita el mantenimiento y pruebas, ya que las operaciones de datos están centralizadas en un único lugar, reduciendo duplicación y errores.

- **Observer notificando cambios en la disponibilidad de estacionamientos**

Descripción:

El patrón **Observer** permite que múltiples componentes del sistema sean notificados automáticamente cuando ocurre un cambio en el estado de un objeto, como la disponibilidad de un estacionamiento.

En este caso, cuando un estacionamiento cambia de estado, el módulo notifica a los observadores internos del servidor, quienes actualizan la base de datos y dejan el estado listo para que el Front-End lo consulte en su próxima petición REST

Flujo:

1. Los servicios de notificación (por ejemplo, un sistema de alertas en tiempo real) se suscriben al módulo que gestiona la ocupación de estacionamientos.
2. Cuando un cambio ocurre (por ejemplo, un estacionamiento es reservado o liberado), el módulo actualiza su estado y notifica a los observadores.
3. Los observadores ejecutan acciones específicas:
 - Los servicios internos del servidor se suscriben al módulo que gestiona la ocupación de estacionamientos mediante el mecanismo de eventos de Spring.
 - Registrar el evento en la tabla Incidentes si el cambio lo requiere.
 - Dejar el estado actualizado disponible para que el Front-End lo obtenga en su próxima consulta REST.

Uso: Esto asegura que todos los componentes dependientes estén sincronizados en tiempo real, mejorando la experiencia del usuario y la consistencia de los datos.

Buenas prácticas de desarrollo

Las buenas prácticas son aquella forma de trabajar que logra tener orden o facilita modificaciones futuras, al igual que con buenas prácticas reduce la dificultad para leer el código a una persona que lo verá o editará en un futuro.

- **Sistema modular:**

Una de las prácticas esenciales implementadas en el desarrollo del sistema es el modularidad. El sistema está dividido en módulos independientes, cada uno encargado de una responsabilidad específica, como la gestión de usuarios, el manejo de estacionamientos, el cálculo de tarifas y los pagos electrónicos. Esta separación permite desarrollar, probar y mantener cada módulo de manera aislada, reduciendo la complejidad del sistema global. Además, facilita la incorporación de nuevas funcionalidades o modificaciones sin afectar el resto de la aplicación, asegurando un diseño escalable y flexible.

- **Control de versiones con Git:**

Para garantizar un control efectivo del código fuente y un trabajo colaborativo eficiente, se utiliza Git como sistema de control de versiones.

El repositorio del proyecto está alojado en GitHub, donde se sigue una estrategia de ramas bien definida:

- La rama principal (main) contiene la versión estable del sistema.
- La rama de desarrollo (develop) sirve como base para las nuevas funcionalidades y correcciones.
- Cada funcionalidad o corrección se desarrolla en ramas específicas que, una vez completadas y probadas, son fusionadas a develop. Esta estructura asegura un historial detallado de cambios, permite realizar revisiones de código mediante pull requests y facilita la recuperación de versiones anteriores en caso de errores (plan de roll-back). Esta lleva un nombre con el siguiente formato “feature/nombre que describa la funcionalidad”.
- Para cada versión se denominará con números (X.0.0) Se incrementa la “X” cuando se realizan cambios que rompen la compatibilidad con versiones anteriores. (1.X.0) Se incrementa la “X” al agregar nuevas funcionalidades compatibles con versiones anteriores. (1.1.X) Se incrementa la “X” cuando se corrigen errores o se hacen ajustes menores.

- **Pruebas unitarias y de integración:**

El sistema cuenta con un enfoque sólido en la calidad del software mediante la implementación de pruebas unitarias e integraciones desde las primeras etapas del desarrollo.

- Las pruebas unitarias hechas con **JUnit** validan la funcionalidad de componentes individuales, como controladores, servicios y repositorios, asegurando que cada uno funcione correctamente de manera aislada. Estas pruebas se realizan a en la fase de programación así asegurándonos de poder completar con las funcionalidades antes de pasar a otra.

- Las pruebas de integración verifican que los módulos interactúen entre sí de forma correcta, especialmente en flujos críticos como la autenticación de usuarios o la gestión de pagos. Se utilizan herramientas como **Postman** para estas pruebas en la fase de integración, asegurando una alta cobertura del código y reduciendo riesgos de errores en producción.

Separación de responsabilidades (SoC):

Siguiendo el principio de separación de responsabilidades, cada componente del sistema tiene una función claramente definida:

- Los controladores gestionan las solicitudes entrantes y delegan la lógica empresarial a los servicios.
- Los servicios encapsulan las reglas de negocio y se encargan de coordinar las operaciones entre los controladores y los repositorios.
- Los repositorios interactúan directamente con la base de datos, encapsulando las consultas y actualizaciones de datos.

Esta separación, implementada bajo el patrón MVC (Model-View-Controller), facilita el mantenimiento del sistema, reduce la complejidad y asegura una alta cohesión dentro de cada módulo.

3.1.3 Integración con el sistema general

La conexión entre el Back-End y el resto del sistema, particularmente con el Front-End desarrollado en Flutter, se logra a través de solicitudes HTTP y respuestas estructuradas en formato JSON. Este método de integración asegura una comunicación clara, eficiente y modular, permitiendo que ambos componentes operen de manera independiente mientras comparten la información necesaria.

El Back-End, construido con Spring Boot, expone una serie de APIs REST diseñadas para manejar la lógica de negocio y la interacción con la base de datos MySQL. Estas APIs son consumidas por el Front-End para mostrar información relevante y procesar acciones iniciadas por los usuarios, como la consulta de disponibilidad de estacionamientos o el registro de nuevas reservas.

Método de integración

1. Interacción mediante APIs REST:

- Cada funcionalidad del sistema está vinculada a endpoints específicos que gestionan operaciones clave, como autenticación, gestión de usuarios y manejo de estacionamientos.
- El Back-End responde con datos en formato JSON, que es fácilmente procesado por el Front-End para su representación en la aplicación móvil.

2. Estructura de las solicitudes y respuestas:

- El Front-End realiza peticiones HTTP utilizando métodos como GET (para consultas), POST (para creación), PUT (para actualizaciones) y DELETE (para eliminaciones).

- El Back-End devuelve un objeto JSON que incluye:
 - **Códigos de estado HTTP:** Indicadores estandarizados del éxito o fallo de la solicitud (e.g., 200 OK, 401 Unauthorized, 500 Internal Server Error).
 - **Mensajes descriptivos:** Informan al cliente sobre el resultado de la operación, ya sea exitosa o con errores.
 - **Datos procesados:** Información relevante como registros de usuario, detalles de estacionamientos o resultados de operaciones solicitadas.

3. Seguridad en las comunicaciones:

Todo el intercambio de datos se realiza sobre HTTPS, garantizando que las solicitudes y respuestas estén protegidas contra interceptaciones no autorizadas.

Documentación de las llamadas esperadas

En este apartado se mostrarán los endpoints (URL específica que permite a una aplicación acceder a recursos o funciones específicas) utilizados en el sistema, además del método que se utiliza y el parámetro que necesita para funcionar.

Descripción	Endpoint	Método	Parámetro
Iniciar sesión	/api/auth/login	POST	Email Contraseña
Registrar nuevo usuario	/api/auth/register	POST	Email Contraseña Nombres Apellidos Cedula/Pasaporte
Mapeo de estacionamientos	/api/estacionamientos/all	GET	
Descuento y aumento de Cupos de estacionamientos	/api/estacionamientos/ActualizarCupo	POST	Id Estacionamiento Texto (Restar o Aumentar)

Figura 7. Tabla de endpoints

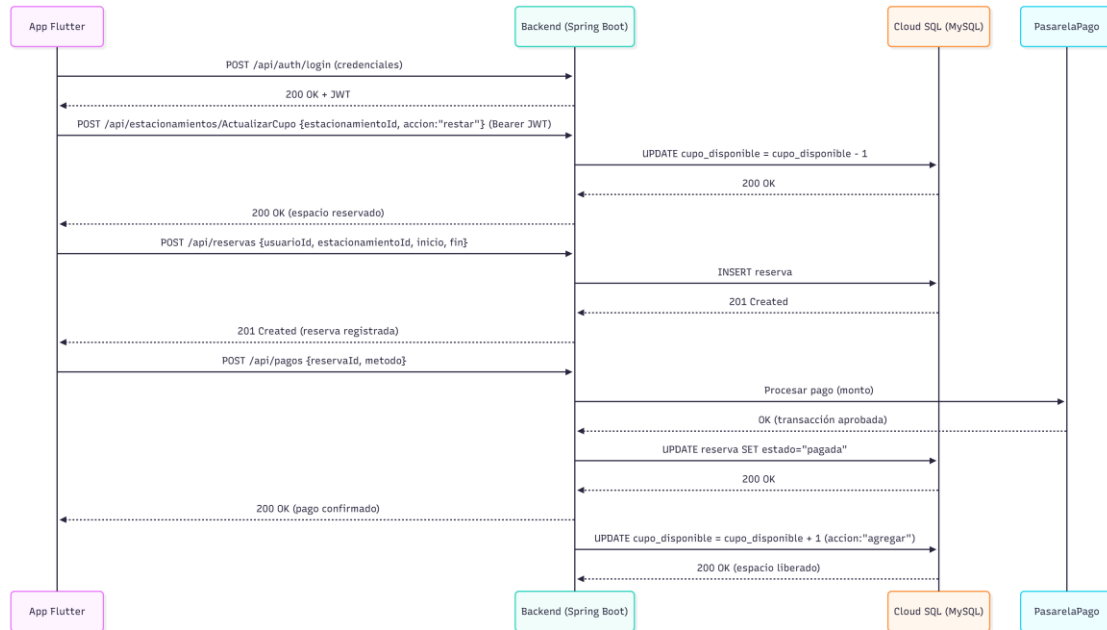


Figura 8. Diagrama de secuencia del flujo de reserva y pago.

La Figura 5 muestra el flujo de comunicación detallado entre la aplicación móvil, el Back-End y la base de datos cada vez que el usuario reserva y paga un espacio de estacionamiento.

Una vez que el usuario realiza la autenticación exitosamente a través de la respuesta ‘200 OK’, la solicitud más Sigue a: “/api/estacionamientos/ActualizarCupo” con el parámetro acción = “restar” para restar el número de cupos disponibles de la base de datos.

Luego incluya la solicitud de una reserva ‘201 Created’ y la realización de un pago a través de la pasarela correspondiente, el sistema luego llama al mismo punto final con acción = “agregar” para liberar el espacio.

Por lo tanto, el flujo puede garantizar la consistencia con respecto a la asignación de estos espacios porque evita sobreasignación y mantiene el estado sincronizado Entre el Back-End y la base de datos.

3.2 Detalles de la codificación y desarrollo.

Metodología de Desarrollo

Para cualquier proyecto en desarrollo se trabaja con diferentes metodologías, en este trabajo se eligió una metodología ágil llamada SCRUM, esta metodología se basa en el uso de roles donde el equipo se divide en cada uno de estos roles para agilizar y delegar tareas específicas a los roles específicos.

Esta metodología trabaja con entregas de corto plazo denominadas “SPRINTS” donde el equipo de programadores trabaja en conjunto para poder completar las historias de usuario.

Cada Sprint se fue determinando por objetivos a cumplir teniendo reuniones habituales para priorizar funcionalidades o para resolver algún problema que surgió en el momento.



Backlog del proyecto

A continuación, veremos cada requerimiento que asignamos a los 3 Sprints realizados:

Sprint 1

- El sistema debe tener cobro diferenciado para los distintos estacionamientos.
- El sistema debe ser capaz de diferenciar entre los usuarios residentes de Viña del mar y los turistas.
- El sistema debe hacer cobro diferenciado entre los usuarios residentes y los turistas.
- El sistema debe de ser capaz de hacer cobro diferenciado dependiendo el tiempo de uso de los estacionamientos.
- El usuario debe ser capaz de registrarse en la aplicación.
- El usuario debe ser capaz de ver un mapa indicando los estacionamientos disponibles.

Sprint 2

- El usuario debe poder registrar el espacio de estacionamiento que esté utilizando y dejarlo en estado de “ocupado”.
- El administrador debe ser capaz de ver los KPI de forma estadística y/o gráfica.
- El usuario debe ser capaz de ver un mapa indicando los estacionamientos disponibles.

Sprint 3

- El fiscalizador debe ser capaz de registrar estacionamientos para otros usuarios.
- El sistema debe estar vinculado a cámaras disponibles para mayor seguridad, si el caso lo amerita.
- El fiscalizador debe tener la capacidad de monitorear las cámaras, si es posible y si el caso lo requiere.
- El fiscalizador debe ser capaz de registrar errores de la aplicación u externos.
- El fiscalizador debe ser capaz de poder ingresar pagos por estacionamientos en efectivo por la aplicación.
- El usuario debe poder pagar el tiempo utilizado el estacionamiento registrado a través de la aplicación.

Casos de Uso

A continuación, se mostrará una figura que demuestra los requisitos funcionales del sistema especificando que tipo de usuario debe tener cada requisito funcional.

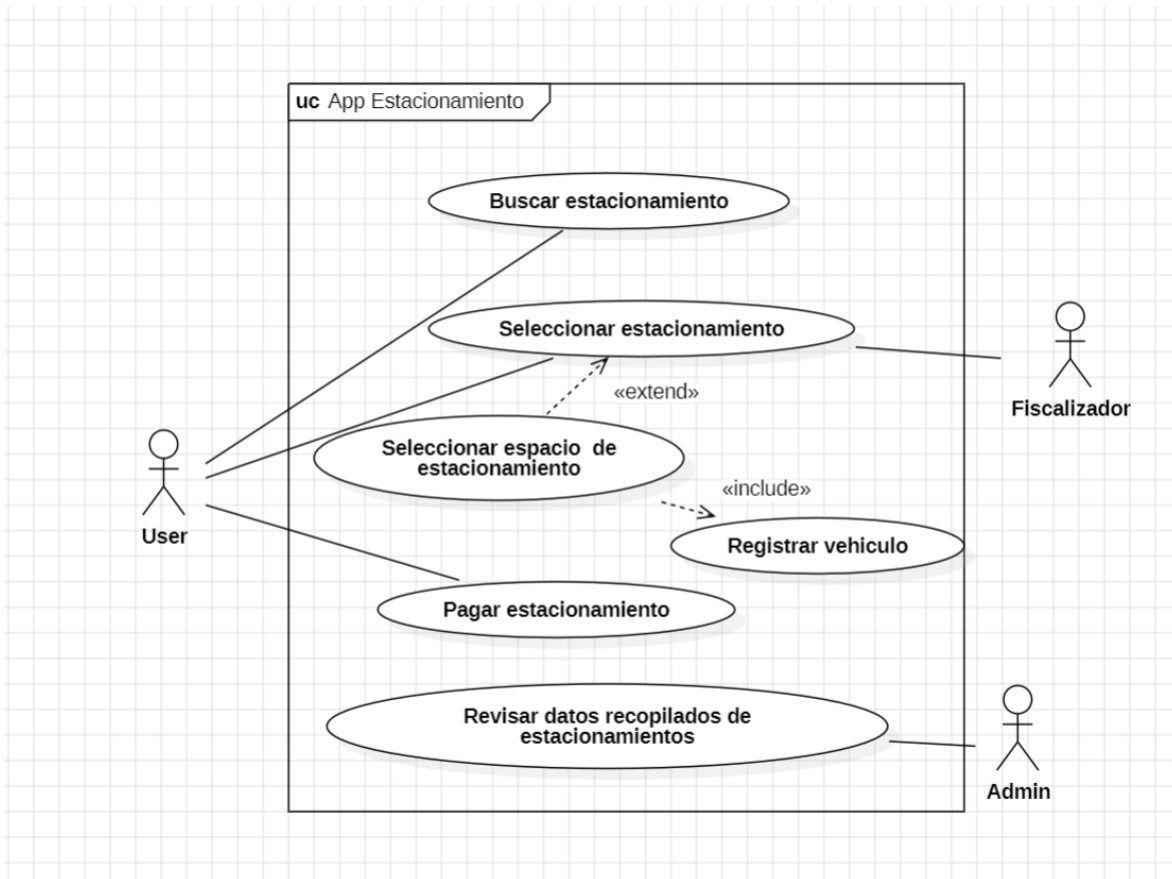


Figura 9. Casos de uso principales: Usuario, Fiscalizador y Administrador.

User: Este es el Usuario común, aquel que descarga su aplicación móvil y la usa en la ciudad para poder estacionarse

Fiscalizador: Este tipo de actor es aquel que verifica en los estacionamientos que la información no este errada y también apoyara con el reservado de estacionamientos en casos situacionales como lo sería que el usuario no pueda completar por sí solo las reservas por múltiples motivos vinculados a la telefonía, conectividad o en caso de que el usuario es un turista

Administrador: Este tipo de usuario es principalmente aquellos que mantienen los estacionamientos, ya sean municipalidades o los dueños de estacionamientos privados

Nombre	Actor	Descripción
Buscar estacionamiento	User	El usuario puede ingresar a la aplicación y poder buscar un estacionamiento cercano a una dirección ingresada
Seleccionar estacionamiento	User / Fiscalizador	El usuario debe ser capaz de seleccionar el estacionamiento además de

		revisar si tiene cupo disponible
Seleccionar espacio de estacionamiento	User	Siendo una extensión del caso de uso anterior, este permitirá seleccionar el estacionamiento y proceder con la reserva de un cupo
Revisar datos recopilados de estacionamientos	Admin	Como una inclusión de la anterior en caso de ser primera vez que reserva un cupo de estacionamiento el usuario deberá registrar su vehículo en base a su patente
Seleccionar estacionamiento	Fiscalizador	En este caso el Fiscalizador podrá seleccionar un estacionamiento independiente este ocupado, mostrando la información, si es que este está en uso podrá corroborar la información puesta, en caso de que la información este errada podrá colocar multas, a su vez podrá registrar nuevos usuarios para casos situacionales
Pagar estacionamiento	User	El usuario podrá pagar su estadía en el estacionamiento, generando el cálculo por el tiempo que estuvo en el aparcamiento
Revisar datos recopilados de estacionamientos	Administrador	El Administrador podrá ver informaciones relevantes para la toma de decisiones de los estacionamientos, en estos se encuentra que estacionamientos tienen mayor uso, entre otros

Figura 10. Tabla descriptiva de Casos de Uso

Diagrama de Entidad Relación

A continuación, se presenta el diagrama entidad-relación del sistema, donde se puede ver cómo quedó estructurada la base de datos para responder a las necesidades de la aplicación.

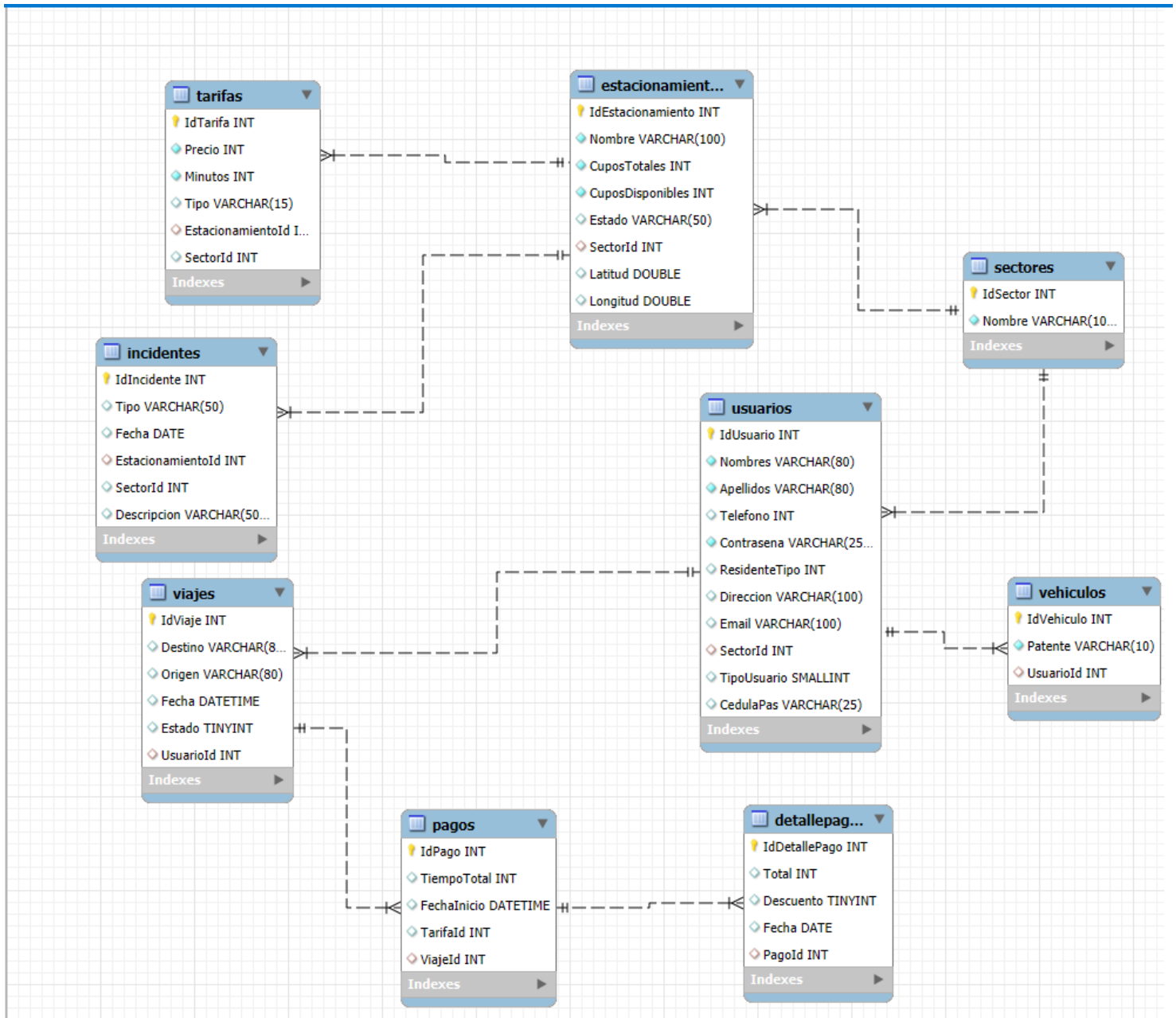


Figura 11. Diagrama de Entidad-Relación

Estacionamientos es la tabla desde la que parten la mayoría de las relaciones, ya que cada recinto pertenece a un sector, maneja sus propias tarifas según el tipo de usuario y registra los incidentes que ocurran en él. Del lado de los usuarios, cada uno puede tener varios vehículos asociados y realizar múltiples viajes; cada viaje genera un pago y ese pago queda detallado en "DetallePago" junto al monto final y los descuentos que correspondan. El diseño se trabajó evitando datos repetidos entre tablas, usando claves foráneas para mantener la consistencia de la información en todo momento.

Mecanismos de CI/CD

Se implementaron mecanismos de Integración Continua (CI) y Despliegue Continuo (CD) con un enfoque en la automatización de pruebas y la preparación para despliegues eficientes y seguros en el entorno de Google Cloud. Estas prácticas aseguran que los cambios en el código sean validados antes de ser desplegados, minimizando errores y garantizando la estabilidad del sistema en producción.

1. Pruebas automatizadas en Google Cloud Build:

- Se configuró un pipeline en **Google Cloud Build** para automatizar la ejecución de pruebas y validaciones del Back-End cada vez que se actualiza el código.
- Este pipeline incluye pasos como:
 - **Compilación del código:** Verificar que el proyecto se compile correctamente utilizando las herramientas de construcción configuradas (Maven).
 - **Pruebas automatizadas de APIs:** Validar las funcionalidades clave del sistema mediante pruebas diseñadas en Postman, asegurando que las APIs expuestas funcionen correctamente antes de ser desplegadas.

2. Despliegue en Google App Engine:

- El despliegue se realiza de forma automatizada en Google App Engine, aprovechando su capacidad de manejar la escalabilidad y la alta disponibilidad de manera nativa.
- Pasos del despliegue:
 - El código empaquetado se sube a Google Cloud y se implementa directamente en App Engine.
 - Cada despliegue genera una nueva versión que puede ser administrada y revertida fácilmente en caso de fallos, permitiendo mantener el entorno de producción estable.

3.3 Pruebas y Validación:

Para garantizar que el Back-End cumple con los requerimientos funcionales y no funcionales, se implementó un enfoque integral de pruebas y validación. Este proceso incluye la validación de componentes clave, la simulación de flujos completos en entornos controlados y la verificación de resultados en condiciones reales.

Automatización de pruebas con Google Cloud Build:

- Se configuraron pipelines en Google Cloud Build para automatizar la ejecución de pruebas y validaciones en el Back-End, lo que permitió detectar errores de manera temprana durante el desarrollo.

- Estas pruebas automatizadas incluyen:
 - **Pruebas de integración con Postman:** Validaron las APIs expuestas por el Back-End asegurando que las respuestas sean correctas y cumplan con los estándares definidos.
 - **Pruebas de compilación:** Verificaron que el proyecto pueda compilarse sin errores, garantizando que el código cumpla con los requisitos técnicos básicos.

Pruebas manuales:

Además de la automatización, se realizaron pruebas manuales para complementar el proceso y garantizar que las funcionalidades clave operaran correctamente en escenarios específicos.

- **Pruebas funcionales específicas:**
 - Se probaron casos de uso particulares, como la autenticación de usuarios y la consulta de estacionamientos disponibles, simulando el comportamiento esperado del sistema en condiciones controladas.
- **Pruebas en el entorno real:**
 - Una vez verificado el correcto funcionamiento de los módulos en un entorno de pruebas, se realizaron pruebas funcionales directamente en el entorno de producción, validando que las APIs respondieran correctamente bajo las condiciones reales de uso.

Validación de resultados:

El proceso de validación de resultados incluyó:

- **Comparación con criterios de aceptación:**
 - Se evaluaron los resultados obtenidos frente a los requisitos establecidos, asegurando que el sistema cumpliera con los objetivos definidos en términos de funcionalidad y rendimiento.
- **Documentación de errores y casos exitosos:**
 - Cada prueba fue documentada, registrando tanto las respuestas exitosas como los errores detectados, lo que permitió un análisis detallado y la corrección oportuna de los problemas encontrados.

3.3.1 Estrategias de testing aplicadas a los componentes.

1. Testing de APIs REST

Las APIs REST, como principal punto de interacción entre el Front-End y el Back-End, fueron validadas exhaustivamente mediante las siguientes estrategias:

- **Pruebas funcionales:**
 - Se probaron los endpoints de las APIs utilizando **Postman** para verificar que las respuestas fueran correctas y en el formato esperado (JSON).
 - Cada endpoint fue probado con datos válidos e inválidos para garantizar un manejo adecuado de errores y validaciones.
- **Pruebas de flujo completo:**
 - Se simularon escenarios típicos, como el inicio de sesión y la consulta de estacionamientos disponibles, para validar que los flujos completos funcionaran correctamente desde la solicitud hasta la respuesta.

2. Testing de integración

Se implementaron pruebas de integración para garantizar que los diferentes módulos del Back-End trabajaran correctamente juntos.

- **Pruebas de interacción entre servicios:**
 - Se verificó que los servicios de lógica empresarial (como el cálculo de tarifas o la gestión de usuarios) interactuaran correctamente con los repositorios encargados del acceso a la base de datos.
 - Se probaron flujos que involucraban múltiples módulos, como la creación de una reserva que impacta tanto en la disponibilidad del estacionamiento como en el historial del usuario.
- **Validación con la base de datos:**
 - Se realizaron pruebas para garantizar que las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) se ejecutaran correctamente en la base de datos MySQL, asegurando la consistencia de los datos.

3. Testing manual en el entorno real

Una vez verificado el correcto funcionamiento de los módulos en el entorno de pruebas, se realizaron pruebas manuales en condiciones reales para validar la funcionalidad en producción.

- **Validación de casos de uso clave:**
 - Se probaron funcionalidades críticas, como la autenticación y la consulta de estacionamientos disponibles, con datos reales y escenarios de uso simulados.

- **Monitoreo de errores:**

- Durante estas pruebas, se registraron los errores detectados en logs mediante herramientas de Google Cloud, lo que permitió su análisis y corrección.

3.3.2 Resolución de bugs y optimización

Durante el desarrollo del sistema, se resolvieron bugs conforme surgían al implementar nuevas funcionalidades. La arquitectura modular del Back-End facilitó el proceso, permitiendo aislar los problemas en componentes específicos y corregirlos sin afectar el funcionamiento del sistema completo.

Este enfoque permitió avanzar de manera constante, garantizando que el sistema llegara a la feria de software como un prototipo bien definido, funcional y alineado con los objetivos iniciales del proyecto.

3.4 Integración con Otros Componentes

Sistema de pagos:

Se planea implementar un sistema de pagos que permita a los usuarios realizar transacciones de manera sencilla, segura y automatizada. Este sistema incluirá:

- **Pago automático al finalizar la estancia:** El sistema calculará automáticamente la tarifa en función del tiempo de uso del estacionamiento y la aplicará al método de pago registrado por el usuario.
- **Opciones de pago diversificadas:** Los usuarios podrán utilizar tarjetas de crédito, débito o billeteras electrónicas para realizar sus pagos.
- **Notificaciones en tiempo real:** Una vez completado el pago, se enviará una notificación al usuario con un comprobante de la transacción, proporcionando transparencia y confianza.

Este componente facilitará la experiencia de los usuarios al eliminar la necesidad de pagos manuales, además de agilizar el flujo de salida en los estacionamientos.

Cámaras con inteligencia artificial (IA):

Otra mejora planificada es la integración de cámaras inteligentes equipadas con algoritmos de IA para optimizar la gestión de los estacionamientos. Estas cámaras podrán:

- **Detectar automáticamente la ocupación de espacios:** Identificar en tiempo real si un estacionamiento está ocupado o libre, actualizando el sistema sin intervención humana.
- **Validar el uso adecuado de los espacios:** Comprobar que los vehículos estén correctamente estacionados, generando alertas en caso de obstrucciones o mal uso.
- **Reconocimiento de placas:** Facilitar el registro de vehículos ingresados, mejorando la seguridad y vinculando automáticamente la información con el sistema de pagos.

El uso de cámaras con IA permitirá reducir errores humanos, optimizar la precisión de la información sobre la disponibilidad de espacios y mejorar la seguridad en el sistema.

4 Conclusiones.

El desarrollo de este proyecto ha sido una experiencia enriquecedora que ha permitido aprender valiosas lecciones sobre la implementación de sistemas complejos. A lo largo del proceso, se tomaron decisiones clave que hoy en día se consideran importantes, pero también se identificaron áreas donde podríamos haber actuado de manera diferente.

Una de las decisiones que, retrospectivamente, podría haberse abordado de otra manera es la implementación de las pruebas automatizadas. Si bien utilizamos Postman para validar las APIs y garantizar la correcta comunicación entre el Front-End y el Back-End, la ausencia de pruebas unitarias automatizadas, como las que se realizarían con JUnit, hizo que algunas validaciones de componentes individuales se realizaran de manera más manual y tardada. En futuros proyectos, la incorporación de pruebas unitarias sería una prioridad, pues facilita la detección temprana de errores y mejora la calidad del código.

Uno de los mayores desafíos fue integrar múltiples tecnologías y asegurarnos de que los distintos módulos, como el Back-End en Spring Boot, la base de datos en MySQL y el despliegue en Google App Engine, funcionaran de manera conjunta y sin problemas. Sin embargo, este reto se transformó en un logro importante, pues el sistema fue diseñado de manera modular y escalable, lo que lo hace fácilmente adaptable a futuras necesidades.

El proyecto también contribuye al bienestar social al mejorar la gestión de los estacionamientos, contribuyendo a la reducción de la congestión vehicular y la huella de carbono, especialmente en áreas urbanas como Viña del Mar. Además, al integrar tecnologías avanzadas como la autenticación de usuarios y la conexión con sistemas en la nube, el proyecto tiene un impacto directo en la transformación digital de los servicios municipales, que pueden servir como referencia para futuras soluciones urbanas inteligentes.

A futuro, el proyecto podría beneficiarse de la integración de sistemas de cámaras con inteligencia artificial para verificar la ocupación de los espacios de estacionamiento en tiempo real, así como de un sistema de pagos automatizado que facilite la experiencia del usuario.

Este proyecto ha sido una excelente oportunidad para aplicar conocimientos de tecnologías modernas, como la computación en la nube, APIs REST y herramientas de desarrollo ágil. El impacto en los usuarios es claro, pues mejora su experiencia al hacer más eficiente el uso del espacio y reduce el tiempo que dedican a encontrar un lugar para estacionar. Para el equipo de desarrollo, este proyecto ha sido una lección de cómo abordar problemas reales utilizando soluciones tecnológicas innovadoras.



5 Agradecimientos

Quiero dar mis principales agradecimientos a mi padrastro Ciro Antonio Keim Bastias, a mis Hermanos Fabian Patricio Ramirez Ibarra y Felipe Marcelo Ramirez Ibarra que me apoyaron en todo este gran proceso al igual a que mi abuelo Eduardo Ibarra Covarrubias y a mi difunta y querida abuela Juana Clorinda Leiva Espejo, a mis amigos con los que viví incontables días durante esta etapa de mi vida, a mi difunto amigo y colega Martin Valenzuela que ya no puede estar a mi lado para celebrar el fin de esta era. Y por último pero la más importante a mi madre María Eugenia Ibarra Leiva, mi heroína, la mujer que me dio la vida, las fuerzas y la motivación para terminar y cumplir este logro, que me enseñó que no importa lo dura que se ponga la vida siempre hay que mirar hacia en enfrente y seguir, más que un agradecimiento quiero que sea una dedicatoria porque sin ella nada hubiera sido como lo es ahora.

6 Bibliografía

- [1] P. Asencio, «Scrum para la gestión de proyectos: Agilidad y Eficiencia,» 10 12 2024. [En línea]. Available: <https://konstruedu.com/es/blog/scrum-para-la-gestion-de-proyectos-agilidad-y-eficiencia>.
- [2] M. Richards, Software Architecture Patterns, Sebastopol: O'Reilly Media, 2015.
- [3] N. Sarmin, «Spring Boot Framework: Features, Benefits, and Use Cases,» 10 Diciembre 2024. [En línea]. Available: <https://staticmania.com/blog/spring-boot>.
- [4] S. Roger, «Python vs JAVA vs PHP vs PERL vs Ruby vs JavaScript vs C++ vs TCL,» 8 Febrero 2021. [En línea].
- [5] J. Simpson, «Nordic Apis,» 9 Febrero 2022. [En línea]. Available: <https://nordicapis.com/20-impressive-api-economy-statistics/>. [Último acceso: 5 Junio 2025].
- [6] 6sense, «Google App Engine – Market Share, Competitor Insights in Platform-as-a-Service (PaaS),»,» 2026. [En línea]. Available: <https://6sense.com/tech/platform-as-a-service-paas/google-app-engine-market-share>. [Último acceso: 15 3 2026].
- [7] Theirstack, «Companies that use Spring Boot,» 2026. [En línea]. Available: <https://theirstack.com/en/technology/spring-boot>. [Último acceso: 2 2 2026].
- [8] J. A. Pérez, «Cómo implementar el patrón MVC,» Mozilla Developer Network, 15 Noviembre 2023. [En línea]. Available: <https://developer.mozilla.org/es/docs/Glossary/MVC>. [Último acceso: 18 Enero 2025].
- [9] R. S. Pressman y B. R. Maxim, Ingeniería del Software: Un enfoque práctico, 8ª ed. México: McGraw-Hill, 2015.
- [10] Google Cloud, “App Engine Documentation,” Google Developers, 2024. [En línea]. Disponible en: <https://cloud.google.com/appengine/docs>
- [11] Flutter, “Introduction to Flutter,” Flutter.dev, 2024. [En línea]. Disponible en: <https://flutter.dev/docs>
- [12] Oracle, “MySQL 8.0 Reference Manual,” Oracle Corporation, 2024. [En línea]. Disponible en: <https://dev.mysql.com/doc/refman/8.0/en/>
- [13] Spring Boot, “Spring Boot Reference Documentation,” Spring.io, 2024. [En línea]. Disponible en: <https://docs.spring.io/spring-boot/docs/current/reference/html/>
- [14] JWT.io, “Introduction to JSON Web Tokens,” Auth0, 2024. [En línea]. Disponible en: <https://jwt.io/introduction>
- [15] Google Maps Platform, “Maps JavaScript API Documentation,” Google Cloud, 2024. [En línea]. Disponible en: <https://developers.google.com/maps/documentation>
- [16] GitHub, “Understanding the GitFlow Workflow,” GitHub Docs, 2024. [En línea]. Disponible en: <https://docs.github.com/>
- [17] Atlassian, “Continuous Integration Explained,” Atlassian.com, 2024. [En línea]. Disponible en: <https://www.atlassian.com/continuous-delivery/continuous-integration>
- [18] IEEE Computer Society, IEEE Citation Reference, IEEE, 2021. [En línea]. Disponible en: <https://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf>