

DESARROLLO E INTEGRACIÓN DE ARQUITECTURA BACKEND PARA PLATAFORMA DE GESTIÓN DIGITAL DE PLANES DE EMERGENCIA FASTPLAN

Giovanni Alexis Alarcón Riquelme

giovanni.alarcon@usm.cl

Pamela Gatica
Profesor Guía

David Larrondo
Profesor Correferente

Resumen: Actualmente, la gestión de emergencias por parte del Cuerpo de Bomberos de Viña del Mar enfrenta la falta de acceso inmediato a los planes de emergencia, obligándolos a operar bajo un modelo reactivo e improvisado de respuesta. Para abordar esta problemática, se diseñó e implementó la plataforma Fastplan, que busca centralizar la información crítica de los edificios que puedan verse afectados. El objetivo en este trabajo de título es abordar el desarrollo de la arquitectura Backend de la plataforma Fastplan (Node.js, Express, MySQL, Firebase), debiendo cumplir el rol de centralizar, resguardar y servir información crítica de forma segura y en tiempo real a un cliente frontend. Para ello, se muestra cómo la solución se evaluó mediante pruebas automatizadas de caja negra con Jest y Supertest sobre sus componentes clave, utilizando una métrica propia que determina la confiabilidad operativa a partir del porcentaje de éxito de las pruebas. Por último, se muestran los resultados e impacto del sistema, el cual logró superar las pruebas propuestas, comprobando su capacidad de identificar y corregir vulnerabilidades lógicas en la capa servidor para garantizar una entrega de datos rápida y confiable.

Palabras Clave: Servidor Backend, Gestión de datos, Planes de Emergencia, información crítica.



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 E INTEGRACIÓN DE ARQUITECTURA BACKEND PARA PLATAFORMA DE GESTIÓN DIGITAL DE PLANES DE EMERGENCIA FASTPLAN

Nombre del candidato(a): Giovanni Alexis Alarcón Riquelme

Carrera / Grado: Ingeniería en Informática

Campus: Viña del Mar Departamento: Departamento de Electrotecnia e Informática (ELINF)

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Pamela Alexandra Gatica Caballero, 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: 05/08/2026

Firma:

Estudiante o Candidato(a):

Fecha: 04-08-2026

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

El presente capítulo tiene como propósito establecer el marco contextual bajo el cual se desarrolló este trabajo de titulación. A continuación, se exponen los fundamentos del proyecto, definiendo la problemática que motivó la investigación y la solución tecnológica propuesta para abordarla. Asimismo, se detallan el objetivo general y los objetivos específicos que guiaron el diseño e implementación del componente Backend para la plataforma Fastplan.

1.1 Problemática

Es sabido que en estos momentos el cuerpo de bomberos de Viña del Mar atiende lo antes posible las emergencias varias que surgen a lo largo de la región, pero existe un problema que radica en que a la hora de llegar a ciertas instalaciones se presentan dificultades en obtener los planes de emergencia junto su información crítica, esto supone un gran problema cuando abordan aquellos incidentes, pues de normal no llegan a contar con los planes correspondientes, pues muchas veces dichos planes se encuentran dentro de aquellos edificios o establecimientos donde están ocurriendo las catástrofes, dificultando su obtención.

Con esto en claro, los bomberos requieren de un sistema capaz de disponer de estos planes de emergencia, donde se pueda acceder a ellos junto su información crítica a la hora de que una emergencia este ocurriendo.

1.2 Solución propuesta

En base al problema anterior se propone como solución el desarrollo de una plataforma web y móvil enfocada en digitalizar, centralizar y facilitar el acceso a los planes de emergencia de los edificios y/o recintos. El sistema está pensado en entregar información crítica para bomberos como lo pueden ser “salidas de emergencias”, “puntos de red húmeda”, “materiales peligrosos”, “accesos especiales”, etc. Esta información debe ser entregada de manera resumida e intuitiva para apoyar la labor de bomberos de Viña del Mar durante las emergencias. En sí, el proyecto busca mejorar la coordinación en emergencias, optimizar la toma de decisiones en terreno y fomentar la preparación ciudadana mediante el uso de tecnología con datos centralizados.

1.3 Aporte del Backend a la solución

El trabajo desarrollado en torno al Back-end constituye la base lógica y funcional de la plataforma Fastplan. Este componente es responsable de la gestión de datos críticos, la integración con la base de datos y la exposición de servicios mediante API que permiten la comunicación entre el Front-end web, la aplicación móvil y otros posibles módulos futuros. En particular, el Back-end asegura la persistencia, disponibilidad y seguridad de la información de planes de emergencia, de manera que el resto de la solución pueda ofrecer acceso confiable y en tiempo real a los usuarios. De esta forma, el Back-end bajo este contexto, no solo articula la conexión entre la información y los clientes de la plataforma, sino que también garantiza extensibilidad, integridad y consistencia de los datos que se utilizarán durante estas situaciones críticas.

1.4 Objetivo general del trabajo

Desarrollar la arquitectura de Backend web para la plataforma Fastplan, asegurando la integración con los demás componentes del sistema, con el fin de garantizar la disponibilidad, seguridad y consistencia de la información de planes de emergencia en entornos críticos.

1.5 Objetivos específicos

- **Diseñar la arquitectura del backend** definiendo sus módulos principales, modelo de datos y puntos de integración con el frontend y la aplicación móvil.
- **Implementar servicios RESTful** que permitan la gestión centralizada de información crítica (planes, mapas, salidas de emergencia, grifos, materiales peligrosos, accesos especiales, etc.).
- **Integrar mecanismos de seguridad** (autenticación, autorización y cifrado de datos sensibles) para resguardar la información.
- **Realizar pruebas unitarias e integrales** que validen el correcto funcionamiento de los endpoints y la estabilidad de la aplicación.
- **Documentar la API y los procesos** de desarrollo para facilitar el mantenimiento y escalabilidad del sistema en el futuro.

1.6 Metodología y plan de trabajo

La gestión del proyecto se fundamentó en la metodología ágil Scrum, estructurando el ciclo de desarrollo en tres iteraciones principales (*sprints*). La planificación operativa se realizó mediante la asignación de Historias de Usuario, cuyo avance se monitoreó a través de reuniones de coordinación con una frecuencia bi-semanal.

Para el control de versiones e integración continua, se utilizó la plataforma GitHub. El equipo implementó un flujo de trabajo basado en ramas (*Feature Branch Workflow*), donde cada nueva funcionalidad es desarrollada y validada en un entorno aislado antes de su fusión (*merge*) con la rama principal (main). Esta estrategia aseguró la integridad del código y evitó regresiones funcionales durante las etapas de integración.

Pasos:

- Designación o repartición de tareas: Se toma en cuenta el rol del integrante dentro del equipo como la carga que se espera que suponga la misma para su designación. La designación debe ser equitativa y no sobrecargar a los miembros, significando que puede haber tareas designadas a más de un integrante
- Proceso de desarrollo: Cada integrante debe desarrollar sus tareas designadas ocupando como base la rama principal del proyecto hasta dicho momento. Cuando un integrante quiere subir una etapa intermedia para compartirla con el equipo, debe hacerlo en una rama única destinada para dicha tarea.
- Reuniones: Las reuniones deben ser al menos 2 por semana, donde se espera que en al menos 15 minutos se discutan tanto los avances como decisiones a tomar con respecto al proyecto. Las reuniones ocupan de idea base las Dailys que se desarrollan a lo largo de las metodologías Scrum reales.
- Entrega: Cuando vence el tiempo de un sprint, se espera una junta de los integrantes del equipo para verificar el avance total de la iteración, donde se espera que se terminen de integrar en el repositorio principal aquellas tareas que no han sido integradas hasta ese momento, esta entrega se hace un día antes al día del vencimiento.

2 Marco Conceptual

El presente capítulo tiene por objetivo asentar las bases teóricas y técnicas que sustentan el desarrollo de la plataforma Fastplan. A lo largo del texto, se abordarán los conceptos fundamentales relacionados con la gestión digital de información crítica, incluyendo la arquitectura de software Backend, la implementación de servicios web y los estándares necesarios para el desarrollo de sistemas escalables y seguros.

Asimismo, se articulará este marco tecnológico con el contexto operativo real, analizando la normativa de planes de emergencia y los requerimientos específicos de la labor del Cuerpo de Bomberos de Viña del Mar.

2.1 Planes de Emergencia

Los planes de emergencia corresponden a documentos formales obligatorios para los distintos edificios y establecimientos, estos documentos tienen como propósito el establecer procedimientos, medidas preventivas, rutas de evacuación y estrategias de respuesta ante situaciones que puedan comprometer la seguridad de las personas dentro de las instalaciones. En el contexto del marco regulatorio de Chile, los planes de emergencia son exigidos por la ley n°21.442 (artículo 40) para edificios y condominios, la ley 16.744 para los entornos laborales y la ley n°21.364 para comunas, provincias o regiones, a parte de estas leyes, se establece que todo edificio debe de cumplir con distintas normativas específicas, por ejemplo el decreto supremo n°594 del Ministerio de Salud que norman deben cumplir las estructuras de construcción, otro ejemplo puede ser la normativa de la Superintendencia de Electricidad y Combustibles (SEC), que regulan aspectos de seguridad industrial y gestión de riesgos asociados a instalaciones energéticas.

En términos generales, un plan de emergencia integra información clave, entre la cual se pueden destacar:

- Rutas y vías de evacuación.
- Sistemas de extinción, tales como red húmeda, red seca, extintores y grifos exteriores.
- Zonas seguras, tanto internas como externas.
- Inventario de materiales peligrosos o sustancias inflamables.
- Puntos críticos de acceso, como barreras, accesos vehiculares o puertas especiales.
- Normas de actuación para brigadas internas y primeros respondedores.

En la práctica, estos elementos no solo conforman una guía de acción para las personas que habitan dichos edificios, sino que los organismos de primera respuesta como Bomberos también dependen de esta información que podemos considerar crítica para planificar de manera rápida y precisa las acciones que se tomarán en terreno. Como bien se nombró en la problemática, la disponibilidad de los planes es contraproducente a su objetivo, ya que, durante la ocurrencia de situaciones críticas, estos suelen almacenarse dentro del mismo edificio afectado por la emergencia y en formatos difíciles para una lectura rápida dadas las circunstancias.

En el contexto del proyecto, es esencial delimitar los puntos que abarca un plan de emergencia, pues el sistema busca disponer de la información crítica de estos para servir la misma al cuerpo de Bomberos.

2.2 Elementos Críticos

Si bien el término "Elemento Crítico" no posee una definición en el diccionario general, se construye conceptualmente a partir de las áreas como la Gestión de Riesgos y la Protección Civil. En estas áreas, se refieren a conceptos como "Infraestructura Crítica" o "Puntos Críticos", donde se destaca que todos identifican activos cuya gestión y conocimiento es prioritario para la seguridad.

Para efectos de este proyecto, se establecerá que lo "crítico" refiere a una *"condición inherente a elementos de alta prioridad estratégica, cuya falla, indisponibilidad o desconocimiento genera un impacto negativo irreversible en la toma de decisiones o en la seguridad operativa"*.

Bajo esta premisa, un Elemento Crítico en el ecosistema Fastplan se define como cualquier activo de información que, de no estar disponible o ser inexacto durante un siniestro, impide la actuación eficaz y segura de los equipos de respuesta.

Importancia en la Respuesta Operativa Dentro de la estructura de un Plan de Emergencia, la correcta identificación y geolocalización de estos elementos es determinante. Se trata de puntos, equipos, materiales o condiciones que actúan como modificadores del riesgo: tienen la capacidad de agravar una emergencia o, por el contrario, facilitar su mitigación. Su reconocimiento anticipado permite transformar la incertidumbre del escenario en una planificación táctica precisa.

Los elementos críticos pueden clasificarse como:

- Elementos estructurales sensibles: Muros portantes debilitados, áreas con riesgo de colapso, zonas de concentración de carga, etc.
- Instalaciones tecnológicas esenciales: Cámaras eléctricas, transformadores, tableros generales, salas de bombas, cuartos de calderas, etc.
- Sistemas de seguridad: Redes húmedas, gabinetes de extinción, rociadores, detectores de humo, paneles centrales contra incendios, etc.
- Fuentes de riesgo químico: Bodegas con materiales inflamables, cilindros de gas comprimido, sustancias corrosivas, etc.
- Puntos estratégicos para la operación de Bomberos: Accesos restringidos, patios internos, techos frágiles, fosos de ascensores, obstrucciones internas, etc.
- Sectores donde se concentran personas vulnerables: Guarderías, salas de pacientes, laboratorios, talleres, zonas de encierro involuntario, etc.

En Retrospectiva, la gestión de esta información en el Backend de Fastplan permite mitigar los daños ante las situaciones de emergencia donde el cuerpo de bomberos podría anticiparse a las condicionantes operacionales del recinto, por ende, es indispensable facilitar la entrega de esta información para una lectura rápida de la misma.

2.3 Backend en el Desarrollo Web

El Backend, se le conoce también como el desarrollo del lado del servidor, constituyendo la capa de la arquitectura de software responsable de la lógica de negocio, la persistencia de datos y la seguridad operativa del sistema. A diferencia del Frontend, esta infraestructura opera de manera invisible para el usuario final, encargándose de procesar las solicitudes, gestionar la comunicación con la base de datos y exponer funcionalidades mediante interfaces de programación de aplicaciones (APIs)

En el ecosistema del proyecto Fastplan, el Backend actúa como el núcleo orquestador que centraliza la gestión de los planes de emergencia y es necesario conocer los siguientes conceptos técnicos para entender el sistema.

2.3.1 Arquitectura Cliente-Servidor

La arquitectura Cliente-Servidor es un modelo de diseño de software distribuido que separa las tareas entre el servidor, que provee los recursos y el cliente, los que demandan estos recursos. Su funcionamiento se basa en el ciclo de Petición-Respuesta, donde el cliente solicita información a través de la red y el servidor procesa la lógica para devolver una respuesta (Tanenbaum & Wetherall, 2012).

A esta arquitectura se le pueden asociar algunas tecnologías estándar para una implementación efectiva, estas son:

- Protocolos de comunicación: HTTP/HTTPS para la transmisión de información.
- Interfaces de comunicación: APIs RESTful para la interacción o peticiones.
- Formato de comunicación: JSON estándar ligero.

Este modelo se ve aplicado en Fastplan, ya que este es fundamental para centralizar la lógica y la seguridad en el Backend y permitir que el Frontend Web funcione como cliente independiente que consume y visualiza los datos en tiempo real.

2.3.2 APIs RESTful

El estilo arquitectónico REST (*Representational State Transfer*) está diseñado para sistemas distribuidos que utilizan el protocolo HTTP para la comunicación y transferencia de datos. Se basa en el concepto de datos como recurso, los cuales son identificados por una URL y manipulados o accedidos mediante métodos estándar: GET (consultar), POST (crear), PUT (actualizar) y DELETE (eliminar). Una característica fundamental es su condición Stateless (sin estado), donde cada petición del cliente al servidor debe contener toda la información necesaria para ser procesada, sin depender de sistemas de almacenaje en el servidor. Esto no se considera un impedimento para que la información sea almacenable o cacheable.

Este estilo arquitectónico fue el utilizado en el proyecto, implementando y definiendo una API RESTful como interfaz de comunicación. En conjunto al uso de solicitudes HTTP y formato JSON, se permitirá que las operaciones críticas como descargar un plan de emergencia o reportar un incidente se realicen asegurando una integración ligera y rápida entre el ecosistema Fastplan.

2.3.3 Base de datos y persistencia de la información

La persistencia se refiere a la capacidad del sistema para conservar los datos de manera duradera, asegurando su disponibilidad más allá de la ejecución del proceso que los generó. Para lograr esto, se utilizan Sistemas de Gestión de Bases de Datos (SGBD), los que permiten almacenar, organizar y recuperar la información de manera eficiente.

En sistemas críticos, se prioriza el uso de Bases de Datos Relacionales (SQL). Estas garantizan la Integridad Referencial y cumplen con las propiedades ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad), asegurando que las transacciones de datos sean seguras y no generen errores o incongruencias en la información almacenada.

Dado que Fastplan maneja información de seguridad vital, la capa de persistencia es fundamental. Se utiliza una base de datos relacional para estructurar jerárquicamente la información de los edificios, sus planes de emergencia y los elementos críticos asociados, asegurando que, ante cualquier fallo del sistema o reinicio del servidor, los datos históricos y operativos permanezcan intactos y consistentes para su consulta inmediata.

A pesar de que esta tesina no se centra en el desarrollo de la Base de datos del proyecto, es necesario conocer estos conceptos, ya que, el backend Web se encarga de integrar la base datos permitiendo que el ecosistema pueda accederla y gestionarla.

2.3.4 Seguridad, autenticación y control de acceso

La seguridad en aplicaciones web se sustenta en garantizar la confidencialidad e integridad de la información. Para ello, se implementan dos mecanismos fundamentales: la Autenticación, siendo el proceso mediante el cual se verifica la identidad de los usuarios llegando a utilizar credenciales o tokens, y la Autorización, que determina los privilegios y permisos específicos que dicho usuario posee sobre los recursos del sistema.

En arquitecturas modernas, el estándar predominante es el uso de Tokens de Sesión, las cuales permiten validar la identidad del usuario en cada petición sin necesidad de mantener sesiones abiertas en el servidor.

Contexto en Fastplan La seguridad es un pilar crítico del proyecto, dado que la gestión de la información estratégica de edificios es netamente accedida por el Cuerpo de Bomberos, necesitando autenticar a los usuarios y dar los permisos necesarios correspondientes.

2.4 Estado del Arte

El desarrollo de software respecto a los sistemas de la información crítica ha migrado desde arquitecturas monolíticas hacia modelos distribuidos y orientados a servicios. En este contexto, el "Estado del Arte" se define por el uso de tecnologías que garantizan alta disponibilidad, escalabilidad y respuesta en tiempo real. A continuación, se describen los estándares industriales que fundamentan la arquitectura de Fastplan. (Bass, Clements & Kazman, 2022; Richards & Ford, 2020).

- **Entornos de Ejecución No Bloqueantes:** La necesidad de manejar múltiples conexiones simultáneas sin degradar el rendimiento ha posicionado a los modelos de E/S no bloqueantes (*Non-blocking I/O*) como el estándar actual. Node.js lidera este paradigma gracias a su arquitectura basada en un Bucle de Eventos (*Event Loop*), la cual permite gestionar operaciones de alta concurrencia con un consumo de recursos significativamente menor que los servidores tradicionales basados en hilos, siendo la elección predilecta para aplicaciones que requieren baja latencia. (Tilkov & Vinoski, 2010; Node.js Foundation, 2025).
- **Intercambio de Datos y Arquitectura API:** La comunicación entre sistemas heterogéneos se realiza habitualmente mediante interfaces de programación de aplicaciones (Application Programming Interfaces, API). Dentro de este ámbito, el estilo arquitectónico REST (*Representational State Transfer*) constituye uno de los modelos más utilizados para el diseño de servicios web debido a su simplicidad, escalabilidad y desacoplamiento entre cliente y servidor (Fielding, 2000). Complementariamente, JavaScript Object Notation (JSON) se ha consolidado como uno de los formatos de intercambio de datos más utilizados gracias a su estructura ligera, legible y compatible con la mayoría de los lenguajes de programación, facilitando la interoperabilidad entre aplicaciones web y móviles (Bray, 2017).
- **Persistencia Políglota (Polyglot Persistence):** La industria ha superado la dicotomía entre SQL y NoSQL mediante estrategias de Persistencia Políglota, donde cada tecnología se utiliza según la naturaleza de la información almacenada. En FastPlan, la información estructural y relaciones entre entidades se almacena en MySQL, debido a que su modelo relacional garantiza integridad referencial, consistencia transaccional mediante propiedades ACID y soporte para consultas complejas, características fundamentales para sistemas donde la pérdida o inconsistencia de información puede afectar la toma de decisiones. Por otra parte, la sincronización en tiempo real entre el servidor y la aplicación móvil se implementa mediante Cloud Firestore, cuya arquitectura NoSQL prioriza la disponibilidad, la baja latencia y la propagación inmediata de cambios, facilitando la entrega instantánea de información operativa durante una emergencia (Silberschatz et al., 2020; Oracle, s.f.; Google Developers, s.f.).

- **Seguridad Stateless:** En aplicaciones web modernas resulta habitual implementar mecanismos de autenticación basados en JSON Web Token (JWT), estándar definido por el IETF mediante la especificación RFC 7519. Este mecanismo permite transportar información firmada digitalmente entre el cliente y el servidor, eliminando la necesidad de mantener sesiones almacenadas en memoria y facilitando arquitecturas Stateless, donde cada solicitud contiene toda la información necesaria para validar la identidad del usuario (Jones, Bradley & Sakimura, 2015). Habitualmente, JWT se integra con protocolos de autorización como OAuth 2.0, ampliamente utilizado para delegar procesos de autenticación a proveedores externos de identidad, entre ellos Microsoft Entra ID y Google Identity Platform (Hardt, 2012; Microsoft, 2025).
- **Geolocalización:** La incorporación de información geoespacial constituye una herramienta relevante en plataformas orientadas a la gestión de emergencias. En este ámbito, OpenStreetMap representa uno de los proyectos colaborativos de cartografía abierta más utilizados, proporcionando información geográfica libre y permanentemente actualizada por la comunidad internacional (OpenStreetMap Foundation, 2025). Sobre esta infraestructura opera Nominatim, servicio de geocodificación que permite convertir direcciones en coordenadas geográficas y viceversa, facilitando la representación espacial de la información dentro de aplicaciones que requieren posicionamiento geográfico (Nominatim Project, 2025).

3 Propuesta de Solución

El presente capítulo tiene como objetivo el presentar la articulación de los componentes lógicos y la infraestructura centrando el análisis en la construcción del Backend. Para ello, se detallará el diseño e implementación técnica de la solución propuesta para la plataforma Fastplan, desglosando la arquitectura del sistema, explicando y justificando la selección de tecnologías y describiendo el flujo integro de la información.

3.1 Arquitectura del Proyecto Fastplan

La propuesta considera 2 componentes esenciales para su realización, estos son la plataforma Web designada al uso de gestión de la información desde la central de bomberos y la aplicación Móvil que por su contraparte se designa al ámbito operativo ejercido por bomberos en terreno. Entonces, se ha diseñado una arquitectura de software multicapa basada en el modelo Cliente-Servidor, esta se subdivide no solo en Cliente y servidor, sino que a su vez en la plataforma web con cliente y servidor web (Frontend y Backend) como la aplicación móvil con cliente y servidor móviles (Frontend y Backend).

Tal como se ilustra en la figura 3.1, la arquitectura puede separarse en las capas: Cliente, Servidor, Capa de Datos y Servicios Externos. Este diseño en la arquitectura busca facilitar el desarrollo modular, mantenimiento escalable y la integración eficiente de servicios como proyecto.

3.1.1 Capa Cliente

Esta capa constituye la interfaz de presentación y el punto de interacción directa con los usuarios finales. Como se nombró anteriormente, se descompone de dos aplicaciones cliente, las cuales pueden ser diferenciadas según el perfil de usuario y el contexto operativo:

- **Plataforma Web:** Una interfaz web diseñada para la gestión administrativa, carga de planos y supervisión del sistema por parte de inspectores. Se espera que el usuario final sea un administrador encargado en la Central de bomberos.
- **Aplicación Móvil:** Una aplicación diseñada para Tablets, debido a que el cuerpo de bomberos Viña del Mar cuenta con una Tablet en cada carro bomba, esta está enfocada en el despliegue rápido de información crítica para los equipos de respuesta durante una emergencia. Se espera que el usuario final sean los bomberos en terreno.

Ambas aplicaciones cliente no procesan ninguna lógica de negocio compleja ni almacenan datos de forma permanente; su función es netamente visualizar la información y capturar las acciones del usuario como solicitudes para enviarlas a su respectiva capa de servicios (Backend).

3.1.2 Capa Servidor

Esta capa representa el núcleo lógico del sistema y el foco principal del presente desarrollo. Esta capa actúa como intermediario inteligente, centralizando la lógica de negocio, la seguridad y la orquestación de datos.

La comunicación entre la Capa Cliente y la Capa de Servicios se realiza mediante peticiones REST, comunicación HTTP y formato JSON como estándar ligero para el intercambio de datos. La arquitectura como en la capa cliente, identifica dos componentes funcionales en esta capa:

- Backend Web: Desarrollado utilizando el entorno de ejecución Node.js y el framework Express. Este servidor procesa las solicitudes provenientes de la Plataforma Web y coordina las interacciones con la base de datos y servicios externos.
- Backend Móvil: Un componente homólogo encargado de atender las peticiones específicas de la Aplicación Móvil, asegurando que los datos entregados estén optimizados para el consumo en terreno.

Esta capa no posee una comunicación directa entre servidores (Backends), sino que se integran los servicios de Firebase Firestore el cual actúa como un repositorio de estado compartido en tiempo real, lo que permite que ambos componentes sincronicen información específica mediante la suscripción a eventos de datos, garantizando que la información fluya sin bloquear los procesos internos de cada servidor. Esto permite una comunicación mucho más rápida y fluida, sin solicitudes de extras de los Backends.

3.1.3 Capa de Datos

Es la responsable de la persistencia y la integridad de la información del sistema. Fastplan utiliza una Base de Datos relacional MySQL como repositorio central. Tanto el Back-end Web como el Back-end Móvil interactúan directamente con esta capa mediante la ejecución de consultas estructuradas conocidas como Querys SQL para realizar operaciones de tipo CRUD (de las siglas inglesas: leer, escribir, actualizar, eliminar) y registrar la información, ya sea para planes de emergencia, usuarios, elementos críticos, etc.

3.1.4 Capa de Servicios Externos

Esta capa explica los servicios de terceros especializados de Fastplan, con el objetivo de ofrecer un buen servicio y sin comprometer el rendimiento interno:

- Georreferenciación y Mapas: Se utiliza la API de Nominatim para servicios de geocodificación y la obtención de coordenadas Latitud/Longitud a partir de direcciones, esta es consumida principalmente por el Frontend y el backend la usa para la verificación de la información. Para la visualización interactiva de mapas por la capa de Clientes, se integra la librería Leaflet.
- Sincronización y Notificaciones: Se emplea la plataforma Firebase como servicio de infraestructura para gestionar la comunicación en tiempo real. Los Backends utilizan esta integración para realizar operaciones *Push*, enviando notificaciones críticas de manera instantánea para que sean consumidas por el otro componente Backend.

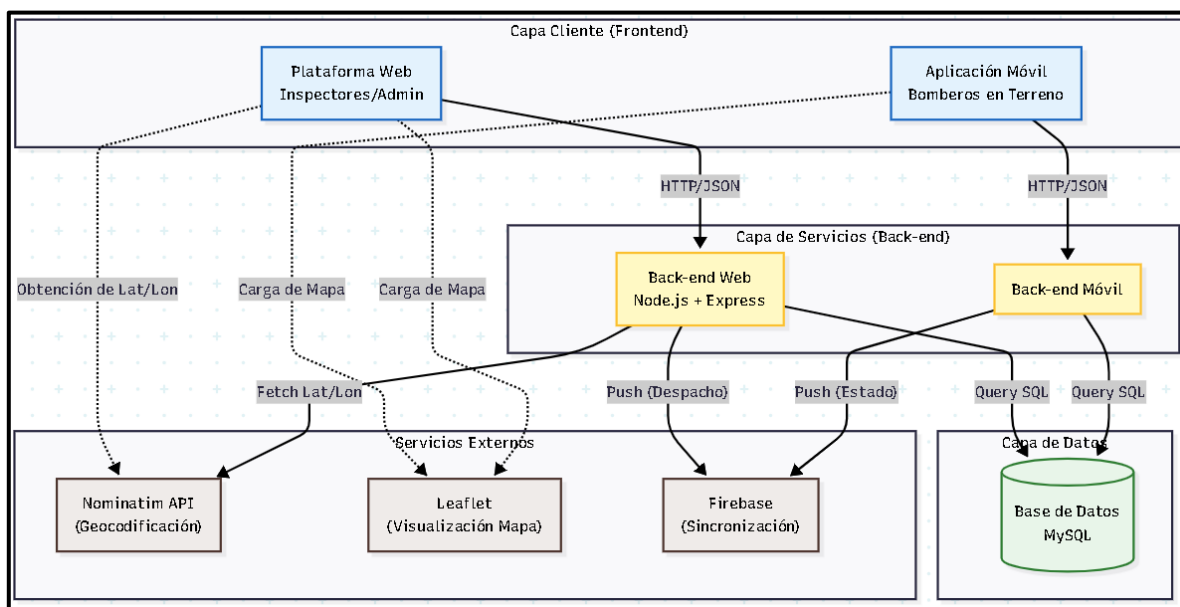


Figura 3.1: Diagrama de la Arquitectura de Software Fastplan.
Fuente: Elaboración Propia.

3.2 Flujo general de la información

El diseño del flujo de información en Fastplan responde a la necesidad de mantener tanto una gestión administrativa robusta como una gestión inmediata de la información crítica requerida durante una operación de respuesta. Como se ilustra en la figura 3.2, este diagrama del flujo de la información muestra como el sistema opera bajo dos estados funcionales diferenciados por momentos: el "Momento de Calma" y el "Momento de la Emergencia".

A continuación, se detalla el comportamiento del flujo de datos en cada uno de estos escenarios con sus respectivas funciones o acciones.

3.2.1 Flujo en Estado Rutinario

Correspondiente al "Momento de Calma" de la figura 3.2, este escenario describe la operatividad estándar del sistema fuera de una situación de crisis, caracterizado por tareas de gestión y monitoreo pasivo de los planes de emergencia como su información crítica. Funciones/Acciones:

- **Gestión Administrativa y Persistencia:** Desde la Central de Bomberos, los operadores utilizan la plataforma web para realizar tareas de administración. Como muestra el diagrama, las acciones administrativas generan un flujo de datos directo hacia la Base de Datos MySQL. Esto garantiza que la información estructural y los planes de emergencia se almacenen de forma persistente y segura en el repositorio central.
- **Monitoreo en Tiempo Real:** Simultáneamente, las unidades móviles situadas en los carros respectivos mantienen sus dispositivos conectados en un estado de escucha o alerta para estados de crisis a través del Servicio Firebase. En este estado, no hay tráfico constante de datos pesados, sino una conexión ligera a la espera de cualquier señal de alerta.

3.2.2 Flujo en Estado Operativo

Correspondiente al “Momento de la Emergencia” de la figura 3.2, este flujo hace referencia al momento que la central de bomberos es notificada de una emergencia, bajo estas circunstancias es que se prioriza la velocidad de comunicación y la sincronización de estados entre todos los actores. Funciones/Acciones:

- **Despacho y Notificación:** Al ocurrir una emergencia, la Central de Bomberos gestiona los carros de los que dispone y los envía a terreno para solventar la emergencia, aquí es cuando dentro de Fastplan se selecciona el plan de emergencia respectivo y se envía al carro correspondiente. A diferencia de las tareas administrativas, este flujo no se enfoca en la persistencia de la información, sino que se canaliza en la comunicación con la aplicación móvil a través del Servicio Firebase. Esto permite que la notificación y los datos críticos del plan sean empujados (push) de manera prácticamente instantánea al dispositivo del Carro de respuesta, que ya se encontraba en estado de escucha.
- **Actualización de Estado y Sincronización:** Cuando la central envía un plan de emergencia como una vez que la unidad despachada ha atendido la emergencia y lo marca en la aplicación móvil, estas interacciones provocan un cambio de estado en la disponibilidad del carro almacenado en la Base de datos.

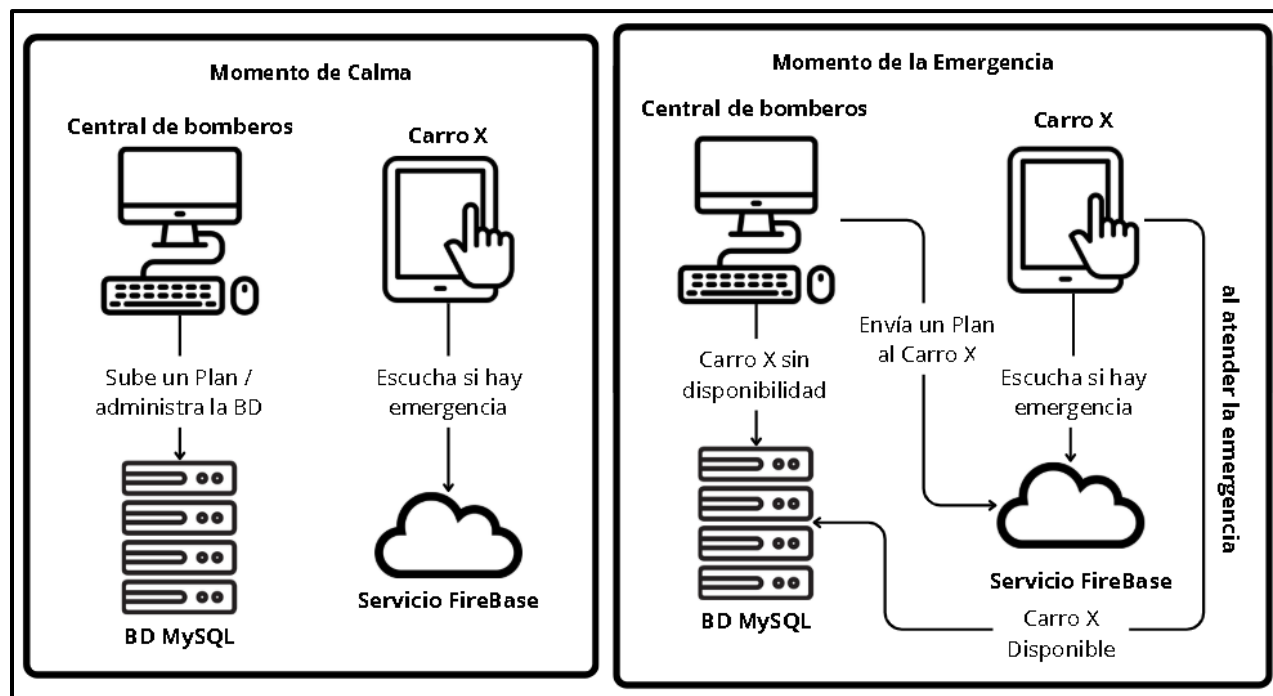


Figura 3.2: Diagrama del Flujo de la información Fastplan.
Fuente: Elaboración propia.

3.3 Tecnologías del Backend Propuesto

La selección de tecnologías para Fastplan se fundamentó en criterios de alta disponibilidad, escalabilidad y capacidad de respuesta en tiempo real, requisitos indispensables para cualquier sistema que pueda ser orientado a la gestión de emergencias. A continuación, se detalla la infraestructura tecnológica agrupada por su función dentro de la arquitectura.

3.3.1 Entorno de Ejecución y Framework

- **Node.js:** Se seleccionó este entorno de ejecución basado en el motor V8 de Google Chrome debido a su modelo de E/S no bloqueante y orientado a eventos. Esta característica permite manejar un alto volumen de conexiones concurrentes con baja latencia, lo cual es crítico en caso de generar alertas simultáneas a múltiples unidades de bomberos sin degradar el rendimiento del servidor.
- **Express.js:** Framework web minimalista utilizado para orquestar la arquitectura del servidor. Su implementación facilita la creación de una API RESTful robusta mediante un sistema organizado de enrutamiento y middlewares correspondientes, permitiendo modularizar la lógica de negocio y simplificar el manejo de las peticiones HTTP (GET, POST, PUT, DELETE).

3.3.2 Persistencia y Sincronización de Datos

Para la gestión de datos se optó por un enfoque híbrido que combina la integridad relacional con la velocidad del tiempo real:

- **MySQL:** Sistema de Gestión de Bases de Datos Relacional (RDBMS) encargado de la persistencia de la información estructural. Se utiliza para almacenar datos que requieren relaciones estrictas y cumplimiento de propiedades ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad), tales como el inventario de edificios, registros de usuarios y elementos críticos.
- **Firebase Firestore:** Servicio de base de datos NoSQL alojado en la nube, implementado exclusivamente como capa de sincronización. Su función en Fastplan es actuar como un canal de comunicación en tiempo real (mediante *listeners*), permitiendo que la aplicación móvil reciba notificaciones de despacho instantáneas sin necesidad de realizar consultas repetitivas al servidor principal.

3.3.3 Servicios Externos y Utilidades

- **Nominatim (OpenStreetMap API):** Interfaz de programación utilizada para servicios de geocodificación inversa. Permite transformar direcciones postales en coordenadas geográficas (Latitud/Longitud) precisas, asegurando que los planes de emergencia sean ubicables en el mapa incluso si carecen de metadatos geoespaciales previos.
- **Passport.js con Microsoft Identity:** Middleware de autenticación modular. Se integra con el proveedor de identidad de Microsoft para validar credenciales institucionales, delegando la gestión de seguridad de contraseñas y permitiendo la emisión de tokens JWT (JSON Web Tokens) para el manejo de sesiones *stateless*.
- **Multer:** Middleware de Node.js especializado en el manejo de datos multipart/form-data. Es esencial para la gestión de archivos, permitiendo la carga, validación y almacenamiento eficiente de los documentos PDF correspondientes a los planes de emergencia.

3.3.4 Infraestructura y DevOps

- **Docker:** Plataforma de contenedorización utilizada para empaquetar la aplicación y sus dependencias en un entorno aislado. Esto garantiza la consistencia del software entre las etapas de desarrollo, pruebas y producción, eliminando conflictos de compatibilidad.
- **GitHub Actions:** Servicio de integración continua (CI/CD) que automatiza el flujo de trabajo del desarrollo. Se utiliza para ejecutar pruebas automáticas y validar la integridad del código en cada actualización (*push*), asegurando que los nuevos cambios no introduzcan errores en la lógica crítica del sistema.

3.4 Estructura del backend (Enrutamiento)

Como se puede ver en la figura 3.3, el enrutamiento corresponde al repositorio GitHub del Backend, donde se pueden ver algunos archivos o carpetas que suelen tener. Con este subcapítulo se busca explicar el diseño del mismo.

Repositorio descrito desde la carpeta Raíz:

- **Github/workflows/:** Esta carpeta posee los archivos para la continuidad operativa para el desarrollo del repositorio. estos archivos utilizan GitHub actions como Docker.
- **.vscode/:** Esta carpeta contiene el archivo settings.json que establece reglas de visualización para que los desarrolladores al ejercer su labor puedan visualizar el código de la misma forma.
- **Db/:** Esta carpeta contiene los archivos relacionados al levantamiento de la Base de datos para el entorno local. Estos archivos destacan en configurar la Base de datos MySQL en el entorno de desarrollo sin afectar el lanzamiento del proyecto en producción.
- **Src/:** Esta carpeta es el núcleo de la lógica de negocio aplicada en el backend, encontrándose carpetas y archivos relacionados a la gestión de la información como la integración con el Ecosistema Fastplan. Esta contiene las siguientes carpetas y archivos a destacar:
 - **Config/:** Podemos encontrar el archivo que configura el servicio de Firebase Firestore para establecer la comunicación con la aplicación móvil de Fastplan.
 - **Controllers/:** Los archivos de esta carpeta siguen una nomenclatura conformada con un nombre que describa el tipo de información que maneja el archivo (ej: cars) junto a la extensión ".controller.js". Estos archivos contienen las acciones de Tipo CRUD (crear, leer, actualizar y eliminar) con el estilo arquitectónico REST que aplica el Backend para gestionar la Base de datos MySQL. Con el fin de servir las APIs que llamen los archivos de la carpeta "routes", pueden acceder a ciertos servicios expuestos en los archivos de la carpeta "middleware" apuntando al archivo correspondiente.
 - **Middlewares/:** En esta carpeta se encuentran las importaciones del backend para la gestión y persistencia de la información. Los archivos configuran el uso de servicios como Microsoft o Multer que utilizan los archivos ".controller.js".

- **Routes/:** Se contienen los archivos con una nomenclatura conformada por las rutas que se relacionaran al archivo (Ej: Cars) terminando con la extensión “.routes.js”. Estos archivos exponen los Endpoints a los cuales el cliente frontend apunta para gestionar la información importando las funciones de los archivos “.controller.js”.
- **Tests/:** En esta carpeta se contienen los archivos que se utilizan para la validación del backend como propuesta. Estos archivos automatizan casos de pruebas específicos para una verificación específica.
- **Archivos sueltos en src/:**
 - **App.js:** Como es normal en los proyectos que utilizan Express, este archivo establece las rutas más simples del proyecto, en este caso hace llamado a cada archivo “.routes.js” para servir los Endpoints correctamente.
 - **Config.js:** se encarga de centralizar y cargar la configuración del proyecto, especialmente las variables de entorno que necesita el backend para funcionar correctamente.
 - **Db.js:** Configura la conexión mediante Pools que se usan para establecer comunicación con la Base de datos MySQL y permitir acciones de tipo CRUD (Crear, leer, actualizar, eliminar).
 - **dbInit.js:** Establece la inicialización y levantamiento de la base de datos local designada al entorno de desarrollo.
 - **Index.js:** es el punto de entrada del backend, es decir, el archivo que efectivamente levanta el servidor. Importa las rutas desde “app.js”, inicializa la Base de Datos con el archivo “dbInit.js” de ser necesario y expone el puerto de acceso para servir los Endpoints al cliente Frontend.
- **Archivos sueltos en Raíz:** Se explicaran solo los archivos clave de la carpeta raíz, ignorando aquellos que no aporten un mejor entendimiento.
 - **Dockerfile:** Se definen las instrucciones necesarias para construir una imagen docker que contiene y ejecuta el backend para el entorno de desarrollo del proyecto. Es capaz de crear y replicar el repositorio haciendo copia de los servicios utilizados, variables de entorno, etc.
 - **Firebase.json:** Este archivo contiene la credencial del Service Account de Firebase, necesaria para permitir que el backend utilice los servicios de Firebase Firestore.

El enrutamiento clave para la escalabilidad del backend web radica en la disposición de archivos en la carpeta sources, donde los archivos de “routes” y “controllers” establecen las APIs RESTful. En caso de la extensión del proyecto para gestionar nuevos tipos de datos, deben crearse las acciones tipo CRUD (crear, leer, actualizar, eliminar) necesarias en un archivo “.controller.js” destinado a la carpeta “controllers”, para usarlas se debe crear el archivo “.routes.js” en la carpeta “routes” con los endpoints a los que se apuntaran en el cliente Frontend. Para un orden claro y fácilmente replicable, los nombres de los archivos “.controller.js” y “.routes.js” deben iniciar con el tipo de dato que gestionan (Ej: usuarios.controller.js y usuarios.routes.js). En caso de modificación de la gestión solo deben aplicarse los cambios en los archivos correspondientes aplicando la misma lógica.

Para la configuración de nuevos servicios, en la carpeta “middlewares” deben ir los archivos que apliquen los servicios mediante importaciones, configurando y estableciendo las funciones claves que se utilizaran. En cambio, las configuraciones para establecer conexión con servicios externos deben ir en la carpeta “config”. Las modificaciones de servicios aplican la misma lógica de los nuevos servicios.

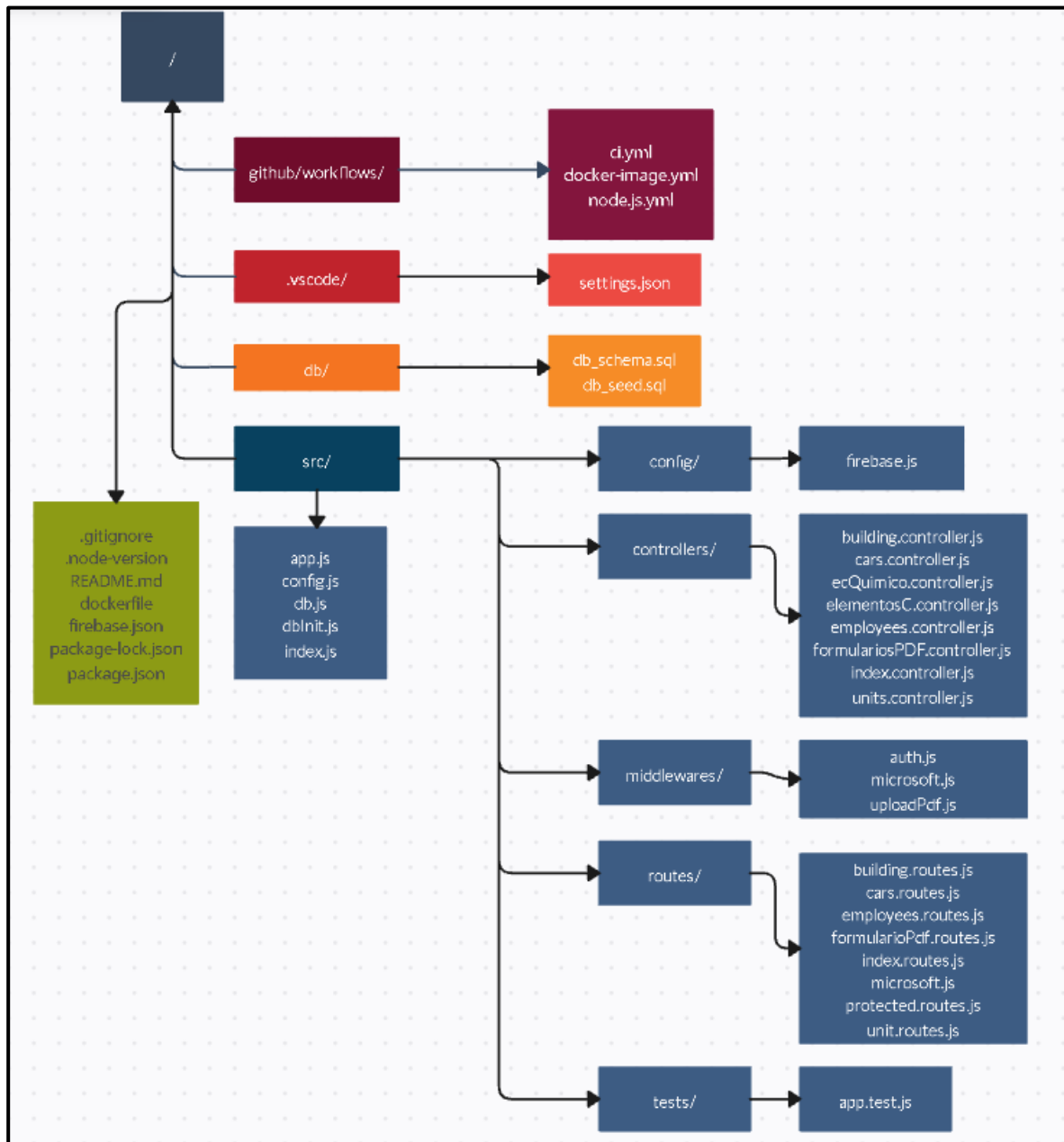


Figura 3.3: Diagrama de enrutamiento Backend Fastplan
Fuente: Elaboración Propia

3.5 Integración con el ecosistema Fastplan

Como se ha establecido previamente, el Backend de Fastplan no opera como una entidad aislada, sino que actúa como el núcleo orquestador del ecosistema. Su función principal es garantizar la interoperabilidad y el flujo coherente de datos entre las distintas interfaces y los servicios distribuidos. A continuación, se detalla la estrategia de integración para cada componente.

3.5.1 Integración con el Cliente Frontend

La integración con el cliente Frontend se efectúa mediante el consumo directo de la API RESTful. El Backend expone endpoints seguros que permiten al panel administrativo realizar operaciones CRUD complejas sobre entidades críticas (edificios, usuarios, planes). El intercambio de datos se realiza estrictamente en formato JSON, asegurando una interpretación ligera y rápida por parte de la interfaz desarrollada en React.

3.5.2 Integración con la aplicación Móvil

El Backend no se comunica directamente con la aplicación móvil, sino que este escribe en el servicio de Firebase Firestore la información que necesita comunicar, delegando así la tarea de "empujar" (*push*) los datos a los dispositivos en tiempo real.

El método de esta comunicación se debe a la naturaleza bajo la que se da, donde dados los avances actuales del proyecto, la comunicación es netamente en caso de emergencias, donde este método destaca por su inmediatez de difusión de la información.

3.5.3 Integración con la Base de Datos

Para entender la integración con la Base de datos hay que entender que, al establecer una conexión entre el Backend y la base de datos, requiere un proceso de apertura y cierre en los cuales se establece la comunicación, generando cierto grado de latencia que, tras varias consultas, puede suponer cierto retraso en la gestión de recursos. Según esto, la comunicación con el servidor MySQL se gestiona mediante un patrón de Pool de Conexiones (*Connection pooling*) que se inicia junto al servidor Backend, instanciando un conjunto predefinido de conexiones reutilizables dirigidas a la base de datos. Esto evita la necesidad de establecer un nuevo enlace TCP (*handshake*) para cada solicitud, donde en cambio el sistema solicita una conexión activa del pool, ejecuta la consulta necesaria y libera el recurso inmediatamente para que quede disponible para nuevas peticiones, optimizando la gestión de recursos del sistema.

Paralelamente a la eficiencia, esta capa asume un rol crítico de seguridad. La integración no permite el paso directo de datos crudos hacia la base de datos; en su lugar, el Backend actúa como una barrera lógica que implementa Consultas Preparadas (*Prepared Statements*) y rutinas de saneamiento. Este enfoque asegura que toda información proveniente de los clientes sea tratada estrictamente como datos y no como código ejecutable, neutralizando amenazas de Inyección SQL y garantizando la integridad referencial de la información antes de su persistencia.

3.6 Seguridad y Autenticación

La estrategia de seguridad de Fastplan se limita al acceso de la plataforma web destinada a la gestión de planes de emergencia como el sistema de alerta de emergencias en respuesta a las mismas, estas son funciones limitadas a usuarios específicos designado por bomberos, por esto el sistema de autenticación y acceso a esta información es crítica en el sistema. No se contempla la necesidad de encriptación de información debido a que la naturaleza de los planes de emergencia no tiene una necesidad de ser reservada dado su propósito.

Con lo anterior, se ha diseñado una seguridad bajo un enfoque de defensa simple pero robusto para evitar vulnerabilidades de acceso, para esto se implementan controles en múltiples capas para garantizar la confidencialidad e integridad del sistema. La arquitectura de seguridad se estructura en tres pilares fundamentales: autenticación delegada, validación con la Base de Datos y gestión de sesiones como control de acceso.

3.6.1 Autenticación Federada (OAuth 2.0)

Para evitar la gestión directa de contraseñas y reducir vectores de ataque, el sistema implementa un modelo de Autenticación delegada utilizando Microsoft Identity Platform como proveedor de identidad. La implementación técnica se realiza mediante la librería Passport.js, la cual gestiona el flujo de autorización bajo el estándar OAuth 2.0. El proceso valida las credenciales del usuario directamente con los servidores de Microsoft; verificando que las credenciales sean de un usuario existente en Microsoft hasta la pertenencia de este en la empresa asociada. El sistema al hacer esta autenticación se dedica a esperar una respuesta de confirmación para continuar el proceso.

3.6.2 Autenticación de permisos en la Base de Datos

Una vez confirmada la autenticación por Microsoft, el sistema es notificado que el usuario es válido dentro de los márgenes de Microsoft, pero no es capaz de indicar el rol o permisos asociados con respecto al proyecto; entonces, tras esta confirmación el backend consulta a la Base de datos si el email del usuario está registrado para tener acceso a la plataforma, si el usuario está efectivamente registrado y la respuesta es satisfactoria, el proceso continuará, en caso contrario se rechazará la conexión.

3.6.3 Gestión de Sesiones y Gestión de Acceso vía JWT (JSON Web Tokens)

Dada la naturaleza RESTful de la API, se optó por un manejo de sesiones Stateless (sin estado). Una vez que el usuario ha sido verificado como usuario con acceso a la plataforma, el servidor genera y firma un JSON Web Token (JWT). Este token encapsula la información esencial (identificador de usuario, correo y rol) y actúa como llave de acceso temporal para las peticiones subsiguientes, estableciendo un tiempo de expiración de la *Token*. El uso de JWT permite validar la identidad en cada solicitud HTTP sin necesidad de consultar repetidamente la base de datos de usuarios, mejorando la eficiencia del sistema.

La JWT permite al backend dar un control de Acceso Basado en Roles. Esta lógica del Backend no está aplicada dado que solo Administradores tendrán acceso, pero le permite al sistema discriminar privilegios y restringir *endpoints* críticos según un rol específico que puede declararse en el token (ej. Administrador o Visualizador), garantizando la escalabilidad de las políticas de seguridad.

3.7 Integración de APIs RESTful

La capa de servicios de Fastplan expone su funcionalidad mediante una interfaz de programación RESTful (*Representational State Transfer*), diseñada bajo un enfoque orientado a recursos. Este estándar garantiza la interoperabilidad entre el Backend y el cliente Frontend, desacoplando la lógica de presentación de la lógica de datos.

3.7.1 Estructura y Semántica HTTP

El diseño de los *endpoints* respeta estrictamente la semántica del protocolo HTTP, asociando las operaciones de gestión de datos (CRUD) con los verbos estandarizados:

- **GET:** Para la consulta y recuperación de recursos (Ej. Obtener un plan de emergencia).
- **POST:** Para la creación de nuevos registros (Ej. Añadir un carro).
- **PATCH:** Para la modificación parcial de recursos (Ej. Actualizar la disponibilidad de un carro).
- **DELETE:** Para la eliminación lógica o física de registros (Ej. Eliminación de un usuario).

Asimismo, la API implementa un manejo consistente de Códigos de Estado HTTP que establecen los resultados de una consulta como el uso de formato JSON para las respuestas, asegurando una estructura de datos predecible y fácil de consumir.

3.7.2 Organización Modular del Código

Como se nombró en el capítulo 3.4, para garantizar la mantenibilidad y escalabilidad del código, la arquitectura interna del Backend implementa una separación de Responsabilidades. Cada módulo funcional se divide en dos componentes lógicos:

- **Enrutadores (Routes):** Archivos encargados exclusivamente de definir los puntos de entrada (*endpoints*).
- **Controladores (Controllers):** Archivos que contienen la lógica de negocio, procesan la entrada de datos e interactúan con la capa de persistencia. Si es necesario se implementan los middlewares necesarios para la gestión.

3.7.3 Dominios Funcionales

La API organiza sus servicios en torno a las entidades críticas del negocio, abarcando los siguientes módulos principales:

- **Gestión Operativa:** Endpoints para la administración de Carros y el seguimiento de Unidades Despachadas.
- **Infraestructura y Riesgo:** Servicios dedicados a la gestión de Edificios, carga de Planes en PDF y geolocalización de Elementos Críticos.
- **Administración y Seguridad:** Rutas para el control de Empleados/Usuarios y los servicios de Autenticación.

A continuación, la tabla 3.1 detalla la especificación técnica de los principales *endpoints* disponibles en el sistema.

Dominio Funcional	Método HTTP	Endpoint	Descripción
Administración y Seguridad	POST	/api/employees	Crear un Usuario con acceso a la plataforma.
	GET	/api/employees	Leer todos los Usuarios.
	GET	/api/employees/:id	Leer un Usuario.
	PATCH	/api/employees/:id	Editar un Usuario.
	DELETE	/api/employees/:id	Eliminar un Usuario.
	GET	/auth/microsoft	Redirección al servicio de autenticación de Microsoft.
	GET	/auth/microsoft/callback	Validación del usuario y generación del JWT
Gestión Operativa	POST	/api/cars	Crear un Carro.
	PATCH	/api/cars/:id	Editar un Carro.
	DELETE	/api/cars/:id	Eliminar un Carro.
	GET	/api/cars	Leer todos los Carros.
	PATCH	/api/dispatch-unit	Marcar un Carro como disponible para su uso en operaciones.
Infraestructura y Riesgo	POST	/api/plans	Crear un Plan de Emergencia.
	GET	/api/plans	Leer todos los Planes de Emergencia.
	GET	/api/plans/:id	Leer un Plan de Emergencia.
	PATCH	/api/plans/:id	Editar un Plan de Emergencia.
	DELETE	/api/plans/:id	Eliminar un Plan de Emergencia.
	GET	/api/types-buildings	Leer los tipos de edificios para uso de selección en la creación de planes.
	POST	/api/upload-pdf	Subir un Archivo PDF para asociarlo a un Plan de emergencia.
	GET	/api/upload-pdf/:id	Obtención del Archivo PDF de un Plan de emergencia.
	POST	/api/elements	Crear un Elemento Crítico.
	GET	/api/elements	Leer todos los Elementos Críticos.
	GET	/api/elements/:id	Leer un Elemento Crítico.
	PATCH	/api/elements/:id	Editar un Elemento Crítico.
	DELETE	/api/elements/:id	Eliminar un Elemento Crítico.
	GET	/api/types-elements	Leer los tipos de Elementos Críticos para uso de selección la creación de uno.
	POST	/api/chemical-elements	Crear un Elemento Químico.
	GET	/api/chemical-elements	Leer todos los Elementos Químicos.
	GET	/api/chemical-elements/:id	Leer un Elemento Químico.
	PATCH	/api/chemical-elements/:id	Editar un elemento Químico.
	DELETE	/api/chemical-elements/:id	Eliminar un elemento Químico.
	POST	/api/upload-sheet	Subir un Archivo PDF y asociarlo a un Elemento Químico.
	GET	/api/upload-sheet/:id	Obtención del Archivo PDF de un Plan de emergencia.

Tabla 3.1: Endpoints mínimos como APIs RESTful de Fastplan

Fuente: Elaboración Propia

3.8 Almacenamiento de Archivos

La plataforma Fastplan requiere la administración de documentos críticos, específicamente los planes de emergencia en formato digital y fichas técnicas de materiales peligrosos. Dado que la arquitectura REST tradicional maneja principalmente datos JSON, se implementó una estrategia específica para la gestión de solicitudes "multipart/form-data" mediante el middleware Multer.

El proceso de almacenamiento se ha diseñado bajo un estricto protocolo de cuatro etapas para garantizar la seguridad y la integridad del servidor:

1. **Intercepción y Buffer:** El sistema intercepta el flujo de datos entrante desde el Frontend antes de que llegue al controlador lógico, procesando el archivo en memoria.
2. **Validación de Tipos (MIME Type Check):** Se aplica un filtro de seguridad que verifica el *Magic Number* o tipo MIME del archivo. El sistema rechaza automáticamente cualquier documento que no corresponda al estándar application/pdf, previniendo la subida de scripts maliciosos o ejecutables.
3. **Normalización y Renombrado:** Para evitar colisiones de nombres (archivos con el mismo nombre que se sobrescriben) y problemas con caracteres especiales, se aplica una rutina de renombrado seguro. Cada archivo recibe un identificador único basado en la marca de tiempo (timestamp) y un sufijo aleatorio, conservando su extensión original.
4. **Persistencia Estructurada:** Finalmente, el archivo físico se aloja en el sistema de archivos local del servidor (*Local File System*), distribuido en directorios segregados según su naturaleza. Simultáneamente, el Backend genera una URL relativa que se almacena en la base de datos MySQL, vinculando el recurso físico con la entidad lógica (Edificio o Elemento).

3.9 Análisis Comparativo del Modelo Operativo

La implementación de Fastplan introduce un cambio significativo en la metodología de respuesta ante emergencias, permitiendo transitar desde un modelo operativo reactivo hacia uno proactivo. La solución agiliza la gestión de la información crítica mediante la visualización geoespacial, transformando datos abstractos en inteligencia táctica desplegable.

Respecto a la dependencia de conectividad, la arquitectura del sistema mitiga los efectos de la latencia; la sincronización de datos se produce en segundo plano durante el trayecto, asegurando que la información cargada esté disponible en la interfaz móvil antes de la llegada a la zona cero, minimizando el impacto de posibles fluctuaciones en las redes de conexión Wifi.

A continuación, se detalla el contraste operativo entre ambos escenarios.

3.9.1 Escenario Actual

Bajo el modelo tradicional, la unidad de Bomberos se desplaza hacia el lugar del siniestro con un nivel considerable de incertidumbre operacional.

- **Acceso a la Información:** Los planos de emergencia suelen ser documentos físicos almacenados *dentro* del propio edificio afectado (a menudo en conserjerías o salas de administración), lo que hace imposible su consulta previa y arriesgado su rescate durante un incendio.
- **Identificación de Riesgos:** El oficial a cargo desconoce la ubicación exacta de materiales peligrosos o zonas de alto riesgo eléctrico hasta que realiza una inspección visual *in situ*.
- **Recursos Hídricos:** La ubicación de los grifos o la conexión a la Red Seca/Húmeda se busca visualmente al llegar, lo que puede retrasar el armado de las líneas de agua si estos están obstruidos o mal señalizados.
- **Consecuencia:** La estrategia de ataque se define recién al llegar al lugar, consumiendo minutos vitales en el reconocimiento del terreno y aumentando el riesgo para el personal.

3.9.2 Escenario implementando fastplan

Con la integración del sistema, la tripulación utiliza el tiempo de traslado (tiempo muerto) para realizar una planificación táctica anticipada.

- **Inteligencia en Tránsito:** Durante el viaje, el personal visualiza en la aplicación móvil la estructura del edificio, los accesos vehiculares habilitados y las restricciones de altura, permitiendo decidir de antemano dónde posicionar la máquina.
- **Geolocalización de Elementos Críticos:** El mapa interactivo revela la posición exacta de grifos cercanos y las conexiones de red seca antes de tener contacto visual con ellos, agilizando el despliegue de material.
- **Gestión de Amenazas:** Se identifican preventivamente las zonas con almacenamiento de sustancias peligrosas, permitiendo preparar los equipos de protección adecuados antes de descender del carro.
- **Consecuencia:** Al llegar a la emergencia, el equipo ya posee un conocimiento detallado del entorno ("Situational Awareness"), lo que permite una intervención inmediata, precisa y más segura, reduciendo drásticamente los tiempos de respuesta efectiva.

Variable Operativa	Escenario Actual	Escenario con Fastplan
Acceso a Planos	Físico, dependiente de estar en el edificio.	Digital, remoto y disponible durante el viaje.
Planificación	Reactiva, se planifica al llegar.	Proactiva, se planifica durante el traslado.
Ubicación de Grifos	Búsqueda personal al arribar (sujeto a problemas de identificación varios).	Ubicación exacta por GPS previo al arribo.
Materiales Peligrosos	Se descubren durante la intervención.	Se alertan anticipadamente en el mapa.
Toma de Decisiones	Basada en lo visible en el momento.	Basada en datos estructurales completos.

Tabla 3.2: Comparación de Escenarios Actual vs Implementación del Proyecto
Fuente: Elaboración propia

4 Validación de la Solución

El propósito de este capítulo es evaluar experimentalmente la funcionalidad, estabilidad y seguridad de la arquitectura Backend desarrollada para la plataforma Fastplan. El proceso de validación somete al servidor a escenarios de uso reales y de borde (*edge cases*) para verificar que la gestión de información crítica cumpla con las reglas de negocio y los estándares del sistema

El sistema *Fastplan* se caracteriza por ofrecer una variedad de servicios críticos que, por la naturaleza del sistema, deben operar con alta fiabilidad. Dada su importancia operativa, el proceso de validación se centrará en los siguientes elementos estructurales:

- **APIs RESTful:** Constituye el canal de comunicación primario con el cliente Web. Su validación es fundamental, ya que representa el punto de acceso a la lógica de negocio y la gestión de datos. Para su validación se considera la correcta implementación de los *Endpoints* destinados a la gestión de la Base de datos.
- **Seguridad y Autenticación:** Es el mecanismo encargado de restringir el acceso al ecosistema, garantizando que solo el personal del Cuerpo de Bomberos y usuarios autorizados puedan interactuar con el sistema. Se validará de la protección de rutas.
- **Gestión de Archivos:** Aunque su impacto es diferido respecto a la emergencia en tiempo real, este servicio es vital para la centralización y respaldo de documentos estratégicos, siendo el módulo encargado del almacenamiento, filtrado y recuperación como los planes de emergencia y hojas de seguridad. Se verificará la integridad en la carga y recuperación de estos activos digitales.
- **Integración con la Aplicación Móvil:** Mecanismo de orquestación y despacho en tiempo real orientado a las unidades de respuesta en terreno. Se validará el envío de información pertinente, asegurando esta fluya correctamente hacia los dispositivos de respuesta.

La identificación de estas áreas críticas permite segmentar la evaluación en pruebas específicas, facilitando el análisis de resultados para comprender el comportamiento integral del sistema.

4.1 Metodología de Validación

La estrategia de validación se fundamenta en los principios de Aseguramiento de Calidad de Software (QA), adoptando el enfoque de Pruebas de Caja Negra (*Black Box Testing*). Bajo este paradigma, los componentes del servidor son sometidos a estímulos de entrada controlados (peticiones HTTP, *payloads* JSON, archivos binarios y tokens de autorización) para auditar que las respuestas producidas (códigos de estado HTTP, estructuras JSON, archivos guardados y eventos emitidos) coincidan estrictamente con las especificaciones del sistema, sin evaluar la estructura interna del código.

El entorno de pruebas se automatizó utilizando el *framework* Jest en conjunto de la librería Supertest para la emulación de clientes HTTP en un entorno de integración continua. Con el objetivo de mantener un entorno determinista, rápido e independiente de factores externos o latencias de red, el proceso se ejecutó mediante un esquema de Pruebas en Aislamiento (*Isolated Testing*), ejecutando los tests sobre un servidor local.

4.2 Validación de las APIs RESTful

La técnica aplicada en este módulo corresponde a pruebas de contrato HTTP e integración con la capa de persistencia. El objetivo es auditar que la API RESTful respete la semántica del protocolo HTTP/1.1 y mantenga la integridad referencial al interactuar con MySQL mediante la ejecución de operaciones CRUD.

Antes de presentar los casos de prueba, es necesario entender la entidad registrada en la base de datos de “plan_emergencia” (Tabla 4.1) porque se utilizarán los *endpoints* dedicados a la gestión de los planes de emergencia para esta validación.

Variable	Tipo de dato	Descripción
ID plan	Numero entero	Identificador lógico. Generado automáticamente
ID usuario	Numero entero	El identificador del usuario que creo el dato. El Backend consulta a la Base de datos el ID del usuario actual en la página web y lo utiliza.
ID tipo edificación	Numero entero	El identificador de un tipo de edificación. El tipo de edificación es una entidad dentro de la Base de Datos.
Nombre Lugar	Texto	Nombre del Edificio o Construcción. No nulo.
Dirección	Texto	Dirección del Edificio o Construcción. No nulo.
Enlace PDF	Texto	Enlace creado para acceder al Archivo. No nulo.
ID estado	Numero entero	Identificador de un tipo de estado del Plan. El estado de un plan es una entidad dentro de la Base de Datos. Se genera automáticamente.
Fecha Creación	Tiempo	El formato de fecha y hora de creación en la Base de datos. Se genera automáticamente.
Fecha Actualización	Tiempo	El formato de fecha y hora de creación en la Base de datos. Se genera automáticamente.
Latitud	Numero flotante	Numero flotante referente a la ubicación geográfica respecto a latitud del edificio o construcción. No nulo.
Longitud	Numero flotante	Numero flotante referente a la ubicación geográfica respecto a longitud del edificio o construcción. No nulo.

Tabla 4.1: Entidad “plan_emergencia” de la Base de Datos Fastplan
Fuente: Elaboración Propia

Caso de Prueba	Entrada	Resultado esperado
Creación de Plan exitoso	Método: POST /api/planes. Body: JSON válido.	Código: 201 (<i>Created</i>). Retorno: JSON del plan creado y con mensaje de creación exitoso.
Creación de Plan incorrecto	Método: POST /api/planes. Body: JSON con variable "latitud": "Texto invalido".	Código: 400 (<i>Bad Request</i>). Retorno: Mensaje de error indicando valores inválidos.
Creación de Plan incompleto	Método: POST /api/planes. Body: JSON con variable "enlace PDF" : NULL.	Código: 400 (<i>Bad Request</i>). Retorno: Mensaje de error indicando valores inválidos.
Edición de un Plan exitoso	Método: PATCH /api/planes/x. Body: JSON con variable "nombre lugar" : "nuevo nombre".	Código: 200 (<i>OK</i>). Retorno: JSON del plan actualizado.
Edición de Plan con errores	Método: PATCH /api/planes/x. Body: JSON con variable "id_tipo_edificacion": ID que no exista en la tabla relacional.	Código: 400 (<i>Bad Request</i>). Retorno: Mensaje de error indicando valores inválidos.
Edición de Plan incompleto	Método: PATCH /api/planes/x. Body: JSON con variable "nombre lugar" : NULL.	Código: 400 (<i>Bad Request</i>). Retorno: Mensaje de error indicando valores inválidos.
Eliminación de un Plan	Método: DELETE /api/planes/x. Parámetro: ID válido de un plan existente.	Código: 200 (<i>OK</i>). Retorno: Mensaje de eliminación exitosa.
Eliminación de un Plan inexistente	Método: DELETE /api/planes/x. Parámetro: ID que no existe en la BD.	Código: 404 (<i>Not Found</i>). Retorno: Mensaje indicando objeto no encontrado.
Consultar todos los planes	Método: GET /api/planes. Parámetro: ninguno.	Código: 200 (<i>OK</i>). Retorno: Lista de JSONs con todos los planes.
Consulta de Plan válido	Método: GET /api/planes/x. Parámetro: ID válido.	Código: 200 (<i>OK</i>). Retorno: JSON del Plan x.
Consulta de Plan inexistente	Método: GET /api/planes/x. Parámetro: ID fuera de rango o borrado.	Código: 404 (<i>Not Found</i>). Retorno: Mensaje indicando Objeto no encontrado.

Tabla 4.2: Casos de Prueba para APIs RESTful

Fuente: Elaboración Propia

Se consideran casos de prueba exitosos únicamente si cumple simultáneamente con las siguientes condiciones:

- **Código de Estado HTTP:** Verificar que el servidor retorne el código esperado para cada caso de prueba definido.
- **Integridad del retorno:** Confirmar que el cuerpo de la respuesta JSON o Mensaje contenga los datos solicitados.

4.3 Validación de Seguridad

El backend implementa dos capas principales de seguridad. La primera corresponde al proceso de autenticación mediante una cuenta Microsoft válida, mecanismo administrado por Microsoft a través de OAuth 2.0 y Microsoft Entra ID (anteriormente Azure Active Directory). Dado que este proceso es gestionado por un proveedor externo y no forma parte del desarrollo propio del sistema, no se consideró necesario realizar pruebas exhaustivas sobre esta etapa.

Una vez superada la autenticación, la segunda capa de seguridad recae en la validación del Json Web Token (JWT) emitido durante el proceso de inicio de sesión. Esta validación es fundamental, ya que garantiza que únicamente usuarios autenticados puedan acceder a los recursos protegidos del backend.

Para verificar el correcto funcionamiento de este mecanismo, se desarrollaron pruebas automatizadas que simulan peticiones HTTP dirigidas a rutas protegidas del sistema. El estímulo consiste en enviar solicitudes HTTP al *endpoint* de validación `"/api/protected-route"` manipulando la cabecera `"Authorization: Bearer <token>"`, comprobando tanto escenarios de acceso autorizado como intentos de acceso mediante tokens inválidos, alterados o inexistentes evaluando si el *middleware* interceptor restringe correctamente el acceso devolviendo códigos `"401 Unauthorized"`.

Caso de Prueba	Entrada	Resultado Esperado
Acceso sin Token	Se omite la cabecera <i>Authorization</i> en la solicitud	Código: 401 (<i>Unauthorized</i>). Retorno: Mensaje de Acceso denegado.
Acceso con Token Inválido	Authorization: Bearer token_falso_123	Código: 401 (<i>Unauthorized</i>). Retorno: Mensaje de Token no válido.
Acceso con Token Expirado	El token asociado posee un tiempo de expiración excedente	Código: 401 (<i>Unauthorized</i>). Retorno: Mensaje el Token ha expirado.
Acceso con Token Válido	Autorization: Bearer [JWT_Vigente]	Código: 200 (<i>Ok</i>). Retorno: Mensaje de Acceso Concedido.

Tabla 4.3: Casos de Prueba respecto a la Gestión de sesiones JWT
Fuente: Elaboración propia

Al igual que en la validación anterior se consideran casos de prueba exitosos únicamente si cumple simultáneamente con las condiciones de código HTTP y mensaje de retorno esperados.

4.4 Validación de la Gestión de Archivos

La técnica aplicada corresponde a pruebas de ingesta *Multipart*, filtrado MIME y Sanitización I/O. El objetivo es verificar que la recepción de archivos externos mediante el middleware Multer no represente un riesgo para la seguridad ni cause colisiones en el sistema de archivos del servidor.

Se emulan envíos de datos binarios bajo la especificación multipart/form-data al *endpoint* "/api/upload-pdf". La verificación audita la inspección del tipo MIME en memoria (Magic Numbers), la restricción exclusiva a formato application/pdf y la generación de nombres únicos (uso de *timestamp*) antes de la persistencia física en disco.

Caso de Prueba	Entrada	Resultado Esperado
Carga de Archivo Valido	Archivo plan.pdf válido (MIME application/pdf)	Código: 200 (OK). Retorno: Mensaje de aceptación del archivo y el nombre generado.
Carga de Archivo incorrecto	Archivo de documento .txt	Código: 400 (Bad Request). Retorno: Mensaje de rechazo del archivo por el tipo MIME invalido.
Carga de Archivo Duplicado	Se repite el plan.pdf de la 1ra Prueba	Código: 200 (OK) Retorno: Mensaje de aceptación del archivo y el nombre generado.
Obtención de Archivo valido	En los parámetros del endpoint /api/upload-pdf/:id se escribe el id de un plan de emergencias valido	Código: 200 (OK). Retorno: Mensaje de éxito y el archivo pdf correspondiente.
Obtención de Archivo invalido	En los parámetros del endpoint /api/upload-pdf/:id se escribe el id de un plan de emergencias invalido	Código: 400 (Bad Request). Retorno: Mensaje de archivo no encontrado.

Tabla 4.4: Casos de Prueba para la Validación de Gestión de Archivos

Fuente: Elaboración Propia

Cabe esclarecer que el sistema no busca impedir duplicados, en cambio sí generar nombres únicos y descriptivos, como lo es en el caso de un archivo duplicado.

La aprobación de los casos de prueba como en los subcapítulos anteriores son tanto el código HTTP y retorno esperado.

4.5 Validación de integración con Aplicación Móvil

La técnica aplicada corresponde a pruebas de sincronización basada en eventos mediante Simulación de Cliente (*Mocking / Stubbing*). Para evitar la dependencia de la red externa en la nube de Firebase durante el ciclo de pruebas locales, se diseñó un módulo simulador denominado `mobileSimulator.js`

El archivo `movilSimulator.js` funciona como un cliente externo que se simula una conexión a Firebase Firestore, derivando de un caso real donde se registra un listener permanente sobre la colección `planes`, el cual es utilizado por el backend para enviar información operativa durante el despacho de unidades.

Este script actúa como un cliente emulado que se mantiene a la escucha de eventos de despacho gatillados por la API `"/api/dispatch-unit"`. La verificación audita que la llamada al Backend ejecute correctamente dos acciones en cadena: la validación y cambio de estado de disponibilidad de la unidad en MySQL y el retorno de información correspondiente.

Caso de Prueba	Entrada	Resultado esperado
despacho adjuntando un solo plan principal	El cuerpo JSON de entrada lleva solo un id plan hacia un carro disponible	Código: 200 (<i>OK</i>). Retorno: Mensaje de Éxito, JSON del plan de emergencia a enviar y JSON del carro marcado como no disponible.
Despacho adjuntando varios planes a la vez	El cuerpo JSON de entrada lleva varios id de planes hacia un carro disponible	Código: 200 (<i>OK</i>). Retorno: Mensaje de Éxito. JSON de planes de emergencia enviados y JSON del carro marcado como no disponible.
Despacho de carro no disponible	El cuerpo JSON de entrada debe de adjuntar un id de carro sin disponibilidad	Código: 400 (<i>Bad Request</i>). Retorno: Mensaje de rechazo de despacho.

Tabla 4.5: Casos de Prueba para la Integración con la Aplicación Móvil

Fuente: Elaboración Propia

La aprobación de los casos de prueba como en los subcapítulos anteriores son tanto el código HTTP y retorno esperado.

4.6 Criterios de Aprobación y Matriz de Riesgo del Software

Para determinar la aceptación técnica de la arquitectura Backend, se definió un modelo cuantitativo y cualitativo que relaciona el Índice de Confiabilidad Operativa (I_{co}) con el nivel de riesgo asociado al fallo de cada componente.

4.6.1 Cálculo del Índice de Confiabilidad Operativa (I_{co})

El desempeño general del servidor corresponde a la proporción porcentual de la suma de porcentajes de éxito de cada elemento estructural respecto sus casos de prueba, derivando en la siguiente ecuación:

$$I_{co} = \left(\frac{I_{co\ 1} + I_{co\ 2} + I_{co\ 3} + I_{co\ 4}}{4} \right)$$

Para entender la ecuación se identifican las siguientes dos variables:

- I_{co} : Índice de confiabilidad operativa general del sistema.
- $I_{co\ x}$: Índice de confiabilidad operativa relativa a un elemento estructural. Estas se enumeran:
 - $I_{co\ 1}$: APIs RESTful.
 - $I_{co\ 2}$: Seguridad (JWT).
 - $I_{co\ 3}$: Gestión de Archivos.
 - $I_{co\ 4}$: Integración Móvil.

Para promediar el Índice de confiabilidad operativa de cada elemento estructural se plantea la siguiente ecuación en base a sus casos de pruebas respectivos:

$$I_{co\ x} = \left(\frac{\text{Pruebas Exitosas}}{\text{Pruebas Totales}} \right) \times 100$$

4.6.2 Matriz de Riesgo y Tolerancia por Componente

Dado que las consecuencias de una falla varían según la naturaleza del servicio, se decidió fijar exigencias de aprobación diferenciadas de cada I_{co} por elemento estructural de acuerdo designando un nivel de criticidad, estas exigencias serán denominadas Tasa Mínima Exigida (TME). La tabla 4.6 clasifica los elementos estructurales según su impacto operativo como criticidad en la gestión de emergencias.

Elemento Estructural	Clasificación de Riesgo	Impacto Operativo de un Fallo	Tasa Mínima Exigida (TME)
APIs RESTful	Medio	Inconsistencia o corrupción de datos relacionales en MySQL.	$\geq 65\%$
Seguridad (JWT)	Crítico	Vulnerabilidad de acceso y exposición de datos de infraestructura sensible.	100% (Tolerancia Cero)
Gestión de Archivos	Medio	Indisponibilidad de adjuntos PDF, manteniendo operativos los datos resumidos.	$\geq 65\%$
Integración Móvil	Alto	Despacho inconsistente de unidades o retraso en alertas tácticas a terreno.	$\geq 85\%$

Tabla 4.6: Matriz de riesgo e impacto por elemento estructural
Fuente: Elaboración Propia

4.6.3 Veredicto final de Aceptación

Por último, declarar el cumplimiento de los siguientes porcentajes de los valores de Índice de confianza operativa para declarar el servidor backend como Idóneo, aprobado o en su defecto, rechazado. La tabla 4.7 contiene cada caso posible según los criterios de Índice de confianza operativa de cada elemento estructural respecto a las tasas mínimas exigidas declaradas (Tabla 4.6) y el Índice de aprobación global, el cual solo será aceptado si su tasa es mayor al 85%.

Rango I_{co} Global	Cumplimiento de I_{co1} a I_{co4} respecto sus TME	Veredicto Final
$I_{co} = 100\%$	Todos cumplen con 100%	APROBADO CON EXCELENCIA
$85\% \leq I_{co} \leq 99\%$	Todo cumplen	APROBADO OPERATIVO
$85\% \leq I_{co} \leq 99\%$	Alguno no cumple	RECHAZADO
$I_{co} < 85\%$	Todos cumplen	RECHAZADO
$I_{co} < 85\%$	Alguno no cumple	RECHAZADO

Tabla 4.7: Criterios de aceptación para el veredicto del software

Fuente: Elaboración Propia

4.7 Script de Ejemplo

El entorno de pruebas automatizadas se estructuró dentro del directorio `src/tests/` del repositorio Backend. Se utilizaron scripts en JavaScript ejecutados por el *framework* Jest en conjunto con la librería Supertest, emulando las peticiones de un cliente HTTP sobre la aplicación Express sin necesidad de desplegar el servidor en red.

Las dinámicas clave implementadas en los scripts comprenden:

- **Estructuración de *Payloads* de Control:** Definición de objetos sintéticos (`ejemploPlan`, `ejemploPlanIncorrecto`) para forzar escenarios válidos, incompletos y con errores de tipo.
- **Aserciones HTTP e Integridad:** Verificación estricta mediante la función `expect()` sobre `res.statusCode` (200, 201, 400, 404) y la presencia de propiedades en la respuesta JSON.
- **Persistencia Dinámica de Estado:** Captura en memoria de identificadores generados dinámicamente (`idPlanCreado = res.body.data.id_plan`) en los tests iniciales de creación, garantizando que las pruebas subsecuentes de edición (PATCH), lectura (GET) y borrado (DELETE) actúen sobre registros reales de MySQL.

Algunos ejemplos de lo mencionado en los scripts.

```
const ejemploPlan = {
  id_usuario: 1,
  id_tipo_edificacion: 1,
  nombre_lugar: "Edificio de Prueba Automatizada",
  direccion: "Avenida Test 123",
  link_pdf: "http://localhost:3000/uploads/test.pdf",
  id_estado: 1,
  latitud: -33.4569,
  longitud: -33.4569
};

const ejemploPlanIncorrecto = {
  id_usuario: 1,
  id_tipo_edificacion: 1,
  nombre_lugar: "Edificio de Prueba Automatizada",
  direccion: "Avenida Test 123",
  link_pdf: "http://localhost:3000/uploads/test.pdf",
  id_estado: 1,
  latitud: "asdsadasd",
  longitud: "asdsadsa"
};

const ejemploPlanEditado = {
  id_usuario: 1,
  id_tipo_edificacion: 1,
  nombre_lugar: "nuevo nombre",
  direccion: "Editado",
  link_pdf: "http://localhost:3000/uploads/test.pdf",
  id_estado: 1,
  latitud: -33.4569,
  longitud: -33.4569
};
```

Figura 4.1: *Payloads* de control en fragmento de Script de Validación de APIs RESTful
Fuente: Elaboración propia

```
// -----  
// 1. Crear Plan  
// -----  
test("POST /api/planes - Creación de plan exitoso", async () => {  
  
  const res = await request(app)  
    .post("/api/planes")  
    .send(ejemploPlan);  
  
  resultados.push({  
    Caso: "Creación correcta",  
    Endpoint: "POST /api/planes",  
    Esperado: 201,  
    Obtenido: res.statusCode,  
    Resultado: res.statusCode === 201 ? "✓ PASS" : "X FAIL",  
    Respuesta: [ res.body.data.id_plan, res.body.data.id_tipo_edificacion, res.body.data.nombre_lugar ]  
  });  
  
  expect(res.statusCode).toBe(201);  
  expect(res.body.success).toBe(true);  
  
  idPlanCreado = res.body.data.id_plan;  
});
```

Figura 4.2: Persistencia Dinámica de Estado y aserciones HTTP como integridad en
Fragmento de Script de validación de APIs RESTful
Fuente: Elaboración propia

```
const resultados = {};  
  
// Token válido generado para las pruebas  
const testPayload = {  
  id: 1,  
  email: "giovanni.alarcon@usm.cl",  
  rol: 1  
};  
  
const validToken = jwt.sign(  
  testPayload,  
  process.env.JWT_SECRET,  
  { expiresIn: "1h" }  
);  
  
// -----  
// 1. Acceder con token válido  
// -----  
test("GET /protected - Debe permitir acceso con token válido", async () => {  
  
  const res = await request(app)  
    .get("/api/protected")  
    .set("Authorization", `Bearer ${validToken}`);  
  
  expect(res.statusCode).toBe(200);  
  expect(res.body.message).toBe("Accediste a la ruta protegida :DD");  
  expect(res.body.user.email).toBe(testPayload.email);  
  
  resultados.casol = {  
    mensaje: res.body.message,  
    usuario: res.body.user  
  };  
});
```

Figura 4.3: Fragmento de Script de validación de Seguridad JWT
Fuente: Elaboración propia

```
// -----
// Mostrar resultados
// -----
afterAll(() => {
  console.log("\n===== RESULTADOS DE LAS PRUEBAS =====");

  console.table([
    {
      Caso: "1. Token válido",
      "Código HTTP": 200,
      Resultado: "Acceso permitido",
      Respuesta: resultados.caso1.mensaje
    },
    {
      Caso: "2. Sin token",
      "Código HTTP": 401,
      Resultado: "Acceso denegado",
      Respuesta: resultados.caso2.mensaje
    },
    {
      Caso: "3. Token inválido",
      "Código HTTP": 403,
      Resultado: "Acceso denegado",
      Respuesta: resultados.caso3.mensaje
    },
    {
      Caso: "4. Token expirado",
      "Código HTTP": 403,
      Resultado: "Acceso denegado",
      Respuesta: resultados.caso4.mensaje
    }
  ]);

  console.log("=====\\n");
});

afterAll(() => {
  console.log("\n===== RESULTADOS DE LAS PRUEBAS =====");

  console.table([
    {
      Caso: "1. Subida de PDF",
      Resultado: resultados.caso1?.mensaje,
      Archivo: resultados.caso1?.nombreGenerado
    },
    {
      Caso: "2. MIME inválido",
      Resultado: resultados.caso2?.mensaje,
      Archivo: "-"
    },
    {
      Caso: "3. Subida repetida",
      Resultado: resultados.caso3?.mensaje,
      Archivo: resultados.caso3?.nombreGenerado
    },
    {
      Caso: "4. Obtener PDF",
      Resultado: resultados.caso4?.mensaje,
      Archivo: resultados.caso4?.archivo
    },
    {
      Caso: "5. Archivo inexistente",
      Resultado: resultados.caso5?.mensaje,
      Archivo: "-"
    }
  ]);

  console.log("=====\\n");
});
```

Figura 4.4: Fragmento de Script con “console logs” para mostrar resultados
Fuente: Elaboración propia

```
// -----
// 3. Subida repetida
// -----
test("POST /api/upload-pdf - Subida repetida del mismo archivo", async () => {
  await request(app)
    .post("/api/upload-pdf")
    .attach("pdf", pdfDuplicadoPath, {
      contentType: "application/pdf",
    });

  const second = await request(app)
    .post("/api/upload-pdf")
    .attach("pdf", pdfDuplicadoPath, {
      contentType: "application/pdf",
    });

  expect(second.statusCode).toBe(201);
  expect(second.body.success).toBe(true);

  resultados.caso3 = {
    mensaje: second.body.message,
    nombreGenerado: second.body.data.link_pdf
  };
});

// -----
// Mock de Firebase
// -----
const mockGet = jest.fn();
const mockSet = jest.fn();
const mockDelete = jest.fn();

jest.unstable_mockModule("../config/firebase.js", () => ({
  db: {
    collection: jest.fn(() => ({
      doc: jest.fn(() => ({
        get: mockGet,
        set: mockSet,
        delete: mockDelete
      })))
    })))
}));

// Importar la app DESPUÉS del mock
const { default: app } = await import("../app.js");
```

Figura 4.5: Dos fragmentos de Script, Caso de archivo duplicado en la validación de gestión de archivos (izquierda) y mock de Firebase en la validación de integración móvil (derecha)

Fuente: Elaboración propia

4.8 Resultados de las Validaciones

4.8.1 Resultados de las pruebas de APIs RESTful

console.log

(index)	Caso	Endpoint	Esperado	Obtenido	Resultado	Respuesta
0	'Creación correcta'	'POST /api/planes'	201	201	'✓ PASS'	[186, 1, 'Edificio de Prueba Automatizada']
1	'Creación inválida'	'POST /api/planes'	400	400	'✓ PASS'	'Latitud inválida'
2	'Creación incompleta'	'POST /api/planes'	400	400	'✓ PASS'	'Faltan campos requeridos'
3	'Edición correcta'	'PATCH /api/planes/:id'	200	200	'✓ PASS'	[186, 1, 'nuevo nombre']
4	'Edición inválida'	'PATCH /api/planes/:id'	400	400	'✓ PASS'	'Latitud inválida'
5	'Edición incompleta'	'PATCH /api/planes/:id'	400	400	'✓ PASS'	'nombre_lugar no puede estar vacío'
6	'Consultar todos'	'GET /api/planes'	200	200	'✓ PASS'	[[Object], [Object], [Object], ... 19 more items]
7	'Consultar existente'	'GET /api/planes/:id'	200	200	'✓ PASS'	[186, 1, 'nuevo nombre']
8	'Consultar inexistente'	'GET /api/planes/:id'	404	404	'✓ PASS'	'Plan not found'
9	'Eliminar existente'	'DELETE /api/planes/:id'	200	200	'✓ PASS'	'Plan Eliminado Correctamente'
10	'Eliminar inexistente'	'DELETE /api/planes/:id'	404	404	'✓ PASS'	'Plan not found'

Figura 4.6: Resultados de las Pruebas de APIs RESTful pt1
Fuente: Elaboración propia

```
PASS src/tests/testPlans.test.js
Validación de Endpoints: Planes de Emergencia
  ✓ POST /api/planes - Creación de plan exitoso (227 ms)
  ✓ POST /api/planes - Creación con datos inválidos (10 ms)
  ✓ POST /api/planes - Creación incompleta (9 ms)
  ✓ PATCH /api/planes/:id - Edición correcta (55 ms)
  ✓ PATCH /api/planes/:id - Edición inválida (8 ms)
  ✓ PATCH /api/planes/:id - Edición incompleta (5 ms)
  ✓ GET /api/planes - Consultar todos (12 ms)
  ✓ GET /api/planes/:id - Consultar existente (37 ms)
  ✓ GET /api/planes/:id - Consultar inexistente (12 ms)
  ✓ DELETE /api/planes/:id - Eliminar existente (53 ms)
  ✓ DELETE /api/planes/:id - Eliminar inexistente (12 ms)

Test Suites: 1 passed, 1 total
Tests: 11 passed, 11 total
Snapshots: 0 total
Time: 2.668 s, estimated 3 s
Ran all test suites matching /src\\tests\\testPlans.test.js/i.
```

Figura 4.7: Resultados de las Pruebas de APIs RESTful pt2
Fuente: Elaboración propia

Los resultados obtenidos son consistentes con el estado actual de desarrollo del proyecto, se trabajó en algunos puertos que no estaban bien expuestos hasta poder satisfacer los casos de prueba. Todos respondieron según lo esperado. El sistema es capaz de exponer puertos de gestión de datos al cliente frontend para mantener conexión e integridad con la base de datos.

4.8.2 Resultados de las pruebas de seguridad

console.log

(index)	Caso	Código HTTP	Resultado	Respuesta
0	'1. Token válido'	200	'Acceso permitido'	'Accediste a la ruta protegida :DD'
1	'2. Sin token'	401	'Acceso denegado'	'Token requerido'
2	'3. Token inválido'	403	'Acceso denegado'	'Token inválido o expirado'
3	'4. Token expirado'	403	'Acceso denegado'	'Token inválido o expirado'

Figura 4.8: Resultados de las Pruebas automatizadas de Seguridad pt1
Fuente: Elaboración propia

```
PASS src/tests/testSecurity.test.js
Validación de Seguridad y Autenticación (JWT)
  ✓ GET /protected - Debe permitir acceso con token válido (50 ms)
  ✓ GET /protected - Debe denegar acceso si no se envía token (6 ms)
  ✓ GET /protected - Debe denegar acceso con token inválido (6 ms)
  ✓ GET /protected - Debe denegar acceso con token expirado (30 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        2.223 s, estimated 3 s
Ran all test suites matching /src\\tests\\testSecurity.test.js/i.
```

Figura 4.9: Resultados de las Pruebas automatizadas de Seguridad pt2
Fuente: Elaboración propia

Todas las pruebas realizadas respondieron con los códigos HTTP y mensajes esperados. Los resultados obtenidos demuestran que el backend valida correctamente la autenticidad y vigencia del Json Web Token (JWT) antes de permitir el acceso a los recursos protegidos. De esta forma, únicamente los usuarios autenticados y autorizados pueden acceder a las funcionalidades restringidas del sistema, evitando intentos de acceso mediante credenciales inválidas, alteradas o expiradas.

4.8.3 Resultados de las pruebas de Gestión de Archivos

console.log

(index)	Caso	Resultado	Archivo
0	'1. Subida de PDF'	'Archivo PDF subido correctamente.'	'archivo-2026-07-24_03-55.pdf'
1	'2. MIME inválido'	'Solo archivos PDF permitidos'	'-'
2	'3. Subida repetida'	'Archivo PDF subido correctamente.'	'archivo_duplicado-2026-07-24_03-55.pdf'
3	'4. Obtener PDF'	'Archivo PDF encontrado.'	'Plan 190 devuelto correctamente'
4	'5. Archivo inexistente'	'Plan de emergencia no encontrado.'	'-'

Figura 4.10: Resultados de las Pruebas Gestión de Archivos pt1
Fuente: Elaboración propia

```
PASS src/tests/files.test.js
Validación de Gestión de Archivos (Multer)
  ✓ POST /api/upload-pdf - Subida exitosa de archivo PDF (143 ms)
  ✓ POST /api/upload-pdf - Rechazo de archivo no permitido (11 ms)
  ✓ POST /api/upload-pdf - Subida repetida del mismo archivo (38 ms)
  ✓ GET /api/uploads/:id - Buscar archivo existente (161 ms)
  ✓ GET /api/uploads/99999 - Archivo no encontrado (7 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:  0 total
Time:        3.43 s
Ran all test suites matching /src\\tests\\files.test.js/i.
```

Figura 4.11: Resultados de las Pruebas Gestión de Archivos pt2
Fuente: Elaboración propia

Todas las pruebas realizadas respondieron con los códigos HTTP esperados. Los resultados demuestran que el sistema es capaz de almacenar, recuperar y gestionar correctamente archivos PDF, garantizando que el principal formato documental utilizado para los planes de emergencia pueda ser cargado y consultado de forma segura y consistente.

4.8.4 Resultados de las pruebas de integración a la Aplicación Móvil

```

console.log
Caso 1:
  at Object.<anonymous> (src/tests/mobileSimulator.test.js:161:17)
console.log
{
  "mensaje": "Despacho con planes principal y secundarios enviados correctamente",
  "planesEnviados": {
    "plan_principal": {
      "id_plan": 2,
      "nombre_lugar": "Mall Marina",
      "direccion": "14 Nte. 821, Av. Libertad 1348, Viña del Mar",
      "tipo_edificio": "Zona de Comercio",
      "elementos_criticos": [
        {
          "tipo_elemento": "Extintor",
          "ubicacion": "Piso 10 - pasillo sur",
          "observaciones": "Extintor necesita recarga",
          "latitud": -33.0088,
          "longitud": -71.5482
        },
        {
          "tipo_elemento": "Salida de emergencia",
          "ubicacion": "Puerta de emergencia oriente",
          "observaciones": "Bloqueada por cajas",
          "latitud": -33.0093,
          "longitud": -71.5479
        },
        {
          "tipo_elemento": "Sustancia química",
          "ubicacion": "Bodega subterránea",
          "observaciones": "Almacenamiento de solventes inflamables",
          "latitud": -33.009,
          "longitud": -71.5484
        }
      ],
      "latitud": -33.0091,
      "longitud": -71.5481,
      "fecha_creacion": "2023-11-20T03:00:00.000Z",
      "fecha_actualizacion": "2025-09-23T18:14:25.000Z"
    },
    "planes_secundarios": [],
    "estado": "enviado",
    "fecha_envio": "2026-07-24T07:58:41.540Z"
  },
  "carroActualizado": {
    "id_carro": 102,
    "patente": "HJQHE31",
    "id_usuario": 1,
    "id_estado": 4,
    "id_compania": 1,
    "codigo_auth": null,
    "descripcion": "Carro de Agua / Emergencias Generales"
  }
}

console.log
Caso 2:
  at Object.<anonymous> (src/tests/mobileSimulator.test.js:164:17)
console.log
{
  "mensaje": "Despacho con planes principal y secundarios enviados correctamente",
  "planesEnviados": {
    "plan_principal": {
      "id_plan": 7,
      "nombre_lugar": "Torre Coraceros",
      "direccion": "Av. Libertad 1405, Valparaíso, Viña del Mar, Valparaíso",
      "tipo_edificio": "Edificio",
      "elementos_criticos": [
        {
          "tipo_elemento": "Extintor",
          "ubicacion": "Recepción Piso 1",
          "observaciones": "Extintor PQS",
          "latitud": -33.0075,
          "longitud": -71.5489
        },
        {
          "tipo_elemento": "Equipo eléctrico",
          "ubicacion": "Sala de bombas - Subterráneo 2",
          "observaciones": "Tablero eléctrico principal",
          "latitud": -33.0078,
          "longitud": -71.5486
        },
        {
          "tipo_elemento": "Salida de emergencia",
          "ubicacion": "Escalera presurizada",
          "observaciones": "Puerta cortafuego operativa",
          "latitud": -33.007,
          "longitud": -71.5487
        }
      ],
      "latitud": -33.0077,
      "longitud": -71.5488,
      "fecha_creacion": "2025-11-10T01:38:27.000Z",
      "fecha_actualizacion": "2025-11-10T01:38:27.000Z"
    },
    "planes_secundarios": [
      {
        "id_plan": 4,
        "nombre_lugar": "Mall Marina Oriente",
        "direccion": "14 Nte. 821, Av. Libertad 1348, Viña del Mar, Valparaíso",
        "tipo_edificio": "Zona de Comercio",
        "elementos_criticos": [
          {
            "tipo_elemento": "Extintor",
            "ubicacion": "Piso 10 - pasillo sur",
            "observaciones": "Extintor necesita recarga",
            "latitud": -33.0088,
            "longitud": -71.5482
          },
          {
            "tipo_elemento": "Salida de emergencia",
            "ubicacion": "Puerta de emergencia oriente",
            "observaciones": "Bloqueada por cajas",
            "latitud": -33.0093,
            "longitud": -71.5479
          },
          {
            "tipo_elemento": "Sustancia química",
            "ubicacion": "Bodega subterránea",
            "observaciones": "Almacenamiento de solventes inflamables",
            "latitud": -33.009,
            "longitud": -71.5484
          }
        ],
        "latitud": -33.0091,
        "longitud": -71.5481,
        "fecha_creacion": "2023-11-20T03:00:00.000Z",
        "fecha_actualizacion": "2025-09-23T18:14:25.000Z"
      }
    ],
    "estado": "enviado",
    "fecha_envio": "2026-07-24T07:58:41.540Z"
  },
  "carroActualizado": {
    "id_carro": 102,
    "patente": "HJQHE31",
    "id_usuario": 1,
    "id_estado": 4,
    "id_compania": 1,
    "codigo_auth": null,
    "descripcion": "Carro de Agua / Emergencias Generales"
  }
}

console.log
Caso 3:
  at Object.<anonymous> (src/tests/mobileSimulator.test.js:167:17)
console.log
La unidad no se encuentra disponible para despacho

  at Object.<anonymous> (src/tests/mobileSimulator.test.js:168:17)

```

Figura 4.12: Resultados de las Pruebas de Integración con Aplicación Móvil
Fuente: Elaboración propia

Como se observa en la Figura 4.12 (*ajustar número de figura según corresponda*), la suite de pruebas automatizadas ejecutada sobre el archivo `src/tests/mobileSimulator.test.js` culminó de manera satisfactoria, registrando la condición de aprobación (PASS) en la totalidad de los escenarios evaluados.

La evidencia de ejecución confirma que el endpoint `/api/dispatch-unit` procesa correctamente el flujo de despacho, retornando el código de estado 200 OK y emitiendo la estructura JSON esperada hacia el módulo `mobileSimulator.js` ante vehículos disponibles (tanto en despachos de un único plan como de múltiples planes adjuntos). Asimismo, ante la condición de borde de intentar despachar una unidad en estado no disponible, el servidor aplicó correctamente la regla de negocio en la capa lógica, rechazando la solicitud con el código 400 Bad Request. Con estos resultados se valida que el Backend gestiona la orquestación de eventos y mantiene la consistencia de los datos de disponibilidad en MySQL.

4.8.5 Resultados generales

Tras la ejecución de las baterías de pruebas automatizadas sobre los cuatro pilares del Backend, se evidencia el valor práctico del enfoque de Aseguramiento de Calidad (QA) implementado. Más allá de actuar como un mecanismo pasivo de verificación, el proceso de pruebas funcionó como un ciclo de retroalimentación activa (*QA Feedback Loop*), permitiendo identificar y corregir fallas de diseño e implementación en etapas tempranas del desarrollo:

- **Refactorización en la Lógica de Despacho Móvil:** La simulación de eventos en *mobileSimulator.js* reveló que la validación de disponibilidad de los vehículos se delegaba al *Frontend*. Esto motivó la implementación de un esquema de Defensa en Profundidad, incorporando la regla de negocio en el controlador en *Node.js* para rechazar despachos no permitidos con el código 400 Bad Request.
- **Corrección en la Exposición de Endpoints RESTful (PATCH):** Durante la automatización con *Supertest* se detectaron errores de enrutamiento y puertos en los endpoints de edición (PATCH */api/planes/:id*), donde las solicitudes no alcanzaban el controlador correspondiente. El análisis de los tests permitió corregir la exposición de las rutas en *Express* y alinear la semántica de las respuestas JSON.
- **Optimización en la Obtención y Servido de Archivos (GET):** Al ejecutar las pruebas de recuperación de archivos PDF (GET */api/upload-pdf/:id*), se identificaron fallas en la resolución de rutas relativas en disco y manejo de excepciones ante archivos inexistentes. Se ajustó el *middleware* de almacenamiento (*Multer*) para asegurar el retorno correcto de los flujos binarios con código 200 OK y el manejo adecuado de respuestas 404 Not Found.

La resolución de estos tres frentes de trabajo transformó las pruebas automatizadas en una contribución directa a la solidez de la arquitectura Backend, garantizando que el servidor responda de forma consistente. A continuación, la Tabla 4.8 sintetiza los resultados de los Índices de Confiabilidad Operativa obtenidos por cada elemento estructural:

Módulo de Validación	Casos aprobados	Porcentaje (I_{co})
APIs RESTful (CRUD)	11 / 11	100%
Seguridad y Autenticación (JWT)	4 / 4	100%
Gestión de Archivos (Multer/PDF)	5 / 5	100%
Integración Móvil (Simulación)	3 / 3	100%

Tabla 4.8: Resultados de Índices de Confianza Operativa (I_{co}) por elemento estructural

Fuente: Elaboración propia

Por último, el cálculo del índice de confianza operativa global del sistema es el siguiente:

$$I_{co} = \left(\frac{I_{co\ 1} + I_{co\ 2} + I_{co\ 3} + I_{co\ 4}}{4} \right) = \left(\frac{100\% + 100\% + 100\% + 100\%}{4} \right) = 100\%$$

Al registrar un $I_{co} = 100\%$ y satisfacer la totalidad de los criterios de aceptación, el prototipo Backend obtiene el veredicto de "Aprobado con Excelencia". En el ámbito operativo del Cuerpo de Bomberos de Viña del Mar, este resultado garantiza la entrega de datos estructurales libres de corrupción en MySQL, un manejo seguro de adjuntos PDF y un tiempo de respuesta en milisegundos durante el despacho táctico.

4.9 Oportunidades de mejora

A pesar de la alta tasa de éxito alcanzada en el entorno de desarrollo, la estrategia de prueba implementada presenta oportunidades de mejora para futuras iteraciones del sistema:

- **Inclusión de Pruebas de Carga y Estrés:** Las pruebas actuales evalúan la correctitud funcional ante peticiones individuales. Para etapas futuras del proyecto se espera una demanda bajo un ambiente de producción, por ende se recomienda incorporar herramientas como *k6* o *Autocannon* para medir la degradación del servidor y la saturación del *Pool de Conexiones* en MySQL bajo escenarios de peticiones concurrentes masivas.
- **Evolución del Entorno de Simulación a Staging:** La sustitución del simulador local (*mobileSimulator.js*) por un entorno de pruebas (*Staging*) conectado de forma real a Firebase Firestore permitirá validar latencias de red Cloud e imponderables de conectividad.
- **Automatización en Pipeline CI/CD:** Integrar la ejecución automatizada de esta suite de Jest directamente en GitHub *Actions*, asegurando que cualquier cambio futuro en la rama principal (*main*) bloquee de forma automática despliegues que no superen el 100% de los *tests*.

4.10 Evaluación Cualitativa y Retroalimentación de Usuarios

Complementando la validación cuantitativa realizada mediante las pruebas automatizadas, se contó con una instancia de evaluación cualitativa a partir del recibimiento y la retroalimentación de los usuarios durante la Feria de Software 2025 de la universidad. Durante esta instancia, Fastplan fue expuesto como uno de los proyectos participantes, dando a conocer el ecosistema completo del sistema y su flujo de uso, especialmente en el contexto operativo del Cuerpo de Bomberos. El recibimiento general fue positivo, validando la propuesta de valor y el avance técnico de la solución. Al analizar la retroalimentación obtenida, se identificaron apreciaciones de dos grupos diferenciados: Usuarios Institucionales (Cuerpo de Bomberos) y Usuarios Generales.

4.10.1 Retroalimentación de Usuarios Institucionales (Bomberos)

Aunque en menor cantidad de usuarios, aquellos pertenecientes a la institución de bomberos ofrecieron una retroalimentación específica y crítica, enfocada en la precisión operativa y conceptual del sistema. Se destacan:

- Validación del potencial, Calificaron el proyecto como “Una muy buena iniciativa”, “útil” y prácticamente “necesario” enfocando la utilidad de la gestión digital con la entrega de información crítica en terreno.
- Detalles conceptuales y precisión técnica, aunque menores, se identificaron observaciones puntuales relacionadas con la nomenclatura y especificaciones de seguridad, principalmente en la representación de elementos químicos y la gestión de materiales peligrosos.

4.10.2 Retroalimentación de Usuarios Generales (Comunidad)

Los participantes que no conformaban parte del Cuerpos de Bomberos, aunque valoraron positivamente el sistema, enfocaron sus comentarios en sugerir nuevas implementaciones y ampliaciones del alcance más allá del objetivo principal de la tesina. Se destacan:

- Enfoque comunitario, las sugerencias apuntaron hacia funciones que fomentarían la participación ciudadana, la prevención comunitaria y la difusión de la información de emergencia a un público más amplio.
- Alcance futuro, estas ideas exceden las limitaciones del alcance definido para el desarrollo de esta tesina, pero demuestran un interés en la posible expansión de la plataforma hacia una herramienta de utilidad comunitaria.

5 Conclusiones

5.1 Cumplimiento de objetivos

El objetivo general del trabajo se cumplió exitosamente tras desarrollar la arquitectura Backend e integrarla con el ecosistema Fastplan, garantizando la disponibilidad, seguridad y consistencia de los planes de emergencia.

En cuanto a los objetivos específicos, se diseñó la arquitectura multicapa, se implementaron los servicios RESTful requeridos, se integró la autenticación delegada (OAuth 2.0/JWT) y se documentaron los endpoints del sistema. Respecto a la realización de pruebas unitarias e integrales, este objetivo se cumplió al alcanzar un Índice de Confiabilidad Operativa del 100% ($I_{CO} = 100\%$) en los cuatro pilares del servidor. La batería de pruebas no solo auditó el software, sino que actuó como un motor de refactorización activa (*Test-Driven Refactoring*), permitiendo corregir fallas en la lógica de negocio antes de la entrega final.

5.2 Limitaciones del Proyecto

El desarrollo de la plataforma se llevó a cabo considerando las siguientes restricciones técnicas y operativas:

- **Validación Institucional en Terreno:** La dificultad para establecer canales burocráticos oficiales restringió la prueba de campo con unidades reales en servicio, limitando la validación al entorno simulado y pruebas de usuario controladas.
- **Sostenibilidad y Despliegue en Producción:** Como prototipo funcional (MVP), la transición a producción requiere definir un modelo de financiamiento para solventar los costos de infraestructura Cloud y mantenimiento recurrente.
- **Automatización en la Ingesta de Datos:** La carga actual de archivos PDF depende de la entrada manual por un administrador, constituyendo un cuello de botella para la escalabilidad masiva.
- **Alcance de Usuarios:** Por requerimiento inicial del patrocinador, el alcance funcional del prototipo se restringió al Cuerpo de Bomberos de Viña del Mar, excluyendo temporalmente el acceso directo a residentes.

5.3 Trabajo Futuro

Para evolucionar la solución hacia un producto maduro y de alto impacto, se proponen las siguientes líneas de trabajo unificadas:

5.3.1 Evolución Funcional e Integración.

- **Módulo Comunitario y Resistencia Offline:** Desarrollar una interfaz pública ligera para residentes y dotar a la app móvil de persistencia *offline* (caching agresivo) para zonas de sombra de red.
- **Ingesta Inteligente con IA/OCR:** Implementar servicios de Inteligencia Artificial para extraer metadatos automáticamente de los PDF subidos.
- **Integración IoT e Interoperabilidad:** Conectar el Backend a sensores inteligentes en edificios y exponer APIs para coordinar la respuesta con Carabineros y SAMU.

5.3.2 Infraestructura, Seguridad y Operaciones (CI/CD)

- **Arquitectura Cloud y Pipeline CI/CD:** Migrar el Backend a microservicios en la nube con auto-escalado e integrar las pruebas de Jest en GitHub Actions para automatizar los despliegues.
- **Pruebas de Carga y Auditoría de Ciberseguridad:** Someter al servidor a pruebas de estrés (*k6*) y contratar auditorías de penetración (*pentesting*) externas antes del despliegue productivo con datos reales.

5.4 Conclusiones generales

El desarrollo de Fastplan representó un hito en mi formación profesional, permitiendo evolucionar desde la escritura de código hacia el diseño de arquitectura de software para entornos de alta disponibilidad. Resolver desafíos como la optimización del pool de conexiones, la seguridad delegada y la orquestación en tiempo real demuestra la capacidad técnica para transformar la respuesta ante emergencias de un modelo reactivo a uno proactivo sustentado en datos.

Concluyo esta etapa satisfecho por entregar como equipo una plataforma escalable y robusta, que no solo cumple con los requisitos académicos, sino que sienta las bases para una innovación real respecto a las tecnologías actuales de seguridad ante emergencias.

Referencias Bibliográficas

Fuentes Académicas y Teóricas:

- **Fielding, R. T. (2000).** *Architectural Styles and the Design of Network-based Software Architectures* (Tesis doctoral). University of California, Irvine.
- **Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020).** *Database System Concepts* (7ma ed.). McGraw-Hill Education.
- **Stallings, W. (2017).** *Cryptography and Network Security: Principles and Practice* (7ma ed.). Pearson.
- **Tanenbaum, A. S., & Wetherall, D. J. (2012).** *Redes de computadoras* (5ta ed.). Pearson Educación.
- **Hohpe, G., & Woolf, B. (2004).** *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley.
- **Sommerville, I. (2016).** *Software Engineering* (10ma ed.). Pearson Education.
- **Myers, G. J., Sandler, C., & Badgett, T. (2012).** *The Art of Software Testing* (3ra ed.). John Wiley & Sons.
- **Bass, L., Clements, P., & Kazman, R. (2022).** *Software Architecture in Practice* (4ta ed.). Addison-Wesley Professional.
- **Richards, M., & Ford, N. (2020).** *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media.
- **Tilkov, S., & Vinoski, S. (2010).** Node.js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, 14(6), 80-83.
- **Internet Engineering Task Force (IETF). (2017).** **RFC 8259:** *The JavaScript Object Notation (JSON) Data Interchange Format* (T. Bray, Ed.).

Estándares internacionales:

- **Internet Engineering Task Force (IETF). (2014).** **RFC 7231:** *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*.
- **Internet Engineering Task Force (IETF). (2015).** **RFC 7519:** *JSON Web Token (JWT)*.
- **Internet Engineering Task Force (IETF). (2012).** **RFC 6749:** *The OAuth 2.0 Authorization Framework*.

Documentación Técnica:

- **Google Developers. Cloud Firestore Documentation.**
<https://firebase.google.com/docs/firestore>
- **Microsoft Learn. Microsoft Identity Platform.**
<https://learn.microsoft.com/entra/identity-platform/>
- **Node.js Foundation. Node.js Documentation.**
<https://nodejs.org/docs/latest/api/>
- **Express.js. Express Documentation.** <https://expressjs.com/>
- **Oracle Corporation. MySQL 8.0 Reference Manual.**
<https://dev.mysql.com/doc/refman/8.0/en/>
- **OpenStreetMap Foundation. Nominatim Documentation.**
<https://nominatim.org/>
- **OpenJS Foundation. Node.js Event Loop Documentation.**
<https://nodejs.org/en/learn/asynchronous-work/event-loop-timers-and-nexttick>

Herramientas de Automatización (Capítulo 4):

- **Jest Framework. (2026). Jest Documentation:** JavaScript Testing Framework. Meta Open Source. <https://jestjs.io/>
- **Supertest. (2026). Supertest:** HTTP assertions made easy via superagent. npm package documentation. <https://www.npmjs.com/package/supertest>

Marco Legal y Normativo Chileno:

- **Biblioteca del Congreso Nacional de Chile (BCN). (2022). Ley N° 21.442:** Aprueba nueva Ley de Copropiedad Inmobiliaria. Diario Oficial de la República de Chile.
- **Biblioteca del Congreso Nacional de Chile (BCN). (1968). Ley N° 16.744:** Establece normas sobre accidentes del trabajo y enfermedades profesionales. Diario Oficial de la República de Chile.
- **Biblioteca del Congreso Nacional de Chile (BCN). (2021). Ley N° 21.364:** Establece el Sistema Nacional de Prevención y Respuesta ante Desastres (SENAPRED). Diario Oficial de la República de Chile.
- **Ministerio de Salud. (1999). Decreto Supremo N° 594:** Reglamento sobre condiciones sanitarias y ambientales básicas en los lugares de trabajo. Biblioteca del Congreso Nacional.