UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

Desarrollo back-end enfocado en agendamiento y provisión eficiente de servicios en tiempo real para proyecto MedMatch (atención médica a domicilio)

José Rolando Zapata Hidalgo

jose.zapatah@usm.cl

Profesor Guía: Dagoberto Cabrera Tapia
Profesor Correferente: Diego Cáceres Solís

Resumen: El presente proyecto se centra en el desarrollo del *back-end* de una aplicación móvil orientada a la gestión y provisión eficiente de servicios médicos a domicilio, en el marco del proyecto *MedMatch* impulsado por la empresa *Kinin*. El problema identificado es la falta de una plataforma integral que conecte de forma segura, rápida y confiable a pacientes con profesionales de la salud, considerando que gran parte de las atenciones actuales se gestionan mediante canales manuales o informales. La propuesta consiste en diseñar e implementar un *back-end* que permita gestionar usuarios, agendar citas y resguardar datos sensibles, integrando además un sistema de geolocalización GPS para favorecer la atención en tiempo real. El desarrollo prioriza la optimización de recursos para dispositivos móviles y la protección de la información. Esta tesina busca contribuir a la digitalización de la atención médica domiciliaria, mejorando la experiencia de pacientes y profesionales, potenciando la confianza en el servicio y abriendo nuevas oportunidades en el mercado de salud digital.

Palabras Clave: Agendamiento médico, Atención domiciliaria, *Back-end*, Geolocalización en tiempo real, Seguridad de datos.

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 back-end enfocado en agendamiento y provisión eficiente de servicios en tiempo real para proyecto MedMatch

Nombre del candidato(a): José Rolando Zapata Hidalgo

Carrera / Grado: Ingeniería en informatica

Campus: Viña del Mar **Departamento:** Electrotecnia e Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Dagoberto Cabrera Tapia, 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.

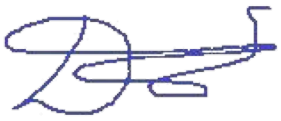
☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por **(marcar una opción)**:

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

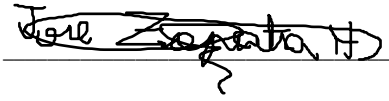
Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

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

Fecha: 24-02-2026 **Firma:** 

Estudiante o Candidato(a):

Fecha: 23/02/26 **Firma:** 

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

1 Introducción

1.1 Contexto y antecedentes

En Chile y en gran parte de Latinoamérica, el sector de la salud convive con tres fenómenos estructurales recurrentes:

1. Sobrecarga de la red asistencial pública y privada: Los centros de salud primaria y hospitales enfrentan listas de espera que superan frecuentemente las 4-6 horas para consultas no urgentes y meses para especialidades, situación agravada postpandemia (MINSAL, 2025; ProSalud, 2024).
2. Nuevas oportunidades y subempleo de profesionales de la salud recién titulados: entre el 18% y 25% de los médicos, kinesiólogos, enfermeros y otros profesionales titulados en los último cinco años no logran inserción laboral estable en su primer año, debido a la saturación institucional y procesos de contratación rígidos (MiFuturo, 2025).
3. Barreras de acceso a atención domiciliaria: Adultos mayores, personas postradas, pacientes de movilidad reducida y habitantes de zonas rurales o extremas enfrentan dificultades físicas, geográficas y económicas para recibir atención médica oportuna fuera de los recintos tradicionales.

En este escenario, la atención médica a domicilio aparece como una alternativa eficiente para descongestionar la red asistencial y mejorar la cobertura. Actualmente existen en Chile plataformas digitales que ofrecen este servicio (Docline, MeyDeey, Mydoc, Medy, Medico a domicilio de MetLife, entre otras), sin embargo, todas presentan limitaciones importantes en seguridad, formalidad, cobertura geográfica, verificación de credenciales y experiencia de usuario.

1.2 Definición del problema

A pesar de la existencia de plataformas de atención domiciliaria en Chile, ninguna logra resolver de manera integral y simultánea las siguientes brechas críticas:

- No existe verificación automática y confiable de la habilitación profesional mediante fuentes oficiales (Superintendencia de Salud).
- No ofrecen geolocalización ni atención inmediata en tiempo real basada en proximidad.
- Carecen de mecanismos robustos de seguridad y trazabilidad clínica que garanticen la validez legal de la atención prestada (historia clínica digital, receta electrónica).
- No generan un canal formal y visible de oportunidades laborales para los miles de profesionales de la salud cesantes o subempleados.
- Excluyen a segmentos importantes de la población (adultos mayores con brecha digital) al no contar con canales alternativos de agendamiento.

Es por esto por lo que la empresa *Kinin* identifica la falta una plataforma tecnológica segura, formal y en tiempo real que conecte eficientemente a pacientes que requieren atención médica domiciliaria con profesionales de la salud habilitados —especialmente aquellos en situación de cesantía—, generando al mismo tiempo una reducción efectiva de la sobrecarga en los centros de salud tradicionales y una mejora significativa en la equidad de acceso.

Este vacío no es solo técnico: representa una oportunidad perdida de mejorar la eficiencia del sistema de salud, reducir desigualdades y activar un recurso humano altamente capacitado que hoy permanece subutilizado.

1.3 Breve descripción sobre la propuesta de solución

Como respuesta a esta problemática, se propone el desarrollo de un sistema denominado proyecto *MedMatch*, orientado a la conexión rápida y confiable entre pacientes y profesionales de la salud a domicilio. La solución contempla el diseño de una aplicación móvil con un *back-end* que permita la gestión de usuarios, el agendamiento de citas y el tratamiento seguro de datos. Además, se integrará un sistema de localización *GPS* para mejorar la atención en tiempo real, optimizando la eficiencia del servicio y elevando la confianza de los usuarios en el proceso.

El principal objetivo que nuestro equipo quiere es el desarrollar una aplicación que permita a pacientes solicitar un profesional de la salud a domicilio. En cuanto a los específicos son los siguientes:

- Conectar a pacientes con profesionales de la salud mediante una aplicación móvil.
- Facilitar la solicitud de atención medica personalizada a domicilio
- Optimizar los tiempos de atención medica mediante agendamiento eficiente.
- Mejorar el acceso a servicios de salud para personas con dificultades de movilidad
- Generar nuevas oportunidades laborales para profesionales del área de la salud.

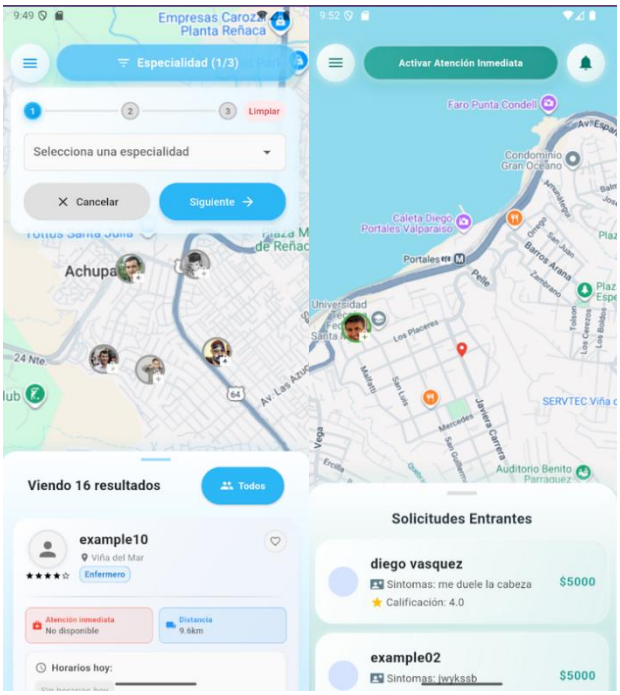


Ilustración 1 Vistas aplicación MedMatch
(Elaboración Propia)

1.4 Objetivos Generales y Específicos de la Tesina

Objetivo General

Desarrollar el *back-end* de una aplicación móvil que permita optimizar el agendamiento y provisión de servicios médicos a domicilio en tiempo real, garantizando seguridad, eficiencia y confiabilidad en la conexión entre pacientes y profesionales de la salud.

Objetivos Específicos

- Diseñar la arquitectura necesaria para el *back-end* de la aplicación.
- Diseñar e implementar la lógica del negocio orientada a la gestión de usuarios y agendamiento de citas.
- Incorporar mecanismos de seguridad que aseguren la protección de datos sensibles de pacientes y profesionales.
- Generar una API que permita manejar la información hacia y desde la base de datos.
- Integrar un sistema de geolocalización GPS para permitir el seguimiento de las atenciones en tiempo real.
- Optimizar el uso de datos y recursos del sistema, considerando las limitaciones propias de dispositivos móviles.
- Evaluar la usabilidad y funcionalidad del sistema en escenarios de atención médica domiciliaria.

1.5 Justificación del proyecto

El desarrollo de esta solución se justifica en tres dimensiones principales:

- **Social**: Frente a la alta demanda en los centros de salud y la dificultad que enfrentan muchas personas para acceder a atención domiciliara por factores geográficos, motrices o económicos, surge la necesidad de una plataforma que facilite el acceso a servicios médicos de forma equitativa, rápida y confiable. Nuestra aplicación busca mejorar la calidad de vida de los pacientes y, al mismo tiempo, generar oportunidades para los profesionales de la salud.

- **Tecnológica**: A pesar de la existencia de aplicaciones similares, muchas de ellas presentan limitaciones en seguridad, formalidad o capacidad de adaptación a distintos escenarios. Este proyecto propone una solución moderna y escalable que incorpore servicios en tiempo real, geolocalización y una gestión segura de datos, contribuyendo a la digitalización de procesos de atención y superando el uso de canales poco formales.

- **Económica y profesional**: El sistema abre nuevas oportunidades laborales para profesionales de la salud, en especial aquellos recién egresados que enfrentan dificultades de inserción laboral. Al ofrecer un canal validado y confiable para prestar sus servicios, se amplía su visibilidad profesional, se dinamiza el mercado de atención domiciliaria y se impulsa un sector con creciente demanda.

En conjunto, nuestro proyecto *MedMatch* busca no solo resolver las limitaciones actuales en la provisión de servicios de salud a domicilio, sino también establecer un modelo innovador que potencie la experiencia del paciente, incremente la confianza en el proceso y promueva la modernización del sector sanitario.

Cabe señalar que, si bien el presente proyecto se centra en una aplicación móvil, se reconoce la brecha digital en adultos mayores que prefieren canales no tecnológicos. Como trabajo futuro, la solución podría escalar hacia un sistema multicanal incorporando, por ejemplo, un agendamiento asistido mediante llamada telefónica, el cual se integraría con el sistema desarrollado.

1.6 Metodología Aplicada

La metodología aplicada durante el transcurso de este proyecto fue la metodología ágil Scrum, la cual facilita el desarrollo de aplicaciones móviles iterativas, además de ser la metodología en la cual el grupo de trabajo tenía mayor conocimiento, permitiendo demostrar avances durante cada iteración, el cual fue separado en 3 Sprint y uno adicional, cada uno con diferentes procesos.

- ***Front-end* y funcionalidad básica** (*Sprint* 1): Este *sprint* se enfoco en el aspecto estético y funcionalidad básica, como es una aplicación enfocada al público general se busca generar una aplicación con una experiencia de usuario que sea fácil de usar y de aprender, durante este Sprint se

realizará el *front-end* y la funcionalidad de pedir agendamiento simple (Seleccionar doctor, ver horarios disponibles, pedir cita en un horario disponible, marcar al doctor como ocupado durante esa hora).

- **Atención inmediata y vista de mapa** (*Sprint* 2): En este *sprint* se enfoco en las funcionalidades en las cuales se presenta la mayor cantidad del valor agregado del proyecto, en este caso, la atención inmediata de profesionales de la salud, vistas de los doctores en el mapa, la funcionalidad de pedir medico de manera inmediata (Permitir al médico elegir su área de cobertura y lugar, tras el medico coloque su marca, esta se verá en el mapa, permitiendo al usuario seleccionar a el medico en el mapa para poder pedir una cita).

- **Funcionalidades GPS** (*Sprint* 3): en este *sprint* se desarrollaron las funcionalidades en tiempo real, enfocándonos en el cálculo de rutas y tiempo que tomaría llegar a ellas, localización GPS a tiempo real, tratamiento de datos de localización para mejorar la seguridad y maneras de ocupar la localización del usuario.

- **Toques finales** (Adicional): Al finalizar los 3 *sprints* anteriores se busco arreglar posibles fallos que se encuentren presentes y mejorar el aspecto estético. Principalmente hacer que la aplicación sea más responsiva para distintos tipos de dispositivos y tamaños de pantalla.

2 Marco teórico

2.1 Back-end

El *back-end* es la parte oculta de una aplicación o sistema, todo lo que ocurre "detrás de escena" y que el usuario no ve directamente. Es el conjunto de procesos, servidores y bases de datos que hacen posible que la aplicación funcione correctamente. Mientras que el *front-end* es lo que el usuario ve e interactúa (pantallas, botones, menús), el back-end es el responsable de:

- Procesar la lógica de negocio.
- Guardar y recuperar datos en la base de datos.
- Controlar la seguridad y la autenticación de usuarios.
- Comunicarse con otros servicios o APIs externas.
- Enviar respuestas al *front-end*.

Los **componentes principales de un back-end** son:

- Servidor: lugar donde se corre el back-end.
- Lógica de negocio: reglas y procesos que controlan cómo funciona la aplicación.
- Base de datos: almacenamiento de información
- APIs: puntos de acceso que permiten que *front-end* y otras aplicaciones se comuniquen con el *back-end*

2.2 Sistema de agendamiento

Los sistemas de agendamiento es una plataforma digital diseñado para la programación y gestión de citas, reservas o programación de eventos, comúnmente usados en entornos comerciales, médicos, educativos o administrativos. Estos sistemas permiten reservar, modificar y cancelar citas de manera automatizada reemplazando así a los métodos manuales tradicionalmente usados. Proporcionando así más tiempo a los negocios para las demás tareas.

En el mercado existen varias soluciones ampliamente conocidas. Cada solución presenta un enfoque diferente, por ejemplo, *Holded*, este es un software de gestión empresarial con funciones de agendamiento integradas, pero que puede ser demasiado complejas para pequeñas empresas, en cuanto a *Kumbio*, es un enfoque más simple, limitado solamente a agendamiento sin mayor personalización o escalabilidad.

2.3 Servicios en tiempo real

Los servicios en tiempo real son llamadas aquellos servicios que transmite, procesa y se responde información de manera inmediata o de latencia mínima, de manera que el usuario recibe la información de manera inmediata.

Características:

- Baja latencia: el tiempo en el que ocurre la acción es casi inmediato o de manera imperceptible
- Actualización inmediata: Si se detecta algún cambio, inmediatamente se transmite este cambio al usuario
- interacción continua: el usuario no necesita actualizar o realizar alguna acción para que se le transmita información.

2.4 Tecnologías

2.4.1 Flutter

Flutter es un framework de desarrollo de software de código abierto creado por Google que permite construir aplicaciones nativas para múltiples plataformas (Android, iOS, web y escritorio) utilizando un único código base.

Principales características:

- Esta desarrollado en Dart, un lenguaje de programación también desarrollado por Google.
- Su principal ventaja es el desarrollo multiplataforma: con un solo código se pueden generar aplicaciones para diferentes sistemas operativos.
- Ofrece un alto rendimiento porque compila código nativo, a diferencia de frameworks híbridos más lentos.
- Tiene una amplia librería de widgets personalizables, lo que permite interfaces modernas y consistentes.

2.4.2 Firebase

Firebase es una plataforma de desarrollo de aplicaciones móviles y web también creada por Google. Está basada en servicios en la nube y proporciona múltiples herramientas para acelerar el desarrollo y la operación de aplicaciones.

Principales características:

- Autenticación de usuarios ya sea mediante email, teléfono, anónimo o servicios de terceros.
- Base de datos en tiempo real y *Firestore*, que permiten sincronizar datos entre clientes y servidor instantáneamente.
- Hosting y almacenamiento en la nube mediante *Cloud Storage*
- Notificaciones *push* y servicios de mensajería
- *Analytics* y monitoreo de rendimiento.

Se utiliza mucho junto con *Flutter*, ya que ambos son de Google cuentan con integración directa.

2.4.3 Firestore

Firestore es una base de datos NoSQL en la nube perteneciente al ecosistema Firebase de Google. Está optimizada para aplicaciones móviles que requieren sincronización en tiempo real, tolerancia a fallos y escalabilidad automática.

Características principales:

- Modelo de datos flexible basado en colecciones y documentos.
- Sincronización en tiempo real, permitiendo que los clientes reciban cambios inmediatamente.

- Auto escalado sin necesidad de administrar servidores.
- Integración directa con Flutter y APIs REST.
- Sistema de reglas de seguridad que permite gestionar accesos por roles y condiciones.
- Rápida respuesta, ideal para operar en escenarios como seguimiento móvil y actualización de disponibilidad de profesionales.

2.4.4 FastAPI

FastAPI es un *framework* moderno para el desarrollo de *APIs* en Python, diseñado para ofrecer alto rendimiento, facilidad de uso y validación estricta de datos. Está basado en el estándar ASGI (*Asynchronous Server Gateway Interface*), lo que le permite soportar operaciones asíncronas y comunicación en tiempo real.

Características principales:

- Alto rendimiento: comparable a *frameworks* en Node.js o Go, gracias al uso de async/await.
- Validación automática de datos mediante Pydantic.
- Documentación automática siguiendo *OpenAPI/Swagger*.
- Soporte nativo para *WebSockets*, indispensable para servicios en tiempo real.
- Esquema modular y escalable, orientado a microservicios.

2.4.5 Pydantic

Pydantic es una librería de Python que permite definir modelos de datos con validación estricta y conversión automática de tipos. FastAPI usa Pydantic como base para definir esquemas de entrada y salida.

Funciones clave:

- Validación automática: verifica tipos, formatos y campos obligatorios.
- Conversión de tipos (por ejemplo, *strings* a fechas o enteros).
- Esquemas JSON claros y consistentes, ideal para APIs REST.
- Detección temprana de errores, reduciendo fallos en producción.
- Integración directa con Firestore al transformar documentos en objetos tipados.

Pydantic es fundamental para garantizar que la información que circula entre cliente, servidor y base de datos sea correcta y segura.

2.4.6 WebSockets

Los *WebSockets* son un protocolo de comunicación bidireccional que permite enviar y recibir información en tiempo real sin la necesidad de hacer peticiones HTTP repetidas.

Características Principales:

- **Conexión persistente** entre cliente y servidor.
- **Latencia extremadamente baja**.
- **Intercambio continuo de mensajes** sin recargar la aplicación.
- **Soporte directo en *FastAPI*** mediante *WebSocketRoute*.
- Permite construir funcionalidades como:
    - seguimiento GPS en tiempo real,
    - cambios instantáneos de disponibilidad de profesionales,
    - notificaciones inmediatas sobre llegada de un médico,
    - estados de cita (aceptada, en camino, completada).

3 Diseño

Este capítulo describe el proceso de diseño e implementación del sistema MedMatch, abordando la arquitectura general, los componentes que conforman el *back-end*, el modelo de datos utilizado, la lógica del negocio y las integraciones necesarias para ofrecer servicios de atención inmediata domiciliara en tiempo real. El desarrollo se basó en las tecnologías previamente descritas en el marco teórico — *FastAPI, Firestore, WebSockets y Pydantic*— cuyo uso permitió construir una solución modular, escalable y capaz de operar con sincronización inmediata.

El diseño se centró en crear un sistema robusto que soportara tres ejes fundamentales:

1. La gestión de usuarios y perfiles de profesionales
2. El sistema de agendamiento tradicional e inmediato
3. El seguimiento en tiempo real de la ubicación y estados de una atención.
4. Comunicación sencilla y clara entre profesional y paciente.

A continuación, se detalla el proceso de diseño y las decisiones técnicas implementadas para cumplir con los objetivos planteados.

3.1 Arquitectura del sistema

La arquitectura del sistema se concibió bajo un modelo Cliente-Servidor centralizado con servicios en la nube, diseñado para garantizar la interoperabilidad, la escalabilidad y la consistencia de datos en un entorno de salud digital.

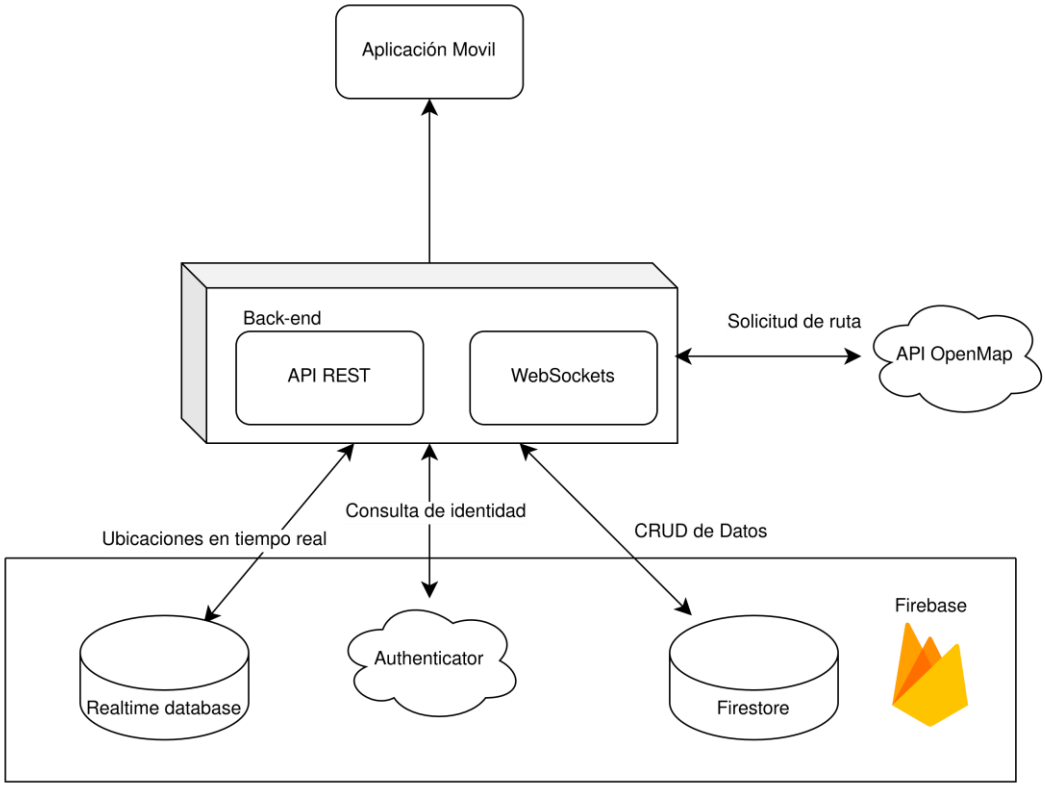


Figura 1 Arquitectura de MedMatch (Elaboración Propia)

El sistema se compone de tres bloques fundamentales que interactúan entre sí:

1. **Cliente Móvil (Front-end):** Actúa como la capa de presentación visual. Su responsabilidad se limita a la captura de datos y visualización de información, delegando el procesamiento pesado al servidor.

2. **Servidor de Aplicaciones (Back-end API):** Desarrollado en Python con FastAPI, actúa como el núcleo lógico del sistema. Funciona como una pasarela segura que centraliza las reglas de negocio, valida la integridad de las solicitudes provenientes del cliente y orquesta la comunicación con los servicios de persistencia. Además, integra servicios externos como OpenRouteService para el cálculo de rutas y tiempos de desplazamiento.

3. **Infraestructura de Datos y Autenticación (BaaS):** Se utiliza el ecosistema de Google Firebase. Firestore opera como la base de datos documental para registros estructurados (pacientes, doctores, historial), mientras que Realtime Database gestiona datos de alta frecuencia y baja latencia, específicamente para la geolocalización de profesionales. La seguridad de acceso se delega al servicio de Firebase Authentication.

Esta separación física y lógica permite que el *back-end* escale independientemente de la aplicación móvil y que la base de datos pueda absorber picos de carga sin afectar la disponibilidad del servicio de API.

3.2 Diseño del *Back-end*

Internamente, el *back-end* se estructuró siguiendo una Arquitectura de Capas (*Layered Architecture*) estricta, implementando el principio de Separación de Responsabilidades (SoC). Este diseño divide el código en capas lógicas con responsabilidades únicas, evitando el acoplamiento y facilitando las pruebas unitarias y el mantenimiento evolutivo.

Los componentes internos se organizan de la siguiente manera:

- **Capa de Presentación (*Routers & Controllers*)**: Es el punto de entrada de la API. Se encarga de la definición de *endpoints* (Rutas) y la gestión del protocolo *HTTP/WebSocket*. Su función principal es la validación de entrada y salida utilizando esquemas *Pydantic (DTOs)*, asegurando que los datos cumplan con los formatos requeridos antes de invocar cualquier lógica. No contiene reglas de negocio.

- **Capa de Servicios (*Service Layer*)**: Constituye el núcleo del sistema. Aquí residen las reglas de negocio y la orquestación de procesos. Los servicios son responsables de coordinar tareas complejas, tales como:
  - Verificar la validez temporal de una atención inmediata.
  - Sincronizar actualizaciones de estado entre las colecciones de doctores y pacientes.
  - Gestionar la lógica asíncrona para no bloquear el hilo principal de ejecución, utilizando *threads* para operaciones de entrada/salidas pesadas.

- **Capa de Acceso a Datos (DAO / Utilidades)**: Implementa el patrón *Data Access Object*. Aísla completamente la lógica de negocio de los detalles técnicos de la base de datos. Consiste en funciones utilitarias que ejecutan las operaciones CRUD (Crear, Leer, Actualizar, Borrar) directas contra el SDK de *Firebase*. Esta capa es agnóstica al negocio; solo sabe cómo guardar o recuperar datos.

- **Integración de Componentes Transversales:** El sistema implementa componentes transversales para funcionalidades críticas:
  - Gestor de *WebSockets*: Un componente dedicado (*ConnectionManager*) que administra el ciclo de vida de las conexiones persistentes para el chat, garantizando la entrega de mensajes en tiempo real (broadcast) exclusivamente a los participantes de una sala.

- Inyección de Dependencias: Se utiliza el sistema de inyección de *FastAPI* para gestionar la seguridad, verificando los tokens de autenticación en cada petición antes de que esta llegue a la capa de servicios.

3.3 Diseño Funcional del Sistema

El sistema se ha diseñado para facilitar la conexión entre profesionales de la salud y pacientes mediante dos modalidades de atención: inmediata (basada en geolocalización en tiempo real) y agendada (basada en disponibilidad futura).

3.3.1 Actores del Sistema

Se identifican dos roles principales que interactúan con la plataforma:

- **Doctor (Profesional):** Usuario proveedor del servicio de salud. Sus funciones principales son gestionar su disponibilidad (inmediata o futura) y atender las solicitudes de los pacientes.
- **Paciente:** Usuario consumidor del servicio. Sus funciones principales son la búsqueda de profesionales geolocalizados y la solicitud de consultas médicas.

3.3.2 Módulos Funcionales

Para organizar la lógica del sistema, las funcionalidades se agrupan en cuatro módulos principales:

- **Módulo de Autenticación y Perfil**: Encargado de la gestión de identidades y acceso seguro al sistema.
  - **Funciones:** Inicio de sesión (*Login*), validación de credenciales mediante *Firebase Auth* y gestión de datos de perfil.
- **Módulo de Atención Inmediata (Real-time)**: Es el núcleo diferenciador de la aplicación. Gestiona la disponibilidad en tiempo real y la geolocalización.
  - **Doctor:** Puede activar una "Marca de Disponibilidad" en el mapa. Al hacerlo, define su ubicación actual, un radio de cobertura (km) y un tiempo de vigencia (duración).
  - **Paciente:** Puede visualizar en un mapa interactivo a los doctores activos cercanos y solicitar una atención inmediata.
  - **Interacción:** Incluye el flujo de solicitud, aceptación, rechazo o cancelación de la consulta inmediata.
- **Módulo de Agendamiento (Citas Futuras)**: Gestiona la reserva de horas para consultas no urgentes.
  - **Doctor:** Define bloques horarios disponibles en su agenda.
  - **Paciente:** Visualiza la agenda del doctor y reserva una cita futura.
- **Módulo de Comunicación (Chat)**: Proporciona un canal de comunicación bidireccional y seguro.
  - **Funciones:** Chat 1 a 1 en tiempo real entre Doctor y Paciente, activado una vez que se confirma una solicitud de atención.

A continuación, se muestra un gráfico con los casos de uso de los 2 actores, además de gráficos de secuencia en los cuales se muestran los principales flujos de la aplicación:

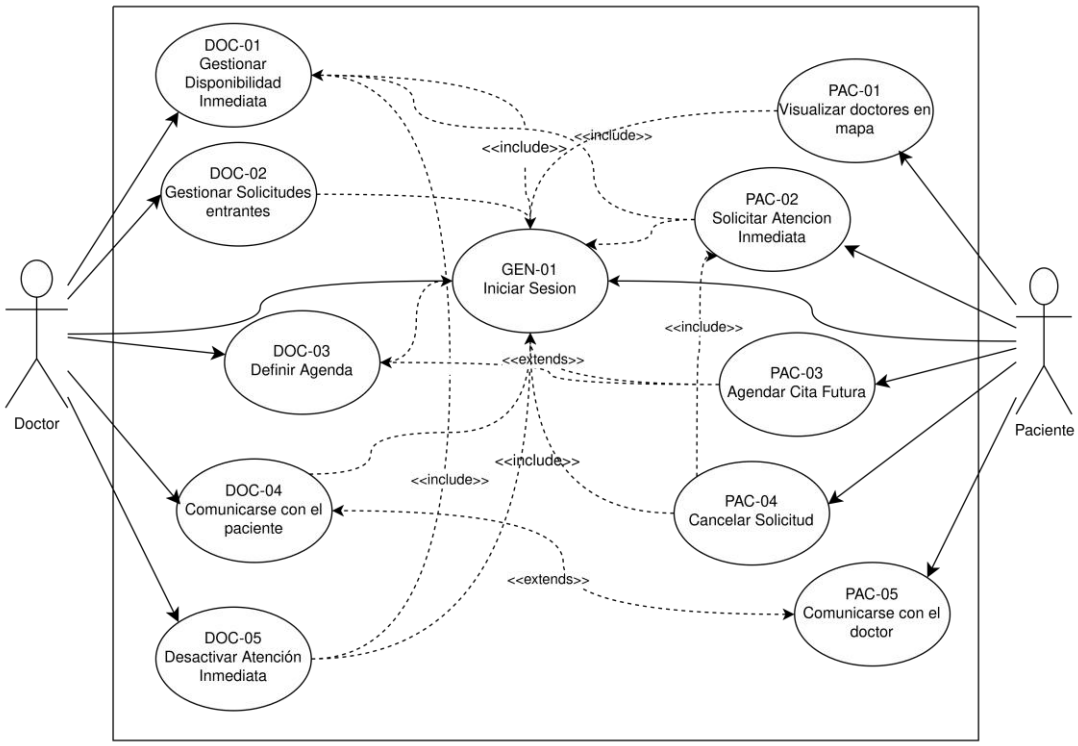


Figura 2 Diagrama de casos de uso (Elaboración Propia)

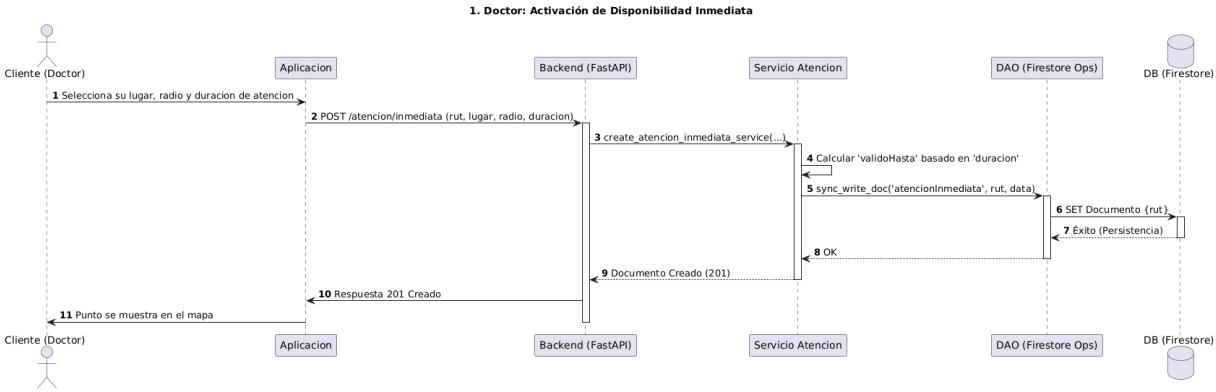


Figura 4 Flujo activación de disponibilidad inmediata (Elaboración Propia)

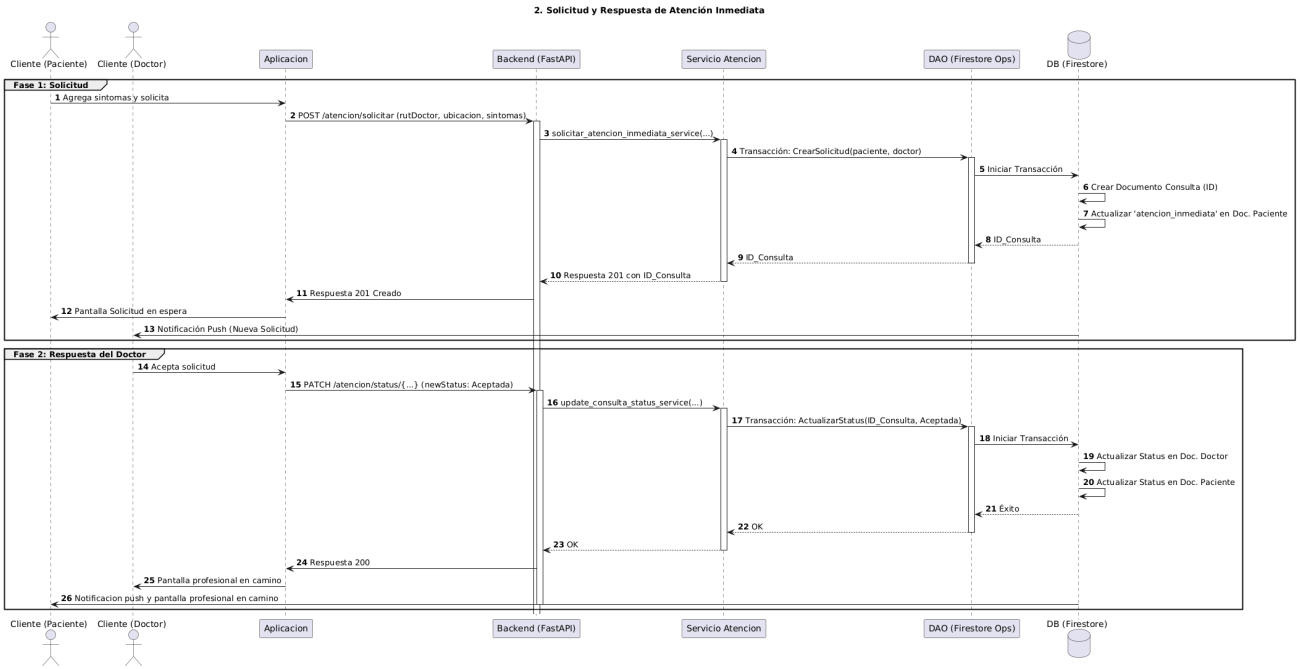


Figura 3 Flujo solicitud y respuesta de atención inmediata (Elaboración Propia)

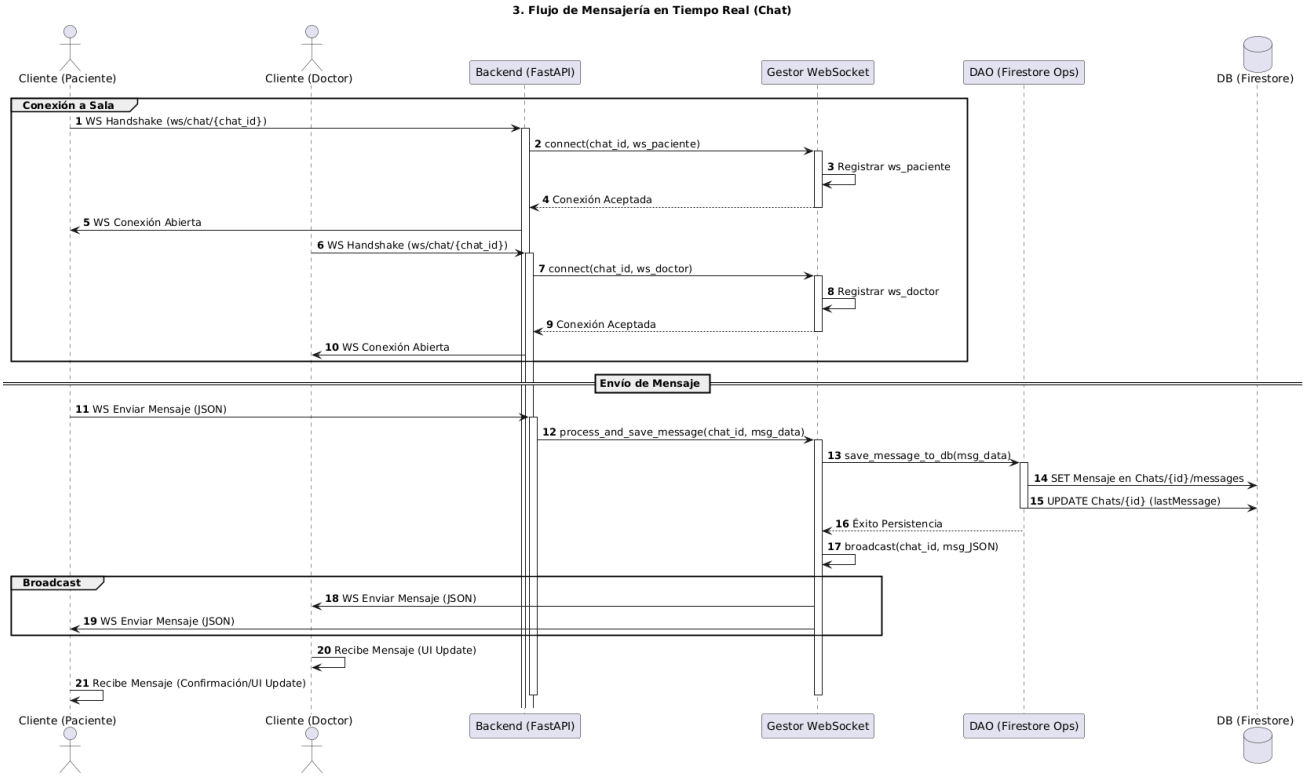


Figura 6 Flujo mensajería en tiempo real (Elaboración Propia)

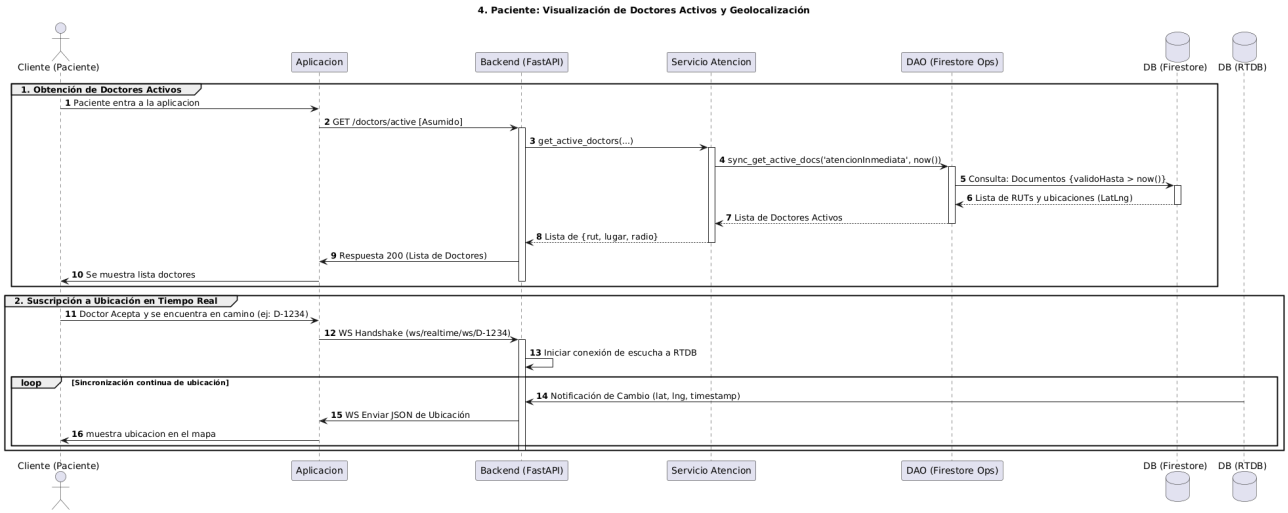


Figura 7 Flujo de visualización de Doctores activos y geolocalización (Elaboración Propia)

3.4 Modelo y estructuras de datos

Los modelos de los datos del sistema fueron basados de acuerdo con la base de datos *Firestore* y se definieron en el *back-end* mediante el uso de *Pydantic*, el cual permite la validación de las variables. *Firestore* está definido como una base de datos NoSQL el cual guarda los datos en colecciones y permite que los documentos tengan sub-colecciones. Además, presenta la opción de agregar referencias a otras colecciones o documentos, permitiendo acelerar y evitar usar consultas adicionales.

3.4.1 Modelos *Patient*

Para los pacientes se generaron dos colecciones principales, *Patient* el cual contiene toda la información del paciente más dos sub-colecciones adicionales que guardan sus atenciones en tiempo real y otra enfocada en la atención a futuro; y *favourites* el cual se encarga de guardar los doctores favoritos de los usuarios.

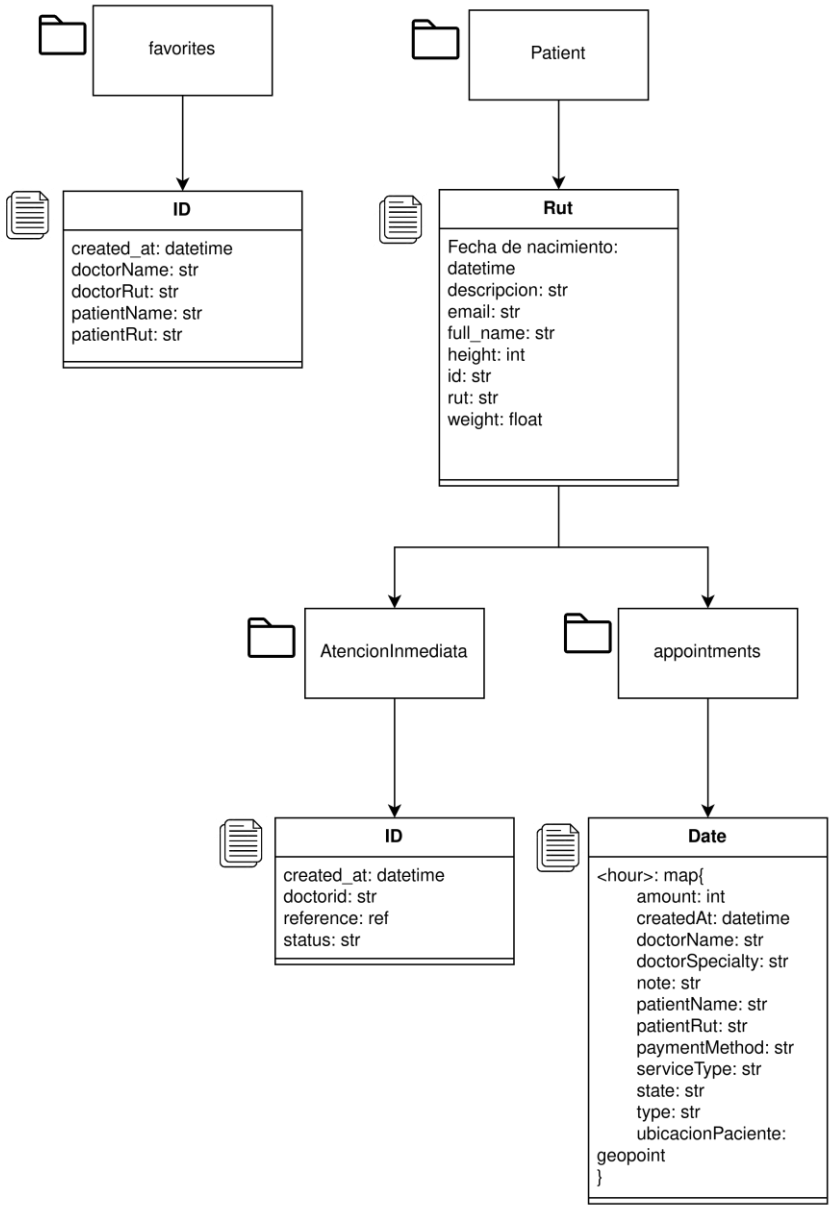


Figura 8 Estructura de BD *Patients* (Elaboración Propia)

Descripción de las colecciones y sus datos:

- Patient [/Patient/{Rut del usuario}]

Nombre de variable	Tipo	Descripción
Fecha de nacimiento	DATETIME	Se indica la fecha de nacimiento del paciente en formato *DateTime.*
Descripción	STR	Es una breve introducción del paciente.
Email	STR	Email Registrado a la cuenta
Full_name	STR	Nombre completo del paciente
Height	INT	Altura del paciente (CM)
Id	STR	Identificador otorgado al paciente proporcionado por *Firebase Authenticator* al momento de crear la cuenta.
Rut	STR	Rut del paciente
Weight	INT	Peso del paciente (Kg)

Tabla 1 Estructura de *Patient* (Elaboración Propia)

- Atención Inmediata [/Patient/{rut del usuario}/AtencionInmediata/{id}]

Nombre de variable	Tipo	Descripción
Created_at	DATETIME	Timestamp de cuando fue generado el documento
doctorId	STR	identificación del doctor
Reference	REF	Referencia a la atención inmediata del doctor
status	STR	Estado actual de la solicitud

Tabla 2 Estructura atención inmediata Paciente (Elaboración Propia)

- Appointments [/Patient/{Rut del usuario}/appointments/{id}]

Nombre de variable	Tipo	Descripción
Hora	MAP	MAP que contiene información de la hora tomada por un doctor, contiene los siguientes datos
Amount	INT	Valor de la consulta en CLP
Created_at	DATETIME	Timestamp de cuando fue generado el documento
doctorName	STR	Nombre del doctor
doctorSpecialty	STR	Especialidad del doctor
Note	STR	descripción adicional de la consulta
patientName	STR	Nombre del paciente
patientRut	STR	Rut del paciente
PaymentMethod	STR	Método de pago elegido por el paciente
ServiceType	STR	Tipo de servicio que se realizó o se realizara (Domicilio o videollamada)
State	STR	Estado de la consulta
Type	STR	Tipo de consulta a realizar
UbicacionPaciente	GEOPOINT	Coordenadas de la ubicación del paciente

Tabla 3 Estructura appointments del paciente (Elaboración Propia)

- Favoritos [/favourites/{id autogenerado}]

Nombre de variable	Tipo	Descripción
Created_at	DATETIME	*Timestamp* de cuando fue generado el documento
doctorName	STR	Nombre completo del profesional
doctorRut	STR	Rut del profesional
PatientName	STR	Nombre completo del paciente
PatientRut	STR	Rut del paciente

Tabla 4 Estructura Favoritos (Elaboración Propia)

3.4.2 Modelos Doctor

Para los doctores de una única colección que contiene la información del doctor, además presenta 3 sub-colecciones, *AtencionInmediata* el cual contiene las atenciones que se realizan en tiempo real, La agenda el cual muestra la agenda completa del profesional, donde solo el profesional puede ver esta colección y horarios el cual muestra una muestra de manera resumida la agenda, esta es usada principalmente para que el usuario vea los horarios disponibles.

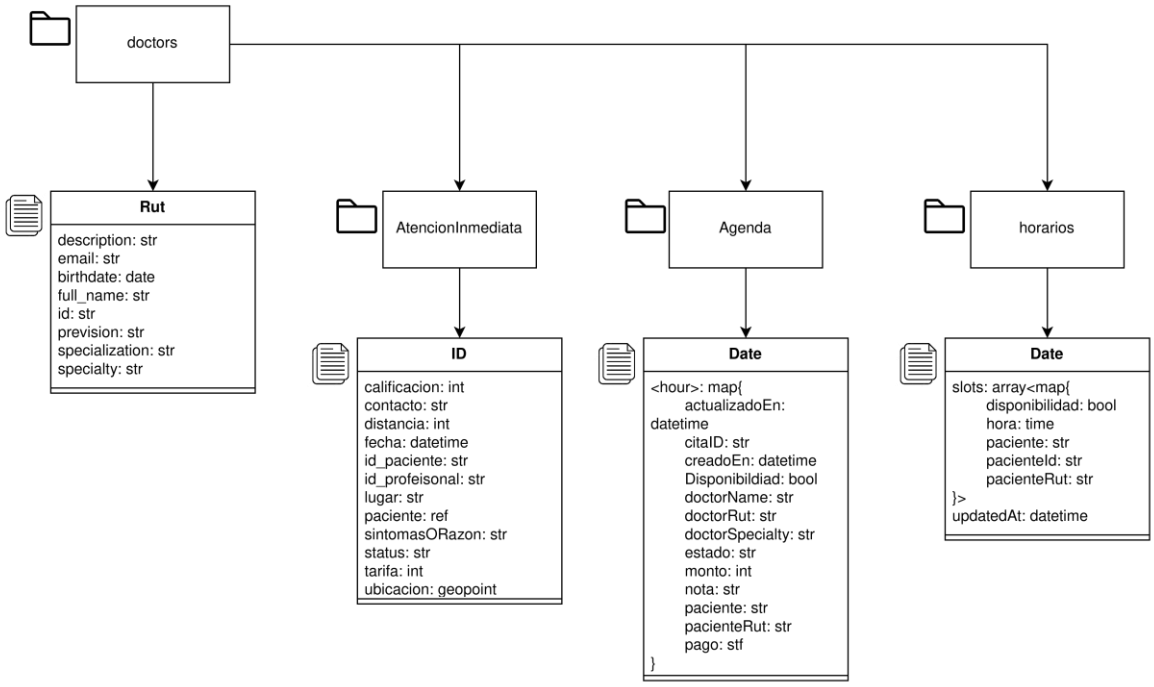


Figura 9 Estructura BD doctors (Elaboración Propia)

- Doctor [/doctors/{Rut doctor}]

Nombre de variable	Tipo	Descripción
Descripcion	STR	Descripción breve del doctor
Email	STR	Email del doctor
birthdate	DATETIME	Fecha del nacimiento en formato *DateTime* del doctor
Full_name	STR	Nombre completo del doctor
Id	STR	Identificador otorgado al doctor proporcionado por Firebase Authenticator al momento de crear la cuenta.
prevision	STR	Previsión del doctor (Fonasa, Isapre, etc.)
specialization	STR	Especialización del doctor
specialty	STR	Especialización especifica dentro de la especialización del doctor.

Tabla 5 Estructura Doctor (Elaboración Propia)

- Atención inmediata[/doctors/{Rut doctor}/AtencionInmediata/{id autogenerado}]

Nombre de variable	Tipo	Descripción
Calificacion	INT	Calificación otorgada al final de la atención. Default: 0
contacto	STR	Vía de contacto del doctor, email.
distancia	INT	Distancia hasta el paciente en metros
fecha	DATETIME	Fecha en la cual se pidió la Atención Inmediata
Id_paciente	STR	Identificador del paciente
Id_profesional	STR	Identificador del profesional
lugar	STR	método de atención (a domicilio o mediante videollamada)
paciente	REF	Referencia al documento *Patient* del paciente solicitante
sintomaORazon	STR	Breve descripción de la razón o síntoma que presenta el paciente
status	STR	Estado de la solicitud
tarifa	INT	Valor de la solicitud
Ubicacion	GEOPOINT	Ubicación del paciente en coordenadas.

Tabla 6 Estructura Atención Inmediata de doctor (Elaboración Propia)

- Agenda [/doctors/{Rut doctor}/Agenda/{Fecha}]

Nombre de variable	Tipo	Descripción
Hora	MAP	Contiene los valores siguientes valores a continuación dentro de él.
ActualizadoEn	DATETIME	*Timestamp* de la última actualización del documento
citaID	STR	Identificador de la cita formado por el Rut del doctor y la fecha y hora de cuando se creó la cita
creadoEn:	DATETIME	*Timestamp* de la creación del documento
Disponibilidad	BOOL	Indica si este horario se encuentra agendado por algún paciente
doctorName	STR	Nombre del doctor
doctorSpecialty	STR	Especialidad del doctor
estado	STR	Estado actual de la cita
monto	STR	Valor de la visita en *String* para facilitar su uso
nota	STR	información adicional de la cita
paciente	STR	Nombre del paciente
pacienteRut	STR	Rut del paciente
pago	STR	método de pago del paciente

Tabla 7 Estructura Agenda (Elaboración Propia)

- Horarios [/doctors/{Rut doctor}/horarios/{Fecha}]

Nombre de variable	Tipo	Descripción
UpdatedAt	DATETIME	*Timestamp* de la última actualización del documento
Slots	ARRAY <MAP>	Una lista que proporciona información resumida del horario del doctor conteniendo lo que se muestra a continuación
Disponibilidad	BOOL	Indica si este horario se encuentra agendado por algún paciente
Hora	TIME	Hora de atención activada por el doctor
Paciente	STR	Nombre del paciente
pacienteId	STR	Identificador del paciente
pacienteRut	STR	Rut del paciente

Tabla 8 Estructura Horarios (Elaboración Propia)

3.4.3 Modelos Compartidos

Los modelos compartidos son modelos que son ocupados por ambos usuarios. Estos están conformados de tres colecciones: Chats, el cual contiene los chats entre el paciente y el doctor, *atencioninmediata* el cual contiene los puntos en el mapa de los doctores e historial el cual contiene la información de todas las consultas terminadas, ya sean agendadas o de forma inmediata

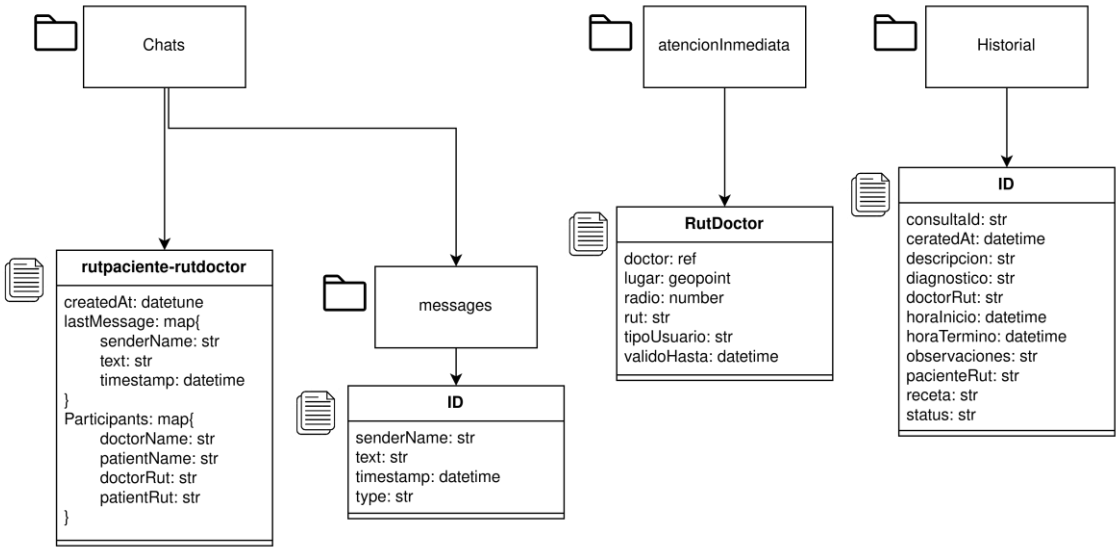


Figura 10 Estructura BD colecciones en común (Elaboración Propia)

- Chats [/Chats/{Rut paciente-rut doctor}/]

Nombre de variable	Tipo	Descripción
CreatedAt	DATETIME	*Timestamp* de cuando fue creado el chat
lastMessage	MAP	Map que contiene la información del último mensaje enviado
Participants	MAP	Map que contiene información del paciente y del doctor

Tabla 9 Estructura Chats (Elaboración Propia)

- Mensajes [/Chats/{rut paciente-rut doctor}/messages/{id autogenerado}]

Nombre de variable	Tipo	Descripción
senderName	STR	Nombre de quien envió el mensaje
Text	STR	Contenido del mensaje
Timestamp	DATETIME	*Timestamp* de cuando fue enviado el mensaje
Type	STR	Tipo de mensaje enviado (Texto, imagen, etc.)

Tabla 10 Estructura mensajes (Elaboración Propia)

- AtencionInmediata [/atencionInmediata/{Rut doctor} /]

Nombre de variable	Tipo	Descripción
doctor	REF	Referencia al documento del doctor
lugar	GEOPOINT	Punto en el mapa (en coordenadas) donde el doctor eligió empezar la atención inmediata
radio	NUMBER	Radio en metros del rango de atención
rut	STR	Rut del doctor que creo el pin
tipoUsuario	STR	Tipo de usuario que creo el pin
Valido hasta	DATETIME	Fecha en la cual vence el pin

Tabla 11 Estructura Atención inmediata en común (Elaboración Propia)

- Historial [/historial/{id autogenerado} /]

Nombre de variable	Tipo	Descripción
consultaId	STR	Identificador de la consulta realizada ya sea mediante atención inmediata o la agenda
createdAt	DATETIME	*Timestamp* de cuando se creó el documento
descripcion	STR	Descripción breve de la consulta
diagnostico	STR	Diagnostico a la que llego el doctor
doctorRut	STR	Rut del doctor
horaInicio	DATETIME	Hora de inicio de la consulta
HoraTermino	DATETIME	Hora de término de la consulta
observaciones	STR	Observaciones que encontró el doctor durante la consulta
pacienteRut	STR	Rut del paciente
receta	STR	Receta generada al final de la consulta
status	STR	Estatus de la consulta

Tabla 12 Estructura Historial médico (Elaboración Propia)

3.5 Seguridad y protección de datos

Debido al manejo de información sensible —datos personales y ubicación— el sistema incorpora múltiples capas de seguridad.

- Autenticación mediante *Firebase Authenticator*: cada usuario recibe un token seguro tras iniciar sesión.
- Sistemas de roles mediante *Firebase Authenticator*: *Authenticator* permite la personalización de tokens permitiendo agregarle variables.
- Reglas de seguridad de Firestore: Definen reglas de CRUD basado en el estado de la autenticación y los roles.
- Cifrado proporcionado por *Authenticator*: los datos de usuario se cifran automáticamente sin manera de ver esa información por parte del administrador.
- Control de acceso por rol: Los pacientes y los doctores solo pueden acceder a recursos permitidos.
- Verificación de la integridad de los datos mediante Pydantic: No se permiten subir datos incompletos o malformados o si el usuario no se encuentra autenticado.

El flujo de seguridad entre la Base de datos *Firestore* y el *back-end* se muestra en el siguiente gráfico.

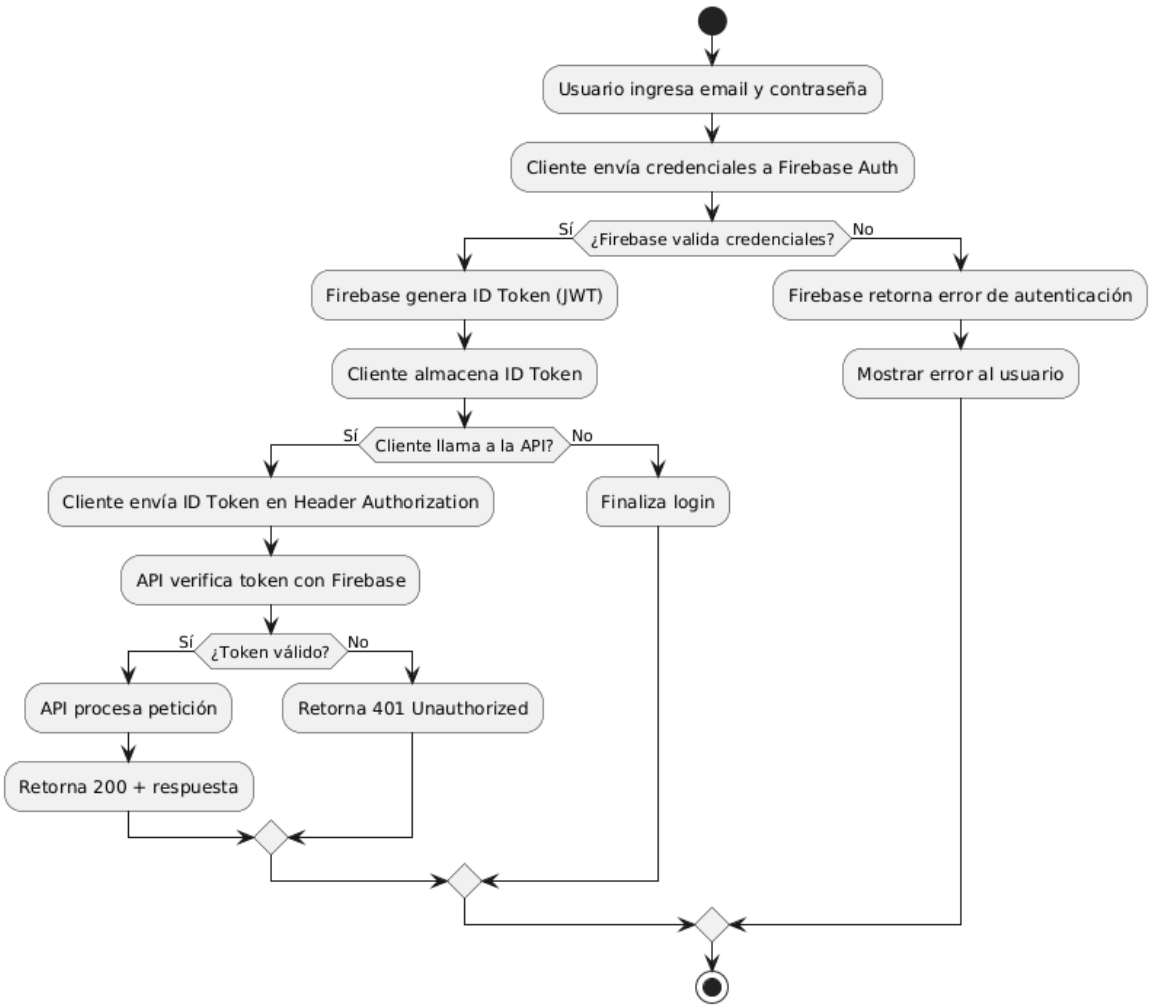


Figura 11 Grafico de flujo de autenticación del *Back-end* (Elaboración Propia)

4 Implementación del Sistema

El diseño de la implementación del *back-end* se materializa en una *API RESTful* robusta y servicios de *WebSocket* para funcionalidades en tiempo real. La arquitectura expone sus funcionalidades a través de un conjunto de *endpoints* organizados por dominio funcional.

4.1 Implementación de la API REST

La API se organiza en torno a dominios de negocio específicos, utilizando la estructura REST para las operaciones CRUD y las operaciones transaccionales.

4.1.1 Gestión de Usuarios y Perfiles (/user, /doctors y /patient)

Esta categoría maneja el ciclo de vida de los actores y la obtención de sus datos estáticos.

Endpoints	Método	Descripción Funcional	Acceso Requerido
/user/register/doctor	POST	Crea un nuevo registro en la colección *doctors*.	Público (requiere token Firebase)
/user/register/patient	POST	Crea un nuevo registro en la colección *Patient*.	Público (requiere token Firebase)
/doctors/{rut}	GET	Obtiene el perfil completo y datos estáticos de un profesional.	Público (requiere token Firebase)
/patient/{paciente_id}	GET	Obtiene el perfil completo y datos estáticos de un paciente.	Público (requiere token Firebase)
/doctors/update/{rut}	PATCH	Actualiza el perfil de un doctor	Doctor
/patient/update/{rut}	PATCH	Actualiza el perfil de un paciente	Paciente

Tabla 13 Endpoints /user, /doctors y /patient (Elaboración Propia)

4.1.2 Módulo de Atención (`/atencion`)

Esta es la sección que orquesta el *core* de la lógica de negocio basada en el estado y la geolocalización.

Endpoints	Método	Descripción Funcional	Acceso Requerido
/atencion/inmediata	POST	Permite al doctor activar su disponibilidad inmediata, definiendo su ubicación (lugar), el radio de atención (radio) y la duración de la activación (duración).	Doctor
/atencion/solicitar	POST	Genera una nueva solicitud de consulta, creando las referencias cruzadas entre el doctor y el paciente y estableciendo el estado inicial en "Pendiente".	Paciente
/atencion/status/{id}	PATCH	Actualiza el estado de la consulta a "Aceptada", "Rechazada", "Cancelada" o "Finalizada". Esta operación es transaccional y modifica los documentos tanto del doctor como del paciente.	Doctor
/atencion/active	GET	Verifica si el servicio de atención inmediata del usuario autenticado se encuentra activo y no ha expirado.	Público (requiere token Firebase)
/atencion/location/{rut}	GET	Obtiene la ubicación del doctor registrada en su documento de disponibilidad inmediata.	Público (requiere token Firebase)
/atención/finalizar/{id}	POST	Finaliza la consulta y guarda la información que escribió el doctor en la colección Historial	Doctor

Tabla 14 Endpoints /atencion (Elaboración Propia)

4.1.3 Módulo de Agenda (/agenda)

Este es el módulo que está orientado principalmente a consultas a futuro mediate el agendamiento de pacientes.

Endpoints	Método	Descripción Funcional	Acceso Requerido
/agenda/ agregarhorario/{Rut}	POST	Permite al doctor añadir horarios de atención a su agenda, definiendo la fecha y la hora	Doctor
/agenda/horarios/{Rut}	GET	Obtiene los horarios del doctor con un Rut en especifico	Público (requiere token Firebase)
/agenda/ horario/{Rut}/{hora}	PATCH	Permite que un paciente tome un horario de un doctor en una hora en específico (Paciente). Permite a un doctor aceptar, rechazar, va en camino o finalizar una solicitud (Doctor).	Público (requiere token Firebase)
/agenda/horario/ {rut}/{hora}/delete	DELETE	Permite al doctor eliminar un horario designado	Doctor

Tabla 15 Endpoints /agenda (Elaboración Propia)

4.1.4 Módulo de Chat (/chat)

Este es el módulo que proporciona métodos REST para el funcionamiento de la comunicación entre actores. Enfocado en los pasos iniciales, obtención de chats activos y un método *fallback* en caso de que el *websocket* falle.

Endpoints	Método	Descripción Funcional	Acceso Requerido
/chat/create_chat/	POST	Inicializa el documento del chat entre paciente y doctor definiendo los participantes y el id conformado por el Rut de ambos en la colección chat	Público (requiere token Firebase)
/chat/ send_message/{chatid}	POST	Alternativa REST para enviar mensajes, guarda el mensaje en la sub-coleccion messages.	Público (requiere token Firebase)
/chat/messages/{chatid}	GET	Recupera el historial de mensajes.	Público (requiere token Firebase)
/chat/my_chats/rut	GET	Obtiene la lista de chats activos	Público (requiere token Firebase)

Tabla 16 Endpoints /chat (Elaboración Propia)

4.2 Implementación de Servicios en Tiempo Real (WebSockets)

Para las funcionalidades de geolocalización y comunicación inmediata, el *back-end* utiliza el protocolo *WebSocket* para establecer canales de comunicación bidireccional y persistentes.

4.2.1 Chat en Tiempo Real

La comunicación 1 a 1 entre el Doctor y el Paciente se implementa a través de un *WebSocket* dedicado.

- **Ruta del WebSocket:** ws://<host>/ws/{chat_id}.
- **Mecanismo:** El chat_id (ejemplo: rutPaciente-rutDoctor) funciona como la clave de la sala virtual donde solo los dos participantes son autorizados a conectarse.
- **Flujo:** Al recibir un mensaje, el *back-end* lo persiste en la sub-colección messages de Firestore y luego lo distribuye (*broadcast*) inmediatamente a todos los *sockets* conectados a esa sala, garantizando la inmediatez.

4.2.2 Seguimiento GPS de Profesionales

Para soportar la funcionalidad de atención inmediata, la ubicación de los doctores activos se transmite en tiempo real.

- **Ruta del WebSocket:** ws://<host>/realtime/ws/{rut_profesional}.
- **Fuente de Datos:** A diferencia del Chat, los datos de ubicación se leen directamente de *Firebase Realtime Database* en la ruta locations/<rut-profesional>/, aprovechando su rendimiento superior para el manejo de datos de alta frecuencia.
- **Formato de Salida:** El WebSocket transmite un objeto JSON simple con la ubicación y marca de tiempo:

```
{
  "latitude": -33.1234,
  "longitude": -70.5678,
  "timestamp": 1732668898
}
```

4.2.3 Pines en el mapa

Para que la aplicación sea responsiva al momento de que un profesional active su atención inmediata y este se muestre de manera automática en la pantalla del usuario.

- **Ruta del WebSocket:** ws://<host>/atención/ws/
- **Fuente de datos:** Colección atencionInmediata presente en *Firestore Database*
- **Flujo:** Realiza un broadcast cada vez que detecta un nuevo pin dentro de la colección.

5 Pruebas e iteraciones

Para asegurar el correcto funcionamiento del *back-end* se validó usando las siguientes pruebas:

- Pruebas unitarias (*pytest* + *pytest-cov*)
- Pruebas de integración (*FastAPI TestClient* + *Firebase Emulator Suite*)
- Pruebas funcionales y de aceptación con usuarios reales.

5.1 Pruebas unitarias

Enfocadas principalmente a la capa de servicio. El objetivo fue verificar la lógica individual de cada componente de manera aislada. Para realizar estas pruebas se utilizó el *framework pytest* de Python. Generando

archivos de prueba por cada función disponible en cada uno de los servicios verificando la lógica de manipulación de fechas y correcto mapeo de datos y la estructura de *Firebase*.

5.2 Pruebas de integración

Estas pruebas validaron el correcto flujo de datos y la comunicación entre dos o más módulos internos. Enfocadas en la capa de presentación y la capa de servicios, así como la correcta interacción con la base de datos. Para esto se usó TestClient integrado en *FastAPI*. Esto permite simular peticiones HTTP contra los *endpoints* definidos en el *FastAPI*. Las pruebas de integración clave se hicieron contra una instancia de *Firebase* de simulación para asegurar la persistencia correcta de los datos.

Ejemplos:

- Creación de atención inmediata -> verificación de referencias cruzadas en colecciones Patient y Doctor.
- Expiración automática de pin de disponibilidad tras X minutos
- Rechazo de solicitud por radio fuera de cobertura

5.3 Pruebas funcionales

Estas pruebas verificaron el comportamiento del sistema bajo escenarios de usuario completo y en condiciones de tiempo real.

Caso de prueba	Actor	Resultado	Tiempo promedio
Registro y login	Ambos	Exito	40 s
Activar marca de disponibilidad inmediata	Doctor	Exito	30 s
Visualizar doctores cercanos en mapa	Paciente	Exito	2 s
Solicitar atención inmediata	Paciente	Exito	50 s
Aceptar/rechazar solicitud	Doctor	Exito	10 s
Chat en tiempo real	Ambos	Exito	320 ms
Seguimiento GPS del doctor en ruta	Paciente	Exito	Actualización cada 3-8 s
Finalización y guardado en historial	Doctor	Exito	-

Tabla 17 Casos de prueba (Elaboración propia)

Observación cualitativa: los participantes calificaron la experiencia como muy intuitiva y fácil de aprender con un promedio de aceptación de 4.6. Esta prueba fue realizada enfocándose en la UX.

6 Limitaciones y trabajo futuro

6.1 Limitaciones Identificadas en el Desarrollo

El proceso de desarrollo e implementación, si bien fue exitoso en la validación de los requerimientos funcionales, permitió identificar limitaciones tanto a nivel de infraestructura como de integración con ecosistemas externos. El análisis de estas barreras es fundamental para la planificación de la siguiente fase del proyecto.

6.1.1 Desafíos de Infraestructura y Seguridad en la Nube

La decisión de desplegar el *back-end* en un entorno de nube para pruebas de carga reveló un desafío de infraestructura no funcional relacionado con la seguridad y la resiliencia del servicio:

- **Ataques de Denegación de Servicio (DoS) y Verificación de Permisos**: Al subir el *back-end* a un servicio en la nube, el sistema fue objeto de tráfico malicioso (*bots*) diseñado para estresar los *endpoints* públicos. Si bien esto tuvo la desventaja operativa de ralentizar significativamente el servicio debido a la sobrecarga de peticiones, sirvió como una prueba de fuego para la capa de seguridad. Se pudo verificar que los mecanismos de autorización y autenticación implementados con *Firebase* funcionaron correctamente al bloquear o limitar el acceso no autorizado, protegiendo la integridad de la base de datos de Firestore. Este evento subraya la necesidad de implementar soluciones de *rate-limiting* y *Web Application Firewall* (WAF).

6.1.2 Restricciones de Acceso a APIs Críticas

La integración de funcionalidades que dependen de datos oficiales o certificados encontró una limitación de carácter administrativo y económico:

- **Acceso a la API del Ministerio de Salud**: No fue posible incorporar la API del Ministerio de Salud para la validación o el enriquecimiento de datos de profesionales. Esto se debe a que la documentación oficial requiere que el solicitante esté registrado y acreditado como un centro de salud formal y, en muchos casos, exige el pago de una suscripción para el acceso a sus servicios. Esta limitación obligó a enfocar el alcance inicial del proyecto en la validación interna de los datos y en la autenticación vía *Firebase*, dejando la validación oficial como un requerimiento para futuras versiones.

6.1.3 Limitación del servicio en la nube y costos asociados

Además del impacto operativo generado por el tráfico malicioso, el despliegue temporal del hacen en un entorno público reveló una limitación adicional relacionada con el modelo de uso de *Firebase*. Debido al volumen inusual de solicitudes provocado por el ataque, el proyecto superó los límites establecidos por el plan gratuito, generando cargos no previstos.

Si bien la integridad del sistema no se vio comprometida, este incidente evidenció una restricción importante del servicio: el consumo derivado de tráfico no legítimo también es contabilizado por la plataforma, lo que puede traducirse en costos adicionales aun cuando la funcionalidad esté correctamente protegida a nivel de permisos y autenticación.

Este hallazgo subraya la importancia de complementar la seguridad lógica con medidas preventivas orientadas al control de consumo —como límites de cuota, monitoreo activo o un *middleware* de *rate-limiting*— para evitar que incidentes externos afecten económicamente el entorno de pruebas o producción.

6.2 Trabajo Futuro y Líneas de Expansión

La arquitectura modular y de capas del *back-end* permite su expansión y mejora en múltiples frentes. Las siguientes líneas de trabajo se proponen para evolucionar el sistema hacia un producto comercialmente viable y más robusto.

6.2.1 Robustez y Rendimiento de la Persistencia

A fin de manejar mayores volúmenes de datos y consultas complejas, se propone una migración y expansión de la capa de persistencia.

- **Migración a Base de Datos Relacional (PostgreSQL):** La base de datos actual (Firestore, NoSQL) es adecuada para la escalabilidad horizontal y la sincronización en tiempo real. Sin embargo, para soportar consultas más complejas, analíticas y relaciones estrictas entre entidades (Doctores, Pacientes, Citas), se recomienda la migración a una base de datos más robusta como PostgreSQL. Esto mejorará la integridad referencial y facilitará futuras operaciones de Inteligencia Empresarial.
- **Servicio de Big Data y Analítica:** Implementar una infraestructura de recolección y análisis de datos para transformar la data transaccional en *insights* de negocio. Esto incluiría la integración con herramientas como Google BigQuery o un data *warehouse* dedicado, permitiendo el análisis del rendimiento de los profesionales, patrones de demanda geográfica y tiempos de respuesta del servicio.

6.2.2 Expansión de Servicios Funcionales y Comerciales

- **Integración de Pasarelas de Pago**: Desarrollar un módulo de pagos que se integre con pasarelas comerciales. Esto requerirá la implementación de *endpoints* de *Webhooks* para manejar las notificaciones asíncronas de pagos exitosos y fallidos, garantizando que el estado de la cita sea consistente con la transacción financiera.

- **Vía de Agendamiento Telefónica (*Call Center API*)**: Expandir la Capa de Presentación para incluir un set de *endpoints* dedicados a la integración con sistemas de *Call Center*. Esto permitiría a los agentes de atención programar citas a nombre del paciente o del doctor, lo que requerirá un servicio de autorización propio que maneje las credenciales y el ámbito de acción del agente.

6.2.3 Refuerzo de Seguridad y Autonomía

- **Implementación de un Servicio de Autorización Propio**: Si bien *Firebase Authentication* ha sido efectivo para la autenticación, se propone la implementación de un servicio de autorización propietario (ej. basado en OAuth 2.0 y JWT) para manejar la emisión, renovación y validación de tokens. Esto proporcionará un control total sobre las políticas de acceso, permitirá la creación de roles y permisos más granulares (admin, agente, profesional acreditado) y eliminará la dependencia directa de Firebase Auth en el futuro.

7 Conclusiones

El desarrollo del sistema de back-end orientado a un servicio de salud a domicilio representó un desafío significativo, pero al mismo tiempo una valiosa experiencia de aprendizaje. Este proyecto permitió comprender en profundidad los requerimientos técnicos, organizativos y de seguridad necesarios para el desarrollo de soluciones informáticas en el ámbito de la salud. Asimismo, se adquirieron competencias clave relacionadas con la correcta estructuración de un back-end, facilitando su mantenimiento, escalabilidad y comprensión, junto con la adecuada configuración de los distintos niveles de seguridad del sistema.

Uno de los principales desafíos enfrentados durante el desarrollo fue el manejo de datos sensibles. La información médica de pacientes y profesionales de la salud requiere un tratamiento riguroso y seguro, lo que implicó un proceso constante de investigación y pruebas, especialmente en lo relacionado con la definición de reglas de seguridad tanto en la base de datos como en la lógica del sistema. Este proceso se vio considerablemente facilitado mediante el uso de los servicios de Firebase, lo que permitió centrar los esfuerzos en la funcionalidad del sistema y en la construcción de una arquitectura modular. Como consecuencia natural de este enfoque iterativo, se realizaron múltiples reestructuraciones del modelo de datos, culminando en el diseño final presentado en este documento.

Durante el desarrollo del proyecto, el uso del framework FastAPI resultó fundamental, principalmente gracias a su sistema de documentación automática, el cual facilitó el seguimiento de las rutas, los tipos de datos y los contratos de la API a lo largo de los meses de trabajo. Adicionalmente, la activa participación del patrocinador del proyecto fue un factor clave para su éxito, ya que permitió validar el sistema con profesionales de la salud reales, obteniendo retroalimentación directa, sugerencias de nuevas funcionalidades y una mejor comprensión de las necesidades reales del entorno clínico.

Si bien Firebase aportó múltiples ventajas durante la implementación del proyecto, también permitió identificar ciertas limitaciones, tales como restricciones de uso, costos asociados y limitaciones arquitectónicas, derivadas de su enfoque hacia el uso integrado de la suite completa de Firestore. Estas limitaciones, junto con la escasa integración con otros servicios externos, motivaron la propuesta de trabajos futuros orientados a la migración hacia una base de datos relacional y la implementación de un sistema de autenticación propio.

Finalmente, se concluye que el proyecto logró cumplir satisfactoriamente con los objetivos planteados, demostrando que la solución propuesta es técnicamente viable para abordar la problemática expuesta en esta tesina. El sistema desarrollado constituye una base sólida y funcional para un emprendimiento real, con impacto social y potencial comercial, además de presentar una alta capacidad de extensibilidad hacia otros ámbitos del sector salud o áreas afines.

8 Agradecimientos

Agradezco de gran manera al profesor Dagoberto Cabrera Tapia junto al profesor Correferente Diego Cáceres Solís por su guía y recomendaciones durante el desarrollo de este trabajo. Asimismo, expreso gratitud a el patrocinador Gonzalo Ortega de Kinin por su apoyo y permitirnos verificar la aplicación con doctores reales y finalmente a mi equipo de trabajo y compañeros del equipo Kineticos por su increíble compromiso y cooperación, este proyecto no pudo haber sido posible realizar sin ellos. Y, finalmente, a mi familia por acompañarme durante todo este proceso, tanto durante los malos momentos como en los buenos.

9 Referencias

[1] Google, “Flutter Documentation,” 2025. [En línea]. Disponible: https://flutter.dev/docs

[2] Google, “Firebase Cloud Firestore Documentation,” 2025. [En línea]. Disponible: https://firebase.google.com/docs/firestore

[3] Google, “Firebase Authentication Documentation,” 2025. [En línea]. Disponible: https://firebase.google.com/docs/auth

[4] T. M. Tiago, “FastAPI – Modern, fast web framework for building APIs with Python,” 2025. [En línea]. Disponible: https://fastapi.tiangolo.com

[5] Pydantic Developers, “Pydantic Documentation,” 2025. [En línea]. Disponible: https://docs.pydantic.dev

[6] W3C, “WebSocket API – Candidate Recommendation,” 2011. [En línea]. Disponible: https://www.w3.org/TR/websockets/

[7] WebSocket.org, “The WebSocket Protocol (RFC 6455),” 2011. [En línea]. Disponible: https://datatracker.ietf.org/doc/html/rfc6455

[8] Organización Panamericana de la Salud (OPS/OMS), La salud en las Américas: Chile 2023. Washington, DC: OPS, 2023. [En línea]. Disponible: https://www.paho.org/es/chile

[9] Ministerio de Salud de Chile, “Más de 2 millones de personas han egresado de la lista de espera en 2025 y un aumento del 35 % en la producción de la red asistencial,” Santiago, Chile, 2025. [En línea]. Disponible: https://www.minsal.cl/mas-de-2-millones-de-personas-han-egresado-de-la-lista-de-espera-en-2025-y-un-aumento-del-35-en-la-produccion-de-la-red-asistencial/

[10] Pro Salud Chile, “Pacientes del sistema público en listas de espera llegan a inédita cifra de 2,5 millones,” Pro Salud Chile, jun. 2024. [En línea]. Disponible: https://prosaludchile.cl/pacientes-del-sistema-publico-en-listas-de-espera-llegan-a-inedita-cifra-de-25-millones/

[11] Ministerio de Educación de Chile, “Buscador de empleabilidad e ingresos – MiFuturo.cl,” Santiago, Chile, 2025. [En línea]. Disponible: https://www.mifuturo.cl/