UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

Arquitectura y Diseño de Software para MedMatch: Estrategias de Implementación, Escalabilidad y Requerimientos del Sistema en una Aplicación Móvil de Salud a Domicilio.

Juan Valdés Sánchez

juan.valdess@usm.cl

Catherine Gómez
Profesora Guía

David Larrondo
Profesor Correferente

Resumen: En la actualidad, el acceso a la atención médica enfrenta múltiples desafíos: la alta demanda en los centros de salud, la cesantía de profesionales recién egresados y las dificultades de traslado de pacientes con movilidad reducida. Estos problemas generan barreras tanto para la calidad de la atención como para la inserción laboral en el ámbito de la salud.
Como propuesta de solución se plantea *MedMatch*, una aplicación móvil desarrollada en Flutter cuyo objetivo es facilitar la conexión entre pacientes y profesionales de la salud para brindar atención domiciliaria. Este trabajo se centra en el diseño y análisis de la arquitectura del sistema, considerando aspectos de escalabilidad, mantenibilidad, seguridad y usabilidad, con miras a lograr una solución sostenible y profesional.
La validación de la propuesta se realiza mediante el diseño de una arquitectura modular, el levantamiento de requerimientos del sistema y la evaluación de tecnologías que permitan garantizar un desempeño confiable y adaptable a futuras mejoras.
Se espera que *MedMatch* no solo aporte valor tecnológico, sino también un impacto social positivo, mejorando el acceso a la salud y ofreciendo nuevas oportunidades laborales a profesionales, consolidándose como un proyecto con proyección real de implementación.

Palabras Clave: Aplicación móvil, arquitectura de software, salud a domicilio, escalabilidad, usabilidad.

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: Arquitectura y diseño de software para MedMatch: Estrategias de implementación, escalabilidad y requerimientos del sistema en una aplicación móvil de salud a domicilio.

Nombre del candidato(a): Juan José Valdés Sánchez.

Carrera / Grado: Ingeniería en informática

Campus: Sede de viña del mar **Departamento:** Electrotecnia e informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Catherine Gómez Barrera, 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

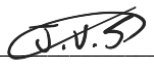
Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

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

Fecha: 22-01-2026 **Firma:** 

Estudiante o Candidato(a):

Fecha: 22-01-2026 **Firma:** 

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

1 Introducción

1.1 Contexto general del problema

Durante los últimos años, la atención médica domiciliaria ha cobrado mayor relevancia debido a la saturación de los centros de salud, la presencia de pacientes con movilidad reducida y la necesidad de generar más oportunidades laborales para profesionales recién egresados. Todo esto ha dejado en evidencia una brecha entre la cantidad de servicios disponibles y la capacidad real de los usuarios para acceder a ellos. Frente a este escenario, surge la necesidad de soluciones tecnológicas que permitan mejorar la gestión, coordinación y acceso a la atención clínica en el hogar.

1.2 Descripción sobre la propuesta de solución

En este contexto surge **MedMatch**, una aplicación móvil desarrollada en Flutter que tiene como propósito facilitar la conexión directa entre pacientes y profesionales de la salud, permitiendo gestionar solicitudes de atención, visualizar perfiles, agendar citas y realizar procesos asociados a la consulta domiciliaria. Si bien la aplicación constituye el caso práctico del proyecto, el foco de este trabajo se centra en su **arquitectura y diseño de software**, analizando las decisiones técnicas adoptadas, los patrones aplicados y la estructura que sustenta su funcionamiento.

1.3 Enfoque de arquitectura y diseño del sistema

El desarrollo de una plataforma orientada a servicios de salud requiere garantizar criterios de escalabilidad, mantenibilidad, seguridad, eficiencia y coherencia arquitectónica, lo cual demanda un diseño capaz de sostener la evolución futura del sistema y permitir su adaptación a escenarios de mayor demanda o nuevas funcionalidades. En consecuencia, este trabajo examina la aplicación desde una perspectiva ingenieril, abarcando tanto su diseño general como la interacción entre módulos, la estructura de sus componentes y la relación con servicios externos y la infraestructura en la nube.

Para lograrlo, se utiliza una arquitectura basada en MVVM, complementada con una organización feature-first. Esta combinación permite ordenar la aplicación en módulos como autenticación, agenda, atención inmediata, mensajería y gestión de perfiles, manteniendo una clara separación de responsabilidades. A esto se suma Firebase como Backend-as-a-Service, aprovechando sus capacidades de autenticación, persistencia, mensajería y sincronización en tiempo real, esenciales para una atención domiciliaria dinámica.

1.4 Alcance del trabajo y metodología general

Este documento presenta el análisis completo de la arquitectura implementada, describiendo el razonamiento detrás de las decisiones de diseño, los módulos que componen el sistema, los mecanismos de comunicación entre capas y las estrategias internas que permiten a MedMatch operar de manera confiable. Además, se incluyen las pruebas de validación necesarias para verificar el comportamiento del sistema y se discuten los lineamientos de escalabilidad que proyectan su crecimiento futuro.

1.5 Metodología y plan de trabajo

Para el desarrollo del sistema se utilizó la metodología ágil **Scrum**, dado que el equipo ya contaba con experiencia en esta metodología y ofrecía la flexibilidad necesaria para un proyecto que requiere ajustes constantes. El trabajo se organizó en **Sprints** con metas definidas y **reuniones Daily** para dar

seguimiento al progreso y resolver problemas e inquietudes de manera oportuna. Una de sus principales ventajas fue la **adaptabilidad**, que permitió incorporar retroalimentación temprana y realizar mejoras continuas sin perder de vista los objetivos de la tesina.

PLAN DE TRABAJO

El desarrollo del sistema se organizó bajo la metodología ágil **Scrum**, estructurando el avance en **tres sprints**. Cada sprint tuvo una duración aproximada de dos semanas y permitió entregar incrementos funcionales del sistema.

Sprint 1: Configuración inicial y bases del sistema

- **Objetivo:** Definir la estructura del sistema, levantar la base de datos en Firebase y establecer la lógica mínima del frontend.
- **Historias de usuario trabajadas:**
    - **H.U.1:** Yo como profesional de la salud quiero poder aceptar solicitudes de pacientes para ser atendidos a futuro.
    - **H.U.2:** Yo como paciente quiero poder solicitar la atención de un profesional para un futuro cercano.
- **Principales logros:**
    - Configuración inicial del proyecto en Flutter.
    - Definición y creación de la base de datos en Firebase (estructura de colecciones y documentos).
    - Desarrollo de pantallas iniciales para la interacción básica entre paciente y profesional.
    - Configuración de GitHub como repositorio central de trabajo colaborativo.

Sprint 2: Integración de funcionalidades clave y backend

- **Objetivo:** Completar el backend y comenzar con la lógica de interacción en tiempo real entre pacientes y profesionales.
- **Historias de usuario trabajadas:**
    - **H.U.3:** Yo como profesional de la salud quiero poder atender a pacientes en tiempo real.
    - **H.U.4:** Yo como profesional de la salud quiero poder escoger mi horario en el que voy a atender a los pacientes.
    - **H.U.7:** Yo como paciente quiero solicitar profesionales de la salud en tiempo real.
- **Principales logros:**

- Implementación de la lógica de atención en tiempo real (uso de StreamBuilder/FutureBuilder con Firebase).
- Definición y aplicación de reglas de validación en la BD (ejemplo: evitar duplicidad en RUT de usuarios).
- Implementación de vistas interactivas que permiten flujo dinámico entre pacientes y profesionales.
- Presentación de avances al patrocinador.

Sprint 3: Implementación del flujo de atención inmediata en tiempo real y mejoras de arquitectura

- **Objetivo:** Desarrollar la funcionalidad principal del sistema correspondiente al flujo completo de atención inmediata en tiempo real. Además, finalizar tareas pendientes de la iteración anterior, optimizar la arquitectura del proyecto y realizar una limpieza estructural de la base de datos para asegurar un funcionamiento eficiente y escalable.
- **Historias de usuario trabajadas:**
  - **H.U.5:** Yo como profesional de la salud quiero que el sistema me guíe sobre cómo puedo llegar hacia donde está el paciente.
  - **H.U.6:** Yo como paciente quiero poder saber por dónde viene el profesional de la salud en tiempo real.
  - **H.U.7:** Yo como paciente quiero solicitar profesionales de la salud en tiempo real.
- **Principales logros:**
  - Implementación del flujo completo de atención inmediata entre paciente y profesional.
  - Integración de visualización de ubicación en tiempo real para paciente y profesional mediante geolocalización.
  - Desarrollo de la vista que muestra la ruta sugerida para ambos perfiles durante la atención inmediata.
  - Implementación del formulario de ficha médica para que el profesional registre la atención realizada.
  - Incorporación del sistema de calificación para que el paciente valore la atención recibida.
  - Mejora y reorganización de la arquitectura del proyecto (aplicación de MVVM, limpieza de vistas, separación de responsabilidades, refactorización de servicios).
  - Limpieza y reestructuración de la base de datos para asegurar coherencia en nodos, documentos y campos utilizados.

1.6 Breve descripción de la organización del informe en capítulos.

El presente documento se organiza en seis capítulos, cuyo contenido se detalla a continuación.

El Capítulo 1 corresponde a la *Introducción y presenta el contexto general del problema*, la motivación del proyecto, la solución propuesta, el enfoque arquitectónico adoptado y una descripción general de la estructura del trabajo.

El Capítulo 2 expone el *planteamiento del problema y los objetivos del estudio*. En esta sección se define con precisión la problemática que da origen al proyecto, su justificación técnica y social, junto con los objetivos generales y específicos que orientan el desarrollo de la solución.

El Capítulo 3 desarrolla *el marco teórico* que sustenta el análisis arquitectónico. Incluye conceptos relacionados con la arquitectura de software, patrones de diseño aplicados, tecnologías utilizadas, requerimientos no funcionales y fundamentos relevantes para comprender la estructura técnica de MedMatch.

El Capítulo 4 aborda el diseño e implementación de la arquitectura del sistema. Este apartado describe la estructura modular de la aplicación, sus componentes principales, los diagramas arquitectónicos, la interacción entre módulos, la organización MVVM y las estrategias técnicas utilizadas para garantizar escalabilidad, mantenibilidad y seguridad.

El Capítulo 5 presenta las *pruebas y validaciones realizadas al sistema*. Se incluyen pruebas funcionales, de integración y de comportamiento, además de la verificación de los módulos críticos y la evaluación del desempeño general de la aplicación en su entorno operativo.

Finalmente, **el Capítulo 6** expone las *conclusiones del proyecto*, destacando los resultados alcanzados, las limitaciones encontradas y las oportunidades de evolución futura para la plataforma.

2 Planteamiento del problema y objetivos

2.1 Definición del problema

El acceso oportuno y adecuado a la atención médica constituye un desafío persistente en distintos territorios del país. La alta demanda en los centros asistenciales, junto con la limitada disponibilidad de recursos humanos y físicos, ha generado una saturación constante del sistema de salud, especialmente en el nivel de atención primaria. Esta situación se ha visto reflejada en el aumento sostenido de las listas de espera y en los tiempos prolongados para acceder a una consulta médica, lo que impacta directamente en la continuidad de los tratamientos y en la calidad de la atención entregada.

Este escenario afecta con mayor intensidad a personas con movilidad reducida, adultos mayores y pacientes dependientes, quienes enfrentan barreras logísticas adicionales para trasladarse a centros de salud, agravando aún más la dificultad de acceso. Diversos informes institucionales han señalado que la sobrecarga del sistema sanitario limita la capacidad de respuesta frente a las necesidades reales de la población, evidenciando brechas estructurales en la provisión de atención oportuna, particularmente fuera del entorno hospitalario tradicional (Ministerio de Salud de Chile, 2023). La ausencia de plataformas tecnológicas especializadas, seguras y orientadas a la atención domiciliaria impide articular de manera eficiente estos dos extremos. Muchas soluciones actuales se limitan a ofrecer canales de comunicación básicos, sin abordar aspectos críticos como verificación de profesionales, disponibilidad en tiempo real, gestión de perfiles clínicos, rutas de atención o

mecanismos de priorización. La carencia de una estructura arquitectónica robusta limita la escalabilidad y la posibilidad de integrarse con servicios futuros.

Por otra parte, un número significativo de profesionales de la salud, particularmente aquellos recién egresados, experimenta dificultades para insertarse laboralmente durante sus primeros años de ejercicio. Aunque los datos más recientes disponibles muestran que la dotación de médicos y enfermeras en Chile ha crecido en los últimos años, la cantidad de profesionales por habitante sigue siendo inferior al promedio observado en países de la Organización para la Cooperación y el Desarrollo Económico, lo que sugiere limitaciones estructurales en la disponibilidad de recursos humanos en el sector sanitario (OCDE, 2025). Esta situación, sumada a restricciones presupuestarias, la capacidad limitada de contratación en instituciones públicas y privadas, y la ausencia de mecanismos tecnológicos modernos que conecten de manera eficiente la oferta y la demanda de atención médica, contribuye a un desacoplamiento entre quienes necesitan recibir atención y quienes están disponibles para prestarla.

En este contexto, se identifica la necesidad de una solución tecnológica que permita conectar de forma confiable a pacientes y profesionales de la salud, asegurando trazabilidad, seguridad de los datos, disponibilidad y una arquitectura flexible que soporte la evolución del sistema. Esta problemática constituye el eje central del proyecto y orienta el diseño arquitectónico presentado en este trabajo.

2.2 Justificación del proyecto

El desarrollo de MedMatch se justifica por su potencial impacto tanto en el ámbito social como en el tecnológico. Desde la perspectiva del usuario, la plataforma ofrece una alternativa concreta para disminuir las barreras asociadas al acceso a la atención domiciliaria, permitiendo que pacientes puedan gestionar solicitudes médicas de forma ágil, confiable y desde un dispositivo móvil. La digitalización de este proceso optimiza los tiempos de respuesta, reduce desplazamientos y contribuye a la continuidad del cuidado.

En términos profesionales, MedMatch favorece la empleabilidad de especialistas de la salud al habilitar un canal moderno para ofrecer sus servicios bajo demanda, con visibilidad georreferenciada y criterios de disponibilidad personalizables. Esta dinámica permite a los profesionales administrar su jornada y ampliar sus oportunidades de trabajo, sin depender exclusivamente de instituciones tradicionales.

Desde una perspectiva técnica, el proyecto permite abordar un desafío de arquitectura de software relevante: diseñar una aplicación móvil con una estructura modular, escalable y mantenible que sea capaz de operar sobre servicios cloud, específicamente Firebase. La arquitectura propuesta combina *el patrón MVVM* para la organización interna del código con una serie de estrategias orientadas a la expansión futura del sistema, tales como modularidad, separación de capas, autenticación segura y gestión eficiente de datos en tiempo real.

Al integrar estos elementos, MedMatch no solo entrega una solución funcional, sino que también constituye un aporte académico en términos de diseño arquitectónico aplicado a sistemas móviles centrados en la salud. Este proyecto promueve buenas prácticas de desarrollo, fomenta el uso de tecnologías modernas y establece una base sólida para futuras iteraciones del sistema.

2.3 Objetivo general

Diseñar y analizar la arquitectura de software de una aplicación móvil orientada a la atención de salud domiciliaria, asegurando un enfoque modular, escalable y seguro que permita conectar de manera

eficiente a pacientes con profesionales de la salud mediante el uso de tecnologías basadas en Flutter y Firebase.

2.4 Objetivos específicos

- **Levantar y estructurar los requerimientos funcionales y no funcionales** del sistema, considerando aspectos críticos como seguridad, rendimiento, usabilidad y mantenibilidad.
- **Diseñar la arquitectura general y modular del sistema**, incorporando diagramas estructurales y de componentes que describan la interacción entre los módulos principales de MedMatch.
- **Aplicar el patrón de diseño MVVM** para organizar la lógica interna de la aplicación móvil, mejorando la separación de responsabilidades y facilitando su evolución futura.
- **Describir las estrategias de implementación utilizadas**, considerando autenticación, gestión de datos en tiempo real, manejo de estados y organización de servicios internos.
- **Analizar la escalabilidad y sostenibilidad técnica** de la solución propuesta, evaluando su capacidad para soportar mayor carga de usuarios, módulos adicionales y futuras integraciones.
- **Documentar los criterios de seguridad aplicados**, incluyendo medidas para la protección de datos sensibles, permisos de acceso y resguardo de información clínica.

3 Marco Teórico

3.1 Conceptos fundamentales

La comprensión de los elementos que dan sustento técnico a MedMatch requiere revisar los principios y tecnologías que intervienen en su diseño e implementación. Este marco teórico articula los conceptos esenciales relacionados con arquitectura de software, patrones de diseño, servicios en la nube y plataformas de desarrollo móvil, con el fin de contextualizar las decisiones técnicas adoptadas en este proyecto.

3.1.1 Arquitectura de software

La arquitectura de software corresponde al conjunto de decisiones estructurales que determinan la organización del sistema, la relación entre sus componentes y los principios que rigen su evolución. Constituye un plano conceptual que orienta tanto el diseño como el desarrollo, permitiendo establecer lineamientos sobre modularidad, cohesión, acoplamiento, comunicación interna y mecanismos de escalabilidad.

En aplicaciones móviles modernas, la arquitectura debe proporcionar una base suficientemente flexible para incorporar nuevas funcionalidades sin comprometer la estabilidad del sistema. Esto implica considerar aspectos como separación de responsabilidades, reutilización de componentes, claridad en los flujos de datos y uso eficiente de los recursos. La arquitectura seleccionada influye directamente en la mantenibilidad, rendimiento y seguridad de la aplicación, factores especialmente relevantes en entornos donde se manejan datos sensibles, como es el caso del ámbito sanitario.

En el contexto de aplicaciones orientadas al ámbito sanitario, el concepto de datos sensibles adquiere un rol central dentro de las decisiones arquitectónicas. De acuerdo con la **Ley N° 19.628 sobre Protección de la Vida Privada** (Biblioteca del Congreso Nacional de Chile, s.f.), en su artículo 2°, letra g), se entiende por dato sensible aquellos datos personales que se refieren a características físicas o morales de las personas, o a hechos o circunstancias de su vida privada, tales como los estados de salud físicos o psíquicos. Esta definición resulta especialmente pertinente para sistemas como MedMatch, que interactúan con información vinculada a la atención médica y al contexto personal de los usuarios.
Asimismo, el artículo 4° de la misma ley establece que el tratamiento de datos personales solo puede realizarse cuando la ley lo autoriza o cuando el titular consiente expresamente en ello, mientras que el artículo 10° restringe de manera más estricta el tratamiento de datos sensibles, exigiendo condiciones específicas para su uso. Estas disposiciones influyen directamente en la forma en que un sistema de salud debe estructurar el acceso, almacenamiento y circulación de la información.

En el caso de MedMatch, al tratarse de un prototipo académico y no de una plataforma clínica institucional en operación real, no se implementó un mecanismo formal de consentimiento explícito mediante políticas de privacidad o términos legales visibles dentro de la aplicación. No obstante, desde el diseño arquitectónico se consideraron principios alineados con la normativa vigente, tales como la autenticación obligatoria de usuarios, la separación de roles entre paciente y profesional, y la restricción de acceso a la información según el perfil autenticado. Estas decisiones buscan limitar el acceso indebido y reducir la exposición innecesaria de datos asociados al contexto de salud.

Por otra parte, la **Ley N° 20.584**, que regula los derechos y deberes de las personas (Biblioteca del Congreso Nacional de Chile, s.f.), en relación con acciones vinculadas a su atención de salud, establece en sus artículos 12° y 13° el derecho de los pacientes a la confidencialidad de su información y a que esta sea tratada con respeto y reserva. Si bien MedMatch no reemplaza sistemas clínicos formales ni gestiona fichas médicas oficiales, su diseño considera estos principios como referencia, evitando la exposición pública de información personal y limitando la visualización de datos únicamente a los actores involucrados en la atención.

Desde una proyección futura, la arquitectura modular y desacoplada de MedMatch permite incorporar de manera natural mecanismos adicionales de cumplimiento normativo, tales como consentimiento informado digital, políticas de privacidad visibles, trazabilidad de accesos y auditoría de operaciones sobre datos sensibles. De este modo, la arquitectura no solo responde a los requerimientos funcionales actuales del proyecto, sino que también sienta una base adecuada para una eventual adaptación a entornos productivos regulados.

3.1.2 Patrón de diseño MVVM

El patrón **Model–View–ViewModel (MVVM)** constituye uno de los enfoques más utilizados en el desarrollo de aplicaciones móviles debido a su capacidad para desacoplar las interfaces gráficas de la lógica de negocio. Este patrón se compone de tres elementos centrales:

- **Model**: Representa la estructura de los datos y las reglas que los gobiernan. Incluye entidades, validaciones, transformaciones y cualquier lógica asociada al dominio.
- **View**: Corresponde a la capa de presentación. Su función principal es mostrar la información al usuario y recibir sus interacciones.
- **ViewModel**: Actúa como intermediario entre la vista y el modelo. Gestiona el estado, procesa entradas del usuario, coordina llamadas a servicios y expone información observable para que la interfaz se actualice sin depender directamente de la lógica interna.

En el contexto de MedMatch, este patrón permite trabajar con módulos independientes, como autenticación, consultas médicas, comunicación y geolocalización, manteniendo la consistencia del flujo de datos y evitando dependencias directas entre capas. La adopción de MVVM favorece la escalabilidad del sistema y contribuye a una mayor claridad en la estructura del código, facilitando su mantenimiento futuro.

3.1.3 Aplicaciones móviles y framework flutter

El desarrollo de MedMatch se basa en **Flutter**, un framework multiplataforma que utiliza el lenguaje Dart y permite generar aplicaciones nativas compiladas para Android e iOS desde un único código fuente. Flutter destaca por su arquitectura declarativa y orientada a widgets, su capacidad para gestionar interfaces reactivas y su rendimiento cercano al nativo, lo que resulta fundamental para aplicaciones que requieren interacciones fluidas y actualizaciones en tiempo real.

Entre los beneficios técnicos de Flutter, destacan:

- Consistencia visual entre plataformas debido a su propio motor de renderizado.
- Desarrollo rápido mediante hot reload.
- Ecosistema de paquetes que facilita la integración con servicios externos.
- Facilidad para encapsular módulos y mantener una estructura limpia mediante patrones como MVVM.

Estas características se alinean con la necesidad de crear una aplicación modular, robusta y fácilmente extensible, acorde con los objetivos del proyecto.

3.1.4 Servicios cloud y ecosistema firebase

Firebase es una plataforma de servicios cloud que provee herramientas integradas para gestionar autenticación, almacenamiento, mensajería en tiempo real y despliegue. MedMatch utiliza principalmente:

- **Firebase Authentication**, para la gestión segura de accesos.
- **Firebase Realtime Database**, para el manejo y sincronización de información en tiempo real, incluyendo perfiles, solicitudes médicas, mensajes y geolocalización.
- **Firebase Cloud Messaging**, como soporte para notificaciones dirigidas a profesionales y pacientes.
- **Firebase Hosting y Firebase Functions**, para lógica de backend opcional y operación de la aplicación en entornos administrados. Si bien no se utilizaron de forma directa en esta versión de MedMatch, estos servicios fueron considerados dentro de la proyección arquitectónica del sistema debido a su potencial para alojar lógica avanzada en futuras iteraciones.

Realtime Database ofrece nodos estructurados en un árbol jerárquico, lo que se sincroniza de manera inmediata entre clientes, lo que permite mantener estados actualizados sin necesidad de un servidor intermedio. Este enfoque resulta apropiado para la naturaleza dinámica de la atención médica domiciliaria, donde la disponibilidad del profesional, el avance de las consultas y la comunicación entre usuarios requieren información actualizada.

En el ecosistema Firebase, la seguridad y el acceso a los datos se gestionaron mediante Firebase Security Rules, las cuales permitieron establecer restricciones de acceso basadas en autenticación e identidad del usuario. Estas reglas aseguran que únicamente usuarios autenticados puedan acceder a la información almacenada y que cada paciente o profesional de la salud solo pueda crear o modificar sus propios datos, evitando accesos no autorizados a información de terceros.

Este enfoque se alinea con los principios establecidos en la Ley N° 19.628 sobre protección de la vida privada, particularmente en lo relativo al tratamiento de datos sensibles, así como con los deberes de confidencialidad definidos en la Ley N° 20.584 sobre derechos y deberes del paciente. En el contexto del proyecto, los datos clínicos y personales fueron considerados información sensible, por lo que su acceso fue restringido mediante validación de identidad a nivel de base de datos.

No obstante, debido al alcance académico del proyecto y a las limitaciones de tiempo, no se implementó un modelo completo de control de acceso basado en roles clínicos ni flujos explícitos de consentimiento informado dentro de la aplicación. Estas medidas se identifican como una línea de mejora futura, especialmente considerando un eventual despliegue productivo del sistema, donde sería necesario reforzar las reglas de seguridad, eliminar permisos globales utilizados durante el desarrollo y formalizar mecanismos de consentimiento conforme a la normativa vigente.

3.1.5 Geolocalización y gestión de coordenadas

La ubicación geográfica es un componente fundamental para la asignación de profesionales en el contexto de atención domiciliaria. La geolocalización en MedMatch se sustenta en coordenadas GPS obtenidas desde el dispositivo móvil y almacenadas en Firebase. Estas coordenadas permiten identificar la posición tanto del paciente como del profesional.
Si bien el proyecto no integra APIs de cálculo de rutas o distancias, el uso de latitud y longitud permite aproximar la localización, identificar disponibilidad geográfica y facilitar la visibilidad entre usuarios. La gestión de coordenadas se complementa con validaciones básicas orientadas a asegurar coherencia y evitar datos inválidos que pudieran afectar la experiencia del usuario.

3.1.6 Seguridad de la información en aplicaciones de salud

Las aplicaciones orientadas al ámbito de la salud requieren un manejo especialmente cuidadoso de la información que procesan, ya que trabajan con datos personales y clínicos que pueden afectar directamente la privacidad de los usuarios. En Chile, este tipo de información se encuentra regulada por normativas como la Ley N° 19.628 y la Ley N° 20.584, las cuales establecen la necesidad de resguardar la confidencialidad y el uso adecuado de los datos asociados a la salud de las personas.

En el contexto de MedMatch, estas consideraciones influyen directamente en las decisiones de diseño del sistema. El control de acceso, la autenticación de usuarios y la separación de roles entre pacientes y profesionales no se abordan solo como aspectos técnicos, sino como mecanismos necesarios para evitar accesos indebidos y asegurar que la información sea utilizada únicamente en el contexto de la atención médica.

Desde esta perspectiva, la seguridad de la información se entiende como un componente transversal del sistema, que debe estar presente desde la arquitectura y no añadirse de forma posterior. Esto permite que la aplicación mantenga coherencia entre su funcionamiento técnico y las exigencias propias de un entorno donde la protección de los datos no es opcional, sino parte fundamental del servicio ofrecido.

3.1.7 Marco regulatorio para aplicación de salud

El desarrollo de aplicaciones tecnológicas orientadas al ámbito de la salud debe considerar el marco regulatorio vigente, dado que este define las condiciones bajo las cuales se pueden recopilar, almacenar y utilizar datos personales y clínicos. En Chile, existen normativas que, si bien no están dirigidas exclusivamente a aplicaciones móviles de salud, establecen lineamientos relevantes para su diseño e implementación.

Una de las principales normativas aplicables es la *Ley N° 19.628 sobre Protección de la Vida Privada*, que regula el tratamiento de datos personales y establece principios como la confidencialidad, el uso legítimo de la información y la protección frente a accesos no autorizados. En el contexto de aplicaciones de salud, esta ley resulta especialmente relevante debido a la naturaleza sensible de los datos tratados, como antecedentes médicos, información de contacto y localización de los usuarios.
Asimismo, la *Ley N° 20.584, que regula los derechos y deberes que tienen las personas en relación con acciones vinculadas a su atención de salud*, establece principios como la dignidad del paciente, la confidencialidad de la información clínica y el derecho a la privacidad. Si bien MedMatch no reemplaza la atención médica formal ni actúa como un sistema clínico institucional, su objetivo de facilitar la atención domiciliaria implica manejar información que podría considerarse parte del contexto asistencial, por lo que estos principios resultan pertinentes como referencia conceptual.

En el desarrollo de MedMatch, el enfoque principal estuvo puesto en el diseño de la arquitectura, la organización del sistema y la validación de los flujos funcionales, más que en una implementación exhaustiva de cumplimiento normativo. Esto se debe, principalmente, a que el proyecto se desarrolló en un contexto académico y como prototipo funcional, sin una puesta en producción real ni integración directa con instituciones de salud. No obstante, se consideraron buenas prácticas alineadas con la normativa vigente, como la autenticación de usuarios, la separación de roles, el control de accesos y la limitación del uso de la información a fines específicos del sistema.

Desde una proyección futura, la incorporación explícita del marco regulatorio se presenta como una línea de mejora relevante. En una versión productiva de MedMatch, sería necesario profundizar en aspectos como el consentimiento informado digital, la trazabilidad de accesos a la información, el almacenamiento seguro de datos clínicos y la adecuación a eventuales regulaciones específicas para plataformas de salud digital que puedan surgir en el país.

De este modo, aunque el marco legal no fue abordado como un eje central del desarrollo actual, su consideración resulta fundamental para la evolución del sistema hacia un entorno real de uso, y constituye un componente clave para garantizar la confianza, seguridad y legitimidad de una aplicación orientada a la atención de salud domiciliaria.

4 Diseño e implementación de la arquitectura

4.1 Diseño general de la arquitectura

La arquitectura propuesta para MedMatch se diseñó con el propósito de ofrecer una aplicación móvil capaz de sostener un flujo continuo de operaciones, con módulos independientes y una adecuada separación de responsabilidades. El objetivo principal fue garantizar que el sistema pudiera crecer sin comprometer su estabilidad, facilitando futuras mejoras, incorporación de nuevas funciones o ampliación del número de usuarios.
El diseño se estructuró en torno al patrón **Model–View–ViewModel (MVVM)**, dado que este enfoque permite aislar la lógica del negocio respecto a la interfaz de usuario, mejorando la mantenibilidad y facilitando la creación de componentes reutilizables. En el caso de MedMatch, este patrón resultó especialmente beneficioso debido a la variedad de pantallas, perfiles de usuario (paciente y profesional) y operaciones simultáneas que la aplicación requiere.

En el núcleo del diseño se incorporaron tres pilares funcionales:

1. **Una capa de presentación**, donde se encuentran las vistas y los widgets reutilizables, responsables de la experiencia de usuario.

2. **Una capa de lógica**, que agrupa los ViewModels y los servicios internos utilizados por ambos perfiles.

3. **Una capa de comunicación externa**, integrada por Firebase, sus servicios asociados y elementos provistos por el dispositivo móvil.

Este enfoque permitió ordenar el proyecto en componentes completamente identificables y, a la vez, reducir el acoplamiento entre módulos críticos como autenticación, gestión de citas, mensajería interna o geolocalización.

4.1.1 Consideraciones del cumplimiento normativo en la implementación

En el desarrollo e implementación de la arquitectura de MedMatch, el cumplimiento normativo fue abordado desde una perspectiva técnica y de diseño, considerando las implicancias asociadas al tratamiento de datos personales y de salud en el contexto chileno. Dado que el proyecto corresponde a una tesina de carácter académico, no se implementó un proceso legal completo que permitiera su uso inmediato en un entorno clínico real; sin embargo, sí se incorporaron decisiones técnicas orientadas a respetar los principios fundamentales establecidos por la normativa vigente.

En primer lugar, se definió un sistema de autenticación obligatoria para todos los usuarios, diferenciando los perfiles de pacientes y profesionales de la salud. Esta separación permitió limitar el acceso a la información según el rol del usuario, evitando exposiciones innecesarias de datos sensibles. Además, la arquitectura modular adoptada facilitó el aislamiento de los componentes encargados de la gestión de información clínica, lo que contribuye a un mayor control sobre su manipulación y a una reducción del impacto ante eventuales fallos.

Desde el punto de vista de la persistencia de datos, se utilizaron servicios cloud que incorporan mecanismos de seguridad nativos, complementados con reglas de acceso que restringen la lectura y escritura de información a usuarios autenticados. Estas reglas fueron diseñadas con el objetivo de asegurar que cada usuario solo pueda interactuar con los datos que le corresponden, alineándose con los principios de confidencialidad y acceso controlado definidos por la legislación nacional.

Si bien estas medidas no constituyen por sí solas un cumplimiento legal completo, establecen una base técnica coherente y preparada para futuras extensiones. En una eventual evolución del sistema hacia un uso productivo, la arquitectura implementada permite integrar mecanismos adicionales, tales como consentimiento explícito del usuario, políticas de privacidad visibles y trazabilidad de accesos, sin necesidad de una reestructuración profunda del sistema. De este modo, MedMatch fue concebido no solo como un prototipo funcional, sino como una plataforma con proyección realista hacia un cumplimiento normativo más amplio.

4.1.2 Diagrama General de la Arquitectura

La Figura 1 presenta la arquitectura general que sustenta el funcionamiento de MedMatch. Este diseño integra los elementos esenciales de la aplicación y establece cómo se relacionan entre sí los distintos módulos, servicios internos y componentes externos. El modelo se estructura en torno a una aplicación móvil desarrollada en Flutter, que opera como punto central de interacción para pacientes y profesionales, y que se comunica directamente con el ecosistema de servicios provistos por Firebase.

El diagrama refleja una separación clara entre la capa de presentación, la lógica interna y los servicios en la nube, permitiendo visualizar el flujo de información que sostiene operaciones como autenticación, sincronización de datos, mensajería y gestión de ubicaciones. Asimismo, se incluyen componentes del dispositivo móvil, como el módulo de geolocalización, que complementan las funcionalidades asociadas a la atención domiciliaria. Este esquema general funciona como una vista global del sistema,

ofreciendo una representación coherente de la arquitectura que orienta las decisiones de diseño abordadas en las siguientes secciones.

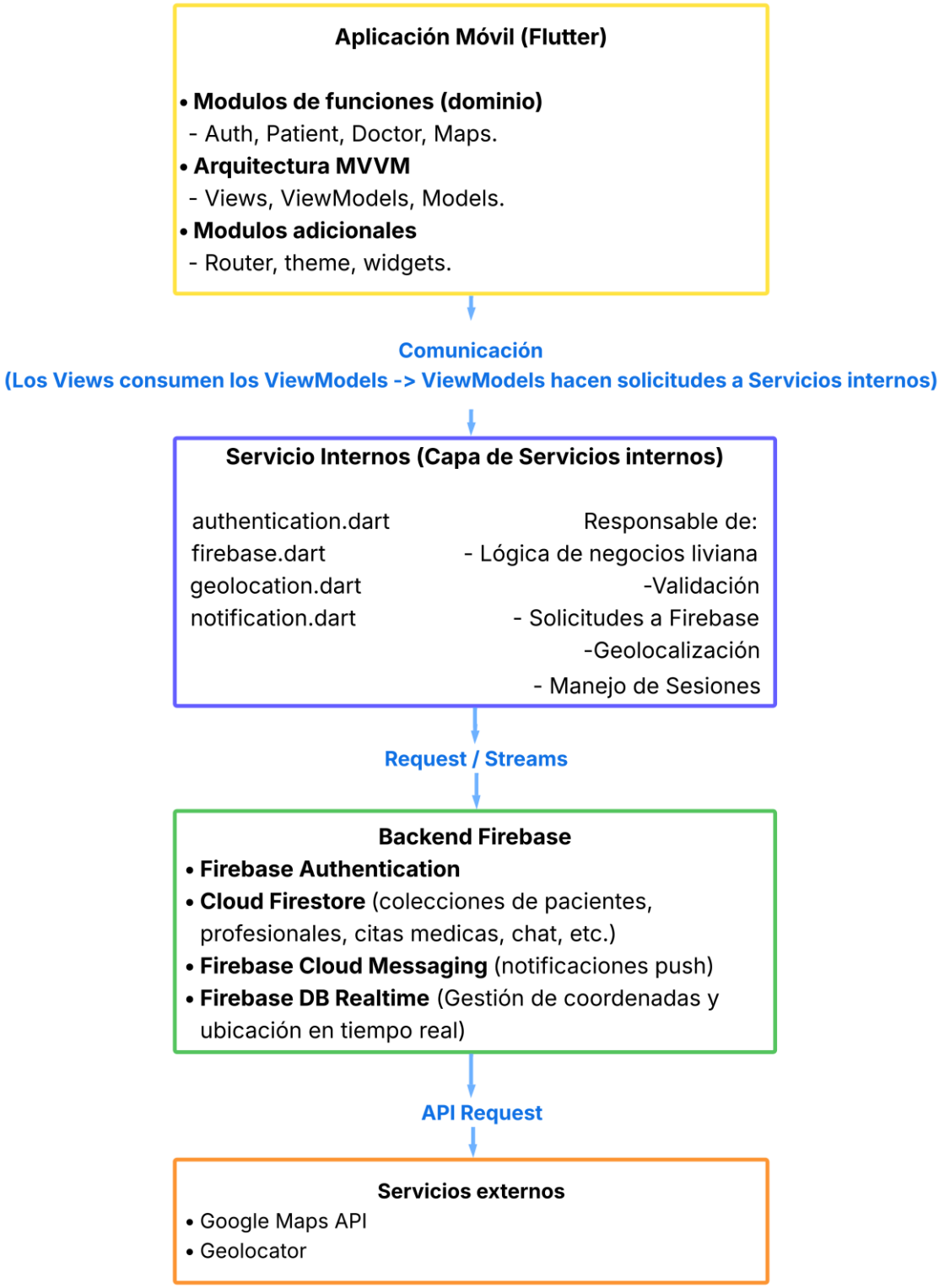


Figura 1. Diagrama General de la Arquitectura. Fuente: elaboración propia a través de lucidchart.

4.2 Patrones de diseño utilizados

Para la construcción del sistema se optó por una combinación de patrones descritos a continuación. Cada uno fue seleccionado en función de los requisitos de escalabilidad, claridad estructural y estabilidad del ciclo de vida dentro de la aplicación.

4.2.1 Patrón MVVM con base estructural

La adopción del patrón MVVM permitió que la aplicación manejara de manera ordenada los cambios en la interfaz sin que estos afectaran a la lógica de negocio. Cada módulo, tanto del paciente como del profesional, se construyó bajo esta misma estructura, lo cual favoreció la homogenización del código.

- **Model:** Representa los datos provenientes de Firebase, además de entidades internas como citas, usuarios y localizaciones.
- **View:** Pantallas y widgets encargados de mostrar información al usuario.
- **ViewModel:** Actúa como intermediario entre la vista y los datos; contiene la lógica de validación, solicitud de datos y gestión de estado.

Este ordenamiento redujo notablemente la duplicidad de código entre vistas similares en perfiles distintos, además de facilitar la incorporación de servicios compartidos como la geolocalización o la actualización de estados en tiempo real.

4.3 Organización Modular del sistema

Con el fin de obtener una estructura clara y coherente con los principios de mantenimiento y desacoplamiento, el proyecto se dividió en módulos independientes. Cada uno responde a un grupo específico de funcionalidades.

4.3.1 Módulo de Autenticación

Incluye las pantallas y la lógica necesaria para el registro e inicio de sesión tanto de pacientes como profesionales. Este módulo controla:

- Validación de credenciales.
- Persistencia de sesión.
- Redirección según el tipo de usuario.
- Uso de un *Auth Wrapper* para resolver el acceso a nivel global.

4.3.2 Módulos del paciente

Reúnen funcionalidades orientadas a la experiencia del paciente, tales como:

- Solicitud de citas.
- Seguimiento de consultas.
- Visualización del perfil del profesional.
- Chat interno.
- Historial médico.

- Pagos y favoritos.

Cada vista se encuentra enlazada a su ViewModel correspondiente, manteniendo un flujo independiente del profesional.

4.3.3 Módulo del profesional

Contienen herramientas destinadas al rol clínico, como:

- Agenda de citas.
- Confirmación o rechazo de solicitudes.
- Revisión de información del paciente.
- Emisión de reportes.
- Mensajería con pacientes.

Este módulo tiene comportamientos propios de gestión médica, permitiendo una interacción ordenada con las colecciones de Firebase.

4.3.4 Servicios internos Compartidos

Se implementaron como componentes independientes utilizados por ambos perfiles:

- Autenticación.
- Acceso a Firebase.
- Geolocalización.
- Notificaciones.
- Navegación.
- Actualizador de datos.

Esta decisión evitó duplicar lógica y facilitó que futuras ampliaciones se implementen desde un punto centralizado.

4.4 Integración con Firebase

La comunicación con Firebase se diseñó para que la aplicación interactúe con los servicios sin generar dependencia directa entre las vistas. A través de servicios internos, cada solicitud se abstrae utilizando funciones específicas que permiten:

- Leer y escribir datos en tiempo real.
- Autenticar usuarios.
- Recibir notificaciones.
- Actualizar estados en base a eventos.

Este diseño permitió que la aplicación no dependiera de una API externa adicional ni de servidores propios, lo cual reduce costos y simplifica la administración del sistema.

4.5 Diagrama de Componentes

La Figura 2 muestra la descomposición estructural de MedMatch en términos de los componentes que intervienen directamente en su funcionamiento. Este diagrama complementa la vista general de la arquitectura al profundizar en los módulos internos que conforman la aplicación y las relaciones que mantienen con los servicios cloud y las capacidades nativas del dispositivo móvil.
El sistema se organiza en torno a tres grupos principales de componentes:

1. **La aplicación Flutter**, donde residen las vistas, los ViewModels y la totalidad de módulos funcionales dirigidos a pacientes y profesionales;

2. **Los servicios internos**, implementados como capas de soporte reutilizables que permiten gestionar autenticación, navegación, notificaciones, geolocalización y operaciones sobre la base de datos; y

3. **El ecosistema Firebase**, responsable de ofrecer almacenamiento sincronizado, autenticación y mensajería en tiempo real.

Cada uno de estos elementos mantiene una relación claramente definida, lo que garantiza que la lógica de negocio permanezca desacoplada de la interfaz y que las funciones críticas puedan ampliarse sin afectar al resto del sistema. El diagrama también incorpora los servicios nativos del dispositivo, como el módulo de geolocalización, que proporcionan información esencial para la atención domiciliaria.

Esta representación permite identificar los límites de cada componente y visualizar cómo se integran para sostener los distintos flujos operativos de MedMatch.

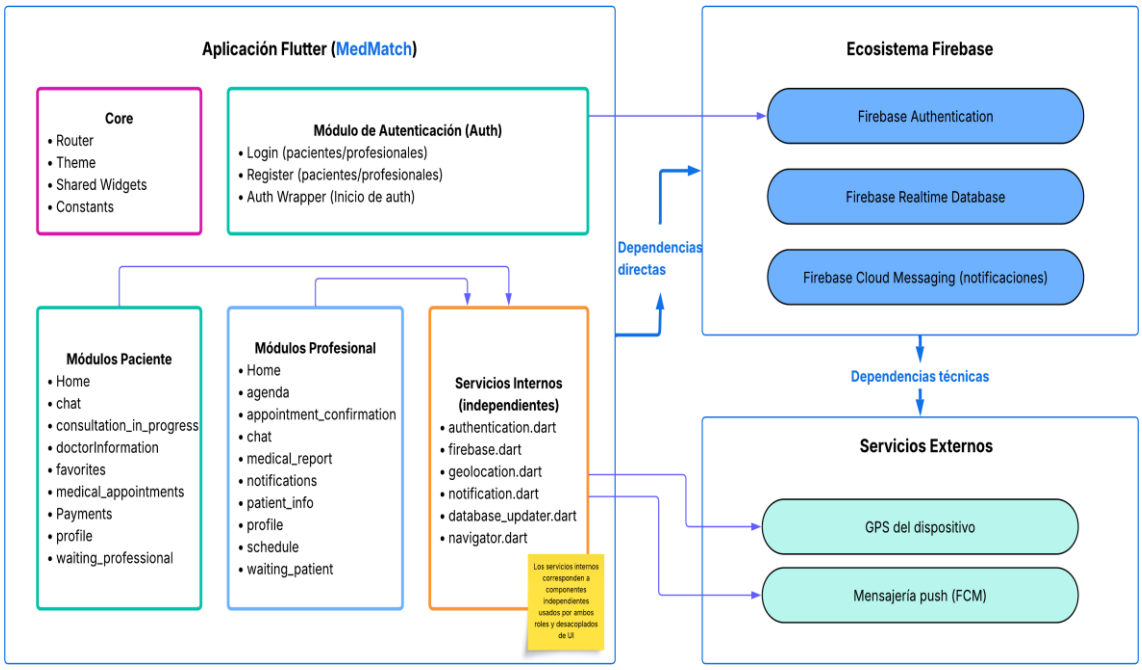


Figura 2. Diagrama de Componentes de MedMatch. Fuente: elaboración propia a través de lucidchart.

4.6 Diseño de la persistencia de datos dentro de la arquitectura

En el caso de MedMatch, la forma en que se almacenan y sincronizan los datos no se definió como un elemento aislado, sino como parte íntegra de la arquitectura del sistema. Debido a que la aplicación funciona con información que cambia constantemente, como solicitudes de atención, mensajes, disponibilidad de profesionales o coordenadas GPS, fue necesario adoptar un modelo de persistencia que respondiera a estos requerimientos sin comprometer el rendimiento ni la experiencia de uso.

La plataforma se construyó utilizando tres servicios principales de Firebase: **Firestore**, **Realtime Database** y **Firebase Authentication**. A primera vista podría parecer innecesario utilizar más de un motor de datos y un servicio de autenticación externo; sin embargo, esta decisión se tomó considerando tanto la naturaleza de la información como la manera en que cada módulo debe interactuar con ella. Desde una perspectiva arquitectónica, la elección de estos servicios no se basó en preferencias técnicas aisladas, sino en la necesidad de sostener un sistema que pudiera escalar, mantenerse en el tiempo y soportar interacciones simultáneas entre pacientes y profesionales.

Firestore se utilizó como base principal debido a su estructura flexible, que permite organizar colecciones y documentos de forma más cercana a los módulos de la aplicación. Esta estructura facilita la lectura por parte de los distintos servicios internos, mejora la separación entre roles y reduce el acoplamiento a nivel de vistas. Además, su capacidad para realizar consultas más específicas aporta orden y hace que el crecimiento futuro del sistema no dependa de ajustes complejos.

Por otro lado, **Firebase Realtime Database** se utilizó exclusivamente para la **gestión de coordenadas GPS**, debido a que el módulo de ubicación requiere actualizaciones prácticamente inmediatas y una sincronización constante entre dispositivos. En este contexto, Realtime Database permite propagar cambios en tiempos del orden de **segundos o fracciones de segundo**, lo que resulta adecuado para escenarios donde la posición del profesional debe reflejarse de manera casi instantánea en el dispositivo del paciente.
Al mantener este flujo separado del resto de los datos, se evita sobrecargar Firestore y se asegura que las operaciones de geolocalización no afecten a otros módulos de la aplicación.

Adicionalmente, **Firebase Authentication** se incorporó como el mecanismo central de gestión de identidades y control de acceso. Desde la perspectiva de la arquitectura, Authentication no es un simple componente para “iniciar sesión”: actúa como la puerta de entrada que condiciona qué datos y acciones están disponibles para cada tipo de usuario (paciente o profesional). Su integración permite aplicar reglas de seguridad en la capa de datos, asociar identificadores persistentes a los perfiles almacenados en Firestore y mantener sesiones seguras en la aplicación móvil. En resumen, Authentication provee la identidad fiable que utilizan tanto la capa de presentación como los servicios internos para recuperar, escribir y suscribirse a información relevante.

Basado en esta distribución de responsabilidades, Firestore para persistencia documental, Realtime Database para sincronización de ubicaciones e Authentication para control de acceso, no solo mejora el rendimiento general, sino que también abre la posibilidad de escalar componentes de manera independiente en el futuro. Por ejemplo, si se incorporaran nuevos mecanismos de seguimiento geográfico o un incremento notable en la mensajería, cada segmento de datos podría atenderse de forma aislada sin afectar la disponibilidad del resto del sistema.

Aunque este apartado se enfoca en la capa de persistencia y autenticación, su incorporación dentro del capítulo de arquitectura e implementación se justifica porque la base de datos y la gestión de identidades influyen directamente en cómo se estructuraron los módulos, cómo se comunican los servicios internos y qué modelo de interacción entre usuarios se pudo lograr. En otras palabras, la forma en que se almacenan, sincronizan y protegen los datos condiciona de manera importante las decisiones de diseño del proyecto, por lo que su análisis resulta necesario para comprender el funcionamiento completo de MedMatch.

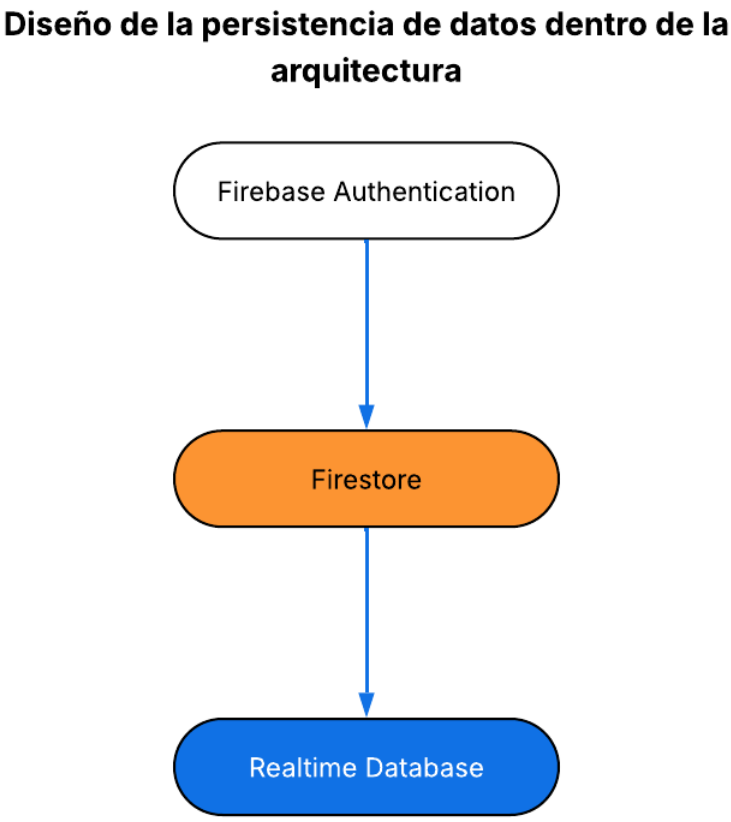


Figura 3. Diseño de la persistencia de datos dentro de la arquitectura. Fuente: elaboración propia a través de lucidchart.

4.7 Estrategias de escalabilidad del sistema

La proyección de MedMatch hacia un entorno de uso real exige que su arquitectura sea capaz de sostener un crecimiento progresivo en volumen de usuarios, operaciones simultáneas y complejidad funcional. Si bien la aplicación se encuentra actualmente enfocada en un entorno académico y de prueba, su diseño incorpora lineamientos que permitirían escalar el sistema sin comprometer su rendimiento o estabilidad.

Este apartado expone los mecanismos de escalabilidad vertical y horizontal aplicables al sistema, así como el modo en que Firebase, junto con la estructura modular de Flutter y el patrón MVVM, contribuye a soportar un aumento sostenido en la demanda.

4.7.1 Escalabilidad vertical del sistema

La *escalabilidad vertical* se refiere a la capacidad de un sistema para mejorar su rendimiento aumentando los recursos asignados a sus componentes actuales, sin modificar la estructura global. En

el caso de MedMatch, este tipo de escalabilidad se manifiesta principalmente en el lado del cliente y dentro de los servicios administrados por Firebase.

Incremento de capacidades desde Firebase
Firebase opera bajo un modelo administrado, lo que significa que la plataforma ajusta automáticamente recursos como:

- Capacidad de procesamiento.
- Número de conexiones simultáneas.
- Rendimiento de consultas.
- Velocidad de sincronización de datos.

Por ejemplo, si un módulo como *Appointments* comienza a recibir una mayor cantidad de solicitudes de lectura y escritura debido a un aumento de usuarios, Firebase asigna más recursos internos sin requerir intervención manual. Este mecanismo evita la necesidad de configurar servidores adicionales o escalar infraestructura de forma tradicional.

Actualización de módulos en Flutter
A nivel de la aplicación, la escalabilidad vertical se relaciona con mejoras en la eficiencia interna del código. Esto puede incluir:

- Optimización de ViewModels para minimizar reconstrucciones innecesarias.
- Uso de streams eficientes para actualizaciones en tiempo real.
- Carga diferida de pantallas según el flujo del usuario.
- Reutilización de widgets comunes para reducir el costo de renderizado.

Estas acciones permiten que la aplicación mantenga un tiempo de respuesta adecuado incluso si las pantallas deben manejar un mayor volumen de datos provenientes de Firebase.

4.7.2 Escalabilidad horizontal del sistema

La *escalabilidad horizontal* se centra en la capacidad del sistema para crecer mediante la distribución de la carga en distintos componentes, módulos independientes o instancias paralelas. Este enfoque es clave en sistemas móviles que pueden expandirse a nivel nacional o regional.

Modularidad como base del crecimiento
La arquitectura de MedMatch está formada por módulos independientes, tales como *Auth*, *Patient*, *doctor* y los servicios internos. Esta separación facilita que cada módulo pueda ampliarse sin afectar al resto. Por ejemplo:

- Si el módulo *chat* requiere incorporar historial, multimedia o cifrado punto a punto, estas funciones pueden añadirse sin modificar la lógica del registro de profesionales o del agendamiento.
- Si se decide incluir un nuevo rol (por ejemplo, paramédicos o kinesiólogos), solo se agregaría un módulo adicional dentro de la estructura de *features*, manteniéndose la consistencia del sistema.

Distribución de carga mediante Firebase
Firebase facilita la escalabilidad horizontal a través de una serie de mecanismos internos, entre ellos:

a) Particionado automático (sharding implícito)
Firestore y Realtime Database utilizan un sistema de distribución interna que fragmenta los datos automáticamente para optimizar su acceso. Aunque este proceso no es visible para el desarrollador, su

efecto es relevante. Un ejemplo concreto:

- Si una colección como *appointments* empieza a recibir miles de escrituras simultáneas, Firestore distribuye las operaciones entre distintos nodos para evitar congestión.
- Esto significa que, sin modificar la estructura del sistema, el rendimiento se mantiene estable incluso ante un volumen de tráfico elevado.

A diferencia de bases de datos tradicionales que requieren configurar particiones manuales, Firebase ejecuta este proceso de forma transparente, permitiendo que aplicaciones como MedMatch crezcan sin un rediseño temprano de la base de datos.

b) Replicación global automática
Firestore replica datos en varios centros de datos. Esto garantiza:

- Menor latencia para usuarios en distintas regiones.
- Continuidad del servicio en caso de caídas.
- Acceso concurrente sin conflictos importantes.

En un escenario donde MedMatch se expanda a múltiples ciudades, Firebase se encargaría de entregar el contenido desde el punto más cercano al usuario, reduciendo tiempos de espera.

4.7.3 Manejo de alta concurrencia en operaciones críticas

Existen módulos que podrían experimentar puntas de carga, como:

- Numerosas solicitudes de cita en horarios similares.
- Múltiples chats activos.
- Actualizaciones continuas de ubicación.
- Notificaciones a profesionales disponibles.

Para estos casos, la arquitectura MVVM y el uso de servicios internos actúan como reguladores, ya que:

- Los ViewModels procesan solo la información necesaria.
- Los servicios encapsulan la interacción con Firebase.
- La interfaz no se ve afectada por cambios masivos en los datos.

Un ejemplo concreto sería la geolocalización:
si muchos profesionales actualizan su ubicación al mismo tiempo, la aplicación únicamente procesa la información asociada al usuario activo, mientras Firebase gestiona el resto del tráfico.

4.7.4 Escalabilidad en comunicaciones y notificaciones

Firebase Cloud Messaging (FCM) permite enviar notificaciones dirigidas a miles de usuarios sin necesidad de servidores propios. En caso de que el sistema requiera aumentar la frecuencia o el volumen de notificaciones, Firebase distribuye la carga internamente.
Por ejemplo:

- Si un conjunto de profesionales debe recibir alertas simultáneas por nuevas solicitudes, FCM gestiona la entrega sin que la aplicación deba abrir conexiones adicionales.

Esto contribuye a que el sistema crezca sin reconfigurar infraestructura o modificar el código base.

4.7.5 Preparación para crecimiento a futuro

Aunque MedMatch aún se encuentra en fase de prototipo, su arquitectura incorpora buenas prácticas que facilitarían su expansión. Entre los escenarios contemplados destacan:

- Incorporación de nuevos módulos clínicos (fichas, recetas digitales, historial avanzado).
- Integración futura con servicios adicionales (pagos más complejos, geolocalización avanzada).
- Mayor número de usuarios en tiempo real.
- Manejo de más datos en colecciones críticas.

En todos estos casos, la estructura modular de Flutter, junto con la administración automática de Firebase, ofrece una base sólida para ampliar funcionalidades sin modificaciones disruptivas.

4.7.6 Diagrama visual de escalabilidad

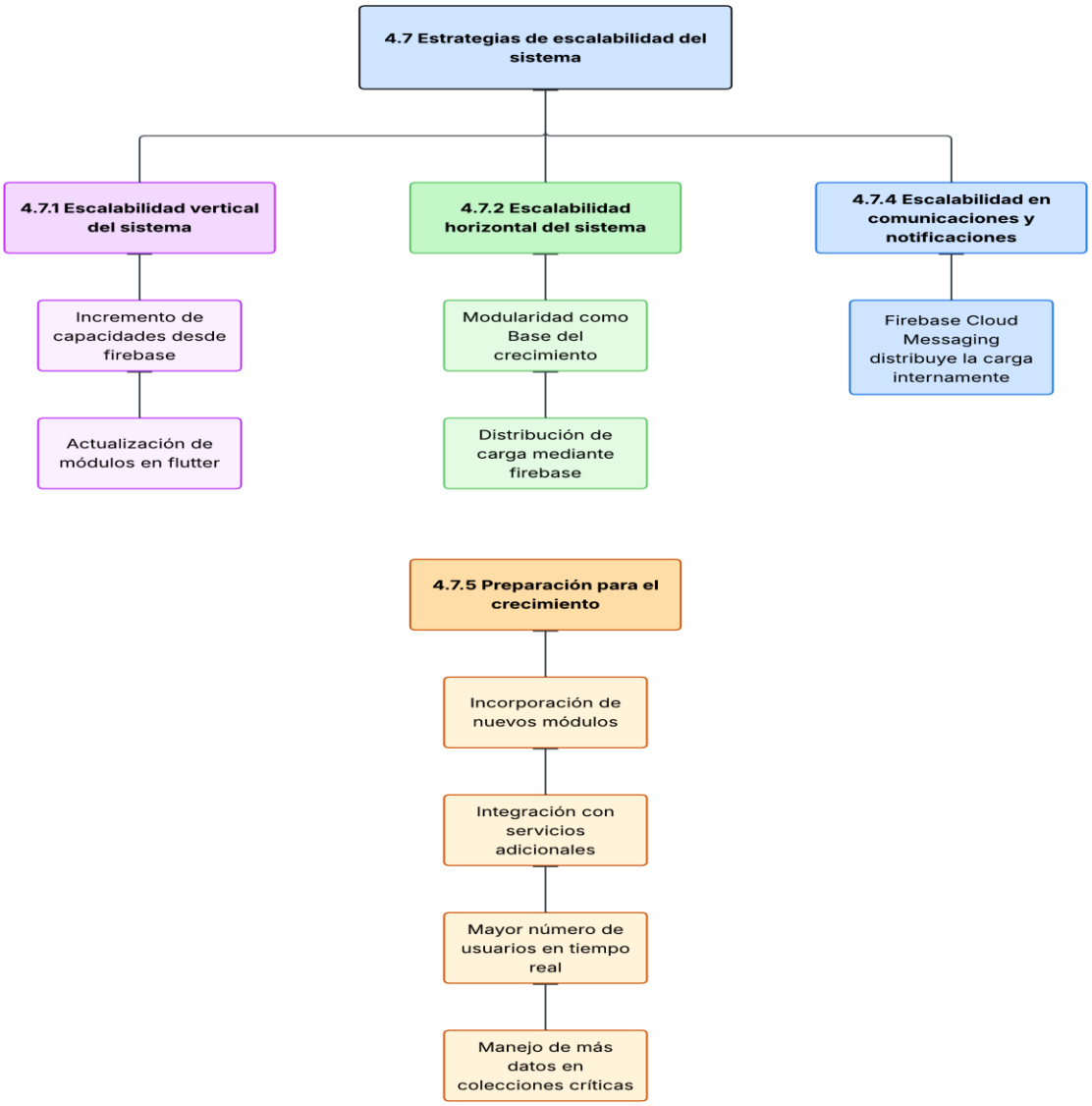


Figura 4. Diagrama visual de escalabilidad. Fuente: elaboración propia a través de lucidchart.

4.8 Detalles de la codificación y desarrollo

El desarrollo de MedMatch se construyó sobre una estructura modular organizada en torno al patrón MVVM. Esta elección permitió mantener un orden claro entre interfaz, lógica y servicios, evitando que la complejidad propia de una aplicación con múltiples roles y flujos terminara mezclando responsabilidades. A lo largo de esta sección, se describe cómo se implementaron estas decisiones en el código, mostrando fragmentos representativos que dan cuenta del modo en que la aplicación gestiona estados, rutas, servicios y comunicación con Firebase.

4.8.1 Estructura de carpetas y organización del proyecto

El proyecto se organizó siguiendo una estructura que divide el código por funcionalidades (features), además de agrupar los elementos reutilizables en una capa base (core). Esta estructura favoreció la identificación rápida de cada módulo, separando pantallas, ViewModels, modelos y servicios.

Una vista simplificada de la estructura es:

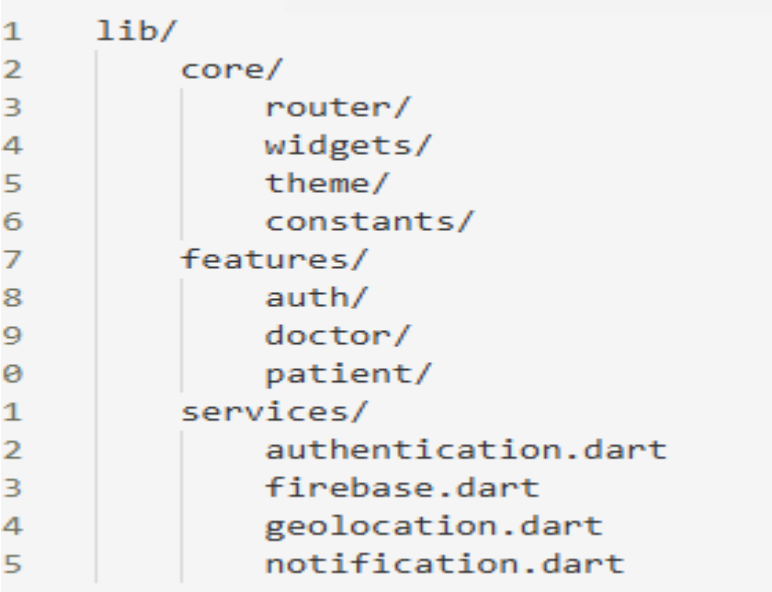


Figura 5. Estructura de carpetas. Fuente: elaboración propia a través de visual studio code.

Este orden permitió trabajar de forma aislada en cada módulo sin afectar al resto, especialmente útil en módulos extensos como el del profesional, que incluye agenda, chat, consultas en curso, reportes y manejo de pacientes.

4.8.2 Implementación práctica del patrón MVVM

La implementación del patrón MVVM se refleja en la manera en que cada pantalla se conecta con su ViewModel. Cada módulo mantiene un flujo similar: la vista solicita acciones, el ViewModel procesa la lógica y, cuando es necesario, se comunica con Firebase u otros servicios internos.

Un ejemplo claro se observa en el Home del Profesional, donde el ViewModel inicializa el estado recuperando el RUT del profesional y vincula la pantalla a un stream de solicitudes entrantes en Firebase:

```
void _init() {
  model.rutProfesional = _authService.rutUser;
  if (model.rutProfesional != null) {
    _solicitudesStream = _firebaseService.getIncomingRequests(model.rutProfesional!);
  }
```

Figura 6. Ejemplo de código en home profesional. Fuente: elaboración propia a través de visual studio code.

Con estas líneas, la pantalla queda automáticamente sincronizada con la actividad del sistema. Cada vez que entra una solicitud nueva, el stream se actualiza sin intervención extra del usuario.
Este tipo de sincronización automática es uno de los motivos principales por los que MVVM resultó adecuado para MedMatch: las vistas no necesitan conocer cómo se obtienen los datos; únicamente reaccionan a los cambios emitidos por el ViewModel.

4.8.3 Manejo de estado y lógica interna

El ViewModel no solo maneja datos, sino también elementos temporales, validaciones y comportamientos propios de cada pantalla.
En el caso del profesional, por ejemplo, se implementó un modo de selección de área sobre el mapa para activar la atención inmediata:

```
void toggleSelectionMode() {
  model.isSelecting = !model.isSelecting;
  selectionModeNotifier.value = model.isSelecting;

  if (!model.isSelecting) {
    model.selectedPoint = null;
  }
  notifyListeners();
}
```

Figura 7. Ejemplo de código en home profesional. Fuente: elaboración propia a través de visual studio code.

Este fragmento resume una idea que se repite en toda la aplicación: las acciones de la interfaz únicamente notifican cambios, mientras que el ViewModel decide qué debe ocurrir internamente.
Además, en este mismo módulo se incorpora la lógica para activar la atención inmediata:

```
final pos = await _geolocationService.getCurrentLocation();
model.currentLocation = pos;
```

Figura 8. Ejemplo de código en home profesional. Fuente: elaboración propia en visual studio code.

Esta operación ilustra cómo el ViewModel actúa como puente entre la capa visual y los servicios externos, manteniendo la estructura limpia y ordenada.

4.8.4 Vistas con lógica mínima y responsabilidad centrada

Las vistas se diseñaron con la intención de evitar que la lógica afectara la presentación. Por esa razón, pantallas como *Login_patient_screen* delegan completamente el manejo del flujo de autenticación a su ViewModel.
Un ejemplo simple es la acción de login:

```
final success = await viewModel.login();

if (success) {
  Navigator.pushReplacementNamed(context, AppRoutes.homePatient);
}
```

Figura 9. Ejemplo de código en login patient. Fuente: elaboración propia a través de visual studio code.

La vista no valida formularios extensos, no se comunica con Firebase y tampoco administra estados complejos.
Únicamente responde a un resultado booleano proveniente del ViewModel. Este enfoque permitió mantener pantallas limpias y más fáciles de modificar sin riesgo de romper la lógica interna.

4.8.5 Servicios internos desacoplados

Los servicios internos cumplen un rol central en la arquitectura, ya que encapsulan operaciones especializadas que se utilizan en distintos módulos.
Entre estos servicios destacan:

- Autenticación (Firebase Auth).
- Acceso a datos (Realtime Database y Firestore).
- Geolocalización.
- Notificaciones locales.
- Navegación centralizada.

Uno de los ejemplos más representativos es el servicio de notificaciones locales, que escucha cambios en Firestore para generar alertas en el dispositivo:

```
_subscription = FirebaseFirestore.instance
  .collection('doctors')
  .doc(userId)
  .collection('AtencionInmediata')
  .snapshots()
  .listen((snapshot) {
    for (final change in snapshot.docChanges) {
      if (change.type == DocumentChangeType.added) {
        NotificationService.showNotification(
          'Nueva solicitud recibida',
          'Un paciente ha solicitado atención.'
        );
      }
    }
  });
```

Figura 10. Ejemplo de código en notifications.dart. Fuente: elaboración propia en visual studio code.

Este diseño logró dos objetivos importantes:

1. Evitar que cada pantalla se conectara manualmente a Firebase.
2. Mantener notificaciones coherentes y centralizadas, independientemente del módulo que las requiriera.

4.8.6 Gestión de rutas y navegación del sistema

La aplicación utiliza un archivo de rutas centralizado que define todos los recorridos posibles dentro del sistema.

Esta decisión ayudó a estandarizar la navegación y a evitar errores comunes relacionados con nombres de rutas duplicados o navegación improvisada.
El archivo *app_router.dart* define cada ruta con un nombre único y su respectiva pantalla:

```dart
case AppRoutes.consultaEnCurso:
  final args = settings.arguments as Map<String, dynamic>? ?? {};
  return MaterialPageRoute(
    builder: (_) => ConsultaEnCurso(
      professionalName: args['professionalName'] ?? 'Profesional',
      consultaId: args['consultaId'] ?? '',
    ),
  );
```

Figura 11. Ejemplo de código en app_router.dart. Fuente: elaboración propia a través de visual studio code.

Este ejemplo muestra cómo la aplicación recibe argumentos desde cualquier módulo sin depender de estructuras fijas, lo que permite que las pantallas puedan reutilizarse en distintos contextos.

4.8.7 Integración con Firebase mediante services especializados

El acceso a Firebase se realiza de forma indirecta, siempre a través de funciones encapsuladas en FirebaseService.
Esto evita acoplar directamente la vista o el ViewModel con la API de Firebase, y además facilita cambiar implementaciones en el futuro sin alterar el resto del código.

Esta llamada permite que Firestore almacene la información sin que el ViewModel conozca detalles de colecciones, índices o conversiones.

```dart
await _firebaseService.addAtencionInmediata(
  model.rutProfesional!,
  Atencioninmediata(
    rut: model.rutProfesional!,
    lugar: model.selectedPoint!,
    radio: model.attentionRadius,
    duracion: model.durationHours.toInt(),
    tipoUsuario: 'profesional',
  ),
);
```

Figura 12. Ejemplo de código en Firebase.dart. Fuente: elaboración propia en visual studio code.

4.8.8 Reflexión final sobre implementación

La estructura aplicada permitió desarrollar una aplicación con múltiples perfiles, flujos paralelos y actualizaciones constantes sin comprometer la claridad del código. Separar la lógica de la interfaz, junto con encapsular los servicios internos y centralizar las rutas, resultó clave para mantener una base de código escalable y sencilla de mantener, incluso considerando la dinámica compleja del ámbito de salud.

4.9 Justificación técnica de las tecnologías utilizadas

La selección de las tecnologías que dan forma a MedMatch no fue un proceso improvisado, sino que respondió a la necesidad de construir una aplicación que fuera estable, rápida y fácil de mantener a largo plazo. Durante el desarrollo consideramos distintas alternativas para cada capa del sistema, evaluando aspectos como la curva de aprendizaje, la integración con servicios externos, el rendimiento final y la capacidad de crecer sin que esto implique rehacer todo desde cero.

Este apartado explica por qué Flutter, Firebase, MVVM y el conjunto de herramientas asociadas fueron finalmente las opciones más adecuadas para este proyecto.

4.9.1 Elección de Flutter como framework principal

Flutter se transformó en la base del desarrollo por varias razones prácticas. La primera fue la posibilidad de trabajar con un único código fuente para Android e iOS. Esto simplificó muchísimo el proceso, sobre todo considerando que el equipo tenía tiempos acotados y requería avanzar de manera ordenada.

Otro punto que pesó bastante fue su arquitectura orientada a widgets. En la práctica, esto permite construir pantallas completas componiendo pequeñas piezas reutilizables, lo que a la larga reduce errores y facilita mantener un estilo visual coherente. Además, Flutter ofrece un rendimiento casi nativo gracias a su motor propio de renderizado, esto es especialmente útil en secciones como el mapa, donde la aplicación necesita mostrarse fluida incluso si está cargando ubicaciones o actualizando coordenadas en tiempo real.

También influyó el ecosistema de paquetes disponibles. Herramientas como geolocator, provider o los SDK oficiales de Firebase hicieron posible integrar funciones complejas sin “reinventar la rueda”. Esto permitió enfocarnos en resolver los problemas reales del proyecto y no en construir librerías desde cero.

4.9.2 Uso de Firebase como plataforma cloud

Se optó por Firebase principalmente porque entregaba un conjunto de servicios integrados que resolvían casi todo lo que la aplicación necesitaba: autenticación, persistencia, sincronización en tiempo real, funciones servidor y notificaciones push.

A diferencia de montar un backend propio, Firebase permitió avanzar más rápido y con menos infraestructura. Esto encajaba bien con un proyecto académico, pero también con un escenario real donde se busca reducir costos y tener un sistema que funcione incluso con equipos pequeños.

Otra ventaja importante fue la seguridad. Las reglas de Firebase permiten controlar exactamente quién puede leer o escribir cada parte de la base de datos, lo que es muy útil considerando que MedMatch maneja perfiles, coordenadas y datos de interacción entre profesionales y pacientes. La plataforma también ofrece cifrado nativo y monitoreo, lo que evita tener que implementar estas medidas desde cero.

Finalmente, su escalabilidad automática, tanto en Firestore como en Realtime Database, asegura que, si la aplicación creciera en cantidad de usuarios, no sería necesario migrar a otro servicio inmediatamente. Firebase absorbe aumentos de tráfico sin que los desarrolladores deban preocuparse por servidores, balanceadores o almacenamiento adicional.

4.9.3 Justificación del uso del patrón MVVM

MVVM fue elegido para mantener el proyecto ordenado. La aplicación tiene muchos módulos distintos (autenticación, citas, agenda, chat, geolocalización, etc.), y mezclar la lógica con las pantallas habría generado un código difícil de mantener. MVVM permitió dividir bien las responsabilidades:

- **Model:** Estructura los datos que vienen desde Firebase.
- **View:** Se encarga de mostrar la información al usuario.
- **ViewModel:** Se posiciona en el medio, actualizando la vista y comunicándose con los servicios.

Gracias a esta separación, el equipo pudo trabajar en distintas partes de la aplicación de manera paralela, sin generar conflictos entre componentes ni interferencias en el desarrollo de otros módulos. Este patrón también ayudó a reutilizar funciones entre el perfil de paciente y el perfil del profesional, lo que redujo mucho la duplicación y facilitó agregar mejoras sin alterar módulos completos.

Además, MVVM funciona muy bien con Flutter porque sus vistas son declarativas y reaccionan a los cambios del estado. Esto se complementa perfecto con los ViewModels que exponen información observable, permitiendo que las pantallas se actualicen en tiempo real de forma natural.

4.9.4 Motivo detrás del uso de Firestore y realtime database

Aunque podría parecer extraño utilizar dos bases de datos dentro de una misma aplicación, en MedMatch esta decisión tiene una lógica clara: cada base resuelve un problema distinto.

- **Firestore** es ideal para datos estructurados que no cambian cada segundo: perfiles, historial, publicaciones, favoritos, agenda médica, etc. Ofrece consultas potentes, escalabilidad automática y documentos fáciles de manejar.
- **Realtime Database**, por el contrario, es muy rápida para sincronización continua, por lo que funcionó mejor para almacenar las coordenadas de usuarios en movimiento. Su forma de operar permite recibir cambios casi al instante, algo fundamental para funciones que dependen de la ubicación.

Esta combinación no solo permitió mejorar el rendimiento general, sino también optimizar costos y simplificar la lógica. Cada servicio hace lo que mejor sabe hacer.

4.9.5 Integración de Firebase Authentication

La autenticación es uno de los pilares de MedMatch, ya que determina qué tipo de usuario ingresa y qué permisos tiene dentro de la aplicación. *Firebase Authentication* se eligió porque ofrece un manejo de sesiones seguro, estable y sin necesidad de implementar un backend propio para validar contraseñas.

Además, se integra naturalmente con *Provider* y con los *wrappers de autenticación* dentro del flujo de Flutter, permitiendo redirigir al usuario según su rol (paciente o profesional) de forma muy ordenada. También simplifica aspectos que normalmente son complejos, como recuperación de contraseña, manejo de errores en login o persistencia automática de sesiones.

En términos prácticos, usar Firebase Auth evitó una gran cantidad de trabajo manual y permitió enfocarse en lo más importante: la experiencia del usuario dentro de la plataforma.

5 Pruebas y Validación

El proceso de pruebas cumplió un rol fundamental en el desarrollo de MedMatch, ya que permitió verificar que las funcionalidades implementadas respondieran a los objetivos del proyecto y que el sistema se comportara de manera estable en diferentes escenarios de uso. Debido a que se trata de una aplicación móvil que depende continuamente de servicios externos como Firebase, la validación no solo consideró aspectos visuales, sino también flujos internos, manejo de estados, operaciones sobre la base de datos y gestión de errores.

Para organizar este trabajo, las pruebas se dividieron en distintos niveles: pruebas unitarias, pruebas funcionales, pruebas de integración, pruebas de experiencia de usuario y pruebas específicas asociadas a la sincronización en tiempo real. Esta clasificación permitió abordar cada módulo de manera ordenada y detectar problemas tempranos, evitando que avanzaran hasta etapas más complejas del desarrollo.

5.1 Criterios de aceptación generales

Para poder validar correctamente el funcionamiento de MedMatch, se definieron una serie de criterios de aceptación transversales a todo el sistema. Estos criterios sirvieron como guía para determinar si cada módulo cumplía con lo esperado desde el punto de vista del usuario y si el comportamiento general era consistente con los objetivos del proyecto.

Los criterios considerados fueron los siguientes:

- **Coherencia entre vistas y datos:**
  La información mostrada debía corresponder estrictamente a lo almacenado en Firebase, evitando cualquier discrepancia perceptible para el usuario, especialmente en módulos sensibles como la agenda, las solicitudes o la ubicación.

- **Flujo continuo para el usuario:**
  El sistema debía permitir moverse entre sus secciones sin interrupciones, evitando caídas inesperadas, demoras excesivas o redirecciones que confundieran al usuario.

- **Sincronización correcta en tiempo real:**
  En los módulos que dependen de información en vivo, como la ubicación y el chat, los cambios realizados debían reflejarse en la interfaz del usuario en un tiempo acotado, sin bloqueos ni desactualizaciones visibles. Como referencia práctica, se consideró aceptable que las actualizaciones se propagaran en un plazo inferior a dos segundos bajo condiciones normales de conectividad.

- **Validaciones mínimas satisfactorias:**
  El registro, inicio de sesión y actualización de datos personales debían bloquear entradas incorrectas (RUT, correos mal escritos, contraseñas débiles, etc.).
  El sistema debía impedir el ingreso de información incompleta o con formatos inválidos, especialmente en campos críticos como correo, contraseña y RUT.

- **Respuestas claras en caso de error:**
  Ante cualquier problema, ya fuera de conexión o de validación, el sistema debía mostrar mensajes entendibles para el usuario, sin códigos técnicos innecesarios.

- **Compatibilidad básica en dispositivos móviles:**
  La aplicación debía funcionar de forma estable en teléfonos con características promedio, sin

exigir hardware de gama alta.

- **Consistencia con los requisitos iniciales:**
  Cada módulo implementado debía cumplir su propósito original: solicitar una cita, aceptarla, iniciar un chat, mostrar la ubicación, etc., sin desviarse de la funcionalidad acordada a nivel del proyecto.

Con estos criterios definidos, fue posible evaluar cada sección de manera organizada, considerando tanto la experiencia del usuario como la responsabilidad técnica de cada módulo.

5.2 Criterios de aceptación por módulo

Además de los criterios generales, fue necesario definir requisitos específicos para cada módulo de MedMatch, ya que cada uno aborda funcionalidades distintas y depende de elementos técnicos diferentes (geolocalización, Firebase Auth, Firestore, Realtime Database, etc.). Esto permitió evaluar cada parte de manera más precisa y detectar errores particulares sin mezclar comportamientos de otros componentes.

A continuación, se detallan los criterios principales por módulo:

A) Módulo de Autenticación (Firebase Auth)

- El usuario debe poder registrarse solo si completa todos los campos con datos válidos.
- El inicio de sesión debe redirigir de forma automática al perfil correspondiente (paciente o profesional).
- En caso de credenciales incorrectas, se deben mostrar mensajes claros sin información sensible.
- El Auth Wrapper debe mantener la sesión activa mientras no se cierre manualmente.

B) Módulo de Solicitudes y Citas

- El paciente debe poder solicitar una cita indicando fecha, hora y motivo.
- El profesional debe visualizar nuevas solicitudes sin necesidad de actualizar la aplicación.
- Las solicitudes deben cambiar de estado (pendiente → aceptada/rechazada) sin inconsistencias.
- Las citas aceptadas deben aparecer correctamente en la agenda del profesional.

C) Módulo de Agenda

- La agenda debe cargar todas las citas del profesional sin duplicados ni registros incompletos.
- Debe diferenciar claramente entre citas futuras, pasadas y realizadas.
- Los cambios realizados en citas deben reflejarse en la agenda sin reiniciar la aplicación.
- La vista debe ser coherente entre pantallas (lista detallada y confirmaciones).

D) Módulo de Ubicación en Tiempo Real

- El profesional debe poder compartir su ubicación y esta debe reflejarse en la aplicación del paciente en un tiempo acotado. Como referencia de validación, se consideró aceptable una latencia de actualización inferior a dos segundos en entornos de prueba controlados. Durante las evaluaciones realizadas en emulador se observaron retrasos puntuales, los cuales se atribuyen principalmente a las limitaciones propias del entorno de simulación y no a sobrecarga del sistema, considerando que la gestión de coordenadas se implementó de forma exclusiva mediante Firebase Realtime Database.
- El paciente debe ver la ubicación actualizada sin que el mapa se quede congelado.
- Los marcadores deben poder seleccionarse correctamente.
- Las coordenadas almacenadas en Realtime Database deben sobrescribirse y no generar entradas duplicadas.

E) Módulo de Chat

- Los mensajes deben enviarse y recibirse en tiempo real.
- No deben aparecer mensajes vacíos o duplicados.
- El chat debe identificar correctamente quién envía cada mensaje.
- El historial debe cargarse sin errores al abrir la conversación.

F) Módulo de Informes Médicos

- La información del paciente debe cargarse completa al generar un informe.
- El informe debe poder guardarse sin errores.
- Los campos obligatorios no deben quedar en blanco.
- La vista debe actualizarse correctamente al regresar desde otra pantalla.

G) Módulo de Perfil y Configuración

- El usuario debe poder editar sus datos personales sin afectar otros módulos.
- Los cambios deben guardarse correctamente en Firebase.
- El sistema debe validar correos y direcciones antes de confirmar la edición.

H) Módulo de Favoritos

- Los profesionales marcados como favoritos deben guardarse correctamente.
- El paciente debe poder ver su lista sin demora.
- Los elementos deben poder eliminarse sin inconsistencias.

Estos criterios fueron la base para construir los casos de prueba y evaluar la aplicación de forma ordenada. Cada módulo se revisó con estos puntos en mente, lo que ayudó a identificar errores específicos y a priorizar correcciones antes de avanzar hacia la validación final del sistema.

5.3 Pruebas Funcionales

Las pruebas funcionales se realizaron directamente sobre la aplicación en ejecución, evaluando cada módulo desde la perspectiva del usuario final. Este tipo de pruebas fue esencial para confirmar que las funcionalidades realmente respondían como se esperaba y que los flujos principales podían completarse sin errores, tanto desde el perfil del paciente como del profesional.

Durante estas pruebas se revisaron escenarios completos de uso, como:

- Registro e inicio de sesión en ambos perfiles utilizando Firebase Authentication.
- Envío de solicitudes por parte del paciente, junto con su respectiva visualización por el profesional.
- Aceptación o rechazo de solicitudes, verificando que los cambios se reflejaran de inmediato en la agenda.
- Revisión del módulo de ubicación, asegurando que el mapa actualizara la posición sin necesidad de recargar la pantalla.
- Envío y recepción de mensajes dentro del chat interno, comprobando que no existiera retraso perceptible.

Durante esta etapa se identificaron algunos problemas derivados del reordenamiento del proyecto hacia la arquitectura MVVM. Por ejemplo, en el módulo de informes la información del paciente no se cargaba de forma correcta y el formulario no permitía guardar el archivo final.
En el módulo de ubicación el marcador no respondía al hacer clic, lo cual fue corregido. También se detectó que algunas citas se registraban directamente como “realizadas”, incluso cuando eran citas futuras, por lo que no aparecían en la sección correspondiente de la agenda. Estos problemas quedaron registrados y forman parte de los ajustes previstos para la siguiente iteración del proyecto.

A pesar de estas dificultades puntuales, las pruebas funcionales permitieron demostrar que el sistema lograba cumplir con los flujos principales y que la mayoría de las funciones respondían de manera estable.

5.3.1 Pruebas unitarias

Las pruebas unitarias se orientaron principalmente a la validación de la lógica interna implementada en los ViewModels, considerando que estos concentran el manejo del estado, las validaciones y la interacción con los servicios de datos. El objetivo de estas pruebas fue asegurar que los procesos críticos del sistema se ejecutaran correctamente de forma independiente a la interfaz gráfica.

Entre los casos abordados se incluyó la validación de datos de entrada durante el registro de usuarios, la correcta transición de estados en las solicitudes médicas y la carga coherente de información en módulos como agenda e informes. Asimismo, se evaluaron escenarios de error controlado, tales como respuestas incompletas desde la base de datos o intentos de guardar información inválida.

Si bien no se implementó una cantidad extensa de pruebas automatizadas, este enfoque permitió detectar inconsistencias tempranas en la lógica del sistema y facilitó el proceso de depuración, especialmente durante la reorganización del proyecto hacia la arquitectura MVVM.

5.4 Pruebas de integración

Las pruebas de integración se centraron en evaluar cómo interactuaban entre sí los distintos módulos del sistema, considerando que MedMatch combina Flutter, MVVM, Firebase Authentication, Cloud Firestore y Realtime Database. Esta etapa fue fundamental, ya que muchos errores no aparecían de manera aislada, sino únicamente cuando varios componentes trabajaban en conjunto.

Entre los casos revisados se encuentran:

- Interacción entre los ViewModels y Firebase durante operaciones críticas como registro, carga de datos personales o envío de solicitudes.
- Sincronización entre Firestore y Realtime Database, especialmente en el caso del módulo de ubicación.
- Flujo completo entre pantallas que dependen de información en vivo, como agenda → chat → informe del paciente.
- Funcionamiento del Auth Wrapper que redirige al usuario según su rol una vez autenticado.
- Comprobación del orden correcto de lectura y escritura al aceptar una solicitud o al actualizar la ubicación.

Estas pruebas permitieron descubrir comportamientos inesperados cuando la aplicación recibía cambios muy rápidos desde Firebase, lo que obligó a ajustar algunos listeners y mejorar el manejo de estados dentro de los ViewModels. También se detectó que ciertos datos no llegaban a tiempo a la vista debido al orden en que se inicializaban los controladores, situación propia del cambio de arquitectura que se está realizando.

Los ajustes derivados de estas pruebas ayudaron a estabilizar la comunicación interna del sistema y mejorar la fluidez de la aplicación durante su uso normal.

5.5 Pruebas de usabilidad y sincronización en tiempo real

Dado que MedMatch está orientada a usuarios con distintos niveles de experiencia digital, se realizaron pruebas informales de usabilidad con estudiantes y personas ajenas al proyecto. El objetivo era

comprobar si la interfaz resultaba clara sin necesidad de instrucciones adicionales.

Estas pruebas permitieron ajustar aspectos como:

- Orden y distribución de botones en las pantallas principales.
- Mensajes de error más descriptivos, especialmente en el registro y en el inicio de sesión.
- Navegación entre módulos que originalmente requería demasiados pasos.
- Tiempos de carga que podían reducirse eliminando consultas innecesarias.

Además, se dedicó un apartado especial a la sincronización en tiempo real, debido a que es uno de los componentes más sensibles de la aplicación. En este punto se evaluó:

- Actualización correcta de la ubicación del profesional desde distintos dispositivos.
- Comportamiento del mapa durante pérdidas temporales de señal.
- Estabilidad del chat ante conexiones inestables.
- Reacción de los listeners cuando se modificaba más de un dato a la vez.

Estas pruebas revelaron ciertos inconvenientes relacionados con la frecuencia de actualización y con la forma en que la aplicación trataba las reconexiones. Varias de estas situaciones se corrigieron ajustando la estructura de la base de datos y reduciendo operaciones innecesarias.

5.6 Validación final del sistema

Una vez que todos los módulos principales estuvieron implementados y se resolvieron los problemas detectados en las pruebas previas, se realizó una validación final del sistema. Esta etapa consistió en recorrer el flujo completo de MedMatch desde el punto de vista del usuario, siguiendo el mismo camino que tendría un paciente y un profesional en un escenario real.

Como complemento a la evaluación cualitativa, se consideraron métricas básicas de desempeño obtenidas durante las pruebas funcionales y de integración. En particular, se observó que los módulos que dependen de sincronización en tiempo real, como chat y geolocalización, reflejaron actualizaciones en plazos inferiores a dos segundos bajo condiciones normales de conectividad.

De igual forma, los flujos principales del sistema, tales como el registro, solicitud de citas, aceptación por parte del profesional y visualización en agenda, pudieron completarse de manera continua durante sesiones prolongadas de uso, sin presentar fallos críticos ni interrupciones inesperadas. Estas mediciones permitieron contar con una referencia objetiva del comportamiento del sistema en su estado actual.

Durante esta validación se evaluó:

- Creación y autenticación de usuarios.
- Solicitud de citas por parte del paciente.
- Aceptación y gestión de solicitudes por parte del profesional.
- Visualización de la agenda, tanto para citas futuras como para realizadas.
- Comunicación mediante el chat interno.
- Funcionamiento del módulo de ubicación en vivo.
- Generación y revisión del informe del paciente.

Desde el punto de vista normativo, durante la validación final se reconoció que varias de las funcionalidades evaluadas involucran el tratamiento de datos sensibles, según lo definido en la Ley N° 19.628, particularmente aquellos asociados a información de salud y datos de localización. En este contexto, módulos como el chat interno, la gestión de informes médicos y la visualización de ubicación

en tiempo real implican riesgos adicionales que, en un escenario de uso real, requieren medidas de protección reforzadas.

Si bien la arquitectura del sistema considera mecanismos de autenticación y control de acceso mediante Firebase Authentication y reglas de seguridad en la base de datos, no se implementó cifrado de extremo a extremo en el módulo de mensajería ni flujos de consentimiento explícito diferenciados para el uso de geolocalización, aspectos que podrían ser exigibles conforme a los artículos 4 y 10 de la Ley N° 19.628, así como a los artículos 12 y 13 de la Ley N° 20.584 sobre derechos y deberes del paciente.

En consecuencia, estas consideraciones se identifican como líneas de mejora para una versión futura de MedMatch, orientadas a fortalecer el cumplimiento normativo, incorporando mecanismos de consentimiento informado específico, mayor control sobre el tratamiento de datos sensibles y técnicas de protección adicionales para la comunicación entre usuarios, especialmente en aquellos módulos que manejan información clínica o datos de localización.

Finalmente, el sistema presentó un comportamiento estable y alineado con los objetivos definidos para esta etapa del desarrollo. Durante la validación se comprobó que los flujos principales, como registro de usuarios, gestión de citas, uso del chat y actualización de ubicación, pudieron completarse de forma continua, sin caídas de la aplicación ni pérdida de información.

Asimismo, los módulos que dependen de sincronización en tiempo real reflejaron actualizaciones en tiempos inferiores a dos segundos bajo condiciones normales de conectividad, y las operaciones sobre Firebase se realizaron sin generar inconsistencias en los datos. Estos resultados permiten afirmar que la arquitectura y la integración de servicios funcionan de manera consistente en el estado actual del sistema.

No obstante, se identificaron mejoras que serán abordadas en iteraciones posteriores, particularmente en el módulo de informes y en el registro automático de citas, las cuales no afectan la validez general de la solución implementada.

5.7 Validación de criterios legales y normativos

Considerando que MedMatch gestiona información personal y datos asociados a la atención de salud, se incorporaron pruebas orientadas a verificar el cumplimiento técnico de criterios legales básicos, en concordancia con la normativa chilena vigente en materia de protección de datos y derechos de los pacientes.
En este contexto, se validó que el acceso a la información almacenada en el sistema estuviera restringido exclusivamente a usuarios autenticados, mediante el uso de Firebase Authentication y reglas de seguridad definidas en Firestore. Asimismo, se comprobó que cada usuario solo pudiera acceder y modificar sus propios datos, evitando la exposición de información a terceros no autorizados.

Estas verificaciones se alinean con los principios establecidos en la Ley N°19.628 sobre protección de la vida privada y la Ley N°20.584 sobre derechos y deberes del paciente. Si bien MedMatch corresponde a un prototipo académico y no a una aplicación en producción, este proceso de validación permite sentar una base técnica adecuada para una futura implementación que incorpore mecanismos formales de consentimiento informado y políticas de privacidad visibles para los usuarios.

6 Conclusiones

El desarrollo de MedMatch fue un proceso que, más allá de construir pantallas y conectar servicios, me permitió entender de forma práctica lo que significa diseñar un sistema pensando en su crecimiento real. Durante el trabajo confirmé que una aplicación no se sostiene únicamente porque "funcione", sino por la arquitectura que la respalda, por cómo se organiza su código y por las decisiones que permiten que evolucione sin volverse un proyecto frágil o difícil de mantener.

La adopción del patrón MVVM y una arquitectura modular por funcionalidades terminó siendo uno de los pilares más importantes del sistema. Gracias a esta estructura, la aplicación no solo se ordenó de mejor manera, sino que quedó preparada para crecer sin que cada cambio afectara a todo lo demás. Problemas que surgieron durante la reestructuración, como datos que no cargaban correctamente en el módulo de informes, citas que se marcaban como realizadas antes de tiempo o la interacción limitada con el mapa, me demostraron en la práctica por qué la mantenibilidad es indispensable. Tener una arquitectura clara fue precisamente lo que permitió aislar esos errores y corregirlos sin comprometer otros módulos.

La escalabilidad fue otro eje central del proyecto. El uso de Firebase hizo posible trabajar con sincronización en tiempo real, algo esencial para una aplicación que depende de la actualización inmediata de información entre pacientes y profesionales. Trabajar con Streams, validaciones y reglas de acceso reforzó la importancia de elegir tecnologías capaces de adaptarse si el sistema creciera a cientos o miles de usuarios.

En cuanto a la implementación de los módulos, cada uno representó un desafío distinto. La agenda, la georreferencia, el registro de usuarios, la lógica de solicitudes y la interacción entre perfiles exigieron pensar tanto en el uso cotidiano como en la forma más limpia de desarrollarlos. Aunque aún quedan mejoras pendientes, como optimizar la persistencia del informe y ciertos flujos del módulo de citas, siento que el proyecto logró una base técnica sólida y preparada para avanzar en siguientes iteraciones sin volver a empezar desde cero.

Un punto importante que mencionar es respecto al uso de microservicios. Si bien en esta etapa del proyecto no se implementaron, principalmente por el alcance, los tiempos y la naturaleza académica de la tesina, la arquitectura modular basada en MVVM dejó el camino abierto para que en un futuro sí puedan integrarse. El proyecto está estructurado de tal forma que cada módulo podría desprenderse como un servicio independiente más adelante, lo que permitiría aumentar la escalabilidad, la tolerancia a fallos y la separación técnica entre funcionalidades críticas. No se usaron microservicios ahora porque habría requerido una infraestructura más compleja, un backend distribuido y un volumen mayor de tiempo de desarrollo, pero la organización actual facilita una transición natural hacia ese modelo si el proyecto evoluciona hacia un despliegue real o comercial.

En lo personal, este trabajo me permitió consolidar conocimientos aprendidos durante la carrera y aplicarlos en un proyecto con impacto social real. La idea de conectar a profesionales de la salud con personas que necesitan atención a domicilio responde a un problema actual y demuestra cómo la ingeniería puede aportar soluciones concretas a la calidad de vida de las personas. Incluso en su estado inicial, MedMatch abre oportunidades laborales para profesionales recién egresados y mejora el acceso a la atención para quienes más lo requieren.

Finalmente, el uso de metodologías ágiles, control de versiones y herramientas modernas me permitió enfrentar desafíos similares a los del mundo laboral. Este proyecto no solo me permitió construir un prototipo funcional, sino también fortalecer mi criterio técnico y mi forma de abordar problemas. Quedan muchas mejoras que me gustaría implementar más adelante, pero siento que el objetivo principal se cumplió: construir una base escalable, organizada y preparada para crecer hacia un producto profesional.

Agradecimientos

Quiero expresar mi agradecimiento a los profesores que me acompañaron durante estos años. Más allá de los contenidos técnicos, valoro su disposición para orientarnos y por las enseñanzas que nos entregaron desde la experiencia y la humanidad, algo que siempre recordaré. Agradezco también a mi pareja y a mis amigos, dentro y fuera de la universidad, por acompañarme en los momentos más importantes de este proceso. Su apoyo constante, su comprensión y su forma de estar presentes hicieron que este camino fuera más llevadero y significativo. Extiendo mi gratitud a mi familia, quienes nunca dejaron de creer en mí. En cada etapa, buena o difícil, estuvieron ahí, sosteniéndome y animándome a seguir avanzando.
Y, de manera muy especial, quiero agradecer a mi madre. Ella ha sido mi mayor soporte, mi guía y la persona que, con su amor incondicional, ha marcado cada uno de mis logros. No existen palabras suficientes para expresar lo que ha significado su apoyo. Ha estado en mis momentos más duros y ha celebrado conmigo cada avance. Este logro también es suyo. Su cariño, su fuerza y su forma de acompañarme son, sin duda, uno de los regalos más grandes de mi vida.

7 Referencias

[1] Amazon Web Services, "What is Architecture Diagramming?", 2024. Disponible en: https://aws.amazon.com/es/what-is/architecture-diagramming/
[2] Universidad Técnica Federico Santa María, "Repositorio Institucional USM – Ingeniería en Informática", 2019–2024. Disponible en: https://repositorio.usm.cl/search?spc.page=1&query=informatica
[3] Uber Technologies Inc., "Plataforma Uber y modelo de funcionamiento", 2024. Disponible en: https://www.uber.com
[4] Lucid Software Inc., "Lucidchart – Herramientas de diagramación profesional", 2024. Disponible en: https://www.lucidchart.com
[5] Firebase, "Firestore Documentation", Google, 2024. Disponible en: https://firebase.google.com/docs/firestore
[6] Firebase, "Security Rules in Firestore", Google, 2024. Disponible en: https:// https://firebase.google.com/docs/rules?hl=es-419
[7] Holistics, "Database Diagram Tool", dbdiagram.io, 2024. Disponible en: https://dbdiagram.io
[8] Firebase Extensions, "Stream Firestore to BigQuery", Google, 2023. Disponible en: https://extensions.dev/extensions/firebase/firestore-bigquery-export
[9] Google LLC, Flutter Documentation, 2024. Disponible en: https://docs.flutter.dev
[10] Google LLC, Introduction to Widgets and the Flutter Framework, 2024. Disponible en: https://docs.flutter.dev/development/ui/widgets-intro
[11] Google LLC, Flutter Architecture Overview, 2024. Disponible en: https://docs.flutter.dev/resources/architectural-overview
[12] Google LLC, Flutter Performance Best Practices, 2024. Disponible en: https://docs.flutter.dev/perf/best-practices
[13] Ministerio de Salud de Chile. (2023). Situación de la atención de salud y listas de espera en Chile. Gobierno de Chile. https://www.minsal.cl
[14] OCDE. (2025). Health at a Glance 2025: OECD Indicators. OECD Publishing. https://www.oecd.org/en/publications/health-at-a-glance-2025_15a55280-en/chile_9e89ef98-en.html
[15] Biblioteca del Congreso Nacional de Chile. (s.f.). Ley N° 19.628 sobre protección de la vida privada. https://www.bcn.cl/leychile/navegar?idNorma=141599
[16] Biblioteca del Congreso Nacional de Chile. (s.f.). Ley N° 20.584, que regula los derechos y deberes que tienen las personas en relación con acciones vinculadas a su atención de salud. https://www.bcn.cl/leychile/navegar?idNorma=1039348