UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

Desarrollo Front-End y diseño de aplicación móvil para Kinin: Plataforma de salud a domicilio

Renato Ortiz Vera

renato.ortiz@usm.cl

Catherine Gómez
Profesora Guía

David Larrondo
Profesor Correferente

Resumen — El propósito de este trabajo es la implementación del Front-end y el diseño de la interfaz de usuario de la aplicación móvil MedMatch, centrándose en la integración de datos con el Back-end y la gestión de la lógica de presentación. MedMatch es una plataforma de telemedicina diseñada para optimizar la atención sanitaria a través de dos modalidades principales: servicio a domicilio y videollamada, permitiendo la solicitud inmediata o la reserva en una hora específica. Desarrollada para la empresa Kinin, la aplicación está destinada al público general que, por factores como la movilidad reducida, requiere una alternativa eficiente a la atención tradicional. Desde una perspectiva social, el proyecto busca generar oportunidades de ingreso para profesionales de la salud recién egresados. La validación funcional se realizó mediante el desarrollo de una interfaz de usuario diferenciada en dos perfiles clave (profesional de la salud y paciente), utilizando tecnologías modernas como la herramienta multiplataforma Flutter y Dart e integrando Firebase y la API de Google Maps para servicios geolocalizados. Se adoptó la arquitectura MVVM para una mayor escalabilidad con la capa de datos. Dicha estructura sienta las bases para el desarrollo de funcionalidades como el seguimiento en tiempo real y el módulo de chat.

Palabras Clave — Front-end, Flutter, Back-end, API, interfaz de usuario, MVVM.

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

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

Título del trabajo: Desarrollo Front-end y diseño de aplicación móvil para Kinin: Plataforma de salud a domicilio

Nombre del candidato(a): Renato Andrés Ortiz Vera

Carrera / Grado: Ingeniería en Informática

Campus: Viña del Mar **Departamento:** ELINF

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.

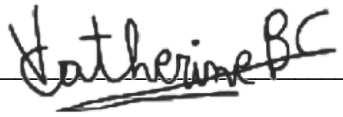
☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por **(marcar una opción)**:

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

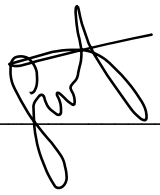
Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

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

Fecha: 14/01/2026 **Firma:** 

Estudiante o Candidato(a):

Fecha: 14/01/2026 **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

En el contexto actual, el crecimiento exponencial de la dependencia a dispositivos móviles ha impulsado una profunda transformación digital en Chile, migrando tareas cotidianas como la comunicación y las transacciones a entornos virtuales. Sin embargo, esta transformación no es solo funcional; ha marcado un cambio de paradigma en el desarrollo de software: se ha transitado de una era donde los usuarios debían adaptarse a interfaces complejas, a un presente donde las interfaces se adaptan intuitivamente al comportamiento y necesidades del usuario. Hoy, el diseño de una aplicación móvil exige un análisis detallado para garantizar que la tecnología sea una facilitadora invisible y no una barrera.

Bajo este escenario, el sector de salud chileno enfrenta el desafío de adoptar estas nuevas dinámicas para mitigar ineficiencias operacionales. La inestabilidad en la atención médica tradicional, marcada por la escasez de vías accesibles para la gestión de citas a domicilio y la rigidez de los sistemas de agendamiento, afecta directamente a usuarios con movilidad reducida o que requieren tratamientos periódicos. Paralelamente, existe la imperante necesidad de crear nuevas oportunidades de ingreso y experiencia para los profesionales de la salud recién egresados que se encuentran fuera del sistema laboral tradicional.

Para responder a estas necesidades, surge la propuesta de MedMatch. Esta aplicación está concebida para actuar como un puente digital entre los profesionales de la salud y los pacientes, poniendo la atención sanitaria "al alcance de la mano". La plataforma utiliza la geolocalización para permitir a los usuarios identificar especialistas en su zona inmediata, gestionar citas y concretar atenciones a domicilio de forma eficiente. Es importante destacar que el alcance de MedMatch trasciende la medicina general, integrando un enfoque multidisciplinario que abarca áreas como kinesiología, nutrición y psicología, entre otras, democratizando así el acceso a un bienestar integral.
Aunque existen diversas plataformas de telemedicina en el mercado (ej. Doctoralia, Medy, MeyDey), la oferta actual carece de una herramienta que combine este enfoque multidisciplinario a domicilio con una experiencia de usuario (UX) amoldado y un agendamiento fugaz. MedMatch busca subsanar este vacío mediante una interfaz moderna que prioriza la adaptabilidad y estética.

Este proyecto se desarrolla en colaboración con la empresa Kinin, organización dedicada a ofrecer soluciones de bienestar y servicios para trabajadores. En este contexto, el trabajo aborda la perspectiva del desarrollo Front-end, teniendo como objetivo principal la implementación técnica y el diseño de la interfaz de usuario de la aplicación móvil. La finalidad es garantizar una interacción accesible y coherente a través de sus dos perfiles clave: profesional y paciente. El alcance de la tesina se delimita estrictamente a la capa de presentación y la integración de la lógica de cliente, empleando tecnologías modernas como el framework Flutter y la arquitectura MVVM para establecer una estructura escalable.

El presente documento está estructurado en cinco capítulos que organizan el desarrollo de la investigación. El Capítulo 2 establece el marco teórico sobre los apartados técnicos del uso y desarrollo de la aplicación. El Capítulo 3 detalla la metodología de diseño y la justificación de la arquitectura MVVM adoptada. El Capítulo 4 abarca la implementación técnica del Front-end y las pruebas funcionales de la interfaz. Finalmente, el Capítulo 5 expone las conclusiones y el impacto social de la propuesta.

1.1 Definición del problema

El ecosistema de salud en Chile enfrenta una paradoja crítica. Mientras la demanda de atención sanitaria aumenta, existe una notable dificultad para la inserción laboral inmediata de los nuevos profesionales. Al analizar los índices de empleabilidad al primer año de egreso, se evidencian brechas significativas en carreras esenciales para la atención domiciliaria. **[1]**

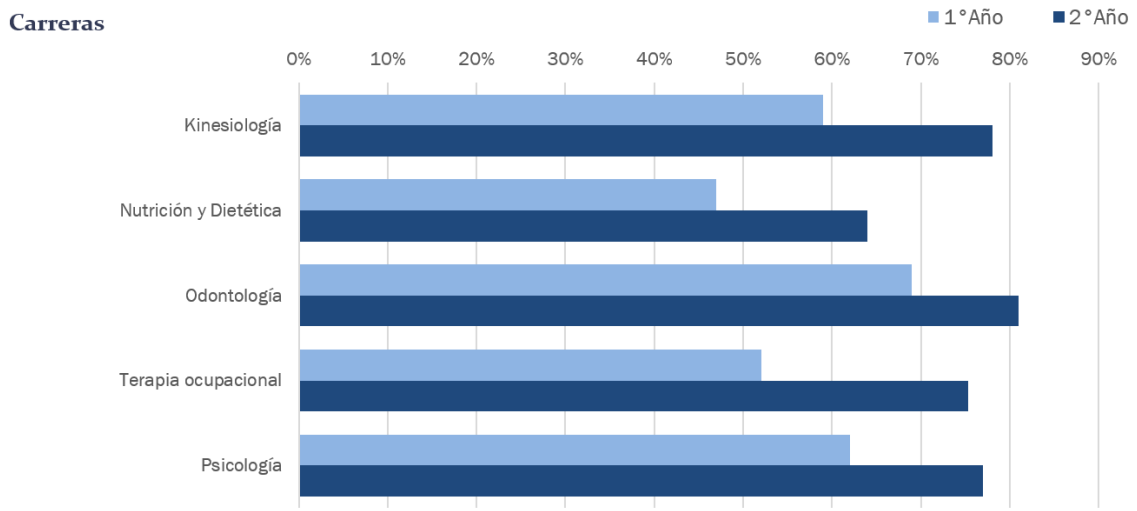


Figura 1. Tasa de empleabilidad 1° y 2° año de carreras de salud.
Fuente: MiFuturo | MINEDUC (Los datos de empleabilidad de 1er y 2° año provienen de los años tributarios 2022–2024). [2]

Tal como se observa en los datos recopilados (ver **Figura 1**), disciplinas como Nutrición, Dietética y Kinesiología presentan tasas de empleabilidad inicial cercanas o inferiores al 60%, cifras considerablemente más bajas en comparación con carreras como Odontología o Psicología. Esta saturación del mercado tradicional deja a un gran contingente de profesionales capacitados sin una vía formal y eficiente para ofrecer sus servicios, subutilizando el capital humano disponible.

Paralelamente, desde la perspectiva del paciente, la experiencia de acceder a la salud tradicional se ha vuelto ineficiente y desgastante. Existe un segmento poblacional, especialmente personas con movilidad reducida, adultos mayores o pacientes crónicos en donde el traslado representa una barrera logística compleja. Sin embargo, el problema no termina al llegar al centro médico. Los usuarios se enfrentan frecuentemente a infraestructuras saturadas, lo que deriva en largas filas de espera y aglomeraciones en las áreas de recepción y atención. Esta fricción en el servicio presencial no solo expone a los pacientes a estrés innecesario y riesgos de contagio, sino que evidencia la falta de canales que permitan descongestionar los recintos mediante una atención domiciliaria.

Si bien el mercado ofrece aplicaciones de telemedicina (como Doctoralia, Medy o servicios de aseguradoras), estas soluciones presentan deficiencias críticas en la Interacción Humano-Computadora (HCI) que limitan su adopción masiva, evidenciando carencias tanto desde la perspectiva técnica del Front-end como de la accesibilidad del servicio:

Enfoque Limitado: Muchas plataformas se centran exclusivamente en la videoconsulta, descuidando la logística necesaria para gestionar visitas domiciliarias geolocalizadas en tiempo real.

Complejidad de Interfaz (UI/UX): Las interfaces actuales suelen poseer una alta carga cognitiva. La falta de adaptabilidad visual y la rigidez en los flujos de navegación generan una barrera de entrada para usuarios que no son nativos digitales.

Restricciones de Acceso y Exclusividad: Un problema crítico en la oferta actual es la accesibilidad administrativa. Gran parte de las plataformas operan bajo modelos cerrados, exigiendo alianzas previas con empresas específicas, seguros comprometidos o modelos de suscripción mensual obligatoria. Esto excluye automáticamente al "público general" o a pacientes esporádicos que requieren una atención puntual sin estar atados a contratos previos.

Figura 2. Brecha digital disminuida por personas mayores.

Fuente: GeriatricArea; revista digital del sector sociosanitario.

1.2 Propuesta de solución

La viabilidad de implementar MedMatch como una solución móvil se sustenta en la progresiva disminución de la brecha digital en la población. Tal como se evidencia en la evolución de usuarios (ver **Figura 2**), existe un incremento sostenido de usuarios habituales de internet, que alcanzó un 59,5% en 2021 **[3]**, reduciendo drásticamente el segmento de "no usuarios". Este contexto favorable valida el desarrollo de una aplicación móvil como el canal idóneo para conectar la oferta de salud con la demanda domiciliaria.

Para materializar esta solución, se ha seleccionado el *framework* Flutter. Esta elección técnica no es arbitraria, sino que responde a la necesidad de ofrecer una interfaz gráfica de alto rendimiento y fidelidad visual. A diferencia de otros enfoques híbridos, Flutter utiliza su propio motor de renderizado **[4]**, lo que permite una personalización granular de la estética y los flujos de interacción, superando las limitaciones de los componentes estándar y garantizando una experiencia fluida tanto para el profesional como para el paciente.

Para definir el alcance, este trabajo se centró en el desarrollo integral del Front-end de la aplicación, abarcando no solo la capa de presentación visual (UI), sino también la integración de la lógica de negocio del lado del cliente y la conexión con servicios externos, como se definirá a continuación:

- **Arquitectura y Gestión de Estado:** Implementación del patrón MVVM para desacoplar la lógica de la vista, asegurando un código mantenible y testeable.

- **Módulos de Interacción:** Desarrollo de las interfaces para la gestión de agendas médicas y perfiles de usuario.

- **Integración de Servicios (Backend-as-a-Service):** Conexión con Firebase Authentication para el manejo seguro de sesiones de usuario y gestión de roles (paciente/profesional).

- **Servicios de Geolocalización:** Implementación y consumo de la API de Google Maps para la visualización de profesionales en tiempo real, cálculo de distancias y la interacción táctil con el mapa para definir direcciones de atención.

- **Comunicación y Agendamiento en Tiempo Real:** Integración de la lógica de Front-end necesaria para soportar un sistema de chat bidireccional y la gestión de citas médicas, permitiendo la interacción entre ambos perfiles.

Queda fuera del alcance de este documento el desarrollo de la infraestructura de servidores dedicados o la creación de APIs REST personalizadas, utilizándose servicios en la nube (Firebase) para suplir estas necesidades durante la etapa de validación.

1.3 Objetivos

1.3.1 Objetivo General

Implementar el desarrollo Front-end y el diseño de interfaz de usuario de la aplicación móvil MedMatch, utilizando el *framework* Flutter bajo el patrón de arquitectura MVVM y principios de Diseño Centrado en el Usuario, con el fin de establecer una arquitectura de software modular y de alto rendimiento, capaz de escalar funcionalmente y gestionar eficientemente la conexión entre profesionales de la salud y pacientes para la atención domiciliaria.

1.3.2 Objetivos Específicos

1. Determinar los requerimientos técnicos y de experiencia de usuario necesarios para la aplicación móvil, identificando los flujos de interacción clave para los perfiles de usuario involucrados.

2. Diseñar los prototipos de interfaz de usuario y la estructura de software bajo el patrón de arquitectura MVVM, asegurando la separación de la lógica de presentación y la navegación intuitiva entre vistas.

3. Implementar los componentes visuales y la lógica de cliente en el *framework* Flutter, integrando los servicios de autenticación, geolocalización, y gestión de datos mediante la conexión con servicios externos (Firebase y Google Maps).

4. Optimizar la gestión de estado y el renderizado de la interfaz mediante la implementación del patrón Provider y estrategias de carga diferida, con el fin de asegurar la eficiencia en el consumo de recursos del dispositivo.

5. Evaluar la escalabilidad y el rendimiento del Front-end mediante la ejecución de pruebas de integración automatizadas, midiendo los tiempos de respuesta y estabilidad visual bajo escenarios de carga de datos.

6. Validar el funcionamiento de la aplicación móvil mediante pruebas funcionales y de usabilidad, verificando la coherencia visual y la correcta interacción de los flujos de agendamiento y comunicación entre ambos perfiles.

1.4 Justificación del proyecto

El proyecto responde a una dualidad de necesidades en el mercado laboral y sanitario. Por un lado, ofrece una solución concreta a la precarización laboral de los profesionales de la salud recién egresados, quienes a menudo enfrentan barreras para conseguir contratos fijos. MedMatch actúa como una plataforma de inserción laboral que permite rentabilizar los tiempos de inactividad, ofreciendo a los especialistas la flexibilidad de generar ingresos complementarios en sus horarios libres,

transformando el tiempo ocioso en productividad económica. Por otro lado, la justificación se centra en la democratización del acceso a la salud. Para pacientes con movilidad reducida, adultos mayores o aquellos que residen en zonas de difícil acceso, la aplicación elimina la barrera física del traslado. Al ofrecer una alternativa cómoda y eficiente para recibir atención domiciliaria, se mejora sustancialmente la calidad de vida del usuario final, evitando desplazamientos innecesarios a centros de salud saturados.

La elección tecnológica es determinante para la viabilidad del proyecto. Se optó por el uso del *framework* Flutter no solo por su capacidad multiplataforma, sino porque su arquitectura permite centrar los esfuerzos en un desempeño excelente y una curva de aprendizaje optimizada frente a otras soluciones del mercado.

La superioridad técnica de esta elección se desglosa en tres ventajas competitivas claves:

- **Personalización Granular:** Gracias a su arquitectura basada en widgets, es posible adaptar la estética y los flujos de interacción a las necesidades específicas de pacientes y profesionales, superando las limitaciones de los componentes nativos estándar.

- **Rendimiento Nativo:** La capacidad de compilar el código a lenguaje máquina nativo asegura una experiencia fluida, crucial para mapas y listas de agendamiento.

- **Eficiencia en el Desarrollo:** Herramientas como el *Hot Reload* permite iterar y visualizar cambios en la interfaz en tiempo real, optimizando los tiempos de implementación, refinamiento del diseño de interfaz de usuario (UI) y reduciendo el *Time-to-Market*.

Desde una perspectiva económica y estratégica, el desarrollo de MedMatch posiciona a la empresa Kinin como un facilitador clave en el mercado de la salud digital, capturando valor mediante la resolución de un problema logístico real: la desconexión entre oferta y demanda. Este modelo de negocio digital elimina la necesidad de infraestructura física (consultorios) para la prestación del servicio, optimizando la estructura de costos; esto permite ofrecer precios más competitivos al usuario final y mejores márgenes de ganancia para el profesional.
En este ecosistema, la implementación de un Front-end de calidad se convierte en un activo estratégico y diferenciador fundamental. A diferencia de las soluciones del mercado, MedMatch reduce la fricción en el proceso de agendamiento mediante una interfaz adaptable, asegurando así una adopción universal del servicio y la sostenibilidad comercial del proyecto.

1.5 Metodología

La metodología que se utilizó para el desarrollo de la aplicación combina principios del Diseño Centrado en el Usuario (UCD) con un enfoque adaptado de la Metodología Ágil. Se centra en las necesidades y expectativas del usuario final, priorizando que la interfaz funcional facilite la interacción entre las vistas. Esto se logra mediante la incorporación de retroalimentación cualitativa de expertos en el sector médico.
Por otro lado, la flexibilidad y adaptabilidad de la Metodología Ágil permiten abordar el desarrollo y la integración de ambas interfaces en fases cortas y efectivas, garantizando entregas incrementales de valor y la mejora continua del producto final.

Para el desarrollo de esta tesina, se ha optado por un enfoque metodológico híbrido que integra los principios del Diseño Centrado en el Usuario (UCD) con la estructura iterativa del marco de trabajo Ágil Scrum

Esta combinación permite abordar el proyecto desde dos frentes críticos:

- **Enfoque Humano (UCD)**
  Garantiza que la interfaz gráfica y los flujos de navegación respondan a las necesidades reales de pacientes con movilidad reducida y profesionales de la salud.

- **Enfoque Técnico (Scrum)**
  Proporciona un marco de trabajo flexible y organizado por entregas incrementales (Sprints), permitiendo adaptar el desarrollo del Front-end en Flutter y la arquitectura MVVM de manera progresiva y controlada."

1.6 Plan de Trabajo

Fase / Hito del Proyecto	Fecha Inicio	Fecha Fin	Duración
PLAN DE TRABAJO	**11/08**	**13/11**	**94 días**
SPRINT 1: Fundamentos y Arquitectura	**11/08**	**28/08**	**17 días**
1.1 Diseño de Wireframes y Prototipado UI	11/08	22/08	11 días
1.2 Configuración de Entorno y Repositorio (GitHub)	22/08	25/08	3 días
1.3 Implementación Auth con Firebase y Login	25/08	26/08	1 día
1.4 Integración SDK Google Maps y Geolocalización	26/08	27/08	1 día
1.5 Maquetación Vistas Home (Paciente/Profesional)	27/08	28/08	1 día

SPRINT 2: Funcionalidades Core (Agendamiento)	**12/09**	**24/09**	**12 días**
2.1 Implementación Lógica y UI de Calendario	12/09	18/09	6 días
2.2 Integración de Lógica de Agenda entre Roles	18/09	23/09	5 días
2.3 Iteración y Refinamiento UI de Autenticación	23/09	24/09	1 día
SPRINT 3: Interacción, Chat y Validación Final	**24/09**	**13/11**	**27 días**
3.1 Desarrollo Flujo de Solicitud de Citas	24/09	01/10	7 días
3.2 Visualización de Citas Confirmadas (Paciente)	01/10	06/10	5 días
3.3 Gestión de Historial de Citas Médicas	06/10	08/10	2 días
3.4 Módulo de Perfil y Configuración de Usuario	08/10	09/10	1 día
3.5 Implementación de Filtros de Búsqueda (Especialidad)	09/10	12/10	3 días
3.6 Optimización de Servicios Asíncronos (Promesas)	12/10	13/10	1 día
3.7 Desarrollo de Chat Bidireccional (Paciente)	13/10	18/10	5 días
3.8 Desarrollo de Chat Bidireccional (Profesional)	18/10	21/10	3 días
SPRINT 4: Optimización y Preparación para la Feria	**21/10**	**13/11**	**23 días**
4.1 Corrección de arquitectura MVVM	21/10	01/11	11 días
4.2 Refinamiento Estético y UX (Polishing)	01/11	06/11	5 días
4.3 Pruebas QA, Debugging y Corrección de Errores	06/11	13/11	7 días

Tabla 1. Cronograma de actividades y desglose de Sprints del proyecto.
Fuente: Elaboración propia.

2 Marco Teórico

2.1 Fundamentos del desarrollo Front-end

El desarrollo Front-end, también conocido como desarrollo del lado del cliente, se define como la disciplina de la ingeniería de software encargada de la implementación gráfica de usuario (GUI) y la gestión de la interacción con el usuario final. A diferencia del desarrollo Back-end que opera en el servidor administrando datos, el Front-end actúa como capa de presentación que traduce los datos crudos en información visual comprensible, logrando gestionar su propio estado y lógica.

Según los estándares actuales de la industria, un desarrollo Front-end robusto se sustenta en tres pilares fundamentales:

1. **Estructura y Semántica**
   Corresponde al esqueleto de la aplicación. En el desarrollo web tradicional, esto se asocia a HTML, pero en el contexto del desarrollo móvil moderno (como en Flutter), se refiere a la jerarquía de componentes o widgets. Su función es definir el orden y la disposición de los elementos informativos, asegurando que la aplicación tenga una estructura lógica legible.

2. Estética y Diseño Visual (UI)

Abarca la capa de estilo que define la apariencia del software. Incluye la tipografía, la paleta de colores, el espaciado y la adaptabilidad (*responsiveness*) a diferentes tamaños de pantalla. El objetivo de esta capa no es meramente ornamental, sino funcional: guiar la atención del usuario y facilitar la comprensión de la interfaz mediante la jerarquía visual.

3. Interactividad y Lógica de Cliente

Es el componente dinámico que permite a la aplicación responder a las acciones del usuario (toques, gestos, entrada de texto). Esta capa es responsable de la gestión de estado, un concepto crítico en aplicaciones transaccionales como MedMatch. Implica capturar eventos, procesar la entrada del usuario, comunicarse con servicios externos (APIs) y actualizar la interfaz gráfica en tiempo real sin necesidad de recargar toda la aplicación.

2.1.1 Paradigma del Desarrollo Móvil Multiplataforma

Dentro del espectro del Front-end, el desarrollo móvil presenta desafíos relacionados con la diversidad de dispositivos y sistemas operativos (principalmente Android y IOS). Esto requería desarrollos paralelos (“Nativo”), duplicando esfuerzos en lenguajes distintos (Kotlin y Swift).
Sin embargo, el enfoque adoptado en esta tesina es sobre el desarrollo multiplataforma. Este paradigma permite escribir una única base de código que se ejecuta para diversos sistemas operativos. Herramientas modernas bajo este enfoque buscan cerrar la brecha de rendimiento respecto de aplicaciones nativas, permitiendo una gestión de los recursos del dispositivo (cámara, GPS, memoria) mientras se mantiene una coherencia visual entre plataformas.

2.2 Tecnologías y Frameworks utilizados

Siguiendo la línea, la investigación fundamenta su implementación técnica en el ecosistema proporcionado por Google a través del *framework* Flutter y el lenguaje de programación Dart.

2.2.1 Framework Flutter

Flutter se define técnicamente como un kit de desarrollo de software (SDK) de código abierto para la creación de interfaces gráficas de usuario (GUI). A diferencia de otros enfoques híbridos que utilizan *WebViews* (como Ionic) o puentes hacia componentes nativos OEM (como React Native), Flutter introduce una arquitectura de renderizado independiente basada en su propio motor gráfico llamado Skia (y recientemente Impeller). Este motor gráfico actúa como un lienzo (*canvas*) directo sobre la pantalla del dispositivo.

Gracias a esta arquitectura, la aplicación tiene la capacidad de ‘dibujar’ cada píxel en la pantalla, otorgando al desarrollador control absoluto sobre la pila de renderizado. En el contexto de esta tesina, esta característica es fundamental, ya que permite ejecutar animaciones complejas a 60 o 120 cuadros por segundo (fps), un requisito vital para la experiencia fluida en mapas y listas de desplazamiento (*scroll*).

El paradigma de widgets: El concepto central de Flutter es que “todo es un Widget”. Desde un botón hasta el diseño de la estructura de la página (*layout*), los widgets son los bloques para construir componentes modulares y reutilizables, facilitando el mantenimiento del código.

2.2.2 Lenguaje de programación Dart

Dart es el lenguaje sobre el cual se construye Flutter. Es un lenguaje optimizado para clientes, orientado a objetos y con tipado estático fuerte. Su elección en este proyecto se justifica por dos características de compilación que optimizan el ciclo de vida del desarrollo de software:

JIT (Just-In-Time)
Durante la etapa de desarrollo, Dart utiliza JIT, lo que habilita la funcionalidad de *Hot Reload*. Esto permite inyectar cambios de código en la máquina virtual en ejecución sin perder el estado de la aplicación, acelerando la experimentación de la interfaz.

AOT (Ahead-Of-Time)
Para la versión de producción (la que usarán los pacientes y médicos), Dart compila el código a instrucciones de máquinas nativas (ARM o x64). Esto elimina la necesidad de un intérprete intermedio, resultando en un tiempo de inicio rápido y una ejecución de alto rendimiento.

2.2.3 Infraestructura en la nube (BaaS)

"Para dotar de funcionalidad dinámica a la interfaz y gestionar la persistencia de datos sin incurrir en el desarrollo de un *Backend* personalizado, se utilizan servicios en la nube (BaaS) que se integran directamente con el cliente Flutter:"

Para la gestión de datos y autenticación, el proyecto utiliza **Firebase**, una plataforma de desarrollo de aplicaciones móviles respaldada por Google que opera bajo el modelo *Backend-as-a-Service* (BaaS). Su elección permite delegar la gestión de infraestructura de servidores, centrando el desarrollo en la lógica del cliente.

Cloud Firestore
Se implementa como base de datos NoSQL orientada a documentos. A diferencia de las bases relacionales (SQL), Firestore permite una estructura de datos flexible y escalable, ideal para el almacenamiento de perfiles de usuarios y registro de citas. Su característica principal para Medmatch es la capacidad de sincronización en tiempo real (*real-time updates),* permitiendo que el estado de una solicitud se actualice al instante en las pantallas del paciente y el profesional.

Firebase Authentication
Servicio utilizado para la gestión segura de identidades. Provee el *Backend* necesario para autenticar usuarios mediante correo electrónico y contraseña, gestionando tokens de sesión seguros y simplificando el manejo de estados de autenticación dentro de la aplicación.

2.2.4 Servicios de Geolocalización

Dado que la propuesta de valor de MedMatch se basa en la conexión física entre usuarios, se integra el **Google Maps SDK for Flutter**. Esta herramienta permite incrustar mapas interactivos directamente en la interfaz de la aplicación, por lo que ofrece una experiencia nativa superior a la de un mapa web incrustado.

El uso de este SDK y sus APIs asociadas permite funcionalidades como:

- **Geocodificación**
  Transformación de direcciones de texto en coordenadas geográficas (latitud/longitud) para ubicar pacientes.

- **Marcadores interactivos**
  Visualización de la posición de los profesionales de la salud en tiempo real, representados mediante marcadores (pins) sobre el mapa.

2.2.5 Herramienta de Diseño y Prototipado

En concordancia con la metodología de Diseño Centrado en el Usuario, se utiliza **Figma** como herramienta principal de diseño de interfaces. Figma es una plataforma de diseño vectorial basada en la nube que permite la creación de *wireframes*, prototipos de alta fidelidad y sistemas de diseño colaborativos. Su uso facilita la exportación de activos gráficos (*assets*) y la inspección de propiedades CSS/Flutter, otorgando una transición fiel del diseño al código.

2.2.6 Entorno y Herramientas de Desarrollo

Para comprometer un flujo de trabajo profesional y trazable, se mencionan las siguientes herramientas de soporte:

Control de versiones (Git)
Se utiliza Git para el control de versiones distribuido, permitiendo gestionar el historial de cambios del código fuente y mantener ramas separadas para una correcta integración de sus funcionalidades (*features*) distribuidas por el equipo del proyecto.

Entorno de Desarrollo Integrado (IDE)
Se emplea **Visual Studio Code** potenciado con extensiones específicas para Dart/Flutter, junto con **Android Studio** para la gestión de dispositivos virtuales (emuladores) que replican diversos entornos de *hardware* para pruebas.

Generación de Datos (Mocking)
Para las etapas de desarrollo y pruebas de carga visual, se utiliza **RandomUser API**, permitiendo poblar la base de datos con fotos ficticias para validar la maquetación de las listas y tarjetas de perfil.

2.3 Arquitectura de software (MVVM)

La arquitectura de software se define como la estructura fundamental de un sistema, comprendiendo los elementos de software, las relaciones entre ellos y las propiedades de ambos. Para el desarrollo de **MedMatch**, y con el objetivo de garantizar la escalabilidad y mantenibilidad del código, se ha adoptado el patrón de diseño **MVVM (Model-View-Model)**.

Este patrón, derivado del clásico MVC (Modelo-Vista-Controlador), se especializa en la separación de intereses (*Separation of Concerns*) **[5]**, desacoplando completamente la lógica de negocio y los datos de la interfaz de usuario. Esta separación es fundamental en el desarrollo móvil moderno con **Flutter**, ya que permite que la interfaz gráfica sea reactiva a los cambios de estado sin conocer el origen de los datos.

El patrón se divide en tres capas primordiales:

1. **El Modelo (Model)**
   Representa la capa de datos y la lógica de negocio pura de la aplicación. Es la abstracción de las fuentes de información y no tiene conocimiento alguno de la interfaz gráfica.

   - **Función en la aplicación**
     Esta capa incluye las clases que modelan a los usuarios (Paciente, Profesional), las citas médicas y los mensajes del chat. Asimismo, integra los repositorios que se comunican directamente con **Cloud Firestore** y la **API de Google Maps** para obtener coordenadas y perfiles.

2. **La Vista (View)**
   Corresponde a la capa de presentación visual e interacción con el usuario. En el contexto de **Flutter**, la Vista está compuesta por el árbol de *Widgets* que se renderizan en la pantalla y es la capa “pasiva” o “tonta”, que se refiere a que no contiene lógica de negocio compleja.

   - **Función en la aplicación**
     Comprende todas las pantallas diseñadas (Login, Mapa, Perfil, Chat) y sus componentes visuales (botones, listas, marcadores).

3. **El Modelo de Vista (ViewModel)**
   Actúa como el intermediario o “puente” entre el Modelo y la Vista. Es el componente encargado de gestionar el estado de la interfaz.

   - **Función en la aplicación**
     Cada pantalla principal cuenta con su propio *ViewModel* (ej. `HomeProfesionalViewModel`, `AgendaViewModel`). Estos gestionan la lógica de validación de formularios, el cálculo de distancias entre usuario y médico, y notifican a la interfaz cuando llegan nuevos mensajes de la atención médica para que se muestren en tiempo real.

Con esta elección la compatibilidad natural se mezcla con la naturaleza declarativa y reactiva de Flutter. A diferencia de otros patrones, MVVM asegura testabilidad, mantenibilidad y gestión de estado reactiva.

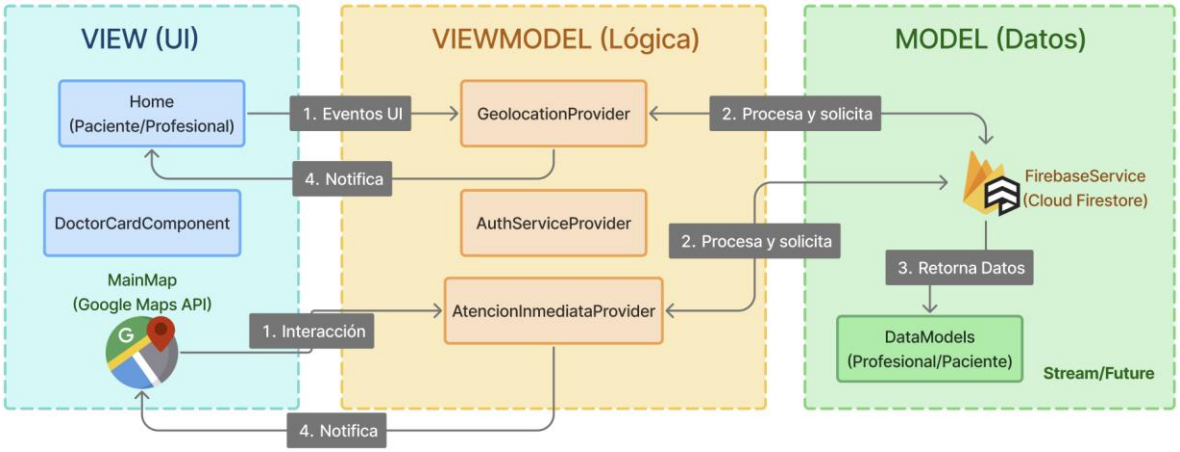


Figura 3. Diagrama de flujo de datos en la arquitectura MVVM aplicada a MedMatch.
Fuente: Elaboración propia.

2.4 Metodología de Diseño Centrado en el Usuario

El Diseño Centrado en el Usuario (UCD) es un enfoque iterativo de diseño en el que los diseñadores se centran explícitamente en los usuarios y sus necesidades en cada fase del proceso. De acuerdo con la norma ISO 9241-210, este enfoque busca mejorar la usabilidad y la experiencia de usuario (UX) mediante la comprensión profunda del contexto de uso, tomando en cuenta a los usuarios, sus tareas y entornos.

Para el presente proyecto, el UCD es fundamental, ya que proporciona las herramientas teóricas clave como son los *User Personas* y *Wireframes* para abordar la accesibilidad y el diseño funcional. Asimismo, este enfoque sitúa las necesidades, limitaciones y comportamientos del usuario final como el eje central de todas las etapas de desarrollo.

En el caso específico de MedMatch, la metodología se estructura en cuatro fases cíclicas: análisis, diseño, prototipado y evaluación.

1. Análisis: Comprensión del Contexto de Uso
Esta fase inicial se centra en la investigación profunda para comprender el "quién", "qué" y "dónde" del sistema. Teóricamente, implica la recolección de requisitos no solo técnicos, sino emocionales y contextuales.

- **Aplicación en el proyecto:** Se estudian los desafíos de los dos arquetipos de usuarios. Para el profesional de la salud, el análisis se enfoca en la gestión eficiente del tiempo y la ergonomía del calendario. Para el paciente, se identifican las barreras de entrada técnicas, evaluando los tiempos de respuesta y los posibles "cuellos de botella" que dificultan la solicitud de una cita médica.

2. Diseño: Definición y Arquitectura

Esta etapa traduce los hallazgos del análisis en soluciones conceptuales y gráficas. Su objetivo es definir la arquitectura de la información y la apariencia del sistema para asegurar una experiencia óptima. Las actividades clave incluyen:

- **Arquitectura de la Información:** Estructuración jerárquica de la plataforma para definir secciones claras y flujos de navegación lógicos.
- **Diseño de Interacción (IxD):** Definición del comportamiento de la interfaz frente a las acciones del usuario, asegurando que la navegación sea intuitiva.
- **Sistema de Diseño (UI Kit):** Creación de una guía de estilo unificada que incluye la paleta cromática, tipografía, retícula y componentes visuales, garantizando la coherencia estética en todas las vistas de MedMatch.

3. Prototipado: Tangibilización de la Solución

El prototipado consiste en la creación de representaciones visuales del sistema con distintos niveles de detalle, permitiendo validar hipótesis de diseño antes de la codificación final:

- **Baja Fidelidad:** Elaboración de *wireframes* y esquemas básicos (*sketches*) que representan el esqueleto estructural y la distribución de elementos.
- **Alta Fidelidad:** Desarrollo de maquetas visuales detalladas (*mockups*) que simulan la apariencia final del producto.
- **Prototipos Interactivos:** Implementación de flujos navegables mediante software especializado, permitiendo simular la experiencia real de uso para detectar fricciones de navegación tempranas.

4. Evaluación: Validación Empírica

La fase final del ciclo implica la verificación del diseño mediante pruebas con usuarios reales o expertos. Su propósito es asegurar que el producto cumple con los estándares de usabilidad requeridos.

- **Pruebas de Usabilidad:** Ejecución de simulaciones de tareas críticas (ej. agendar una cita) en un entorno controlado para observar el comportamiento del usuario.
- **Métricas de Evaluación:** Medición de factores cualitativos y cuantitativos, como la tasa de éxito en la tarea y la carga cognitiva (esfuerzo mental) requerida para interactuar con las funcionalidades.
- **Iteración:** Implementación de mejoras y ajustes en la interfaz basándose en la retroalimentación (*feedback*) obtenida, cerrando el ciclo de mejora continua.

2.5 Metodologías Ágiles y Marco de Trabajo Scrum

Las metodologías ágiles son un conjunto de prácticas de gestión que priorizan la entrega incremental de valor, la colaboración constante y la adaptación al cambio sobre el seguimiento rígido de un plan. Dentro de este espectro, Scrum se define como un marco de trabajo liviano que ayuda a las personas y equipos a generar valor a través de soluciones adaptativas para problemas complejos.
Scrum se estructura en ciclos cortos de desarrollo llamados Sprints, los cuales permiten revisar y adaptar el producto funcional de manera periódica. En el contexto de esta tesina, Scrum proporciona la estructura organizativa necesaria para gestionar el desarrollo del Front-end en Flutter, permitiendo iterar sobre los diseños propuestos por el UCD.

2.6 Principios de Interfaz y Experiencia de Usuario

Para fundamentar las decisiones de diseño tomadas en MedMatch, es imperativo distinguir y definir conceptos rectores de la interacción digital: Interfaz de usuario (UI), Experiencia de Usuario (UX) y los estándares de calidad asociados.

2.6.1 Interfaz de usuario

La interfaz de usuario se define como el espacio donde interactúan los humanos con las máquinas, diseñado para facilitar la comunicación efectiva entre el usuario y el sistema. Una UI eficiente debe ser intuitiva, accesible y orientada a cumplir los objetivos del usuario. Para la aplicación MedMatch, la interfaz presenta distintos apartados que guían la simulación de un cliente tratando de solicitar una cita médica, con funciones de transporte. Además de mostrar secciones ajustables a las preferencias de un profesional de la salud a la hora de aceptar o denegar una solicitud.

2.6.2 Diferenciación entre UI y UX

Interfaz de Usuario (UI)
Se refiere a la capa tangible y visual con la que el usuario interactúa. Incluye los elementos gráficos como botones, iconos, tipografía, paleta de colores y la disposición (*layout*) de los elementos en pantalla. Su objetivo es la claridad y la estética **[6]**.

Experiencia de Usuario (UX)
Abarca la percepción total del usuario al interactuar con el sistema. Según **Don Norman**, quién acuñó el término en su obra “La Psicología de los Objetos Cotidianos” **[7]**, la UX no se limita a la pantalla, sino que incluye la utilidad, la facilidad de uso y la eficiencia del sistema para resolver el problema del usuario. En MedMatch, una buena UI (bonita) debe estar subordinada a una buena UX (que el paciente logre agendar sin frustración).

2.6.3 Usabilidad y Heurísticas

La usabilidad se define, según la norma **ISO 9241-11**, por la eficacia, eficiencia y satisfacción con la que usuarios específicos logran objetivos específicos en un contexto de uso determinado. Para garantizar este atributo, el diseño de la aplicación se rige por las **10 Heurísticas de Usabilidad de Jakob Nielsen [8]**. En el contexto de esta tesina, se priorizan especialmente:

- **Visibilidad del estado del sistema:** El usuario siempre debe saber qué está pasando (ej. Indicadores de carga o avisos al buscar horarios del médico).

- **Correspondencia entre el sistema y el mundo real:** Uso de lenguaje y conceptos familiares (ej. Icono de calendario para agendar, icono de "pin" para ubicación).

- **Prevención de errores:** Diseños que eviten que el usuario cometa fallos críticos, como realizar un viaje a domicilio de una cita médica de videollamada.

2.6.4 Accesibilidad Digital (WCAG)

Dado el enfoque social del proyecto hacia pacientes con movilidad reducida y adultos mayores, la accesibilidad no es un agregado, sino un requisito funcional. Este principio se basa en las Pautas de Accesibilidad para el Contenido Web (**WCAG 2.1**) del W3C.

El desarrollo del Front-end en Flutter busca cumplir con los principios de ser:

Operable
Presencia de elementos táctiles (botones) con el tamaño mínimo estándar de *Material Design* (que suele ser 48 píxeles aproximadamente) para facilitar la interacción motora.

Comprensible
Flujos de navegación lineales y predecibles que no requieran una carga cognitiva excesiva. Este enfoque sigue los postulados de usabilidad de **Steve Krug (2014) [9]**, quien argumenta que una interfaz eficiente debe ser evidente por sí misma, eliminando la incertidumbre en la toma de decisiones del usuario.

3 Desarrollo

3.1 Configuración de entorno y dependencias

Para el desarrollo de MedMatch, se utilizó la versión estable de **Flutter: 3.35.2** bajo el lenguaje **Dart (SDK de Flutter): 3.9.0**.

A continuación, se muestran las dependencias críticas seleccionadas:

```yaml
dependencies:
  # Gestión de Estado y Arquitectura
  provider: ^6.1.5+1

  # Servicios Backend (Firebase)
  firebase_core: ^4.1.0
  firebase_auth: ^6.0.2
  cloud_firestore: ^6.0.0
  firebase_storage: ^13.0.2
  firebase_messaging: ^16.0.2

  # Geolocalización y Mapas
  flutter_map: ^8.2.1   # Motor de mapas (OpenStreetMap)
  latlong2: ^0.9.1      # Cálculos de coordenadas
  geolocator: ^14.0.2   # Acceso al GPS del dispositivo

  # Funcionalidades de Agendamiento y Utilidades
  table_calendar: ^3.2.0      # Gestión visual de citas
  intl: ^0.20.2               # Formato de fechas y horas
  shared_preferences: ^2.5.3  # Persistencia local
  permission_handler: ^12.0.1 # Gestión de permisos
```

Figura 4. Extracto de dependencias críticas en pubspec.yaml.
Fuente: Elaboración propia.

3.2 Matriz Justificación de cada dependencia

Categoría	Dependencia	Justificación de su Implementación
Arquitectura	**provider**	Habilita la inyección de dependencias y la gestión de estado, fundamental para implementar el patrón **MVVM** y separar la lógica de la vista.
Seguridad y Datos	**firebase_auth**	Gestiona la autenticación segura de usuarios (Paciente/Profesional).
	cloud_firestore	Base de datos NoSQL utilizada para almacenar perfiles, historial médico y el estado de las citas en tiempo real.
Georreferencia	**flutter_map** y **latlong2**	Proveen el motor de renderizado para visualizar mapas interactivos y realizar cálculos de distancia entre coordenadas geográficas.
	geolocator	Permite acceder al hardware GPS del dispositivo para obtener la ubicación del usuario al solicitar una atención
Funcionalidad	**table_calendar**	Componente visual crítico para el módulo de agendamiento, permitiendo a los profesionales bloquear horarios y a los pacientes seleccionar fechas.
Notificaciones	**firebase_messaging**	Infraestructura para el envío de notificaciones *push* (en segundo plano) para alertar sobre confirmaciones de citas.
Persistencia	**shared_preferences**	Almacenamiento local ligero tipo *Key-Value* para guardar configuraciones del usuario o banderas de estado (ej. "Inicio de sesión automático") sin consultar la base de datos.

Tabla 2. Matriz de dependencias tecnológicas y justificación de implementación.
Fuente: Elaboración propia.

3.3 Implementación de la Arquitectura MVVM

La estructura de directorios del proyecto se organizó bajo el patrón **MVVM (Model-View-ViewModel)**, bajo la organización por capas hacia una **Arquitectura basada en Funcionalidades (Feature-First)**. Este esquema, encapsula cada unidad lógica de negocio en su propio módulo dentro del directorio 'features/'.

A continuación, se presenta la principal estructura de carpetas implementada en MedMatch:

```
lib/
├── main.dart/                    # Punto de entrada
├── app.dart/                     # Configuración de raíz
├── firebase_options.dart/        # Configuración para conexión con Firebase
├── core/                         # Capa Transversal (Utilidades compartidas)
│   ├── constants/                # Variables estáticas
│   ├── router/                   # Configuración de rutas de navegación
│   ├── theme/                    # Sistema de Diseño
│   └── widgets/                  # Componentes UI atómicos reutilizables
└── features/                     # Archivo de exportaciones
    ├── auth/                     # Módulo de Autenticación
    │   ├── login/                # Vistas y lógica de inicio de sesión
    │   └── register/             # Vistas y lógica de registro de usuarios
    ├── maps/                     # Servicios de Geolocalización compartidos
    ├── models/                   # Modelos
    ├── onboarding/               # Pantallas de introducción y bienvenida
    ├── doctor/
    │   ├── agenda/               # Gestión de disponibilidad horaria
    │   ├── chat/                 # Lógica de mensajería (lado doctor)
    │   ├── home/                 # Dashboard principal del profesional
    │   ├── notifications/        # Centro de notificaciones y alertas
    │   └── schedule/             # Visualización de cronograma diario
    └── patient/
        ├── chat/                 # Lógica de mensajería (lado paciente)
        ├── home/                 # Búsqueda y visualización de mapa
        ├── medical_appointments/ # Historial y estado de citas
        ├── payments/             # (Futuro) Pasarela de pagos
        ├── profile/              # Configuración de cuenta
        └── waiting_proffesional  # Estado de espera de atención
```

Figura 5. Estructura de directorios en /lib.
Fuente: Elaboración propia.

A diferencia de una organización monolítica, esta estructura permite aplicar el patrón **MVVM** de manera localizada: Dentro de cada sub-carpeta funcional, coexisten la **Vista**, el **ViewModel** y los **Modelos** específicos de esa funcionalidad, ofreciendo ventajas críticas para el desarrollo como la escalabilidad, encapsulamiento y la colaboración.

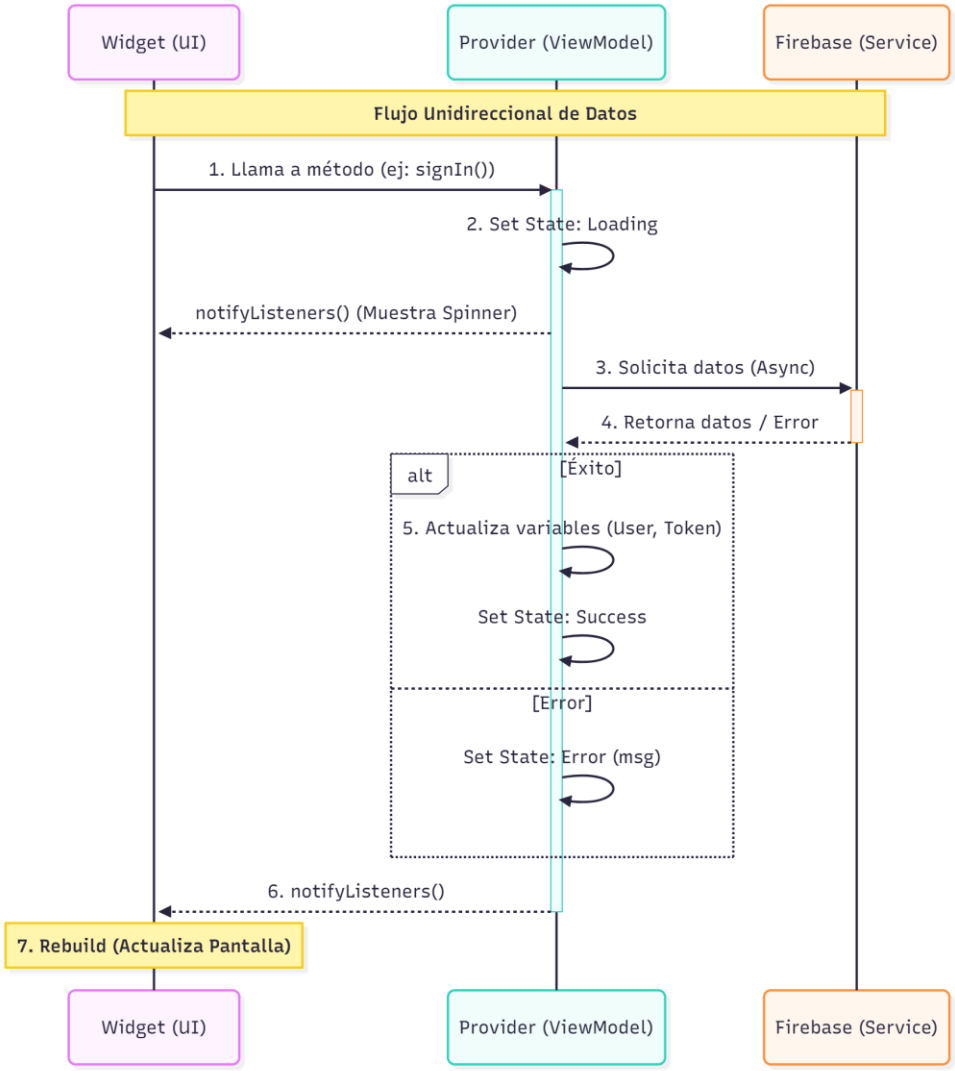


Figura 6. Diagrama de Secuencia del Flujo de Datos con Provider.
Fuente: Elaboración propia.

3.3.1 Descripción del flujo MVVM con Provider

1. **Interacción (Paso 1)**
   La Vista (*Widget UI*), actuando como un componente centrado en el dibujado, captura el evento del usuario y delega la responsabilidad invocando al método correspondiente en el ViewModel (*Provider*).

2. **Gestión de Estado Inicial (Paso 2)**
   El ViewModel asume el control ("Lógica de Negocio") y actualiza su estado interno a `Loading`. Al ejecutar `notifyListeners()`, fuerza a la UI a reconstruirse para mostrar retroalimentación visual (*Spinner*).

3. **Comunicación Asíncrona (Pasos 3 y 4)**
   Posteriormente, el ViewModel solicita los datos al Servicio (*Firebase*) utilizando la sentencia `await`. Esto permite esperar la respuesta de la red sin congelar el hilo principal de la interfaz.

4. **Resolución y Notificación (Pasos 5, 6 y 7)**
   Dependiendo de la respuesta del servidor (bloque `alt`), el ViewModel actualiza sus variables de estado (Éxito o Error) y emite una notificación final. Esto desencadena el Rebuild de la Vista, permitiendo que la pantalla refleje el nuevo estado (por ejemplo, mostrando el usuario autenticado o el mensaje de error).

3.4 Arquitectura de Navegación

La arquitectura de información de la aplicación se diseñó bajo un esquema de segregación de roles, lo que implica que el sistema opera funcionalmente como dos aplicaciones en una. Esta separación estructural ofrece seguridad y garantiza la pertinencia de la interfaz, diferenciando estrictamente el entorno del Paciente del entorno del Profesional.

3.4.1 Estructura Jerárquica y Flujo de Usuario

Como se ilustra en la **Figura 7**, el flujo de navegación no es necesariamente lineal, sino que se bifurca desde el inicio para crear ecosistemas de navegación distintos.

A continuación, se describen los componentes estructurales del flujo representados en el diagrama:

A. **Bifurcación Inicial y Autenticación**
   El punto de entrada ("Introducción") actúa como un distribuidor de tráfico. Dependiendo de las credenciales ingresadas en las pantallas de *Login*, el sistema de enrutamiento dirige al usuario hacia una de las dos ramas principales del árbol de navegación. **(Véase Anexo 1)**

B. **Flujo del paciente (Rama Izquierda)**
   El ecosistema del paciente (representado en tonos azules) está diseñado bajo un modelo de descubrimiento y contratación.

   - **Navegación principal**
     Desde el `Home Paciente`, se accede a funcionalidades transversales como la gestión de `Perfil` y el historial de `Citas Médicas`.

- **Túnel de Conversión**
  El flujo crítico inicia en la búsqueda y selección (`Información del Profesional`), avanza hacia la toma de decisión (`Confirmar Cita` y `Método de Pago`) y culmina en la sala de espera virtual.

- **Estado de Espera**
  La conexión entre "Sala de Espera" y "Atención en Curso" representa un cambio de estado reactivo, donde la interfaz se bloquea temporalmente hasta recibir la confirmación de disponibilidad por parte del profesional.

C. Flujo del Profesional (Rama Derecha)

El ecosistema del profesional (representado en tonos verdes) sigue un modelo de gestión operativa y respuesta.

- **Tablero de Control**
  El `Home Profesional` se presenta de manera similar al `Home Paciente`, pero con la diferencia que este entorno permite al usuario establecer su ubicación activa en el mapa (para ser visible en la búsqueda), además de gestionar su agenda y visualizar su calendario de disponibilidad.

- **Gestión de Solicitudes**
  Mientras que el paciente "busca", el profesional "recibe". El diagrama destaca la recepción de la "Notificación de Reserva", evento que habilita el acceso a la `Información del Paciente` y su geolocalización (`Ver Ubicación Paciente`). **(Ver Anexo 2 para más detalles)**

D. Puntos de Convergencia e Interacción

Aunque los flujos son independientes, el diagrama evidencia los nodos críticos donde ambos roles interactúan dentro de la plataforma:

- **Síncronos (Chat)**
  Módulo central compartido donde ambos usuarios se comunican en tiempo real.

- **Asíncronos (Transacción)**
  Las líneas punteadas en el diagrama (por ejemplo, conectando `Confirmar Pago` con `Notificación de Reserva`) ilustran interacciones reactivas. Representan cómo una acción en el *Front-end* del paciente dispara eventos y notificaciones en el *Front-end* del profesional, orquestados por el *Backend*.

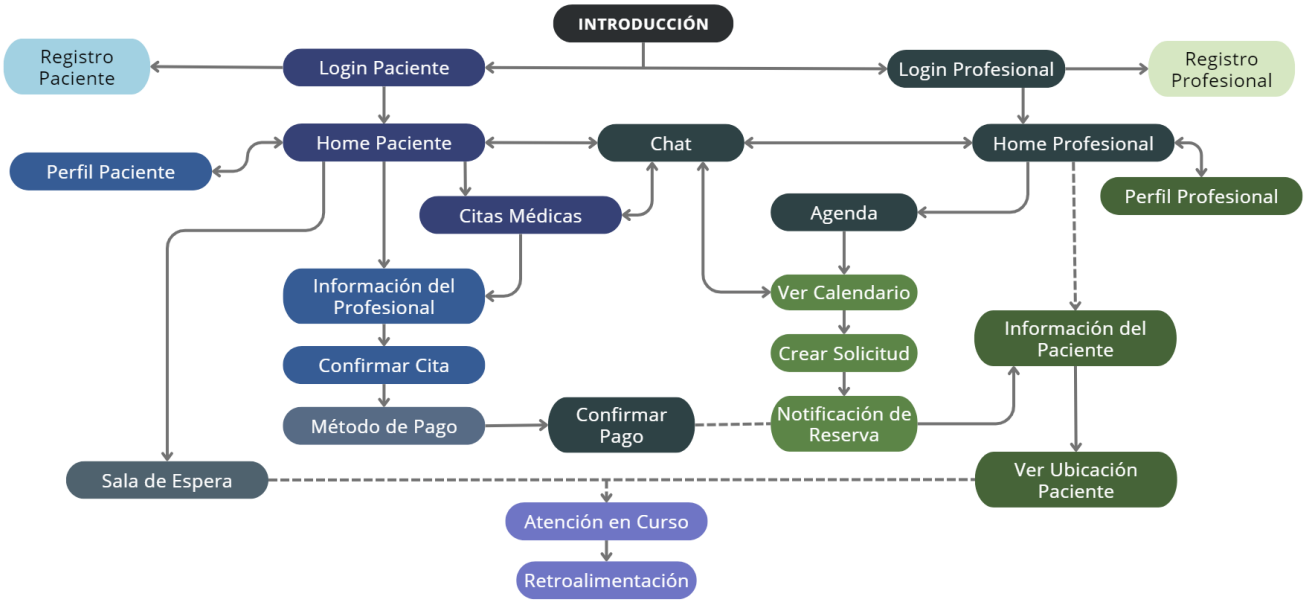


Figura 7. Navegación y estructura de pantallas.
Fuente: Elaboración propia.

3.4.2 Implementación del Enrutamiento (AppRouter)

Para materializar la arquitectura de navegación descrita, se implementó una estrategia de **Enrutamiento Centralizado** utilizando la API nativa `Navigator` de Flutter. Este sistema gestiona el historial de navegación como una estructura de datos tipo Pila (*Stack*), donde las pantallas (*Widgets*) se apilan o eliminan según la interacción del usuario.

En lugar de dispersar la lógica de navegación dentro de cada botón, se creó la clase `AppRouter` (ver **Figura 8**). Esta clase actúa como un interceptor global que utiliza el método `onGenerateRoute` para mapear identificadores de ruta constantes (por ejemplo, `AppRoutes.Introduction`) hacia sus vistas correspondientes.

```
Route<dynamic> onGenerateRoute(RouteSettings settings) {
  switch (settings.name) {
    case AppRoutes.loginPatient:
      return MaterialPageRoute(builder: (_) => const
LoginPatientPage());

    case AppRoutes.homeProfessional:
      return MaterialPageRoute(builder: (_) => const
HomeProfesional());

    // ... casos para cada ruta
  }
}
```

Figura 8. Implementación del AppRouter en Código.
Fuente: Elaboración propia.

3.5 Desarrollo de módulos funcionales

En esta sección se detalla la implementación técnica de las funcionalidades centrales de **MedMatch**. Para cada módulo, se expone la lógica de negocio (*ViewModel*) y su integración con la interfaz gráfica (*View*).

3.5.1 Módulo de Autenticación y Gestión de Identidad

Este módulo centraliza todas las operaciones relacionadas con el ciclo de vida del usuario, abarcando desde el registro inicial de credenciales hasta la gestión de sesiones y perfiles. La implementación integra dos servicios de **Firebase**: **Authentication** para la seguridad de acceso y **Cloud Firestore** para el almacenamiento de datos demográficos.

A. Mecanismos de Registro (Onboarding)

El primer punto de contacto ocurre a través de las interfaces de ingreso. El registro de usuarios se implementó mediante un formulario reactivo que valida en tiempo real la integridad de los datos críticos (correo, contraseña y RUT). Al confirmar, el sistema ejecuta una transacción dual:

1. Crea la credencial segura en **Firebase Auth**.
2. Genera simultáneamente un documento inicial en la colección correspondiente (`patient` o `doctor`) en Firestore. Para garantizar integridad referencial, se vinculan ambos registros utilizando el RUT como **Identificador de Documento (Doc ID)**, facilitando la indexación y búsqueda directa en la base de datos.

B. Arquitectura de Gestión de Sesiones y Enrutamiento

El sistema de autenticación se diseñó bajo una arquitectura reactiva basada en eventos. El estado de la sesión en Firebase actúa como la "fuente de la verdad" que dicta el comportamiento de la interfaz.

- **Componentes de Seguridad**
  Se inyectó el `AuthenticationService` mediante **MultiProvider [10]** en la raíz del árbol de *widgets*. Este servicio expone un *Stream* (`authStateChanges`) que actúa como observador perpetuo del estado de conexión.

- **Lógica de Enrutamiento**
  Al detectar un inicio de sesión exitoso, el sistema consulta la persistencia local (**SharedPreferences**) para identificar el rol del usuario. Utilizando una `GlobalKey<NavigatorState>`, se fuerza la navegación imperativa hacia el entorno correspondiente (`HomePatient` o `HomeProfessional`).

C. Gestión de Perfil y Persistencia de Datos

Una vez autenticado, la aplicación utiliza **SharedPreferences** para almacenar en cache el rol del usuario. Esta estrategia de persistencia local permite acceder a la navegación principal de manera inmediata en sesiones posteriores, eliminando la latencia de espera por consultas de red.

Respecto a la edición del perfil, existe una distinción técnica importante: mientras **Firebase Auth** administra exclusivamente las credenciales de acceso, la información extendida reside en **Firestore**. En consecuencia, las actualizaciones de perfil ejecutan operaciones de escritura (`update`) en la base de datos, cuyos cambios se reflejan inmediatamente en la interfaz gracias a la reactividad de los widgets.

(**Véase Anexo 1 para más detalles)**

3.5.2 Implementación del Flujo de Atención Inmediata

Este módulo constituye el núcleo operativo de la plataforma. Su implementación exigió resolver desafíos de sincronización en tiempo real y mantenimiento de contexto visual. A continuación, se detalla la arquitectura de los componentes:

A. Punto de Entrada y Solicitud (Lado Paciente)

Para iniciar el flujo sin saturar la interfaz principal, se implementó una estrategia de integración no intrusiva en el *Home* del paciente.

- **Lógica Reactiva**

  Se dispuso de un botón de acción condicional envuelto en un `StreamBuilder`. Este componente "escucha" en tiempo real la colección `atencionInmediata`. Si el profesional no está disponible, el botón cambia de estado visual y se deshabilita, proporcionando *feedback* preventivo antes de cualquier interacción.

- **Estrategia de Navegación**

  Al confirmar la solicitud, se optó por una transición mediante `MaterialPageRoute` directo hacia la vista `WaitingProfessionalView`. Esta decisión de diseño, que se desvía del enrutamiento nominal estándar, se justifica por la necesidad de inyectar objetos complejos (la instancia completa del modelo del profesional) directamente al constructor del *widget* de destino.

B. Arquitectura de la Sala de Espera Reactiva

El desafío principal en esta fase fue mantener al paciente informado sobre el estado de su solicitud sin perder el contexto espacial.

- **Composición por Capas (`Stack`)**

  La interfaz se construyó sobre un widget `Stack`. La capa base es un `FlutterMap` centrado en la ubicación del usuario, sobre la cual se superponen paneles de información utilizando

Glassmorphism (desenfoque y transparencia). Está técnica permite visualizar el mapa "detrás" de los datos de estado, reduciendo la sensación de bloqueo.

- **Doble Sincronización**
  La vista mantiene suscripciones activas a dos bases de datos:
  1. **Firestore**: Monitorea el estado de la solicitud.
  2. **Realtime Database**: Actualiza la posición del marcador del profesional en vivo.

- **Feedback Marcador**
  Se implementó un `AnimationController` para generar un efecto de "pulso" en los marcadores, otorgando una sensación de actividad y "vida" a la espera.

C. Centro de Recepción de Alertas (Lado Profesional)

Para el profesional, se requería un mecanismo de alerta que no interrumpiera la navegación en el mapa.

- **Panel Deslizante (`DraggableScrollableSheet`)**
  Se implementó un panel inferior expandible que permite gestionar las solicitudes entrantes sin ocultar la vista del mapa interactivo.

- **Renderizado de Tarjetas**
  Cada solicitud se renderiza como un componente `Card` dentro de este panel. La interacción táctil navega hacia la vista `InformacionPaciente` donde se reutilizan los componentes de visualización de perfil, pero envueltos en una lógica de decisión (Aceptar/Rechazar).

D. Confirmación del Servicio y Previsualización de Ruta

Este módulo actúa como el puente crítico de sincronización. Antes de aceptar la solicitud, el sistema ejecuta un cálculo de ruta forzado utilizando el `GeolocationService`.

- **Visualización de Esfuerzo**
  El resultado se traduce visualmente en una `Polyline` dibujada sobre el mapa que conecta las coordenadas del doctor y el paciente, proporcionando una estimación inmediata del trayecto.

- **Transacción Atómica**
  Al confirmar, se produce un cambio de estado atómico **[11]**. La interfaz bloquea las interacciones (mostrando un *spinner* de carga) mientras se actualiza el estado del documento en **Firebase** a `Aceptada`. Esta escritura dispara instantáneamente la transición en la pantalla del paciente (gracias al *Stream* configurado previamente), asegurando que ambas partes entren en la fase de seguimiento simultáneamente.

E. Asistencia de Navegación y Seguimiento en Tiempo Real

Durante la fase de traslado, la aplicación opera como un sistema de GPS asistido. En la vista `WaitingPatientView`, se inicia un *stream* de posición mediante `Geolocator`.

- **Optimización de Renderizado (Interpolación)**
  Para evitar saltos bruscos en el movimiento del marcador causados por la latencia del GPS, se implementó una lógica de interpolación suave (`animateTo`) en el controlador del mapa. El marcador no se teletransporta, sino que se desliza hacia la nueva coordenada, mejorando la fluidez visual.

- **Lógica de Arribo (Geofencing Local)**
  La interfaz monitorea constantemente la distancia restante. Cuando esta es inferior a **25 metros**, se habilita visualmente el botón "Continuar". Este control actúa como interruptor manual de seguridad que cambia el estado global a `atencion_en_curso`, certificando que el encuentro físico ha ocurrido.

F. Digitalización del Encuentro Clínico y Cierre

La etapa de conclusión del servicio se formaliza mediante una ficha clínica digital.

- **Captura de Datos Estructurada**
  Se construyó un formulario validado (`Form`, `TextFormField`) que mantiene la consistencia visual (*Teal/Glassmorphism*). El foco se centró en la captura eficiente de datos críticos: hora de término, diagnóstico y prescripción médica.

- **Sincronización de Cierre**
  El paciente permanece en una pantalla de "Atención en Curso" a la espera del evento de cierre. El sistema detecta automáticamente el cambio de estado a `Finalizado` y despliega un Modal de Clasificación (`RatingBar`).

- **Gestión de Memoria y Navegación**
  Al finalizar el flujo, se ejecuta una limpieza profunda de la pila de navegación utilizando `pushNamedAndRemoveUntil`. Esto asegura que el usuario regrese al *Home* sin posibilidad de volver atrás a pantallas de una sesión ya concluida, liberando recursos y reiniciando el ciclo de la aplicación.
  (Véase Anexo 2 para más detalles)

3.5.3 Módulo de Agendamiento y Comunicación Continua

A diferencia de la "Atención Inmediata" que resuelve urgencias en tiempo real, los módulos de Agendamiento y Chat se diseñaron para construir la continuidad de la atención. Esta sección detalla la arquitectura implementada para permitir la planificación y la comunicación asíncrona a largo plazo.

Arquitectura del Sistema de Agendamiento

El desafío técnico principal radicó en transformar una grilla de datos abstracta en una experiencia de usuario fluida, integrando la disponibilidad directamente en el flujo de descubrimiento.

A. Experiencia de Descubrimiento y Reserva (Lado Paciente)

En lugar de aislar el calendario en una pantalla separada, se optó por una integración contextual:

- **Filtrado Contextual Temporal**
  Se implementaron selectores rápidos (*Chips*) en la pantalla principal (filtrando por Hoy, Mañana, Cualquier fecha). Al seleccionar una opción, la lista de profesionales se filtra localmente comparando sus bloques horarios (*slots*) disponibles.

- **Feedback Visual**
  Si un profesional no cuenta con disponibilidad para el filtro seleccionado, se excluye de la lista mediante animaciones implícitas, mejorando la limpieza visual de la interfaz.

- **Componentes Reactivos (`DoctorCard`)**
  Cada tarjeta de profesional actúa como un componente inteligente. Procesa internamente los horarios recibidos vía propiedades (*props*) y, si existe una fecha seleccionada, renderiza los "Bloques Disponibles" como botones interactivos directamente en la tarjeta, reduciendo la cantidad de *clics* necesarios para concretar la reserva (`Click-Depth Reduction`).

B. Tablero de Planificación y Gestión de Slots (Lado Profesional)

Para el médico, la agenda constituye su herramienta operativa diaria, por lo que se priorizó la densidad de información y la claridad semántica.

- **Visualización Semanal Híbrida**
  La vista principal implementa indicadores visuales de carga laboral. Un sistema de "pines" bajo cada día permite evaluar el estado de la semana de un vistazo: el color rojo denota agenda saturada, mientras que el verde indica disponibilidad abierta.

- **Estructura de Datos en Firestore**
  La persistencia se gestiona en una estructura anidada bajo la ruta `doctors/{rut}/horarios/{fecha}`. Cada documento contiene un *array* de objetos `slots`.
  Al agendar una cita, el sistema no elimina el *slot* (lo cual perdería trazabilidad); en su lugar, actualiza su propiedad a `disponibilidad: false` e inyecta los datos del paciente, manteniendo la integridad histórica de la agenda.

Sistema de Mensajería y Arquitectura MVVM

Si la atención inmediata es una llamada efímera, el chat representa el historial clínico conversacional. Para este módulo crítico (`lib/features/patient/chat/`), se adoptó estrictamente el patrón **MVVM** para garantizar robustez y testabilidad.

A. Lógica de Negocio (`ChatViewModel`)

El *ViewModel* actúa como el "cerebro" de la operación, desacoplado totalmente de la interfaz gráfica:

- **Inicialización Determinística**
  Maneja la lógica de conexión verificando la existencia previa de una sala de chat. Para ello, genera un ID único basado en los RUTs de ambos participantes (`Chat.generateCharId`).

- **Gestión de Estado**
  Expone `Streams` de mensajes y variables reactivas de carga (`isLoading`) que la *View* consume, asegurando que la UI solo se preocupe de renderizar datos y no procesarlos

B. Sincronización e Interfaz en Tiempo Real

- **Escucha Activa**
  La aplicación se suscribe a la sub-colección `messages`, ordenada cronológicamente por *timestamp*.

- **Optimización de UX (`Auto-Scroll`)**
  Se implementó `WidgetsBinding.instance.addPostFrameCallback` para manipular el controlador de desplazamiento (*ScrollController*). Esta técnica asegura que la vista se desplace al último mensaje automáticamente solo cuando llega una novedad, garantizando que el usuario siempre visualice el contexto más reciente de la conversación.

- **Resiliencia y Manejo de Errores**
  Se integró una lógica de reintento explícita. Ante fallos de conexión, el *ViewModel* expone un estado de error controlado que permite a la interfaz renderizar un botón de "Reintentar", evitando fallos silenciosos o cierres inesperados de la aplicación (*App Crash*).
  (Véase Anexo 3 para más detalles)

3.5.4 Consumo de APIs y Robustez de Datos

Más allá de la implementación funcional, **Medmatch** incorpora una capa estricta de validación y manejo de errores para garantizar la estabilidad del sistema frente a fallos de red o datos corruptos. Como se ilustra en la **Figura 9**, el consumo de servicios externos sigue un protocolo defensivo de 3 etapas.

1. **Validación Preventiva del Cliente**
   Antes de iniciar cualquier petición asíncrona, el *ViewModel* ejecuta una validación local de los parámetros de entrada y del estado de la conectividad. Esto previene llamadas innecesarias al servidor, optimizando el consumo de datos móviles de usuario y batería del dispositivo.

2. **Manejo Controlado de Excepciones (Try-Catch)**
   La comunicación con *Firebase* y *Google Maps* se encapsula dentro de bloques `try-catch`. Esto permite capturar excepciones específicas (como `SocketException` por falta de internet o `FirebaseAuthException` por credenciales inválidas) y transformarlas en estados de error controlados que la UI puede mostrar amigablemente, evitando que la aplicación se cierre inesperadamente.

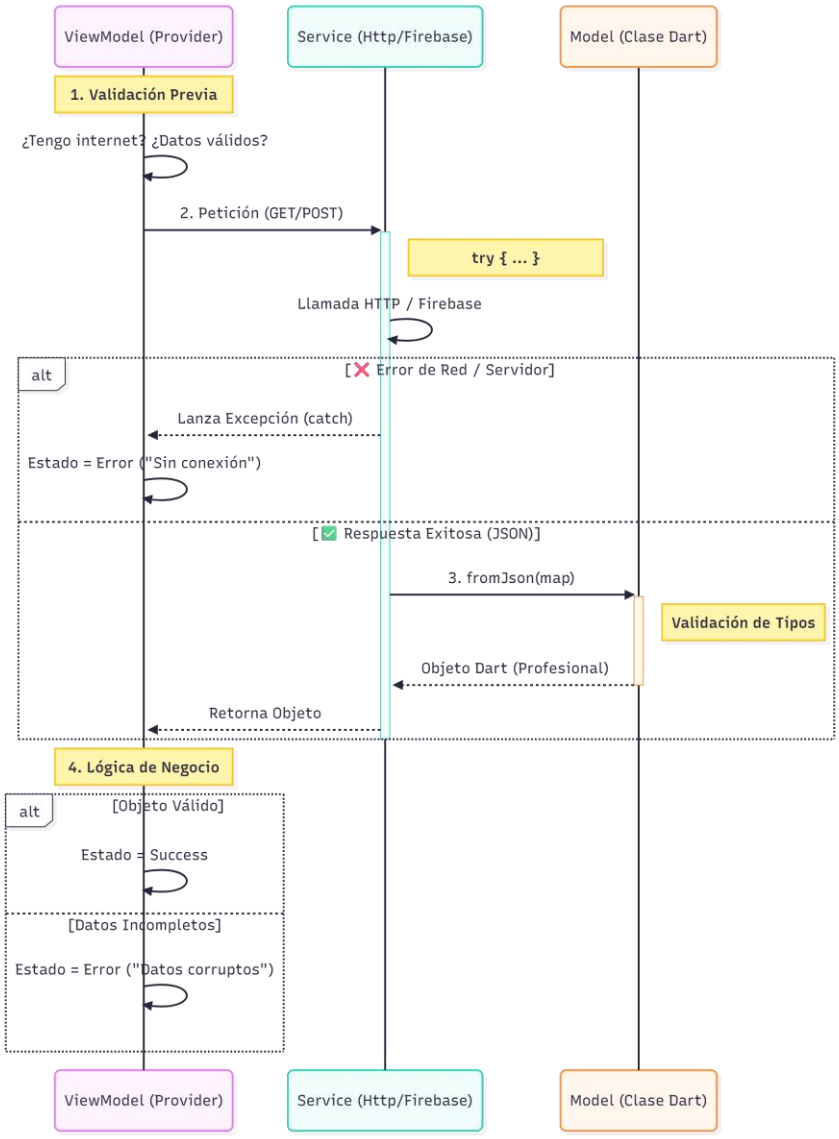


Figura 9. Diagrama de Secuencia de Consumo de Datos y Estrategia de Validación. Fuente: Elaboración propia.

3. **Serialización y Seguridad de Tipos**
   Al recibir la respuesta cruda (JSON) del *backend*, la capa de **Modelos** actúa como un filtro de integridad. Mediante métodos `fromJson`, se verifica que los datos obligatorios existan y correspondan al tipo de dato esperado. El sistema descarta cualquier respuesta malformada antes de que llegue a la vista, asegurando que la interfaz siempre renderice información consistente.

3.6 Diseño de Interfaz y Componentización

Siguiendo el principio de ingeniería **DRY (Don't Repeat Yourself) [12],** la construcción de la interfaz gráfica se rigió por los principios de **Componentización y Reutilización**, implementando un Sistema de Diseño propio basado en la estética **Glassmorphism** (estilo con apariencia de cristal). Esta estrategia permitió reducir la inconsistencia visual entre las vistas esenciales.

Algunos Componentes

- **Botones Estandarizados (`CustomButton`)**
  Se desarrolló un *widget* envolvente que centraliza los estilos de borde, elevación y sombras. Además, gestiona internamente los estados de carga (*loading*).
- **Tarjetas Polimórficas (`DoctorInfoCard`)**
  Este componente se diseñó con alta flexibilidad para adaptarse a diferentes contextos. Se utiliza tanto en la lista de búsqueda (versión resumida), como en el detalle de la cita y en el historial, renderizando su contenido dinámicamente según los parámetros recibidos.

- **Campos de Entrada (`CustomTextField`)**
  Se estandarizaron los *inputs* de texto, integrando validaciones de formulario (`FormValidator`) y unificando los estilos visuales para los estados de foco, error y deshabilitado.

3.6.1 Configuración del Tema Global y Diferenciación Semántica

Para garantizar la coherencia visual y facilitar futuros cambios de marca (*rebranding*) para la empresa patrocinadora, se centralizan estilos en la configuración del `ThemeData` de Flutter.

Estrategia de Color por Roles
Se determinó una nomenclatura de esquema de colores diferenciada para disminuir la carga cognitiva del usuario y facilitar el reconocimiento del contexto:

- **Rol Profesional (Verde Teal)**
  Se utiliza una paleta en tonos *Teal* para evocar seriedad y precisión clínica.

- **Rol Paciente (Azul Institucional)**
  Se mantiene el azul primario de la marca para generar confianza y calma.
  (Véase Anexo 1 para ver los detalles visuales)

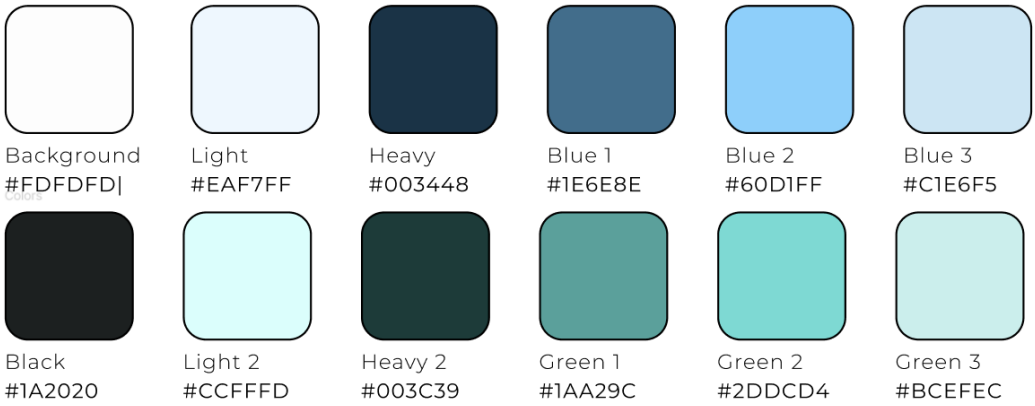


Figura 10. Paleta de Colores en FIGMA.
Fuente: Elaboración propia.

La selección de elementos visuales se orientó a crear una jerarquía clara, donde las acciones críticas destacan sobre la información pasiva. La paleta cromática utiliza tonos azules para identificar el entorno del Paciente y tonos verdes para el Profesional, combinados con una base de colores neutros que aportan una estética estilizada y moderna.

3.6.2 Fidelidad de Implementación

Un desafío crítico en la ingeniería Front-end reside en asegurar que el producto final codificado sea similar a los prototipos de alta fidelidad aprobados. Gracias a la potencia del motor de renderizado de Flutter, se logró una implementación exitosa respecto a las referencias gráficas definidas en Figma **[13]**. Para materializar las jerarquías visuales, se utilizaron *widgets* de posicionamiento preciso como `Stack`, `Positioned` y `Flexible`.

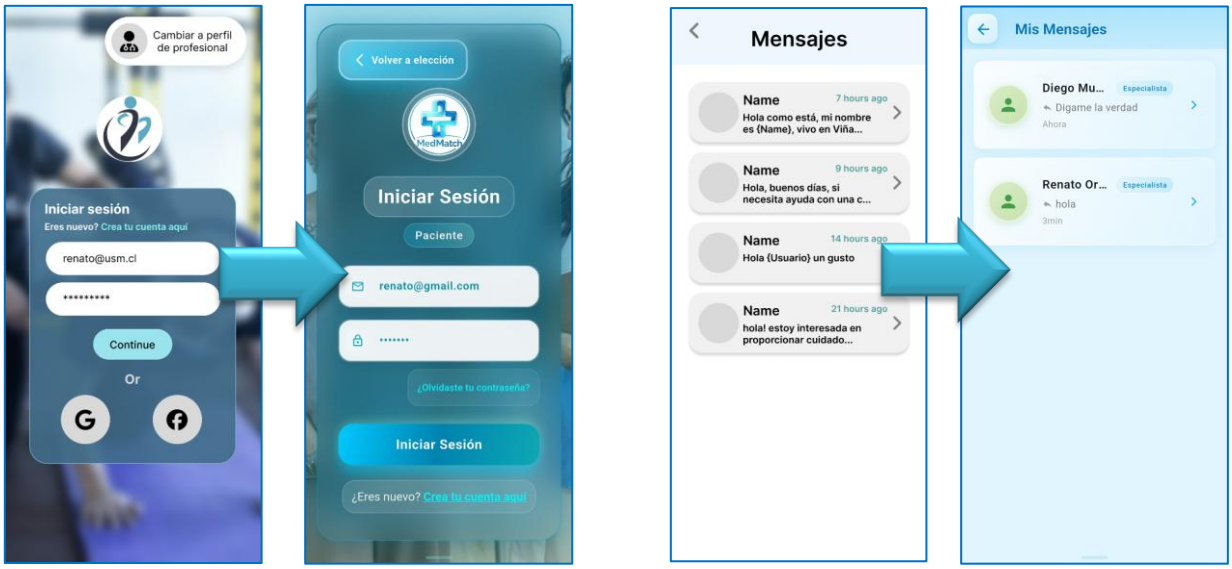


Figura 11. Comparación del Prototipo y la Aplicación Final.
Fuente: Elaboración propia.

4 Implementación y Validación

4.1 Planificación Técnica y Backlog del Proyecto

Para abordar la complejidad del desarrollo de **MedMatch,** se estructuró un *Product Backlog*, dividiendo la implementación propia en hitos funcionales. A continuación, se detalla el desglose de tareas técnicas ejecutadas para cada módulo del sistema.

A. Infraestructura y Configuración Inicial
Fase que estableció los cimientos del proyecto.

- Configuración del Entorno: Inicialización del proyecto en Flutter.
- Integración de Servicios: Vinculación del proyecto con la consola **Firebase** y configuración de las credenciales de acceso.
- Enrutamiento Base: Implementación del `AppRouter` para la navegación y definición de rutas nominales.

B. Gestión de Identidad y Usuarios (Core)
Desarrollo de las interfaces y lógica para el ciclo de vida del usuario.

- Interfaces de Acceso: Implementación de pantallas de *Login* y *Register* con validación de formularios en tiempo real.
- Gestión de Perfiles: Diseño de las vistas de perfil para el Paciente y Profesional, habilitando la edición y actualización de datos en Firestore.
- Tarjetas de Información: Diseño de componentes UI (`Cards`).

C. Módulo de Agenda y Disponibilidad
Implementación de la lógica de negocio para la gestión del tiempo y reservas.

- Calendario del Profesional: Desarrollo de la interfaz de gestión de disponibilidad, permitiendo la apertura y bloqueo de *slots* horarios.
- Visualización de Disponibilidad: Integración de la lógica de lectura de horarios en las tarjetas del profesional, filtrando bloques ocupados.
- Motor de Agendamiento: Implementación del flujo de reserva para el paciente, incluyendo el cálculo de tarifas y selección de modalidad.
- Historial Clínico: Conexión de las citas generadas con el historial médico de ambos perfiles.

D. Herramientas de Valor y Comunicación
Funcionalidades extendidas para mejorar la experiencia de servicio.

- Filtros Avanzados (Paciente): Implementación de lógica de filtrado por especialidad, disponibilidad (Hora/Día) y gestión de “Favoritos”.

- Chat en Tiempo Real: Configuración de servicios de *Streams* en Firestore para la mensajería bidireccional.

E. Diseño, Optimización y Pruebas

Fase transversal de aseguramiento de calidad y refinamiento visual.

- Refinamiento de Arquitectura: Revisión y corrección de la implementación MVVM para desacoplamiento de capas.
- Diseño Responsivo: Adaptación de las interfaces para garantizar la correcta visualización en diferentes tamaños de pantalla.
- Feedback Visual: Desarrollo de pantallas de carga (*Spinners*) específicas y pantalla de espera para el módulo de 'Atención Inmediata'.
- Pruebas de Integración: Validación del flujo de datos entre el *Front-end* (Flutter) y el *Back-end* (Firebase) en entornos de *Staging*.

4.2 Metodología de Trabajo y Gestión de Sprints

El ciclo de vida del desarrollo se estructuró bajo una metodología ágil, organizada en tres iteraciones (*Sprints*) de aproximadamente cuatro semanas de duración. Esta estrategia permitió segmentar la complejidad del proyecto en fases específicas, facilitando la entrega progresiva de incrementos funcionales y la validación iterativa.

Herramientas de Gestión y Colaboración

Para asegurar la trazabilidad del proyecto, se implementó un entorno de trabajo colaborativo:

- **Gestión de tareas**
  Se utilizó **Notion** como tablero central para la documentación y seguimiento del alcance, asignando roles y funcionalidades específicas a los miembros del equipo.

- **Comunicación**
  Se establecieron canales en **WhatsApp** y **Discord** para la coordinación diaria y gestión de archivos multimedia rápidos.

- **Control de Versiones**
  La gestión del código fuente se centralizó en un repositorio de **GitHub**. Se adoptó un flujo de trabajo basado en ramas (*Git Flow*) para mantener la integridad del código en producción mientras se desarrollaban nuevas características en paralelo.

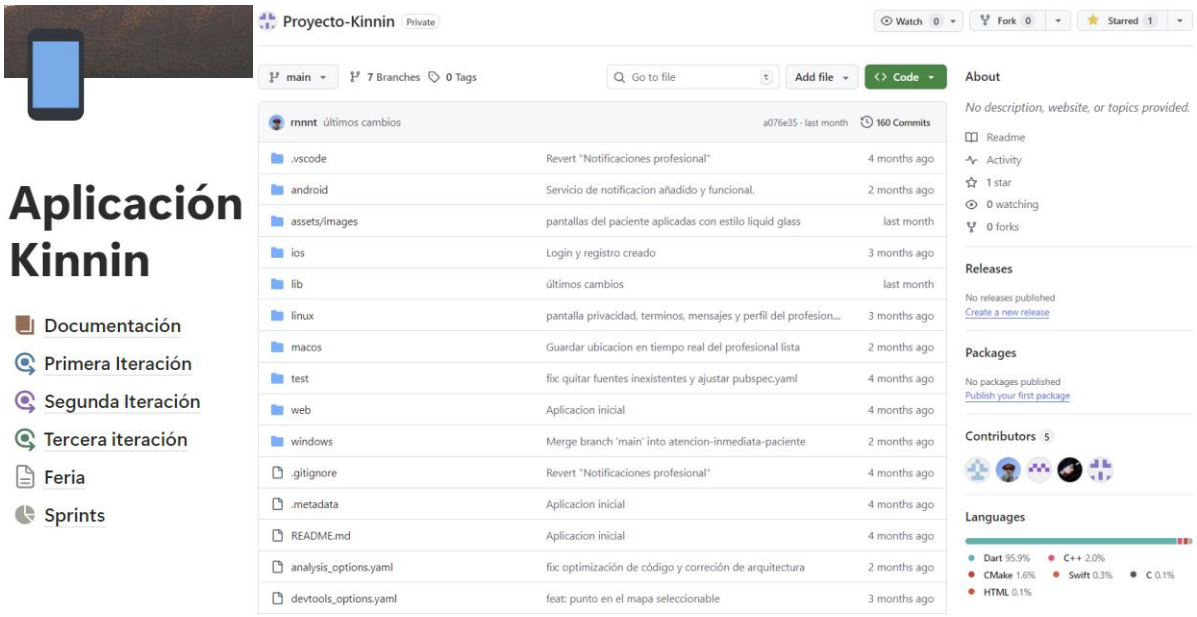


Figura 12. Captura del Notion y GitHub del proyecto.
Fuente: Elaboración propia.

A continuación, se detalla el desglose técnico de las tareas ejecutadas en cada iteración:

Primera Iteración: Conexión, Configuración y Geolocalización
Esta fase se centró en la base del sistema y la identidad visual.

- **Infraestructura**
  - Configuración del entorno de desarrollo en Flutter 3.35.2.
  - Definición de la arquitectura de carpetas y patrones de diseño.
- **Sistema de Diseño (UI)**
  - Definición de la guía de estilos (paleta de colores, tipografía y *assets*).
  - Implementación de maquetas de alta fidelidad para las vistas principales.
- **Desarrollo Base**
  - Maquetación de las pantallas *Home* para los perfiles de Profesional y Paciente.
  - Integración de *widgets*, incluyendo las tarjetas de información (*Cards*) del profesional.

Segunda Iteración: Lógica de Negocio y Agendamiento
Se abordó el núcleo transaccional: la gestión del tiempo.

- **Módulo de Agenda (Profesional)**
  - Implementación de la navegación hacia el módulo de gestión horaria.
  - Desarrollo de la vista de calendario mensual y selectores de bloques de horarios.
  - Implementación de la lógica de disponibilidad (validación de estado disponible u ocupado)

- **Flujo de Solicitud (Paciente)**
  - Desarrollo de la lógica de confirmación y reserva de horas.
  - Integración visual de los horarios disponibles en el perfil del médico.
  - Resolución de incidencias y optimización de las reglas de seguridad en Firebase.
- **Refinamiento UX**
  - Mejoras de usabilidad en la interfaz general y desarrollo del componente de perfil de usuario editable.

Tercera Iteración: Comunicación, Filtrado y Estabilización
La fase final se enfocó en la implementación de herramientas de valor agregado.

- **Módulos de Comunicación**
  - Desarrollo del sistema de Chat en tiempo real mediante *WebSockets*, sincronizando la conexión bidireccional entre paciente y profesional.
  - Implementación de la interfaz de mensajería para ambos roles.
- **Herramientas de Búsqueda**
  - Implementación de lógica de filtrado avanzado (Especialidad, Horario, Disponibilidad) en el *Home* del paciente.
  - Desarrollo de la funcionalidad "Favoritos" para la gestión de profesionales.
- **Pruebas y Optimización de Rendimiento**
  - Ejecución de pruebas unitarias para validar componentes críticos.
  - Se realizó una medición de tiempos de carga "Antes y Después" (*Benchmarking*) mediante la optimización de la lógica de renderizado.
  - Ajustes finales de diseño responsivo y experiencia de usuario (UX).

4.3 Validación y Aseguramiento de la Calidad (QA)

Para garantizar la fiabilidad del software, se ejecutó una estrategia de pruebas escalonadas, abarcando desde la validación de la lógica unitaria hasta la verificación del flujo completo de usuario (*End-to-End*). Esta estrategia de validación no se limitó solo a verificar la funcionalidad, sino que también buscó cuantificar la mejora de rendimiento entre una versión base y la versión final.

4.3.1 Pruebas Unitarias (Lógica del Negocio)

Se implementó una estrategia de pruebas unitarias aisladas enfocadas en la capa de **ViewModels** y **Modelos**, con el objetivo de asegurar la integridad de la lógica de negocio sin depender de servicios externos o de la interfaz gráfica. Mediante técnicas de simulación de dependencias (*Mocking*), se replicaron las respuestas de Firestore para validar escenarios deterministas de éxito y error.

Caso de Prueba: Validación de Conflicto en Agendamiento

- **Entrada:** Inyección de un intento de registro de cita en un bloque horario (`TimeSlot`) que ya posee un ID de paciente asignado en el *Mock* de la base de datos.

- **Resultado Esperado:** El `AppointmentViewModel` debe capturar la colisión, abordar la llamada de escritura al repositorio y emitir un estado de error (`FailureState`) hacia al *view*.

- **Resultado Obtenido: [PASSED]** El sistema gestionó la excepción correctamente. El ViewModel interceptó la solicitud y notificó el error sin comprometer la estabilidad de la aplicación.

Lógica Desacoplada y Testeabilidad
El código se diseñó bajo principios de desacoplamiento. Al separar la lógica en *Providers*, fue posible testear funciones puras, como `calcularDistancia()`, de manera aislada, sin necesidad de inicializar la interfaz gráfica ni el emulador del dispositivo.

Robustez en Modelos de Datos
Las clases de modelo (por ejemplo, `Paciente`) implementan métodos constructores `fromFirestore` con validaciones de nulidad integradas. Esto actúa como una "prueba unitaria implícita" en tiempo de ejecución: cada vez que se reciben datos, el sistema verifica su integridad, asegurando que la aplicación no falle ante documentos corruptos o incompletos.

4.3.2 Pruebas Funcionales y de Integración

Con el objetivo de certificar la integridad de los flujos críticos y la comunicación entre los distintos componentes del sistema, se realizaron pruebas funcionales y de integración. Estas validaciones se instrumentaron utilizando el entorno de `integration_test` de Flutter, lo que permitió simular la interacción real del usuario de extremo a extremo (*End-to-End*), abarcando desde la capa de presentación (UI) hasta la lógica de negocio y la persistencia en base de datos.

A continuación, se detallan los escenarios de validación principales:

Integración Mapa-Formulario
Se verifico la correcta interoperabilidad entre la interfaz de Google Maps y el gestor de estado. Se validó que, al seleccionar un punto geográfico, las coordenadas (`GeoPoint`) sean capturadas y serialicen correctamente hacia el formulario de solicitud en Firestore.

Flujo End-to-End (E2E) – Agendamiento
Se ejecutó el ciclo de "Agendamiento" para validar la secuencia lógica y la navegación:

- Ruta de prueba: Inicio de Sesión (Profesional) → Home → Menú Lateral (*MainDrawer*) → Agenda → Selección de Día → Apertura de Bloque Horario.

- Resultado: Se confirmó la correcta navegación entre pantallas. Adicionalmente, el reporte de ejecución automatizada registró un tiempo total de **1067 ms** para la confirmación del bloque. Este resultado valida que la transacción se completa dentro de los umbrales de usabilidad aceptables.

Gestión de Sesión y Seguridad

Se sometió a prueba el ciclo de vida de la autenticación (*Login, Logout* y reingreso). Las pruebas confirmaron que el `AuthenticationService` gestiona correctamente la persistencia del *token* de seguridad y redirige al usuario a la vista correspondiente según su rol (`userType`) al reiniciar la aplicación.

Resiliencia en Geolocalización

Se validó el manejo de excepciones ante la denegación de permisos de GPS. La aplicación demostró capacidad de recuperación (*resiliencia*), desplegando mensajes informativos al usuario y cargando una ubicación por defecto para evitar cierres inesperados (*crashes*).

Implementación de pruebas de integración

Para garantizar la reproducibilidad de estos escenarios, se implementaron *scripts* de prueba que orquestan las interacciones mediante el `WidgetTester`. Esta técnica permite una validación rigurosa asegurando que todos los *widgets* interactivos respondan correctamente a los eventos simulados.

4.3.3 Evaluación de Rendimiento y Escalabilidad Front-end

Con el objetivo de validar la escalabilidad técnica, se ejecutó un estudio cuantitativo de rendimiento (*Benchmarking*) enfocado en el comportamiento de la vista principal del paciente (*Home Paciente*) bajo condiciones de estrés de datos.
El experimento contrastó el desempeño entre dos versiones de la aplicación:

- **Versión Base (Control):** Implementación estándar que carga y renderiza todos los widgets en memoria simultáneamente, simulando una arquitectura no optimizada.

- **Versión Optimizada (Experimental):** Implementación final con gestión de estado granular (Provider) y virtualización de listas (*Lazy Loading*).

Las mediciones se automatizaron utilizando `integration_test` para simular el desplazamiento (*scroll*) en listas masivas. Para capturar con precisión los tiempos de renderizado, se utilizó la función `binding.traceAction`, la cual permite aislar la ejecución del renderizado y extraer métricas directas del motor gráfico (*FrameTiming*), como se ilustra en la **Figura 13**.

```
await binding.traceAction(
        () async {
          await app.main();
          await tester.pumpAndSettle();
```

Figura 13. Implementación de traceAction para la captura de métricas de rendimiento. Fuente: Elaboración Propia.

A través de esta instrumentación se analizaron:

- **Tiempo Promedio de Construcción (*Avg Build*):** Tiempo que tarda el hilo de UI en dibujar un cuadro. El objetivo es mantenerlo bajo 16.6 ms.
- **Cuadros Perdidos (*Missed Frames*):** Cantidad de cuadros que causan "saltos" visuales (*Jank*) al superar el presupuesto de tiempo de 16.6 ms (necesario para mantener 60 FPS).

Para simular el crecimiento exponencial de usuarios, se sometió la interfaz a cargas de **10, 50, 150 y 300 registros** profesionales renderizados simultáneamente.

Dataset	Versión	Avg Build (ms)	90% (ms)	Frames perdidos
10	Base	27.04	55.94	8
10	Optimizada	12.59	17.51	5
50	Base	51.29	76.52	4
50	Optimizada	13.57	19.86	5
150	Base	67.84	205.97	4
150	Optimizada	12.60	20.48	4
300	Base	88.89	111.49	2
300	Optimizada	12.78	18.47	5

Tabla 3 - Comparativa de tiempos de construcción (*Build Times*).
Fuente: Elaboración Propia.

Los datos de la **Tabla 3** demuestran una mejora crítica en la escalabilidad:

- **Evidencia de Cuello de Botella (Versión Base)**
  Se observa una degradación lineal. Al procesar **300 registros**, el tiempo de construcción se dispara a **88.89 ms**, lo que hace la aplicación inusable (5 veces más lento que el límite de fluidez).

- **Validación de la Solución (Versión Optimizada)**
  La implementación propuesta logró estabilizar el rendimiento. Bajo la misma carga de **300 registros**, el tiempo de renderizado se redujo drásticamente a **12.78 ms**.

- **Conclusión**
  Se logró una optimización del **85% en tiempos de respuesta de UI** en escenarios de alta demanda, validando que la aplicación escala funcionalmente sin comprometer la experiencia de usuario.

4.3.4 Pruebas de Usabilidad (UX)

Considerando el enfoque en pacientes que pueden presentar movilidad reducida, se realizaron pruebas específicas de accesibilidad y respuesta visual.

- **Feedback Visual (Retroalimentación)**
  Se implementaron indicadores de carga *CircularProgressIndicator* o *SpinKit*) para todas las acciones asíncronas. Esto garantiza que el usuario nunca permanezca ante una "pantalla congelada".

- **Manejo de Errores en UI**
  Se validó que, ante fallos de conectividad o errores de API, el sistema despliegue notificaciones no intrusivas (*SnackBars*) o diálogos de alerta explicativos, ocultando al usuario final las trazas de error técnico.

- **Navegación Intuitiva**
  Se verificó la consistencia del `BottomNavigatorBar` y el sistema de rutas nombradas, asegurando que el usuario mantenga siempre el contexto de su ubicación dentro de la jerarquía de la aplicación y disponga de mecanismos claros de retorno.

5 Conclusiones

El desarrollo de MedMatch ha culminado exitosamente, logrando materializar una solución tecnológica robusta capaz de mitigar las barreras de acceso en la atención de salud domiciliaria. La integración de tecnologías como **Flutter** y **Firebase** se sustentó en una arquitectura cuya escalabilidad fue validada cuantitativamente: el sistema demostró mantener tiempos de respuesta estables incluso ante un incremento progresivo en el volumen de datos.

Desde una perspectiva metodológica, la hibridación de prácticas **Ágiles (Scrum)** con el enfoque de **Diseño Centrado en el Usuario (UCD)** demostró ser una estrategia eficaz para gestionar la incertidumbre del desarrollo. Esta aproximación iterativa facilitó la validación temprana de funcionalidades críticas, asegurando que el producto final presentado en la instancia de evaluación práctica (Feria de Software) cumpliera con los estándares de usabilidad requeridos.

No obstante, el análisis ingenieril del proyecto revela limitaciones a las decisiones arquitectónicas adoptadas. Si bien la arquitectura *Serverless* (Firestore) aceleró el despliegue, introduce un **acoplamiento tecnológico** que podría condicionar la portabilidad futura del sistema. A nivel funcional, la ausencia de una pasarela de pagos transaccional y validación biométrica nativa representan deudas técnicas críticas que deben resolverse para garantizar la integridad y seguridad en un entorno productivo real.

Una de las conclusiones técnicas más relevantes es el valor de la planificación arquitectónica temprana. La adopción estricta del patrón **MVVM (*Model-View-ViewModel*)** y la estrategia **Componentización (*Atomic Design*)** generaron un código desacoplado y mantenible.

En un futuro, la evolución de la plataforma exige transitar desde una validación de prototipo hacia una ingeniería de producto escalable. Esto incluye:

- **Evolución de Infraestructura:** Migración de lógica de cómputo intensivo hacia *Cloud Functions* o microservicios para reducir la carga en el dispositivo.
- **Inteligencia Artificial y Machine Learning:** Incorporación de modelos predictivos para estimar la demanda de atención basándose en variables geoespaciales y estacionales, de manera que optimicé la asignación de recursos.
- **Automatización (DevOps):** Implementación de pipelines de integración y Despliegue Continuo (CI/CD) para asegurar la calidad del código ante la integración de nuevos módulos de seguridad avanzada y biometría.

En última instancia, MedMatch trasciende la solución técnica para convertirse en una herramienta de impacto social. No solo habilita la inmediatez en la atención sanitaria, sino que formaliza y dignifica la labor de los profesionales de la salud independientes. La plataforma sienta las bases para un modelo de salud digital más democrático, sostenible y adaptable a las necesidades del entorno domiciliario moderno.

6 Agradecimientos

Doy agradecimientos a mis padres, fueron el pilar fundamental en este proceso y los que me apoyaron en las buenas y en las malas. También a mis compañeros del proyecto, de carrera y amigos que hicieron este camino más llevadero y entretenido. Y a mi familia en general que siempre estuvo al tanto compartiendo ánimos conmigo a lo largo de estos 5 años.

7 Referencias

[1] Mineduc, “Proyección laboral de las carreras, profesionales y técnicas,” *Mi Futuro*. Disponible en: https://www.mifuturo.cl/buscador-de-empleabilidad-e-ingresos/.

[2] Mineduc, “Metodología Buscador Empleabilidad e ingresos 2025-2026,” *Mi Futuro*, 2025. Disponible en: Metodologia-Buscador-Empleabilidad-e-Ingresos_2025-2026.pdf

[3] GeriatricArea, “La brecha digital disminuye entre el colectivo de personas mayores,” Geriatricarea.com. Disponible en: https://www.geriatricarea.com/la-brecha-digital-disminuye-entre-el-colectivo-de-personas-mayores/.

[4] Google, “Flutter – Build apps for any screen,” *Flutter.dev*, 2024. Disponible en: https://flutter.dev.

[5] Microsoft, "The MVVM Pattern," *Microsoft Learn*, 2022. Disponible en: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm.

[6] Integra, “UX vs UI. Mejor no las separes,” *Integra Tecnología, 2023*. Disponible en: https://www.integratecnologia.es/la-innovacion-necesaria/ux-vs-ui-mejor-no-las-separes/.

[7] D. A. Norman, *La psicología de los objetos cotidianos*. Madrid: Editorial Nerea, 1990.

[8] J. Nielsen, "10 Usability Heuristics for User Interface Design," *Nielsen Norman Group*, Abr. 24, 1994. Disponible en: https://www.nngroup.com/articles/ten-usability-heuristics/.

[9] S. Krug, *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability*. New Riders, 2014.

[10] R. Rousselet, “Provider Package Documentation,” *Pub.dev*. Disponible en: https://pub.dev/packages/provider.

[11] B. Frost, *Atomic Design*. 2016. Disponible en: https://atomicdesign.bradfrost.com.

[12] D. Thomas y A. Hunt, *The Pragmatic Programmer: From Journeyman to Master*. Boston, MA: Addison-Wesley, 1999.

[13] “Prototipo App Kinin,” *Figma*. Disponible en: https://www.figma.com/design/StM0yF5oDszZWjiXAGghQx/PROTOTIPO-APP-KININ?node-id=0-1&t=7kj8BopFkjVmZySn-1.

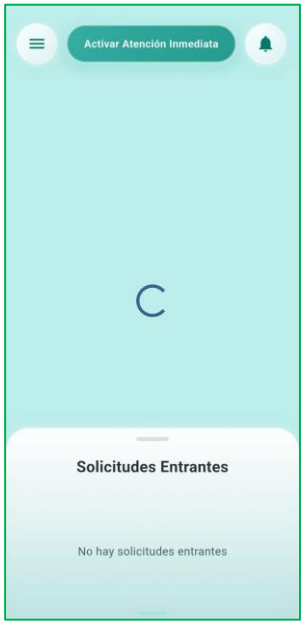
8 Anexos

8.1 ANEXO 1. Flujo de autenticación

A. Paciente (Muestra el login y el error de credenciales)

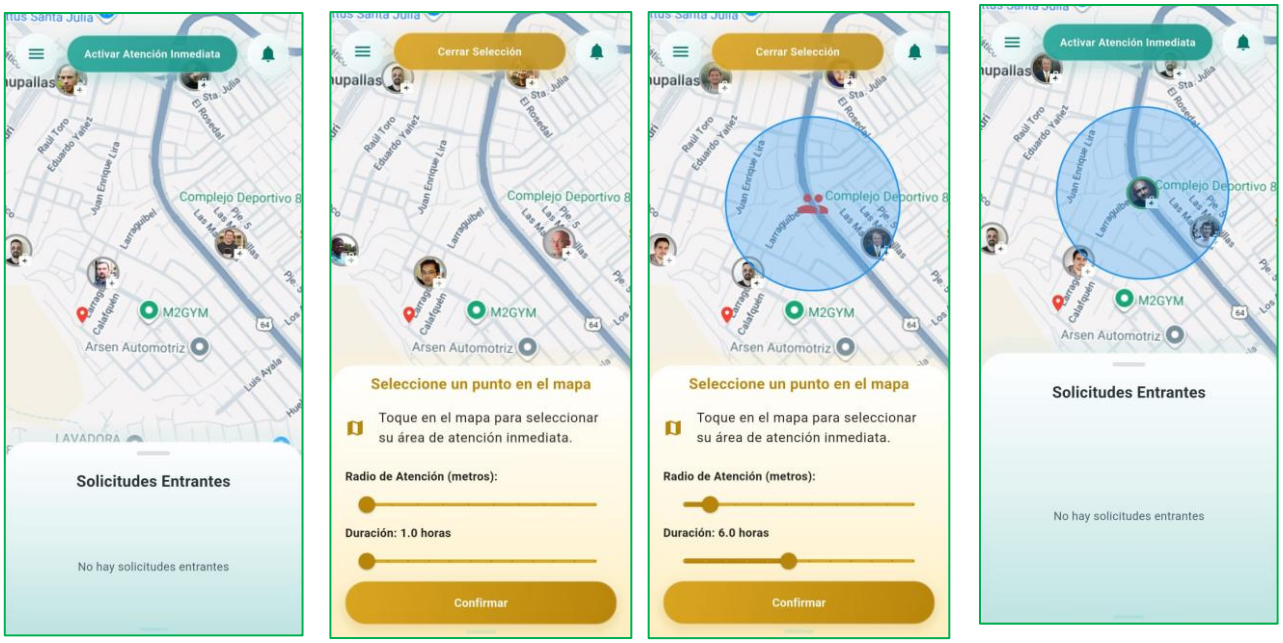
Gracias a los bloques `try-catch`, captura los códigos de error específicos. Esto se expone a la Vista para notificar al usuario mediante textos de alerta.

B. Profesional (Login exitoso y estado de carga)

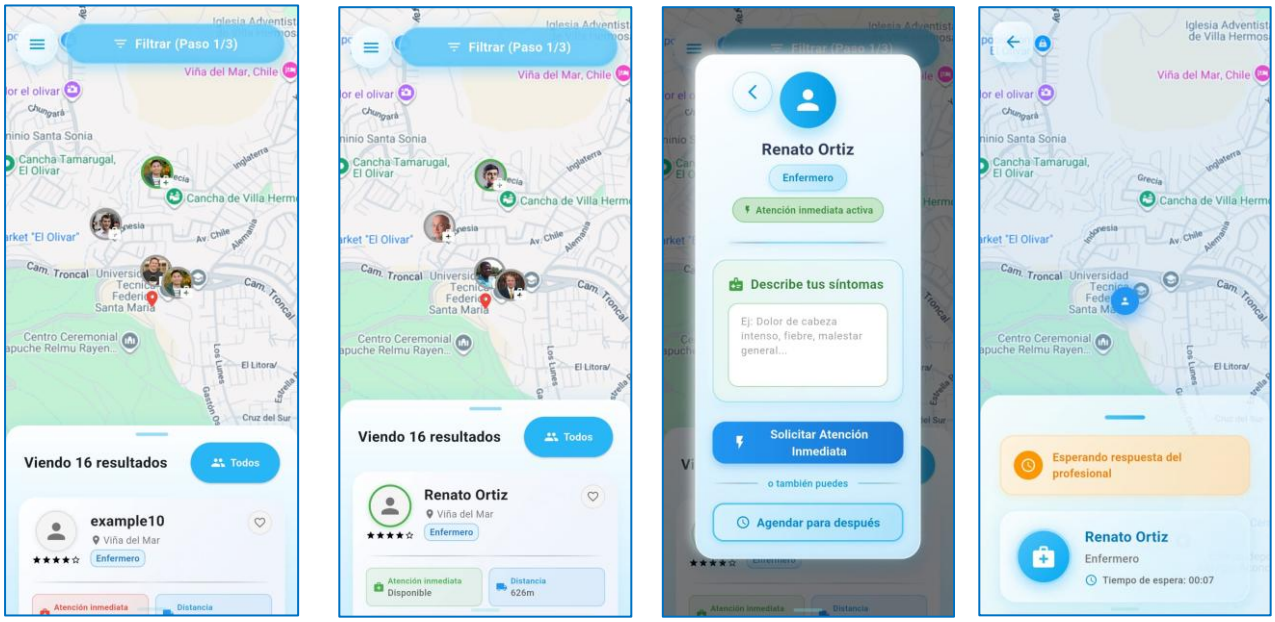


8.2 ANEXO 2. Flujo de Atención Inmediata

A. Profesional (Gestión de Estado y Captura de Eventos Táctiles)

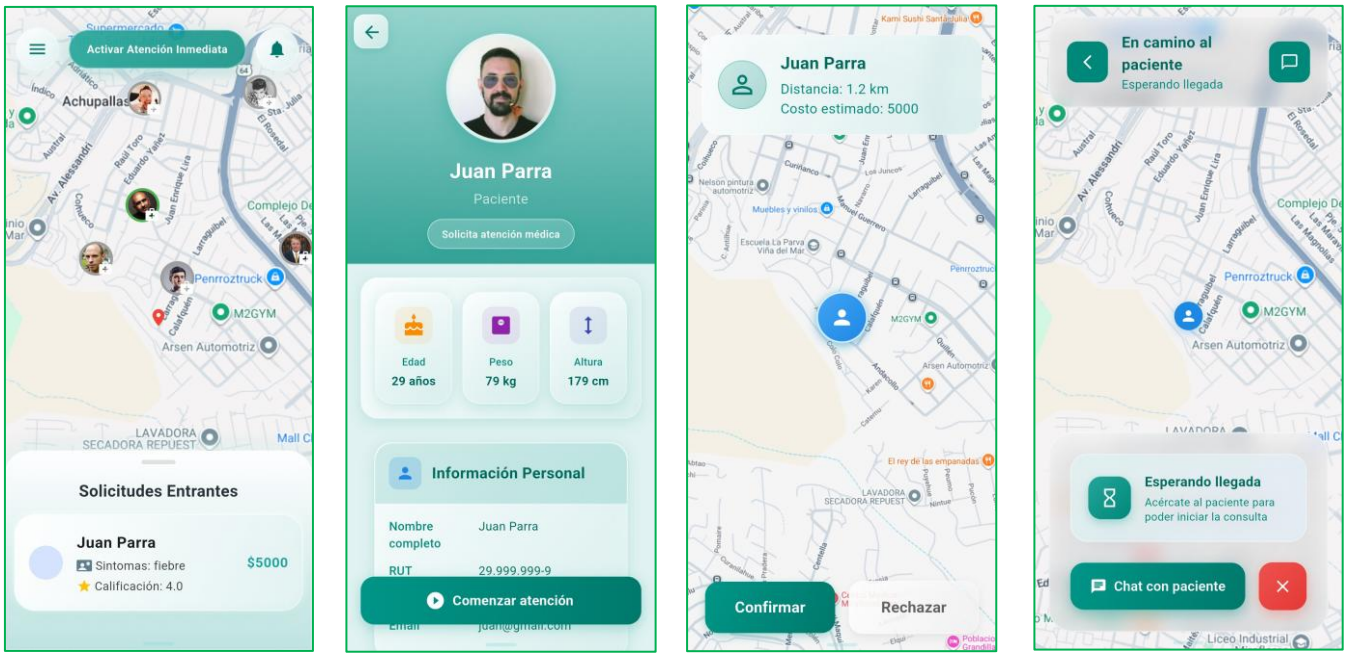


B. Paciente (Selecciona el profesional en el mapa)



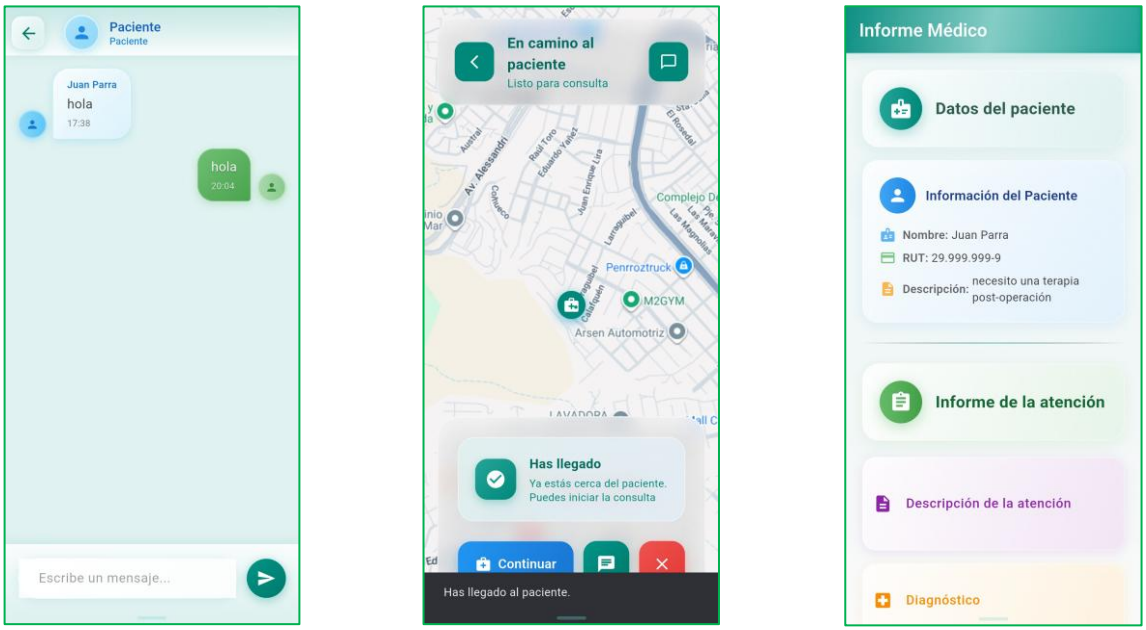
Una vez que el profesional activa su disponibilidad, se crea un marcador en el mapa. Su disponibilidad se distingue visualmente mediante un borde verde (indicador de estado “activo”) alrededor de su fotografía. Una vez se selecciona su foto, se abre el modal, solicita la ‘Atención Inmediata’ y queda en la pantalla de espera.

C. Profesional (Acepta la Solicitud)



El profesional ve la solicitud en su bandeja de entrada, posteriormente ve los datos del paciente y acepta la solicitud viendo su ubicación. Una vez aceptada, se traslada a la vista del trayecto del profesional hacia el paciente.

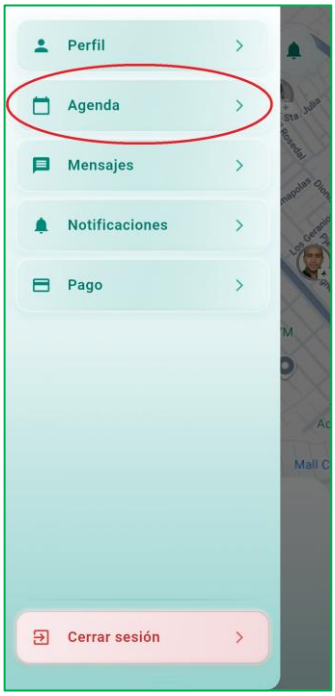
D. Profesional (Realiza la Cita Médica)

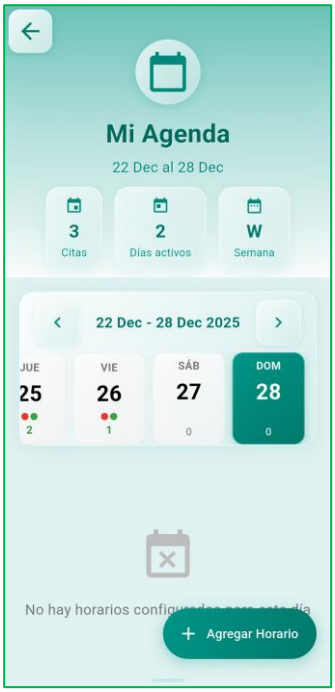


Mientras está en el trayecto puede recibir y enviar mensajes con el paciente, una vez ha llegado a su destino, se habilita el botón “Continuar” y se traslada al informe médico correspondiente a su servicio.

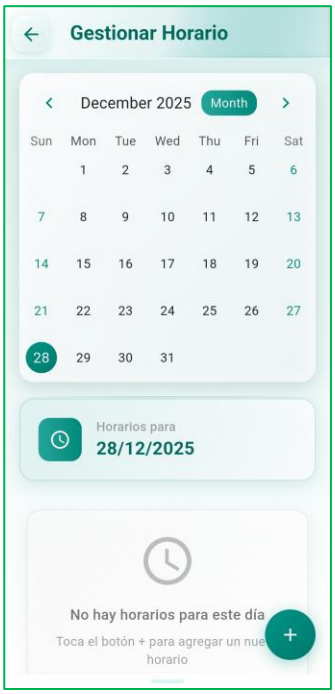
8.3 ANEXO 3: Agendamiento

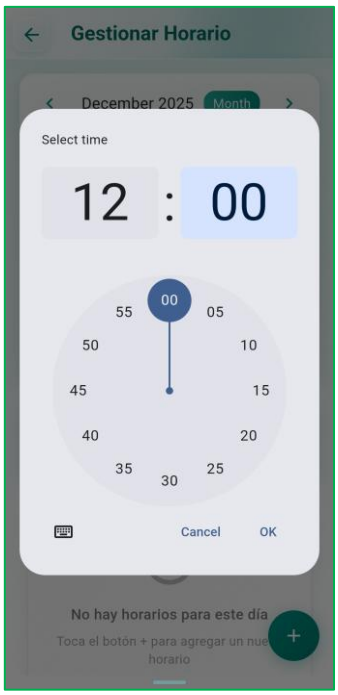
A. Profesional agenda una hora en su perfil

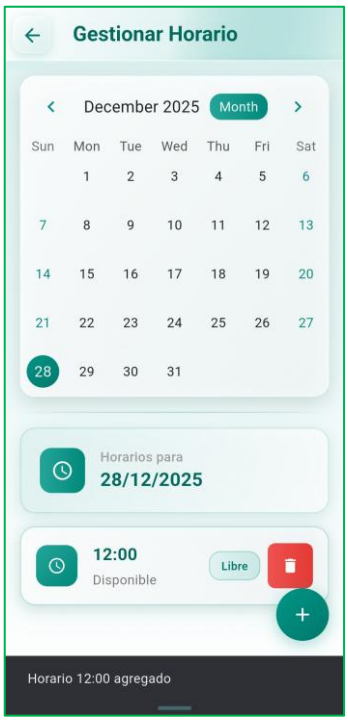




El profesional de la salud selecciona su agenda, en el cual se refleja el número de citas que tiene en la semana, seleccionando la sección "Agregar Horario" entra a otra pantalla que permite gestionar el horario, aquí puede ver todo el movimiento del mes, apretando el *button (+)* establece la hora exacta de atención.

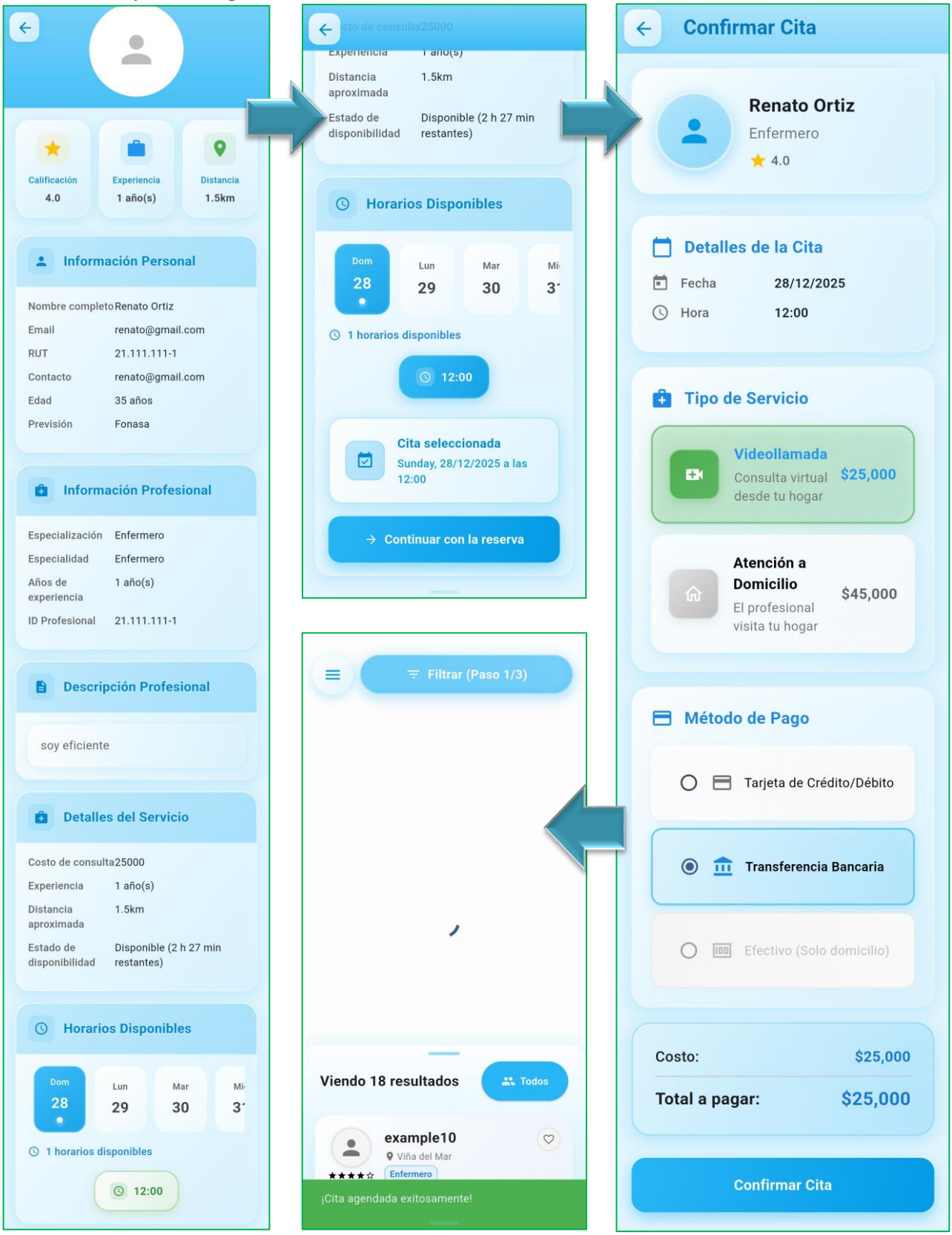






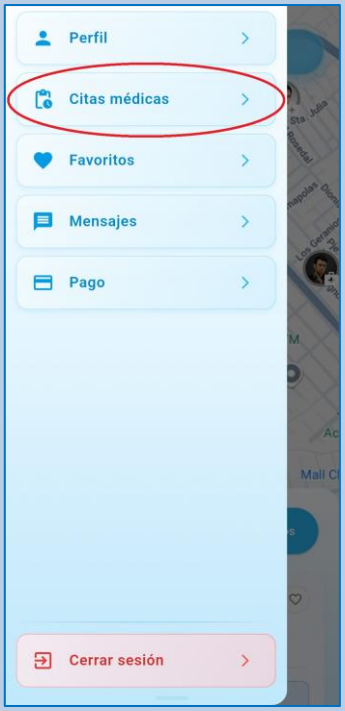
Una vez agregado el horario se establecerá de manera disponible para el paciente en su bandeja de entrada en un formato autónomo o si el usuario lo desea, filtrando por especialidad, horario del día y hora disponible.

B. El paciente agenda la cita médica recién creada

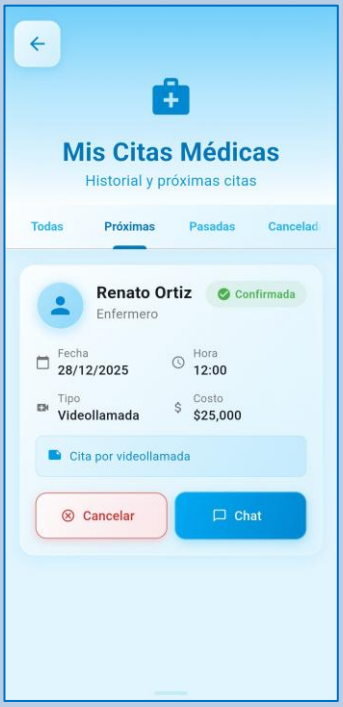


C. Se almacena la cita médica en ambos usuarios

a. Vista Paciente

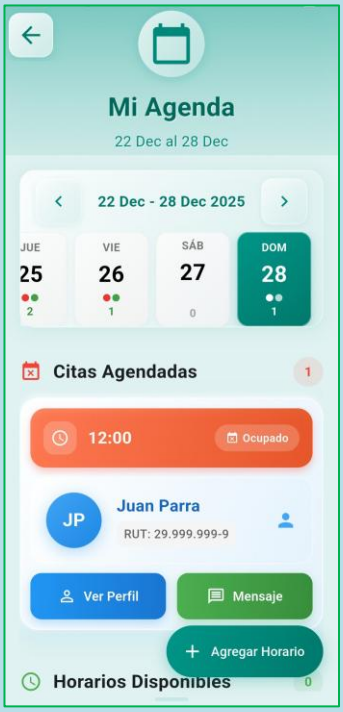


El paciente entra a la sección Del mainDrawer 'Citas médicas' para ver el Estado de la Cita agendada.

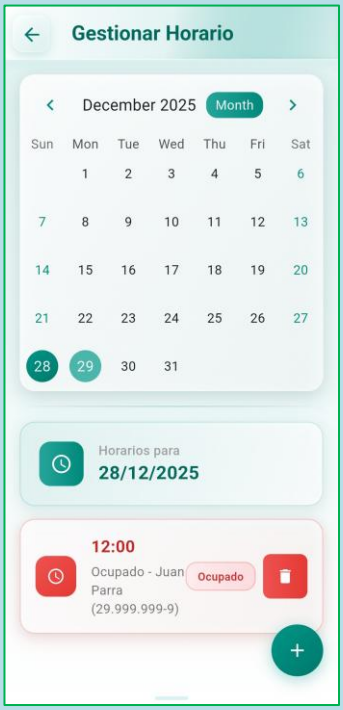


En este apartado, en la sección de 'Próximas' se ve almacenada la cita médica mostrando la información y las funciones precisas.

b. Vista Profesional



La cita médica recién agendada se refleja en la agenda con los siguientes apartados "Ver perfil" para comenzar la atención y "Mensaje" para el contacto con el paciente.



Entrando en el botón de agregar horario (+) se ve reflejado la hora que está ocupada la atención para ese día, logrando que no se repitan dos horarios con la misma hora