

Proyecto Desarrollo Movil: FisioCare

Natalia Rojas

Universidad del Rosario, Bogotá D.C. Colombia

Abstract

FisioCare es una aplicación móvil diseñada para optimizar la gestión de citas de fisioterapia para la fisioterapeuta Cathalina. Actualmente, los procesos de agendamiento y gestión del historial clínico se realizan de manera manual, lo que genera ineficiencias y oportunidades de mejora en la organización del trabajo. La aplicación tiene como objetivo automatizar estos procesos, beneficiando tanto a la profesional de la salud como a sus pacientes.

La app contará con un módulo de agendamiento que permitirá a los pacientes reservar, modificar y cancelar citas de forma sencilla. Además, incluirá un módulo de historia clínica que almacenará y estructurará la información médica de los pacientes de manera segura y accesible. La funcionalidad de telemedicina permitirá la realización de consultas virtuales, ampliando el alcance de la fisioterapeuta a pacientes que no puedan asistir presencialmente. Asimismo, la aplicación integrará un sistema de facturación automática, facilitando la gestión administrativa del consultorio.

Para garantizar la seguridad y eficiencia en el almacenamiento de datos, se utilizará Firebase como base de datos y backend. El desarrollo seguirá una metodología ágil, permitiendo ajustes y mejoras continuas según las necesidades del usuario. Como resultado, FisioCare busca modernizar y optimizar la práctica de fisioterapia, mejorando la eficiencia del trabajo de la fisioterapeuta y ofreciendo una experiencia fluida y accesible a sus pacientes.

Desarrollo movil, Kotlin, Fisioterapia, Aplicación, Android.

1 Definición de Problema

Nuestro cliente actualmente utiliza métodos manuales y poco centralizados para el agendamiento de citas y la gestión del historial clínico de los pacientes. Aunque reconoce que su método actual es funcional, identifica oportunidades para mejorar la eficiencia y la organización de su trabajo. La falta de automatización en el envío de recordatorios y la gestión de información clínica genera una carga adicional y puede afectar la experiencia del paciente. Además, nuestro cliente está interesado en implementar la telemedicina para ampliar su alcance y ofrecer consultas remotas, pero carece de una plataforma confiable para gestionar este servicio.

Se requiere un sistema automatizado para el agendamiento de citas que no dependa exclusivamente de WhatsApp, junto con recordatorios automáticos enviados a los pacientes un día antes de la cita. También es necesario un sistema centralizado para almacenar y acceder al historial clínico y las notas de los pacientes, proporcionando un espacio digital seguro para organizar esta información. Además, se busca crear y establecer una marca

personal para profesionalizar y diferenciar el servicio. En cuanto a la telemedicina, se necesita una plataforma confiable que permita realizar videollamadas y gestionar pagos de consultas virtuales, integrada con el sistema de agendamiento y gestión de historiales clínicos.

2 Objetivos

- **Objetivo General:** Realizar una aplicación móvil utilizando Kotlin y Firebase orientada al agendamiento de citas de fisioterapia para la doctora Cathalina, que además tenga otras funcionalidades como generar recordatorios de las citas agendadas, ver los diferentes tratamientos que ofrece la doctora, la historia clínica de sus pacientes, la opción de telemedicina para los pacientes que no puedan movilizarse y el poder generar facturación de los servicios, para que automatice y organice procesos que actualmente se hacen a mano y puedan tener un mayor control sobre su servicio médico.
- **Objetivos Específicos:**
 1. Diseñar prototipos funcionales de baja y alta fidelidad en Figma, garantizando una interfaz intuitiva y accesible para todos los módulos del sistema, alineados con los estándares de UX/UI.
 2. Ejecutar pruebas de usabilidad con usuarios reales en entornos controlados, aplicando metodologías de Design Thinking para validar el cumplimiento de requerimientos y la eficacia de la experiencia de usuario.
 3. Documentar la arquitectura técnica del sistema, elaborando diagramas de secuencia y flujo para clarificar las interacciones y la estructura general de la aplicación.
 4. Definir la identidad corporativa y foundation del proyecto, estableciendo guías de estilo (color, tipografía, componentes) para mantener coherencia visual en toda la plataforma.
 5. Implementar la interfaz gráfica en Android Studio (Kotlin), traduciendo los diseños aprobados a código funcional, optimizado para dispositivos Android 10 o superiores.
 6. Garantizar eficiencia y escalabilidad, evaluando la arquitectura para optimizar tiempos de respuesta, consumo de recursos y capacidad de integración con futuras actualizaciones.
 7. Generar informes técnicos periódicos, registrando avances, ajustes en diagramas, resultados de pruebas de usabilidad y retroalimentación de stakeholders.
 8. Coordinar el desarrollo mediante Trello, organizando tareas por módulos, priorizando hitos críticos y asegurando el seguimiento transparente del progreso.

3 Alcance

Se desarrollará una aplicación para gestionar citas de fisioterapia de una doctora independiente mediante la aplicación de Android Studio y el lenguaje de programación Kotlin

y una base de datos en Firebase, este proyecto se realizará en un tiempo de cuatro meses y que va a contar con los siguientes módulos: Usuarios, Citas, Tratamiento, Historia Clínica, Telemedicina, Facturación.

4 Metodología

4.1 Enfoque Metodológico

Para el desarrollo del proyecto se empleará la metodología **Kanban**, permitiendo organizar ideas y objetivos en plazos determinados. Este enfoque facilitará la gestión eficiente del flujo de trabajo, asegurando un desarrollo estructurado y adaptable a los requerimientos del proyecto.

El proceso se dividirá en las siguientes fases clave:

4.2 Fase de Análisis

En esta fase, se aplicará la estrategia de **Design Thinking** junto con el **mapa de empatía** para comprender mejor las necesidades del cliente. Se definirá el problema utilizando estrategias de **levantamiento de información**, lo que permitirá plantear posibles soluciones de manera efectiva.

Además, se utilizará el **Mapa de Stakeholders** para identificar a los actores clave involucrados en el proyecto y determinar sus respectivos roles. Finalmente, se definirán los **alcances y objetivos** del proyecto.

Herramientas requeridas:

- **Gestión de tareas:** Trello, Miro.

4.3 Fase de Planeación

Se definirán los **módulos principales** que conformarán la aplicación, junto con sus **requisitos funcionales (RQF)**:

- **Módulo de Citas:** Implementación de gestión de agendamiento.
- **Módulo de Usuarios:** Administración de roles y perfiles.
- **Módulo de Tratamiento:** Registro y seguimiento de tratamientos.
- **Módulo de Historia Clínica:** Almacenamiento y consulta de datos médicos.

Además, se desarrollarán **historias de usuario** para cada módulo, con el fin de definir sus funcionalidades clave. Asimismo, se establecerán los **criterios de aceptación y evaluación** del sistema.

4.4 Fase de Diseño

Se desarrollarán **prototipos de baja fidelidad** para obtener una conceptualización inicial del proyecto. Posteriormente, se crearán **prototipos de alta fidelidad** en Figma para definir con precisión la interfaz de usuario.

Herramientas requeridas:

- **Modelado y diagramas:** Draw.io, Lucid.app.
- **Prototipado:** Balsamiq, Figma.

4.5 Fase de Desarrollo

En esta fase se implementarán las funcionalidades definidas, utilizando los lenguajes de programación requeridos. Se realizará la integración con un servicio de almacenamiento en la nube para gestionar y almacenar los datos de los usuarios de manera segura.

Herramientas requeridas:

- **Desarrollo:** Kotlin.
- **Base de datos:** Firebase.

Cada una de estas fases contribuirá al desarrollo eficiente del proyecto, asegurando que cumpla con los objetivos planteados y las necesidades del usuario final.

5 Alternativas de solución

5.1 Diagrama de flujo por módulo

1. Usuario

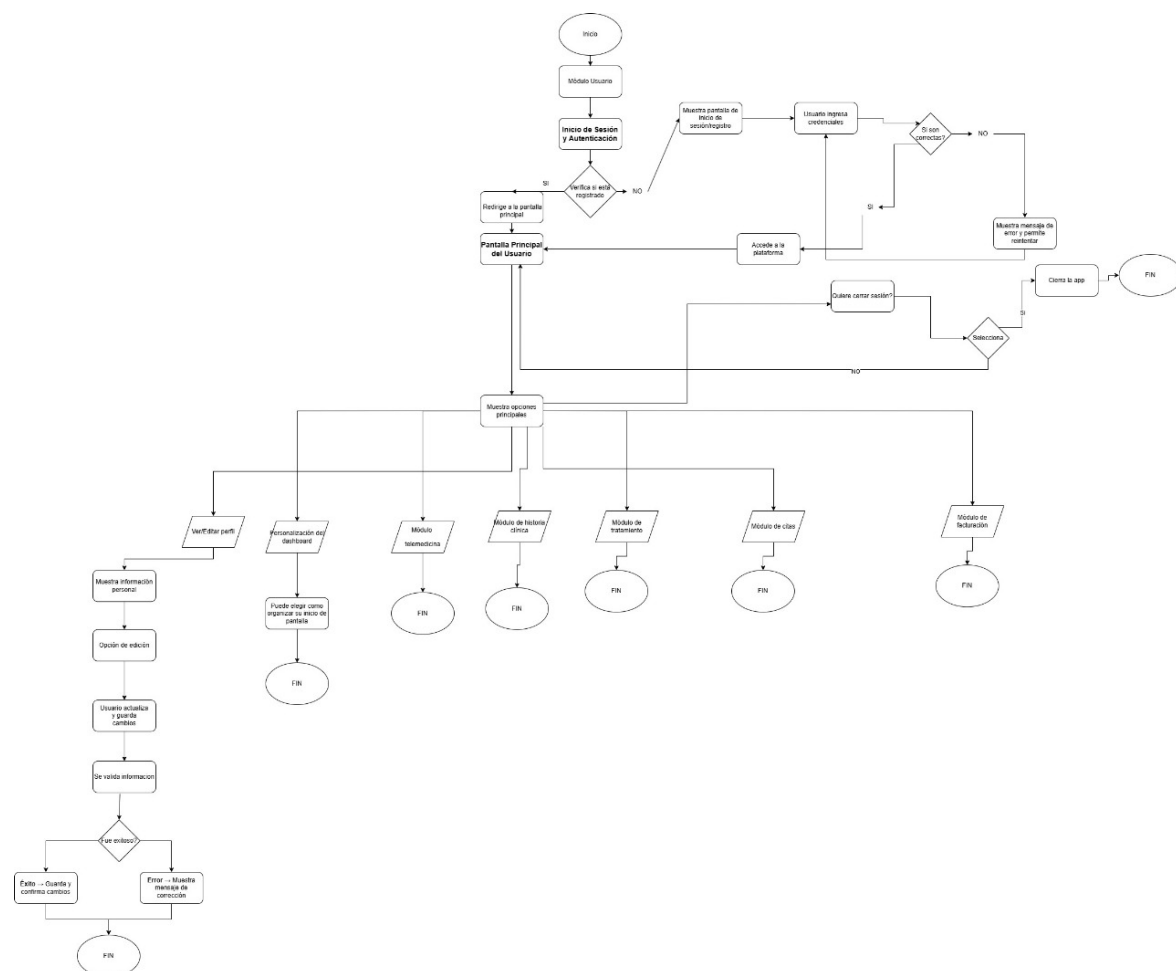


Figure 1: Diagrama de Flujo Módulo Citas

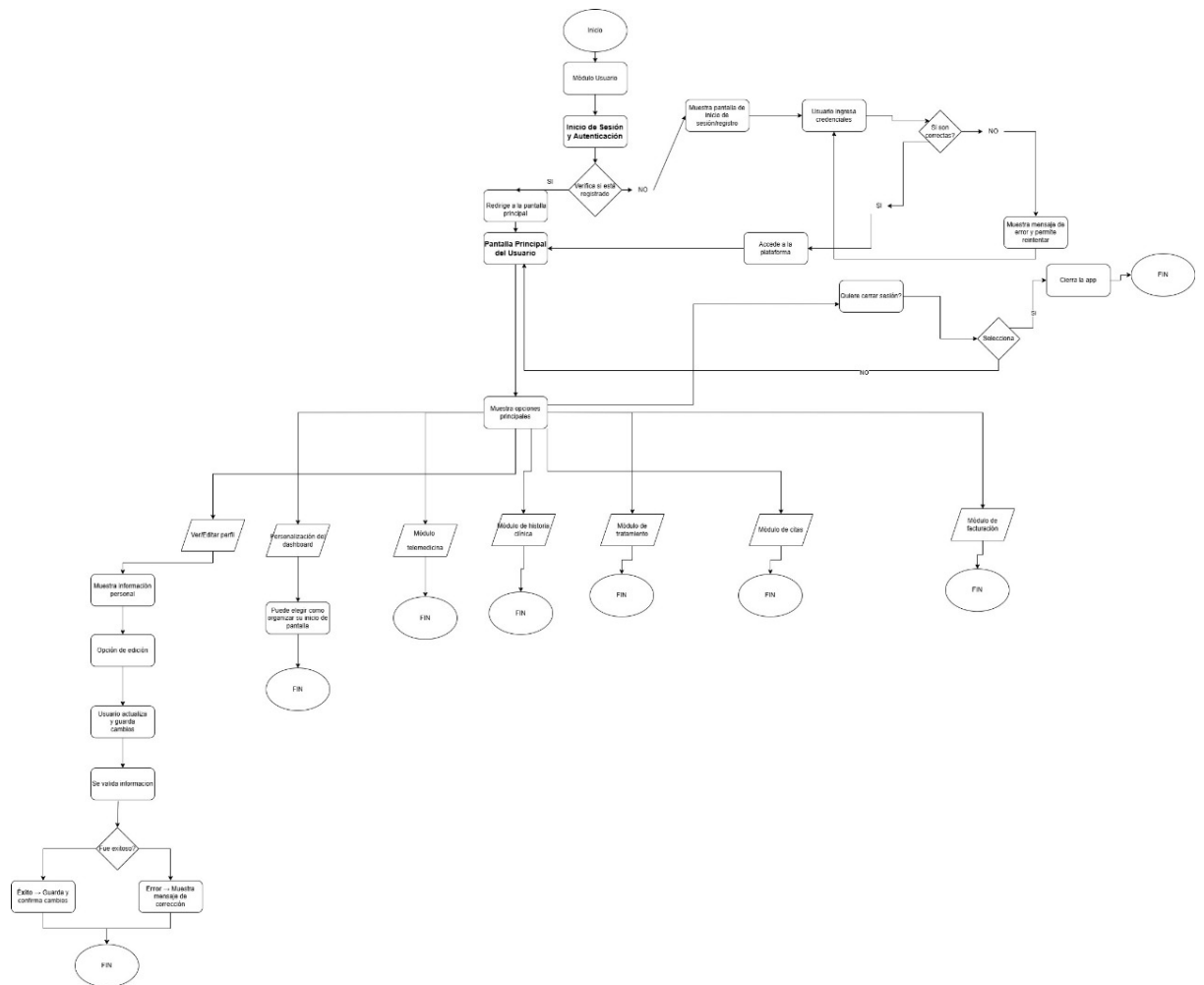


Figure 2: Diagrama de Flujo Módulo Usuario

2. Citas

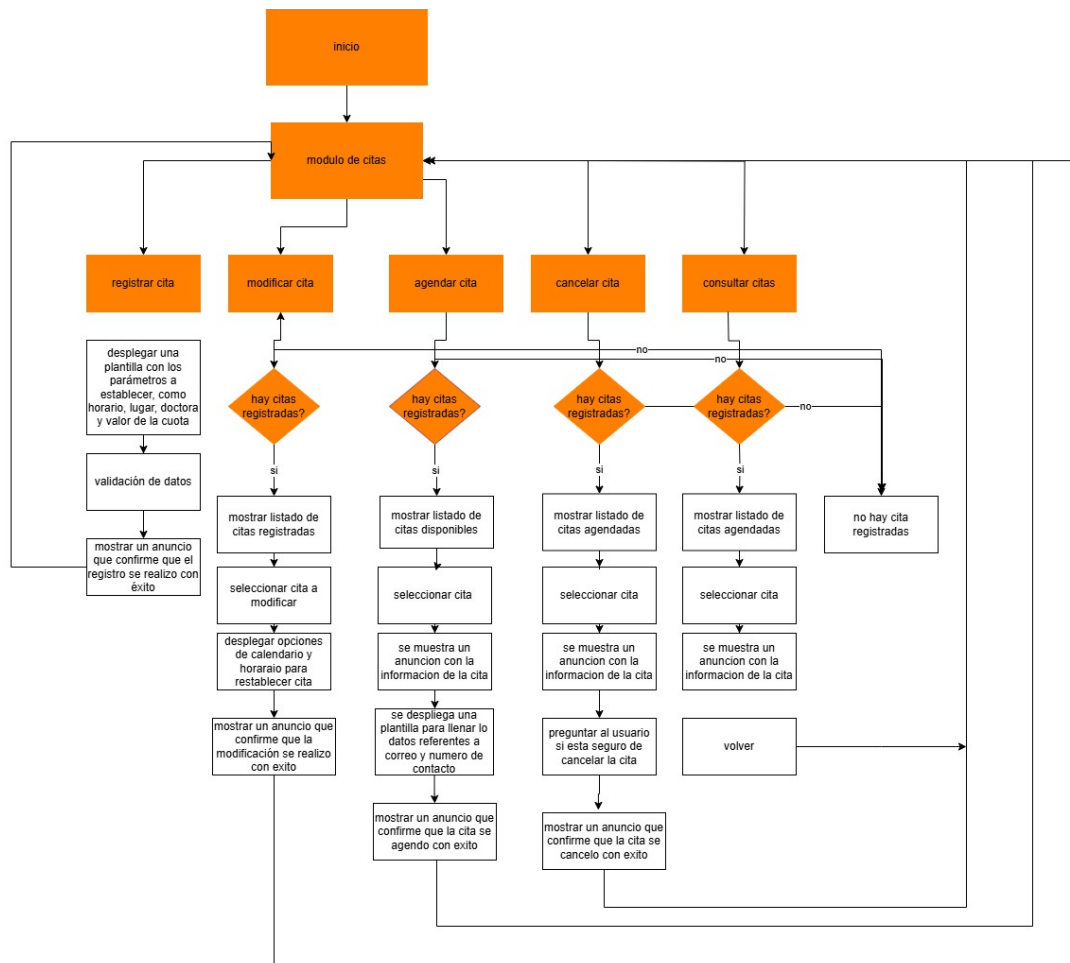


Figure 3: Diagrama de Flujo Módulo Citas

3. Historia Clínica

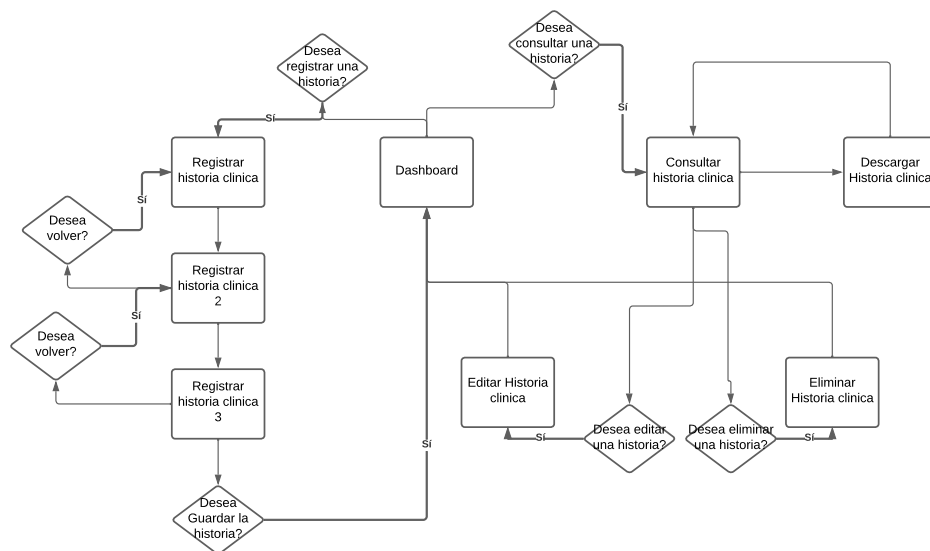


Figure 4: Diagrama de Flujo Módulo Historia Clínica

4. Tratamiento

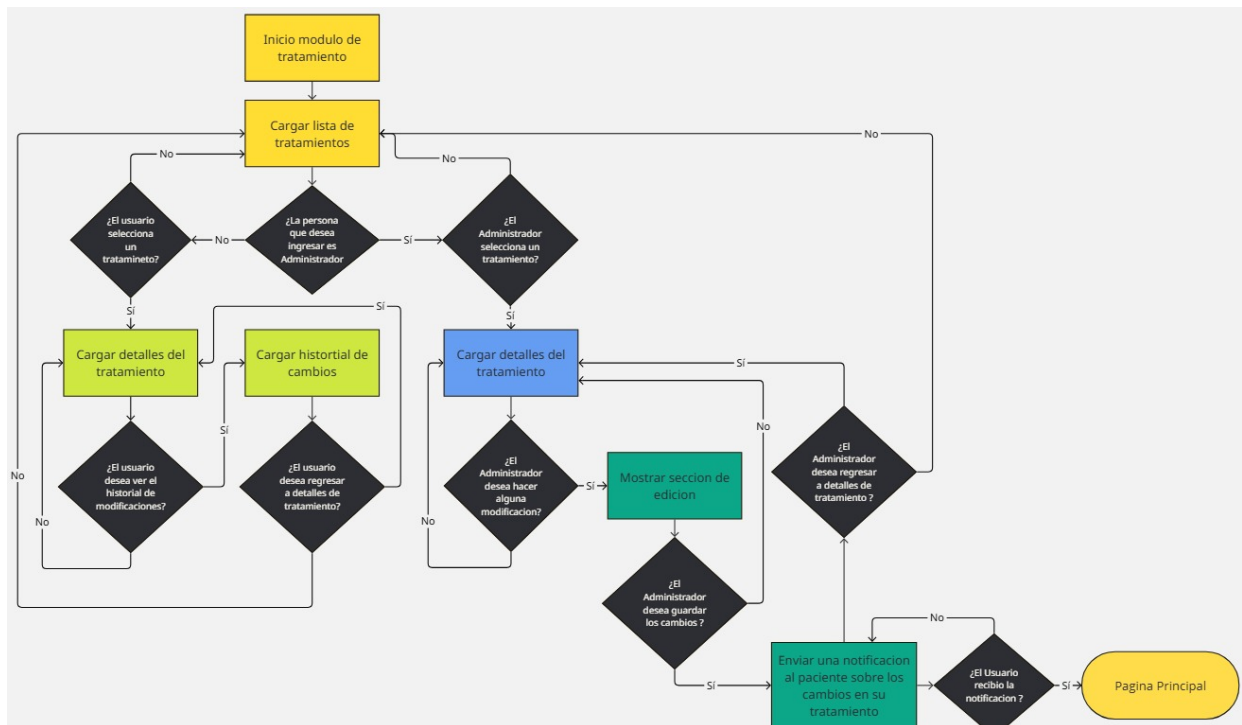


Figure 5: Diagrama de Flujo Módulo Tratamiento

5. Telemedicina

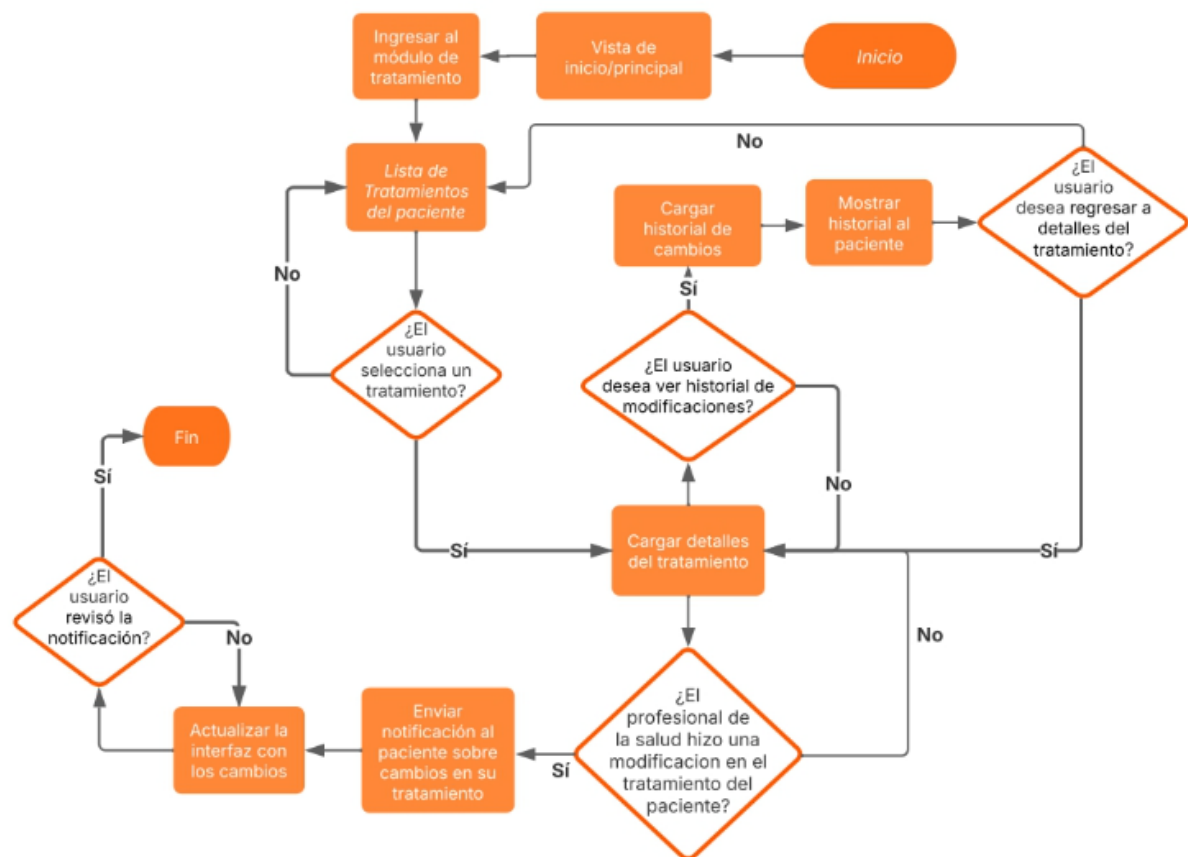


Figure 6: Diagrama de Flujo Módulo Telemedicina

6. Facturación

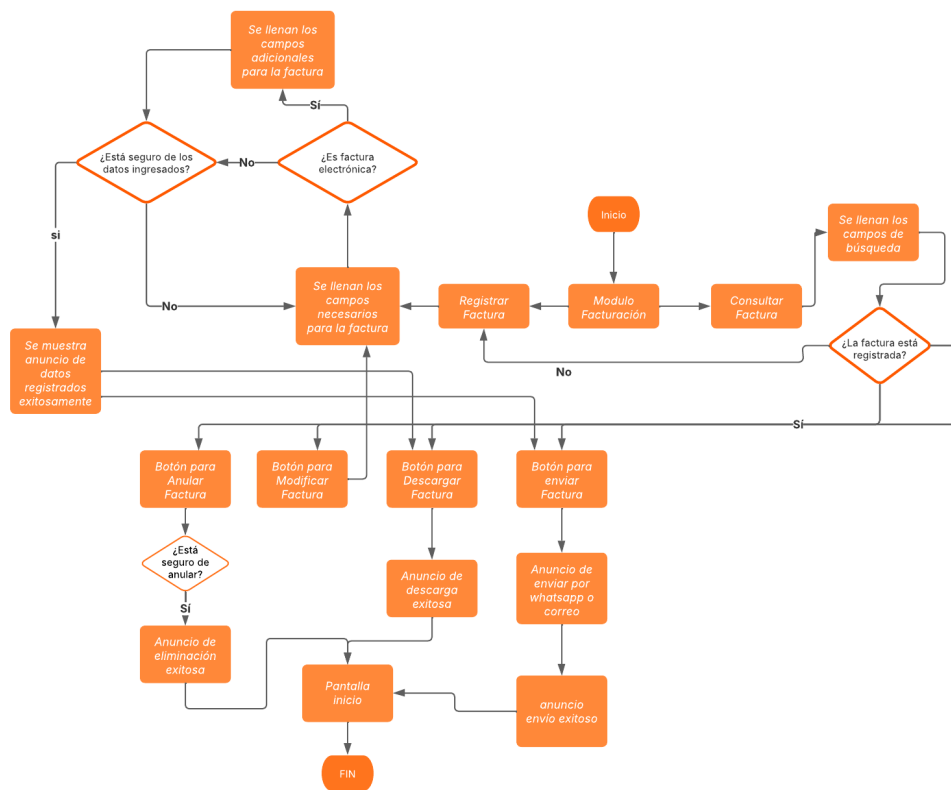


Figure 7: Diagrama de Flujo Módulo Facturación

5.2 Diagrama de Flujo de toda la app

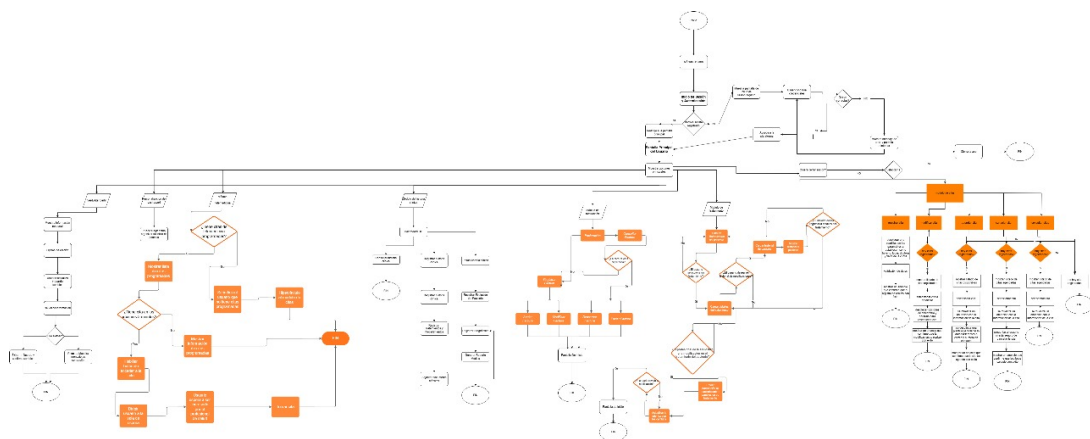


Figure 8: Diagrama de Flujo completo

6 Levantamiento de Información

6.0.1 Introducción

El presente documento tiene como objetivo recopilar la información necesaria para definir los requerimientos y el alcance de una aplicación móvil enfocada en el acompañamiento de citas de fisioterapia. Esta aplicación busca optimizar la gestión de pacientes, permitir la consulta de historiales clínicos y mejorar la accesibilidad a tratamientos a través de herramientas digitales, incluyendo la telemedicina.

Para lograrlo, se ha llevado a cabo un levantamiento de información que incluye entrevistas con profesionales de la salud, revisión de plataformas existentes y análisis de tendencias tecnológicas en el ámbito de la fisioterapia. Esta información servirá como base para la definición de los criterios de evaluación, restricciones y requerimientos del sistema, garantizando que la solución propuesta se alinee con las necesidades del sector.

6.1 Fuentes de Información

6.1.1 Fuentes Primarias

Para obtener información directa sobre los requerimientos y desafíos del sector, se realizó una entrevista con la doctora Cathalina, fisioterapeuta con experiencia en la atención de pacientes y en la gestión de citas médicas. Durante la entrevista, se identificaron necesidades clave, como:

Automatización del agendamiento de citas

- Actualmente, usa WhatsApp manualmente para programar citas.
- Necesita un sistema que permita a los pacientes agendar de forma autónoma sin depender de la comunicación directa.

Recordatorios automáticos de citas

- Actualmente, los pacientes pueden olvidar sus citas porque no hay un sistema que envíe recordatorios.
- Se requiere un sistema que notifique vía WhatsApp o notificaciones en la app.

Almacenamiento seguro de notas e historiales clínicos

- Usa notas en su tablet sin un sistema centralizado.
- Necesita una plataforma digital para guardar, acceder y gestionar las notas e historiales clínicos de cada paciente.

Implementación de telemedicina con seguridad

- Desea ofrecer consultas virtuales para pacientes que no pueden asistir presencialmente.
- Necesita una herramienta que permita videollamadas y gestión de datos de manera segura y profesional.

6.1.2 Fuentes Secundarias

Además de la entrevista con la doctora Cathalina, se realizó una investigación sobre herramientas similares y tendencias en el sector de la fisioterapia digital. Se analizaron:

- Aplicaciones de gestión de clínicas y telemedicina, para identificar funcionalidades clave.
- Normativas de protección de datos médicos, como HIPAA y GDPR, para garantizar la seguridad de la información.
- Estudios sobre el impacto de la digitalización en la fisioterapia, con el fin de evaluar los beneficios y desafíos de la implementación de tecnología en este campo.

6.2 Métodos de Recolección

- Entrevista con la doctora Cathalina, fisioterapeuta, para identificar sus necesidades en la gestión de pacientes, almacenamiento de información y telemedicina.
- Observación de su flujo de trabajo, analizando cómo organiza sus citas, almacena notas clínicas y gestiona la comunicación con los pacientes.
- Análisis de herramientas actuales, como la app de la EPS y WhatsApp, para entender qué funcionalidades resultan útiles y cuáles son sus limitaciones.
- Investigación de plataformas de telemedicina y software de gestión médica, para evaluar buenas prácticas y estándares de la industria.

6.3 Información Recolectada

6.3.1 Necesidades y problemas detectados

- Falta de automatización en la gestión de citas: actualmente depende de WhatsApp y un calendario manual.
- Dificultad para recordar citas a los pacientes: no cuenta con un sistema de recordatorios automáticos.
- Almacenamiento descentralizado de notas clínicas: las notas se guardan en la tablet sin un sistema estructurado.
- Limitaciones en la implementación de telemedicina: necesita una herramienta confiable para hacer videollamadas y gestionar datos de manera segura.

6.4 Requerimientos técnicos

- Compatibilidad con Android, para garantizar acceso desde dispositivos móviles.
- Uso de Kotlin y Firebase, para aprovechar bases de datos en la nube y mejorar la accesibilidad de la información.
- Sistema de notificaciones y recordatorios automatizados, para mejorar la comunicación con los pacientes.

- Módulo de videollamadas seguro, que garantice la privacidad y confidencialidad de la información médica.

6.5 Restricciones y limitaciones

- Seguridad y privacidad de datos médicos: cumplimiento de normativas como HIPAA o GDPR.
- Facilidad de uso para la doctora y los pacientes: debe ser intuitivo, sin necesidad de conocimientos técnicos avanzados.
- Integración con herramientas ya utilizadas: como WhatsApp o sistemas de EPS, si es viable.

6.6 Análisis de la Información

- La doctora Cathalina necesita digitalizar su flujo de trabajo sin cambiar drásticamente sus hábitos.
- La automatización de citas y recordatorios es una prioridad.
- La telemedicina es una opción viable si se garantiza seguridad y facilidad de uso.

6.7 Principales desafíos

- Implementar una solución accesible sin complejidad excesiva.
- Asegurar que la plataforma sea adaptable al flujo de trabajo actual.

7 Stakeholders

- **Satisfacer sus necesidades:** Son los usuarios finales que dependen de la aplicación. Estos stakeholders son críticos para el éxito de la aplicación.
- **Actores clave:** Aseguran el funcionamiento y escalabilidad de la aplicación. Estos stakeholders no usan la aplicación como usuarios finales, pero son cruciales para su funcionamiento.
- **Menos importantes:** No usan la aplicación directamente, pero pueden influir. Estos stakeholders no tienen una relación directa con la aplicación, pero pueden influir en su funcionamiento.
- **Mostrar consideración:** Pueden impactar la percepción y crecimiento. Estos stakeholders pueden influir en la adopción de la aplicación a través de recomendaciones, publicidad o normativas.



Figure 9: Mapa de los Stakeholders

8 Requisitos

8.1 Requisitos Funcionales

Código	Nombre	Descripción	Usuarios
RQF 001	Gestionar Facturas	El sistema permitirá al administrador realizar las acciones de registro, consulta, modificación, anulación, generación de informe, descarga y envío de facturas. El usuario paciente solo podrá consultar sus propias facturas y descargarlas.	Administrador, Paciente
RQF 002	Gestionar Usuarios	El sistema permitirá al usuario paciente y administrador registrarse, iniciar sesión, cerrar sesión, recuperar contraseña, consultar y actualizar sus datos. El administrador podrá eliminar y hacer inactivación de la información en la base de datos.	Administrador, Paciente
RQF 003	Gestionar citas	El sistema permitirá al administrador registrar citas estableciendo fechas, horario, lugar, tipo de tratamiento, tipo de modalidad y consultar las citas agendadas.	Administrador, Paciente

Continúa en la siguiente página

Table 1 – continuación

Código	Nombre	Descripción	Usuarios
RQF 004	Acceder a consulta telemedicina	El sistema permitirá al usuario acceder a su cita programada 5 minutos antes de la hora programada. El paciente será direccionado a una “sala de espera”. El profesional de salud será el que inicie la videollamada. Posterior a la cita, el módulo mostrará el historial de citas atendidas bajo la modalidad de telemedicina.	Administrador, Paciente
RQF 005	Consultar tratamientos asignados	El sistema permitirá a los pacientes visualizar sus tratamientos asignados, incluyendo información como tipo de terapia, sesiones programadas, evolución y observaciones. Los profesionales de salud podrán consultar los tratamientos de sus pacientes.	Paciente, Administrador
RQF 006	Gestionar historia clínica	Permite registrar, consultar, modificar, cargar documentos clínicos como radiografías, informes y órdenes médicas, cargar la firma digital, registrar evolución de la historia clínica de cada paciente con información médica relevante y consultar modificaciones realizadas en las historias médicas.	Administrador
RQF 007	Consultar historia clínica	Permite a los fisioterapeutas acceder a la historia clínica de un paciente registrado en el sistema.	Administrador
RQF 008	Modificar historia clínica	Permite modificar datos de la historia clínica en caso de errores o actualizaciones.	Administrador
RQF 009	Cargar documentos médicos	Permite adjuntar documentos médicos como radiografías, informes y órdenes médicas en la historia clínica del paciente.	Administrador
RQF 010	Registrar historia clínica	Permite registrar la historia clínica de cada paciente con información médica relevante.	Administrador
RQF 011	Registrar evolución del paciente	Permite registrar avances en el tratamiento del paciente dentro de su historia clínica.	Administrador
RQF 012	Gestionar antecedentes médicos	Permite ingresar, consultar y editar antecedentes médicos como alergias, cirugías previas, enfermedades crónicas y otros datos relevantes.	Administrador

Continúa en la siguiente página

Table 1 – continuación

Código	Nombre	Descripción	Usuarios
RQF 013	Gestionar diagnósticos	Permite ingresar, consultar y editar diagnósticos médicos y fisioterapéuticos asociados a la historia clínica.	Administrador
RQF 014	Ver detalle de un tratamiento	El sistema permitirá a los pacientes acceder a los detalles de su tratamiento, mostrando información sobre sesiones realizadas, indicaciones médicas y observaciones. Los profesionales de salud podrán ver esta información para hacer seguimiento.	Paciente, Administrador
RQF 015	Consultar historial de tratamientos	Permite llevar un registro de los cambios realizados en una historia clínica y quién los realizó.	Paciente, Administrador
RQF 016	Registrar Usuario	El sistema permitirá a los usuarios registrarse proporcionando nombre, correo electrónico, teléfono y contraseña. Se validará que los datos sean correctos y únicos. Se enviará un correo de verificación para activar la cuenta.	Paciente, Administrador
RQF 017	Iniciar Sesión	Los usuarios podrán iniciar sesión ingresando su correo electrónico y contraseña. El sistema validará las credenciales y otorgará acceso según su rol.	Paciente, Administrador
RQF 018	Recuperar Contraseña	El usuario podrá solicitar la recuperación de su contraseña ingresando su correo electrónico. Se enviará un enlace de restablecimiento con un tiempo de expiración.	Paciente, Administrador
RQF 019	Gestionar Perfil	Los usuarios podrán actualizar su información personal, como nombre, teléfono y foto de perfil. No podrán modificar su correo una vez registrado.	Administrador
RQF 020	Cerrar Sesión	El usuario podrá cerrar sesión de manera segura desde la aplicación, eliminando cualquier sesión activa en el dispositivo.	Administrador
RQF 021	Eliminar la Cuenta	Los usuarios podrán solicitar la eliminación de su cuenta. El sistema pedirá confirmación antes de eliminar permanentemente los datos.	Administrador

Continúa en la siguiente página

Table 1 – continuación

Código	Nombre	Descripción	Usuarios
RQF 022	Diferenciar Roles y Permisos	El sistema diferenciará entre profesionales de salud y pacientes, otorgando permisos específicos. Los profesionales podrán gestionar citas, tratamientos y telemedicina, mientras que los pacientes solo podrán visualizar su información.	Administrador, Pacientes
RQF 023	Registrar firma digital	Permite que los fisioterapeutas firmen digitalmente documentos de la historia clínica.	Administrador
RQF 024	Registro de información de cuidado de tratamiento	Permite agregar información de apoyo de manera gráfica y/o escrita.	Administrador
RQF 025	Registrar y modificar tratamientos	El sistema permitirá a los profesionales de salud asignar nuevos tratamientos a los pacientes, modificar sesiones, actualizar indicaciones médicas y agregar observaciones sobre la evolución del paciente.	Administrador
RQF 026	Notificación de cambios en el tratamiento	El sistema enviará una notificación a los pacientes cuando su tratamiento sea modificado, indicando los cambios realizados, como ajustes en sesiones o nuevas indicaciones médicas.	Paciente
RQF 027	Historial de modificaciones del tratamiento	El sistema registrará todas las modificaciones realizadas en un tratamiento, incluyendo cambios en sesiones, indicaciones médicas y observaciones, permitiendo su consulta por parte del profesional de salud y del paciente.	Administrador, Paciente
RQF 028	Gestionar tratamientos	El administrador podrá gestionar la información de los tratamientos, verificando registros, eliminando tratamientos erróneos y asignando permisos de modificación a los profesionales de salud.	Administrador
RQF 029	Modificar citas	El administrador podrá modificar las citas agendadas cambiando lugar, fecha, hora, tipo de profesional y otros detalles.	Administrador
RQF 030	Cancelar citas registradas	El usuario podrá cancelar citas, eliminándolas de la base de datos.	Usuario, Administrador
RQF 031	Agendar citas	El sistema permitirá consultar disponibilidad de citas registradas en la app mediante opciones de búsqueda.	Paciente

Continúa en la siguiente página

Table 1 – continuación

Código	Nombre	Descripción	Usuarios
RQF 032	Recibir notificaciones de citas	El sistema enviará notificaciones recordatorias a los pacientes sobre sus citas.	Paciente
RQF 033	Cancelar citas	El administrador podrá cancelar una cita registrada.	Administrador
RQF 034	Ingresar a cita telemedicina	El paciente podrá acceder a su cita 5 minutos antes de la hora programada.	Paciente
RQF 035	Ingresar a cita telemedicina (Administrador)	El administrador podrá acceder 15 minutos antes de la cita programada.	Administrador
RQF 036	Ver lista de citas telemedicina	El sistema le permite al paciente ver una lista de citas programadas bajo la modalidad de telemedicina.	Paciente
RQF 037	Adjuntar archivos en cita	El sistema le permite al usuario adjuntar archivos durante la videollamada.	Administrador, Paciente
RQF 038	Enviar mensajes en cita	El sistema le permite al usuario enviar mensajes durante la videollamada.	Administrador, Paciente
RQF 039	Activar/desactivar cámara	El sistema le permite al usuario activar o desactivar su cámara durante la videollamada.	Administrador, Paciente
RQF 040	Activar/desactivar micrófono	El sistema le permite al usuario activar o desactivar su micrófono durante la videollamada.	Administrador, Paciente

8.2 Requisitos no Funcionales

Código	Nombre	Descripción
RQNF 001	Dispositivo	Se requiere de un dispositivo móvil con sistema operativo Android 10+.
RQNF 002	Conexión a internet	Se requiere que el dispositivo se encuentre conectado a internet.
RQNF 003	Soportar archivos	Se requiere que para la descarga de alguna factura o reporte el dispositivo debe soportar la exportación de informes en PDF, Excel y CSV.

9 Arquitectura de Software

Para el proyecto implementamos la arquitectura MVVM (Model-View-ViewModel) que es un patrón de diseño muy usado en desarrollo de aplicaciones, especialmente en entornos como Android o aplicaciones con interfaces gráficas. Su objetivo es separar la lógica de la interfaz gráfica, facilitando el mantenimiento, pruebas y escalabilidad del código.

9.1 Estructura extendida MVVM

- **Model:** Contiene las clases de datos (por ejemplo, User, Product) y representa las entidades que usarás.
- **Database (data/local):** Implementación de Room u otra base de datos local. Define las entidades (@Entity) y DAOs (@Dao). Se comunica solo con el Repository.
- **Repository:** Punto único de acceso a los datos.
Decide si obtener datos de la red, base local, caché, etc.
Se comunica con DAOs o APIs y le da los datos al ViewModel.
Ejemplo: UserRepository, ProductRepository.
- **ViewModel:** Pide datos al Repository. Expone datos como LiveData, StateFlow, etc. para que la View los observe.
Contiene lógica de UI pero no conoce la View directamente.
- **View:** Fragments, Activities o Layouts.
Observa los datos del ViewModel.
Reacciona a eventos del usuario y los envía al ViewModel.
- **Adapter (UI/adapters):** Adaptadores de RecyclerView, ViewPager, etc.
Transforman los datos del ViewModel a vistas individuales.
Se conectan con la View, pero no contienen lógica de negocio.

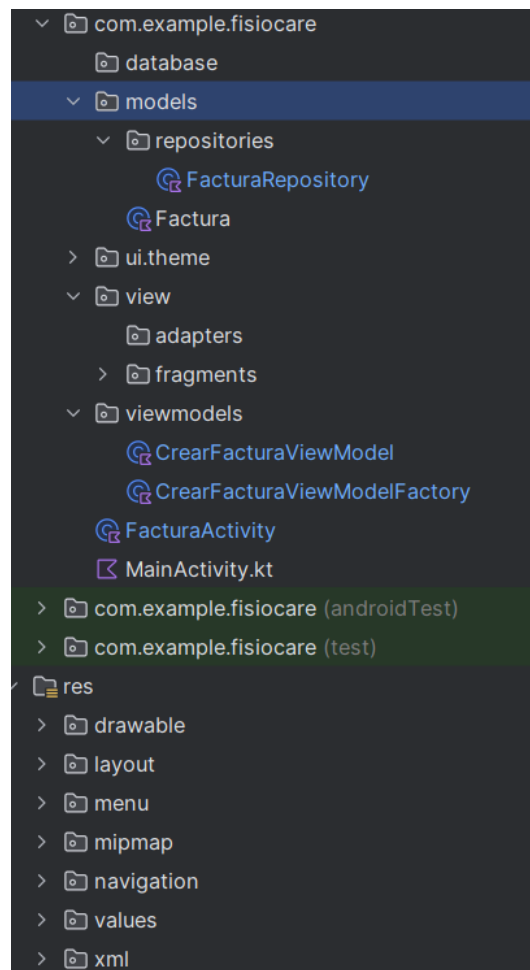


Figure 10: Ejemplo de implemetación

9.2 Diagramas de Arquitecturas

Diseñamos unos diagramas de arquitectura para cada módulo:

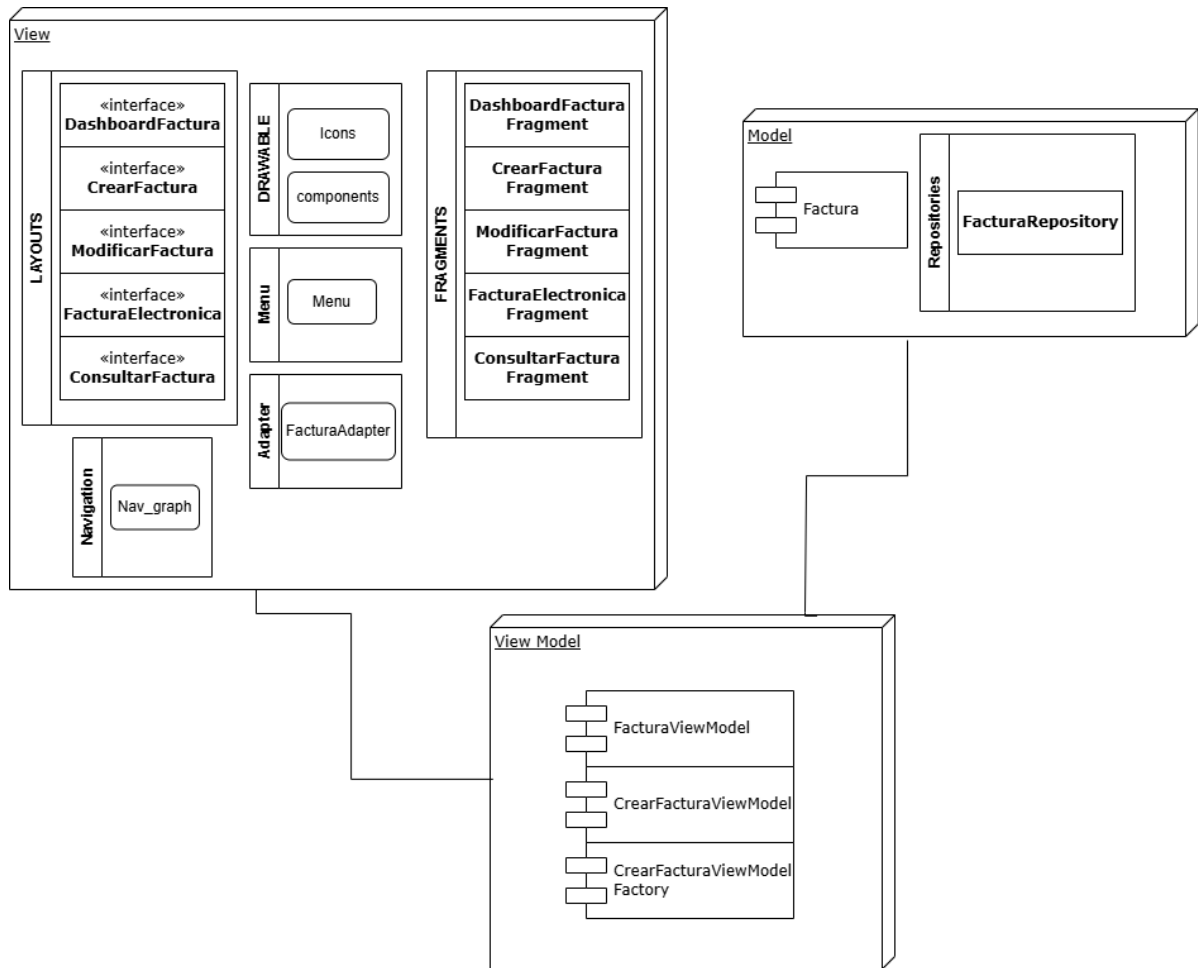


Figure 11: Diagrama MVVM Factura

9.3 Diagrama de Secuencias

Basado en la arquitectura MVVM, en los diagramas anteriores y en los caso de uso, diseñamos los siguientes diagramas de secuencia para observar cómo interactúan los objetos o componentes del sistema a lo largo del tiempo, enfocándose en el orden de los mensajes que se envían entre ellos para realizar una tarea o caso de uso.

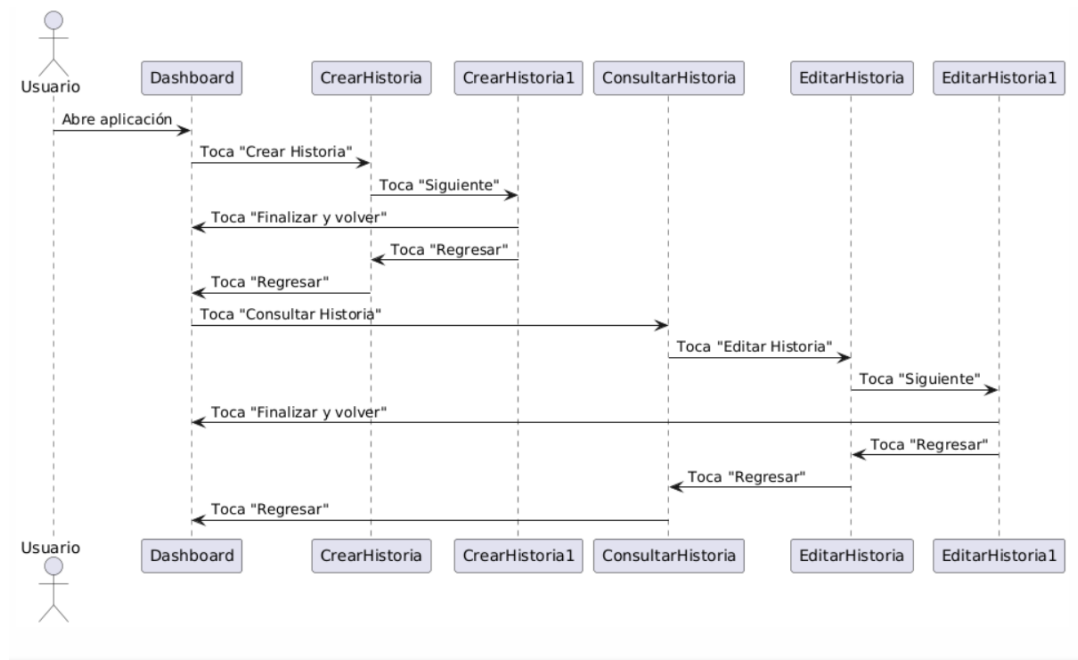


Figure 13: Diagrama de secuencia Historia Clínica

10 Desarrollo Interfaces

- Se desarrollaron las interfaces de usuario en Kotlin, implementando la estructura visual y la navegabilidad entre pantallas, sin funcionalidad completa.
- El diseño se basó en el prototipo de alta fidelidad, incorporando ajustes derivados del testing realizado previamente.
- Se crearon componentes gráficos compartidos como la carpeta `menu` y componentes en la carpeta `drawable`, con el objetivo de mantener coherencia visual entre los distintos módulos.
- Cada módulo fue implementado por un integrante del equipo y se encuentra alojado en su respectiva rama del repositorio de GitHub.
- Se aplicó la arquitectura MVVM para garantizar escalabilidad, organización y separación de responsabilidades.

11 Design Thinking

11.1 Empatía

El mapa de empatía nos ayuda a comprender qué siente, piensa y necesita la profesional de la salud (fisioterapeuta) al gestionar sus citas, permitiéndonos diseñar una app que realmente solucione sus problemas y se adapte a su rutina.

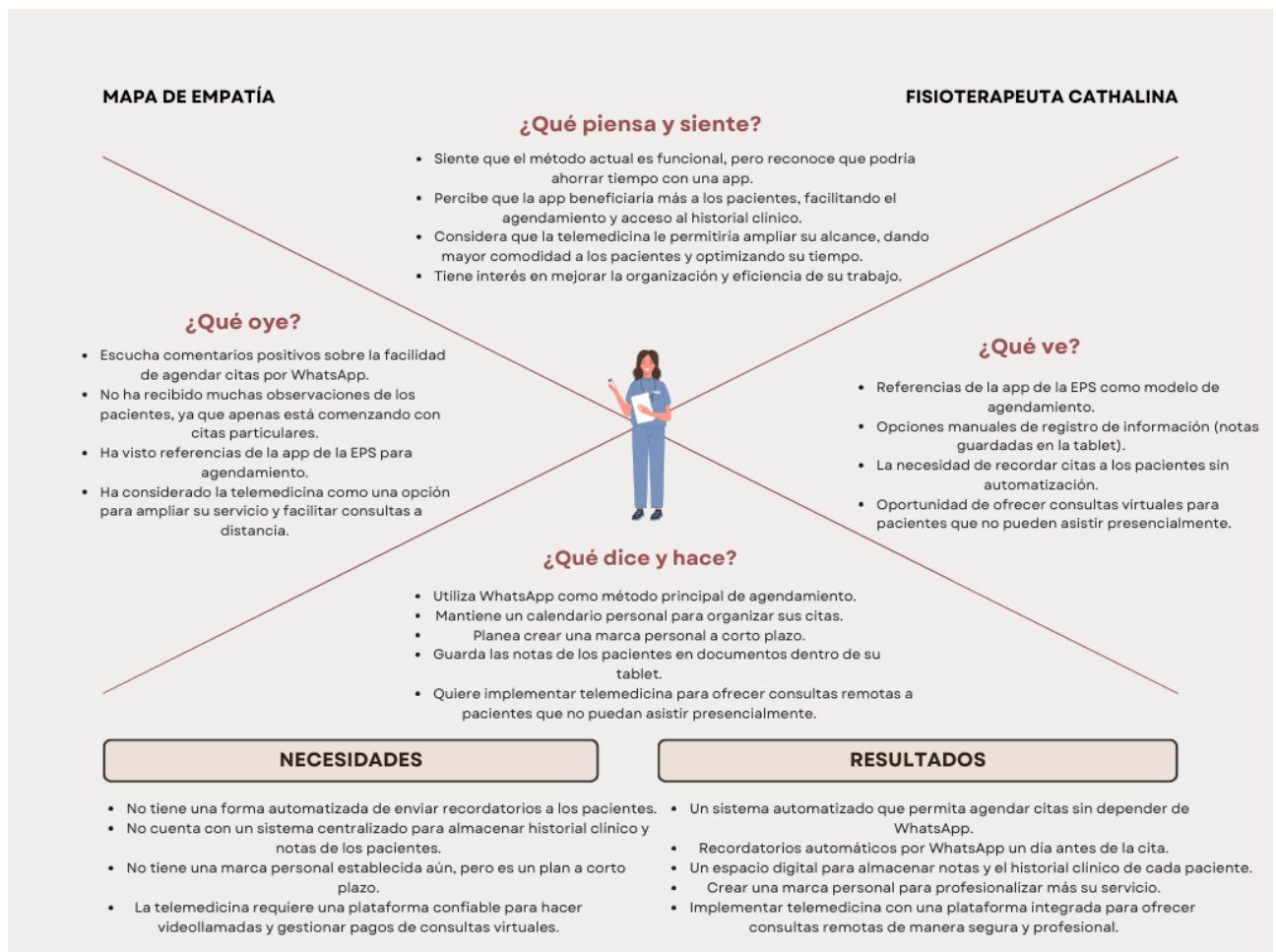


Figure 14: Mapa de Empatía

11.2 Definir

En esta fase, buscamos definir las necesidades del usuario en un enunciado concreto para enfocar las soluciones, para esto realizamos un tablero de ideas donde cada miembro del equipo escribió en notes las necesidades que identificó de la fase de empatía. Usamos una plantilla de un tablero de notas que se puede encontrar en:

Tablero de ideas en Miro

11.3 Idear

La fase de Idear es el momento en el que se generan posibles soluciones para el problema definido. Transformar el problema identificado en ideas concretas que puedan convertirse en soluciones funcionales. En esta fase, utilizamos la estrategia de historias de usuario, las cuales nos permiten describir cómo interactuarán los usuarios con la aplicación. Las historias de usuario del proyecto se organizan y visualizan en Trello, lo que facilita la gestión del desarrollo y la priorización de tareas.

11.4 Prototipado

La fase de prototipado consiste en crear una versión inicial del sistema con el fin de visualizar y validar ideas antes del desarrollo completo. Esta etapa permite recolectar retroalimentación temprana de usuarios o stakeholders, identificar errores y ajustar funcionalidades, reduciendo así riesgos y malentendidos en fases posteriores. En este caso, utilizamos prototipos de baja y alta fidelidad.

- **Prototipos de baja fidelidad:** Para estos prototipos utilizamos herramientas como Balsamiq o Canva, donde cada miembro del equipo realizó unas vistas según el diagrama de flujo correspondiente. Estos prototipos se pueden encontrar en el repositorio GitHub en el siguiente enlace: Prototipos Baja Fidelidad.
- **Prototipos de alta fidelidad:** Para estos prototipos utilizamos la herramienta de Figma, definiendo un Foundation para la aplicación y donde desarrollamos la navegabilidad entre las vistas de todos los módulos para poder hacer la simulación de la aplicación completa. Estos prototipos se pueden encontrar en el en el siguiente enlace: Prototipos Alta Fidelidad.

11.5 Testeo

La fase de testeo se enfoca en verificar que el software funcione correctamente y cumpla con los requisitos definidos. Se realizan pruebas en distintos usuarios para detectar errores, validar el comportamiento del sistema y asegurar su calidad, rendimiento y estabilidad antes de su desarrollo y despliegue. Para esta fase cada miembro del equipo realizó un pertinente testeo con el prototipo de Alta Fidelidad del módulo correspondiente en el rol de patient. Adicionalmente, se realizó un testeo de toda la aplicación a Cathalina en el rol de Administrador. De lo anterior se utilizaron las técnicas de "Prueba de Usabilidad" combinado con la "Evaluación de la Experiencia".

La prueba de usabilidad consiste en evaluar qué tan fácil y eficiente resulta para los usuarios interactuar con un sistema o aplicación. Se enfoca en aspectos como la facilidad de aprendizaje, la efectividad para completar tareas, la rapidez, los errores cometidos y la satisfacción general. Por otro lado, la experiencia de usuario abarca la percepción global que tiene el usuario al interactuar con el producto, incluyendo aspectos emocionales, estéticos y funcionales. No se limita solo a la interfaz, sino a todo el recorrido del usuario: desde el primer contacto hasta la interacción final, buscando generar una experiencia positiva, fluida y satisfactoria.

Toda esta evaluación la recopilamos en una plantilla diseñada en canva que se puede encontrar en el repositorio en el parte de TESTEO al igual que en el siguiente enlace: Fase de Testing.

12 Cogido app Tratamientos

Estructura y Componentes Principales La aplicación está organizada para separar sus diferentes partes, siguiendo un enfoque que incorpora el patrón de arquitectura ****MVVM** (Model-View-ViewModel)**. Los componentes clave son:

- **Actividades (Screens):** Son las pantallas que el usuario ve y con las que interactúa.

- **MainActivity.kt:** Es la pantalla principal de la aplicación. Muestra una lista de tratamientos obtenidos de la base de datos. Permite hacer clic en un tratamiento para ver sus detalles y tiene un botón para agregar nuevos tratamientos.
 - **DetalleTratamientoActivity.kt:** Muestra la información detallada de un tratamiento específico. Incluye opciones para ver el historial o modificar el tratamiento.
 - **HistorialTratamientoActivity.kt:** Muestra un resumen o detalles del historial de un tratamiento.
 - **ModificarTratamientoActivity.kt:** Permite al usuario editar los detalles de un tratamiento existente, incluyendo el número de sesiones y la descripción. Gestiona diálogos para confirmar y guardar los cambios en la base de datos.
 - **AgregarTratamientoActivity.kt:** Una nueva pantalla con un formulario para que el usuario introduzca y guarde los detalles de un nuevo tratamiento en la base de datos.
 - **BaseActivity.kt** Una actividad base de la que heredan las demás. Configura una estructura de pantalla común, que incluye un espacio para el contenido y una barra de navegación inferior, además de un botón de "atrás" en la barra superior.
- **Layouts (Diseños de Pantalla):** Archivos '.xml' que definen cómo se ve cada pantalla.
 - **activitytratamientos.xml:** El diseño principal de MainActivity para mostrar la lista de tratamientos.
 - **activitydetalletratamiento.xml:** Diseño para la pantalla de detalles del tratamiento. Incluye 'TextView's para mostrar el número de sesiones y la descripción.
 - **activityhistorialtratamiento.xml:** Diseño para la pantalla del historial del tratamiento.
 - **activitymodificartratamiento.xml:** Diseño para la pantalla de modificación del tratamiento, con 'TextView's y 'EditText's intercambiables para los campos editables y botones de edición.
 - **activityagregartratamiento.xml:** Diseño para el formulario de adición de un nuevo tratamiento.
 - **dialogtratamiento.xml:** Diseño para los diálogos de confirmación personalizados.
 - **cardtratamiento.xml:** Diseño de una única "tarjeta" de tratamiento que se repite en la lista principal.
 - **Modelos de Datos ('model'):** Tratamiento.kt': Una clase 'data class' que define cómo se representa un tratamiento (ID, fecha, especialidad, doctor, número de sesiones, descripción). Ahora está anotada como una '@Entity' de Room y es '@Parcelize' para ser pasada entre actividades.

- **ViewModels ('viewmodels/')**: Son las clases que contienen la lógica de negocio y gestionan los datos para sus respectivas Actividades, proporcionando datos observables.
 - **TratamientosViewModel.kt**: Gestiona la lista de todos los tratamientos para 'MainActivity'. Lee los datos de la base de datos a través del 'Repository'.
 - **DetalleTratamientoViewModel.kt**: Gestiona los datos y la lógica para la pantalla de detalles del tratamiento.
 - **HistorialTratamientoViewModel.kt**: Gestiona los datos y la lógica para la pantalla del historial del tratamiento.
 - **ModificarTratamientoViewModel.kt**: Gestiona los datos y la lógica para la pantalla de modificación del tratamiento. Es responsable de la actualización de un tratamiento existente en la base de datos.
 - **AgregarTratamientoViewModel.kt**: Gestiona la lógica para ****agregar**** un nuevo tratamiento a la base de datos.
- **Repositorios ('repositories/')**: TratamientosRepository.kt: Actúa como una capa de abstracción entre los ViewModels y las fuentes de datos (en este caso, la base de datos Room). Centraliza la lógica de acceso a datos (obtener, insertar, actualizar, eliminar).
- **Acceso a Datos ('dao/')**: TratamientoDao.kt: Es una interfaz de Room ('@Dao') que define los métodos para interactuar directamente con la tabla 'tratamientos' en la base de datos [ej. 'getAllTratamientos()', 'insert()', 'update()', 'deleteById()'].
- **Base de Datos ('database/')**: AppDatabase.kt: Una clase abstracta que extiende 'RoomDatabase'. Es el punto de entrada principal para interactuar con la base de datos persistente de la aplicación. Configura la base de datos e incluye un 'Callback' que, en modo de desarrollo, pre-pobla datos de prueba al crearse por primera vez.
- **Utilidades ('utils/')**: SingleLiveEvent.kt: Una herramienta especial para enviar "eventos" (como una navegación o un diálogo) desde los ViewModels a las pantallas de forma segura, asegurando que solo se activen una vez.

12.1 Cómo Funciona el Código

La aplicación sigue el patrón de arquitectura MVVM (Model-View-ViewModel). Esto significa que:

- **Las Actividades (Views) se encargan de lo visual**: Muestran los elementos en pantalla (texto, botones, campos de entrada) y detectan cuando el usuario interactúa con ellos (hace clic, introduce texto). Su principal responsabilidad es observar los cambios en los datos que les proporciona el ViewModel y actualizar la UI, y comunicar las interacciones del usuario al ViewModel.
- **Los ViewModels se encargan de la lógica de negocio y el estado de la UI**: Son independientes de la Actividad y sobreviven a cambios de configuración

(como rotaciones de pantalla). Reciben las interacciones del usuario de la Actividad, procesan la lógica (validación, llamadas al repositorio), y actualizan los datos observables ('LiveData').

- **Los Repositorios gestionan el acceso a datos:** Proporcionan una API limpia para los ViewModels para interactuar con los datos, sin que el ViewModel necesite saber si los datos vienen de una base de datos local, una API remota o cualquier otra fuente.

Ejemplo de Flujo (Modificación de Tratamiento con Persistencia):

1. Usuario en 'MainActivity' (View): Toca una tarjeta de tratamiento.
2. MainActivity': Crea un 'Intent' y pasa el objeto 'Tratamiento' ('Parcelable') completo a 'DetalleTratamientoActivity'.
3. DetalleTratamientoActivity' (View): Recibe el 'Intent', extrae el 'Tratamiento' y lo pasa a su 'DetalleTratamientoViewModel' (usando 'SavedStateHandle'). Muestra los detalles del tratamiento.
4. Usuario en 'DetalleTratamientoActivity': Toca el botón "Modificar".
5. 'DetalleTratamientoActivity': Crea un 'Intent' y pasa el objeto 'Tratamiento' ('Parcelable') completo a 'ModificarTratamientoActivity'.
6. 'ModificarTratamientoActivity' (View): Recibe el 'Intent', extrae el 'Tratamiento' y lo pasa a su 'ModificarTratamientoViewModel' (usando 'SavedStateHandle'). La Activity inicializa sus 'TextView's y 'EditText's con los datos del 'Tratamiento'. Cuando el usuario toca un icono de edición, la Activity alterna la visibilidad del 'TextView' y el 'EditText' correspondiente.
7. Usuario en 'ModificarTratamientoActivity': Modifica el número de sesiones y/o la descripción y toca el botón "Guardar".
8. ModificarTratamientoActivity': Obtiene los valores actuales de los 'EditText' y los pasa al 'ModificarTratamientoViewModel': 'viewModel.onGuardarClicked(nuevasSesiones, nuevaDescripcion)'.
9. ModificarTratamientoViewModel' (ViewModel): Actualiza su copia interna del objeto, luego 'Tratamiento' ('tratamientoAModificar.value') con los nuevos valores y despues muestra un diálogo de confirmación al usuario (a través de 'showDialog' que la Actividad observa).
10. Usuario en el Diálogo: Toca "Confirmar".
11. ModificarTratamientoActivity': Llama a 'viewModel.performSaveAction()'.
12. ModificarTratamientoViewModel': Inicia una corrutina ('viewModelScope.launch'), Llama al 'TratamientosRepository': 'repository.updateTratamiento(tratamientoActualizado)'.
13. TratamientosRepository' (Repository): Recibe el tratamiento actualizado y llama a 'tratamientoDao.update(tratamientoActualizado)'.

14. `TratamientoDao` (DAO): Ejecuta la operación SQL `'UPDATE'` en la base de datos Room para ese tratamiento específico (basado en su `'id'`).
15. `TratamientosViewModel`: Recibe la confirmación del guardado, y notifica a la `'ModificarTratamientoActivity'` para mostrar un diálogo de éxito y, si el usuario lo elige, navegar de vuelta a `'MainActivity'`.
16. `MainActivity`: Como su `'TratamientosViewModel'` está observando un `'Flow'` de la base de datos, cuando el tratamiento se actualiza en la base de datos, el `'Flow'` emite la nueva lista, y `'MainActivity'` se actualiza automáticamente con los datos modificados.

12.2 TManejo General de Datos (Persistencia con Room)

La aplicación utiliza la Room Persistence Library, parte de Android Jetpack, para gestionar la base de datos local y asegurar que los datos persistan incluso si la aplicación se cierra o se reinicia. Esto proporciona una fuente única y confiable para todos los datos de tratamientos.

1. **Tratamiento** (`'@Entity'`): La clase `'Tratamiento'` no es solo un simple objeto de datos; está marcada con la anotación `'@Entity'`. Esto le indica a Room que esta clase representa una tabla en la base de datos SQLite de la aplicación. Cada instancia de `'Tratamiento'` corresponde a una fila en la tabla. El campo `'id'` está marcado con `'@PrimaryKey(autoGenerate = true)'`, lo que significa que Room le asignará automáticamente un valor único (autoincremental) a cada nuevo tratamiento insertado. Esto es fundamental para identificar tratamientos de forma exclusiva. La anotación `'@Parcelize'` (parte de `'kotlin-parcelize'`) permite que los objetos `'Tratamiento'` se pasen fácilmente entre diferentes componentes de Android (como Actividades) a través de `'Intent's`, sin necesidad de escribir código manual para la serialización y deserialización.
2. **TratamientoDao** (`'@Dao'`): `'TratamientoDao' ('@Dao')`: Este es el Objeto de Acceso a Datos (DAO)**. Es una interfaz de Kotlin que define cómo interactuar con la tabla `'tratamientos'`. Contiene métodos abstractos anotados con `'@Query'`, `'@Insert'`, `'@Update'`, y `'@Delete'`. Room genera automáticamente la implementación de estos métodos en tiempo de compilación, traduciéndolos a sentencias SQL. Por ejemplo, `'@Query("SELECT * FROM tratamientos ORDER BY fecha ASC")'` fun `getAllTratamientos(): Flow<List<Tratamiento>?>'` permite obtener todos los tratamientos ordenados por fecha. El uso de `'Flow'` (de Kotlin Coroutines) es clave: emite una nueva lista de tratamientos cada vez que hay un cambio en los datos subyacentes de la tabla, lo que permite que la interfaz de usuario se actualice en tiempo real. Los métodos `'suspend'` (como `'insert'`, `'update'`, `'deleteById'`) indican que estas operaciones son de larga duración y deben ejecutarse en una corrutina (en un hilo de fondo) para no bloquear el hilo principal de la UI.
3. **AppDatabase** (`'@Database'`): Esta es la clase principal que representa la base de datos Room en la aplicación. Extiende `'RoomDatabase'` y está anotada con `'@Database'`, especificando qué `'entities'` (clases `'@Entity'` como `'Tratamiento'`) forman parte de esta base de datos y su `'version'`. Proporciona un método abstracto que devuelve una instancia del `'TratamientoDao'`. Utiliza el patrón Singleton (`'companion object'`) para asegurar que solo haya una instancia de la base

de datos en toda la aplicación, evitando problemas de concurrencia y recursos. El `'RoomDatabase.Callback'` anidado (`'DatabaseCallback'`) permite ejecutar código la primera vez que la base de datos se crea (`'onCreate'`) o se abre (`'onOpen'`). En esta aplicación, `'onCreate'` se usa para pre-poblar la base de datos con datos de prueba, lo cual es muy útil durante el desarrollo para tener contenido inicial. `fallbackToDestructiveMigration()` se usa para reconstruir la base de datos si la versión cambia, lo cual facilita la depuración de cambios en el esquema.

4. **TratamientosRepository:** Sirve como intermediario entre los ViewModels y el `'TratamientoDao'` (y potencialmente otras fuentes de datos en una aplicación más compleja, como una API web). Los ViewModels solicitan datos al `'Repository'` (ej., `'repository.getAllTratamientos()'`, `'repository.insertTratamiento()'`), y el `'Repository'` sabe cómo obtener o guardar esos datos usando el `'TratamientoDao'`. Esto mantiene la lógica de acceso a la base de datos encapsulada y separada de la lógica de negocio de los ViewModels.


En resumen, la aplicación utiliza Room para manejar la persistencia de datos de manera robusta y asíncrona, garantizando que la UI siempre muestre la información más reciente y que los cambios del usuario se guarden de forma fiable.

12.3 Tecnologías Clave

- **Kotlin:** El lenguaje de programación utilizado.
- **Android Jetpack:** Un conjunto de librerías de Google que facilitan el desarrollo, incluyendo:
 - **'LiveData' y 'ViewModel':** Fundamentales para implementar el patrón MVVM y gestionar el ciclo de vida de la UI.
 - **'SavedStateHandle':** Permite a los ViewModels persistir y restaurar el estado de la UI después de cambios de configuración (como rotaciones) o de la terminación del proceso por parte del sistema.
 - **'kotlinx.coroutines.flow':** Utilizado con Room para observar cambios en la base de datos en tiempo real de forma asíncrona.
 - **Material Design:** Para el diseño visual moderno de la interfaz de usuario, incluyendo `'TextInputLayout'` para campos de entrada y `'FloatingActionButton'` para acciones principales.
- **Gradle:** La herramienta de automatización de compilación utilizada para construir la aplicación y gestionar las dependencias.

13 Testeo de la segunda parte

Test de usuario paciente



Usuario 1

Nombre y edad: Luz Marina, 59 años.

Sexo: Femenino.

Ocupación: Ing. Catastral.

Experiencia: Nula.

Escenario: Presencial.

Tiempo de prueba: 10 min

Enunciado y objetivo

Debe interactuar con el prototipo de la app FisiCare en el módulo de tratamiento y realizar las acciones que en su rol puede realizar que son consultar los tratamientos, el historial de modificaciones y su progreso. Al igual dar su opinión del diseño y demás propio de la interfaz.

¿Qué te ha parecido?

Interesante y muy amigable.

¿Qué cambiarías o añadirías?

Cambio de colores para algunos botones.

Dudas y comentarios

Ninguna.

Tarea

El usuario en rol de paciente debe poder consultar sus tratamientos, ver el historial de sus modificaciones del tratamientos y su progreso en los mismos.

¿Qué funciona?

El acceso a la consulta de los tratamientos en curso es fácil y permite hacerle un seguimiento secuencial en el tiempo de los tratamientos pendientes, así como el avance que se ha tenido en cuanto a las sesiones programadas.

¿Qué preguntas nos hacen?

En el panel principal se accede a la opción de tratamientos y ya en esta opción, se accede al tratamiento que se quiere consultar y hay una pregunta para confirmar si quiere seleccionar o no este tratamiento. También me muestra si deseo o no guardar los cambios en una modificación.

¿Qué cosas te han gustado?

Desde el ingreso la aplicación es muy amigable, fácil para el entendimiento y permite consultar y acceder a las diferentes opciones disponibles de manera sencilla y precisa. También se mueve de manera cómoda y no se siente lenta.

¿Qué se puede mejorar?

Que tenga mensajes emergentes de ayuda con un listado de preguntas frecuentes, en toda la app.

¿Qué ideas nos proponen?

Los colores en algunas zonas tienden a perderse un poco en especial en algunos botones.

Figure 15: Testeo 2

14 Video funcionalidades del proyecto

Aca se podrá ver las funcionalidades del proyecto. <https://youtube.com/shorts/VYrqE90etyU>.

15 Referencias

References

- [1] Natalia-Rojas-Suarez, "Repositorio de Desarrollo Móvil," GitHub. [En línea]. Disponible en: <https://github.com/Natalia-Rojas-Suarez/Desarrollo-Movil>.
- [2] "Desarrollo Móvil App," Trello, 2024. [En línea]. Disponible en: <https://trello.com/invite/desarrollo-movil-app>. [Accedido: 10-feb-2025].
- [3] "Tablero de Miro: Desarrollo Móvil," Miro, 2024. [En línea]. Disponible en: <https://miro.com/welcomeonboard/...> [Accedido: 10-feb-2025].

- [4] “Design Thinking España,” 2024. [En línea]. Disponible en: <https://designthinking.es/>. [Accedido: 10-feb-2025].