OralData- Sistema de gestión de pacientes para prácticas clínicas

Juan Camilo Ossa Guzmán

Juan David Nanclares Pulgarín

Trabajo de grado presentado como requisito parcial para optar al título de:
Especialista en Ingeniería de Software

Asesor(es)

Alicia Osorio Builes

Edwin Andrés Cubillos

Julián Alberto Uribe Gómez

Instituto Tecnológico Metropolitano - ITM

Facultad de Ingenierías

Departamento de Antioquia

Medellín, Colombia

2024

1. RESUMEN

Este proyecto, titulado “OralData - Sistema de Gestión de Pacientes para Prácticas Clínicas”, se centra en el desarrollo de una aplicación móvil diseñada para mejorar la gestión de pacientes en las consultas clínicas odontológicas para las prácticas de los estudiantes de odontología. A través de una metodología ágil, concretamente Scrum, se ha realizado un estudio de mercado para identificar Aplicaciones existentes con funcionalidades similares, como My Dental Clinic, Smile y Dentalink. A partir de las fortalezas y debilidades de estas aplicaciones, se ha propuesto un sistema que ofrece autogestión bilateral de programación y autorregistro de usuarios en la plataforma. La aplicación se desarrollará utilizando tecnologías como Flutter y Firebase, que permitirán a los estudiantes registrar y autenticar usuarios, gestionar perfiles, programar y cancelar citas y comunicarse con pacientes a través de chat en tiempo real. Además, proporcionará notificaciones para eventos importantes y garantizará la seguridad de los datos personales de los usuarios. El principal objetivo del desarrollo de OralData es mejorar la eficiencia y experiencia en las prácticas clínicas de los estudiantes de odontología, facilitando la gestión de pacientes y optimizando las tareas operativas. Se proyecta que el sistema ayude a superar los desafíos actuales en la formación práctica, mejorando tanto la calidad de la educación como la atención al paciente.

Palabras Clave: Gestión de pacientes en Oral Data, prácticas clínicas odontológicas, aplicación móvil Oral Data, metodología ágil Scrum, Flutter, Firebase.

2. ACRÓNIMOS

GIT: Grupo de investigación en integración de soluciones con tecnología de información y comunicación

MIRP: Máquinas Inteligentes y Reconocimiento de Patrones

NWK: Capa de la red de pila ZigBee

MDS: Escalamiento multidimensional

TABLA DE CONTENIDO

Tabla de Ilustraciones

Tablas

3. INTRODUCCIÓN

El proyecto "OralData - Sistema de gestión de pacientes para prácticas clínicas" surge de la necesidad de optimizar la gestión de pacientes en las facultades de odontología. En la actualidad, los estudiantes están experimentando dificultades al buscar pacientes y administrar sus prácticas clínicas de manera efectiva, lo cual está teniendo un impacto negativo en la calidad de su formación. Durante la investigación de mercado se descubrieron varias aplicaciones como My Dental Clinic, Smile y Dentalink. Estas Másterciones ofrecen características similares, pero también presentan ciertas limitaciones en diferentes aspectos. OralData es una solución innovadora que se basa en estas observaciones. Permite a los usuarios autogestionar tanto el agendamiento como el autorregistro, mejorando la experiencia y eficiencia de estudiantes y pacientes por igual.

El proyecto tiene como objetivo proponer un sistema de gestión que esté enfocado en aplicaciones móviles para automatizar las tareas operativas relacionadas con los pacientes, con el fin de mejorar la eficiencia y la experiencia durante el proceso de prácticas clínicas. Para alcanzar este objetivo general, se han establecido los siguientes objetivos específicos: Entender las necesidades técnicas requeridas para el desarrollo de una aplicación web de gestión de pacientes en odontología; determinar las funcionalidades esenciales del sistema de gestión; crear un diseño orientado a la web que organice eficientemente los procesos clínicos; y finalmente, elaborar un sistema de gestión acorde con dichas necesidades.

Este trabajo se compone de varios conceptos fundamentales. En primer lugar, se ofrece una descripción general del proyecto que abarca su pertinencia y justificación, así como los objetivos a alcanzar con el mismo. En este escrito se detalla el tema problemático que está siendo tratado, así como también se expone la importancia de la solución planteada. Después se explica detalladamente la metodología ágil aplicada en el proceso de desarrollo del proyecto, centrándose principalmente en Scrum. Además, se realiza una descripción exhaustiva de las tecnologías utilizadas en el proyecto, como Flutter y Firebase; destacando su contribución fundamental durante todo el proceso de creación de la aplicación. En primer lugar, se examina el déficit existente en la atención de pacientes en las escuelas de odontología en Colombia. Se discuten los efectos negativos que estos problemas tienen sobre la educación de los estudiantes y se argumenta a favor de implementar una solución tecnológica para abordar esta situación. A continuación, se detalla el diseño y desarrollo del sistema OralData, abarcando el modelo de datos utilizado, la arquitectura del sistema y los patrones arquitecturales aplicados. Además, se presentan las funcionalidades implementadas y cómo han sido diseñadas para satisfacer las necesidades identificadas. Por último, se evalúan los resultados obtenidos durante el proceso de desarrollo del sistema, considerando las pruebas y validaciones realizadas. Se examinan las lecciones aprendidas y se presentan recomendaciones para mejorar la plataforma OralData. Además, se plantea una estrategia sobre los próximos pasos a seguir para continuar perfeccionando esta aplicación.

4. ANTECEDNETES Y MARCO TEÓRICO

4.1 Antecedentes

El presente proyecto propone la creación de una plataforma en la cual los estudiantes de la facultad de odontología tanto de pregrado como posgrado puedan administrar a través de su usuario y su contraseña, los espacios de práctica y gestionar a las personas para ser atendidas durante las clínicas, ya que esto debe cumplirse como requisito para aprobar las competencias que cada clínica practica requiere.

Después de una investigación de mercado se encuentran algunas aplicaciones con funcionalidades similares al software que se propone desarrollar. En primer lugar, se encontró, la aplicación de origen escandinava llamada *My Dental Clinic*, el cual es una herramienta útil para estudiantes, en el cual pueden obtener y gestionar pacientes. Dicha aplicación, permite la gestión de la información de pacientes, teniendo acceso en todo momento a dichos datos, adicionalmente, permite rastrear los gastos de la clínica, crear, imprimir, enviar tratamientos a los pacientes, tener colaboración entre profesores, de manera que varios estudiantes y profesores puedan atender al paciente al mismo tiempo. Además, permite tener respaldo de los datos de pacientes. (Dental Clinic App, s.f.)

Dentro de dicha revisión, se encontró además, un software mexicano llamado *Smile* que está especializado en clínicas universitarias, el cual presenta similitudes a las del proyecto propuesto, entre estás se encuentran el agendamiento y control de citas de los pacientes por cada estudiante, registro de expediente clínico con notas de evolución y plan de tratamiento, control de actividades de estudiantes por parte de profesores e incluye otras funcionalidades como lo son controles de órdenes de laboratorio, imágenes médicas al expediente clínico, control de inventarios e ingresos económicos de la clínica; mostrando altos beneficios tanto en el ámbito clínico-académico como a nivel administrativo. (Software Dentalink, s.f.) De esta misma manera, y compartiendo gran cantidad de funcionalidades que presentan las aplicaciones anteriores, se encontró un software chileno llamado Dentalink, que además tienen un módulo integrado de toda la parte de contabilidad, ingresos y trazabilidad de los pacientes, cuentan también con un dashboard que permite por medio de graficas tener informes precisos y en tiempo real de las citas asignadas, pacientes agendados, pacientes en consulta y pagos online. (Software Dentalink, s.f.)

En resumen, se encontraron 3 aplicaciones con gran robustez, las cuales presentan beneficios similares a los esperados al desarrollo propuesto. Estos son: ordenamiento de la agenda para evitar dificultades con la programación de citas, diligenciamiento de la historia clínica (almacenando de manera integral, eficiente y segura toda la información del paciente), obtención de reportes de gestión para reconocer puntos de mejora y facilitar la toma de decisiones.
Una vez finalizada esta investigación, en la cual de describen las soluciones actuales, se concluye que la propuesta a desarrollar posee como valor agregado, la autogestión bilateral de agendamiento (tanto pacientes como estudiantes podrán gestionar su atención) y todo, a partir del autorregistro de usuarios en la plataforma (tanto pacientes como estudiantes).

4.2 Marco Teórico

La gestión de pacientes es un componente esencial de la educación odontológica. Los estudiantes necesitan tener la oportunidad de practicar sus habilidades en pacientes reales para desarrollar las competencias necesarias para ejercer su profesión.

Aprendizaje práctico: Es un enfoque de aprendizaje centrado en la experiencia de aprendizaje activo y en la aplicación de conocimientos para situaciones reales. Dicho enfoque, se basa en la teoría de aprendizaje experiencial de John Dewey, quien afirma que el conocimiento se construye a través de la experiencia y son los estudiantes quienes son los protagonistas de su propio aprendizaje. (Ausubel, Novak y Hanesian, 1978)
Este aprendizaje, puede incluir actividades como resolución de problemas, exploración y experimentación, por tanto, los estudiantes construyen su propio conocimiento a medida que avanzan (Ausubel, Novak y Hanesian, 1978), incluyendo estrategias como el aprendizaje cooperativo y basado en el servicio, permitiendo abordar problemas en situaciones reales con el objetivo de beneficiar a la comunidad.
Este tipo de aprendizaje es importante para la formación de los odontólogos, ya que les permite desarrollar las competencias necesarias para ejercer su profesión. En las áreas de la salud, dicho aprendizaje, se da a través de prácticas clínicas.

Practicas clínicas odontológicas: Las prácticas clínicas en el área odontológica son esenciales para la formación de estudiantes, en ellas los estudiantes pueden aplicar los conocimientos adquiridos durante las clases teóricas en un entorno real (Universidad de Santiago de Compostela, 2023). Durante dichas prácticas, se realiza evaluación y tratamiento a pacientes, las cuales son realizadas bajo la supervisión de profesores con experiencia tanto en docencia como en atención clínicas.

Las prácticas clínicas, incluyen adquisición de competencias de comunicación asistencial, gestión clínica, razonamiento y juicio crítico, por tanto, no solo aprenden a realizar procedimientos dentales, sino que también a comunicarse efectivamente con pacientes, gestionar citas, recursos y a tomar decisiones basados en evidencia (Universidad de Santiago de Compostela, 2023).

Para desarrollar dichas prácticas, es necesario, contar con un sistema que permita realizar tareas administrativas, las cuales permitan realizar la gestión de pacientes.

Sistemas de gestión de pacientes: Es una parte fundamental de la atención médica, en la cual se realizan procesos administrativos, en los cuales se coordina de punta a punta la atención al paciente. Durante la gestión, se recopila información del paciente, se realizan actividades administrativas, se gestionan las horas de atención, los profesionales que prestarán la atención con los profesionales de la red asistencial y se coordina los cuidados (F. Aguayo & R. Mella, 2015)

Dicho proceso, implica, planificación, organización, motivación en la atención y control de provisión de cuidados a prestarse de manera oportuna, segura e integral, sustentado en políticas de cada centro de atención. (F. Aguayo& R. Mella, 2015).

Los sistemas de gestión de pacientes han evolucionado significativamente en los últimos años, por tanto, su evolución ha estado marcada por los siguientes hitos:

- En 1960 comienzan a usarse en la década de 1960, estos sistemas eran simples, dicha información se diligenciaba de manera manual en papel y era almacenada en carpetas físicas
- En 1970 se comienza a usar tecnología informática, con sistemas que permitían a médicos y profesionales en salud, acceder a información de los pacientes de manera rápida y fácil.
- En 1980: se introducen los primeros sistemas de gestión de pacientes con sistemas integrados, los cuales combinaban información de los pacientes como historial médico, resultado de exámenes médicos y prescripciones, consultando dicha información desde diferentes fuentes.
- En 1990: Se comienza a desarrollar sistemas para gestión de pacientes en la web, permitiendo a profesionales de la salud consultar información de los pacientes desde cualquier lugar
- En la década del 2000, se comienzan a realizar sistemas para gestión de pacientes para aplicaciones móviles
- En la década del 2010, se introducen los sistemas para gestión en la nube, los cuales almacenan información de pacientes desde cualquier lugar con acceso a internet
- En la década del 2020, se introduce el aprendizaje automático y la inteligencia artificial a los sistemas de gestión de pacientes, permitiendo tomar decisiones sobre el tratamiento de los pacientes

Al ser la odontología una rama de la salud, la evolución de los sistemas de gestión de pacientes se ha visto influenciada por los sistemas de gestión de pacientes, en los cuales se hacen ajustes de acuerdo con las necesidades específicas del sector, por tanto, ha sido un reflejo de la evolución de la tecnología en los sistemas de gestión en salud.

La gestión de pacientes de manera efectiva puede ayudar a recopilar información la cual permita comprender mejor las necesidades de los usuarios de manera que se pueda prestar una atención de calidad.

Gestión de la calidad: Es un proceso llevado a cabo con el fin de mejorar la eficiencia y efectividad de las operaciones y servicios de una organización. Se enfoca en la mejora continua de los procesos, productos y servicios para satisfacer o superar las expectativas y necesidades de los clientes.
Este proceso implica la recopilación y análisis de la información de los clientes para entender sus necesidades y expectativas, y luego utilizar esta información para desarrollar productos y servicios que satisfagan o superen estas expectativas (Definición ABC, gestión de la calidad)

Esta, además, implica la implementación de sistemas y procedimientos para controlar y mejorar la calidad de los productos y servicios. Esto puede incluir la inspección de los productos o servicios, la realización de pruebas y auditorías, y la implementación de acciones correctivas cuando se detectan problemas de calidad.

El desarrollo de un sistema que permita desarrollar un sistema de gestión en salud requiere un lenguaje de programación potente y flexible y un marco de desarrollo que permita amplia gama de bibliotecas y herramientas, como las presentadas a continuación:

Flutter: Flutter es un avanzado kit de desarrollo de software creado por Google que permite a los desarrolladores construir aplicaciones de alta calidad para dispositivos móviles, web y de escritorio a partir de una única base de código. Utilizando el lenguaje de programación Dart, Flutter ofrece un enfoque innovador y eficiente para el desarrollo de aplicaciones, caracterizado por su capacidad de compilar código nativo, lo que garantiza un rendimiento óptimo en plataformas Android e iOS.

Una de las características más destacadas de Flutter es su uso de widgets, componentes reutilizables que permiten crear interfaces de usuario personalizables y atractivas. Estos widgets pueden combinarse y estilizarse de múltiples formas, proporcionando una gran flexibilidad en el diseño de interfaces de usuario complejas y adaptables a diversas plataformas.

El desarrollo con Flutter se beneficia enormemente de la función "hot reload", que permite a los desarrolladores ver los cambios en el código en tiempo real sin necesidad de reiniciar la aplicación. Esto acelera considerablemente el proceso de desarrollo y facilita la depuración.

Además, Flutter es reconocido por su capacidad multiplataforma, permitiendo que una sola base de código pueda generar aplicaciones para iOS, Android, la web y sistemas de escritorio como Windows, macOS y Linux. Esta versatilidad, junto con una comunidad activa y un vasto ecosistema de paquetes y plugins, ha convertido a Flutter en una herramienta muy popular entre los desarrolladores que buscan crear aplicaciones modernas, eficientes y con una excelente experiencia de usuario.

FireBase: Es una plataforma en la nube para el **desarrollo de aplicaciones web y móvil**. Está disponible para distintas plataformas (iOS, Android y web), con lo que es más rápido trabajar en el desarrollo.

5. PLANTEAMIENTO DEL PROBLEMA

La formación de los odontólogos requiere de un equilibrio entre la teoría y la práctica. Sin embargo, en los últimos años, las clínicas prácticas en las facultades de odontología de Colombia han enfrentado un déficit importante. Esto se debe a que los estudiantes, en gran medida, tienen dificultades para encontrar pacientes que les permitan desarrollar sus competencias prácticas y habilidades manuales, adquiridas inicialmente en la teoría (Rojas & Galvis, 2023).

Este déficit puede afectar la calidad de la formación de los odontólogos y la atención que brindan a los pacientes. Los estudiantes que no tienen la oportunidad de practicar sus habilidades en pacientes reales pueden tener dificultades para adquirir la confianza y la experiencia necesarias para ejercer su profesión. Además, esto les genera un estancamiento académico ya que al no demostrar sus competencias estos reprueban y por ultimo los pacientes que reciben atención odontológica de estudiantes con poca experiencia pueden estar expuestos a un mayor riesgo de complicaciones.

Todo esto, ha hecho que el proceso de gestión de pacientes en las facultades de odontología sea un proceso complejo y desafiante, generando dificultades para los estudiantes y profesores. Los profesionales de la salud deben dedicar mucho tiempo a tareas operativas, como la búsqueda de pacientes, el diligenciamiento de la historia clínica y la asignación de citas (F. Arevalo, 2015). La gestión de pacientes de manera tradicional, sin un sistema que permita automatizar estas tareas, es ineficiente y puede generar otras dificultades, como el aumento de los errores, la disminución de la productividad y la mala experiencia del paciente (Journal of Dental Education,2021).

Actualmente no se cuenta con un sistema que sirva de puente directo entre los estudiantes y pacientes que permita la gestión, atención, fluidez y trazabilidad sin que exista un rol intermedio en las instituciones que en su mayoría de veces no cumple con estos requerimientos ya que usa una hoja de cálculo y no contempla una valoración acertada o información detallada que permita direccionar bien a los estudiantes para asignar sus clínicas, dejando un panorama de incertidumbre en los estudiantes para cumplir con los requisitos académicos necesarios para aprobar y continuar con su proceso de aprendizaje, estos actualmente tienen truncados estos procesos y les está generando retrasos por no aprobar los niveles o semestres a tiempo. Por otro lado, los pacientes no tienen la posibilidad de identificar la existencia de jornadas o clínicas de atención en las cuales ellos pueden acceder a servicios de forma gratuita o a un bajo costo.

Dichas dificultades permiten evidenciar la necesidad de un sistema el cual permita realizar gestión, trazabilidad y obtención de pacientes por parte de los estudiantes para que puedan ejercer sus clínicas de atención y que los pacientes con pocos recursos tengan la posibilidad de acceder a una atención respaldada por un docente con amplia experiencia de manera gratuita o más económica.

Por lo anterior que, para abordar este problema, es necesario implementar un sistema de gestión de pacientes que permita automatizar las tareas operativas ayudándole a los estudiantes a elegir, contactar y agendar pacientes a través de un sistema que permita mejorar la eficiencia, la seguridad y la experiencia entre paciente y el estudiante.

¿Es posible diseñar un sistema de gestión de pacientes para estudiantes de odontología que mejore la eficiencia y la experiencia en el proceso de las practicas clínicas?

5.1 Diagrama causa efecto

https://drive.google.com/file/d/1S1QX03DzBY_dLFb9iUUoNMzbuxiyCwJj/view?usp=drive_link

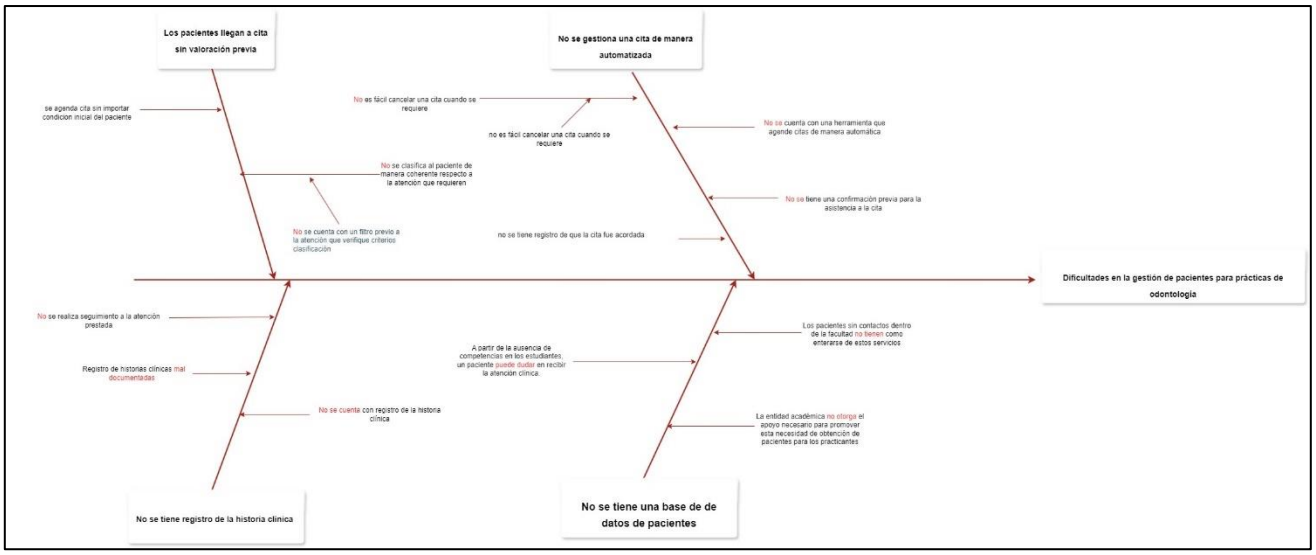


Ilustración 1. Diagrama causa efecto

5.2 OBJETIVOS

General

Proponer un sistema de gestión de pacientes orientado a aplicativos móviles para que los estudiantes de odontología que puedan automatizar las tareas operativas, mejorando la eficiencia y la experiencia en el proceso de prácticas clínica.

Específicos

Perceptivo: Comprender las necesidades técnicas que se tendrían al momento de diseñar una aplicación para gestión de pacientes para las facultades de odontología
Aprehensivo: Seleccionar de las necesidades técnicas anteriormente establecidas las funcionalidades que debe llevar el sistema de gestión de pacientes en su diseño
Comprensivo: Diseñar un sistema de gestión orientado a la web capaz de estructurar el proceso de prácticas clínicas
Integrativo: Desarrollar un sistema de gestión de pacientes que cumpla con las necesidades técnicas anteriormente identificadas.

6. METODOLOGÍA

El presente proyecto busca crear un aplicativo móvil para estudiantes de facultades de odontología, el cual facilite el proceso de gestión de pacientes, siendo desarrollado en .dart el cual es un lenguaje de programación flexible, con el uso del flutter y firebase debido a que este facilita la interacción con bases de datos relacionales y no relacionales las cuales se usarán en el proceso para almacenamiento de información de usuarios e historias clínicas y proporciona características de seguridad muy importante para proteger los datos de los pacientes. Teniendo en cuenta esto, se hace uso de .dart como marco para crear el frontend de la aplicación debido a que facilita la creación de interfaces de usuario, la creación de código modular y su alta compatibilidad con diferentes dispositivos móviles. Las imágenes de perfil de los usuarios y las solicitadas para la clasificación de pacientes se almacenarán en firebase. El desarrollo tendrá en cuenta el patrón MVVM, el cual mejora la seguridad de la aplicación y permite la mantenibilidad y separa la aplicación a través de capas.

Esto, junto con la arquitectura responsiva, permitirá que la aplicación se vea bien en diferentes dispositivos y tamaños de pantalla, sin que se tenga que generar código adicional para esto, adicionalmente cargará gradualmente la aplicación, mejorando la experiencia.

6.1 Planificación, Gestión y Administración del Proyecto

6.1.1. Requisitos del Proyecto

Requisitos Funcionales:

1. **Registro y Autenticación de Usuarios:**
    - El sistema debe permitir el registro de estudiantes y pacientes.
    - El sistema debe permitir la autenticación de usuarios registrados.
2. **Gestión de Perfiles:**
    - Los usuarios deben poder actualizar su información personal.
    - Los estudiantes deben poder subir una imagen de perfil.
    - Los pacientes deben poder subir una imagen de valoración y una imagen de perfil.
3. **Gestión de Citas:**
    - Los estudiantes deben poder agendar citas con pacientes.

- Los estudiantes deben poder cancelar citas.
- Los pacientes deben recibir notificaciones locales cuando se agenden o cancelen citas.
- Los pacientes deben poder ver sus citas programadas.

4. **Chat en Tiempo Real:**
    - Los estudiantes y pacientes deben poder comunicarse a través de un chat en tiempo real.
    - Los usuarios deben recibir notificaciones locales cuando se envíe un nuevo mensaje en el chat.
5. **Notificaciones:**
    - Implementar notificaciones locales para eventos importantes como nuevas citas y nuevos mensajes de chat.

Requisitos No Funcionales

1. **Seguridad:**
    - La aplicación debe garantizar la seguridad de los datos personales de los usuarios.
    - Los datos deben ser encriptados durante la transmisión.
2. **Usabilidad:**
    - La interfaz de usuario debe ser intuitiva y fácil de usar.
    - La aplicación debe ser accesible en dispositivos móviles.
3. **Rendimiento:**
    - La aplicación debe ser capaz de manejar múltiples usuarios simultáneamente.
    - La aplicación debe ser optimizada para funcionar eficientemente en dispositivos de gama baja.

6.1.2 Plan de Metodología Ágil

Metodología Ágil: Scrum

Roles

- **Product Owner:** Responsable de definir y priorizar el backlog del producto.
- **Scrum Master:** Facilita el proceso Scrum y se asegura de que se sigan las prácticas ágiles.
- **Development Team:** Equipo de desarrollo que implementa las funcionalidades del producto.

Ceremonias

1. **Sprint Planning:** Reunión para planificar las tareas del sprint.
2. **Daily Stand-up:** Reuniones diarias para revisar el progreso y resolver impedimentos.
3. **Sprint Review:** Reunión al final del sprint para revisar y demostrar el trabajo realizado.
4. **Sprint Retrospective:** Reunión para reflexionar sobre el sprint y planificar mejoras para el siguiente sprint.

Artefactos

1. **Product Backlog:** Lista priorizada de todas las funcionalidades y requisitos del proyecto.
2. **Sprint Backlog:** Lista de tareas seleccionadas para el sprint actual.
3. **Incremento:** Conjunto de funcionalidades completadas durante el sprint.

Sprints:

- Cada sprint tiene una duración de 2 semanas.
- Al final de cada sprint, se debe entregar un incremento funcional del producto.

Duración Total del Proyecto: 8 meses

Sprint Duration: 2 semanas

Total Sprints: 16 sprints

Roles en Scrum

- **Product Owner:** Juan Camilo Ossa

- **Scrum Master:** Juan David Nanclares
- **Development Team:** Compuesto por los estudiantes de odontología y desarrolladores del proyecto.

Ceremonias de Scrum

- **Sprint Planning:** Reunión al inicio de cada sprint para planificar las tareas.
- **Daily Stand-up:** Reuniones diarias de 15 minutos para revisar el progreso.
- **Sprint Review:** Reunión al final de cada sprint para revisar y demostrar el trabajo realizado.
- **Sprint Retrospective:** Reunión al final de cada sprint para reflexionar sobre el sprint y planificar mejoras.

Artefactos de Scrum

- **Product Backlog:** Lista priorizada de todas las funcionalidades y requisitos del proyecto.
- **Sprint Backlog:** Lista de tareas seleccionadas para el sprint actual.
- **Incremento:** Conjunto de funcionalidades completadas durante el sprint.

6.1.3 Gestión Ágil

La gestión ágil se centra en el seguimiento del progreso del proyecto, la identificación y resolución de problemas y la adaptación a los cambios. Se basa en los principios de:

- **Transparencia:** Todo el equipo debe tener acceso a la información sobre el estado del proyecto.
- **Inspección:** El trabajo del equipo debe inspeccionarse regularmente para identificar problemas.
- **Adaptación:** El equipo debe adaptarse a los cambios en las necesidades del proyecto.
- **Enfoque en el valor:** El equipo debe centrarse en entregar valor al negocio.
- **Mejora continua:** El equipo debe buscar constantemente formas de mejorar su proceso de trabajo.

¿Quién la lidera?

La gestión ágil para el proyecto es liderada por un miembro del equipo que asume el rol de **Scrum Master**, quien tiene la responsabilidad de asegurar que el equipo siga la metodología Scrum y de remover los impedimentos que puedan afectar el progreso del proyecto.

¿Cuáles son las responsabilidades?

Las responsabilidades de la gestión ágil incluyen:

- **Monitorear el progreso del proyecto.**
- **Identificar y resolver problemas.**
- **Gestionar los riesgos.**
- **Comunicarse con las partes interesadas.**
- **Facilitar la mejora continua.**

Componentes de la gestión ágil aplicados en el proyecto:

- **Tablero Kanban:** Una herramienta visual que se utiliza para mostrar el estado del trabajo en curso. En este se pueden visualizar las tareas por hacer, que hay en proceso, que está en pruebas y que se ha
- **Gráficos de burndown:** Gráficos que muestran la cantidad de trabajo restante en un sprint. Con esta herramienta se podrá monitorear el progreso del trabajo restante de los sprints
- **Herramientas de comunicación:** Herramientas que se utilizan para facilitar la comunicación entre los miembros del equipo. Para estas se usarán Herramientas como Teams, Google meet, whatsapp y Zoom.

Actividades de la gestión ágil seguidas en el proyecto:

- **Reuniones diarias de scrum:** Ajustada al proyecto, se realizan 2 reuniones a la semana estilo daily, en las cuales se manejan los puntos a tener en cuenta de una daily de seguimiento a las tareas para ver que si se estén cumpliendo los objetivos del sprint y en caso de haber impedimentos realizar acciones para eliminarlos en el menor tiempo posible.
- **Reunión de revisión de sprint:** Se programaron reuniones al final de cada sprint en la que el equipo muestra el trabajo completado y recibe feedback del Product Owner.
- **Reunión de retrospectiva de sprint:** Se programó para el final de cada sprint, una reunión donde se reflexiona sobre el sprint que acaba de pasar para tener en cuenta aspectos a mejorar para los siguientes sprints.
- **Monitoreo del progreso del proyecto:** Seguimiento del avance del proyecto en relación con el plan original.

- **Gestión de riesgos:** Identificación de riesgos potenciales que puedan afectar el proyecto y desarrollo de planes de contingencia.
- **Comunicación con las partes interesadas:** Mantener a las partes interesadas informadas sobre el progreso del proyecto.
- **Facilitar la mejora continua:** Identificar oportunidades para mejorar el proceso de trabajo e implementar cambios.

Tarea	Rol responsable
Monitorear el progreso del proyecto	Scrum Master
Identificar y resolver problemas	Equipo de desarrollo
Gestionar los riesgos	Scrum Master
Comunicarse con las partes interesadas	Product Owner
Facilitar la mejora continua	Scrum Master

Tabla 1. Interpretación de la gestión agil

De la herramienta AzureDevops para la parte de gestión ágil en el proyecto, se hace uso de las siguientes funcionalidades:

- Uso de tableros Kanban para visualizar el flujo de trabajo (actividades en estado por hacer, en progreso, listo para revisión, completado)
- Visualización de informes para el seguimiento en tiempo real, se crearon allí gráficos Burndown
- Integración con repositorio del proyecto.

6.1.4 Administración Ágil

¿Cómo se realizará la administración ágil del proyecto?

La administración ágil del proyecto OralData se basa en los principios y prácticas de la metodología Scrum, que promueve un enfoque iterativo e incremental para el desarrollo de software.

¿Quién liderará la administración ágil del proyecto?

El **Scrum Master** será el responsable de liderar la administración ágil del proyecto. El Scrum Master es un facilitador que guía al equipo en la adopción de los principios y prácticas Scrum, y ayuda a eliminar los impedimentos que puedan surgir durante el desarrollo del proyecto.

¿Cuáles son las responsabilidades del Scrum Master para administración ágil?

Las responsabilidades del Scrum Master incluyen:

- **Facilitar las reuniones Scrum:** Planificación del sprint, revisión del sprint, retrospectiva del sprint y reunión diaria.
- **Proteger el tiempo del equipo:** Asegurar que el equipo tenga el tiempo y el espacio necesarios para trabajar de manera eficiente.
- **Servir como coach del equipo:** Ayudar al equipo a mejorar sus habilidades y prácticas ágiles.
- **Promover la transparencia:** Asegurar que todos los miembros del equipo tengan acceso a la información necesaria.

¿Cuáles son los componentes de la administración ágil del proyecto?

Los componentes de la administración ágil del proyecto incluyen:

- **Tablero Kanban:** Una herramienta visual para gestionar el flujo de trabajo y visualizar el progreso del equipo.
- **Gráficos Burndown:** Herramientas para monitorear el progreso del trabajo restante en un sprint.

¿Cuáles son las actividades de la administración ágil del proyecto?

Las actividades de la administración ágil del proyecto incluyen:

- **Revisión del sprint:** El equipo demuestra el trabajo completado en el sprint al Product Owner y recibe feedback.
- **Retrospectiva del sprint:** El equipo reflexiona sobre el sprint pasado y busca oportunidades de mejora para el próximo sprint.

¿Cuáles son las tareas a administrar con sus roles responsables?

Tarea	Rol Responsable
Gestión del backlog del producto	Product Owner
Planificación del sprint	Scrum Master, Equipo de desarrollo
Revisión del sprint	Product Owner, Scrum Master, Equipo de desarrollo
Retrospectiva del sprint	Scrum Master, Equipo de desarrollo
Reunión diaria	Scrum Master, Equipo de desarrollo
Monitoreo del progreso	Scrum Master, Equipo de desarrollo

Eliminación de impedimentos	Scrum Master, Equipo de desarrollo
Comunicación con las partes interesadas	Product Owner, Scrum Master

Tabla 2. Tareas por rol de responsable

Para esta parte, se hace uso de encuestas donde se recopila el feedback de los usuarios teniendo en cuenta encuestas de satisfacción y herramientas donde se documentan las lecciones aprendidas.

6.1.5 Plan de Gestión de Riesgos

La gestión de riesgos es un componente importante en la planificación y ejecución de proyectos, debido a que permite responder a posibles desafíos que pueden comprometer los objetivos del proyecto, por tanto, hace posible la prevención de problemas antes de que estos ocurran, de manera que el equipo esté preparado para enfrentar situaciones imprevistas.

Un plan para gestionar los riesgos resulta ser muy útil para este proyecto donde se implementa la metodología ágil, debido a que actúa como una guía estratégica en el flujo del proyecto, facilitando la colaboración entre los miembros del equipo, promueva la transparencia y la creación de un producto de alta calidad. Por tanto, a continuación, se detalla el Plan de Gestión de riesgos para el proyecto OralData, donde en cada etapa del plan, se asignan responsabilidades para la planificación, gestión y administración de riesgos, buscando que el proyecto avance de forma segura.

Proceso de gestión de riesgos

1. **Identificación de riesgos**: Para esta etapa en la gestión de riesgos, se realizaron reuniones de lluvia de ideas, donde se identificaron posibles riesgos del proyecto, se documentaron los hallazgos.

   Planificación: el responsable de coordinar las sesiones fue el Scrum Master

   Gestión: El scrum Master documenta los riesgos hablados durante la sesión

2. **Evaluación de los riesgos**: Una vez los riesgos fueron identificados, en esta etapa se analizó para cada riesgo su causa raíz, consecuencia, probabilidad de ocurrencia e impacto (como se muestra en el punto XX), permitiendo priorizar los riesgos.

   Planificación: El Scrum master documentó la priorización de los riesgos basándose en su relevancia y la capacidad del equipo para manejarlos.

   Gestión: El Product Owner evaluó los riesgos desde la perspectiva del negocio.

Administración: Los responsables de administrar la matriz de riesgos construida fueron los miembros del equipo de desarrollo.

3. **Priorización de riesgos**: De acuerdo con los riesgos, en esta etapa se validó cuales riesgos atacar primero, de acuerdo con su impacto y probabilidad.
   Planificación: el Product Owner priorizó los riesgos, basándose en los elementos presentes en el backlog relacionados con ellos, buscando mantener el foco en las áreas más importantes.
   Gestión: El Scrum Master, se encargó de realizar la priorización, tomando decisiones sobre los riesgos abordados.
   Administración: El equipo de desarrolló tomó liderazgo de la administración de los riesgos, teniendo en cuenta que estuvieran presentes en los sprints.

4. **Tratamiento de riesgos**: En esta etapa, se definieron acciones para mitigar los riesgos, incluyendo estrategias de verificación y validación.
   Planificación: El responsable fue el equipo de desarrollo, quienes fueron responsables de desarrollar estrategias y acciones para mitigar los riesgos
   Gestión: El miembro del equipo con conocimiento en entrega continua es el responsable de la gestión de esta etapa, realizando pruebas para mitigar los riesgos.
   Administración: La administración en esta etapa, está a cargo del Scrum Master, quien administra el seguimiento del tratamiento de los riesgos.

5. **Monitoreo y revisión**: En esta etapa, se realiza seguimiento de los riesgos y revisión de la efectividad de los planes implementado, validando si se presentan cambios en la probabilidad e impacto de cada riesgo.
   Planificación: El responsable del equipo de entrega continua, realiza seguimiento y revisión de los riesgos.
   Gestión: El Scrum Master facilita las reuniones donde se revisa el estado de los riesgos.
   Administración: El Product Owner se asegura de que se actualice la lista de riesgos y además se asegura de que se mantenga alineada con los objetivos del producto.

6.1.6 Resultados de Plan de gestión de riesgos para Oral Data

Para el primer análisis, se tuvieron 2 reuniones donde el objetivo fue evaluar los riesgos en el ciclo de vida del producto, como resultado de las etapas de identificación y evaluación del riesgo, se construyó la siguiente tabla en conjunto:

Rol/Evento/ Producto	Item específico	Riesgo	Consecuencias del riesgo	Probabilidad de ocurrencia	Impacto
Producto	Product backlog	El product backlog carece de alineación estratégica	-Desviación de la visión y objetivos del producto -No se tiene claridad en los elementos del product backlog -el equipo de desarrollo trabaja en funcionalidades que no aportan al negocio -el producto final no satisface las necesidades del cliente - la entrega del proyecto puede retrasarse - Dificultades en la toma de decisiones que afectan la calidad y entrega del producto	Muy probable	Grave
Producto	Sprint backlog	Estimaciones incorrectas	-El equipo no tendrá suficiente tiempo para completar las tareas del sprint -el equipo podría estar sobrecargado de trabajo	Muy probable	Mayor

			-pérdida de motivación del equipo		
Evento	Sprint planning	Inviabilidad en las expectativas	- no se cumple con los objetivos del sprint -un equipo desmotivado - desalineación entre el equipo y el Product Owner	Moderada	Mayor
Evento	Daily	larga discusión sobre problemas no relacionados	- Falta de concentración y eficacia - desviación de la atención del objetivo principal de la reunión - no se planifica adecuadamente el trabajo del día - no se resuelven obstáculos	Muy probable	Significativo
Rol	Product Owner	El Product Owner no comunica de forma efectiva las necesidades del producto al equipo de desarrollo	- El equipo de desarrollo puede malinterpretar los requisitos del producto - construir funcionalidades que no cumplen con las expectativas del Product Owner. - Se pueden producir errores y defectos en el producto	Moderada	Grave
Rol	Equipo de desarrollo	El equipo de desarrollo no escribe código de alta calidad	- El producto final puede ser susceptible a errores - Producto difícil de mantener - Se tendrá problemas de escalabilidad	Moderado	Grave

Tabla 3. Tabla de plan de gestión de riesgos

A partir de allí, se construyó la siguiente matriz, con ayuda de la Product Owner:

		Gravedad				
		Insignificante	Menor	Significativo	Mayor	Grave
Probabilidad	Casi seguro					
	Muy probable			larga discusión sobre problemas no relacionados	Estimaciones incorrectas	El product backlog carece de alineación estratégica
	Moderado					El equipo de desarrollo no escribe código de alta calidad
					Inviabilidad en las expectativas	El Product Owner no comunica de forma efectiva las necesidades del producto al equipo de desarrollo
	Improbable					
	Raro					

Ilustración 2. Matriz de riesgos

Entre las medidas usadas, se crearon listas de chequeo, las cuales son poderosas para la gestión de riesgos, porque garantizan que se consideren los aspectos importantes, se reduce la probabilidad de errores y se ayuda a prevenir errores y contratiempos. Entre las acciones de mitigación, para cada riesgo, se construyeron listas de validación y verificación:

- Lista de chequeo para riesgo El product backlog carece de alineación estratégica (VyV)

Nombre del item	Descripción	Si/No
Propósito del producto	¿Todos los miembros entienden la misión y objetivos del producto?	
Involucrar stakeholders	¿Se incluye a los stakeholders en la validación del producto backlog?	

Herramientas de visualización de la visión	¿Se hizo uso de herramientas para la visualización de la visión?	
Identificación del público	¿Se tienen en cuenta usuarios potenciales y sus necesidades?	
Documentación del producto	¿Se tiene documentación donde se detalle visón del producto, propósito y público objetivo?	
Conexión con un product roadmap	¿Se conecta el product backlog con un elemento accionable en un product roadmap?	
disponibilidad del Product Backlog	¿Es el product backlog accesible para todos los miembros del equipo?	
Uso de herramientas adecuadas para creación	¿Se hace uso de herramientas que facilitan la gestión del product backlog?	
Revisión y retrospectivas	¿Se realizó una revisión del producto backlog? ¿y en caso de identificar elementos de mejora fueron ajustados?	

Tabla 4. Lista de chequeo para riesgo El product backlog

- Lista de chequeo para riesgo Estimaciones incorrectas (VyV)

Nombre del Item	Descripción	Si/No
Entendimiento historias de usuario	Los miembros del equipo entendieron el detalle de las historias de usuario	
Capacitación uso de técnicas	¿Se capacitó al equipo en el uso de técnicas de estimación de acuerdo a cada tipo de tarea?	
Conocimiento y experiencia	¿Para las estimaciones, se tuvo en cuenta la experiencia y conocimiento de cada miembro del equipo?	
Estimación conjunta	Se realizó estimación en conjunto con el equipo	
Dudas aclaradas	En caso de haber dudas, ¿fueron estas resueltas?	
Dependencias externas	¿Se evitó tener dependencias externas o en caso de haberlas fueron estas dirigidas para atenderse lo más pronto?	
Técnicas de estimación	Se aplicaron técnicas de estimación como Planning Poker, Wideband Delphi o T-Shirt Sizing	
Margen de error	¿Se tuvo en cuenta un margen de error para historias dependientes de tareas o respuestas de otros?	
Medidas correctivas	¿Se aplicaron medidas correctivas en caso de ser necesario?	

Tabla 5. Lista de chequeo para riesgo Estimaciones incorrectas

- Lista de chequeo para riesgo El Product Owner no comunica de forma efectiva las necesidades del producto al equipo de desarrollo

Nombre del Item	Descripción	Si/No
Plan de comunicación efectivo	Se han implementado prácticas de comunicación efectiva.	
Requisitos del producto documentados	Los requisitos del producto están claramente documentados.	
Equipo de desarrollo involucrado en la definición del producto	El equipo de desarrollo ha definido el producto.	
Feedback oportuno y constructivo	Se brinda feedback oportuno y constructivo al equipo de desarrollo.	

Tabla 6. Lista de chequeo para riesgo El Product Owner no comunica de forma efectiva

- Lista de chequeo para riesgo El equipo de desarrollo no escribe código de alta calidad

Nombre del Item	Descripción	Si/No
Implementar prácticas de desarrollo de software: TDD, refactorización.	Se implementan prácticas como TDD (Desarrollo Dirigido por Pruebas) y refactorización para mejorar la calidad del código.	
Realizar revisiones de código regulares	Se realizan revisiones de código de forma regular para detectar y corregir errores, mejorar la legibilidad y asegurar la calidad del código.	
Asegurar que el código esté bien documentado	El código está documentado de forma clara y precisa para facilitar su comprensión, mantenimiento y escalabilidad.	
Brindar capacitación al equipo de desarrollo en las mejores prácticas de desarrollo de software	Se brinda capacitación al equipo de desarrollo en las mejores prácticas de desarrollo de software para mejorar sus habilidades y conocimientos.	

Tabla 7. Lista de chequeo para riesgo El equipo de desarrollo

- Lista de chequeo para riesgo sprint planning Inviabilidad en las expectativas

Nombre del item	Descripción	Si/No
Estimación de Esfuerzo	Los integrantes del equipo realmente estimaron la cantidad de horas de trabajo necesarias para completar los elementos del backlog y la duración total de las actividades del sprint	
Asignación de Recursos	Se determino las necesidades de recursos, se esclareció quién hará qué y calcular el trabajo propio, asegurando de que las asignaciones coincidan con la capacidad del equipo	
Definición de Criterios de Aceptación	Se definió los criterios de aceptación para cada elemento del backlog	
Gestión de Dependencias	Se identifico y gestiono las dependencias entre las tareas para minimizar los efectos de la falta de tiempo y asegurar que el trabajo pueda completarse en el tiempo asignado	
Revisión de Criterios de Aceptación	Es de alta importancia asegurar de que cada elemento del backlog tenga criterios de aceptación claros y objetivos, para acelerar los debates durante la planificación y evitar malentendidos	
Revisión de la Planificación	¿El equipo si reviso el plan de sprint?, de esta manera se asegura de que todos los miembros estén cómodos con el objetivo y que este esté alineado con la visión estratégica y la hoja de ruta del producto	

Tabla 8. Lista de chequeo para riesgo sprint planning

- Lista de chequeo para riesgo daily larga discusión sobre problemas no relacionados

Nombre del item	Descripción	Si/No
Establecer un Tiempo Fijo	¿Se definió un tiempo específico para las reuniones diarias?	
Agenda Clara	El equipo debe preparar una agenda que incluya los temas a discutir	
Líder de la Reunión	El equipo debe de designar a un líder de la reunión que se encargue de mantener el enfoque en los temas de la agenda	
Reglas de Discusión	Se de planear sobre qué se puede discutir y qué no, para evitar que se aborden temas no relacionados con el trabajo del sprint	
Seguimiento de Decisiones	El grupo debe de Asegurarse de que todas las decisiones tomadas durante la reunión se documenten y se asignen responsabilidades claras para su seguimiento.	
Seguimiento de Problemas	El equipo debe de utilizar herramientas de seguimiento de problemas para documentar	

Tabla 9. Lista de chequeo para riesgo daily

Para el segundo análisis, se tuvo una tercera sesión para evaluar los riesgos, pero esta vez tuvo lugar la evaluación de riesgos del producto de software, donde a continuación se presentan los resultados de las etapas de identificación del riesgo, evaluación del riesgo y priorización de los riesgos.

Riesgo	Consecuencia	Causa	Requisito no funcional	Acciones de mitigacion	Probabiliad	Impacto	Riesgo inherente
Almacenamiento de información confidencial de forma no segura	Robo de datos	No se tienen en cuenta decisiones arquit	Seguridad	Implementar medidas de seguridad en el sistema, usar componentes confiables y actualizados	Media	Alto	Alto
El sistema no es compatible con diferentes plataformas o dispositivos	Fuga de potenciales usuarios del sistema	Elección inadecuada de tecnologías	Portabilidad	Considerar la necesidad de que funcione en diferentes plataformas o dispositivos, seleccionar tecnologias compatibles con las plataformas requeridas, Realizar pruebas de compatibilidad exhaustivas	Media	Medio	Medio
Interfaz de usuario compleja de usar	Insatisfacción de usuarios	Diseño de interfaz de usuario inadecuad	Usabilidad	Diseño de interfaz de usuario de acuerdo a principios de usabilidad (fácil e intuitiva), realizar pruebas de usabilidad con usuarios reales	Alta	Medio	Alto
El código fuente difícil de entender o modificar	Dificultad para mantener y escalar el sistema	No se aplican estándares de buenas prá	Mantenibilidad	Establecer buenas prácticas de desarrollo a tener en cuenta, utilizar herramientas de análisis de código estático	Media	Medio	Medio
Tiempos de respuesta lentos	Lentitud en el sistema	Mala optimización de código	Rendimiento	Relizar pruebas de rendimiento para identificar problemas	Media	Medio	Medio
Gestión de errores n está implementada completamente	Frustración en los usuarios	No se consideraron posibles casos de error durante el diseño	Protección contra Errores de Usuario -- tolerancia a fallos	Realizar análisis de casos de error, implementar pruebas de error exhaustivas	Media	Bajo	Medio
Cobertura de pruebas unitarias insuficiente	Mayor riesgo de errores que no se detectan durante las pruebas unitarias.	El equipo de desarrollo no tiene suficient	Confiabilidad	Requerir pruebas unitarias como obligatorias en fase de desarrollo, automatizar pruebas de desarrollo	Media	Bajo	Medio
Diseño modular deficiente	Dificultad en mantenimieto a largo plazo	Desconocimiento en principios de modul	Modularidad	Utilizar herramientas de diseño modular, realizar revisiones, usar pruebas de integración modular	Media	Alto	Alto

Ilustración 3. Evaluación de riesgo

6.1.7 Stakeholders

Stakeholders Internos

- **Estudiantes de Odontología:** Usuarios principales que utilizarán el sistema para gestionar sus prácticas clínicas.
- **Pacientes:** Usuarios que serán gestionados por los estudiantes a través del sistema.
- **Facultad de Odontología:** Institución educativa que supervisa las prácticas clínicas y el uso del sistema.

Stakeholders Externos

- **Proveedores de Tecnología:** Empresas o individuos que proporcionan soporte técnico y servicios de infraestructura.
- **Reguladores:** Entidades gubernamentales que establecen las normativas para la educación y prácticas odontológicas.

6.1.8 Roadmap del Proyecto

Fase 1: Inicio del Proyecto

- **Duración:** 1 mes
- **Actividades:**
    - Definición de objetivos y alcance del proyecto.
    - Identificación de stakeholders y recolección de requisitos.
    - Planificación inicial y creación del backlog del producto.

Fase 2: Diseño y Planificación

- **Duración:** 2 meses
- **Actividades:**
    - Diseño de la arquitectura del sistema.
    - Diseño del modelo de datos.
    - Creación de prototipos de interfaz de usuario.
    - Planificación detallada de sprints.

Fase 3: Desarrollo

- **Duración:** 6 meses

- **Actividades:**
    - Implementación de funcionalidades en sprints iterativos.
    - Desarrollo de registro y autenticación de usuarios.
    - Desarrollo de gestión de perfiles.
    - Implementación de gestión de citas.
    - Desarrollo de chat en tiempo real.
    - Implementación de notificaciones locales.
    - Pruebas unitarias y de integración.

Fase 4: Pruebas y Validación

- **Duración:** 2 meses
- **Actividades:**
    - Pruebas de usuario.
    - Validación de funcionalidades.
    - Corrección de errores y optimización del sistema.

Fase 5: Implementación y Despliegue

- **Duración:** 1 mes
- **Actividades:**
    - Preparación del entorno de producción.
    - Despliegue de la aplicación en dispositivos móviles.
    - Capacitación de usuarios finales.

6.1.9 Diagrama de Gant

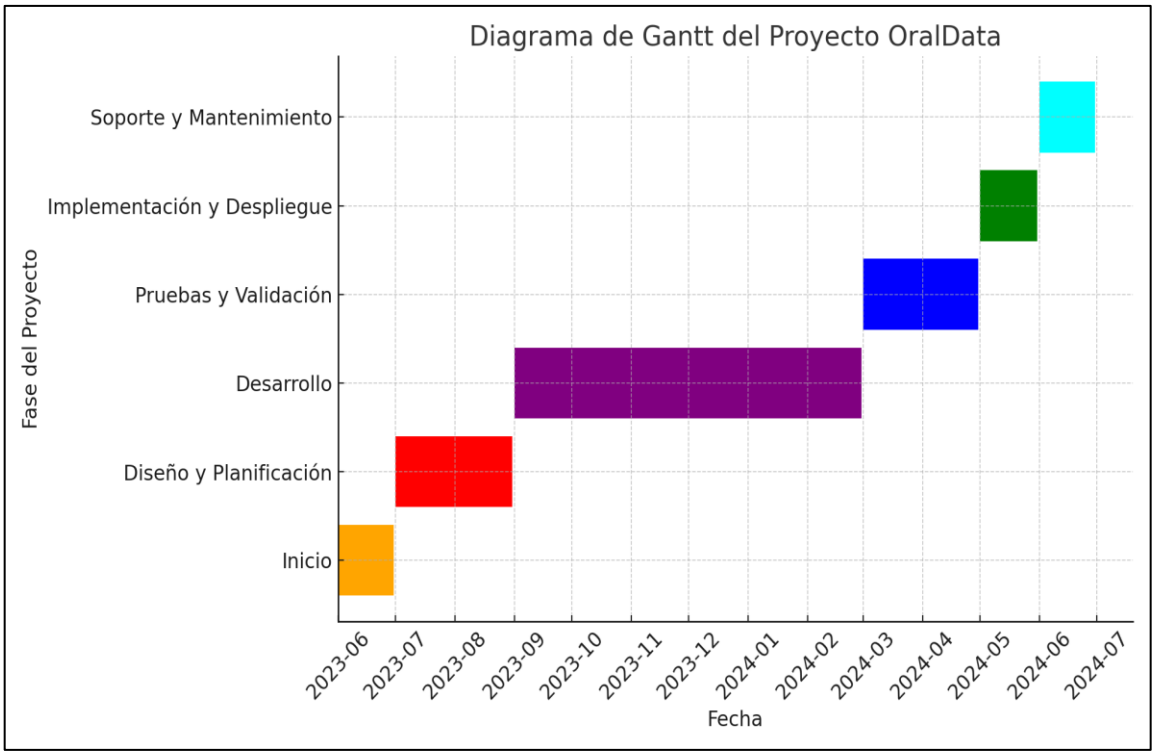


Ilustración 4. Diagrama de Gant

6.1.10 Modelo de Datos

El modelo de datos del sistema OralData se estructura principalmente en las siguientes colecciones:

- **users**
- **appointments**
- **messages**
- **available_slots**

Users

Campo	Tipo	Descripción
uid	String	Identificador único del usuario
tipoDocumento	String	Tipo de documento del usuario
numeroDocumento	String	Número de documento del usuario

Campo	Tipo	Descripción
name	String	Nombre del usuario
email	String	Correo electrónico del usuario
numcelular	String	Número de celular del usuario
genre	String	Género del usuario
bornDate	Date	Fecha de nacimiento del usuario
role	String	Rol del usuario (estudiante/paciente)
profImage	String	URL de la imagen de perfil
imageUrl	String	URL de la imagen de valoración
token	String	Token de notificación

La tabla anterior nos especifica el siguiente contenido

uid (String): Identificador único del usuario. Este campo almacena una cadena de texto que representa un ID único para cada usuario, utilizado para identificar a los usuarios de manera exclusiva en el sistema.

TipoDocumento (String): Tipo de documento del usuario. Este campo almacena una cadena de texto que indica el tipo de documento de identificación del usuario (por ejemplo, cédula, pasaporte, etc.).

numeroDocumento (String): Número de documento del usuario. Este campo almacena una cadena de texto que contiene el número del documento de identificación del usuario.

name (String): Nombre del usuario. Este campo almacena una cadena de texto que contiene el nombre completo del usuario.

email (String): Correo electrónico del usuario. Este campo almacena una cadena de texto que contiene la dirección de correo electrónico del usuario.

numcelular **(String):** Número de celular del usuario. Este campo almacena una cadena de texto que contiene el número de teléfono celular del usuario.

genre (String): Género del usuario. Este campo almacena una cadena de texto que indica el género del usuario (por ejemplo, masculino, femenino, etc.).

bornDate (Date): Fecha de nacimiento del usuario. Este campo almacena una fecha que representa la fecha de nacimiento del usuario.

role (String): Rol del usuario. Este campo almacena una cadena de texto que indica el rol del usuario en el sistema, ya sea estudiante o paciente.

profImage (String): URL de la imagen de perfil. Este campo almacena una cadena de texto que contiene la URL de la imagen de perfil del usuario.

imageUrl (String): URL de la imagen de valoración. Este campo almacena una cadena de texto que contiene la URL de la imagen de valoración asociada al usuario.

token (String): Token de notificación. Este campo almacena una cadena de texto que contiene el token utilizado para enviar notificaciones push al usuario.

Appointments

Campo	Tipo	Descripción
id	String	Identificador único de la cita
studentId	String	Identificador del estudiante
patientId	String	Identificador del paciente
date	Date	Fecha de la cita
timeSlot	String	Hora de la cita
clinicType	String	Tipo de clínica (niños/adultos)

id (String): Identificador único de la cita. Este campo almacena una cadena de texto que representa un ID único para cada cita, utilizado para identificar las citas de manera exclusiva en el sistema.

studentId (String): Identificador del estudiante. Este campo almacena una cadena de texto que contiene el ID del estudiante asociado a la cita, permitiendo identificar al estudiante que atenderá la cita.

patientId (String): Identificador del paciente. Este campo almacena una cadena de texto que contiene el ID del paciente asociado a la cita, permitiendo identificar al paciente que será atendido.

date (Date): Fecha de la cita. Este campo almacena una fecha que representa el día en que se llevará a cabo la cita.

timeSlot (String): Hora de la cita. Este campo almacena una cadena de texto que indica el intervalo de tiempo asignado para la cita (por ejemplo, "10:00 AM - 11:00 AM").

clinicType (String): Tipo de clínica. Este campo almacena una cadena de texto que indica el tipo de clínica en la que se llevará a cabo la cita, especificando si es una clínica para niños o adultos.

messages

Campo	Tipo	Descripción
senderId	String	Identificador del remitente
receiverId	String	Identificador del receptor
message	String	Contenido del mensaje
timestamp	Date	Fecha y hora del mensaje

senderId (String): Identificador del remitente. Este campo almacena una cadena de texto que representa el ID único del usuario que envía el mensaje, permitiendo identificar al remitente en el sistema.

receiverId (String): Identificador del receptor. Este campo almacena una cadena de texto que representa el ID único del usuario que recibe el mensaje, permitiendo identificar al receptor en el sistema.

message (String): Contenido del mensaje. Este campo almacena una cadena de texto que contiene el contenido del mensaje enviado entre usuarios.

timestamp (Date): Fecha y hora del mensaje. Este campo almacena la fecha y hora en que se envió el mensaje, permitiendo rastrear cuándo se realizó la comunicación.

Available_slots

Campo	Tipo	Descripción
date	Date	Fecha del slot disponible
timeSlot	String	Hora del slot disponible
clinicType	String	Tipo de clínica (niños/adultos)
available	Boolean	Disponibilidad del slot

date (Date): Fecha del slot disponible. Este campo almacena una fecha que representa el día en que hay un slot (intervalo de tiempo) disponible para una cita.

timeSlot (String): Hora del slot disponible. Este campo almacena una cadena de texto que indica el intervalo de tiempo disponible para una cita (por ejemplo, "10:00 AM - 11:00 AM").

clinicType (String): Tipo de clínica. Este campo almacena una cadena de texto que indica el tipo de clínica en la que está disponible el slot, especificando si es una clínica para niños o adultos.

available (Boolean): Disponibilidad del slot. Este campo almacena un valor booleano que indica si el slot está disponible (true) o no (false).

Relaciones entre las colecciones

- **users - appointments:** Los usuarios (estudiantes y pacientes) están relacionados con las citas. Un estudiante puede tener múltiples citas y un paciente puede estar agendado en múltiples citas.
- **users - messages:** Los usuarios pueden enviar y recibir mensajes. Cada mensaje tiene un remitente y un receptor.
- **users - available_slots:** Los estudiantes gestionan los slots disponibles para agendar citas.
- **appointments - messages:** Los mensajes pueden estar relacionados con las citas, facilitando la comunicación entre estudiantes y pacientes respecto a las citas.

6.2. Ingeniería de Requisitos

6.2.1 Elicitación de Requisitos

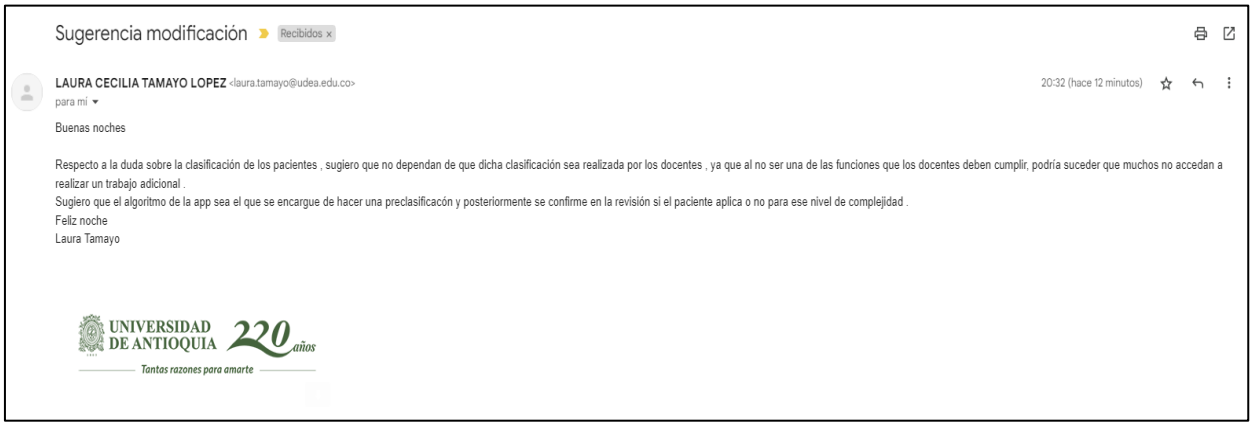
Se realizaron entrevistas a 2 personas con el objetivo de conocer el estado del proceso (una de la Universidad Cooperativa de Colombia y otra de la universidad de Antioquia), las cuales se adjuntan como hipervínculo:

entrevista elicitación.docx

Entrevista elicitación UCC.mp4

Entrevista elicitacion UdeA.mp4

Luego de revisar el proceso, con los profesores, quienes serían dueños del producto a realizar, se acordó que estos tendrán un rol de visualización para temas de historias clínicas.


Sugerencia modificación Recibidos ×

LAURA CECILIA TAMAYO LOPEZ <laura.tamayo@udea.edu.co> 20:32 (hace 12 minutos)
para mí

Buenas noches

Respecto a la duda sobre la clasificación de los pacientes , sugiero que no dependan de que dicha clasificación sea realizada por los docentes , ya que al no ser una de las funciones que los docentes deben cumplir, podría suceder que muchos no accedan a realizar un trabajo adicional .
Sugiero que el algoritmo de la app sea el que se encargue de hacer una preclasificacón y posteriormente se confirme en la revisión si el paciente aplica o no para ese nivel de complejidad .
Feliz noche
Laura Tamayo

UNIVERSIDAD DE ANTIOQUIA 220 años
Tantas razones para amarte

6.2.2 Descripción de los roles del área problemática (Funciones, información personal requerida para el sistema)

Paciente: Es la persona que requiere atención médica.

Puede realizar las siguientes actividades:

- Registrarse en el sistema.
- Diligenciar encuesta de clasificación
- Cancelar una cita.

Estudiante: Es el alumno que está cursando la asignatura.

Puede realizar las siguientes actividades:

- Visualizar los pacientes en la base de datos.
- Agendar una cita.
- Cancelar una cita.

6.3 Descripción gráfica del proyecto en base a una infraestructura simulada

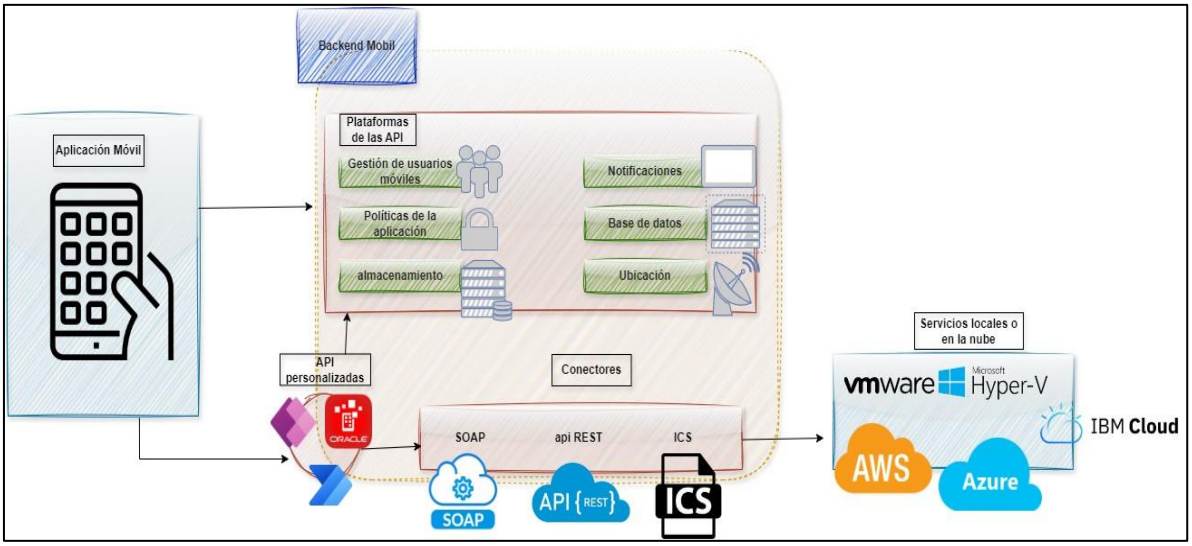


Ilustración 5. *Arquitectura simulada del proyecto de Oral Data.*[1]

Componentes Principales

Aplicación Móvil

Función: La interfaz de usuario con la cual interactúan los usuarios finales.

Conexiones: Se conecta con el backend móvil y con las API personalizadas.

Backend Móvil

Función: Gestión central de las aplicaciones móviles, incluyendo la lógica de negocio y la interacción con bases de datos y otros servicios.

Plataformas de las API

Gestión de Usuarios Móviles: Maneja las cuentas y autenticación de usuarios.

Políticas de la Aplicación: Define y aplica reglas y configuraciones de uso.

Almacenamiento: Gestión de datos almacenados, tanto en bases de datos como en otros repositorios.

[1] Fuente: Elaboración propia

Notificaciones: Envío de mensajes y alertas a los dispositivos móviles.

Base de Datos: Almacena información estructurada y no estructurada.

Ubicación: Servicios de geolocalización y manejo de datos de ubicación.

Conectores

Función: Permiten la integración con diferentes servicios y protocolos.

SOAP: Protocolo estándar para intercambio de información estructurada en la implementación de servicios web.

API REST: Arquitectura de transferencia de estado representacional para la interacción con servicios web.

ICS (Integration Cloud Service) : Servicios de integración en la nube para conectar diferentes aplicaciones y servicios.

Servicios en la Nube:

VMware: Ofrece soluciones de virtualización.

Microsoft Hyper-V: Proveedor de servicios de virtualización.

AWS (Amazon Web Services): Proveedor de servicios en la nube que ofrece una variedad de soluciones para almacenamiento, bases de datos, cómputo, etc.

Azure: Plataforma de servicios en la nube de Microsoft.

IBM Cloud: Proveedor de servicios en la nube de IBM, incluyendo soluciones de inteligencia artificial, blockchain, IoT, etc.

Flujo de Información

La Aplicación Móvil interactúa directamente con el Backend Móvil y las API Personalizadas.

El Backend Móvil maneja la lógica de negocio y la interacción con las plataformas de las API, incluyendo la gestión de usuarios, políticas de la aplicación, almacenamiento, notificaciones, bases de datos y servicios de ubicación.

Las API Personalizadas permiten la integración específica y personalizada con otras aplicaciones y servicios, usando conectores como SOAP, API REST y ICS.

Los Conectores permiten la conexión del sistema con diferentes Servicios en la Nube, facilitando la escalabilidad y flexibilidad del sistema.

Los Servicios en la Nube ofrecen la infraestructura necesaria para alojar y operar las aplicaciones y servicios, proporcionando soluciones escalables y manejables.

Esta arquitectura modular y escalable permite una gestión eficiente de aplicaciones móviles, integrando múltiples servicios y garantizando la interacción fluida entre diferentes componentes del sistema.

6.5 Arquitectura con Diagrama Preconceptual

https://drive.google.com/file/d/1UWM0giwLd_H_d_J0pdLn0ZbYMPL1ckFe/view?usp=sharing

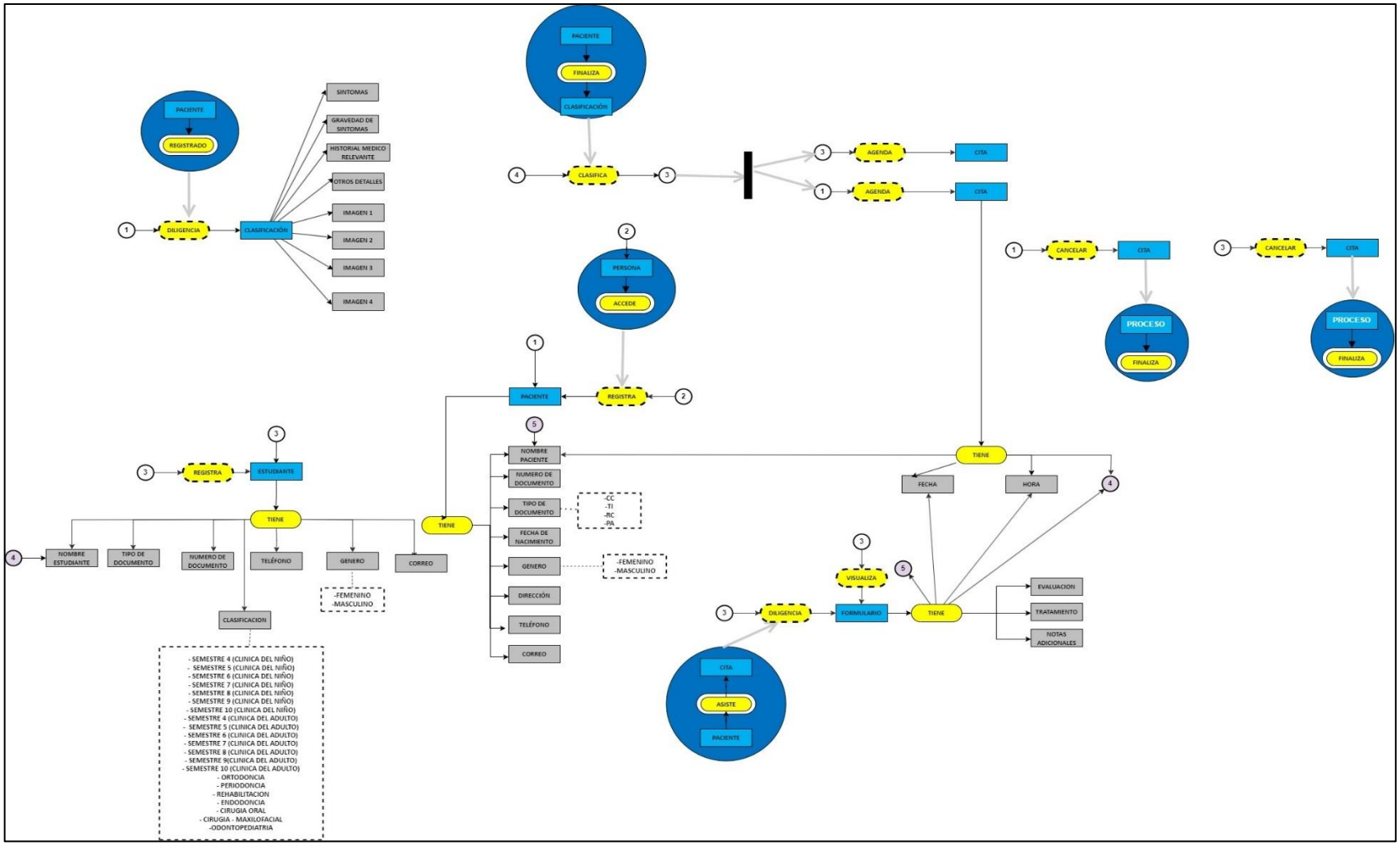
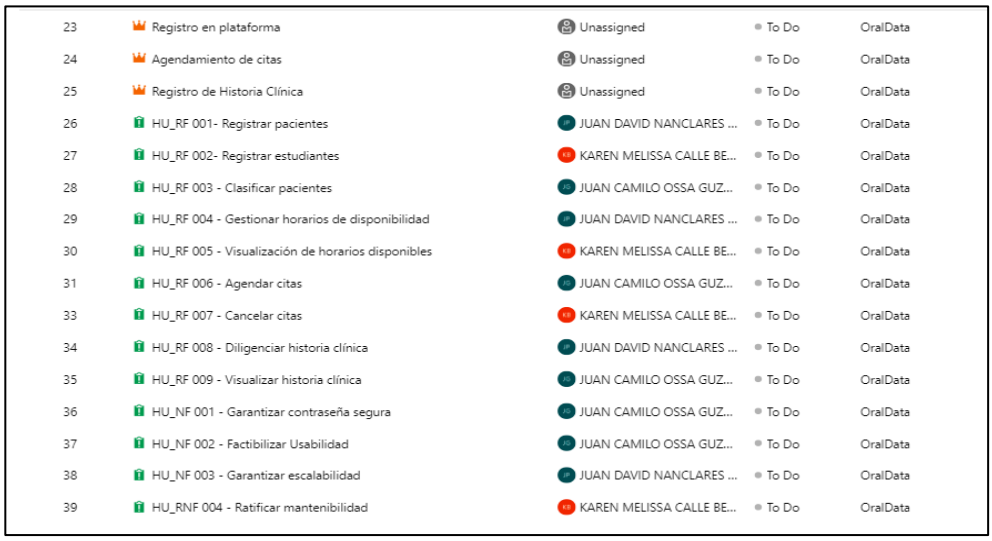


Ilustración 6. Diagrama preconceptual

6.6 Requerimientos

6.6.1 Historias de usuario

En la Ilustración 4, se presenta el backlog de Oral Data, el cual está dividido en 3 épicas que contienen las 3 grandes funcionalidades principales de la aplicación, las cuales son registro en plataforma, agendamiento de citas y registro de la historia clínica



23	Registro en plataforma	Unassigned	To Do	OralData
24	Agendamiento de citas	Unassigned	To Do	OralData
25	Registro de Historia Clínica	Unassigned	To Do	OralData
26	HU_RF 001- Registrar pacientes	JUAN DAVID NANCLARES ...	To Do	OralData
27	HU_RF 002- Registrar estudiantes	KAREN MELISSA CALLE BE...	To Do	OralData
28	HU_RF 003 - Clasificar pacientes	JUAN CAMILO OSSA GUZ...	To Do	OralData
29	HU_RF 004 - Gestionar horarios de disponibilidad	JUAN DAVID NANCLARES ...	To Do	OralData
30	HU_RF 005 - Visualización de horarios disponibles	KAREN MELISSA CALLE BE...	To Do	OralData
31	HU_RF 006 - Agendar citas	JUAN CAMILO OSSA GUZ...	To Do	OralData
33	HU_RF 007 - Cancelar citas	KAREN MELISSA CALLE BE...	To Do	OralData
34	HU_RF 008 - Diligenciar historia clínica	JUAN DAVID NANCLARES ...	To Do	OralData
35	HU_RF 009 - Visualizar historia clínica	JUAN CAMILO OSSA GUZ...	To Do	OralData
36	HU_NF 001 - Garantizar contraseña segura	JUAN CAMILO OSSA GUZ...	To Do	OralData
37	HU_NF 002 - Factibilizar Usabilidad	JUAN CAMILO OSSA GUZ...	To Do	OralData
38	HU_NF 003 - Garantizar escalabilidad	JUAN DAVID NANCLARES ...	To Do	OralData
39	HU_RNF 004 - Ratificar mantenibilidad	KAREN MELISSA CALLE BE...	To Do	OralData

Ilustración 7. *Backlog de product – OralData*

Cada historia de usuario contiene un requerimiento y criterios de aceptación de dicha historia, a continuación, se presentan algunas de las historias de usuario con requerimientos funcionales de la aplicación. A continuación, en las Figuras 4, 5 6 y 7 se muestra el detalla de algunas de las historias de usuario del proyecto

Ilustración 8. HU_RF 001 – Registrar pacientes

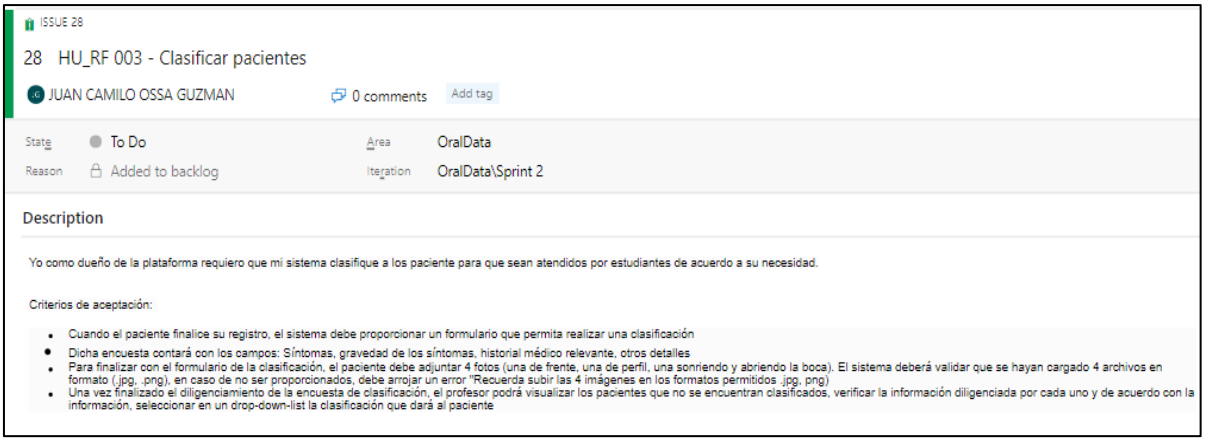


Ilustración 9. HU_RF Clasificar paciente

Ilustración 10. HU_RF Visualización de horarios disponibles

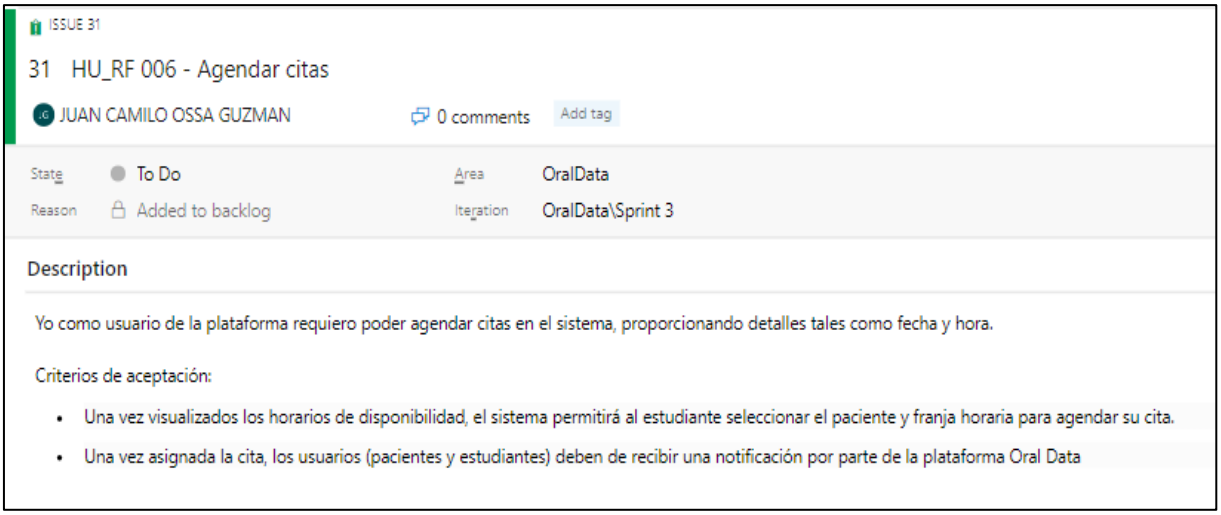


Ilustración 11. HU_RF Agendar Citas

7.RESULTADOS Y DISCUSIÓN

7.1 Esquematización de resultados esperados

Vistas funcionales: En las siguientes figuras, se presentan las vistas funcionales correspondientes a las historias de usuario presentadas:

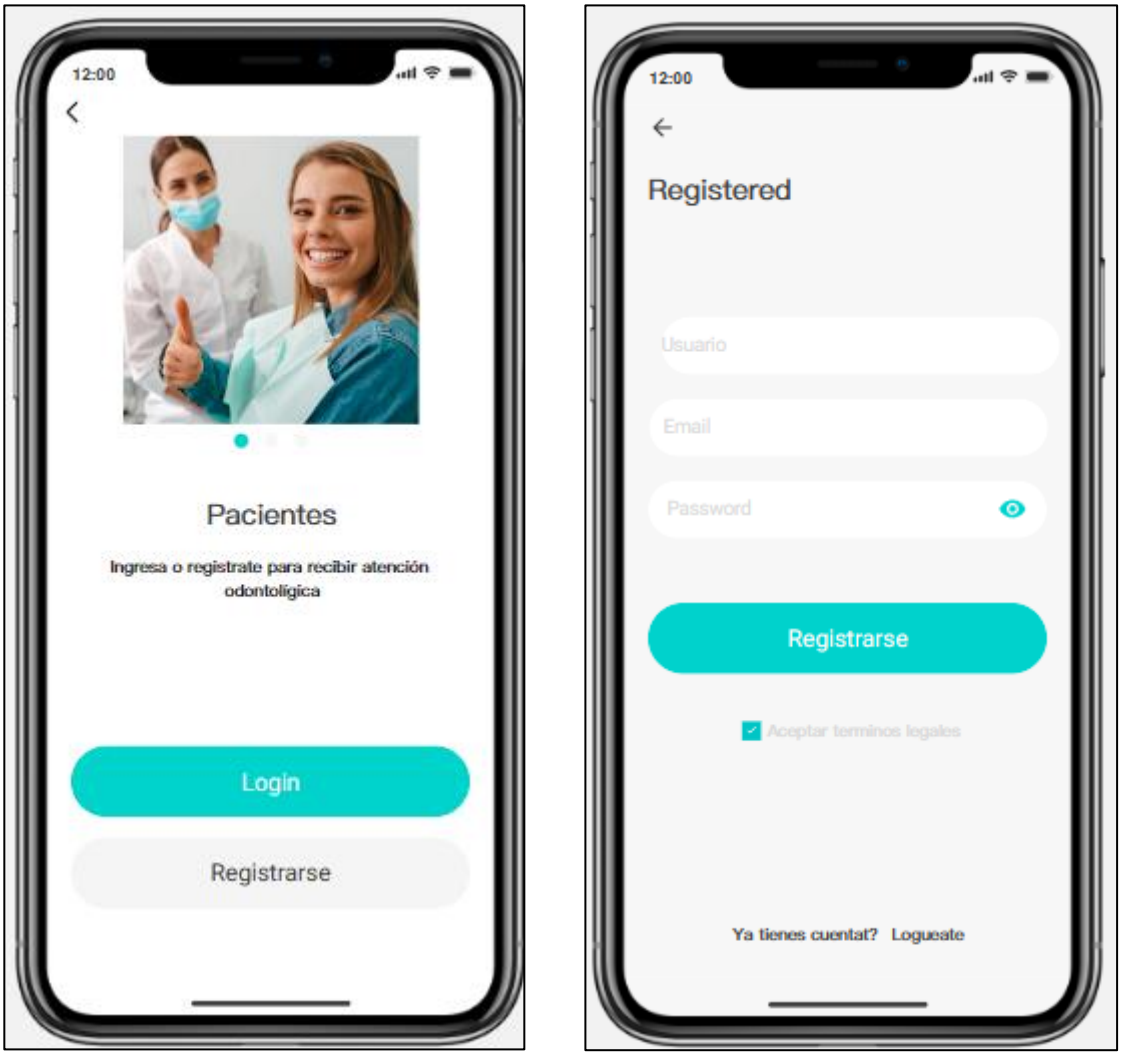


Ilustración 12. Vista funcional relacionada con HU_RF 001- Registrar pacientes

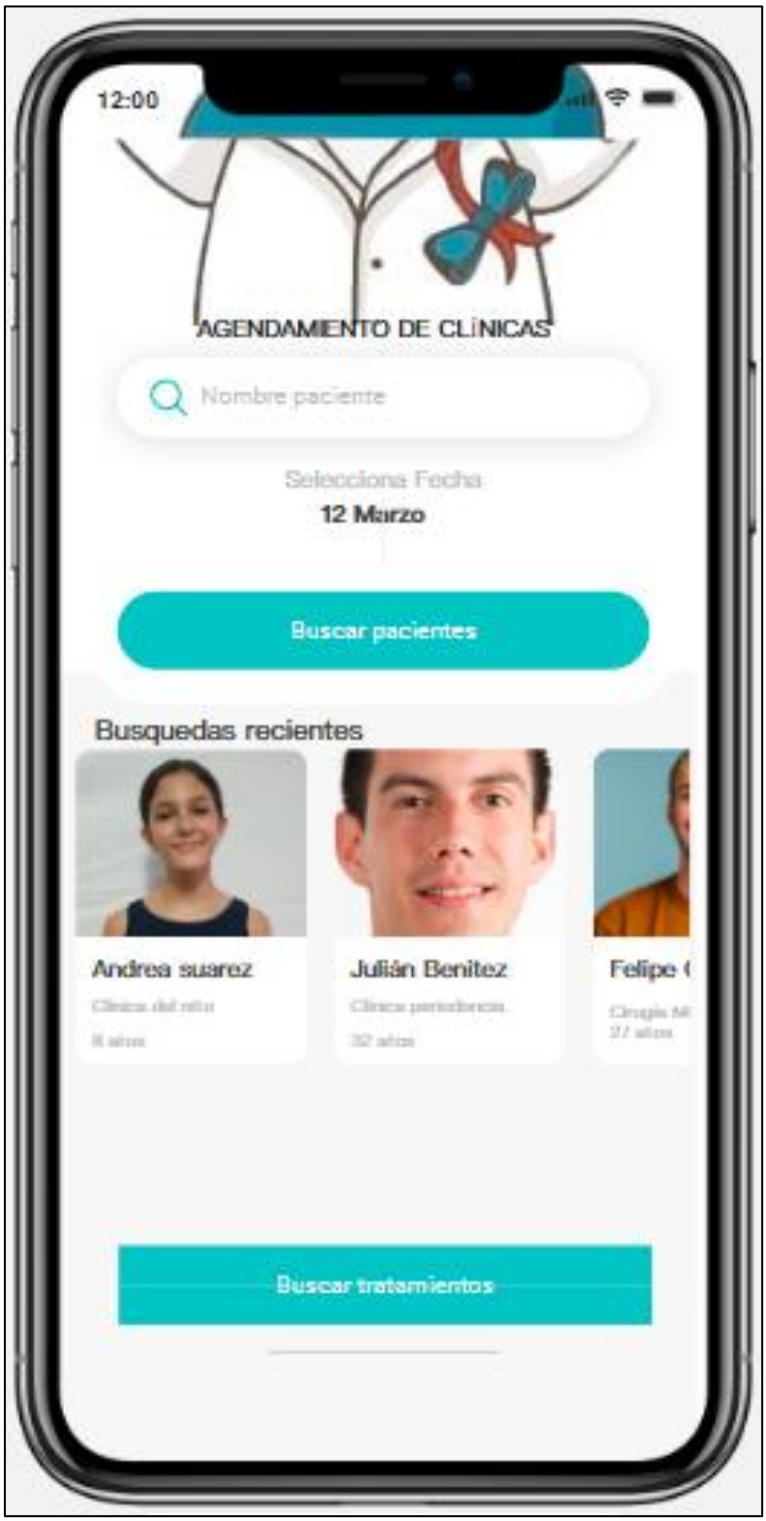


Ilustración 13. Vista funcional relacionada con HU_RF 003- Clasificar paciente

Ilustración 14 Vista funcional relacionada con HU_RF 004- Gestionar horarios de disponibilidad

7.2 Arquitectura del Proyecto

El sistema de gestión de pacientes para prácticas clínicas (OralData) se basa en una arquitectura de tres capas, utilizando Flutter para el desarrollo de la aplicación móvil y Firebase como backend. Las tres capas principales son:

1. **Capa de Presentación:** Interfaz de usuario desarrollada en Flutter.
2. **Capa de Lógica de Negocio:** Implementación de la lógica de la aplicación, incluidas las funcionalidades de gestión de usuarios, citas y chat.
3. **Capa de Datos:** Almacenamiento y gestión de datos utilizando Firebase Firestore y Firebase Auth.

2. Componentes Principales

- **Aplicación Móvil (Flutter):** Interfaz de usuario y lógica de negocio.
- **Firebase Auth:** Autenticación y gestión de usuarios.
- **Firebase Firestore:** Almacenamiento de datos.
- **Firebase Storage:** Almacenamiento de imágenes de perfil y valoración.
- **Firebase Cloud Messaging (FCM):** Envío de notificaciones locales.

7.3 Diagrama de Arquitectura

https://drive.google.com/file/d/1vs0m_CxRWAYiSND7tEQ907rBDkk1Gn25/view?usp=sharing

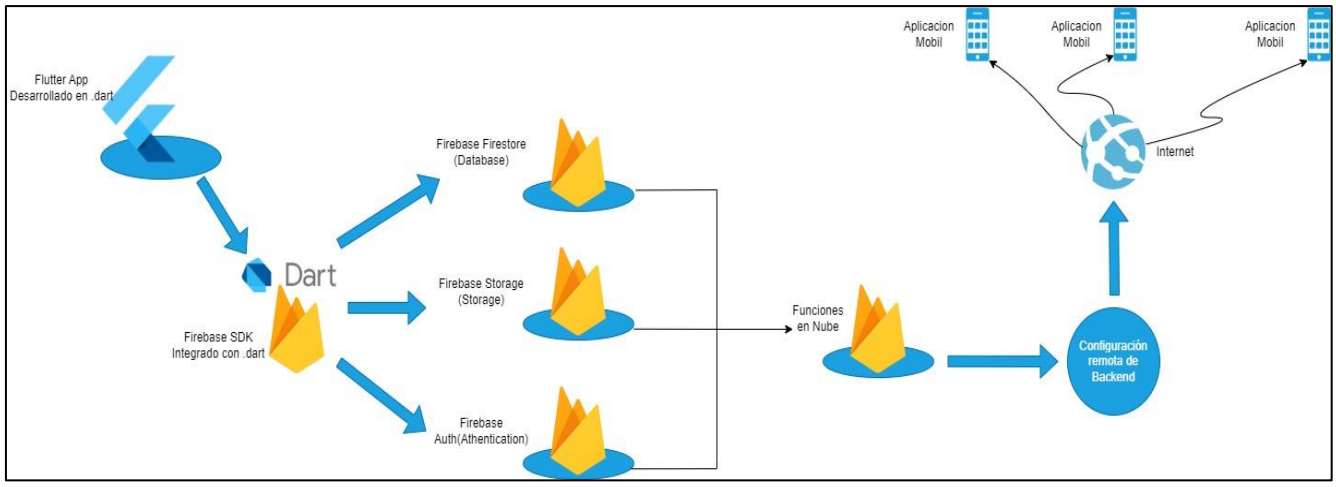


Ilustración 15. Arquitectura de la app Oral Data en dispositivos móviles

La imagen anterior es el diagrama de arquitectura para la aplicación de OralData, desarrollada con Flutter y Dart, a continuación, se describe cada componente del diagrama:

OralData App:

Es la aplicación móvil desarrollada en Dart.

Firebase SDK Integrado con Dart:

La aplicación OralData utiliza el SDK de Firebase, que está integrado con el lenguaje Dart.

Firebase Firestore (Database):

Firebase Firestore se utiliza para la gestión de la base de datos, almacenando información relevante como las citas, los pacientes y los estudiantes. Las flechas indican que el SDK de Firebase en Dart interactúa con Firestore.

Firebase Storage (Storage):

Firebase Storage se utiliza para el almacenamiento de archivos, como imágenes de panorámicas dentales, dentaduras, entre otros. El SDK de Firebase en Dart interactúa con el servicio de almacenamiento.

Firebase Auth (Authentication):

Firebase Authentication se utiliza para la autenticación de usuarios, garantizando que solo los usuarios autorizados puedan acceder a la aplicación OralData. El SDK de Firebase en Dart interactúa con el servicio de autenticación.

Funciones en Nube:

Las funciones en la nube de Firebase se utilizan para ejecutar lógica en el backend, como el procesamiento de datos de las citas o el envío de notificaciones. Estas funciones pueden interactuar con Firebase Firestore y Firebase Storage.

Configuración Remota de Backend:

Indica la configuración remota del backend, que se realiza a través de Internet para actualizar y gestionar la aplicación OralData de manera eficiente.

Aplicación Móvil:

Los dispositivos móviles con la aplicación OralData instalada se conectan a Internet para interactuar con el backend configurado remotamente, permitiendo a los usuarios agendar y gestionar citas odontológicas

7.4 Descripción de Componentes

- **Aplicación Móvil (Flutter):** Proporciona la interfaz de usuario y maneja la lógica de presentación. Los usuarios pueden registrarse, iniciar sesión, gestionar citas y comunicarse a través del chat.
- **Firebase Auth:** Maneja la autenticación de usuarios, incluyendo el registro y el inicio de sesión.
- **Firebase Firestore:** Base de datos NoSQL que almacena información de usuarios, citas, mensajes de chat y tokens de notificación.
- **Firebase Storage:** Almacena imágenes de perfil y de valoración.
- **Firebase Cloud Messaging (FCM):** Servicio de mensajería que permite enviar notificaciones locales a los dispositivos móviles de los usuarios.

7.5 Patrones de diseño

7.5.1 Patrones Arquitectónicos Utilizados

Modelo-Vista-Controlador (MVC)

El patrón MVC separa la aplicación en tres componentes principales: el modelo, la vista y el controlador. En el proyecto:

Modelo: Representado por las clases como `Patient`, `PatientDiligency`, `MyUser`, `AppointmentSlot`, que manejan la lógica de datos y la interacción con Firebase.

Vista: Representada por las páginas y widgets de Flutter como `ProfilePage`, `HomePage`, `AppointmentDetailsPage`, etc., que manejan la interfaz de usuario.

Controlador: Representado por los métodos y funciones en las páginas y los archivos de API (`firebase_api.dart`, `auth_api.dart`, etc.), que manejan la lógica de negocio y la interacción entre la vista y el modelo.

Proveedor de Servicios (Service Provider)

Este patrón se usa para encapsular la lógica de negocio y los servicios de la aplicación en clases de servicio. En el proyecto, los archivos `firebase_api.dart`, `auth_api.dart` y `notificationservices.dart, actúan como proveedores de servicios que abstraen las operaciones con Firebase y otras funcionalidades de la aplicación.

Repositorio (Repository Pattern)

El patrón Repositorio se usa para separar la lógica de acceso a datos de la lógica de negocio. Este patrón se puede observar en la forma en que se ha estructurado la interacción con Firebase a través de archivos de API como `firebase_api.dart`, `agenda_api.dart`, `Appointment_api.dart`, etc. Estos archivos sirven como repositorios que manejan la comunicación con Firebase Firestore y otros servicios de Firebase.

Singleton

El patrón Singleton asegura que una clase tenga solo una instancia y proporciona un punto de acceso global a esa instancia. En aplicaciones Flutter, esto se puede ver en la instancia de Firebase que se inicializa una sola vez y se usa en toda la aplicación.

Factory Method

El patrón Factory Method se usa para crear objetos sin especificar la clase exacta de objeto que se creará. Esto se puede ver en métodos como `fromJson` y `toJson` en tus modelos de datos (`Patient`, `MyUser`), que manejan la creación de instancias de estas clases a partir de datos JSON.

Observer (Observador)

Este patrón permite que un objeto notifique a otros objetos sobre cambios en su estado. En el contexto de Firebase, las actualizaciones en tiempo real de Firestore utilizan un mecanismo similar al patrón Observer, donde los cambios en los datos se notifican a los listeners registrados.

7.6 Resultados obtenidos en el desarrollo del software de Oral Data

7.6.1 Historias de Usuario en la infraestructura de Oral Data

1. Registro de Paciente

Historia de Usuario: Como paciente, quiero registrarme en la aplicación para poder acceder a los servicios de gestión de citas.

Criterios de Aceptación:

- El paciente debe poder ingresar su información personal, incluyendo nombre, correo electrónico, teléfono y contraseña.
- El sistema debe verificar que el correo electrónico no esté registrado previamente.
- El paciente debe recibir una confirmación de registro exitoso.

2. Agendamiento de Citas

Historia de Usuario: Como estudiante, quiero agendar citas con pacientes para gestionar mis prácticas clínicas.

Criterios de Aceptación:

- El estudiante debe poder seleccionar un paciente, una fecha y un horario disponible.
- El sistema debe verificar la disponibilidad del horario seleccionado.
- El estudiante y el paciente deben recibir una notificación de la cita agendada.

3. Gestión de Disponibilidad

Historia de Usuario: Como estudiante, quiero gestionar mi disponibilidad para que pueda agendar pacientes en los horarios disponibles.

Criterios de Aceptación:

- El estudiante debe poder agregar, modificar y eliminar horarios de disponibilidad.
- El sistema debe evitar la duplicación de horarios.
- La disponibilidad debe actualizarse en tiempo real.

4. Comunicación a través del Chat

Historia de Usuario: Como paciente, quiero comunicarme con el estudiante a través del chat para resolver dudas antes de la cita.

Criterios de Aceptación:

- El paciente debe poder iniciar una conversación con el estudiante.
- Ambos usuarios deben recibir notificaciones de nuevos mensajes.

- El chat debe permitir el envío de texto e imágenes.
-

7.6.2 Casos de Uso

Caso de Uso: Registro de Paciente

Actor Principal: Paciente

Precondiciones: El usuario no debe estar registrado.

Flujo Principal:

1. El paciente accede a la pantalla de registro.
2. El paciente ingresa su información personal.
3. El sistema valida los datos ingresados.
4. El sistema registra al paciente y envía una confirmación.

Flujos Alternativos:

- Si el correo electrónico ya está registrado, el sistema muestra un mensaje de error.

Caso de Uso: Agendamiento de Citas

Actor Principal: Estudiante

Precondiciones: El estudiante debe estar autenticado.

Flujo Principal:

1. El estudiante selecciona un paciente y una fecha.
2. El estudiante elige un horario disponible.
3. El sistema verifica la disponibilidad del horario.
4. El sistema registra la cita y notifica al estudiante y al paciente.

Flujos Alternativos:

- Si el horario no está disponible, el sistema muestra un mensaje de error.

7.6.3 Requisitos Funcionales y No Funcionales

Requisitos Funcionales

1. **Registro y Autenticación:**
    - El sistema debe permitir el registro de pacientes y estudiantes.
    - El sistema debe autenticar a los usuarios antes de permitirles acceder a las funcionalidades.
2. **Gestión de Citas:**
    - Los estudiantes deben poder agendar, modificar y cancelar citas.
    - Los pacientes deben poder ver sus citas agendadas.
3. **Gestión de Disponibilidad:**
    - Los estudiantes deben poder gestionar sus horarios de disponibilidad.
    - El sistema debe evitar la duplicación de horarios.
4. **Chat:**
    - El sistema debe permitir la comunicación en tiempo real entre estudiantes y pacientes.

Requisitos No Funcionales

1. **Seguridad:**
    - La información personal de los usuarios debe estar protegida mediante autenticación y cifrado.
    - El acceso a los datos debe estar restringido según el rol del usuario.
2. **Performance:**
    - El sistema debe ser capaz de manejar múltiples usuarios concurrentes sin degradar el rendimiento.
    - Las acciones de gestión de citas y disponibilidad deben reflejarse en tiempo real.
3. **Usabilidad:**

- La interfaz de usuario debe ser intuitiva y fácil de usar.
- Las notificaciones deben ser claras y proporcionar feedback adecuado a los usuarios.

7.6.4 Integración de Imágenes

Usuario anónimo

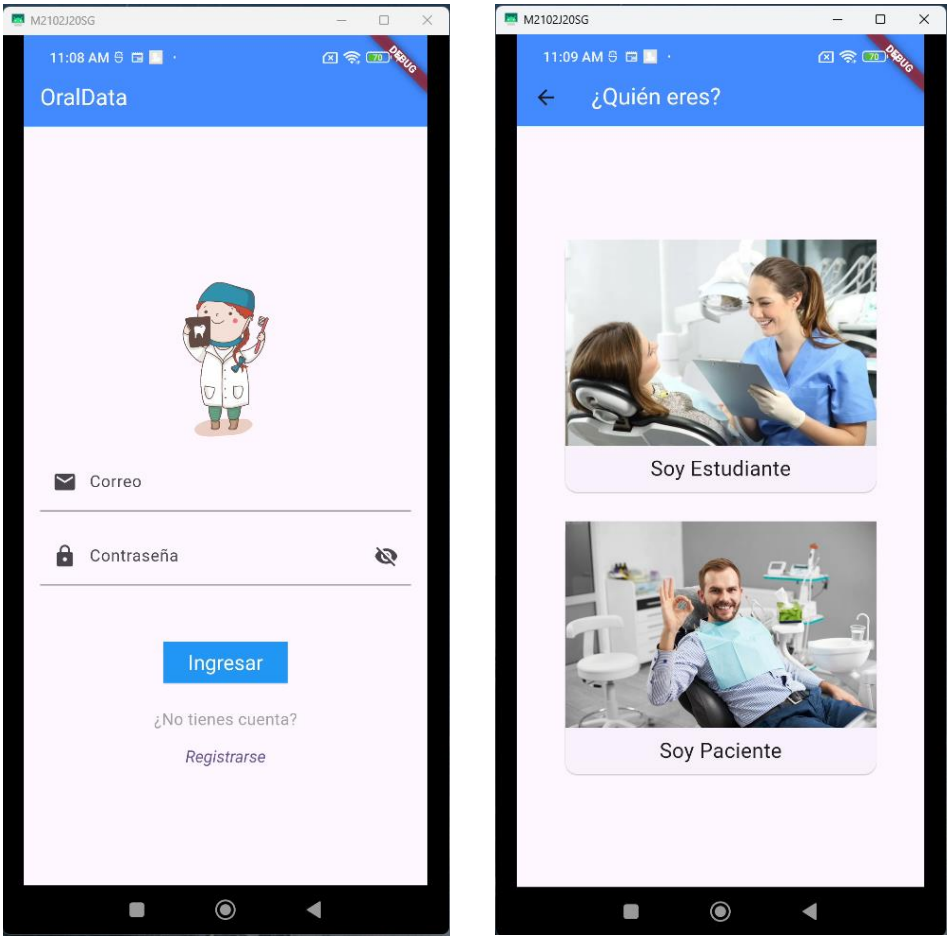


Ilustración 16. Interfaz de usuario anónimo

Pantalla inicial de la aplicación cuando un usuario no se ha registrado y cuando este le da en registrarse existe la posibilidad de identificarse con el rol con el que va a interactuar.

Registro de Paciente: Captura de pantalla de la pantalla de registro para pacientes

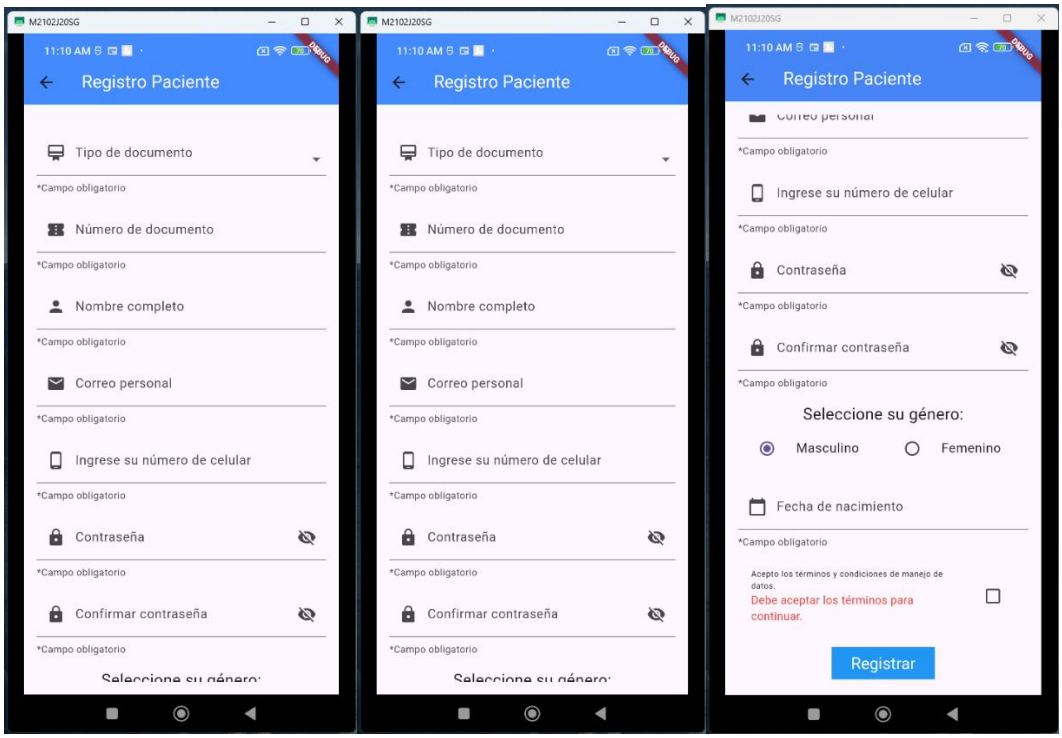


Ilustración 17. Formulario de registro en calidad de paciente.

Estas interfaces son las que representan la metodología de registro en calidad de paciente, en estas se pide información referente a la necesidad que se tiene en la disciplina de atención de odontología.

Registro de Estudiante: Captura de pantalla de la pantalla de registro para estudiantes.

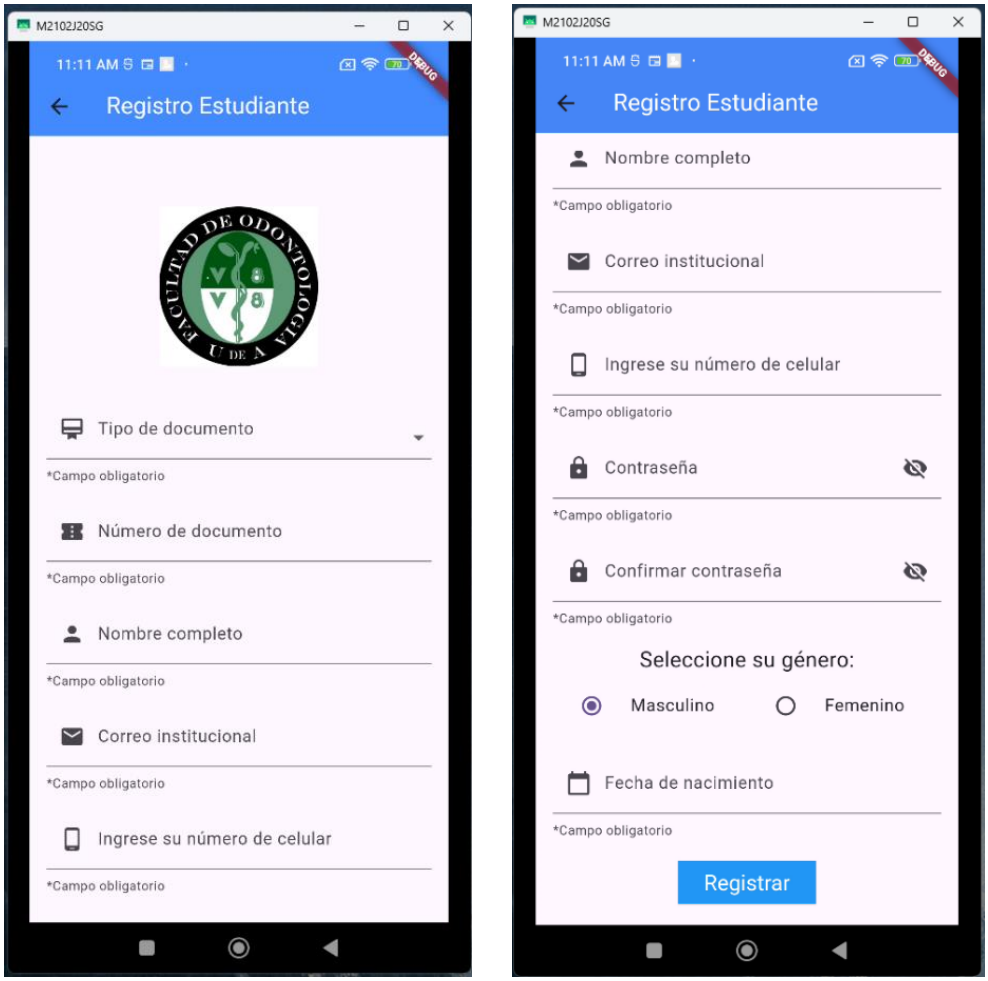


Ilustración 18. Formulario registro estudiantes

Esta interfaz, es la que relaciona el registro a nivel de formulario para los usuarios que se encuentran en calidad de estudiante, este registro se mantiene principalmente cuando el dominio de correo es institucional.

Proceso de agendamiento: Capturas de pantalla del proceso de agendamiento desde la gestión de disponibilidad horaria rol estudiante

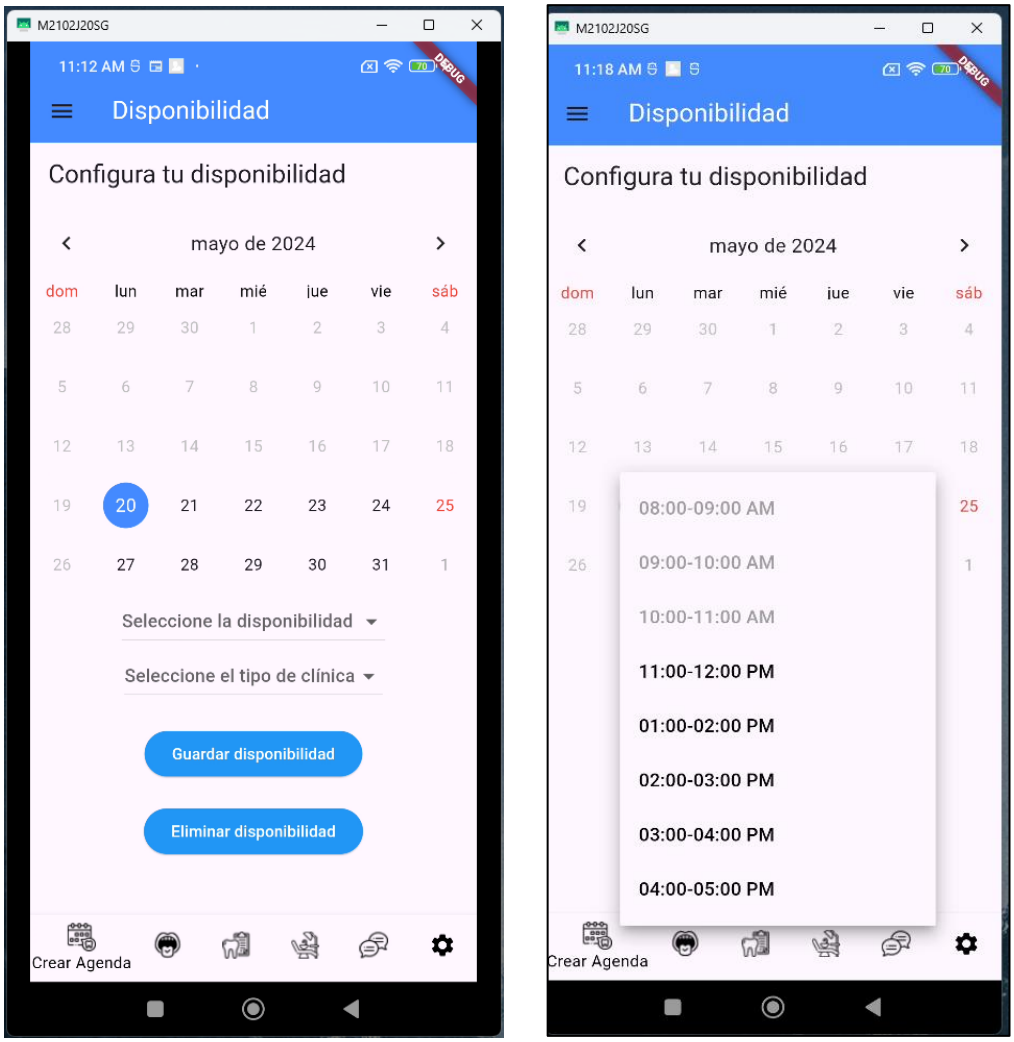


Ilustración 19. Agendamiento de citas odontológicas

Aquí el estudiante tiene la posibilidad de parametrizar su horario y días disponibles para las practicas clínicas, el sistema solo permite crear agendas a partir del día actual y hasta máximo un mes.

Detalles usuario registrado como paciente

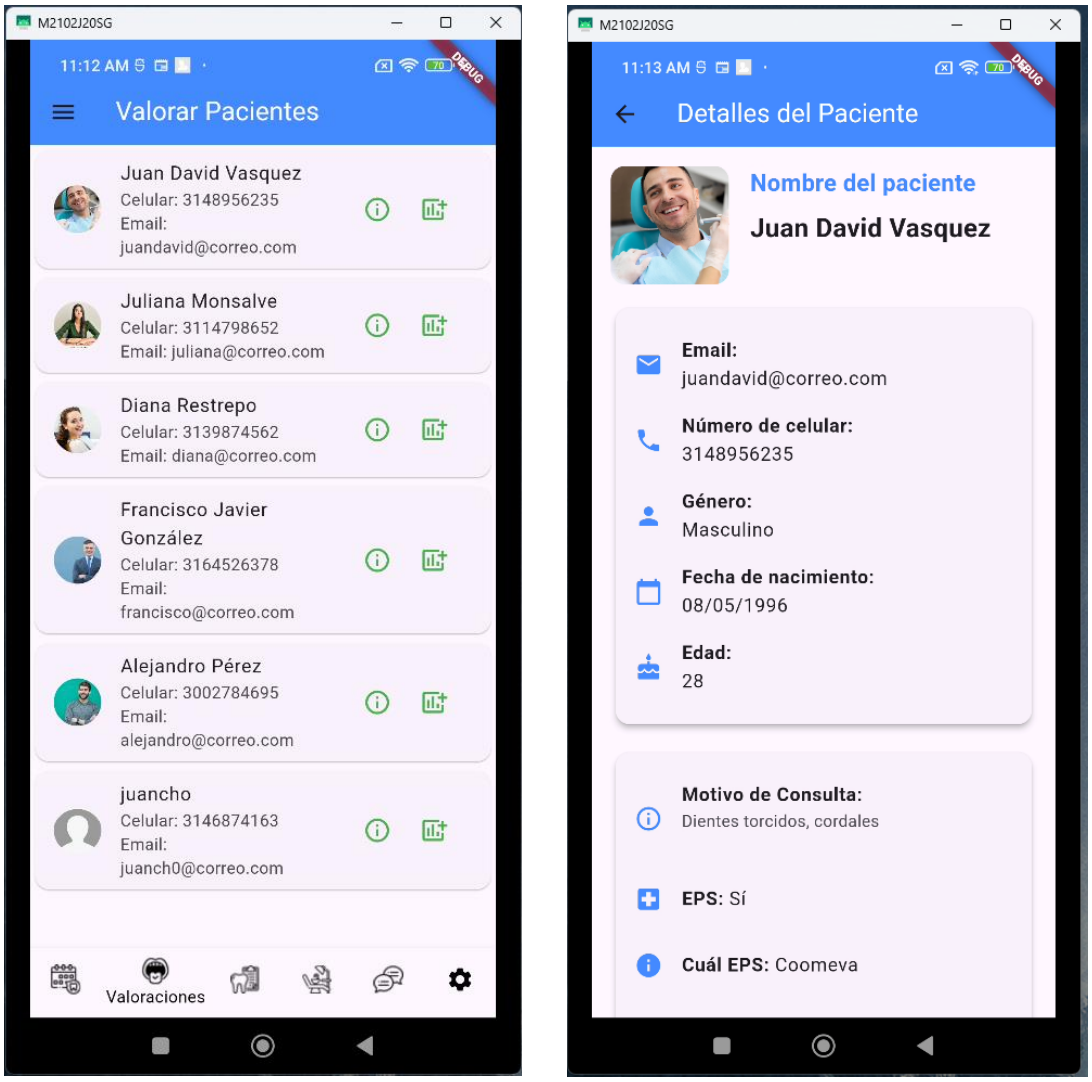


Ilustración 20. Detalles pacientes

Una vez parametrizada la disponibiilidad el estudiante podra visualizar el listado de pacientes que se han registrado en la aplicación, donde al dar clic en el icono de informacion, podrá valorar si el paciente es idoneo para sus prácticas clínicas.

Metodologia de valoracion de pacientes

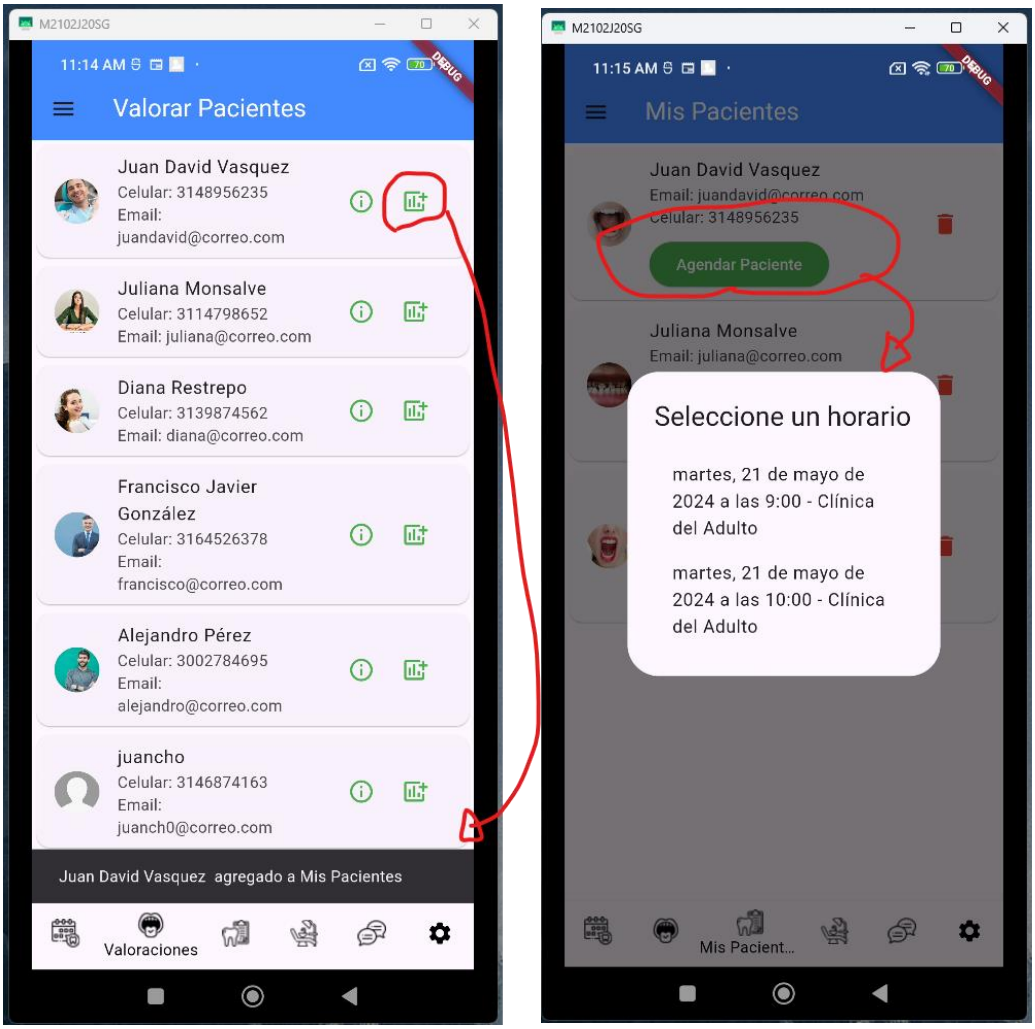


Ilustración 21. Valoración de pacientes

En caso de ser un paciente idóneo para su atención con el icono de agregar podrá agregar el paciente a su listado de pacientes, donde posteriormente podrá agendar a este.

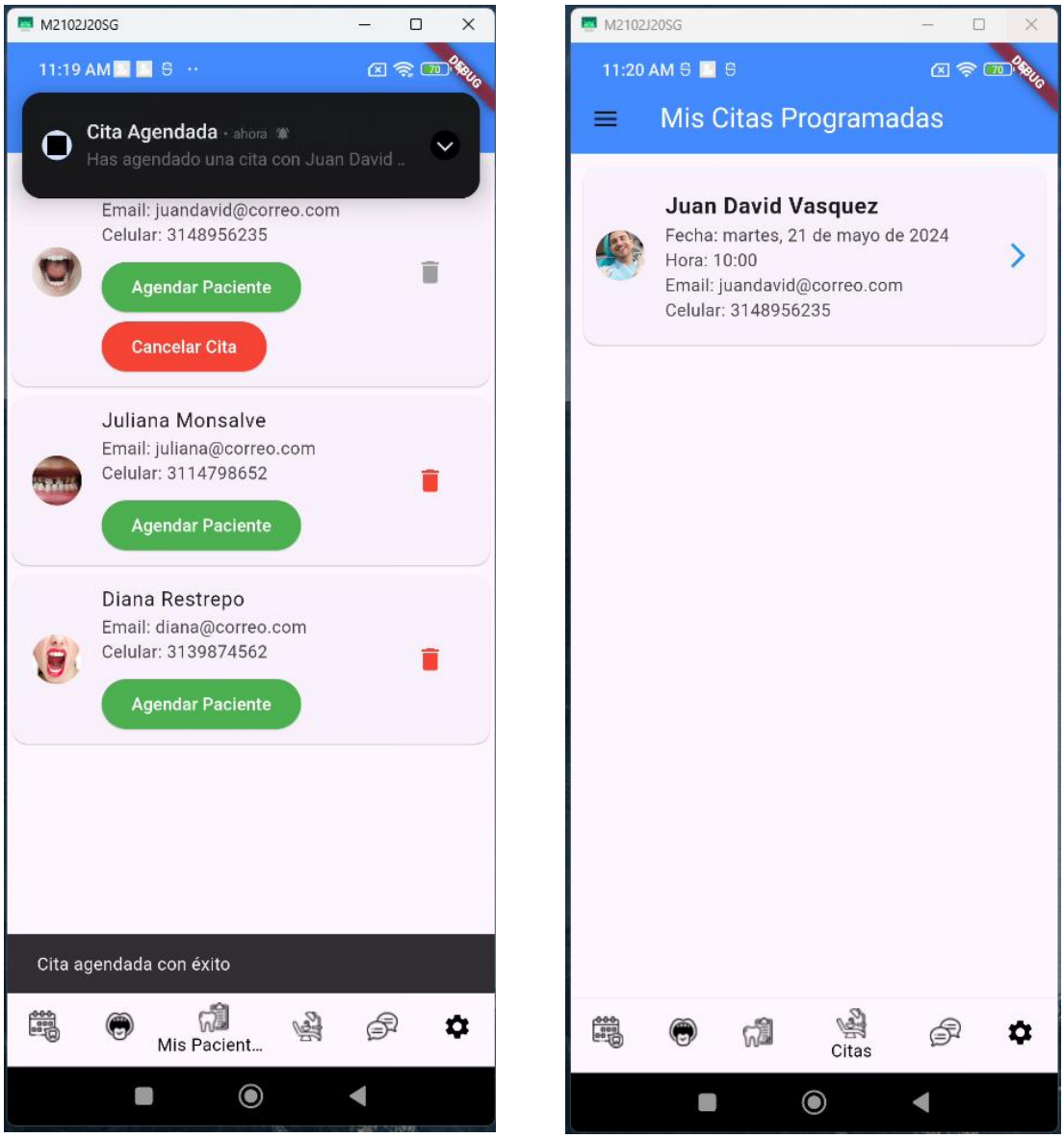


Ilustración 22. Notificación de agendamiento de cita

Una vez agendada la cita del paciente en las horas elegidas para disponibilidad anteriormente parametrizadas, llegará un mensaje de confirmación y ya en el menú cita podrá tener información de recordatorio de sus próximas citas.

Detalles de la cita agendada

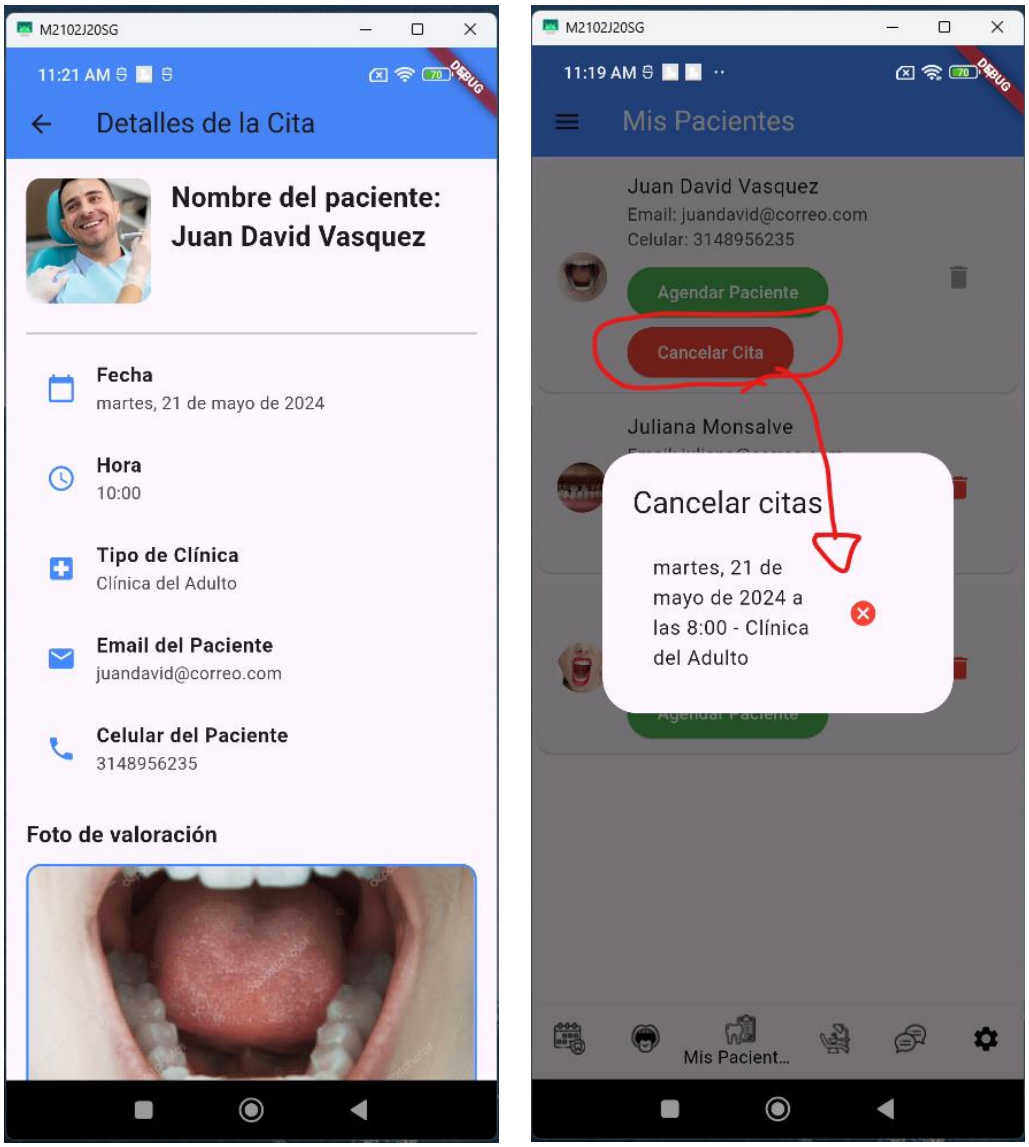


Ilustración 23. Detalles de la cita agendada

Al dar tap en la card del paciente podrá visualizar la información de la cita agendada, a su vez el estudiante puede cancelar la cita si así lo desea.

Chat: Captura de pantalla de la interfaz de chat.

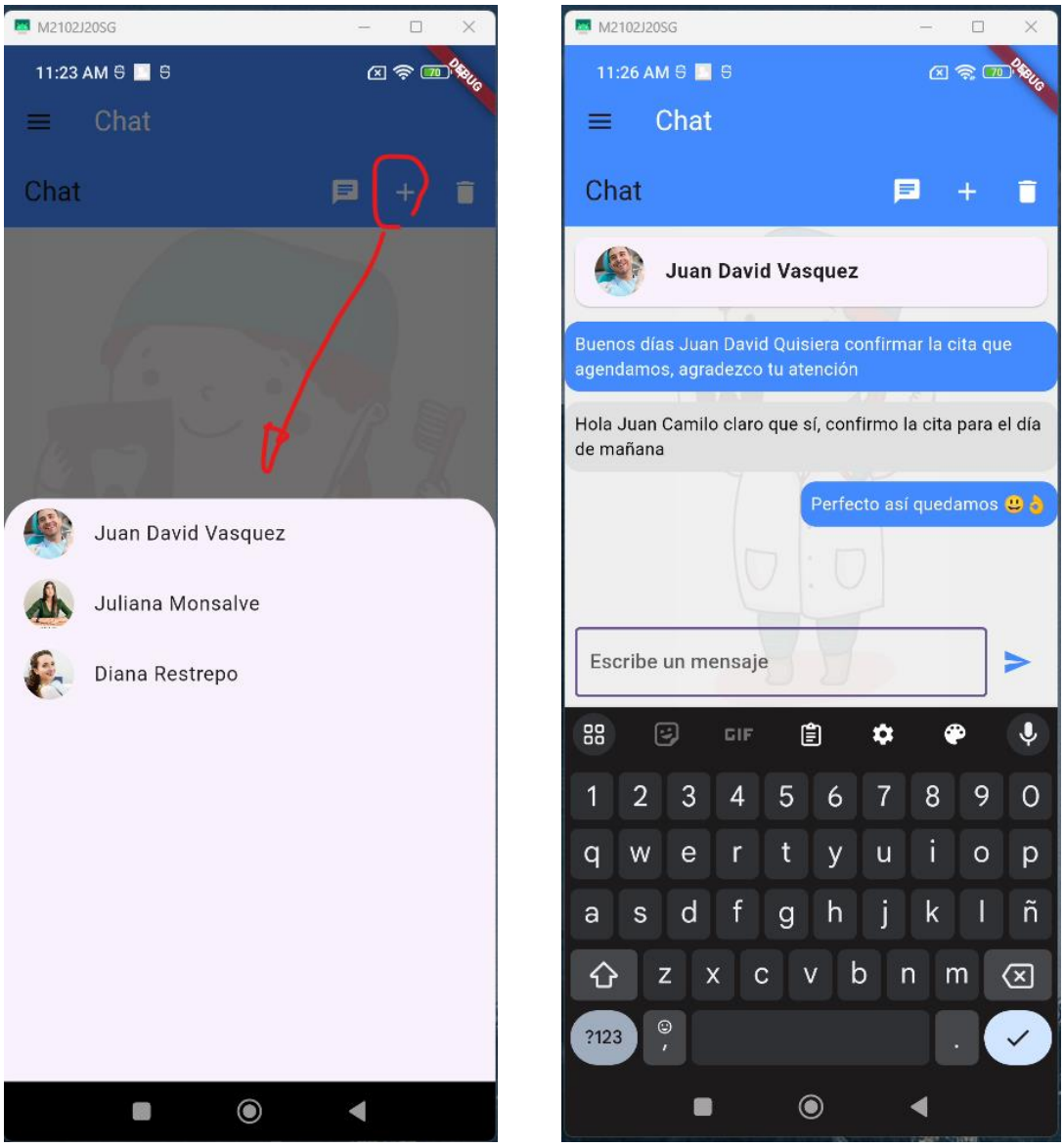


Ilustración 24. Chata de oraldata entre paciente y estudiante

Una vez el paciente se encuentre agendado a una cita, el sistema valida y permite que el estudiante pueda chatear con este para ultimar detalles de la cita y reconfirmar.

Editar Perfil: Captura de pantalla de la interfaz de editar perfil.

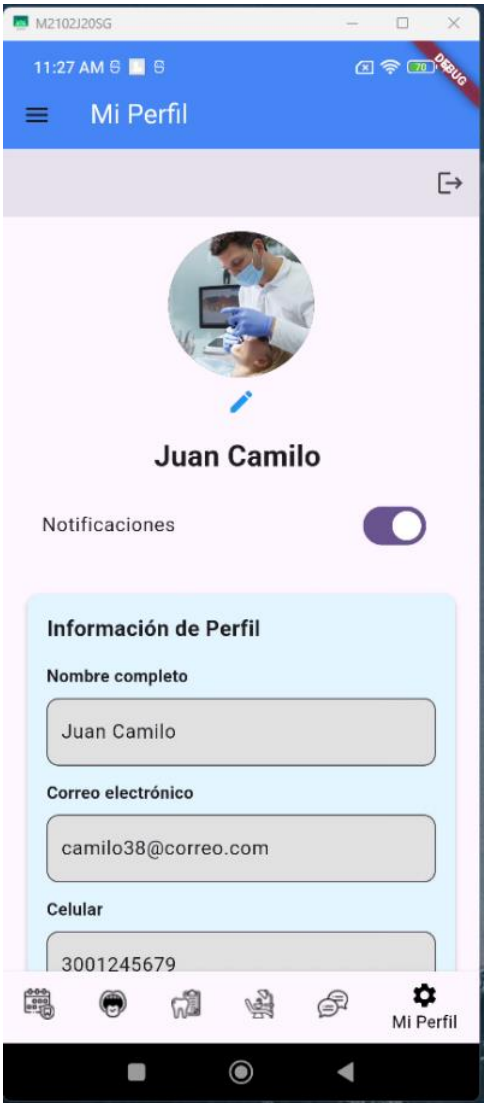


Ilustración 25. Editar perfil

Tanto pacientes como estudiantes pueden editar la información de su perfil, como nombre, celular y cambiar su contraseña si así lo desean, el campo correo al ser usuario no está disponible para ser editado.

7.6.5 Roles funcionales en Oral Data

Rol Administrador

- **Rol Administrador:** Captura de interfaz del sistema como rol administrador

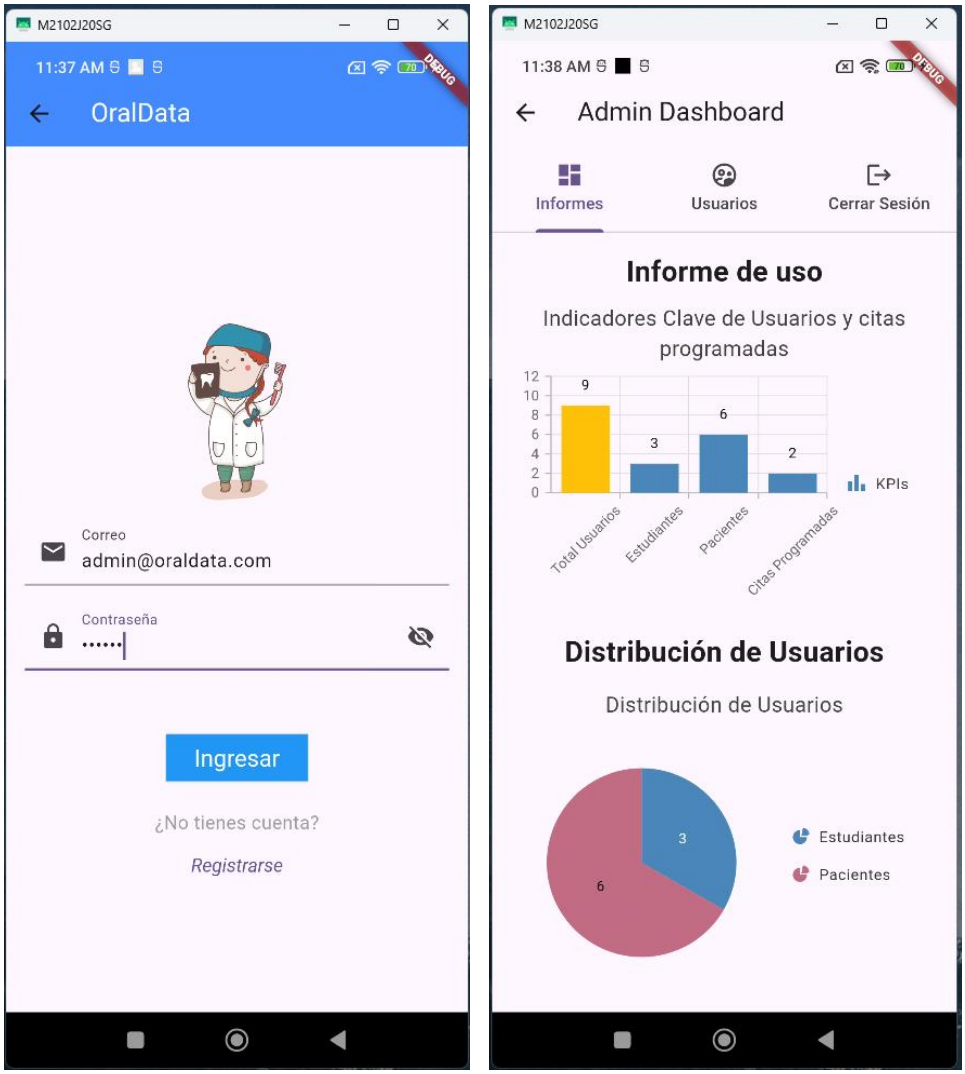


Ilustración 26. Rol de administrador de Oral Data

El administrador de la aplicación tiene acceso a información detallada sobre pacientes y estudiantes registrados, esta información se carga en tiempo real.

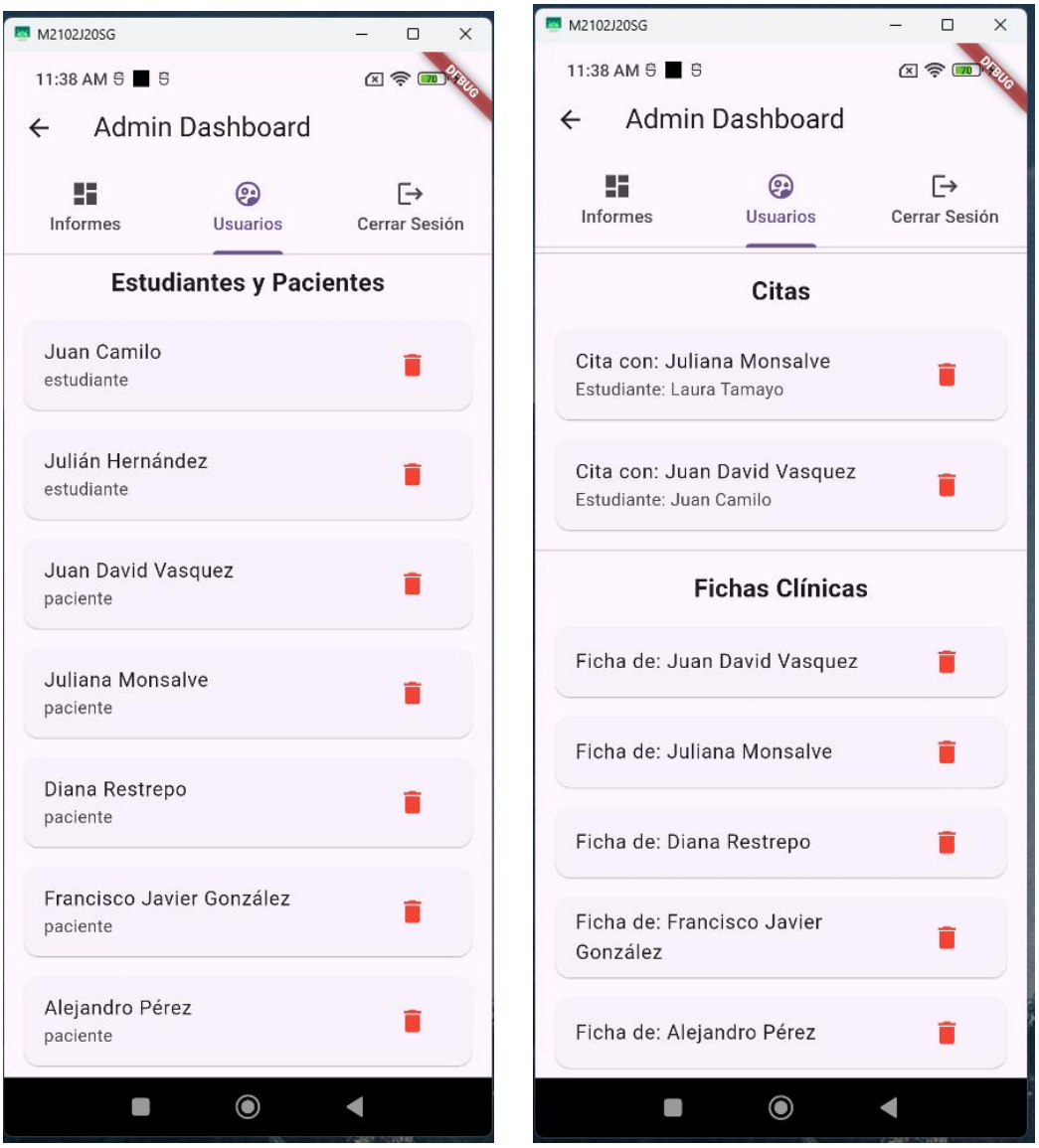


Ilustración 27. Administración de fichas clínicas y usuarios

El administrador también cuenta con un crud donde puede eliminar usuarios, citas y fichas clínicas (formulario de caracterización).

Rol Paciente

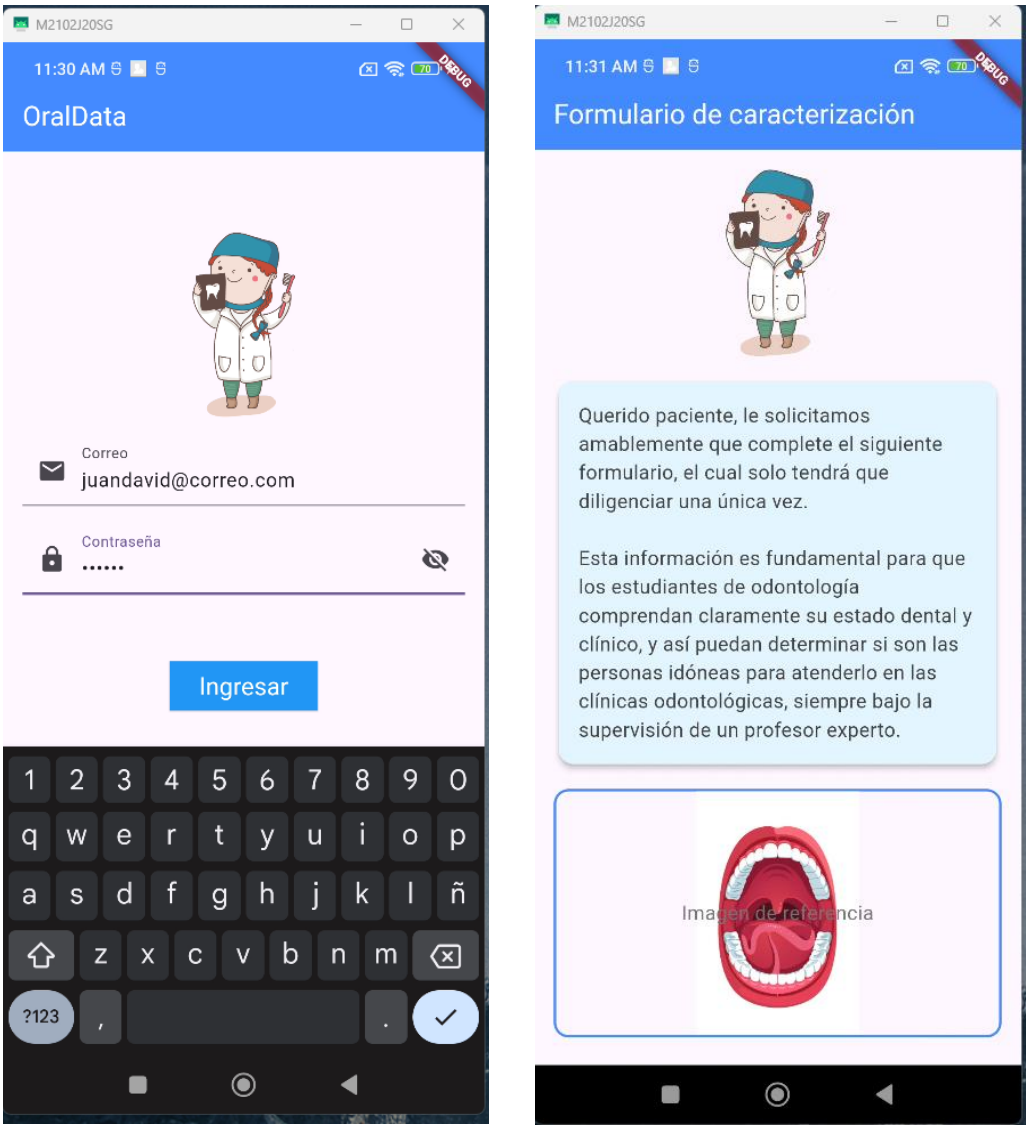


Ilustración 28. Rol paciente

El paciente una vez registrado en el sistema, deberá llenar un formulario de caracterización el cual lo hará una única vez y ayudará a que los estudiantes sepan si son las personas idóneas para atenderlos.

Sistematización de información odontológica expuesta en el perfil de paciente

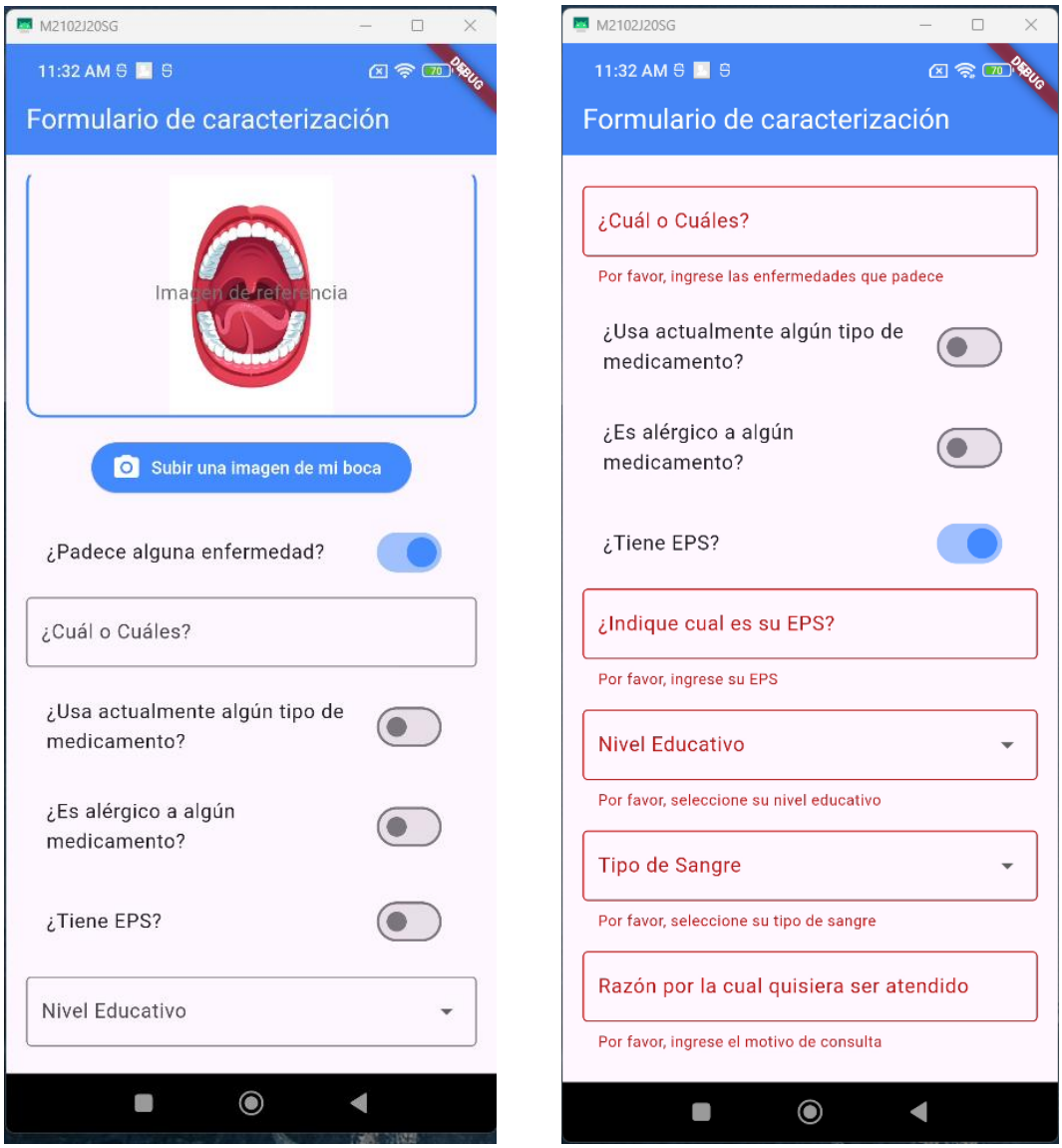


Ilustración 29. Formulario de consulta odontológica

Los paciente como primera tarea luego del registro, deben de sistematizar una infomacion de estado odontolgico, con el fin de que al momento de consignar esta informacion, sea claro que tipo de situacion dental se tiene como paciente.

Detalle citas paciente

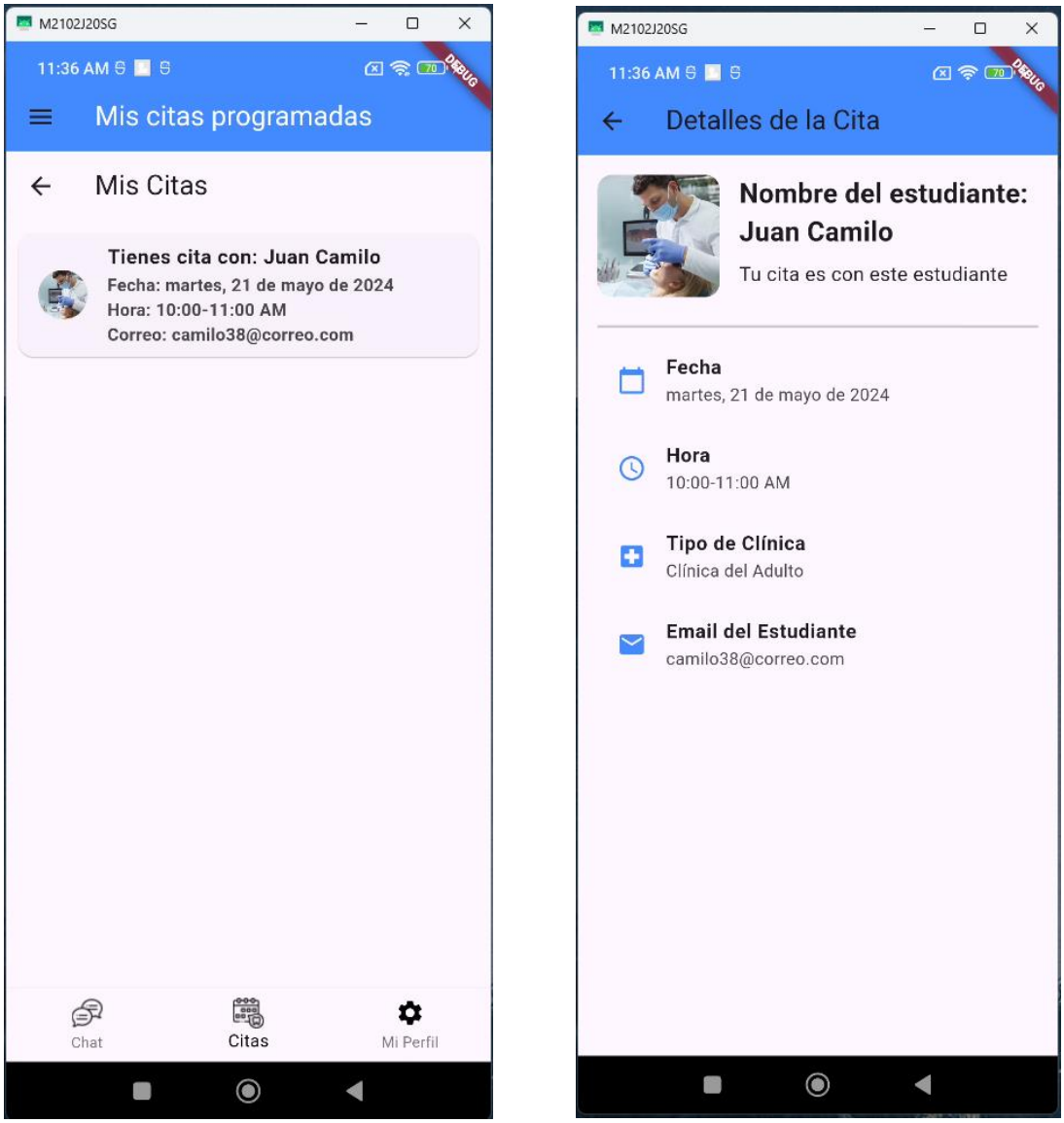


Ilustración 30. Detalle de citas pacientes

El paciente tambien puede visualizar la informacion detallada de sus proximas citas

Panel de información y acceso rápido

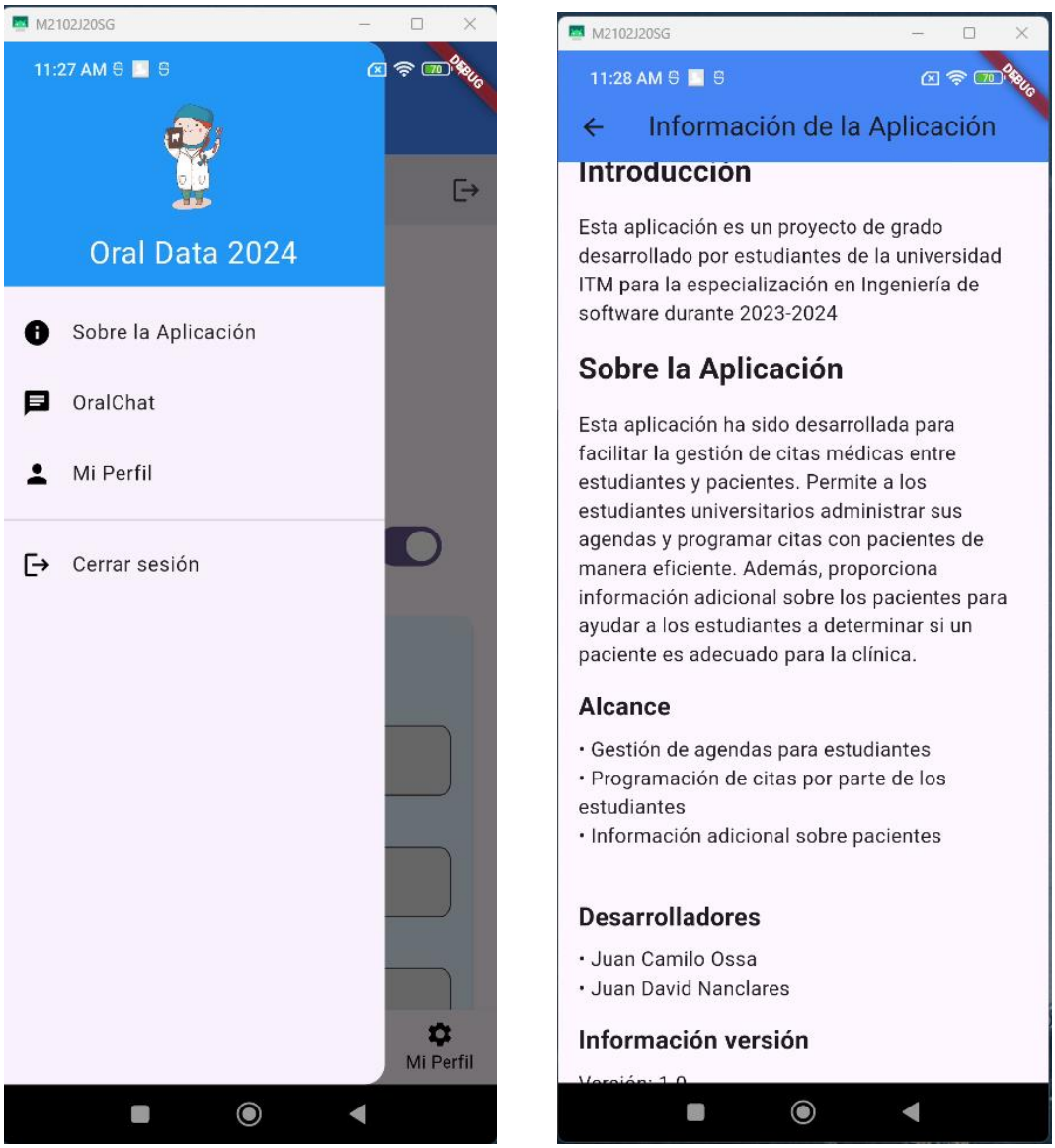


Ilustración 31. Panel de acceso rápido perfil de paciente

Ambos perfiles tienen la posibilidad de desplegar un menú de atajos donde además se cuenta con la información sobre la aplicación y su alcance actual.

7.6.6 Descripción de Archivos

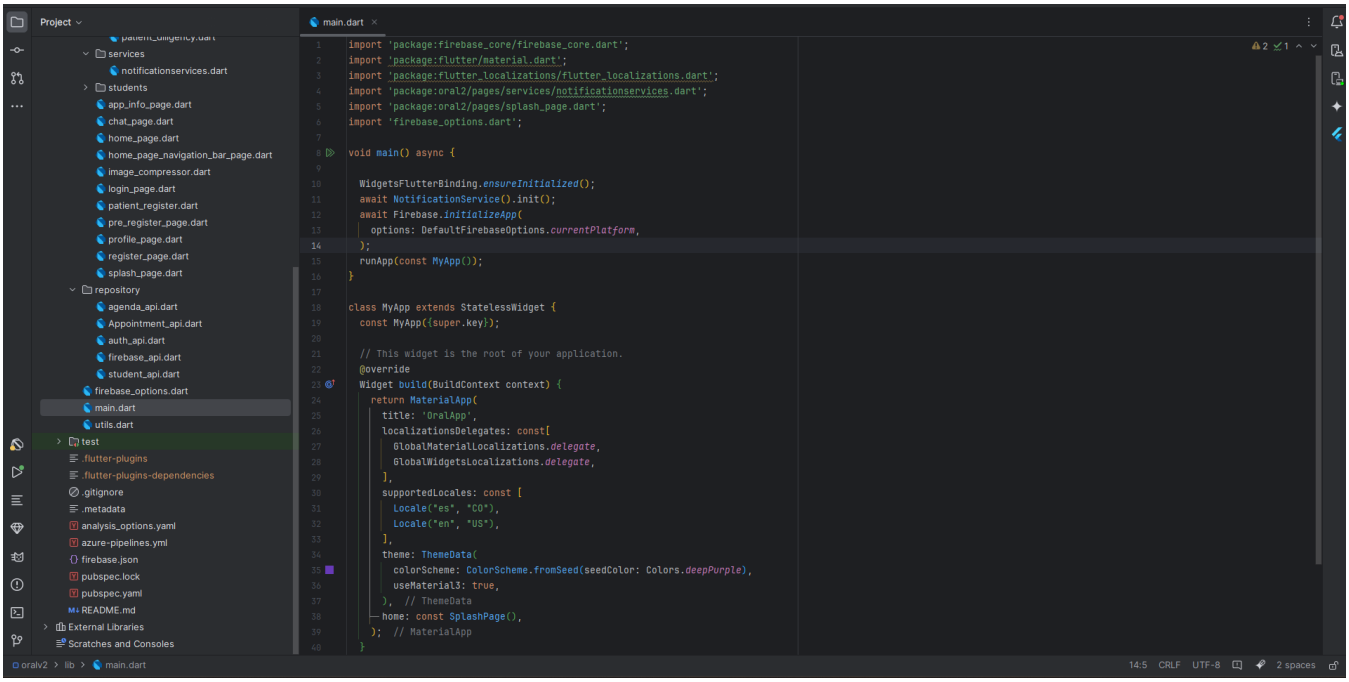


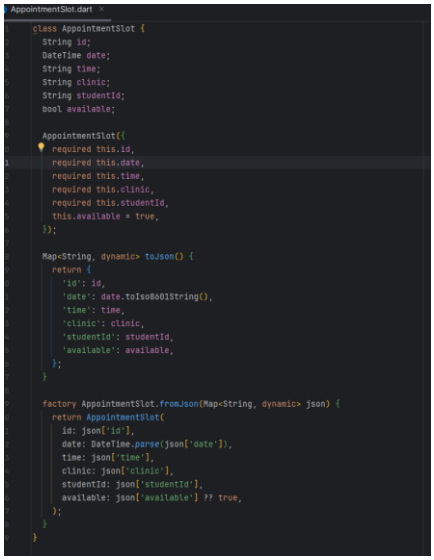
Ilustración 32. Menu de visualización de Oral Data en su código fuente

Aquí se entrega una vista general de la distribución del desarrollo de OralData realizado en Android studio

7.6.7 Carpeta Models

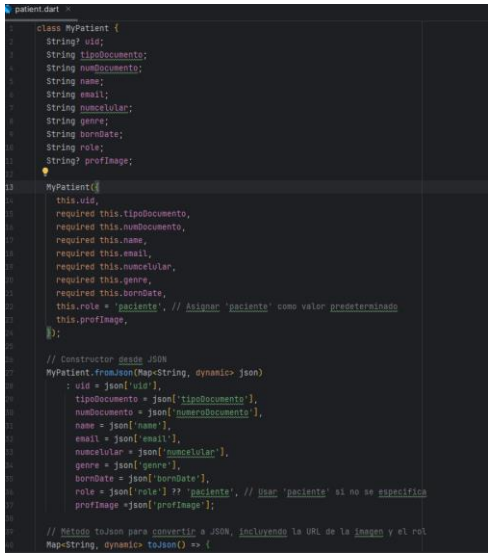
AppointmentSlot.dart

El archivo `AppointmentSlot.dart` define la clase `AppointmentSlot`, que representa un intervalo de tiempo disponible para citas médicas. Esta clase incluye atributos como `date`, `timeSlot`, y `available` para gestionar la disponibilidad de los slots.



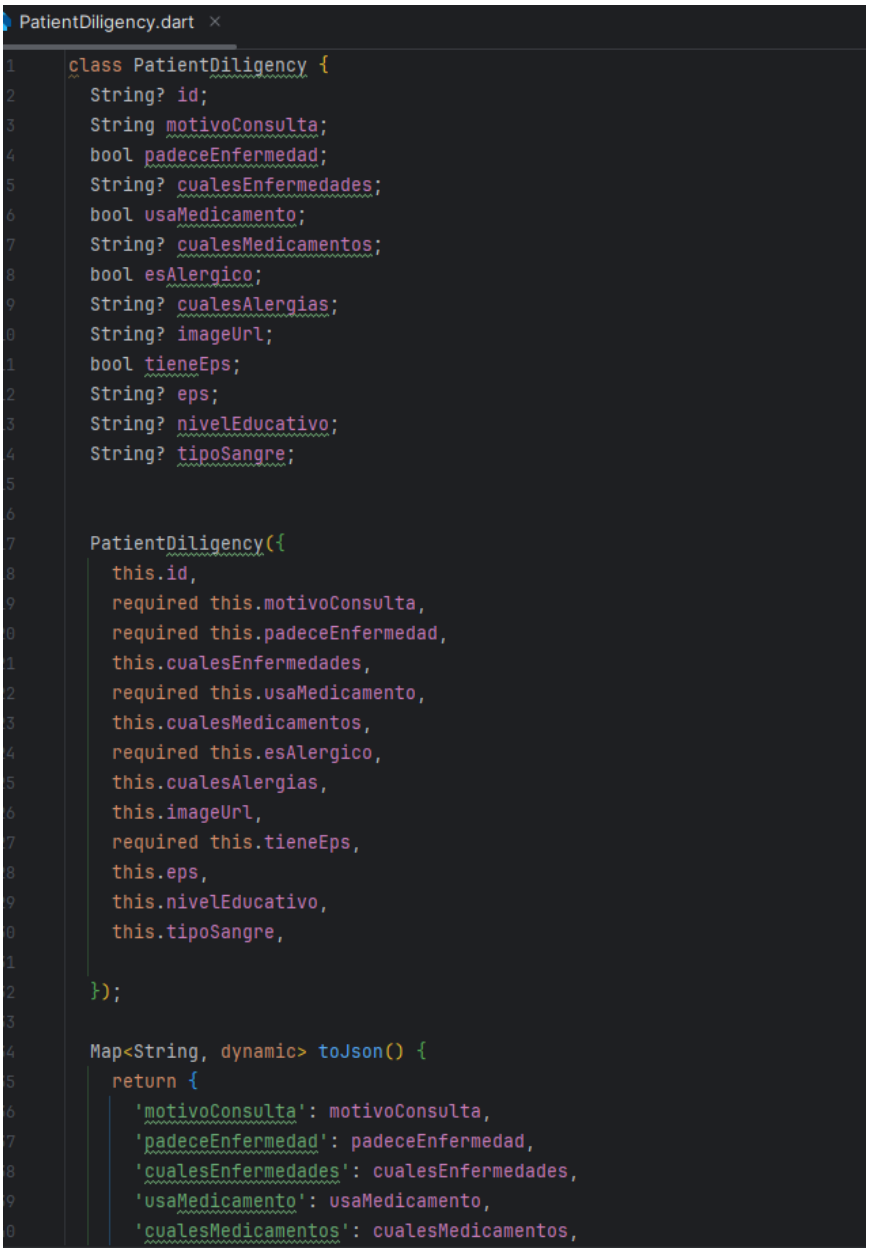
patient.dart

El archivo `patient.dart` define la clase `Patient`, que almacena la información del paciente, incluyendo datos personales y detalles médicos. Incluye métodos para la serialización y deserialización de objetos `Patient` desde y hacia Firebase.



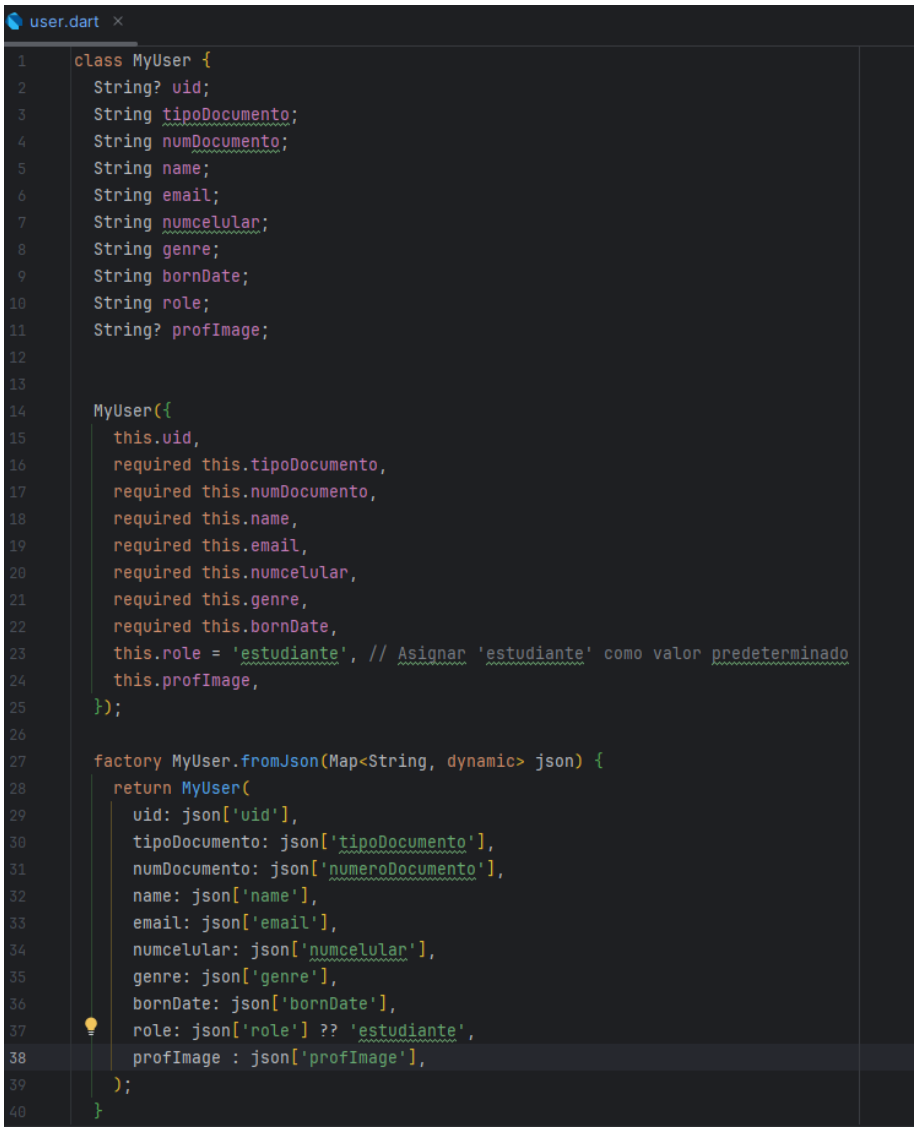
PatientDiligency.dart

El archivo `PatientDiligency.dart` define la clase `PatientDiligency`, que representa la diligencia o evaluación de la paciente realizada por un estudiante de odontología. Esta clase incluye atributos como `motivoConsulta`, `padeceEnfermedad`, `cualesEnfermedades`, y otros detalles clínicos.



user.dart

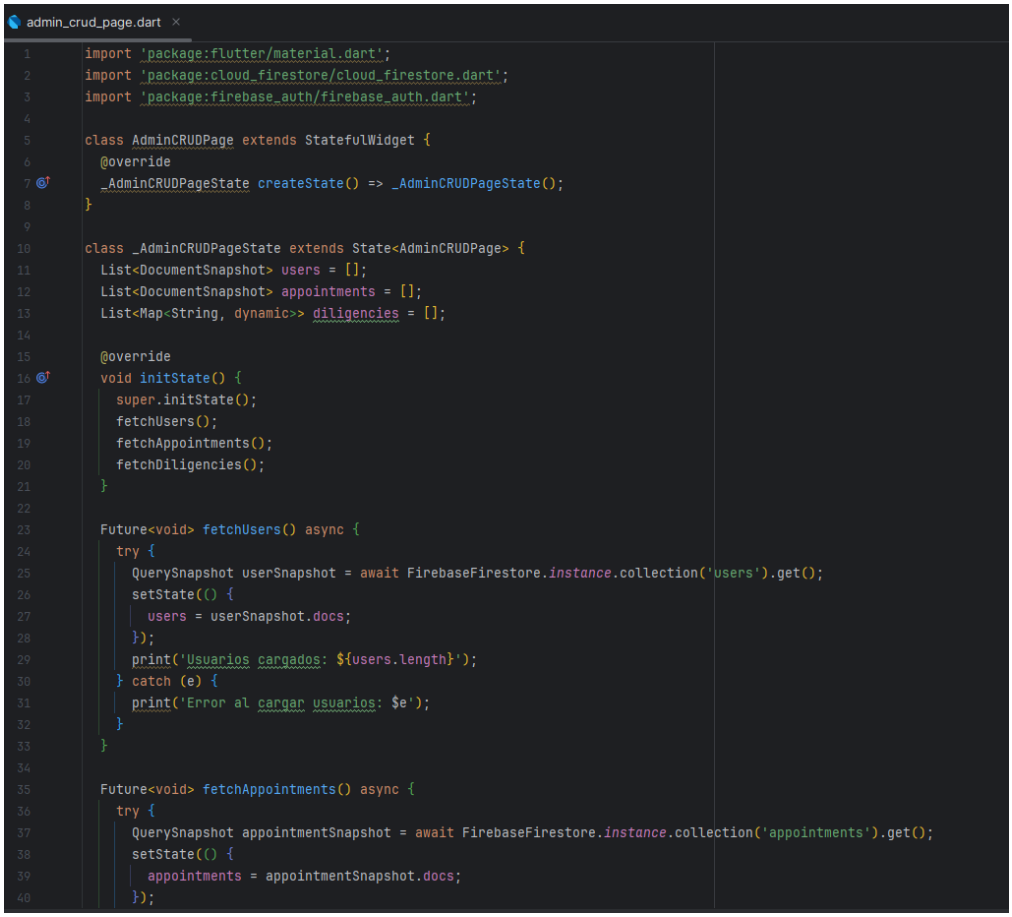
El archivo `user.dart` define la clase `MyUser`, que representa a los usuarios del sistema, ya sean pacientes o estudiantes. Incluye atributos como `uid`, `tipoDocumento`, `numDocumento`, `name`, `email`, `numcelular`, `genre`, `bornDate`, `role`, y `profImage`.



7.6.8 Carpeta Admin

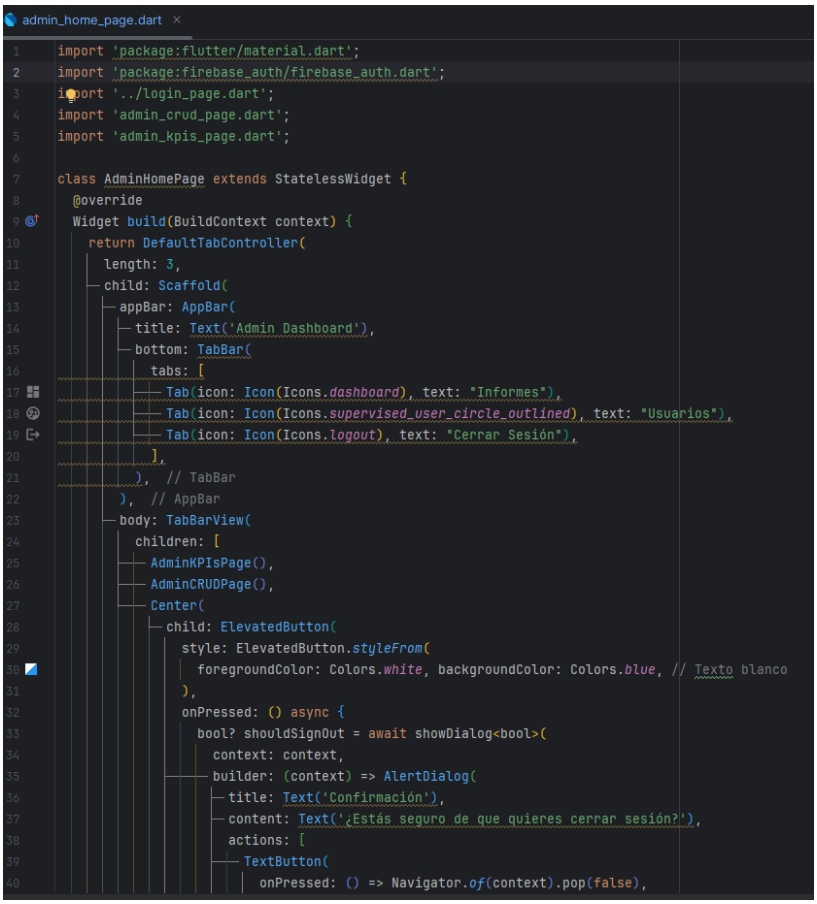
admin_crud_page.dart

El archivo `admin_crud_page.dart` define la clase `AdminCRUDPage`, que proporciona una interfaz de usuario para que el administrador pueda eliminar usuarios, citas o fichas clínicas del sistema. La clase incluye métodos para la construcción de la interfaz de usuario y la gestión de las operaciones CRUD.



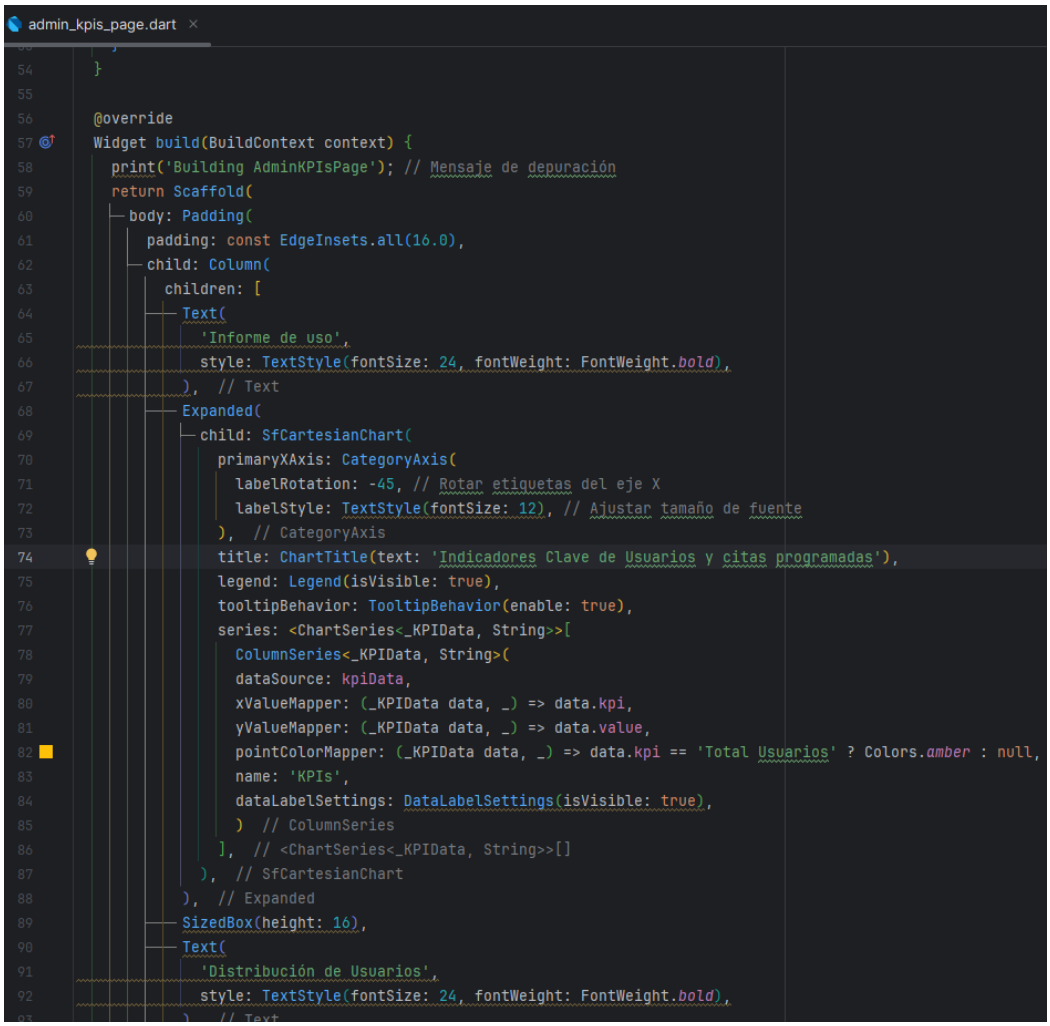
admin_home_page.dart

El archivo `admin_home_page.dart` define la clase `AdminHomePage`, que proporciona una interfaz de usuario para la página principal del administrador. La clase incluye métodos para la construcción de la interfaz de usuario y la navegación a diferentes secciones de la aplicación.



admin_kpis_page.dart

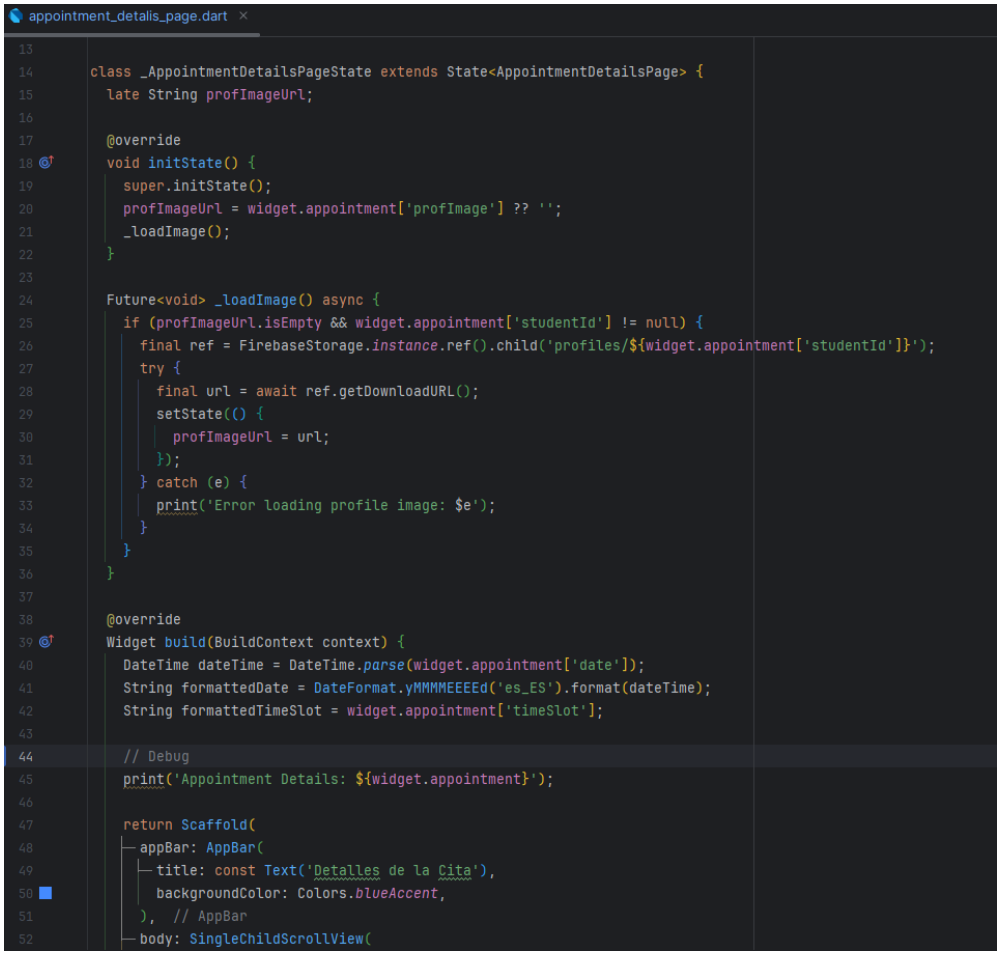
El archivo `admin_kpis_page.dart` define la clase `AdminKPIsPage`, que muestra los indicadores clave de rendimiento (KPIs) y el uso de la aplicación a lo largo del tiempo. La clase incluye métodos para la construcción de la interfaz de usuario y la visualización de los KPIs utilizando gráficos.



7.6.9 Carpeta Patients

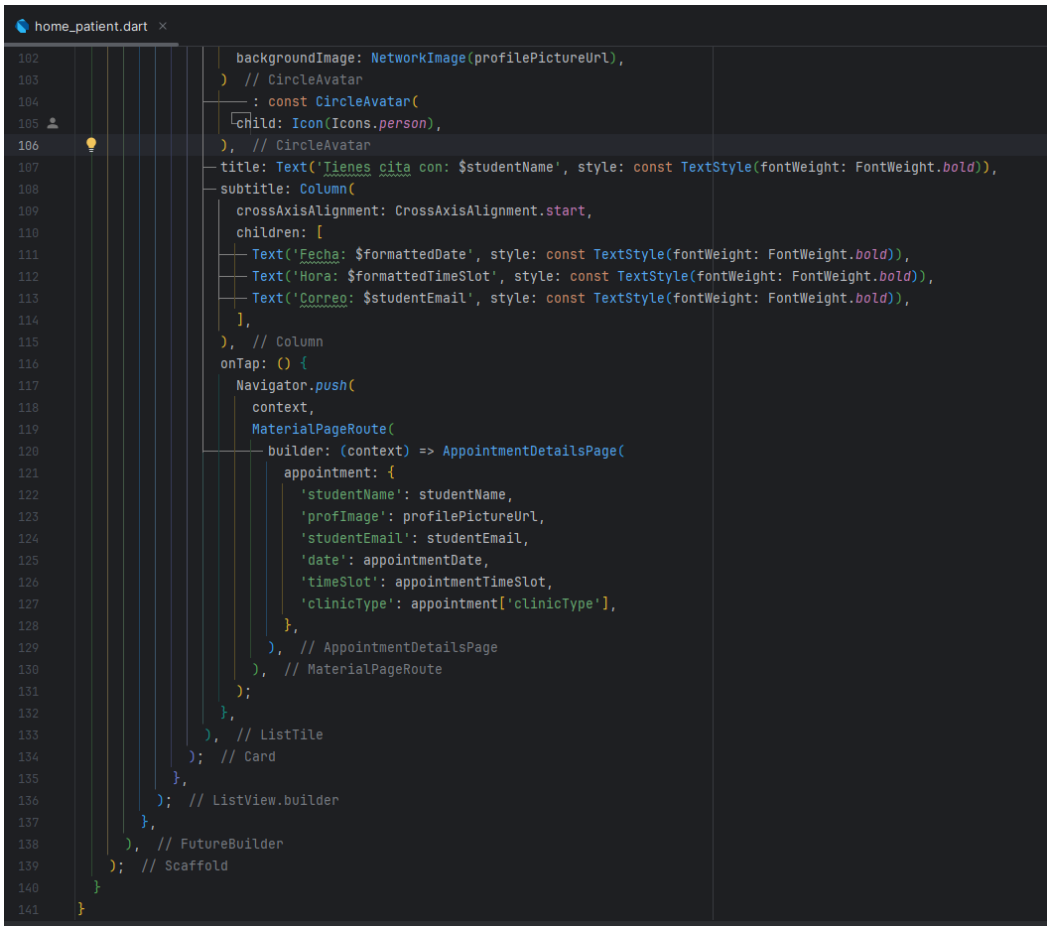
appointment_detalis_page.dart

El archivo `appointment_detalis_page.dart` define la clase `AppointmentDetailsPage`, que proporciona una interfaz de usuario para mostrar los detalles de una cita específica. La clase incluye métodos para la construcción de la interfaz de usuario y la visualización de la información de la cita.



home_patient.dart

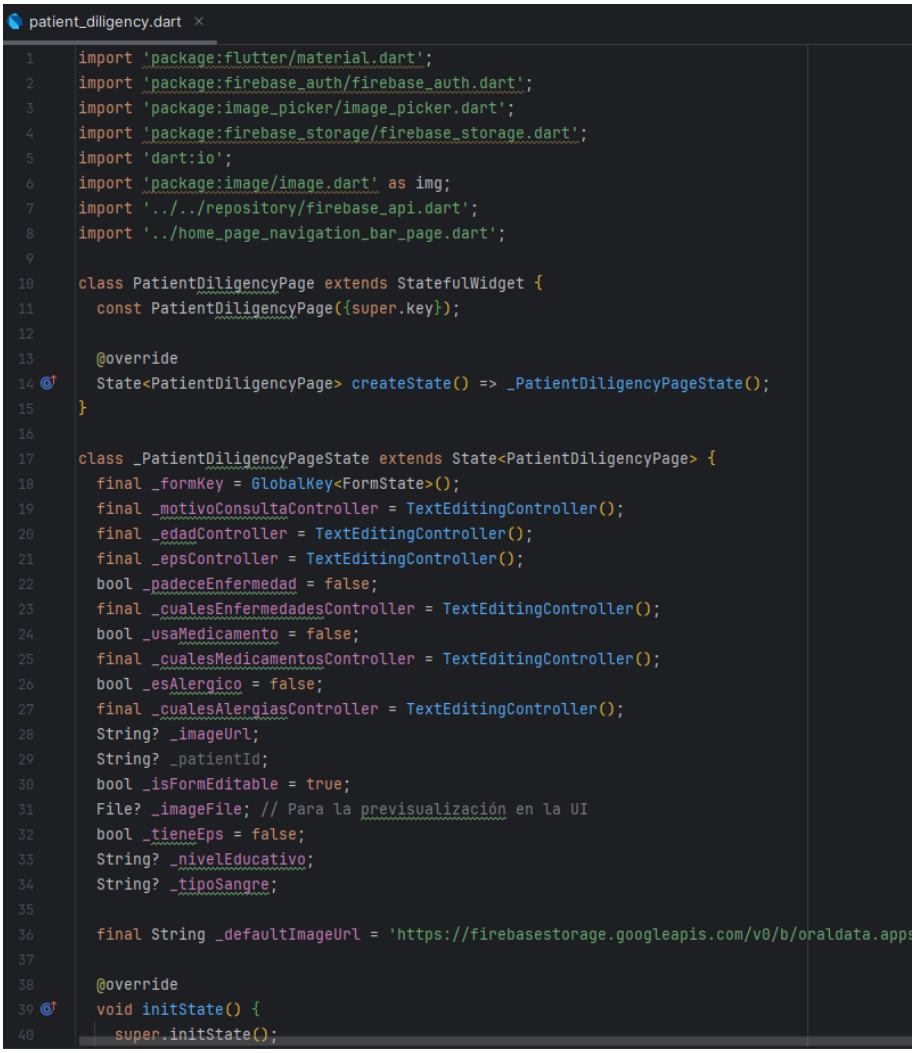
El archivo `home_patient.dart` define la clase `HomePatientPage`, que proporciona una interfaz de usuario para la página principal del paciente. La clase incluye métodos para la construcción de la interfaz de usuario y la navegación a diferentes secciones de la aplicación.



```
home_patient.dart
                backgroundImage: NetworkImage(profilePictureUrl),
              ) // CircleAvatar
            : const CircleAvatar(
              child: Icon(Icons.person),
              ), // CircleAvatar
            title: Text('Tienes cita con: $studentName', style: const TextStyle(fontWeight: FontWeight.bold)),
            subtitle: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text('Fecha: $formattedDate', style: const TextStyle(fontWeight: FontWeight.bold)),
                Text('Hora: $formattedTimeSlot', style: const TextStyle(fontWeight: FontWeight.bold)),
                Text('Correo: $studentEmail', style: const TextStyle(fontWeight: FontWeight.bold)),
              ],
            ), // Column
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (context) => AppointmentDetailsPage(
                    appointment: {
                      'studentName': studentName,
                      'profImage': profilePictureUrl,
                      'studentEmail': studentEmail,
                      'date': appointmentDate,
                      'timeSlot': appointmentTimeSlot,
                      'clinicType': appointment['clinicType'],
                    },
                  ), // AppointmentDetailsPage
                ), // MaterialPageRoute
              );
            },
          ), // ListTile
        ); // Card
      },
    ); // ListView.builder
  },
), // FutureBuilder
); // Scaffold
}
}
```

patient_diligency.dart

El archivo `patient_diligency.dart` define la clase `PatientDiligencyPage`, que proporciona una interfaz de usuario para la diligencia o evaluación que servirá como ficha clínica del paciente. La clase incluye métodos para la construcción de la interfaz de usuario y la gestión de la información de la diligencia del paciente.



7.6.10 Carpeta Services

notificationservices.dart

El archivo `notificationservices.dart` define la clase `NotificationService`, que gestiona las notificaciones locales y push en la aplicación. La clase incluye métodos para la inicialización de las notificaciones locales, mostrar notificaciones locales y enviar notificaciones push.

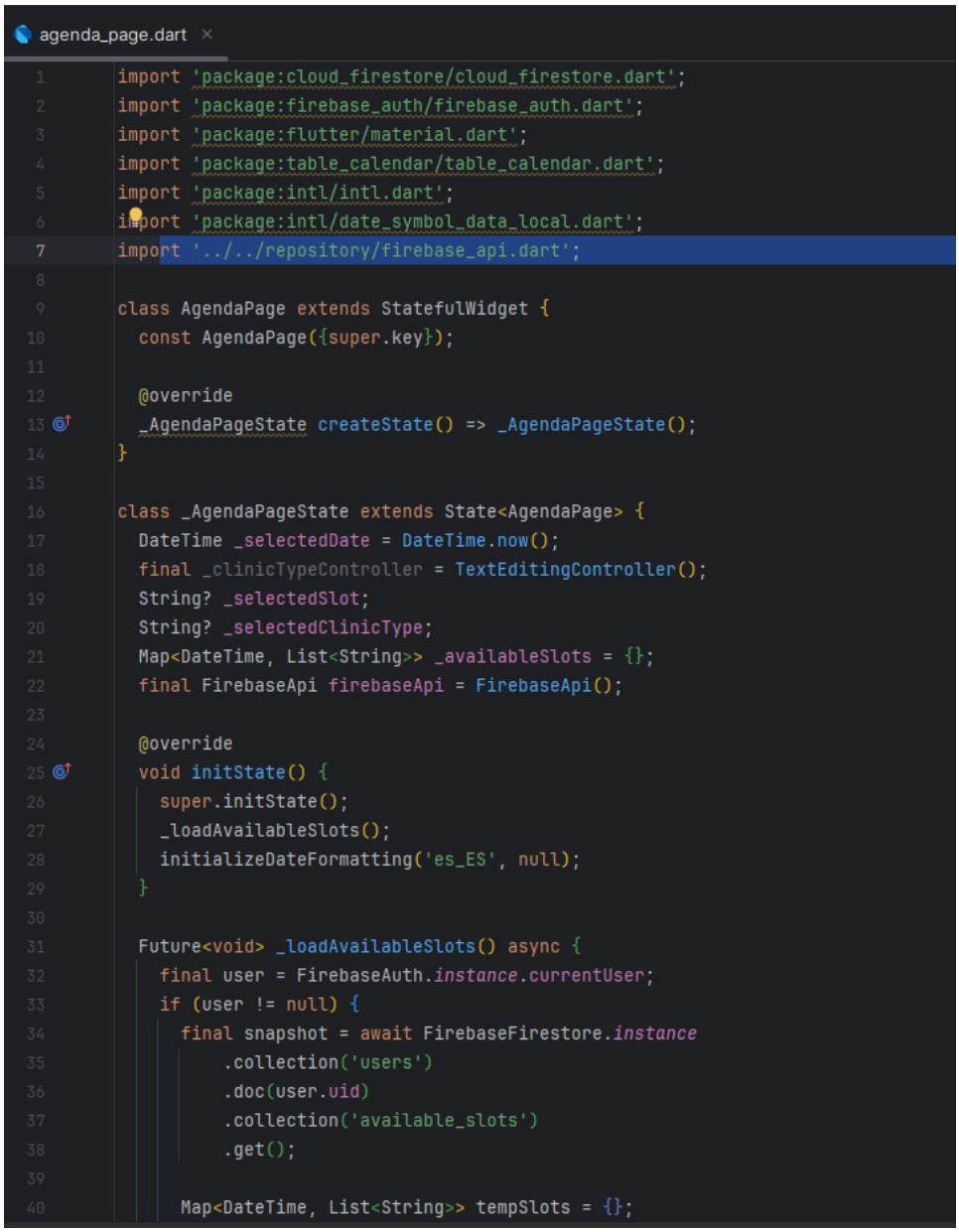
notificationservices.dart

```dart
import 'package:flutter_local_notifications/flutter_local_notifications.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

class NotificationService {
  final FlutterLocalNotificationsPlugin flutterLocalNotificationsPlugin =
  FlutterLocalNotificationsPlugin();

  Future<void> init() async {
    const AndroidInitializationSettings initializationSettingsAndroid =
    AndroidInitializationSettings('@mipmap/ic_launcher');

    final InitializationSettings initializationSettings =
    InitializationSettings(android: initializationSettingsAndroid);

    await flutterLocalNotificationsPlugin.initialize(initializationSettings);
  }

  Future<void> showNotification(int id, String title, String body) async {
    const AndroidNotificationDetails androidPlatformChannelSpecifics =
    AndroidNotificationDetails(
      'your_channel_id',
      'your_channel_name',
      importance: Importance.max,
      priority: Priority.high,
    );

    const NotificationDetails platformChannelSpecifics =
    NotificationDetails(android: androidPlatformChannelSpecifics);

    await flutterLocalNotificationsPlugin.show(
      id,
      title,
      body,
      platformChannelSpecifics,
      payload: 'item x',
    );
  }

  Future<void> sendPushNotification(String token, String title, String body) async {
```

7.6.11 Carpeta Student

agenda_page.dart

El archivo `agenda_page.dart` define la página que permite a los estudiantes configurar su disponibilidad para citas. Incluye funcionalidades para agregar y eliminar slots de tiempo disponibles.



```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:table_calendar/table_calendar.dart';
import 'package:intl/intl.dart';
import 'package:intl/date_symbol_data_local.dart';
import '../../repository/firebase_api.dart';

class AgendaPage extends StatefulWidget {
  const AgendaPage({super.key});

  @override
  _AgendaPageState createState() => _AgendaPageState();
}

class _AgendaPageState extends State<AgendaPage> {
  DateTime _selectedDate = DateTime.now();
  final _clinicTypeController = TextEditingController();
  String? _selectedSlot;
  String? _selectedClinicType;
  Map<DateTime, List<String>> _availableSlots = {};
  final FirebaseApi firebaseApi = FirebaseApi();

  @override
  void initState() {
    super.initState();
    _loadAvailableSlots();
    initializeDateFormatting('es_ES', null);
  }

  Future<void> _loadAvailableSlots() async {
    final user = FirebaseAuth.instance.currentUser;
    if (user != null) {
      final snapshot = await FirebaseFirestore.instance
          .collection('users')
          .doc(user.uid)
          .collection('available_slots')
          .get();

      Map<DateTime, List<String>> tempSlots = {};
```

appointment_detalis_page.dart

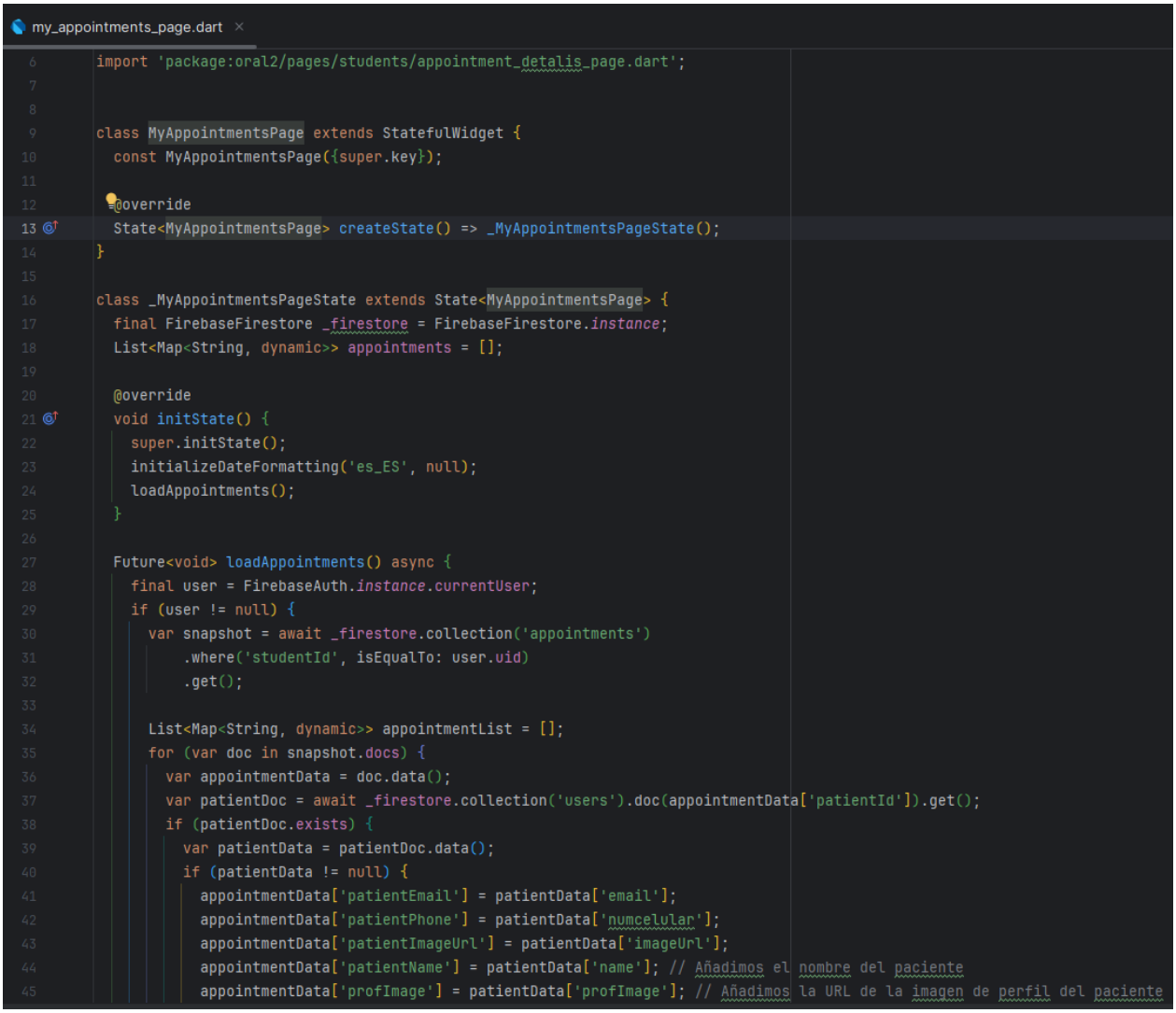
El archivo `appointment_detalis_page.dart` define la página de detalles de la cita. Muestra información detallada sobre una cita específica, incluyendo el paciente y el estudiante involucrados.

appointment_detalis_page.dart

```dart
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
import 'package:firebase_storage/firebase_storage.dart';

class AppointmentDetailsPage extends StatefulWidget {
  final Map<String, dynamic>? appointment; // Hacer nullable

  const AppointmentDetailsPage({super.key, required this.appointment});

  @override
  _AppointmentDetailsPageState createState() => _AppointmentDetailsPageState();
}

class _AppointmentDetailsPageState extends State<AppointmentDetailsPage> {
  late String profImageUrl;
  late String patientImageUrl;

  @override
  void initState() {
    super.initState();
    if (widget.appointment == null) {
      // Manejar el caso donde la cita es null
      Navigator.of(context).pop();
      return;
    }

    profImageUrl = widget.appointment!['profImage'] ?? '';
    patientImageUrl = widget.appointment!['patientImageUrl'] ?? '';

    _loadImages();
  }

  Future<void> _loadImages() async {
    if (profImageUrl.isEmpty && widget.appointment!['studentId'] != null) {
      final ref = FirebaseStorage.instance.ref().child('profiles/${widget.appointment!['studentId']}');
      try {
        final url = await ref.getDownloadURL();
        print('Profile image URL: $url'); // Imprimir la URL de la imagen de perfil
        setState(() {
          profImageUrl = url;
```

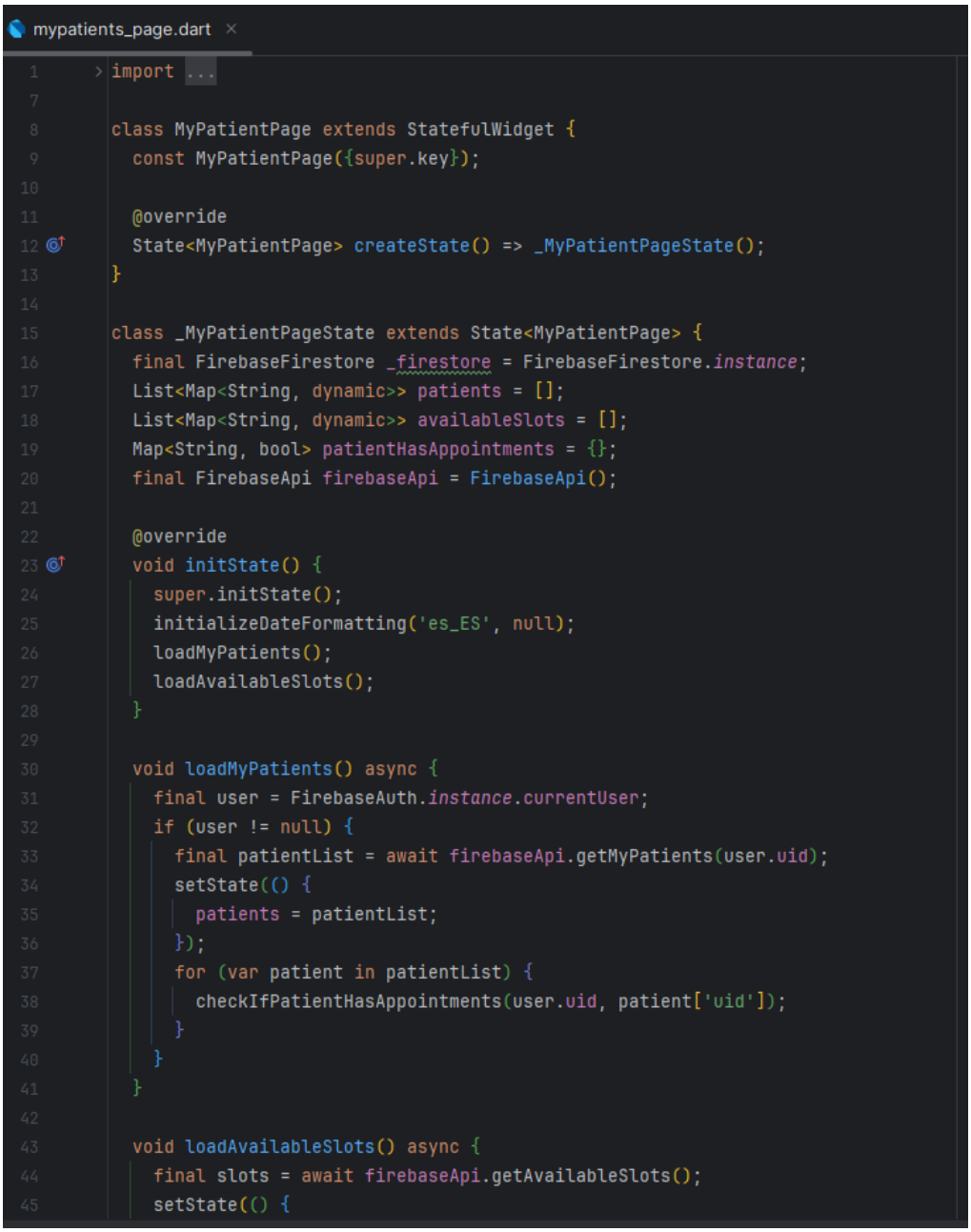
my_appointments_page.dart

El archivo `my_appointments_page.dart` define la página que muestra las citas programadas del usuario actual. Permite a los usuarios ver sus citas y acceder a los detalles de cada una.



mypatients_page.dart

El archivo `mypatients_page.dart` define la página que muestra la lista de pacientes del estudiante. Permite a los estudiantes ver y gestionar la información de sus pacientes.



```dart
import ...

class MyPatientPage extends StatefulWidget {
  const MyPatientPage({super.key});

  @override
  State<MyPatientPage> createState() => _MyPatientPageState();
}

class _MyPatientPageState extends State<MyPatientPage> {
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;
  List<Map<String, dynamic>> patients = [];
  List<Map<String, dynamic>> availableSlots = [];
  Map<String, bool> patientHasAppointments = {};
  final FirebaseApi firebaseApi = FirebaseApi();

  @override
  void initState() {
    super.initState();
    initializeDateFormatting('es_ES', null);
    loadMyPatients();
    loadAvailableSlots();
  }

  void loadMyPatients() async {
    final user = FirebaseAuth.instance.currentUser;
    if (user != null) {
      final patientList = await firebaseApi.getMyPatients(user.uid);
      setState(() {
        patients = patientList;
      });
      for (var patient in patientList) {
        checkIfPatientHasAppointments(user.uid, patient['uid']);
      }
    }
  }

  void loadAvailableSlots() async {
    final slots = await firebaseApi.getAvailableSlots();
    setState() {
```

patients_details_page.dart

El archivo `patients_details_page.dart` define la página que muestra los detalles de un paciente específico. Proporciona información detallada sobre el paciente y sus citas.

patients_details_Page.dart

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:intl/intl.dart'; // Asegúrate de importar intl para formatear fechas.

import '../../models/PatientDiligency.dart';

class PatientDetailsPage extends StatefulWidget {
  final String patientUid;

  const PatientDetailsPage({super.key, required this.patientUid});

  @override
  _PatientDetailsPageState createState() => _PatientDetailsPageState();
}

class _PatientDetailsPageState extends State<PatientDetailsPage> {
  late Future<Map<String, dynamic>> patientDetailsFuture;

  @override
  void initState() {
    super.initState();
    patientDetailsFuture = fetchPatientDetails(widget.patientUid);
  }

  Future<Map<String, dynamic>> fetchPatientDetails(String patientUid) async {
    final userSnapshot = await FirebaseFirestore.instance
        .collection('users')
        .doc(patientUid)
        .get();

    final diligencySnapshot = await FirebaseFirestore.instance
        .collection('diligencies')
        .doc(patientUid)
        .get();

    if (userSnapshot.exists && diligencySnapshot.exists) {
      final userData = userSnapshot.data() as Map<String, dynamic>?;
      final diligencyData = diligencySnapshot.data() as Map<String, dynamic>?;

      if (userData != null && diligencyData != null) {
```

home_student.dart

El archivo `home_student.dart` define la página principal para estudiantes. Permite a los estudiantes ver sus citas programadas y acceder a otras funcionalidades relevantes como visualizar el listado de pacientes que se han registrado.

home_student.dart

```dart
  State<StudentHomePage> createState() => _StudentHomePageState();
}

class _StudentHomePageState extends State<StudentHomePage> {
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;

  Future<List<Map<String, dynamic>>> getPacientes() async {
    try {
      QuerySnapshot querySnapshot = await _firestore
          .collection('users')
          .where('role', isEqualTo: 'paciente')
          .get();

      return querySnapshot.docs
          .map((doc) {
        var data = doc.data() as Map<String, dynamic>;
        data['uid'] = doc.id;
        return data;
      })
          .toList();
    } catch (e) {
      print("Error al obtener pacientes: $e");
      return [];
    }
  }

  void addToMyPatients(Map<String, dynamic> patient) {
    final user = FirebaseAuth.instance.currentUser;
    if (user != null) {
      // Obtener la referencia a la colección de pacientes del estudiante
      var patientCollection = FirebaseFirestore.instance
          .collection('users')
          .doc(user.uid)
          .collection('myPatients');

      // Verificar si el paciente ya está agregado
      patientCollection.doc(patient['uid']).get().then((doc) {
        if (doc.exists) {
          // Si el documento existe, mostrar mensaje
          ScaffoldMessenger.of(context).showSnackBar(
```

home_page_navigation_bar_page.dart

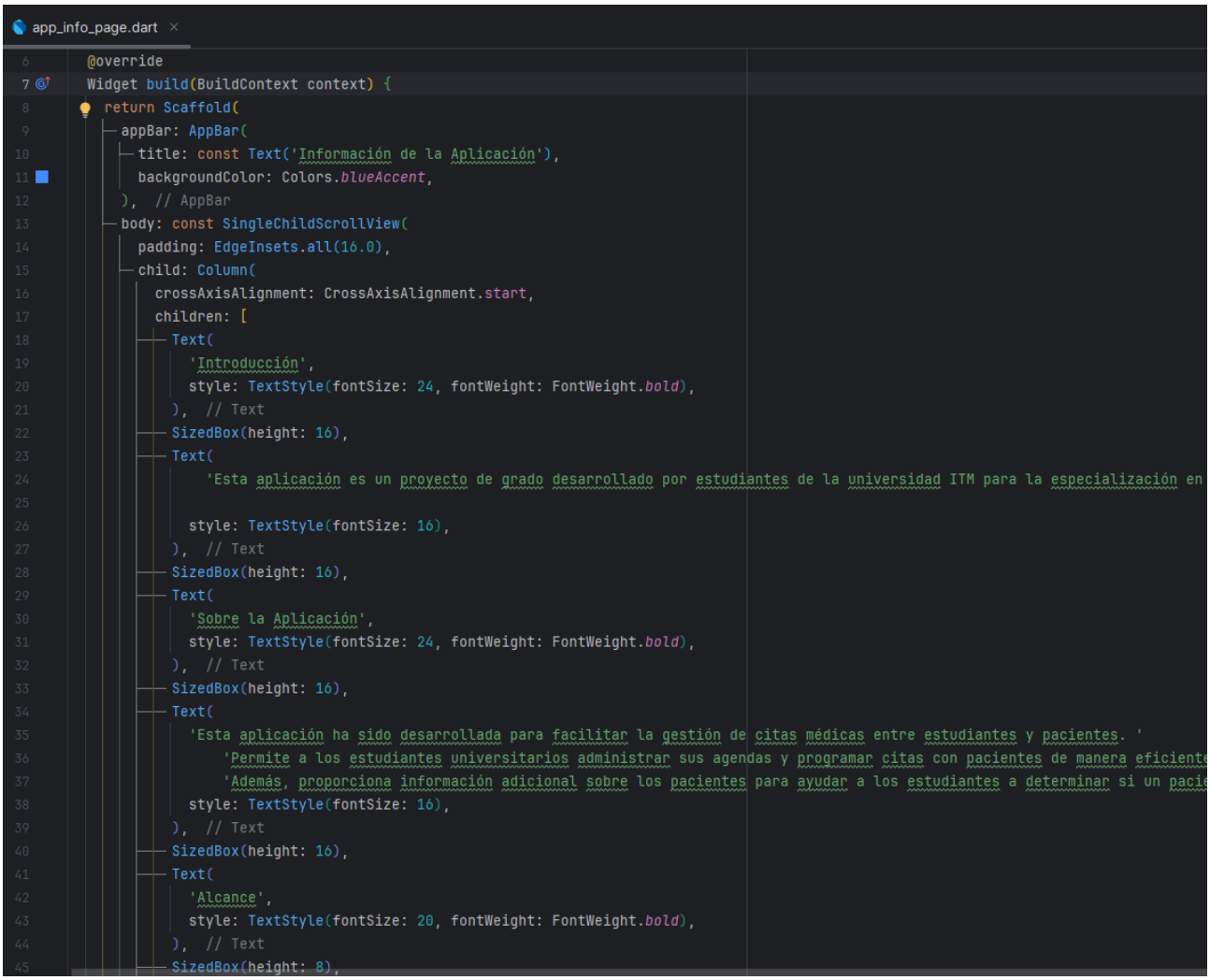
El archivo `home_page_navigation_bar_page.dart` define la barra de navegación de la página principal. Proporciona navegación entre diferentes secciones de la aplicación.

home_page_navigation_bar_page.dart

```dart
void _showPatientDiligencyForm() {
  Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) => const PatientDiligencyPage(),
    ),  // MaterialPageRoute
  ).then((_) {
    _checkPatientDiligencyCompletion(); // Recheck after completing the form
  });
}

void _setupRoleBasedUI(String role) {
  if (role == "estudiante") {
    _widgetOptions = [
      const AgendaPage(),
      const StudentHomePage(),
      const MyPatientPage(),
      const MyAppointmentsPage(),
      const ChatPage(),
      const ProfilePage(),
    ];
    _menuBar = [
      BottomNavigationBarItem(
        icon: Image.asset('assets/images/agenda.png'),
        label: 'Crear Agenda',
      ),  // BottomNavigationBarItem
      BottomNavigationBarItem(
        icon: Image.asset('assets/images/boca.png'),
        label: 'Valoraciones',
      ),  // BottomNavigationBarItem
      BottomNavigationBarItem(
        icon: Image.asset('assets/images/historia.png'),
        label: 'Mis Pacientes',
      ),  // BottomNavigationBarItem
      BottomNavigationBarItem(
        icon: Image.asset('assets/images/clinica.png'),
        label: 'Citas',
      ),  // BottomNavigationBarItem
      BottomNavigationBarItem(
        icon: Image.asset('assets/images/chat.png'),
```

app_info_page.dart

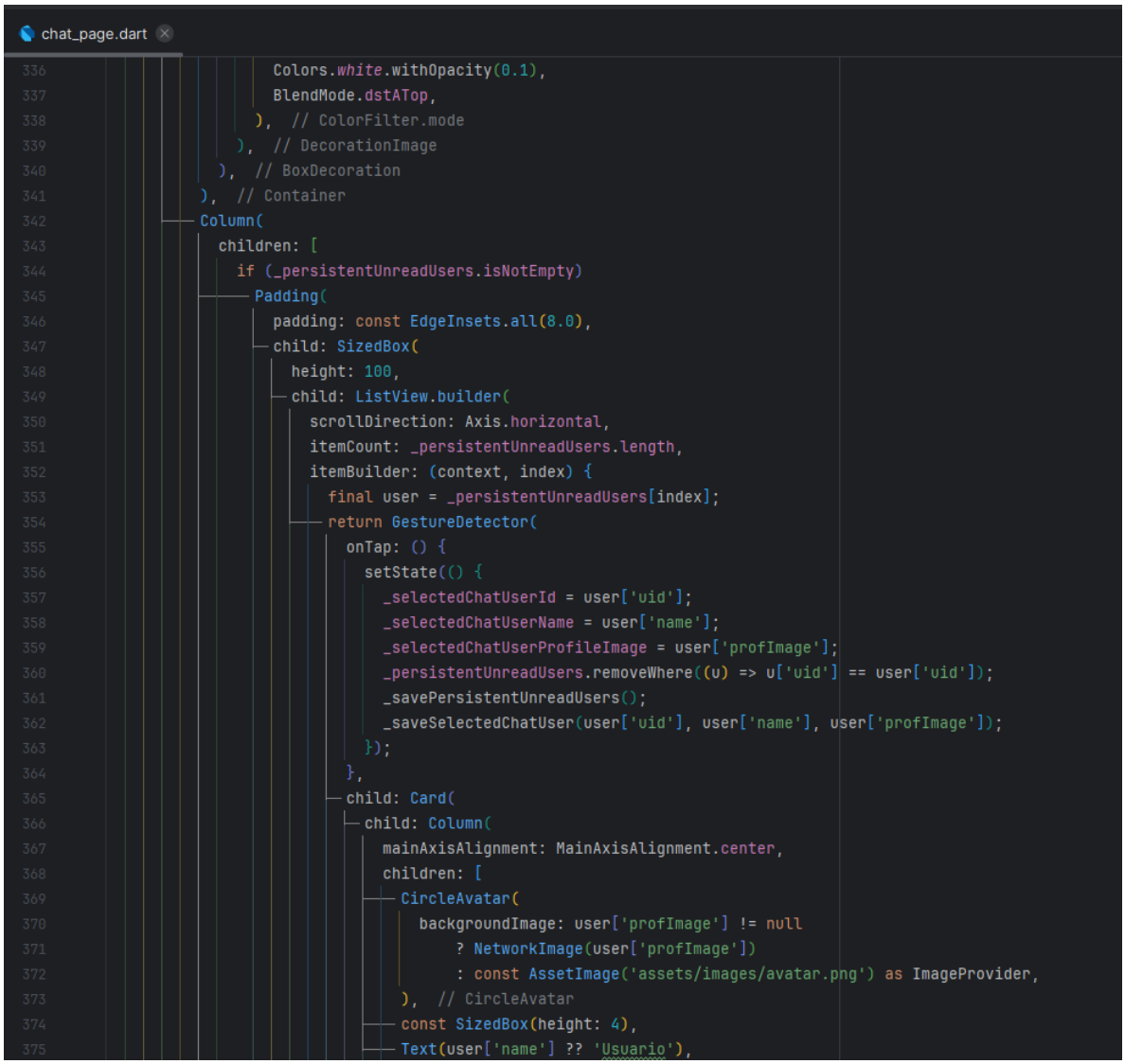
El archivo `app_info_page.dart` define la página de información de la aplicación. Proporciona detalles sobre la aplicación y su uso.



```dart
app_info_page.dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Información de la Aplicación'),
      backgroundColor: Colors.blueAccent,
    ),  // AppBar
    body: const SingleChildScrollView(
      padding: EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            'Introducción',
            style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
          ),  // Text
          SizedBox(height: 16),
          Text(
              'Esta aplicación es un proyecto de grado desarrollado por estudiantes de la universidad ITM para la especialización en

            style: TextStyle(fontSize: 16),
          ),  // Text
          SizedBox(height: 16),
          Text(
            'Sobre la Aplicación',
            style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
          ),  // Text
          SizedBox(height: 16),
          Text(
            'Esta aplicación ha sido desarrollada para facilitar la gestión de citas médicas entre estudiantes y pacientes. '
                'Permite a los estudiantes universitarios administrar sus agendas y programar citas con pacientes de manera eficiente
                'Además, proporciona información adicional sobre los pacientes para ayudar a los estudiantes a determinar si un paci
            style: TextStyle(fontSize: 16),
          ),  // Text
          SizedBox(height: 16),
          Text(
            'Alcance',
            style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
          ),  // Text
          SizedBox(height: 8),
```

chat_page.dart

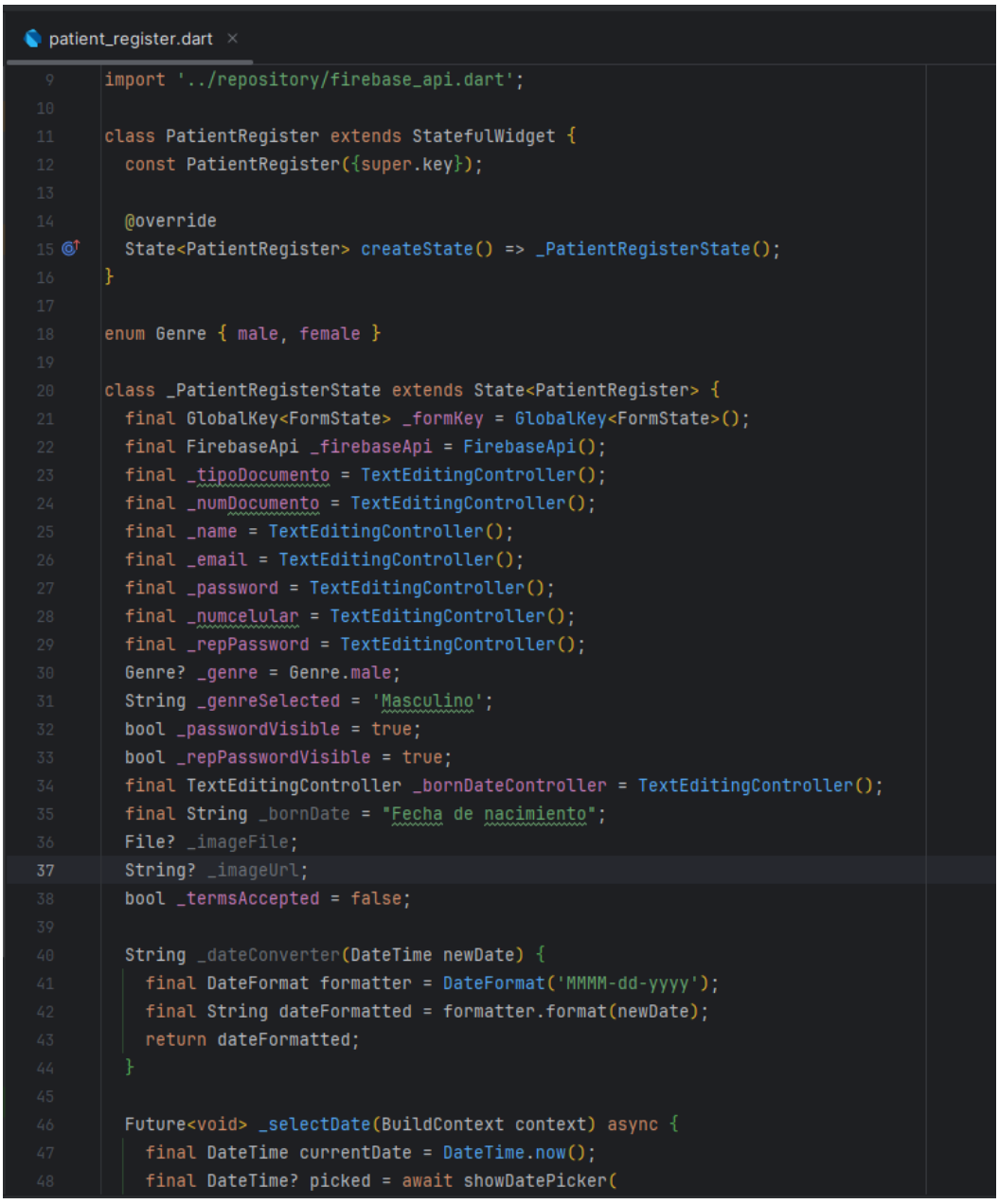
El archivo `chat_page.dart` define la página de chat. Permite la comunicación entre pacientes y estudiantes mediante mensajes en tiempo real.



```
chat_page.dart
                    Colors.white.withOpacity(0.1),
                    BlendMode.dstATop,
                  ),  // ColorFilter.mode
                ),  // DecorationImage
              ),  // BoxDecoration
            ),  // Container
          Column(
            children: [
              if (_persistentUnreadUsers.isNotEmpty)
                Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: SizedBox(
                    height: 100,
                    child: ListView.builder(
                      scrollDirection: Axis.horizontal,
                      itemCount: _persistentUnreadUsers.length,
                      itemBuilder: (context, index) {
                        final user = _persistentUnreadUsers[index];
                        return GestureDetector(
                          onTap: () {
                            setState(() {
                              _selectedChatUserId = user['uid'];
                              _selectedChatUserName = user['name'];
                              _selectedChatUserProfileImage = user['profImage'];
                              _persistentUnreadUsers.removeWhere((u) => u['uid'] == user['uid']);
                              _savePersistentUnreadUsers();
                              _saveSelectedChatUser(user['uid'], user['name'], user['profImage']);
                            });
                          },
                          child: Card(
                            child: Column(
                              mainAxisAlignment: MainAxisAlignment.center,
                              children: [
                                CircleAvatar(
                                  backgroundImage: user['profImage'] != null
                                      ? NetworkImage(user['profImage'])
                                      : const AssetImage('assets/images/avatar.png') as ImageProvider,
                                ),  // CircleAvatar
                                const SizedBox(height: 4),
                                Text(user['name'] ?? 'Usuario'),
```

patient_register.dart

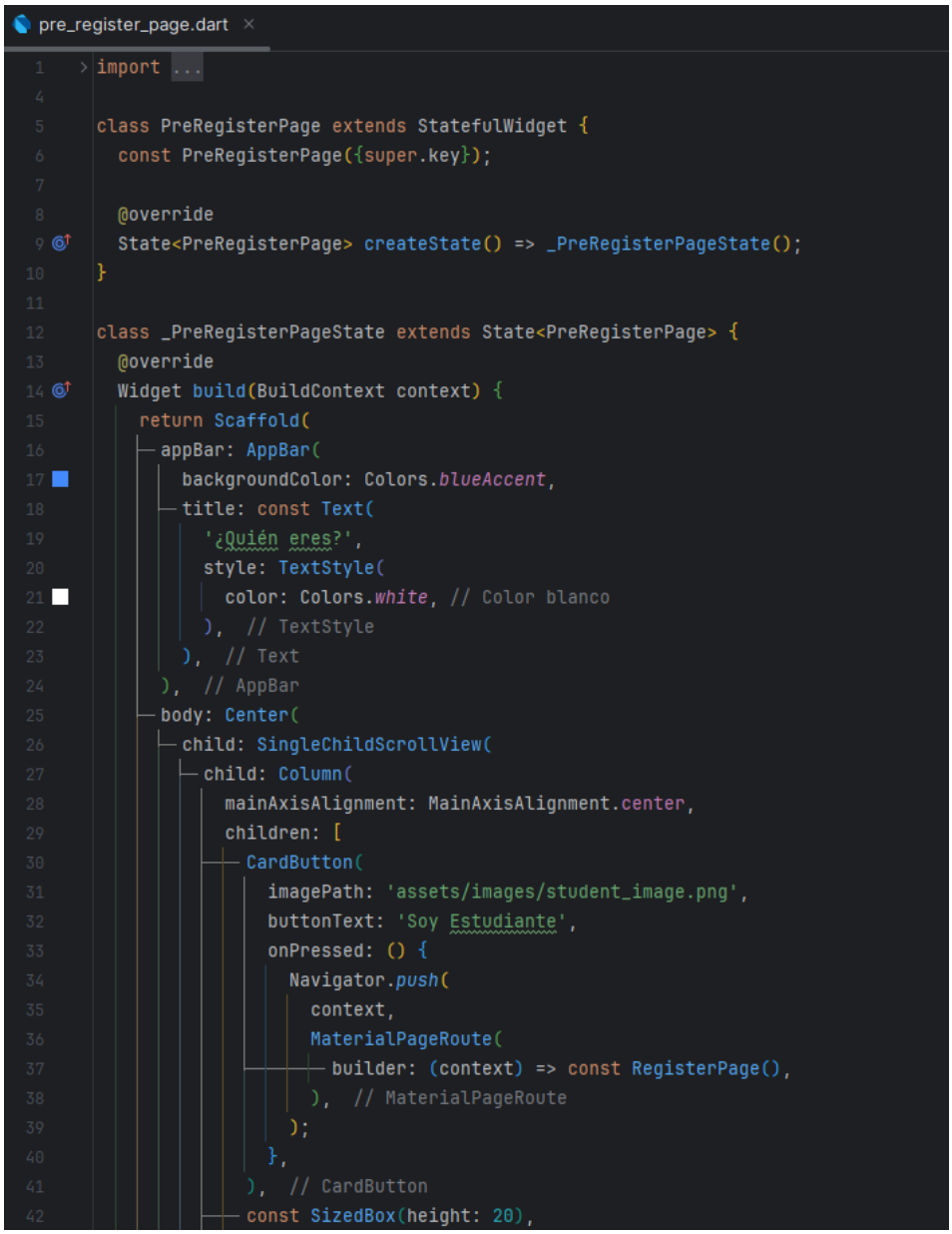
El archivo `patient_register.dart` define la página de registro de pacientes. Permite a los pacientes registrar sus datos personales y médicos en la aplicación.



```dart
import '../repository/firebase_api.dart';

class PatientRegister extends StatefulWidget {
  const PatientRegister({super.key});

  @override
  State<PatientRegister> createState() => _PatientRegisterState();
}

enum Genre { male, female }

class _PatientRegisterState extends State<PatientRegister> {
  final GlobalKey<FormState> _formKey = GlobalKey<FormState>();
  final FirebaseApi _firebaseApi = FirebaseApi();
  final _tipoDocumento = TextEditingController();
  final _numDocumento = TextEditingController();
  final _name = TextEditingController();
  final _email = TextEditingController();
  final _password = TextEditingController();
  final _numcelular = TextEditingController();
  final _repPassword = TextEditingController();
  Genre? _genre = Genre.male;
  String _genreSelected = 'Masculino';
  bool _passwordVisible = true;
  bool _repPasswordVisible = true;
  final TextEditingController _bornDateController = TextEditingController();
  final String _bornDate = "Fecha de nacimiento";
  File? _imageFile;
  String? _imageUrl;
  bool _termsAccepted = false;

  String _dateConverter(DateTime newDate) {
    final DateFormat formatter = DateFormat('MMMM-dd-yyyy');
    final String dateFormatted = formatter.format(newDate);
    return dateFormatted;
  }

  Future<void> _selectDate(BuildContext context) async {
    final DateTime currentDate = DateTime.now();
    final DateTime? picked = await showDatePicker(
```

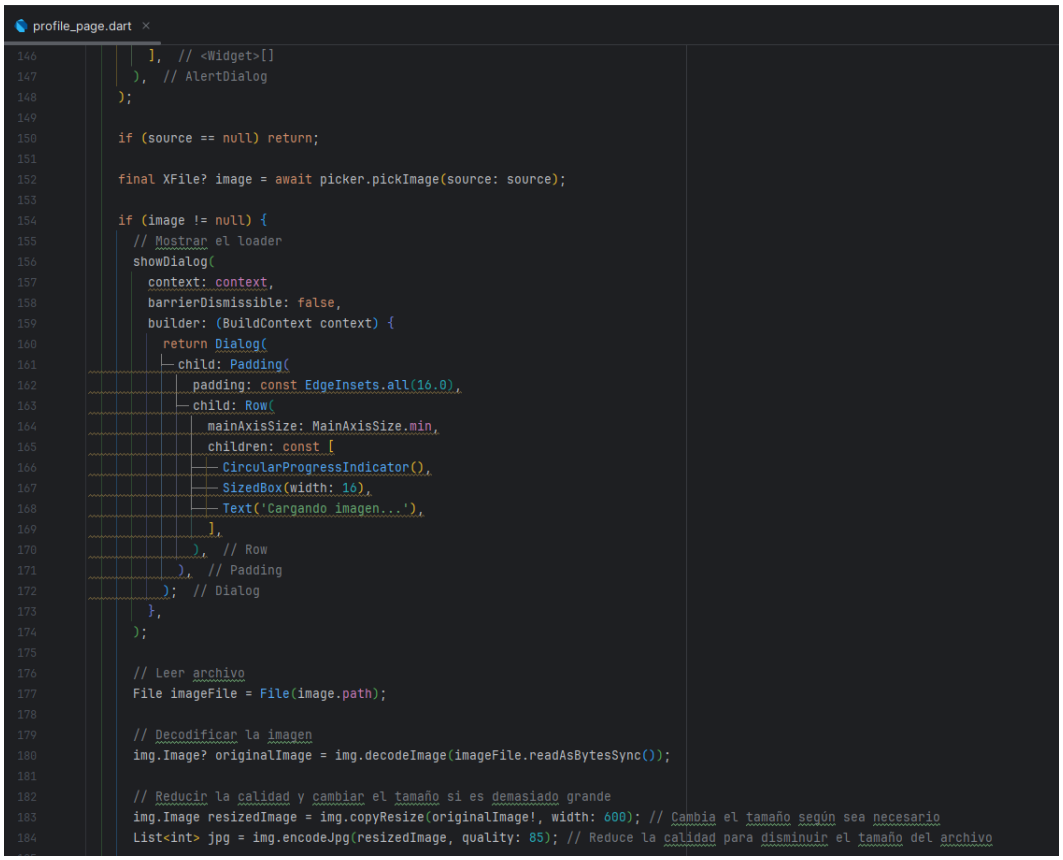
pre_register_*page*.dart

El archivo `pre_register_page.dart` define la página de pre-registro. Esta página es utilizada para recopilar información inicial antes de permitir el acceso completo a la aplicación.



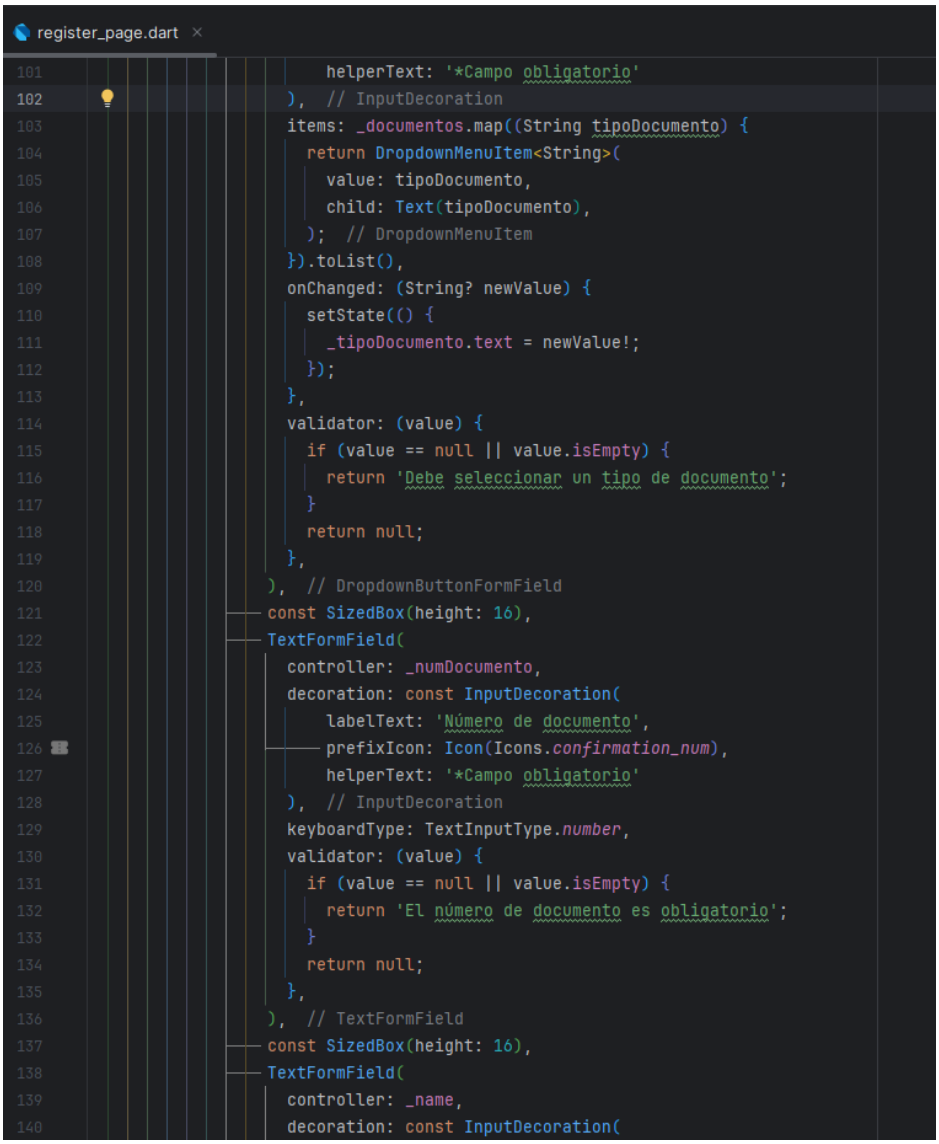
profile_page.dart

El archivo `profile_page.dart` define la página de perfil del usuario. Permite a los usuarios ver y editar su información personal y subir una imagen de perfil.



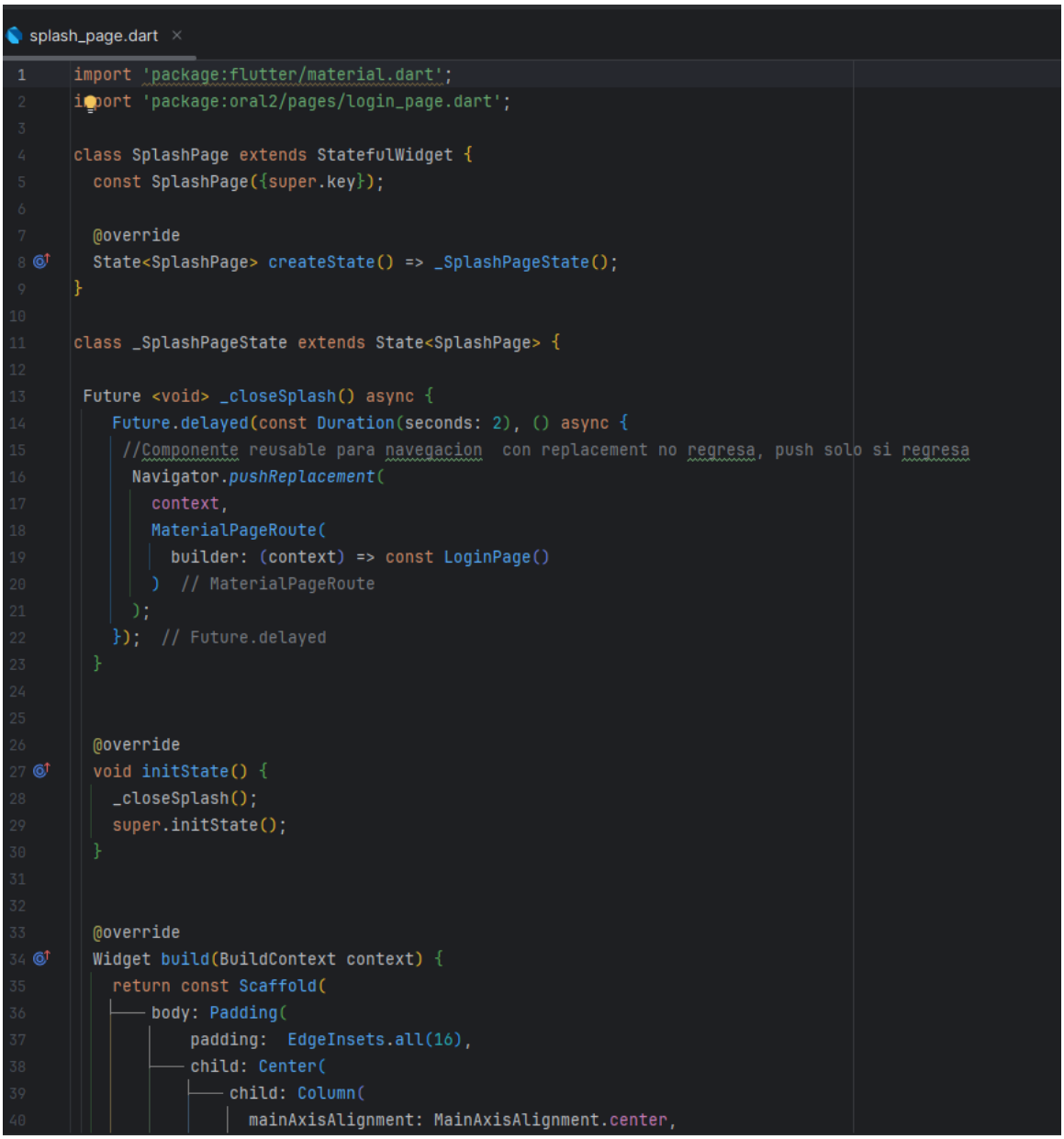
register_page.dart

El archivo `register_page.dart` define la página de registro general para nuevos usuarios. Permite a los usuarios crear una nueva cuenta en la aplicación.



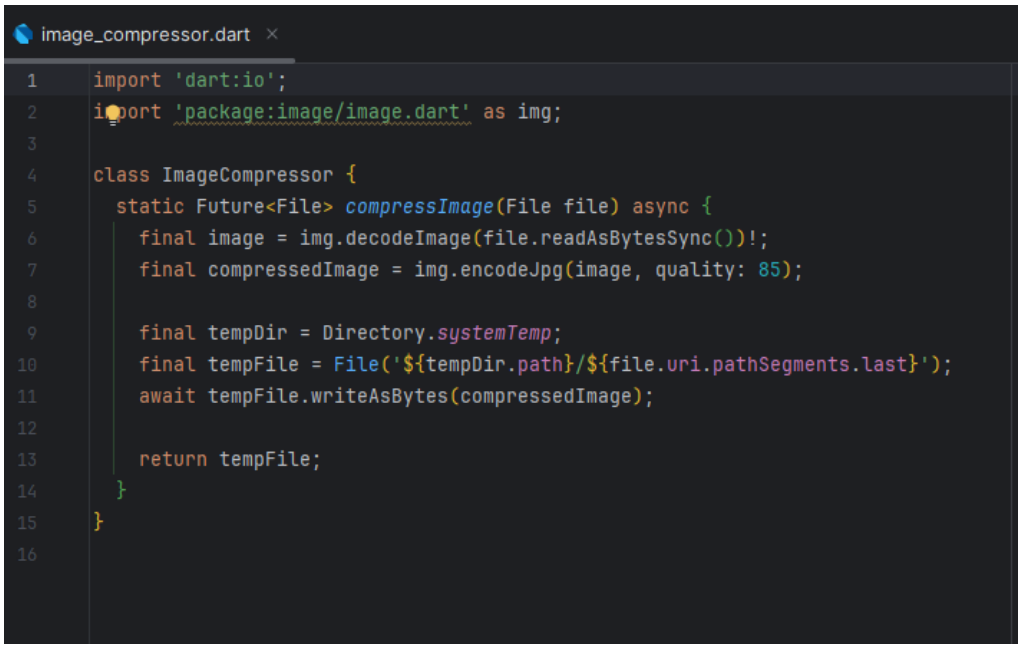
splash_page.dart

El archivo `splash_page.dart` define la página de splash. Esta página es la primera que se muestra al iniciar la aplicación y generalmente se utiliza para mostrar el logo y cargar los datos iniciales.



image_compressor.dart

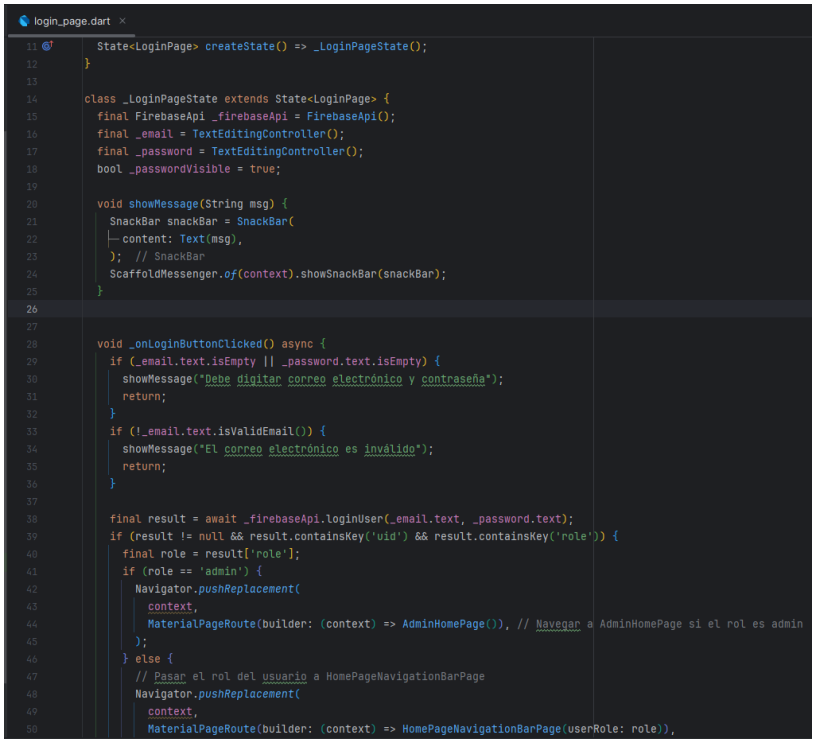
El archivo `image_compressor.dart` define la funcionalidad de compresión de imágenes. Permite reducir el tamaño de las imágenes antes de subirlas a Firebase Storage.



```dart
import 'dart:io';
import 'package:image/image.dart' as img;

class ImageCompressor {
  static Future<File> compressImage(File file) async {
    final image = img.decodeImage(file.readAsBytesSync())!;
    final compressedImage = img.encodeJpg(image, quality: 85);

    final tempDir = Directory.systemTemp;
    final tempFile = File('${tempDir.path}/${file.uri.pathSegments.last}');
    await tempFile.writeAsBytes(compressedImage);

    return tempFile;
  }
}
```

login_page.dart

El archivo `login_page.dart` define la página de inicio de sesión. Permite a los usuarios autenticarse en la aplicación utilizando sus credenciales.



```dart
    State<LoginPage> createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  final FirebaseApi _firebaseApi = FirebaseApi();
  final _email = TextEditingController();
  final _password = TextEditingController();
  bool _passwordVisible = true;

  void showMessage(String msg) {
    SnackBar snackBar = SnackBar(
    content: Text(msg),
    );  // SnackBar
    ScaffoldMessenger.of(context).showSnackBar(snackBar);
  }


  void _onLoginButtonClicked() async {
    if (_email.text.isEmpty || _password.text.isEmpty) {
      showMessage("Debe digitar correo electrónico y contraseña");
      return;
    }
    if (!_email.text.isValidEmail()) {
      showMessage("El correo electrónico es inválido");
      return;
    }

    final result = await _firebaseApi.loginUser(_email.text, _password.text);
    if (result != null && result.containsKey('uid') && result.containsKey('role')) {
      final role = result['role'];
      if (role == 'admin') {
        Navigator.pushReplacement(
          context,
          MaterialPageRoute(builder: (context) => AdminHomePage()), // Navegar a AdminHomePage si el rol es admin
        );
      } else {
        // Pasar el rol del usuario a HomePageNavigationBarPage
        Navigator.pushReplacement(
          context,
          MaterialPageRoute(builder: (context) => HomePageNavigationBarPage(userRole: role)),
```

7.6.12 Carpeta Repository

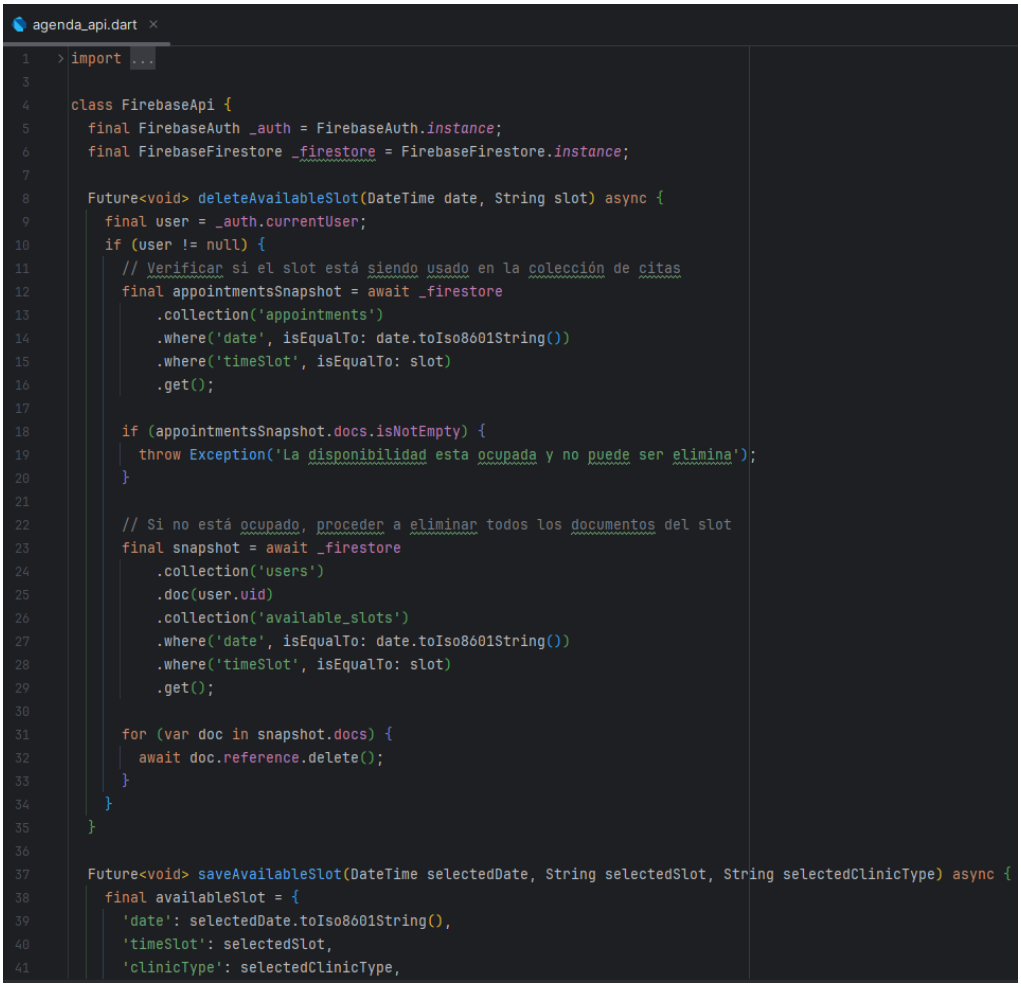
Appointment API

Archivo: Appointment_api.dart

Propósito: Este archivo contiene las funciones relacionadas con la gestión de citas en la aplicación. Maneja la creación, actualización, eliminación y obtención de citas de la base de datos de Firestore.

Funciones Principales:

- **createAppointment**: Crea una nueva cita en Firestore.
- **updateAppointment**: Actualiza la información de una cita existente.
- **deleteAppointment**: Elimina una cita de Firestore.
- **getAppointments**: Obtiene todas las citas de la base de datos.



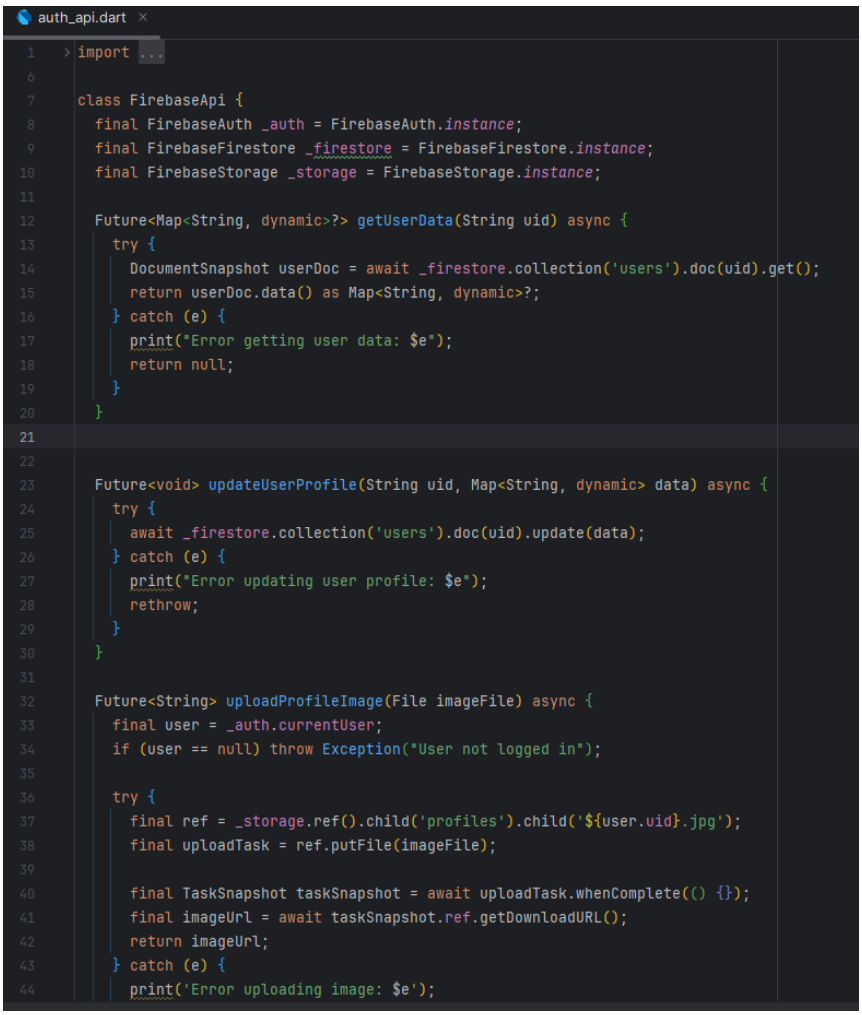
Authentication *API*

Archivo: auth_api.dart

Propósito: Este archivo maneja la autenticación de usuarios utilizando Firebase Auth. Contiene funciones para registrar, iniciar sesión, cerrar sesión y reautenticar a los usuarios.

Funciones Principales:

- **signUp**: Registra a un nuevo usuario.
- **signIn**: Inicia sesión con un usuario existente.
- **signOut**: Cierra la sesión del usuario actual.
- **reauthenticateUser**: Reautentica a un usuario utilizando su correo electrónico y contraseña.



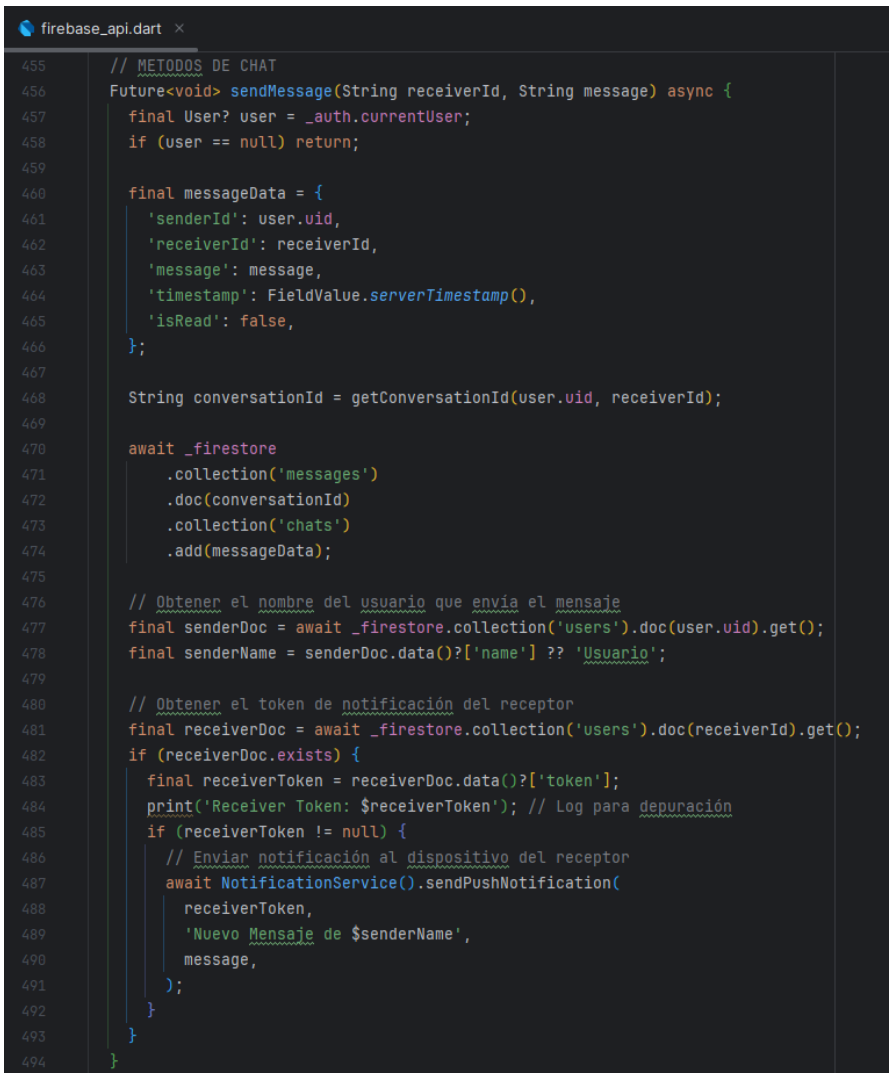
Firebase API

Archivo: firebase_api.dart

Propósito: Este archivo proporciona una interfaz para interactuar con varios servicios de Firebase, incluyendo Firestore y Firebase Storage. Maneja la carga y descarga de datos e imágenes.

Funciones Principales:

- **uploadImage**: Sube una imagen a Firebase Storage.
- **downloadImage**: Descarga una imagen de Firebase Storage.
- **getUserData**: Obtiene los datos de un usuario desde Firestore.
- **updateUserProfile**: Actualiza el perfil de un usuario en Firestore.



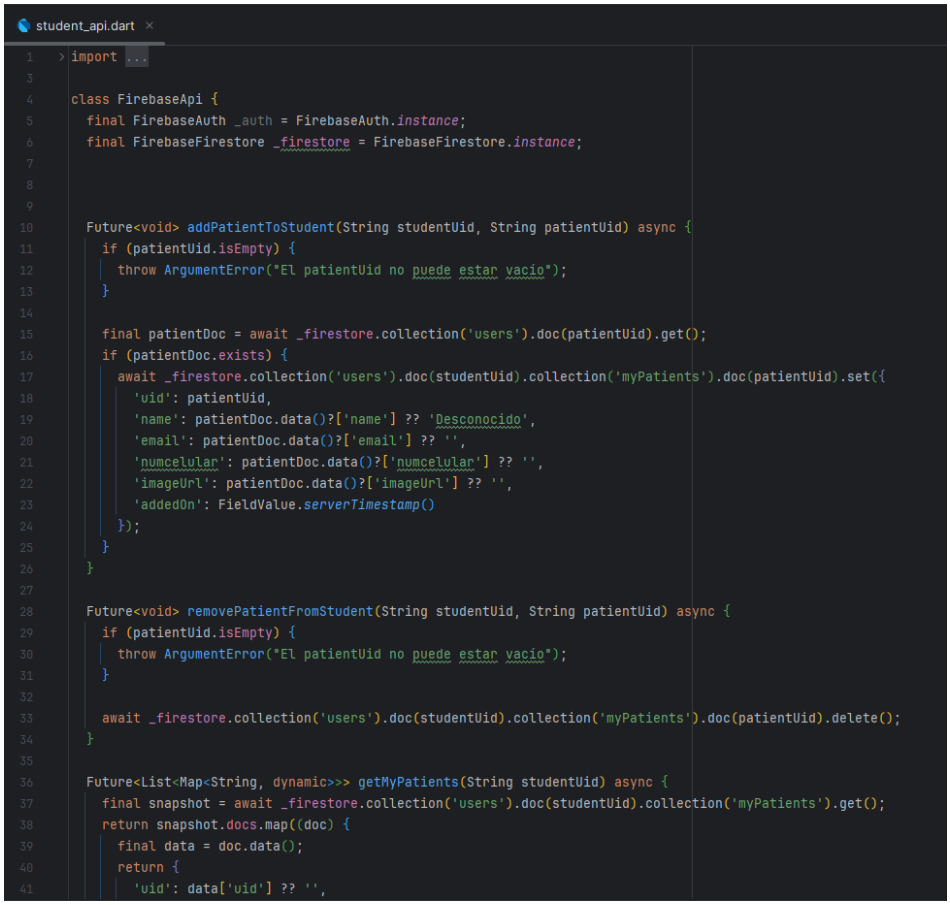
Student API

Archivo: student_api.dart

Propósito: Este archivo contiene funciones específicas para gestionar la información de los estudiantes dentro de la aplicación. Maneja la obtención y actualización de los datos de los estudiantes en Firestore.

Funciones Principales:

- **getStudentData**: Obtiene los datos de un estudiante desde Firestore.
- **updateStudentProfile**: Actualiza el perfil de un estudiante en Firestore.



```dart
import ...

class FirebaseApi {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;



  Future<void> addPatientToStudent(String studentUid, String patientUid) async {
    if (patientUid.isEmpty) {
      throw ArgumentError("El patientUid no puede estar vacío");
    }

    final patientDoc = await _firestore.collection('users').doc(patientUid).get();
    if (patientDoc.exists) {
      await _firestore.collection('users').doc(studentUid).collection('myPatients').doc(patientUid).set({
        'uid': patientUid,
        'name': patientDoc.data()?['name'] ?? 'Desconocido',
        'email': patientDoc.data()?['email'] ?? '',
        'numcelular': patientDoc.data()?['numcelular'] ?? '',
        'imageUrl': patientDoc.data()?['imageUrl'] ?? '',
        'addedOn': FieldValue.serverTimestamp()
      });
    }
  }

  Future<void> removePatientFromStudent(String studentUid, String patientUid) async {
    if (patientUid.isEmpty) {
      throw ArgumentError("El patientUid no puede estar vacío");
    }

    await _firestore.collection('users').doc(studentUid).collection('myPatients').doc(patientUid).delete();
  }

  Future<List<Map<String, dynamic>>> getMyPatients(String studentUid) async {
    final snapshot = await _firestore.collection('users').doc(studentUid).collection('myPatients').get();
    return snapshot.docs.map((doc) {
      final data = doc.data();
      return {
        'uid': data['uid'] ?? '',
```

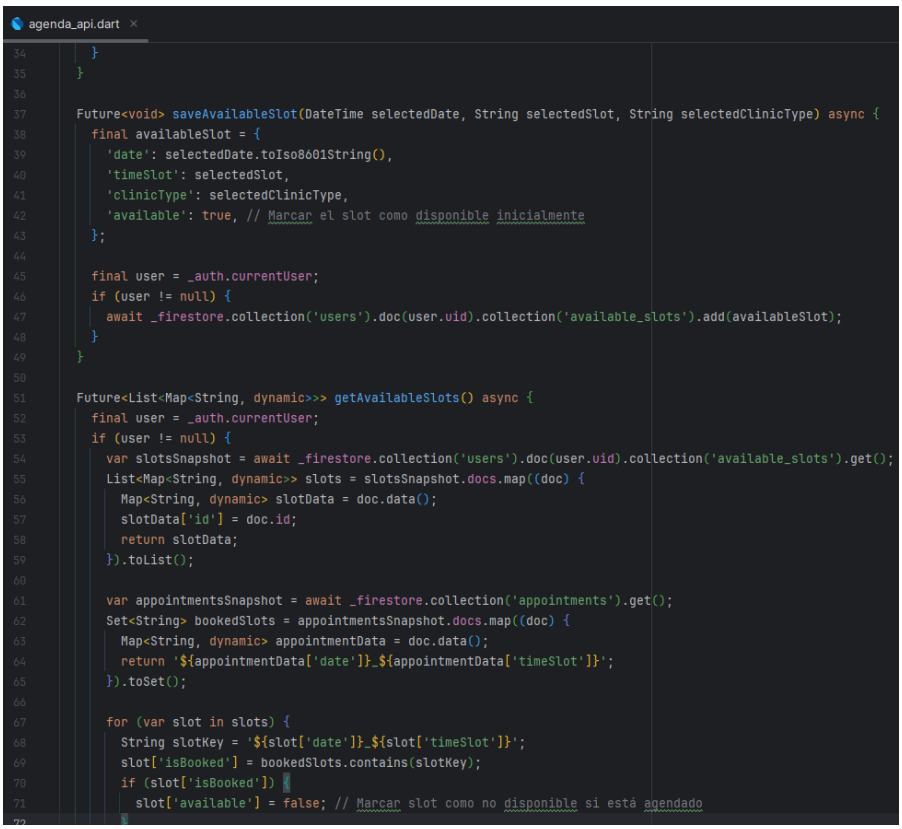
Agenda API

Archivo: agenda_api.dart

Propósito: Este archivo gestiona la disponibilidad de los estudiantes para las citas. Maneja la creación, actualización y eliminación de slots de disponibilidad en Firestore.

Funciones Principales:

- **saveAvailableSlot**: Guarda un nuevo slot de disponibilidad.
- **deleteAvailableSlot**: Elimina un slot de disponibilidad existente.
- **getAvailableSlots**: Obtiene los slots de disponibilidad desde Firestore.



Main

Archivo: main.dart

Propósito: Este es el archivo principal de la aplicación Flutter. Inicializa la aplicación, configura Firebase y define la estructura básica de la aplicación, incluyendo las rutas y la pantalla inicial.

Funciones Principales:

- **main**: Punto de entrada de la aplicación.
- **MyApp**: Clase principal que define la estructura y navegación de la aplicación.

main.dart

```dart
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'package:flutter_localizations/flutter_localizations.dart';
import 'package:oral2/pages/services/notificationservices.dart';
import 'package:oral2/pages/splash_page.dart';
import 'firebase_options.dart';

void main() async {

  WidgetsFlutterBinding.ensureInitialized();
  await NotificationService().init();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'OralApp',
      localizationsDelegates: const[
        GlobalMaterialLocalizations.delegate,
        GlobalWidgetsLocalizations.delegate,
      ],
      supportedLocales: const [
        Locale("es", "CO"),
        Locale("en", "US"),
      ],
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),  // ThemeData
      home: const SplashPage(),
    );  // MaterialApp
  }
```

Utils

Archivo: utils.dart

Propósito: Este archivo contiene funciones utilitarias y constantes que se utilizan en toda la aplicación para diversas tareas como formateo de fechas, validaciones, y otros.

Funciones Principales:

- **formatDate**: Formatea una fecha en un formato legible.
- **validateEmail**: Valida si un correo electrónico tiene el formato correcto.

```dart
import 'package:intl/intl.dart';

class Utils {
  static bool isEmail(String email) {
    String stringRegex = r'^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()[\]\\.,;:\s@\"]+)*)|(\".+\"))@((\[[0-9]{1,
    RegExp regExp = RegExp(stringRegex);
    return regExp.hasMatch(email);
  }

  static bool isSizePasswordValid(String password) {
    return password.length > 5;
  }

  static String dateConverter(DateTime newDate) {
    final DateFormat formatter = DateFormat('MMMM-dd-yyyy');
    return formatter.format(newDate);
  }
}
```

8. CONCLUSIONES, RECOMENDACIONES Y TRABAJO FUTURO

8.1 Tendencias y Desviaciones

Las tendencias, de forma genérica las podemos enumerar, haciendo un análisis coherente de lo que a hoy nos aborda de forma negativa y positiva en la distribución técnica de Oral Data

Inciso 1 Recopilación de Datos

- **Uso del Sistema**: Registra el número de veces que el sistema es accedido, el tiempo promedio de sesión, y las funcionalidades más utilizadas.
- **Agendamiento de citas**: Analiza el volumen de citas agendadas, canceladas, y confirmadas, así como el tiempo promedio de espera para obtener una cita.
- **Solicitudes de soporte**: Cuantifica las consultas de soporte técnico recibidas y cómo se resolvieron estas consultas.
- **Registros**: Garantiza de forma gradual que la información almacenada al interior de la base de datos no tiene riesgo de ser modificada.

Inciso 2 Análisis de Tendencias

- **Tendencias de Agendamiento**: Observa si hay aumentos o disminuciones en la demanda de citas en momentos específicos, como antes o después de vacaciones escolares.

Inciso 3: Identificación de Desviaciones

- **Desviaciones en el uso**: Detectamos comportamientos inusuales en el acceso o uso del sistema, como sesiones muy largas o intentos repetidos de acceso fallidos, teniendo en cuenta que esto hace parte de una prueba funcional al momento de realizar un registro para el acceso a la plataforma de Oral Data.
- **Desviaciones en el agendamiento de citas odontológicas**: Identificamos patrones relacionados en la cancelación de citas, al cancelar la cita el registro aún se sigue conservando en la base de datos.

Inciso 4: Evaluación y Acción

- **Evaluación de impacto**: Asociamos las tendencias y desviaciones observadas con posibles factores externos, como cambios en la oferta de servicios dentales o en la política académica, el *feedback* puede variar en base las funcionalidades nuevas que se le pueden ir aplicando al software.

- **Acciones correctivas**: Proponemos mejoras o correcciones basadas en los hallazgos, como la adición de nuevas funcionalidades para facilitar el agendamiento de citas, otro de estas mejoras radica en la fluidez del software en vista de que mucho de lo que a hoy se debe de depurar a nivel de *backend*, se está haciendo en el *frontend*.

Inciso 5: Seguimiento y acciones correctivas

- **Seguimiento continuo**: Realizar análisis periódicos para rastrear el impacto de las acciones tomadas y detectar nuevas tendencias o desviaciones.
- **Reevaluación estratégica**: Revisa y ajusta los objetivos y estrategias del software de gestión de citas en función de los resultados obtenidos y las necesidades cambiantes de los usuarios.

8.2 Lecciones Aprendidas y Recomendaciones

8.2.1 Lecciones

A continuación, vamos a hacer una descripción de las lecciones aprendidas:

1- Identificación de los riesgos:

La identificación temprana de los riesgos y sugestión permitió evitar retrasos en las entregas y bloqueos futuras

2- Planes establecidos

A pesar de tener un plan establecido inicialmente, a medida que avanzo el proyecto surgieron nuevos requisitos o procesos que requiero tomar decisiones en las cuales se pudo medir la capacidad de respuesta ante cambios inesperados.

3-Avance en el proyecto:

El avance del proyecto permitió evidenciar la necesidad de mantener una comunicación constante y regular con los diferentes miembros del equipo y las partes interesadas (stakeholders), de manera que se brindara información transparente y se evitaran malentendidos en etapas más avanzadas.

8.2.2 Recomendaciones

- Es importante realizar capacitaciones al equipo de desarrollo en temas técnicos necesarios para el desarrollo de la aplicación, de manera que se aumente la eficiencia y se eviten bloqueos.

- Es importante contar con herramientas de seguimiento del proyecto las cuales permitan evaluar el estado actual, identificando riesgos que puedan afectar las entregas en los tiempos estimados y tomar medidas preventivas.

- Establecer reuniones de retrospectivas permite identificar oportunidades de mejora, para así realizar acciones que permitan

- Es importante construir una cultura en la que se reconozca la importancia de las retrospectivas, cambiando la mentalidad de que son una pérdida de tiempo, debido a que las pequeñas medidas de acción tomadas para mejorar falencias ayudan a mejorar problemas recurrentes.

- Es importante tener una comunicación clara y activa con los *stakeholders* sobre los alcances del proyecto y avance de este para así alinear las expectativas y ser transparentes con el progreso del proyecto.

8.3 Conclusiones y Próximos Pasos

- los Indicadores del proyecto permiten evaluar el progreso y el rendimiento del software Oraldata, proporcionando una visión integral del desarrollo e identificando áreas de mejora, de tal manera que se cumpla con los objetivos inicialmente planteados.
- La satisfacción del público objetivo de la aplicación es de gran importancia para el éxito del software, por tal motivo el uso de encuestas y feedback son herramientas que permiten evaluar la percepción del software y evaluar áreas de mejora

Las lecciones aprendidas evidencian la importancia de identificar riesgos en etapas tempranas, la adaptabilidad ante cambios inesperados y la necesidad de mantener una comunicación constante entre los miembros del equipo y los stakeholders

9. REFERENCIAS

- Software Dentalink. (2023). Recuperado el 23 de octubre de 2023, de https://www.softwaredentalink.com
- My Dental Clinic. (2023). Recuperado el 23 de octubre de 2023, de https://web.dentalclinicapp.com/login
- Arevalo, F. (2015). Gestión de pacientes en facultades de odontología: Una revisión de literatura. Revista de Odontología de la Universidad de Costa Rica, 27(2), 115-124.
- OPS/OMS Chile. (2009). Tecnologías de la información en salud para la atención primaria: Guía para la toma de decisiones. Santiago de Chile: OPS/OMS.
- Rojas, R., & Galvis, G. (2023). El impacto de la tecnología en la educación dental. Journal of Dental Education, 45(3), 150-160
- DefiniciónABC. (2023). Definición de Gestión de Calidad. Recuperado el 23 de octubre de 2023, de https://www.definicionabc.com/economia/gestion-de-calidad.php
- Universidad de Santiago de Compostela. (2023). Prácticas de la Facultad de Medicina y Odontología. Recuperado el 23 de octubre de 2023, de https://www.usc.gal/es/centro/facultad-medicina-odontologia/practicas
- Entity Framework Tutorial. (2023). What is Entity Framework? Recuperado el 23 de octubre de 2023, de https://www.entityframeworktutorial.net/what-is-entityframework.aspx
- Blazor University. (2023). What is Blazor? Recuperado el 23 de octubre de 2023, de https://blazor-university.com/overview/what-is-blazor/
- Microsoft. (2023). Introduction to Azure Blob Storage. Recuperado el 23 de octubre de 2023, de https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blobs- introduction
- Ausubel, D. P., Novak, J. D., & Hanesian, H. (1978). Aprendizaje y desarrollo humano. Holt, Rinehart and Winston.

10. ANEXO

10.1 Anexo A Cronograma de actividades

Objetivo 1 Objetivo2 Objetivo 3 Objetivo 4

		Agosto		Sept		Octubre		Noviembre		Febrero		Marzo		Abril		Mayo	
Actividad	**Semana**	**S1-S2**	**S3-S4**	**S1-S2**	**S3-S4**	**S1-S2**	**S3-S4**	**S1-S2**	**S3-S4**	**S1-S2**	**S3-S4**	**S1-S2**	**S3-S4**	**S1-S2**	**S3-S4**	**S1-S2**	**S3-S4**
Elaboración de la propuesta inicial																	
Técnicas de elicitación --> Entrevistas a profesores de facultades de odontologia para comprender sus necesidades y expectativas																	
Diseño de modelo de datos para el sistema de gestión de pacientes (entidades, las relaciones y atributos)																	
Análisis de los requisitos técnicos necesarios para el sistema de gestión de pacientes (requisitos																	

funcionales y requisitos no funcionales)															
Priorizarción de las necesidades técnicas identificadas (teniendo en cuenta importancia, complejidad)															
Definición de requisitos funcionales (Funciones y los procesos que el sistema debe realizar)															
Definición de requisitos no funcionales específicos del sistema															
Desarrollar de diagramas del proceso, para comprender: pasos, interacciones y actores															
Definición de los componentes del sistema de gestión de paciente(lenguaje, Framework, otras tecnologias a usar)															
Desarrollo de un prototipo del sistema OralData															
Desarrollo de todas las funcionalidades de la aplicación															

Testing de requisitos funcionales y no funcionales del sistema																
Implementación del sistema en una facultad de odontología																

10.2 Anexo B Compromiso para el desarrollo de trabajos de grado

Fecha	15	11	23

Información general

Título del Trabajo de Grado	OralData- Sistema de gestión de pacientes para prácticas clínicas
Modalidad	Proyecto de grado

Estudiante(s)

Nombre completo	Juan David Nanclares
Cédula de ciudadanía	1036659074
Programa	Especialización en ingeniería de software

Nombre completo	Juan Camilo Ossa Guzmán
Cédula de ciudadanía	8104828
Programa	Especialización en ingeniería de software

Asesor(es)

Nombre completo	
Cédula de ciudadanía	
Departamento	

Nombre completo	
Cédula de ciudadanía	
Departamento	

El(Los) asesor(es) del trabajo de grado en mención se compromete(n) con los siguientes deberes:

a) Orientar el desarrollo técnico y metodológico necesario para cumplir con los objetivos definidos en el trabajo de grado.
b) Hacer seguimiento del cronograma aprobado.
c) Informar al Comité de Trabajos de Grado de la Facultad sobre cualquier anomalía en relación con el desarrollo del trabajo de grado.
d) Cumplir y velar por el respeto a las normas de propiedad intelectual y derechos de autor.

El(Los) estudiante(s) se compromete(n) con los siguientes deberes:

a) Manejar de forma confidencial la información institucional a la cual tendrá acceso en virtud del desarrollo del trabajo de grado, a usarla única y exclusivamente para los fines del trabajo de grado mencionado y a no revelarla directa o indirectamente a ninguna persona, durante la vigencia de

este trabajo, ni después de su terminación. En caso de violar la confidencialidad

a la que se compromete responderá en los términos de ley y normativa interna de la Institución.

b) Respetar la propiedad intelectual de terceros y con ello evitar el plagio u otra clase de reclamación que al respecto pudiera sobrevenir y que resulte violatoria de los derechos de autor y relativos a la propiedad intelectual.

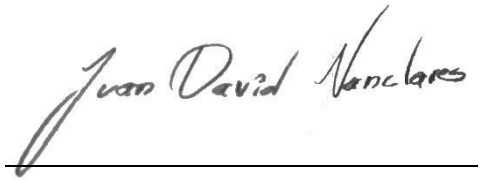
c) Asumir toda la responsabilidad en caso de presentarse cualquier reclamación o acción por parte de un tercero en cuanto a los derechos de autor sobre la obra en cuestión.

d) Abstenerse de llevar a cabo cualquier actuación que implique la comisión de falta disciplinaria o delito, tales como falsificación de firmas o su uso indebido, suplantación, alteración de documentos públicos o privados, y demás actuaciones que atenten contra la fe pública.

Los aspectos relacionados con la propiedad intelectual del trabajo de grado se regirán de acuerdo con lo establecido en el Estatuto de Propiedad Intelectual del Instituto Tecnológico Metropolitano - ITM, Acuerdo No. 34 de julio 23 de 2013 del Consejo Directivo.

En señal de aceptación se firma este documento.

FIRMA ESTUDIANTES



FIRMA ASESORES

FECHA ENTREGA: