DESARROLLO DE UNA PLATAFORMA WEB Y MÓVIL PARA PACIENTES Y PROFESIONALES MÉDICOS

DEVELOPMENT OF A WEB AND MOBILE PLATFORM FOR PATIENTS AND MEDICAL PROFESSIONALS

TRABAJO FIN DE GRADO
CURSO 2023-2024

AUTORES
FRANCISCO JAVIER ANTORANZ ESTEBAN
SERGIO SÁNCHEZ CHAMIZO
CARLOS PEÑA PAN

DIRECTOR
ANTONIO SARASA CABEZUELO

GRADO EN INGENIERÍA DEL SOFTWARE
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

DESARROLLO DE UNA PLATAFORMA WEB Y MÓVIL PARA PACIENTES Y PROFESIONALES MÉDICOS

DEVELOPMENT OF A WEB AND MOBILE PLATFORM FOR PATIENTS AND MEDICAL PROFESSIONALS

TRABAJO DE FIN DE GRADO EN INGENIERÍA DEL SOFTWARE

AUTORES

FRANCISCO JAVIER ANTORANZ ESTEBAN

SERGIO SÁNCHEZ CHAMIZO

CARLOS PEÑA PAN

DIRECTOR

ANTONIO SARASA CABEZUELO

CONVOCATORIA: JUNIO 2024

GRADO EN INGENIERÍA DEL SOFTWARE

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

14 DE MAYO DE 2024

DEDICATORIA

A todos aquellos que nos han dado la
oportunidad para llegar hasta aquí y
realizar este trabajo.

AGRADECIMIENTOS

Nos gustaría agradecer a Antonio Sarasa por su dedicación y atención hacia nosotros en este duradero proceso, a todos los profesores que nos han transmitido los conocimientos necesarios para poder llevarlo a cabo, a Víctor Giménez por su apoyo incondicional en el grupo de Whatsapp y por último y no menos importante a nuestras familias.

RESUMEN

DESARROLLO DE UNA PLATAFORMA WEB Y MÓVIL PARA PACIENTES Y PROFESIONALES MÉDICOS

El objetivo de este trabajo de fin de grado es elaborar un prototipo de aplicación que sea capaz de gestionar la interacción entre doctores y pacientes facilitando diferentes servicios.

El proyecto cuenta con una aplicación web destinada a los doctores y una aplicación móvil para los pacientes.

Los doctores pueden planificar sus citas con los pacientes, crearles un historial médico, configurarles un tratamiento y consultar con ellos a través de un chat.

Los pacientes pueden solicitar una cita, consultar sus citas en un horario, recibir alarmas de los tratamientos para la toma de un medicamento y crear una consulta con un médico.

Palabras clave

Doctor, Paciente, Cita, Tratamiento, Consulta, Historial, Web, Móvil, Servidor.

ABSTRACT

DEVELOPMENT OF A WEB AND MOBILE PLATFORM FOR PATIENTS AND MEDICAL PROFESSIONALS

The objective of this final degree project is the development of an application prototype that is capable of managing the interaction between doctors and patients by facilitating different services.

The project has a web application for doctors and a mobile application for patients.

Doctors can plan their appointments with patients, create a medical history for them, set up a treatment for them, and consult with them via chat.

Patients can request an appointment, consult appointments in their schedule, receive treatment alarms for taking medication, and create a consultation with a doctor.

Keywords

Doctor, Patient, Appointment, Treatment, Consultation, History, Web, Mobile, Server.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS CONTENIDOS

ÍNDICE DE FIGURAS APÉNDICES

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

1.1 Motivación

La motivación de este proyecto de Fin de Grado surge de la necesidad de superar las barreras de acceso y mejorar la calidad de vida de las personas a través de una adecuada atención médica.

Son múltiples las dificultades a las que se enfrentan los médicos y los pacientes. Entre estas dificultades destacan la escasez de tiempo para los desplazamientos, la falta de familiaridad de ciertos sectores de la población la tecnología, y las limitaciones geográficas, especialmente para personas con movilidad reducida.

La solución ideada para abordar estos desafíos consiste en el desarrollo de una aplicación móvil dirigida a los pacientes, complementada con una plataforma web diseñada específicamente para los doctores. Estas herramientas están creadas con el propósito de facilitar y optimizar la comunicación de ambas partes.

1.2 Objetivos

El objetivo de este trabajo de fin de grado es elaborar una aplicación capaz de facilitar el uso a pacientes y doctores, mejorando la comunicación entre ambos y optimizando el proceso de atención médica. Para cumplir este objetivo se crearon las siguientes marcas:

- Investigar las dificultades: Su objetivo es identificar las principales necesidades tanto de los médicos como de los pacientes para determinar cuáles son las demandas más usuales y centrarse en solucionarlas de manera efectiva. Esta investigación permite comprender y orientar el desarrollo de la aplicación.
- Diseñar la aplicación web y móvil: El segundo objetivo sería plantear el diseño de ambas aplicaciones para que la comunicación sea efectiva. Para ello, ambas contarán con una estructura sencilla e intuitiva para que

todo tipo de personas puedan navegar libremente independientemente de su nivel de experiencia tecnológica.

- Garantizar la seguridad y la privacidad de datos: El tercer objetivo es comprobar que los datos personales que se tratan están guardados de manera segura y no existen vulnerabilidades que hagan accesibles estos datos.
- Elaborar pruebas: El cuarto objetivo es elaborar pruebas sobre las diferentes funcionalidades para asegurarse que los usuarios no tendrán ningún tipo de problema.
- Probar la aplicación en una parte del sector médico: El quinto objetivo consiste en identificar una pequeña parte del sector médico, como un hospital o un centro de salud. Se distribuye la aplicación y se proporciona asistencia de ser necesario.
- Analizar los resultados: El último objetivo es analizar los datos recogidos durante las pruebas y utilizar esta información para realizar mejoras. Este análisis evaluará el rendimiento de la aplicación en términos de usabilidad, eficacia y satisfacción del usuario. Basándose en estos datos se llevarán a cabo ajustes y mejoras continuas para optimizar la funcionalidad de la aplicación.

1.3 Plan de Trabajo

En esta sección se muestra el plan de trabajo a seguir para la consecución de los objetivos descritos en el apartado anterior. Al comienzo del Trabajo de Fin de Grado se separaron los hitos y se establecieron fechas aproximadas de entregas recomendadas para cada uno (ver *Tabla 1,* Figura *1.1*).

Hitos	Fecha de entrega
Planificación de requisitos	Mediados de octubre
Gestión de usuarios	Mediados de noviembre
Gestión de pacientes	Mediados de diciembre
Gestión del historial médico	Mediados de enero
Gestión de tratamientos	Final de enero
Gestión de citas médicas	Final de Febrero
Gestión de consultas	Final de Marzo
Finalización de código	Final de Abril
Realización de la memoria	14 de mayo

Tabla 1- Fecha de entregas de hitos

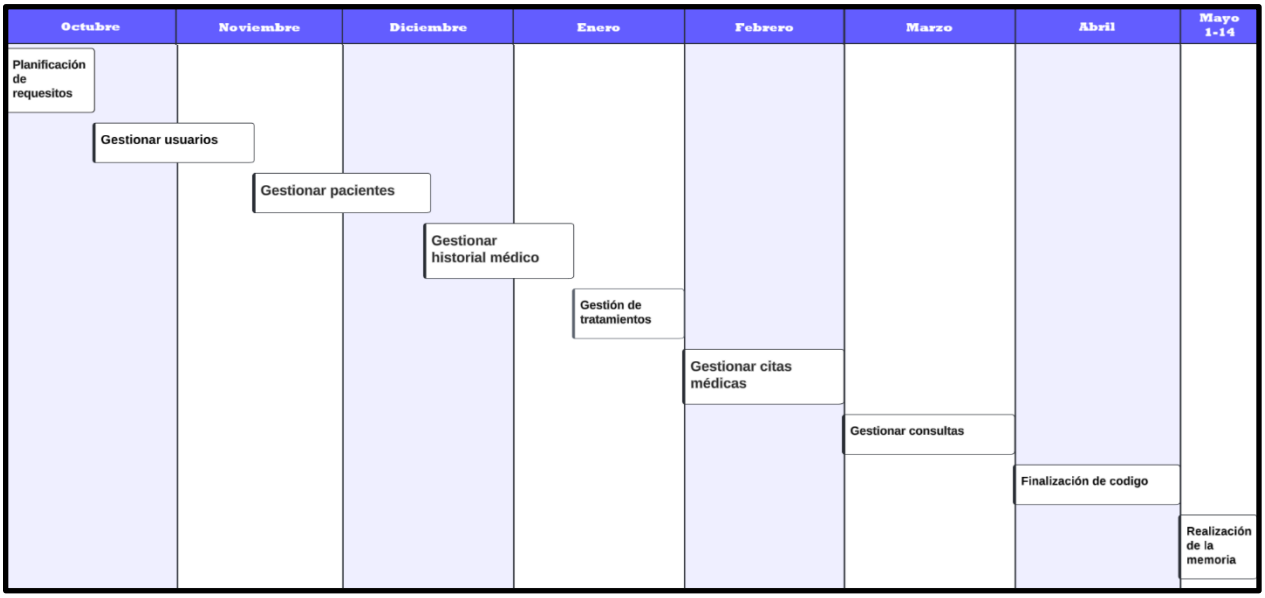

Figura 1.1- Mapa de plan de trabajo

A continuación, se explican los hitos que se han llevado a cabo a lo largo del desarrollo del proyecto.

Planificación de requisitos: El hito inicial del proyecto, se dedicó a la creación de los casos de uso que engloban las acciones de los usuarios de la aplicación. Se coordinó una reunión con el tutor para debatir y confirmar estos requisitos.

Gestión de usuarios: Este segundo hito se concentra en permitir a los usuarios realizar operaciones básicas de registro, inicio de sesión, cierre de sesión, modificación y eliminación de perfil.

Gestión de pacientes: En este tercer hito, el enfoque se dirige a los médicos, quienes desde la aplicación web podrán poder agregar, editar y eliminar a pacientes a sus asignaciones. Además, podrán listar, buscar, ver información básica, filtrar y ordenar entre todos sus pacientes.

Gestión de historial médico: En este cuarto hito, se espera que los doctores puedan crear, ver, modificar y eliminar los historiales médicos de sus pacientes a través de la aplicación web.

Gestión de tratamientos: En este quinto hito, se prevé que los doctores puedan recetar medicaciones con un tratamiento mediante aplicación web y que los pacientes puedan recibir dichos tratamientos y recibir las alarmas correspondientes.

Gestión de citas médicas: En este sexto hito, se espera que tanto desde la aplicación web como móvil se puedan proponer o crear citas, confirmarlas, cancelarlas y observarlas a través de un calendario u horario.

Gestión de consultas: En este séptimo hito, los pacientes crear una consulta con sus doctores desde la aplicación móvil, permitiendo una comunicación fluida entre ambas partes a través de las consultas.

Finalización de código: En este octavo hito, se ha llevado a cabo la fase de prueba exhaustiva de ambas aplicaciones en su totalidad. Se han evaluado los posibles fallos y se ha verificado que todas las funcionalidades estén implementadas de manera correcta.

1.4 Estructura de la memoria

La memoria del TFG se ha dividido en diferentes capítulos, los cuales corresponden a los siguientes: La portada del TFG, el apartado de dedicatoria y agradecimientos, un resumen que cuenta en qué consiste la aplicación desarrollada, el índice global donde se pueden ver los diferentes capítulos que formar parte de la memoria del proyecto como el capítulo 1 donde se realiza la introducción que engloba la motivación, objetivos, plan de trabajo seguido y la estructura de esta memoria, el capítulo 2 donde explican aplicaciones cuya funcionalidad es similar a la aplicación diseñada, el capítulo 3 donde se exponen las tecnologías utilizadas durante el desarrollo del código, el capítulo 4 donde se explica la arquitectura de ambas aplicaciones, del servidor y de la base de datos, el capítulo 5 en el que se detalla la implementación y diseño de la aplicación móvil, el capítulo 6 en el que se expone la implementación y la aplicación de la aplicación web, y el capítulo 7 donde se documentan las conclusiones y el trabajo futuro a desarrollar y las 3 últimas partes de esta memoria son apéndices donde se encuentran todos los casos de uso y los manuales de usuario tanto de la aplicación web como del móvil.

Chapter 1 - Introduction

1.1 Motivation

The motivation of this Final Degree Project arises from the need to overcome access barriers and improve people's quality of life through adequate medical care.

There are many difficulties that doctors and patients face. These difficulties include the lack of time for travel, the lack of familiarity with technology among certain sectors of the population, and geographical limitations, especially for the people with reduced mobility.

The solution designed to address these challenges consists of the development of a mobile application aimed at patients, complemented by a web platform designed specifically for doctors. These tools are created with the purpose of facilitating of optimizing communication for both parties.

1.2 Objective

The objective of this final project is to develop an application capable of facilitating use for patients and doctors, improving communication between them, and optimizing the medical care process. To achieve this objective, the following milestones were created:

- Investigate the difficulties: The aim is to identify the main needs of both doctors and patients to determine the most common demands and focus on addressing them effectively. This research helps to understand and guide the development of the application.
- Design the web and mobile application: The second objective is to design both applications to ensure effective communication. Both will have a simple and intuitive structure so that all types of users can navigate freely regardless of their level of technological experience.

- Ensure data security and privacy: The third objective is to ensure that personal data is stored securely and that there are no vulnerabilities that make this data accessible.
- Develop tests: The fourth objective is to develop tests for the various functionalities to ensure that users will not encounter any problems.
- Test the application in a part of the medical sector: The fifth objective is to identify a small part of the medical sector, such as a hospital or a health center, to distribute the application and provide assistance if necessary.
- Analyze the results: The final objective is to analyze the data collected during the tests and use this information to make improvements. This analysis will evaluate the application's performance in terms of usability, effectiveness, and user satisfaction. Based on this data, continuous adjustments and improvements will be made to optimize the application's functionality.

1.3 Workplan

This section shows the work plan to follow to achieve the objectives described in the previous section. At the beginning of the Final Degree Project, the milestones were separated and approximate recommended delivery dates were established for each one (see Table 1.1- en and Figure 1.2- en).

Milestones	Date of delivery
Requirements planning	Mid-October
User Management	Mid-November
Patient management	Mid-December
Medical history management	Mid-January
Treatment management	End of January
Medical appointment management	End of February
Query management	End of March
Code Completion	End of April
Memory realization	May 14th

Tabla 2- en. Milestone delivery date

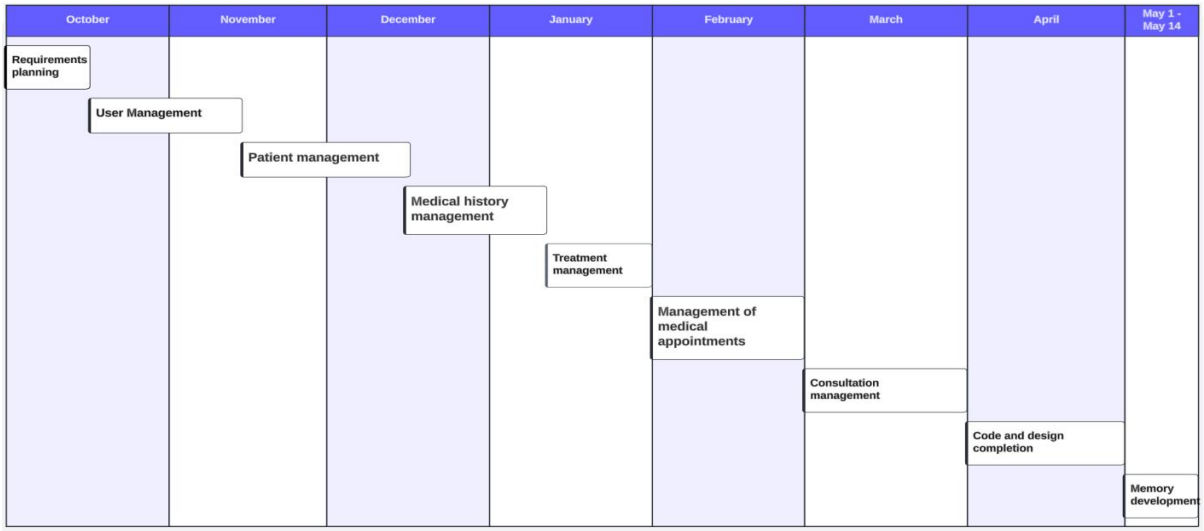

Figura 1.2- en. Work plan map

The milestones accomplished throughout the project's development are explained below.

Requirements Planning: The initial milestone of the project focused on creating use cases that encompass the actions of the application's users. A meeting was coordinated with the advisor to discuss and confirm these requirements.

User Management: In this second milestone, we focused on enabling users to perform basic operations such as registration, login, logout, profile modification, and deletion.

Patient Management: In this third milestone, the focus was directed towards doctors, who, through the web application, will be able to add, edit, and remove patients from their assignments. Additionally, they can list, search, view basic information, filter, and sort through all their patients.

Medical History Management: In this fourth milestone, doctors are expected to create, view, modify, and delete their patients' medical histories through the web application.

Treatment Management: In this fifth milestone, doctors are anticipated to prescribe medications with treatment plans via the web application, and patients will be able to receive these treatments and the corresponding alerts.

Medical Appointment Management: In this sixth milestone, it is expected that both the web and mobile applications will allow users to propose or create appointments, confirm, cancel, and view them through a calendar or schedule.

Consultation Management: In this seventh milestone, patients will be able to create consultations with their doctors from the mobile application, enabling smooth communication between both parties through the consultations.

Code Completion: In this eighth milestone, an exhaustive testing phase of both applications has been carried out. Potential issues have been evaluated, and it has been verified that all functionalities are correctly implemented.

1.4 Memory Structure

The final project report is divided into different chapters, which correspond to the following: the cover page of the final project, the dedication and acknowledgements section, a summary describing the developed application, the global index showing the various chapters that form part of the project report such as Chapter 1, which contains the introduction covering the motivation, objectives, work plan followed, and the

structure of this report; Chapter 2, which explains applications with functionality similar to the designed application; Chapter 3, which presents the technologies used during code development; Chapter 4, which explains the structure of both applications, the server, and the database; Chapter 5, which details the implementation and design of the mobile application; Chapter 6, which details the implementation and design of the web application; and Chapter 7, which documents the conclusions and future work to be developed. The last three parts of this report are appendices, which include all the use cases and user manuals for both the web and mobile applications.

Capítulo 2 - Estado de la Cuestión

En este capítulo se van a exponer y explicar brevemente las diferentes aplicaciones en las que se ha inspirado para elaborar la aplicación.

2.1.1 Whatsapp

WhatsApp [1] es una aplicación de mensajería instantánea, ampliamente utilizada a nivel mundial. Lanzada en 2009, permite a los usuarios enviar mensajes de texto, realizar llamadas de voz y video, compartir imágenes, videos, documentos, ubicaciones y contactos.

WhatsApp utiliza un sistema de notificaciones para alertas a los usuarios sobre nuevos mensajes, llamadas, o cualquier mención. Estas notificaciones aparecen en la pantalla del dispositivo incluso cuando la aplicación no está en uso.

2.1.2 MyChart

MyChart [2] es un portal de salud desarrollado por Epic Systems, que permite a los pacientes acceder a sus registros médicos y comunicarse con profesionales de la salud. A través de esta plataforma, los pacientes pueden ver sus historiales médicos, resultados de pruebas y diagnósticos, y gestionar citas médicas. También proporciona recordatorios de citas y medicamentos, y permite realizar visitas virtuales.

2.1.3 HealthTap

HealthTap [3] es una plataforma de telemedicina que conecta a los pacientes con los médicos para consultas en línea, ofreciendo una manera conveniente y rápida de recibir atención médica. A través de HealthTap, los pacientes pueden realizar consultas médicas por videollamadas, llamadas de voz o mensajes de texto, y obtener respuestas a preguntas médicas en tiempo real. Los médicos pueden emitir recetas electrónicas y proporcionar planes de tratamiento personalizas.

2.1.4 Zocdoc

Zocdoc [4] es una plataforma que facilita a los pacientes a encontrar médicos y programar citas de manera rápida y sencilla. A través de Zocdoc, los pacientes pueden buscar médicos por especialidad, ubicación y disponibilidad, leer reseñas y calificaciones de otros pacientes, y reservar citas en línea. La plataforma también envía recordatorios automáticos de citas por correo electrónico y mensajes de texto. También proporciona información detallada sobre los médicos, incluyendo su formación, experiencia y especialidades, lo que ayuda a los pacientes a tomar decisiones informadas sobre su atención médica.

2.1.5 Practice Fusion

Practice Fusion [5] es un sistema de registros médicos electrónicos (EMR) basado en la nube, ideal para pequeñas y medianas prácticas médicas. Este sistema permite a los médicos gestionar los registros de sus pacientes, incluyendo información demográfica, historiales médicos y notas de consulta.

Practice Fusion facilita la programación y gestión de citas médicas, la emisión de recetas electrónicas directamente de las farmacias, y proporciona una interfaz intuitiva y fácil de usar. Además, ofrece herramientas de reportes y análisis para mejorar la gestión de la práctica médica y la atención al paciente, y cuenta con una comunidad en línea para que los usuarios compartan experiencias y mejores prácticas.

Capítulo 3 - Tecnología Empleada

En este capítulo se van a tratar las diferentes tecnologías usadas en el proceso de desarrollo de la aplicación. Se expondrán las diferentes herramientas y lenguajes de programación utilizados para los dos *fronts*: la aplicación híbrida móvil y la aplicación web, así como en el *backend*.

3.1 Lenguajes de Programación

En este apartado se van a tratar los diferentes lenguajes de programación utilizados en la aplicación.

3.1.1 JavaScript

JavaScript [6] es un lenguaje destacado en el desarrollo web. Suele utilizarse para crear y controlar la carga dinámica en el lado del cliente, es decir, en el navegador web del usuario. En otras ocasiones, como en esta aplicación, se utiliza en el lado del servidor a través del *framework Express.js.*

En el lado del servidor, se ha utilizado para: la gestión de solicitudes o peticiones *GET* o *POST*, la configuración de *middlewares, el* control y acceso a la base de datos, el envío de correos electrónicos, entre otros.

Para la lectura y almacenamiento de los archivos y carpetas se han utilizado las librerías *fs[7]* y *path[8]* de *node.js*.

En el lado del cliente, se ha utilizado para la carga dinámica de datos en las pantallas del calendario, citas, gestión de usuarios, creación de historial médico y otros pequeños scripts. En el calendario concretamente, se ha utilizado la librería externa *FullCalendar.js* [9] de *JavaScript* y para la creación del historial médico se ha utilizado *PDFKit*[10].

3.1.2 Cascading Style Sheets (CSS)

CSS [11] es un lenguaje utilizado para el diseño de páginas web elaboradas con *HTML*. Se ha utilizado en la mayoría de las vistas para mejorar la estética del proyecto, esto implica definir atributos como el diseño de los iconos, las dimensiones de los títulos, tipografía, colores, estilizar mensajes y colores de fondo.

Dentro del proyecto web, se encuentra en la carpeta "public/stylesheets/style/", y dentro de cada *HTML* se llama a este archivo para poder modificar los atributos deseados. Además, se ha utilizado la librería *CSS* y *JavaScript* de *Bootstrap*.

3.1.3 HyperText Markup Language (HTML)

HTML [12] es un lenguaje de marcado utilizado para la creación de páginas web. Se ha usado para definir la estructura inicial de las páginas, su contenido, la barra de navegación, los botones, el pié de página, entre otras. A su vez, esta estructura se puede estilizar y manipular mediante el uso de *CSS* para aspectos visuales y es capaz de añadir interactividad y funcionalidad dinámica con JavaScript.

3.1.4 Java

Java [13] es el lenguaje de programación orientado a objetos, utilizado junto con *Kotlin* en *Android Studio* para desarrollar aplicaciones.

Se optó por *Java* debido a su amplio uso por desarrolladores y la existencia de una amplia familiaridad con este lenguaje obtenido a lo largo de la carrera.

Se ha utilizado en el entorno de la aplicación móvil para conectar la implementación de las funcionalidades con los componentes definidos en el *XML*, gestionar la creación de alarmas, notificaciones y además también se usa para enviar información mediante solicitudes *HTTP* al servidor *node.js*.

3.1.5 Extensible Markup Language (XML)

XML [14] es un lenguaje de marcado utilizado para definir la estructura y el contenido de diversos elementos en el desarrollo de aplicaciones Android.

Se ha empleado para definir la interfaz de usuario. Los archivos *XML* de diseño guardados dentro de la carpeta "res/layout/", se utilizan para definir componentes como pantallas, fragments, mensajes de las consultas, mensajes que muestran los datos de una cita o una consulta, cuadros de texto y botones entre otros.

También se pueden definir recursos estáticos como cadenas de texto y colores en la carpeta "res/values/", iconos en la carpeta "res/drawable/", y el sonido de la alarma en la carpeta "res/raw/".

En todas las aplicaciones hay un archivo de configuración llamado *AndroidManifest.xml*.

3.2 Herramientas/Entornos de Desarrollo

En este apartado se van a tratar las diferentes herramientas y entornos de desarrollo utilizados en el proyecto.

3.2.1 Node.js

Es un entorno de ejecución de JavaScript que permite diseñar la lógica de la aplicación del lado del servidor. Se utiliza principalmente para crear la aplicación web usando tecnologías como *HTML*, *CSS* y *Javascript*.

Node.js [15] es la plataforma base sobre la que se ejecutan paquetes con "npm install <paquete>" o *frameworks* como *Express.js*, que facilita plantillas, estructura de carpetas básicas, manejo del enrutamiento y gestión de solicitudes y respuestas *HTTP*.

Con *node.js* se pueden instalar dependencias como *express-session* que permite la creación y gestión de sesiones con los usuarios que interactúan con la aplicación, o como *nodemailer* para gestionar el envío de correos electrónicos.

Se da uso de *node.js* también para conectar el servidor con la base de datos *MySql* utilizando "npm run start" previamente habiendo instalado su dependencia con "npm install mysql".

3.2.2 Bootstrap

Bootstrap [16] es un *framework front-end* ampliamente utilizado para el desarrollo de sitios web y aplicaciones web. Se consideró oportuno usarlo debido al ahorro de tiempo y a la facilidad que proporciona al crear interfaces de usuario adaptables. Esta librería ofrece componentes pre-estilizados como botones, formularios, barras de menú y otros elementos que son visualmente atractivos.

Bootstrap utiliza *HTML*, *CSS* y *Javascript* para su implementación.

3.2.3 Embedded JavaScript (EJS)

EJS [17] es un motor de plantillas que posibilita la generación de *HTML* dinámico usando *JavaScript*. La plantilla permite la inserción de fragmentos de código *JavaScript* dentro de los archivos de plantilla, facilitando la creación eficiente de contenido web interactivo como listas de paciente, listas de consultas, títulos e implementación de mensajes entre otros.

3.2.4 Visual Studio Code

Visual Studio Code[18] es un editor de código altamente popular y gratuito desarrollado por *Microsoft*. Se optó por utilizar este entorno debido a la facilidad que brinda a la hora de escribir código, de depurar y de gestionar proyectos. Además de que se cuenta con experiencia previa en el uso de este editor, también se eligió por la comodidad que proporciona respecto a la integración y colaboración con *GIT*.

3.2.5 Android Studio

Android Studio [19] es el entorno de desarrollo por excelencia para aplicaciones *Android*. Se utilizó este entorno debido a la gran variedad de herramientas y funcionalidades avanzadas que proporciona, las cuales facilitan la creación de una aplicación móvil para *Android*. En particular resultó sencillo el uso de sus emuladores móviles, que ha permitido probar la aplicación de manera precisa y eficaz.

3.2.6 MySQL

MySQL [20] es una base de datos relacional, donde se guarda toda la información de la aplicación tanto web como de móvil, al ser relacional implica que la información se organiza en tablas compuestas por filas y columnas. Estas tablas contienen datos específicos como pacientes, citas, tratamiento y doctores, entre otros, donde en el caso de pacientes, se especifican atributos como el nombre, apellidos, email, código postal, fecha de nacimiento, dni, dirección y contraseña de cada paciente, lo mismo sucede con el resto de las tablas.

3.2.7 Xampp

Xampp [21] es un paquete de software libre que proporciona un entorno de desarrollo que incluye *MySQL* como gestor de base de datos, esto ha permitido probar la aplicación y manejar datos localmente de manera eficiente.

Capítulo 4 - Arquitectura de la Aplicación y modelo de datos

En este capítulo se describe la arquitectura utilizada y las tablas que forman la base de datos.

4.1 Arquitectura de la aplicación

Durante el desarrollo del proyecto se ha utilizado la arquitectura cliente-servidor, con dos clientes: los pacientes en la aplicación móvil, y los médicos utilizando la plataforma web.

Por la parte de la aplicación de Android, los pacientes realizan peticiones al servidor que se encuentra alojado en la parte de la página web.

Por parte de la plataforma web, los médicos también hacen solicitudes o peticiones y reciben respuestas del servidor.

En cuanto al servidor se utiliza XAMPP, que permite trabajar localmente. En este entorno, Apache actúa como el servidor web que maneja las solicitudes HTTP entrantes desde los clientes, mientras que MySQL, por otro lado, actúa como el sistema de gestión de bases de datos que almacena y gestiona los datos de la aplicación. Los datos de la aplicación pueden verse representados en la figura 4.1. con un modelo entidad-relación.

En resumen, los clientes son los que toman la iniciativa al enviar solicitudes al servidor. El servidor, por su parte, permanece a la espera de recibir estas solicitudes de los clientes. Una vez llegan, el servidor procesa y envía las respuestas de vuelta a los clientes que, en este caso, son tanto los médicos como los pacientes.

Esta configuración se puede observar en la figura 4.2.

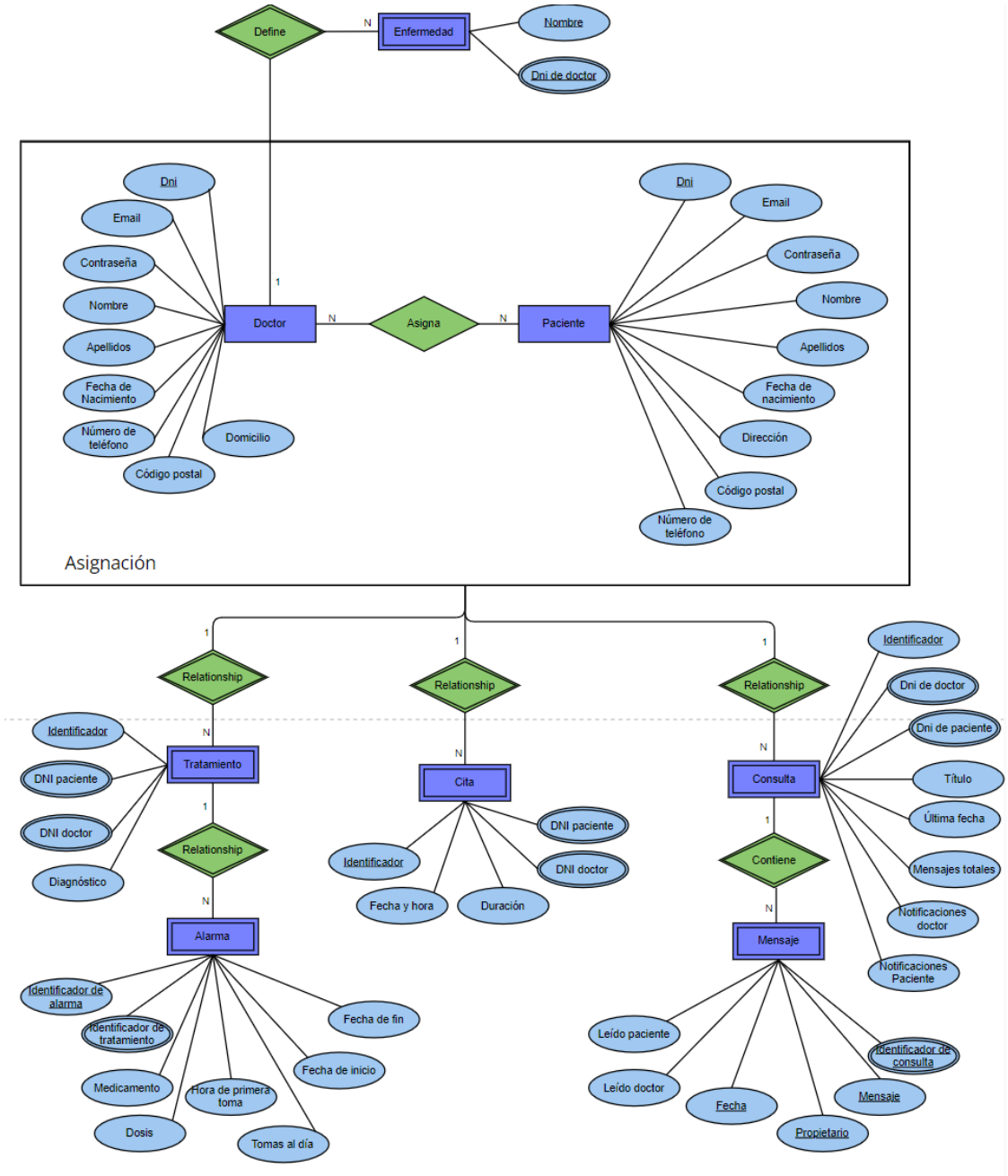

Figura 4.1- Modelo entidad - relación de la aplicación

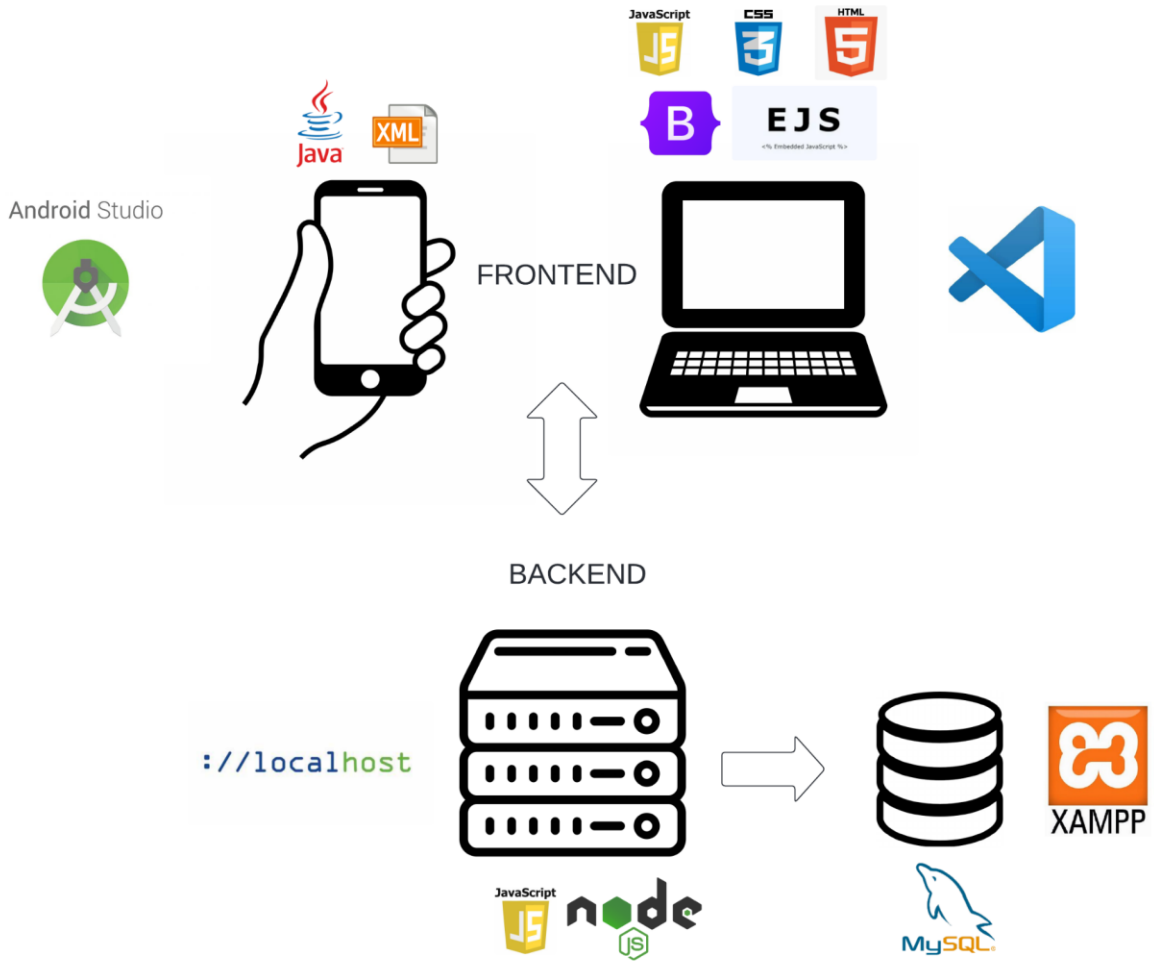

Figura 4.2- Arquitectura y herramientas de la aplicación

4.2 Estructura de la Base de Datos

Tanto la aplicación web como la aplicación móvil requieren un almacenamiento persistente de datos para su funcionamiento. Para esta labor se decidió *XAMPP*, un paquete de *software* libre que se ha usado en la aplicación como sistema de gestión de bases de datos *MySQL*. Los datos que se almacenan en tablas *SQL* siguen ciertas reglas.

1. Cada tabla debe tener una clave única que sea capaz de identificar cada dato de dicha tabla.
2. Cada dato debe almacenarse con un tipo de variable que varían entre:

- *Int*: Un número entero, usado para contadores, identificadores y variables binarias.
- *Varchar*: Una cadena de carácteres, utilizada para almacenar nombres,teléfonos y cualquier variable de tamaño pequeño que se represente con carácteres.
- *Longtext* o *mediumtext*: Cadena de carácteres similar a *varchar* pero para datos que requieran un tamaño mayor.
- *Date*, *time* o *datetime*: Diferentes formatos para almacenar fechas y horas.

Adicionalmente las tablas pueden relacionarse entre sí.

La base de datos consta de un total de 11 tablas, cada una con sus respectivas claves únicas, variables y relaciones (ver Figura 4.3).

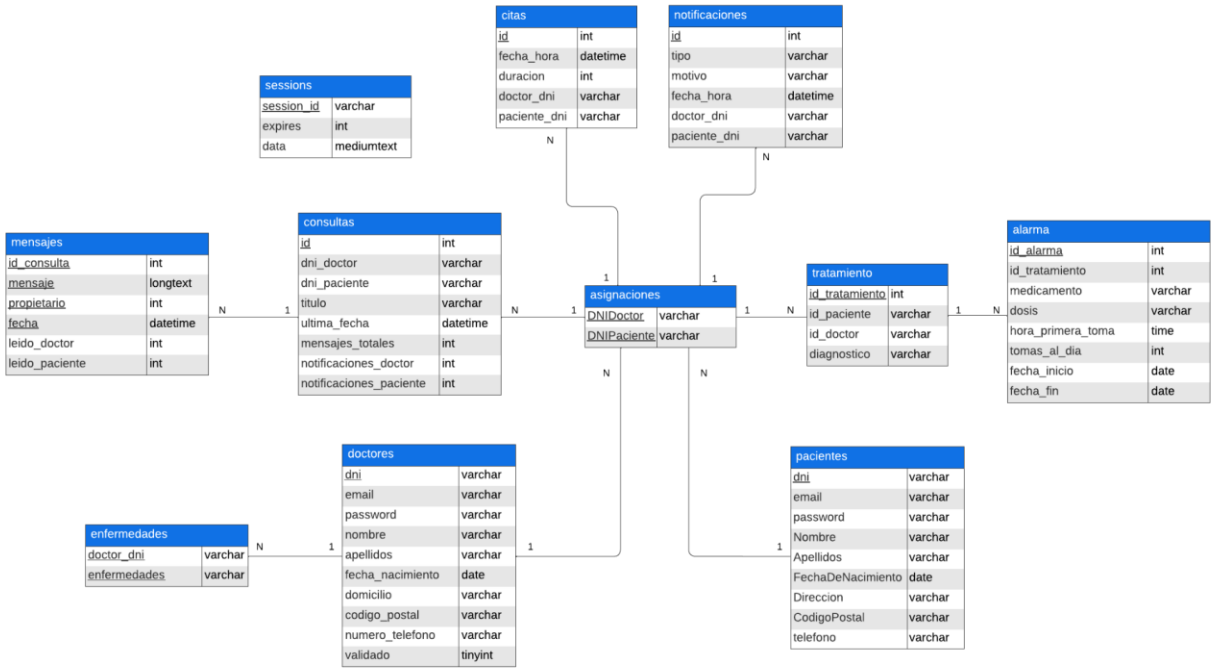

Figura 4.3- Arquitectura de la base de datos. Modelo entidad-relación

4.2.1 Tabla Doctores

En esta tabla (ver figura 4.4) se almacenan los datos de los usuarios definidos como doctores. Se almacena el dni (como clave única), email, contraseña, nombre, apellidos, fecha de nacimiento, domicilio, código postal y número de teléfono.

doctores	
dni	varchar
email	varchar
password	varchar
nombre	varchar
apellidos	varchar
fecha_nacimiento	date
domicilio	varchar
codigo_postal	varchar
numero_telefono	varchar
validado	tinyint

Figura 4.4- Tabla doctores

4.2.2 Tabla Pacientes

En esta tabla (ver figura 4.5) se almacenan los datos de los usuarios categorizados como pacientes. Se almacena el dni (como clave única), email, contraseña, nombre, apellidos, fecha de nacimiento, dirección de domicilio, código postal y número de teléfono.

pacientes	
dni	varchar
email	varchar
password	varchar
Nombre	varchar
Apellidos	varchar
FechaDeNacimiento	date
Direccion	varchar
CodigoPostal	varchar
telefono	varchar

Figura 4.5- Tabla pacientes

4.2.3 Tabla Asignaciones

La tabla asignaciones (ver figura 4.6) representa las asignaciones entre doctores y pacientes. Esta almacena el dni de los doctores y el dni de los pacientes y utiliza ambos como clave única para formar una clave única compuesta.

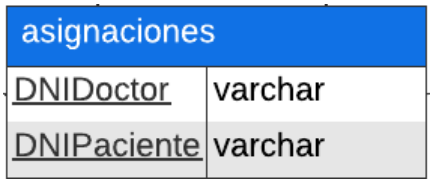

Figura 4.6- Tabla asignaciones

4.2.4 Tabla Consultas

En la tabla consultas (ver figura 4.7) se almacenan los datos de cada consulta. Estos datos pertenecientes a cada consulta tienen un identificador como clave única e incluyen el dni del doctor y del paciente, el título, la fecha del último mensaje mandado, el número de mensajes totales y notificaciones.

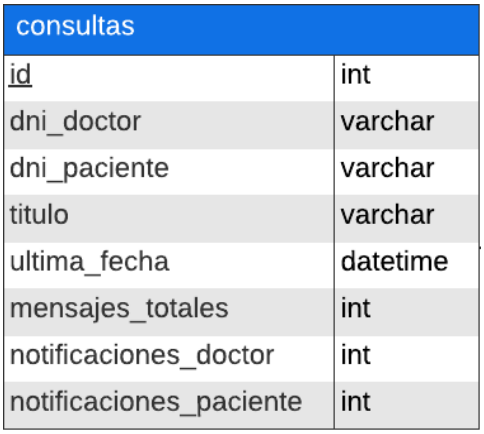

Figura 4.7- Tabla consultas

4.2.5 Tabla Mensajes

En esta tabla (ver figura 4.8) se almacenan los mensajes de las consultas. Se almacena el id de la consulta, el mensaje enviado, propietario del mensaje (1 si el propietario es un paciente y 0 si es un doctor), fecha de envío del mensaje y dos contadores para el de leído para el doctor y el paciente que cuentan el número de

mensajes sin leer de cada uno. Esta tabla tiene el conjunto de id de la consulta, mensaje, propietario y fecha como clave única compuesta.

mensajes	
id_consulta	int
mensaje	longtext
propietario	int
fecha	datetime
leido_doctor	int
leido_paciente	int

Figura 4.8- Tabla mensajes

4.2.6 Tabla Tratamiento

La tabla tratamiento (ver figura 4.9) representa los tratamientos entre doctores y pacientes. Esta almacena un identificador único de cada tratamiento que utiliza como clave única, los identificadores de paciente y doctor (que en el sistema son DNIs) y, finalmente un diagnóstico, que es una breve descripción de un diagnóstico otorgado por el doctor.

tratamiento	
id_tratamiento	int
id_paciente	varchar
id_doctor	varchar
diagnostico	varchar

Figura 4.9- Tabla tratamiento

4.2.7 Tabla Alarma

En la tabla alarma (ver figura 4.10) se almacenan los datos de cada alarma. Cada dato tiene un identificador como clave única, un identificador del tratamiento al que pertenecen, medicamento que se receta, la dosis que se receta, la hora de la primera toma, el número de tomas al día y su fecha de inicio y fin.

alarma	
id_alarma	int
id_tratamiento	int
medicamento	varchar
dosis	varchar
hora_primera_toma	time
tomas_al_dia	int
fecha_inicio	date
fecha_fin	date

Figura 4.10- Tabla alarma

4.2.8 Tabla Citas

En esta tabla (ver figura 4.11) se guardan los datos de las citas. Las citas tienen un identificador como clave única, una fecha y hora de inicio de la cita, una duración en minutos de la cita y el dni del paciente y doctor de cada cita.

citas	
id	int
fecha_hora	datetime
duracion	int
doctor_dni	varchar
paciente_dni	varchar

Figura 4.11- Tabla citas

4.2.9 Tabla Notificaciones

La tabla notificaciones (ver figura 4.12) son peticiones del paciente al doctor para organizar diferentes eventos. Posee un identificador como clave única, un tipo que categoriza al evento (Cita es el único evento actualmente disponible), motivo de la petición, fecha del evento y el dni del doctor y del paciente involucrados en dicha petición.

notificaciones	
id	int
tipo	varchar
motivo	varchar
fecha_hora	datetime
doctor_dni	varchar
paciente_dni	varchar

Figura 4.12- Tabla notificaciones

4.2.10 Tabla Enfermedades

En la tabla enfermedades (ver figura 4.13) almacena las enfermedades que se muestran para asignar en el historial médico. Su clave única es la unión del dni del doctor y el nombre de la enfermedad.

enfermedades	
doctor_dni	varchar
enfermedades	varchar

Figura 4.13- Tabla enfermedades

4.2.11 Tabla Sessions

En esta tabla (ver figura 4.14) se guarda toda la información necesaria para cada sesión de usuario. Posee un identificador de sesión como clave única, un número que indica la expiración de la sesión y por último una variable que almacena aquellos datos que se quieran mantener en cada sesión. Estos datos se guardan en un mismo campo con una estructura ". json".

sessions	
session_id	varchar
expires	int
data	mediumtext

Figura 4.14- Tabla sessions

Capítulo 5 - Implementación, Estructura y Diseño de la Aplicación Móvil

En esta sección se exponen los detalles de la implementación, la estructura y el diseño de la aplicación móvil, organizando las diversas funcionalidades en los siguientes módulos funcionales:

- Módulo Gestión de usuarios
- Módulo Gestión de citas
- Módulo Gestión de tratamientos
- Módulo Gestión de consultas

5.1 Diseño de la Aplicación Móvil

La aplicación móvil de *Android* presenta un diseño sencillo y altamente intuitivo. Ha sido diseñado para garantizar la accesibilidad y sencillez para personas de cualquier edad y nivel de experiencia. Se busca que la persona se sienta cómoda y pueda visualizar al momento las funciones de la aplicación sin causarle ningún tipo de confusión.

La elección de la paleta de colores se ha centrado en la simplicidad y efectividad visual.

Se ha optado por tonos sencillos como un azul oscuro y blanco. Este último por un lado ayuda a que el contenido sea fácilmente legible, mientras que el azul oscuro por otro lado es un buen contraste con el blanco mejorando aún más la legibilidad del texto. Además, estos colores se han extendido del icono de la aplicación, el cual también presenta tonalidades para crear una sensación de profesionalidad y confianza.

En determinadas ocasiones se presentan colores como el negro en cuadros texto para que resalte con el blanco y sea legible, por ejemplo, los botones de "¿Has olvidado la contraseña?" y "Cambiar contraseña" están en rojo para resaltar dicha funcionalidad, ya que la pérdida o el olvido de una contraseña puede ser una situación urgente para los pacientes y tienen que ver que estas opciones están fácilmente

disponibles con el objetivo de que puedan proteger sus cuentas ante cualquier eventualidad.

En el caso de las consultas con los doctores, se usa un color amarillo suave de fondo en el nombre del doctor, ya que este color transmite una sensación de amabilidad y cercanía con él, además de crear un contraste suave con los colores de los mensajes que son azules de diferentes tonos para que se diferencien fácilmente entre sí. El nombre del doctor en este caso está en negro ya que presenta un contraste agradable respecto al amarillo suave de fondo.

En cuanto a la navegación por la aplicación, una vez se inicie sesión, se muestra una estructura clara y sencilla en la que se observa una barra de menú con tres botones situada en la parte inferior de la pantalla. En todo momento la barra está presente para poder cambiar de vista cuando se desee.

En la parte superior de las pantallas se encuentra el icono de la aplicación, el nombre del paciente, y un botón con el icono de tres puntos colocados verticalmente para indicar que se despliega un menú. En el caso de las consultas con los doctores, no se muestra ni la barra inferior del menú, ni la barra superior para mayor comodidad a la hora de escribir y leer mensajes.

5.2 Estructura en la Aplicación Híbrida de Android

En la aplicación móvil se han estructurado las carpetas como se ven en la Figura 5.1 Utilizando *gradle* que permite añadir dependencias al proyecto, y construir la aplicación a partir de los ficheros.

Dentro de la aplicación se hace una primera división en tres paquetes: *manifest, java y res.*

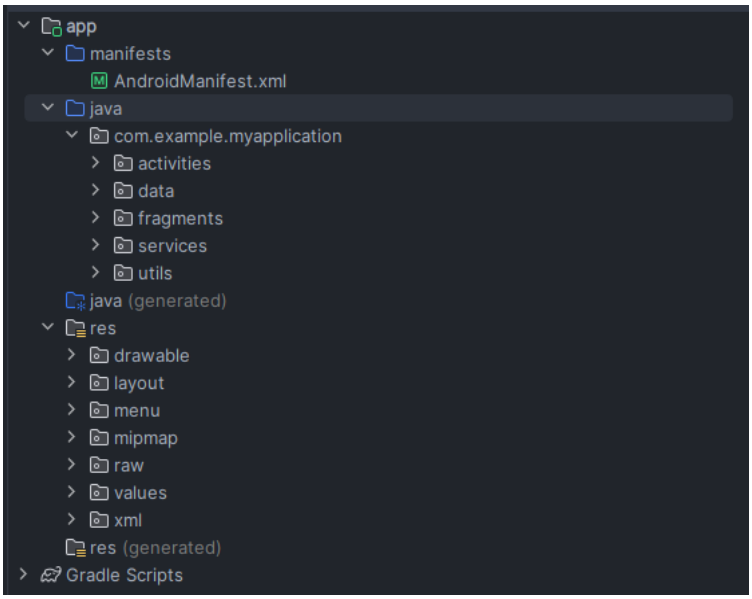

Figura 5.1- Estructura de la aplicación móvil

Manifest contiene un fichero *XML* del mismo nombre con una serie de configuraciones para la aplicación: en ella se indican los permisos que debe tener, como por ejemplo acceso a internet, vibración, mandar notificaciones, acceder al calendario, entre otras(ver figura 5.2); se establecen también las diferentes actividades de la aplicación indicando cual es la primera en ejecutarse; se indican también los elementos receptores que reciben un evento que ocurre en el móvil, como por ejemplo el evento de parar el sonido de la alarma(ver figura 5.3); y por último, la configuración de una serie de servicios que lanzan un evento en el móvil, como por ejemplo para lanzar una notificación(ver figura 5.4).

```
<uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
<uses-permission android:name="android.permission.WRITE_CALENDAR" />
<uses-permission android:name="android.permission.READ_CALENDAR" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
<uses-permission android:name="com.google.android.gms.premission.ACTIVITY_RECOGNITION" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission android:name="android.permission.INTERNET" />
```

Figura 5.2- Manifest XML permisos

```
<receiver
    android:name=".utils.receiver.NotificationButtonReceiver"
    android:enabled="true"
    android:exported="false" >
    <intent-filter>
        <action android:name="ACTION_STOP_MEDIA_PLAYER" />
    </intent-filter>
</receiver>
```

Figura 5.3- Manifest XML receptor de un evento

```
<service
    android:name=".utils.async.NotificarMensajesAsync"
    android:enabled="true"
    android:exported="false"
    android:permission="TODO">
</service>
```

Figura 5.4- Manifest XML servicio para lanzar una notificación

En la carpeta *Java* se encuentran los ficheros ".java" de la aplicación, que se han organizado siguiendo un modelo vista controlador.

Las clases *java* asociadas a las diferentes vistas de la aplicación se encuentran dentro de las carpetas "activities/" y "fragments/" (ver Figura 5.5).

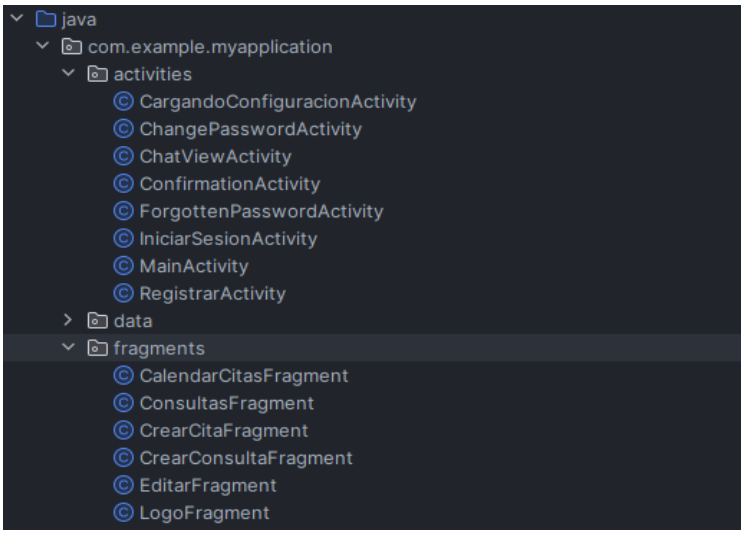

Figura 5.5- Manifest XML servicio para lanzar una notificación

Los *fragments* son ciertas partes de una pantalla o actividad que van cambiando sin la necesidad de refrescar toda la pantalla. En este caso, se hace uso de estos para mantener los *navbar* y botones inferiores sin la necesidad de estar configurando estos en cada una de las actividades.

El controlador se encuentra dentro de la carpeta *utils*(ver Figura 5.8), este comunica el modelo con la vista mandando y recibiendo información encapsulada en clases *Java* que se encuentran dentro de la carpeta "data/" (ver Figura 5.6).

Figura 5.6- Estructura de la aplicación móvil carpeta data

El modelo de la aplicación se encuentra dentro de la carpeta "services/" que contiene una serie de clases e interfaces que se encargan de hacer las llamadas *API* al servidor *node.js (ver figura 5.7)*.

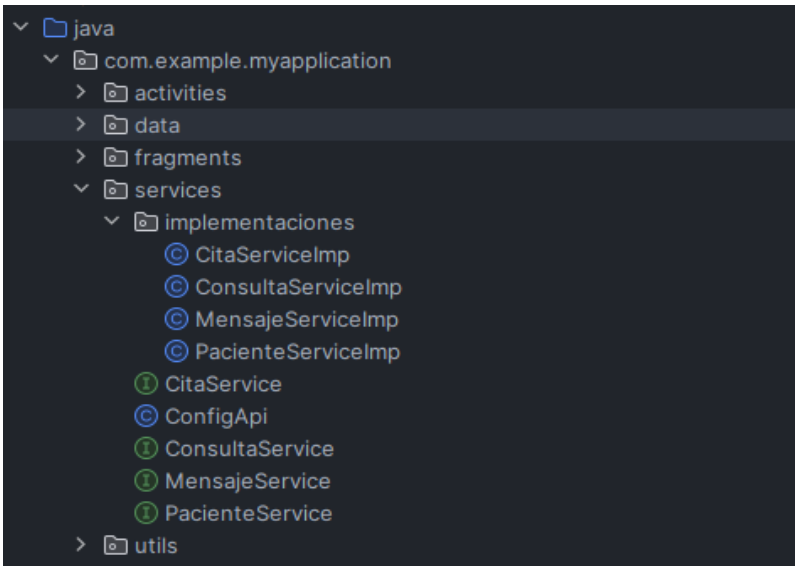

Figura 5.7- Estructura de la aplicación móvil carpeta services

Una vez se ha mencionado las clases que intervienen en el modelo vista controlador, existen otra serie de clases dentro del proyecto en la carpeta de "utils/"(ver figura 5.8) que sirven de utilidad. Dentro de esta se encuentran:

- Los *adapters,* sirven para mostrar una lista de elementos en un *recycler view* en las vistas.
- Clases que ejecutan métodos asíncronos, es decir, en hilos secundarios al principal que permiten mandar peticiones *API* al servidor y dentro del proyecto destacan las clases *ActualizarMensajesAsync* y *NotificarMensajesAsync* que mandan peticiones al servidor para llevar a cabo la gestión del *chat* de consultas en un segundo plano.
- Las clases *manager* ayudan en la gestión del dispositivo móvil. *NotificacionesManager* se encarga de la creación de las notificaciones y sus canales. *NavigationManager* ayuda a encapsular en una sola clase la lógica del cambio de pantallas. Por último, el *SessionManager* se centra en el mantenimiento de una sesión en el dispositivo móvil guardando los datos necesarios del usuario.
- Los *receiver* implementan las acciones a cometer cuando se recibe un evento en el móvil. En el proyecto se tienen tres tipos: el *AlarmReceiver,* que actúa al recibir una alarma, el *InicioMovilReceiver,* que se ejecuta al iniciarse el móvil y el *NotificationButtonReceiver, que se ejecuta cuando el usuario le da al botón de parar una alarma.*

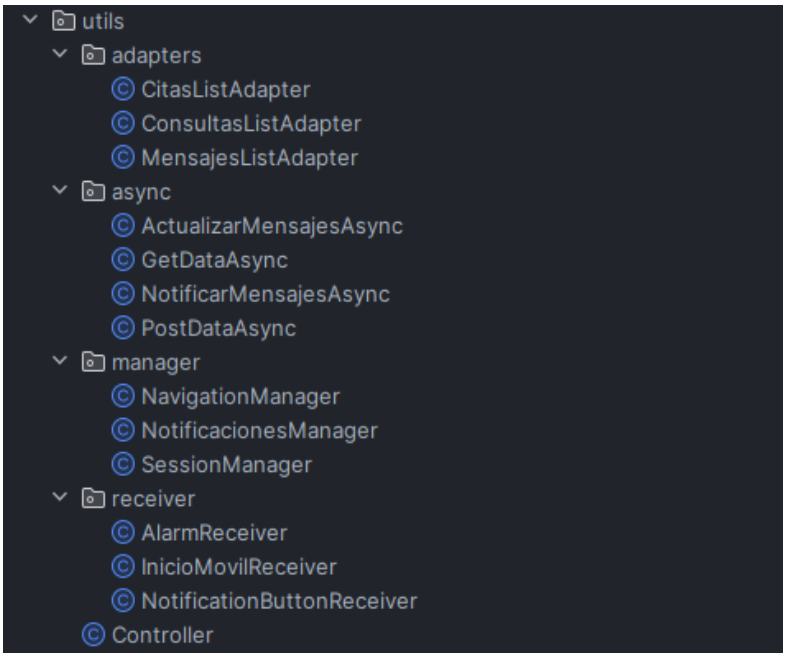

Figura 5.8- Estructura de la aplicación móvil carpeta utils

5.3 Implementación de la Aplicación Móvil

La implementación de la aplicación móvil se ha dividido en los módulos o funcionalidades, previamente mencionados, en los que el actor principal es el paciente.

Las capturas de las pantallas se podrán visualizar en el manual del usuario paciente con el fin de no repetir las imágenes.

5.3.1 Módulo Gestión de Usuarios Paciente

Dentro de esta sección están implementadas las acciones que puede realizar el paciente respecto a la creación y administración del perfil.

5.3.1.1 Iniciar Sesión

Al iniciar la aplicación por primera vez, la primera pantalla que aparece es la de iniciar sesión, en ésta se tienen que introducir los campos que corresponden al dni y a la contraseña, en la figura 5.9 se muestra un trozo del código utilizado para crear el diseño de esta pantalla. El código donde se implementan los botones y los cuadros de texto se muestran en la figura 5.10.

```
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:height="48dp"
            android:text="Iniciar Sesión"
            android:textColor="@color/blue"
            android:textAlignment="center"
            android:textSize="16sp" />
        <EditText
            android:id="@+id/idDNI"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginBottom="5dp"
            android:autofillHints=""
            android:ems="10"
            android:hint="DNI"
            android:inputType="textEmailAddress"
            android:minHeight="48dp"
            tools:ignore="VisualLintTextFieldSize" />
        <EditText
            android:id="@+id/idPassword"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginBottom="5dp"
            android:autofillHints=""
            android:hint="Contraseña"
            android:inputType="textPassword"
            android:minHeight="48dp"
            tools:ignore="VisualLintTextFieldSize" />
        <TextView
            android:id="@+id/crearCuentaId"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:clickable="true"
            android:minHeight="48dp"
            android:text="Crear cuenta"
            android:textAlignment="center"
            android:textColor="@color/blue" />
```

Figura 5.9- Código del diseño de Inicio de Sesión

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_iniciar_sesion);
    sessionManager = new SessionManager( context: this);
    dni = findViewById(R.id.idDNI);
    password = findViewById(R.id.idPassword);

    inicio = findViewById(R.id.editButton);
    inicio.setOnClickListener(v -> {
        if(dni.getText().toString() != "" && password.getText().toString() != ""){
            Controller.getInstance().checkPaciente(dni.getText().toString(), password.getText().toString(), context: this);
        }else{
            makeTextToast("No puede haber ningun campo vacio");
        }
    });
```

Figura 5.10- Código de la implementación de Inicio de Sesión

Hay dos botones más que son "Crear cuenta" y "¿Has olvidado la contraseña?" que se explicarán más adelante. Una vez introducidos los datos, se pulsa el botón "Iniciar Sesión" y el sistema comprueba que los datos introducidos coinciden con alguno de los usuarios de la tabla pacientes de la base de datos. Dentro del método "*checkPaciente(parámetros)*" del controlador es dónde tiene lugar llamada al *service* del paciente (ver figura 5.11) para realizar la llamada *API* al servidor para comprobar las credenciales del usuario como se ve en la figura 5.12. En el punto 1 de esta imagen se observa la *url* del servidor a la que está apuntando dicho método. También se le pasa el dni y la contraseña del usuario.

```
1 usage   Antoranz
public void checkPaciente(String dni, String pass, Context context) {
    PacienteService service = PacienteServiceImp.getInstance();
    service.checkPaciente(dni,pass,context);
}
```

Figura 5.11- Controller - método checkPaciente

```
@Override
public void checkPaciente(String dni, String pass,Context context){
    Executor executor = Executors.newSingleThreadExecutor();
    String urlServidor = ConfigApi.BASE_URL+"pacientes/usuario/checkPaciente";
    SessionManager sessionManager = new SessionManager(context);
    JSONObject postData = new JSONObject();
    try {
        postData.put( name: "dni", dni);
        postData.put( name: "password", pass);
    } catch (JSONException e) {
        e.printStackTrace();
    }
    postDataAsync(urlServidor, executor, (PostDataAsync.OnTaskCompleted) result -> {
        ((Activity) context).runOnUiThread(() -> {
            if (result != null) {
                try {
                    JSONArray jsonArray = new JSONArray(result);
                    if (jsonArray.length() > 0) {

                        JSONObject firstObject = jsonArray.getJSONObject( index: 0);

                        String dniUser = firstObject.getString( name: "dni");
                        String email = firstObject.getString( name: "email");
                        String name = firstObject.getString( name: "Nombre");

                        sessionManager.createSession(dniUser, email,name);
                        NavigationManager.getInstance().navigateToDestination(context,ConfirmationActivity.class);
                    } else {
                        Log.d(TAG, msg: "Error: No data found in the JSON array");
                        Toast.makeText(context, text: "Credenciales incorrectas", Toast.LENGTH_LONG).show();
                    }
                }catch (Exception e){
                    Log.d(TAG, e.getMessage());
                }
            }else{
                Log.d(TAG, msg: "Error deleting account");
```

Figura 5.12- Llamada a la API del servidor - método checkPaciente

Se recibe la petición *API* en el *router* "usuarioP.js" en el servidor(ver Figura 5.13). En la variable "hashedPassword" se añade la frase "caminar es bueno para la salud" para aumentar la seguridad de la contraseña. Esta función del *router* devuelve la información del paciente coincidente de la base de datos con el fin de llevar sus datos de vuelta a la función de "checkPaciente" service del móvil (ver figura 5.12) para continuar con el inicio de sesión.

Por último, se realiza una llamada a la base de datos en el DAO de pacientes como se muestra en la siguiente figura 5.14 para comprobar que efectivamente ese usuario ya está registrado y son correctas sus credenciales.

Si esto se hace correctamente, se realiza la creación de la variable "sessionManager" con el método "createSession(dni, nombre, email)" cogiendo los datos del paciente que se obtienen de la base de previamente (mente (Figura 5.12 punto 2).

```
router.post('/checkPaciente', async function(req, res, next) {
    try {

        var dni=req.body.dni;
        var password=req.body.password;

        var hashedPassword = cifrarContrasena(password,dni + "caminar es bueno para la salud");

        var paciente = await dao.checkPaciente(dni,hashedPassword);

        res.json(paciente)
    } catch (error) {
        console.error("Error durante la operación:", error);
        res.json(null)
      }

});
```

Figura 5.13- Iniciar Sesión - endPoint checkPaciente del router usuarioP

```
checkPaciente(dni,password){

    return new Promise((resolve, reject) => {
        this.pool.getConnection((err, connection) => {
            if(err){
                console.error(`Error al realizar la conexión: ${err.message}`);
                reject(err);
            }else{
                console.log("Exito al conectar a la base de datos");
                var queryCheckPaciente ="SELECT * FROM pacientes WHERE dni = ? AND password = ?"
                connection.query(queryCheckPaciente,[dni,password], (err, res) => {
                    connection.release();
                    if(err){
                        reject(err);
                    }
                    else{
                        resolve(res);
                    }
                });
            }
        });
    });
}
```

Figura 5.14- DAOPacientes - método checkPaciente

5.3.1.2 Registrar

Los usuarios que quieran registrarse como pacientes en la *app*, deberán darle al botón "Crear cuenta" que aparece en la pantalla de inicio de sesión. Si un usuario no está registrado, no podrás iniciar sesión.

Una vez en la pantalla de registro, el usuario deberá rellenar los siguientes campos: email, contraseña, repetir contraseña, nombre, apellidos, dni, fecha de nacimiento, dirección, código postal y número de teléfono.

Todos los campos son obligatorios, además hay restricciones: el dni tiene que ser un dni real y único, el teléfono debe tener 9 dígitos, la fecha debe ser posterior a la fecha actual, el email debe ser válido y único, los campos no pueden estar vacíos y las contraseñas deben coincidir. En el caso de que alguno no se cumpla, aparecerá un mensaje *toast* de dicho error.

Si todos los campos están correctos, se pulsa en "Registrar" y se creará el paciente a la vez que se muestra la pantalla de inicio de sesión de nuevo. En la figura 5.15 se puede observar un trozo del diseño de esta página y en la figura 5.16 se encuentran todas éstas restricciones y la implementación del registro.

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:height="48dp"
    android:text="Información de Login"
    android:textAlignment="center"
    android:textSize="16sp" />
<EditText
    android:id="@+id/idEmail"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="5dp"
    android:autofillHints=""
    android:ems="10"
    android:hint="Email"
    android:inputType="textEmailAddress"
    android:minHeight="48dp"
    tools:ignore="VisualLintTextFieldSize" />
<EditText
    android:id="@+id/idPassword"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="5dp"
    android:autofillHints=""
    android:hint="Contraseña"
    android:inputType="textPassword"
    android:minHeight="48dp"
    tools:ignore="VisualLintTextFieldSize" />
<EditText
    android:id="@+id/idRepitPassword"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="5dp"
    android:autofillHints=""
    android:hint="Repetir Contraseña"
    android:inputType="textPassword"
    android:minHeight="48dp"
    tools:ignore="VisualLintTextFieldSize" />
```

Figura 5.15- Diseño de la pantalla de registro

```
if(!passwordText.getText().toString().equals(repitPasswordText.getText().toString())){
    makeTextToast("Las contraseñas no coinciden");
}else if(passwordText.getText().toString().isEmpty() || emailText.getText().toString().isEmpty()){
    makeTextToast("Ningún campo debe estar vacío");
}else if(!emailText.getText().toString().contains("@")){
    makeTextToast("Email no válido");
}else if(!Objects.equals(Controller.getInstance().obtenerEmailRepetido(emailText.getText().toString()), b: "")){
    makeTextToast("Email ya registrado");
}else if(!esDniValido(dniText.getText().toString())){
    makeTextToast("DNI inválido");
} else if(!Objects.equals(Controller.getInstance().obtenerDniRepetido(dniText.getText().toString()), b: "")){
    makeTextToast("Ese DNI ya está registrado");
}
else if(phoneText.getText().toString().length() != 9){
    makeTextToast("El teléfono debe tener 9 dígitos");
}
else if(nameText.getText().toString().isEmpty() || surnameText.getText().toString().isEmpty() ||
        domicilioText.getText().toString().isEmpty() || postalAddressText.getText().toString().isEmpty()){
    makeTextToast("Todos los campos son obligatorios");
}
else{
    String fechaString = editTextDate.getText().toString();
    DateFormat formatoSalida = new SimpleDateFormat( pattern: "yyyy-MM-dd");
    Date fecha = null;
    Date fechaActual = new Date();
    try {
        fecha = formatoSalida.parse(fechaString);
        if(fecha.after(fechaActual)){
            makeTextToast("La fecha de nacimiento no puede ser posterior a la fecha actual");
            return;
        }
    } catch (ParseException e) {
        throw new RuntimeException(e);
    }
    Paciente paciente = new Paciente(dniText.getText().toString(),
            emailText.getText().toString(),nameText.getText().toString(), surnameText.getText().toString(),
            postalAddressText.getText().toString(), fecha,
            domicilioText.getText().toString(), phoneText.getText().toString(), passwordText.getText().toString()
    );
    Controller.getInstance().registrarPaciente(paciente, context: this);
```

Figura 5.16- Comprobaciones e implementación del registro

Se recibe la petición *API* en el método del *router* "/registrarPaciente" del servidor(ver figura 5.17) en la que se realiza una llamada a la base de datos en el DAO de pacientes para insertar los valores del nuevo paciente como se muestra en la siguiente figura 5.18.

```
router.post('/registrarPaciente', async function(req, res, next) {

    const {Nombre,Apellidos,FechaDeNacimiento,Direccion,CodigoPostal,telefono,email,DNI,password} = req.body;

    try {
        var hashedPassword = cifrarContrasena(password,DNI + "caminar es bueno para la salud");

        await dao.registrarPaciente(Nombre,Apellidos,FechaDeNacimiento,Direccion,CodigoPostal,telefono,email,DNI,hashedPassword);

        res.json(true)
    } catch (error) {
        console.error("Error durante la operación:", error);
        res.json(null)
    }
});
```

Figura 5.17- Registrar paciente - endPoint registrarPaciente del router usuarioP.

```
registrarPaciente(Nombre,Apellidos,FechaDeNacimiento,Direccion,CodigoPostal,telefono,email,DNI,password){

    return new Promise((resolve, reject) => {
        this.pool.getConnection((err, connection) => {
            if(err){
                console.error(`Error al realizar la conexión: ${err.message}`);
                reject(err);
            }else{
                console.log("Exito al conectar a la base de datos");
                var queryRegistrarPaciente ="INSERT INTO pacientes (Nombre,Apellidos,FechaDeNacimiento,Direccion,CodigoPostal,telefono,email,DNI,password) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)"
                connection.query(queryRegistrarPaciente,[Nombre,Apellidos,FechaDeNacimiento,Direccion,CodigoPostal,telefono,email,DNI,password], (err, res) => {
                    connection.release();
                    if(err){
                        reject(err);
                    }
                    else{
                        resolve(res);
                    }
                });
            }
        });
    });
}
```

Figura 5.18- DAOPacientes - método registrarPaciente.

Una vez finalizado el registro, el usuario es redirigido a la pantalla de inicio de sesión donde deberá ingresar las credenciales con las que se acaba de registrar para iniciar sesión.

5.3.1.3 Validar Cuenta

Justo después de iniciar sesión, se cargará una pantalla donde el dispositivo genera un número aleatorio de 5 caracteres(ver figura 5.19) y lo envía a través del controlador al *service* del paciente donde realiza un metodo *POST* como los que se han visto previamente pasando en el cuerpo de la llamada el correo del paciente y el código.

```java
private Integer generarNumeroAleatorio() {
    Random random = new Random();

    Integer numeroAleatorio = random.nextInt( bound: 90000) + 10000;

    return numeroAleatorio;
}
```

Figura 5.19- Generación número Aleatorio

Una vez el *router* recoge los datos y los envía por correo haciendo uso del *mailer*(librería de *node.js*) a través de la función "sendVerificationEmail" dentro del archivo *mailer.js*. Para lograr enviar un correo se ha creado una cuenta de gmail para este proyecto. Dentro de la cuenta se genera una contraseña de la aplicación (ver figura 5.21) y se le pasa a la configuración de una variable "transport" *(ver figura 5.20)* que se usa para enviar el correo.

```javascript
const transporter = nodemailer.createTransport({
  host: "smtp.gmail.com",
  port: 465,
  secure: true,
  auth: {
    user: "medalertatfg@gmail.com",
    pass: [redacted],
  },
});
```

Figura 5.20- Variable "transport"

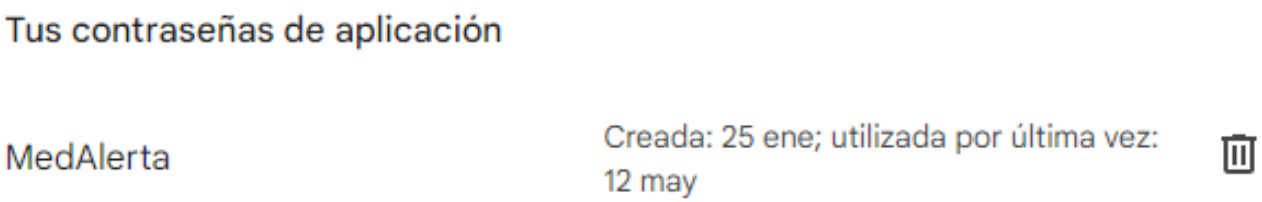

Figura 5.21- Contraseña de aplicación

Dentro de la función mencionada(ver Figura 5.22) "transport" indica:

1. Correo del destinatario pasado

2. Asunto del mensaje
3. El contenido del mensaje en formato HTML. Pasándole el código de verificación.

```
async function sendVerificationEmail(correo_destinatario,contenido) {
    try {
        const info = await transporter.sendMail({
            from: '"MedAlerta" <medalertatfg@gmail.com>',
            to: correo_destinatario,
            subject: "Código de verificación",
            html: generateNotificationHTML(contenido)
        });

        console.log("Correo de notificación enviado:", info);

    } catch (error) {
        console.error("Error al enviar el correo de notificación:", error);
    }
}

function generateNotificationHTML(contenido) {
    return`
        <p>Codigo de verificación: <strong>${contenido}</strong></p>
        <br>
        <p>Atentamente,</p>
        <p>MedAlerta</p>`
    ;
}
```

Figura 5.22- Función sendVerificationEmail

Una vez recibido el correo(ver figura 5.23) el usuario introduce el código en la pantalla y le da a comprobar, si es correcto el usuario accede a la pantalla cargando configuraciones del sistema y por último a la pantalla principal, en caso contrario se le muestra un mensaje de error.

En caso de no recibir un correo, el paciente puede darle al botón de volver a enviar el correo y se volverá a hacer una petición al servidor. Si el envío ha ido bien mostrará un mensaje de correo electrónico enviado a "nombre del correo", en caso

contrario mostrará un mensaje de error.

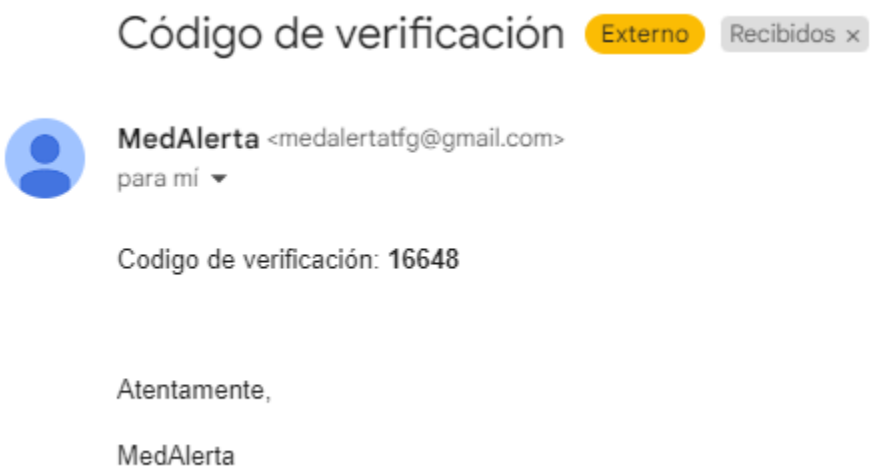

Figura 5.23- Ejemplo correo de verificación

5.3.1.4 Editar perfil

Para editar el perfil del usuario en el móvil, el paciente deberá cambiar los datos de un formulario previamente rellenado por él en el registro. Los datos que se permiten cambiar son: nombre, apellidos, fecha de nacimiento, dirección, código postal y número de teléfono. Como en otros formularios parecidos, como por ejemplo en el registro, todos los datos deben ser válidos. Los datos se envían al servidor a través del controlador que llama al *service* del paciente mediante el método "registrarPaciente". En el servidor se realiza una llamada a la base de datos para modificar mediante una llamada *UPDATE* los datos que tengan el dni del paciente enviado (ver figura 5.24).

```
editarPaciente(Nombre,Apellidos,FechaDeNacimiento,Direccion,CodigoPostal,telefono,DNI){

    return new Promise((resolve, reject) => {
        this.pool.getConnection((err, connection) => {
            if(err){
                console.error(`Error al realizar la conexión: ${err.message}`);
                reject(err);
            }else{
                console.log("Exito al conectar a la base de datos");
                var queryObtenerPaciente ="UPDATE pacientes SET Nombre =?, Apellidos =?,FechaDeNacimiento =?,Direccion =?, CodigoPostal =?,telefono =?WHERE DNI = ?"
                connection.query(queryObtenerPaciente,[Nombre,Apellidos,FechaDeNacimiento,Direccion,CodigoPostal,telefono,DNI], (err, res) => {
                    connection.release();
                    if(err){
                        reject(err);
                    }
                    else{
                        resolve(res);
                    }
                });
            }
        });
    });
}
```

Figura 5.24- Dao pacientes - método editarPaciente

Finalmente los datos son modificados en la base de datos y por consiguiente los datos del móvil también cambian.

5.3.1.5 Eliminar Cuenta

Para eliminar una cuenta una vez se ha pulsado sobre el botón, se le muestra un diálogo de confirmación a la vista donde si lo acepta, envía una petición al *service* a través del controlador para que el servidor borre al paciente de la base de datos (ver figura 5.25).

Dentro del servidor el *router* llama al dao para que mediante sentencias DELETE sobre la base de datos se consigan eliminar todos los datos de un paciente en la tablas (ver figura 5.26), si ocurre un error se le muestra un mensaje de error, en caso contrario te redirige a iniciar sesión y elimina la cuenta del "sessionManager" (ver figura 5.27).

Figura 5.25- Diálogo de confirmación

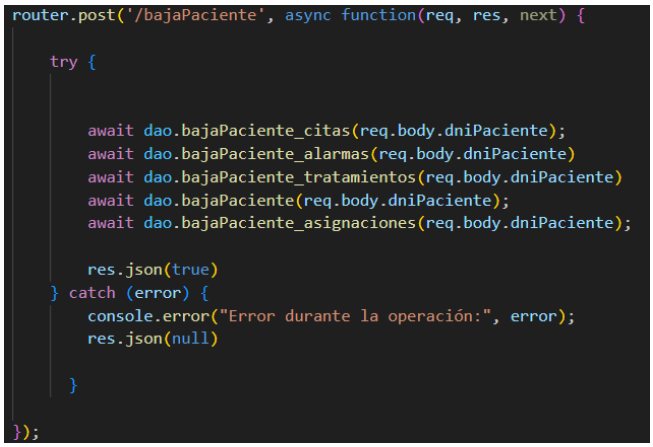

Figura 5.26- Llamadas dato bajaPaciente

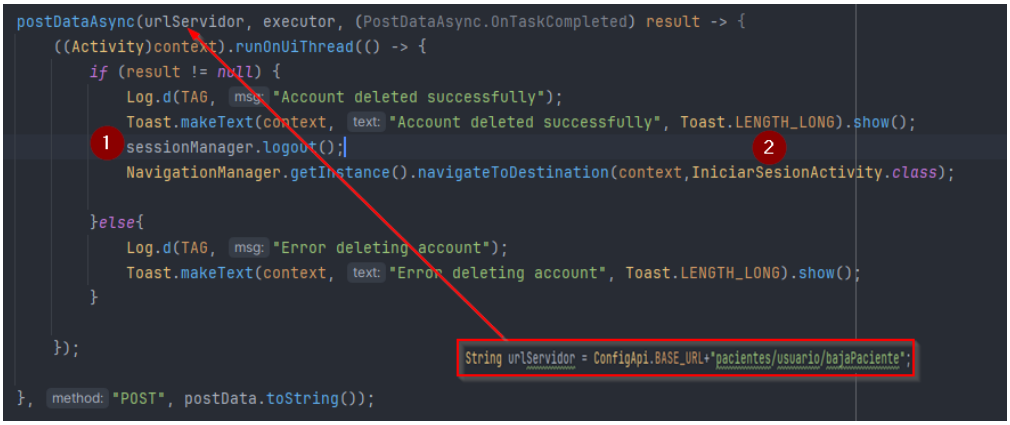

Figura 5.27- Llamada API bajaPaciente.

5.3.2 Módulo Gestión de Citas Paciente

Dentro de esta sección están implementadas las acciones que puede realizar el paciente respecto a la solicitud y administración de citas.

5.3.2.1 Mostrar Citas

En la plantilla *XML* (ver figura 5.28) de la pantalla de citas se destacan dos elementos principales:

1. "CalendarView" para mostrar el calendario del mes, si se hace clic sobre un día mostrará un *dialog* con la información de la cita más temprana ese día y si no un mensaje de no existen citas ese día.
2. Un "recyclerView", que utiliza un *adapter* para mostrar una lista de elementos.

```
<CalendarView
    android:id="@+id/calendarView"                                    1
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    app:layout_constraintTop_toTopOf="parent" />

<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerView"
    android:layout_width="match_parent"
    android:layout_height="0dp"                                       2
    android:layout_marginTop="16dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/calendarView" />
```

Figura 5.28- Código XML Mostrar citas

Al crear la actividad se manda una petición al servidor a través del Controlador para obtener la lista de citas y enviarla al adaptador usado en el "recyclerView" (ver figura 5.29).

```
@Nullable
@Override
public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
    View rootView = inflater.inflate(R.layout.fragment_calendario_citas, container,  attachToRoot: false);

    sessionManager = new SessionManager(requireContext());
    citas = Controller.getInstance().getAllCitas(requireContext(), sessionManager.getUserId());

    RecyclerView recyclerView = rootView.findViewById(R.id.recyclerView);
    recyclerView.setLayoutManager(new LinearLayoutManager(requireContext()));

    CitasListAdapter adapter = new CitasListAdapter(new LinkedList<>(citas), requireContext());
    recyclerView.setAdapter(adapter);

    CalendarView calendarView = rootView.findViewById(R.id.calendarView);
    calendarView.setOnDateChangeListener((view, year, month, dayOfMonth) -> mostrarInformacionDeCita(year, month, dayOfMonth));

    return rootView;
}
```

Figura 5.29- Obtener citas

La clase "CitasListAdapter" contiene una clase "ViewHolder" estática que es la encargada de definir el comportamiento de un elemento. En la figura 5.30 se observa:

1. En el método "OnCreateViewHolder" se vincula el adaptador con el *holder* que va definir el comportamiento del elemento. En este caso "CitasListAdapter.ViewHolder" *(*clase estática dentro de *adapter)* que recibe la referencia a la vista *XML* del *ítem* que se ve en la figura 5.30.
2. El método "OnBindViewHolder" recibe el índice del elemento de la lista de citas actual y llama al método "bind" del *holder* para definir un elemento pasándole los datos de ese elemento (clase cita), este método recibe los datos y modifica los dos *textView* del *ítem* para que muestren la información de la cita que se le ha enviado("getNombre_doctor()+ ' ' + getApellidos_doctor()" y "c.writeDate()").

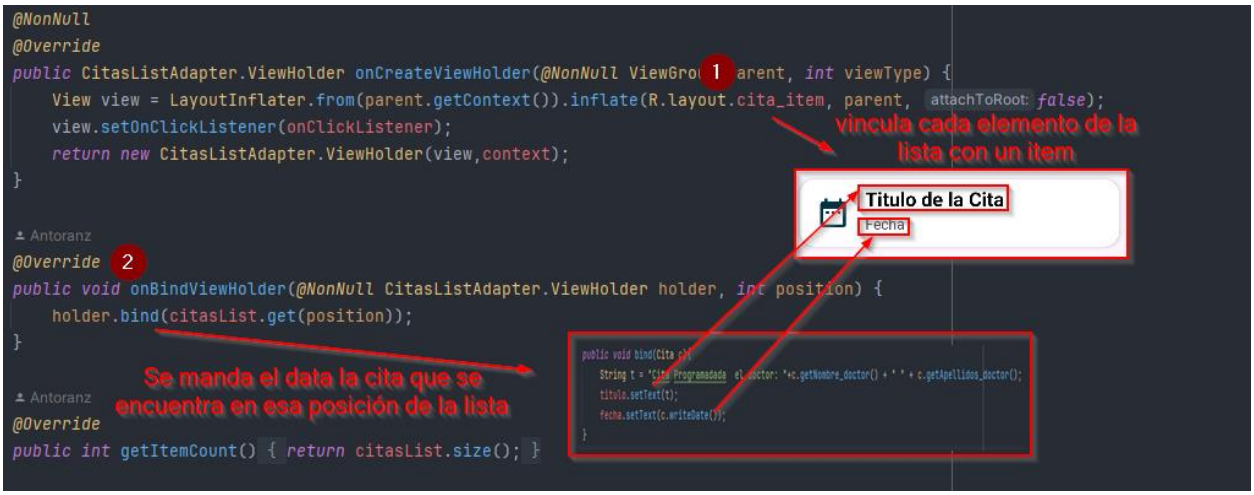

Figura 5.30- Código del adapter de citas

Esta clase permite tener una manera personalizada de mostrar una lista de elementos. En este caso no se define una acción cuando se hace clic sobre el *ítem*, pero más adelante en consultas se mostrará como el *holder* del adaptador de consultas maneja esa posibilidad.

5.3.2.2 Solicitar Cita

El paciente para solicitar una cita ha de rellenar un formulario donde selecciona un doctor de un *spinner* en el que se le muestran todos los doctores con los que está vinculado, en caso contrario se le mostrará un mensaje de que no existen doctores disponibles para crear una cita.

Una vez rellenado los campos fecha, hora y asunto se comprueba que no se manda ningún campo vacío y se envía al service a través del controlador que mediante un método *POST* envía los datos al servidor. Estos datos son guardados en la tabla de notificaciones con el tipo cita.

Una vez finalizado correctamente el guardado en la base de datos, el doctor tendrá accesible la cita para poder aceptarla o rechazarla y en caso que se acepte se le mostrará al usuario la cita en la pantalla de mostrar citas.

5.3.3 Módulo Gestión de Tratamientos Paciente

5.3.3.1 Configurar Alarmas

Se comienza en el momento en que el doctor ha iniciado un tratamiento, que será explicado más en profundidad en la parte de la web.

Una vez creado el tratamiento, la configuración de las alarmas en el móvil comienzan en el momento que el paciente inicie sesión, más concretamente en la actividad "CargandoConfiguracionesActivity". Esta actividad cuenta con una pantalla de carga que muestra una barra de progreso que dura 5 segundos, dentro de la cual hay una función que inicializa las alarmas del paciente (ver figura 5.31).


```
public class CargandoConfiguracionActivity extends AppCompatActivity {
    2 usages
    private ProgressBar progressBar;
    3 usages
    private int progressStatus = 0;
    5 usages
    SessionManager sessionManager;
    1 usage
    private Handler handler = new Handler();
    sergiosan118 +1 *
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_cargando_configuraciones);
        sessionManager = new SessionManager( context: this);
        if (sessionManager.isLoggedIn()) {
            if(sessionManager.isVerificated()){

                initAlarms();

                Intent intentService = new Intent( packageContext: this, NotificarMensajesAsync.class);
                intentService.putExtra( name: "dni", sessionManager.getUserId());
                startService(intentService);

            }else {
                NavigationManager.getInstance().navigateToDestination( context: this, ConfirmationActivity.class);
            }

        } else {
            NavigationManager.getInstance().navigateToDestination( context: this,IniciarSesionActivity.class);
        }
    }
```

Figura 5.31- CargandoConfiguracionesActivity

Dentro del método "initAlarms()" se llevan a cabo tres acciones:

1. ProgressBar (figura 5.32).

2. Obtención de las alarmas de la base de datos (figura 5.33).
3. Creación de sub-alarmas (figura 5.34).

La barra de progreso es meramente visual (figura 5.32), es una pantalla que aparece justo después del inicio de sesión con el objetivo de que resulte más atractiva visualmente la carga de las alarmas, y de esta manera el paciente es consciente de que en el caso de tener algún tratamiento, las alarmas se configuran al inicio.

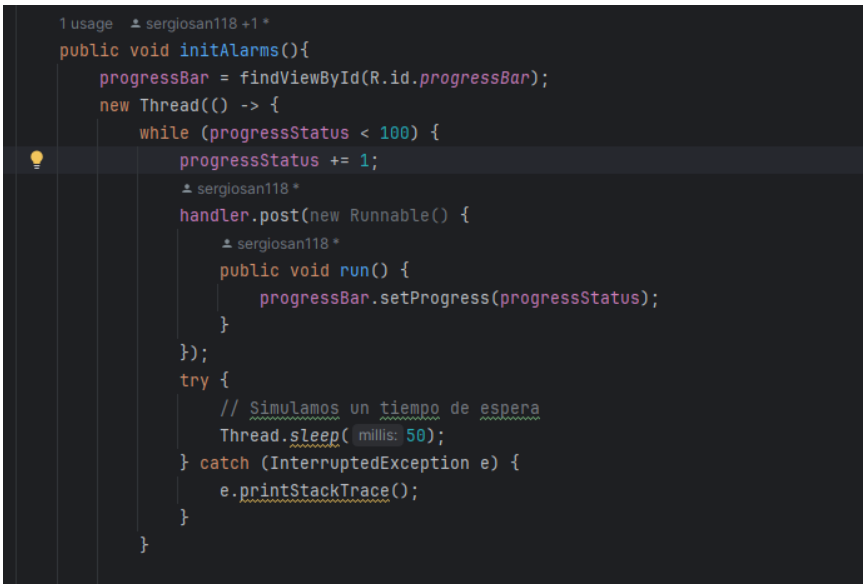

Figura 5.32- CargandoConfiguracionesActivity - método initAlarms() - progressBar

En la base de datos no se encuentran todas las alarmas, sólo se encuentran la primera de cada medicamento (figura 5.33). Cada alarma tiene los siguientes campos: id_alarma, id_tratamiento (ya que cada alarma tiene un tratamiento asociado y como se ve en este caso, estas dos alarmas corresponden al mismo tratamiento), medicamento, dosis, hora de la primera toma, tomas al día, fecha de inicio y fecha de fin del tratamiento.

	id_alarma	id_tratamiento	medicamento	dosis	hora_primera_toma	tomas_al_dia	fecha_inicio	fecha_fin
Editar Copiar Borrar	5	54	Aspirina	200 mg	08:00:00	3	2024-05-15	2024-05-17
Editar Copiar Borrar	6	54	Ibuprofeno	300 mg	09:00:00	4	2024-05-15	2024-05-17

Figura 5.33- MySQL - Tabla alarmas

En la figura 5.34 se realiza una llamada al servidor para que obtenga la información de las alarmas de dicho paciente. También se realiza el parseo de las

fechas y del tiempo para poder comparar las fechas de la base de datos con las del móvil.

```
Executor executor = Executors.newSingleThreadExecutor();
String urlServidor = ConfigApi.BASE_URL+"pacientes/obtenerAlarmas/" + sessionManager.getUserId() ;
getDataAsync(urlServidor, executor, (GetDataAsync.OnTaskCompleted) result -> {
    if (result != null) {
        int idAlarmaActual=0;
        try {
            JSONArray jsonArray = new JSONArray(result);
            Log.i( tag: "DOCTORES",jsonArray.toString());
            long currentTimeMillis = System.currentTimeMillis();
            for (int i = 0; i < jsonArray.length(); i++) {
                JSONObject jsonObject = jsonArray.getJSONObject(i);
                int idAlarma = jsonObject.getInt( name: "id_alarma");
                int idTratamiento = jsonObject.getInt( name: "id_tratamiento");
                String medicamento = jsonObject.getString( name: "medicamento");
                String dosis = jsonObject.getString( name: "dosis");
                String horaPrimeraToma = jsonObject.getString( name: "hora_primera_toma");
                int tomasAlDia = jsonObject.getInt( name: "tomas_al_dia");
                String fechaInicio = jsonObject.getString( name: "fecha_inicio");
                String fechaFin = jsonObject.getString( name: "fecha_fin");

                SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'", Locale.getDefault());
                Date dateInicio = sdf.parse(fechaInicio);
                Date dateFin = sdf.parse(fechaFin);
                Calendar calendar = Calendar.getInstance();
                calendar.setTime(dateInicio);
                calendar.add(Calendar.DAY_OF_MONTH,  amount: 1);
                String[] horaArray = horaPrimeraToma.split( regex: ":");
                calendar.set(Calendar.HOUR_OF_DAY, Integer.parseInt(horaArray[0]));
                calendar.set(Calendar.MINUTE, Integer.parseInt(horaArray[1]));
                calendar.set(Calendar.SECOND, Integer.parseInt(horaArray[2]));
                long timestamp = calendar.getTimeInMillis();
                int intervaloHoras = 24 / tomasAlDia;
                Calendar calInicio = Calendar.getInstance();
                calInicio.setTime(dateInicio);
                Calendar calFin = Calendar.getInstance();
                calFin.setTime(dateFin);
                long diferenciaEnMillis = calFin.getTimeInMillis() - calInicio.getTimeInMillis();
                int diasEntreFechas = (int) (diferenciaEnMillis / (1000 * 60 * 60 * 24)) + 1;
```

Figura 5.34- CargandoConfiguracionesActivity - método initAlarms() - Obtención de alarmas

Una vez se tenga la información de las alarmas de la base de datos, se crea un bucle (Ver figura 5.35) con el objetivo de crear no solo la primera alarma, sino todas las alarmas desde el día del inicio, hasta la fecha final. Para ello, se crea la primera alarma utilizando la información de la hora de la primera toma y la fecha de inicio. Las siguientes se distribuyen a lo largo del día en función de las tomas al día elegidas. Por ejemplo, si se eligen tres tomas al día, y la primera hora de la alarma está establecida a las 08:00 de la mañana, se dividen entre las 24 horas del día entre las tres tomas (24/3), lo que

significa que las siguientes alarmas sonarán cada ocho horas (si fueran cuatro tomas al día sería 24/4, es decir, cada seis horas). De esta manera la siguiente alarma sonará a las 16:00 y la última a las 00:00 del día siguiente. Por último, las alarmas de los días siguientes son copias de las alarmas del primer día hasta que llegue el día de fin del tratamiento.

Finalmente se llama al método "creaciónAlarma" del *controller* pasándole como parámetro el id de cada alarma, la fecha en la que va a sonar parseada a segundos (al imprimir por pantalla se muestra con una fecha legible), el medicamento y la dosis correspondiente.

Las alarmas que ya han pasado su fecha no suenan. En el logcat del móvil se pueden observar ver las alarmas que han sido creadas imprimiendolas por pantalla (ver figura 5.36).

Figura 5.35- CargandoConfiguracionesActivity - método initAlarms() - creación de sub-alarmas

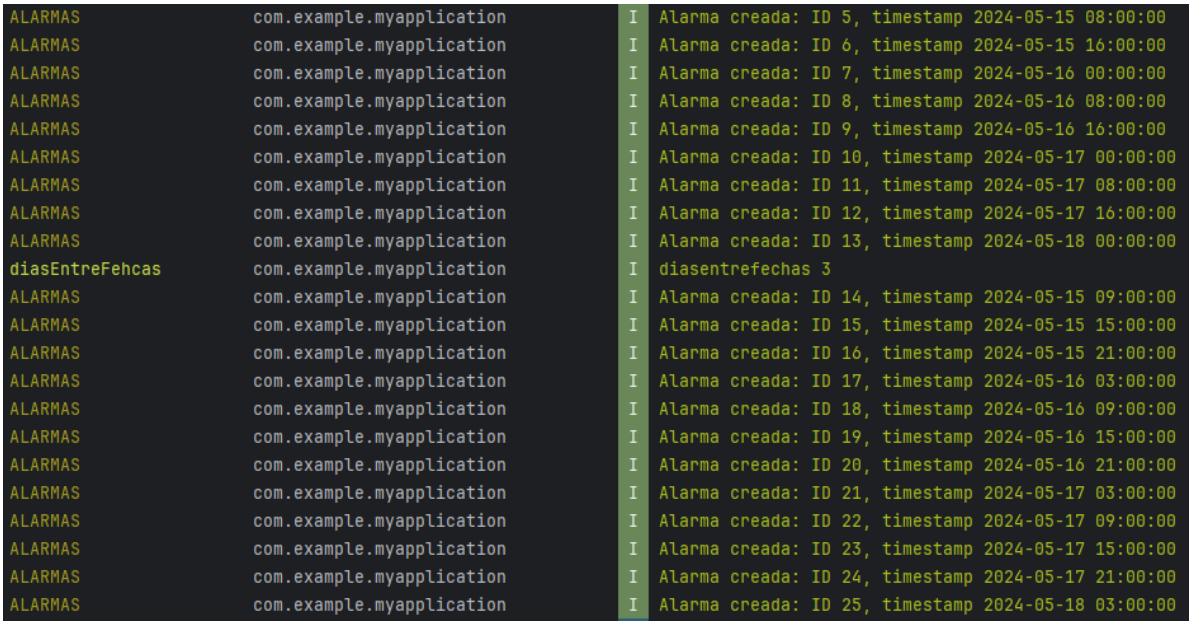

Figura 5.36- Logcat - alarmas creadas al iniciar el móvil

La creación de las alarmas en el móvil tiene lugar en el método "creaciónAlarma" del *controller* (ver figura 5.37) que se llama desde la figura 5.35 como se vió anteriormente.

Se crea una instancia de la clase "AlarmManager", que permite acceder al servicio de alarmas del sistema. Su principal función es poder programar operaciones en un tiempo establecido y fuera del ciclo de vida de la aplicación. Dentro de la misma clase, esta línea "Intent alarmIntent = new Intent(ctx, AlarmReceiver.class)" crea un *intent* que especifica el componente que manejará la emisión (*broadcast*) cuando se active la alarma. En este caso, la clase "AlarmReceiver" (se verá más adelante). La creación de la alarma tiene lugar con el método set del "AlarmManager". Este método recibe como parámetro el sonido que se reproducirá al activarse, la fecha en la que tiene que sonar en segundos y un pendingIntent (ver figura 5.37 de nuevo) que en el momento de ser ejecutada, transmite una emisión a un receptor (*broadcast receiver*) en el sistema. Esta emisión será manejada en la recepción de las alarmas. El resto del código se utiliza para ver las alarmas que han sido creadas por pantalla.

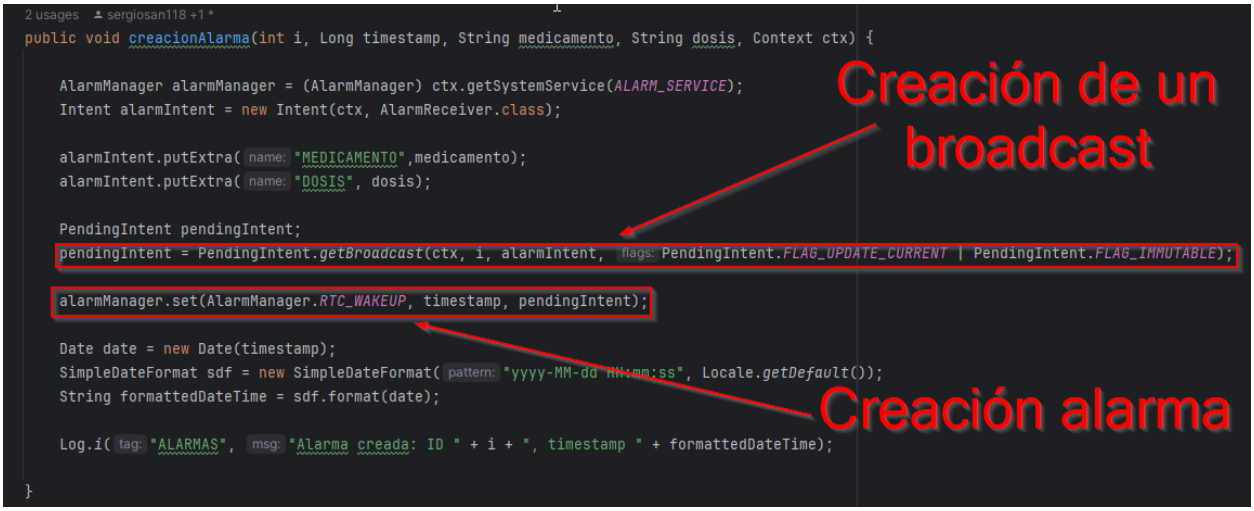

Figura 5.37- Controller - método creaciónAlarma

5.3.3.2 Recibir Alarmas

Una vez las alarmas han sido creadas, se procede a la segunda fase, que consiste en la recepción de las mismas.

La clase "AlarmReceiver" (ver figura 5.38) es un componente crucial que se encarga de recibir y manejar las emisiones (*broadcast*) que son enviadas cuando se activan las alarmas en la aplicación. Esta clase extiende de "BroadcastReceiver" con lo que al recibir la emisión de que una alarma se va a ejecutar, se llama al método "onReceive()", y este método se encarga de lanzar una notificación en el caso de que la sesión de dicho usuario esté abierta.

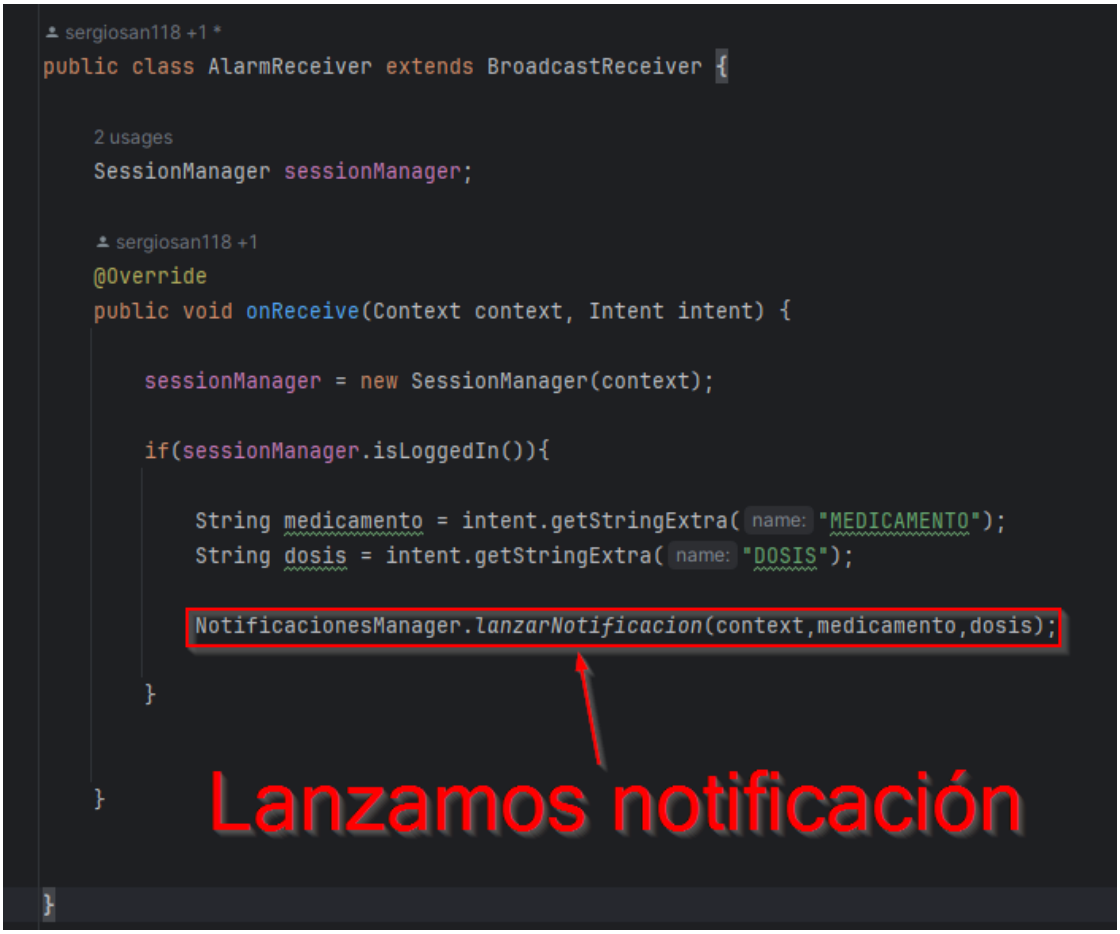

Figura 5.38- AlarmReceiver

En el método "onReceive", antes de llamar al "lanzarNotificacion", se obtiene el medicamento y la dosis correspondientes a la alarma activada. Estos atributos se pasan en el momento de la creación de las alarmas en el controlador (ver figura 5.39).

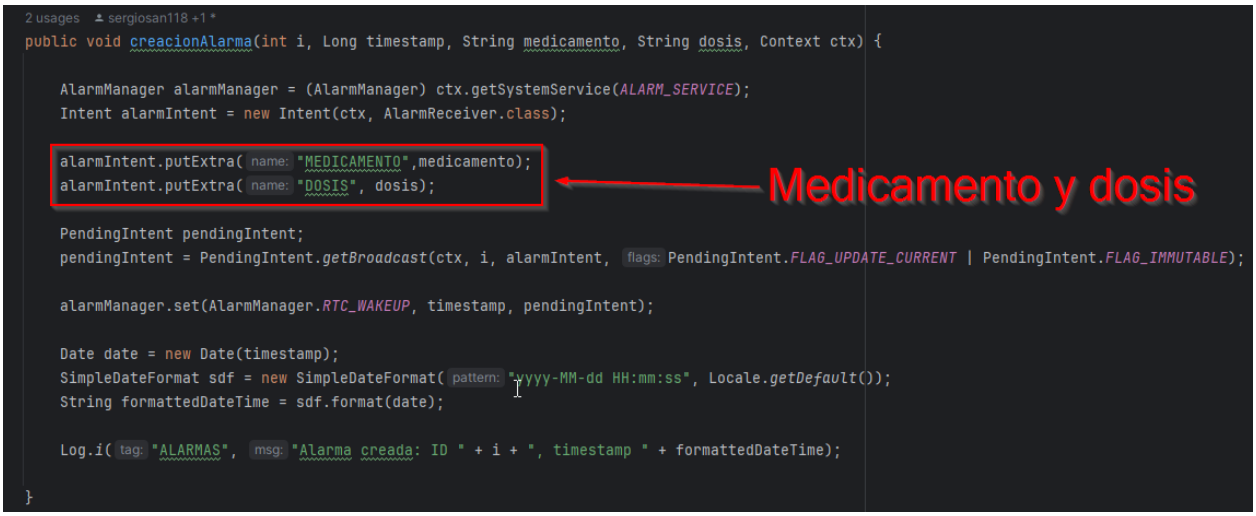

Figura 5.39- Controller – creacionAlarma

En el método "lanzarNotificacion" (ver figura 5.38 previamente vista) de la clase "NotificacionesManager", que es una clase que se utiliza para gestionar y mostrar notificaciones, se procede a mostrar la notificación con el mensaje correspondiente. A continuación, se desglosa lo que hace cada parte del método (ver figura 5.40):

1. Obtención del servicio de notificaciones mediante una instancia del servicio "NotificacionesManager".
2. Creación del canal de notificaciones (ver figura 5.41 y 5.42), en el que se establece y configura un canal de notificaciones para asegurar que las notificaciones enviadas desde la aplicación lleguen al dispositivo móvil a través de ese canal específico. Otros atributos también son ajustados como la vibración entre otros para *Android Oreo* y versiones posteriores.
3. Creación del *intent* y creación del *pendingIntent* para detener la reproducción del sonido. Se configura la acción que va a realizar el botón (en el punto 4 es creado y se le asigna este *pendingIntent*) al ser pulsado. En este caso, se va a llamar a la clase "NotificationButtonReceiver" (ver figura 5.43) que recibe la acción de que el botón ha sido pulsado y se detiene la reproducción del sonido en bucle.
4. Configuración de la notificación como el icono, título, acción y contenido.

.

5. Inicio del reproductor de medios. Se crea una instancia de "MediaPlayer" y se inicia un archivo de audio de alarma "R.raw.alarmclock". Este reproductor se establece en bucle para reproducir continuamente el sonido de la alarma.
6. Intención de parar el sonido al tocar la notificación ejecutando "stopPendingIntent". El sonido se detiene al deslizar la notificación, pulsarla o al presionar sobre el botón de "Detener" como se explica en el punto 3.
7. Envío de la notificación. Finalmente, la alarma suena y aparece la notificación (ver figura 5.44) en el móvil.

```
public static void lanzarNotificacion (Context context,String medicamento, String dosis){

    NotificationManager notificationManager = (NotificationManager) context.getSystemService(NOTIFICATION_SERVICE);
    NotificationChannel notificationChannel =null;                                                                          1.

    if(Build.VERSION.SDK_INT>=Build.VERSION_CODES.O){

        notificationChannel = crearCanalNotificaciones(context,NOMBRE_CANAL,ID_CANAL);                                      2.
        notificationManager.createNotificationChannel(notificationChannel);
    }                                                                                                                       3.

    Intent stopIntent = new Intent(context, NotificationButtonReceiver.class);
    stopIntent.setAction("ACTION_STOP_MEDIA_PLAYER");
    PendingIntent stopPendingIntent = PendingIntent.getBroadcast(context, requestCode: 0, stopIntent, flags: PendingIntent.FLAG_UPDATE_CURRENT
            | PendingIntent.FLAG_IMMUTABLE);

    NotificationCompat.Builder nb = new NotificationCompat.Builder(context,ID_CANAL);
    nb.setPriority(NotificationCompat.PRIORITY_HIGH);
    nb.setVisibility(NotificationCompat.VISIBILITY_PUBLIC);
    nb.setSmallIcon(android.R.drawable.ic_dialog_info);
    nb.setLargeIcon(BitmapFactory.decodeResource(context.getResources(),R.mipmap.ic_launcher_round));                      4.
    nb.setContentTitle(medicamento + ", dosis: " + dosis);
    nb.setDeleteIntent(stopPendingIntent);
    nb.addAction(android.R.drawable.ic_media_pause, title: "Detener", stopPendingIntent);

    mp = MediaPlayer.create(context,R.raw.alarmclock);
    mp.setLooping(true);                                                                                                    5.
    mp.start();

    nb.setContentIntent(stopPendingIntent);
    nb.setAutoCancel(true);                                                                                                 6.

    Notification notification = nb.build();
    notificationManager.notify( id: 57,notification);                                                                       7.
}
```

Figura 5.40- NotificacionesManager - método lanzarNotificacion

Figura 5.41- NotificacionesManager - método crearCanalNotificaciones

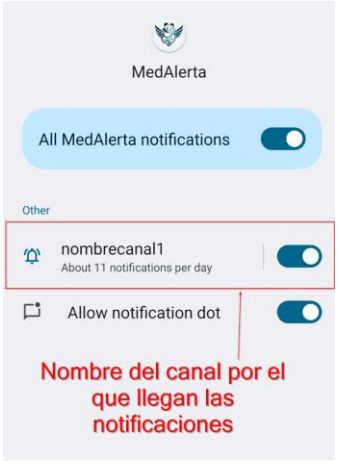

Figura 5.42- Notificaciones MedAlerta - Nombre del canal

```java
sergiosan118
public class NotificationButtonReceiver extends BroadcastReceiver {
    sergiosan118
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if ("ACTION_STOP_MEDIA_PLAYER".equals(action)) {
            // Detener el MediaPlayer
            if (NotificacionesManager.mp != null && NotificacionesManager.mp.isPlaying()) {
                NotificacionesManager.mp.stop();
                NotificacionesManager.mp.release();
                NotificacionesManager.mp = null;
            }
        }
    }
}
```

Figura 5.43- NotificationButtonReceiver - Método onReceive

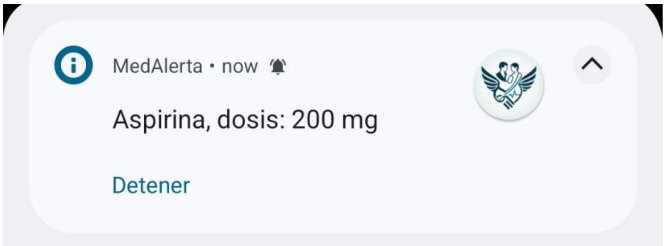

Figura 5.44- Notificación - Notificación recibida.

5.3.4 Módulo Gestión de Consultas Paciente

Dentro de esta sección están implementadas las acciones que puede realizar el paciente respecto a la solicitud y administración de citas.

5.3.4.1 Ver Consultas

En la pantalla de las consultas, aparecen dos elementos, un "recyclerView" muy parecido al que se ha visto anteriormente y un botón.

En este caso, el "recyclerView" en vez de ser de citas, se muestra una lista de las consultas de dicho paciente, para acceder a cada consulta, se pulsa sobre el *ítem* de la consulta (ver figura 5.45) deseada. Mediante el método "onClick" (ver figura 5.46) de la clase "consultasListAdapter" se puede acceder a una consulta específica mediante el método "navigateToDestinationWithData" del "NavigationManager" pasándole como parámetro la clase que se quiera mostrar ("ChatViewActivity.class") y la consulta ("e") que se ha pulsado, que tiene la información de los usuarios que van a comunicarse.

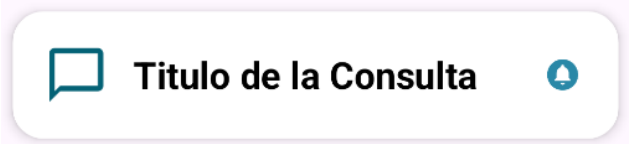

Figura 5.45- XML - Item de consulta.

```
public void onClick(View v) {
    image_notificacion.setVisibility(View.GONE);
    NavigationManager.getInstance().navigateToDestinationWithData(context, ChatViewActivity.class, e);
}
```

Figura 5.46- ConsultasListAdapter - Método onClick().

El botón "Crear consulta" redirige a una pantalla para la creación de una nueva consulta.

5.3.4.2 Crear Consultas

El paciente para crear una consulta debe rellenar un formulario como en el solicitar citas donde selecciona el doctor que desee mediante un *spinner* en el que se muestran todos los doctores que tiene asignado el paciente, en caso de que no haya doctores, se mostrará un mensaje de que no existen doctores disponibles para crear una consulta.

El paciente una vez que elija al doctor deseado, deberá poner un título a dicha consulta, este campo no puede estar vacío, y al crear la consulta se envían los datos desde el controlador hasta al servidor mediante el *service* con una llamada *POST*. Estos datos son guardados en la tabla de consultas con el dni del paciente, dni del doctor, el título, la fecha del último mensaje que será en ese momento la fecha en la que se ha creado dicha consulta y los mensajes no leídos por parte del doctor y el paciente que se inicializan a 0.

Una vez los datos han sido guardados en la base de datos se mostrará un mensaje de que la consulta se ha creado correctamente. El paciente podrá ver la nueva consulta en la pantalla "Ver consultas", pudiendo acceder al chat y comunicarse con el doctor.

5.3.4.3 Ver Mensajes Consultas

Una vez accedes a un *chat*, en la parte superior se mostrará el nombre del doctor con el que estás hablando y casi toda la pantalla es ocupada por un "recyclerView" con dos *items*, los cuales son el mensaje del doctor, y el mensaje del paciente. Ambos elementos están colocados de manera que parezca un chat real, por lo que el mensaje del doctor está colocado en la parte izquierda y el del paciente en la parte derecha alineando a la derecha con "android: layout_gravity=end" (ver figuras 5.47 e 5.48).

Figura 5.47- Item del mensaje del doctor.

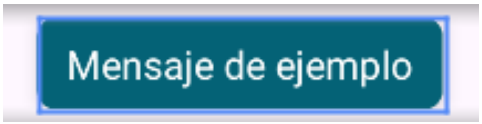

Figura 5.48- Item del mensaje del paciente.

Para determinar qué *ítem* corresponde a cada usuario (paciente o doctor) es determinado por un atributo de la tabla mensajes llamado propietario. Este atributo indica 1 si el mensaje es del paciente y 0 si el mensaje es del doctor. En la actividad "chatViewActivity" se llama al método "getAllMensajes" del *service* a través del *controller* para enviar los datos al servidor y posteriormente hacer una llamada al DAO para traer todos los mensajes de una consulta de la base de datos. Estos mensajes se insertan dentro del *adapter* que es una instancia de "MensajesListAdapter" y mediante el método "setMensajesList" del *adapter* (ver figura 5.49), se introducen todos los mensajes para procesarlos.

Los mensajes se guardan en una lista de objetos tipo "Mensaje" (ver figura 5.50), donde cada mensaje tiene el id de la consulta a la que pertenece, el mensaje, el propietario y la fecha en la que fue escrito.

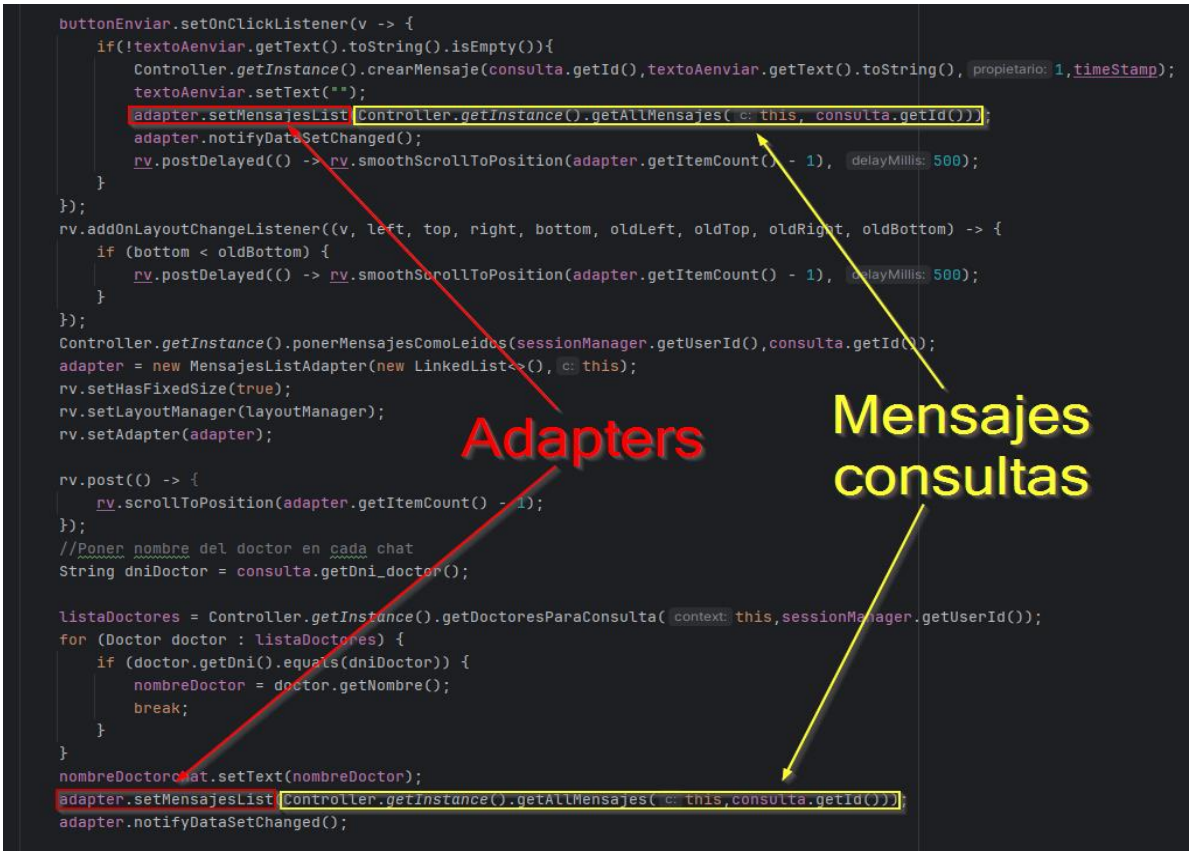

Figura 5.49- ChatViewActivity - Método getAllConsultas()

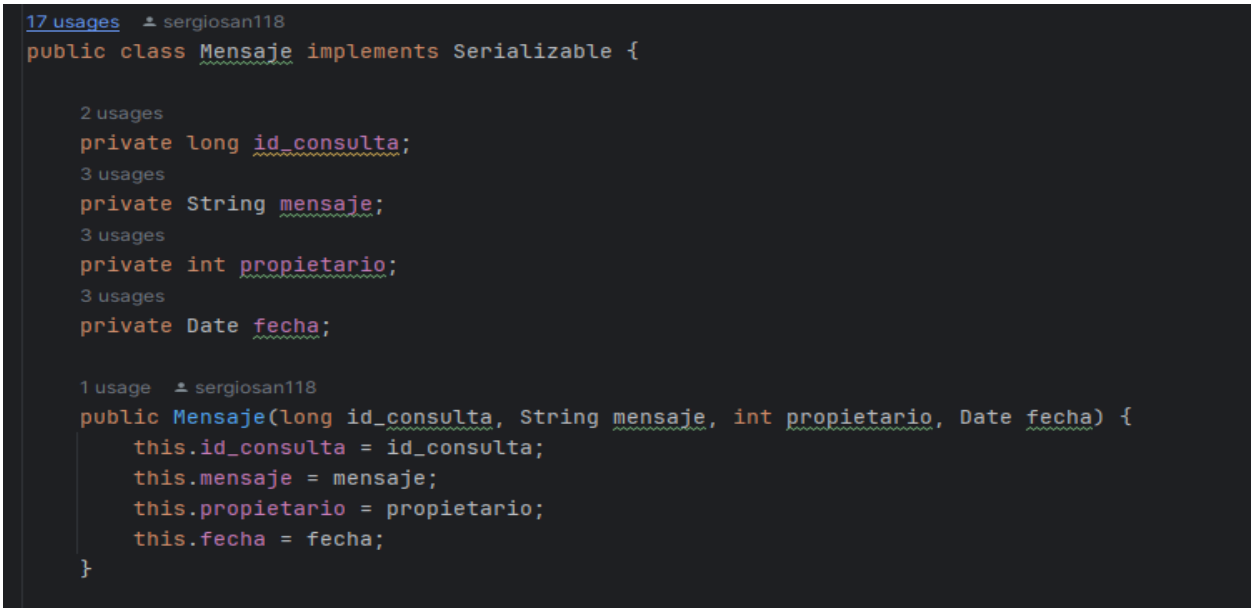

Figura 5.50- Clase de un Mensaje

Dentro de la clase "MensajesListAdapter", existe un método llamado "onCreateViewHolder" (ver figura 5.51), donde en función del "viewType" de cada mensaje (que es lo mismo que el propietario) se asigna un *ítem*, es decir, es aquí donde se diferencia de quién es cada mensaje.

Para detectar qué atributo se está pasando (propietario) se sobreescribe una función de "MensajesListAdapter" llamada "getItemViewType()" (ver figura 5.52) que devuelve el propietario de cada mensaje y para que en el método explicado arriba se pueda diferenciar de quién es cada mensaje.

```
sergiosan118 *
@NonNull
@Override
public MensajesListAdapter.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {

    View view;
    if (viewType == 0) {
        view = LayoutInflater.from(parent.getContext()).inflate(R.layout.mensaje, parent, attachToRoot: false);

    } else {
        view = LayoutInflater.from(parent.getContext()).inflate(R.layout.mensaje0, parent, attachToRoot: false);
    }
    view.setOnClickListener(onClickListener);
    return new ViewHolder(view,context,viewType);
}
```

Figura 5.51- MensajesListAdapter - Método onCreateViewHolder

```
@Override
public int getItemViewType(int position) {
    int propietario = mensajesList.get(position).getPropietario();

    return propietario;
}
```

Figura 5.52- MensajesListAdapter- Método getItemViewType

El paciente al recibir un mensaje que no ha sido leído, recibe una notificación al móvil. Los mensajes al ser creados tienen dos atributos que se inicializan a 0 ("leido_doctor" y leido_paciente"). En el caso del paciente, el que importa es "leido_paciente". Si en una consulta hay mensajes que el paciente aún no ha visto, en la pantalla de consultas aparecerá un icono de una campana azul (ver figura 5.53).

Para saber qué mensajes no han sido leídos se envía una señal al servidor cada diez segundos que lo compruebe en la clase "NotificarMensajesAsync" (ver figura 5.54).

Éstos mensajes se guardan en una lista y si la lista no está vacía, se envía una notificación al móvil (ver figura 5.55), la cual si se pulsa, redirige a la pantalla de las consultas.

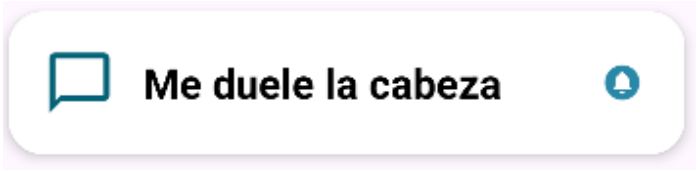

Figura 5.53- Consultas - Icono de notificación de mensajes no leídos

```
@Override
public void onCreate() {
    super.onCreate();

    mHandler = new Handler();
    sergiosan118 *
    mRunnable = new Runnable() {
        sergiosan118 *
        @Override
        public void run() {

            SharedPreferences pref = getSharedPreferences( name: "datos_persona", Context.MODE_PRIVATE);
            dni = pref.getString( key: "dni",  defValue: "");

            LinkedList<String> mensajesNoLeidos = obtenerConsultasSinLeer();


            if (mensajesNoLeidos.size() > mensajesNoLeidosAnteriores.size()) {
                NotificacionesManager.lanzarNotificacionMensajeNoLeido(getApplicationContext());
            }

            mensajesNoLeidosAnteriores = mensajesNoLeidos;

            mHandler.postDelayed( r: this,  delayMillis: 10000); // Ejecutar cada 10 segundos
        }
    };
    mHandler.postDelayed(mRunnable,  delayMillis: 10000); // Ejecutar por primera vez después de 10 segundos
}
```

Figura 5.54- NotificarMensajesAsync - Método onCreate()

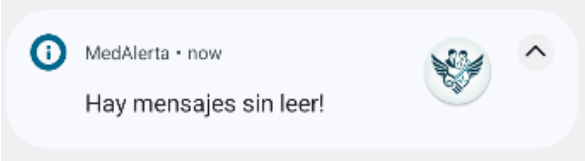

Figura 5.55- Notificación - Mensajes no leídos

En la figura 5.56 se observa cómo se guarda el atributo "dni" en SharedPreferences, que permite el almacenamiento de datos simples y persistentes, es decir, guardar datos incluso después de que la aplicación se cierre. De esta manera se consigue que las notificaciones sigan apareciendo, ya que, si se conserve el dni del paciente, se puede acceder a sus consultas y por lo tanto a sus mensajes.

```
            SharedPreferences pref = getSharedPreferences( name: "datos_persona", Context.MODE_PRIVATE);
            dni = pref.getString( key: "dni",  defValue: "");

            LinkedList<String> mensajesNoLeidos = obtenerConsultasSinLeer();

            if (mensajesNoLeidos.size() > mensajesNoLeidosAnteriores.size()) {

                NotificacionesManager.lanzarNotificacionMensajeNoLeido(getApplicationContext());
            }
            mensajesNoLeidosAnteriores = mensajesNoLeidos;
            mHandler.postDelayed( r: this,  delayMillis: 10000); // Ejecutar cada 10 segundos
        }
    };
    mHandler.postDelayed(mRunnable,  delayMillis: 10000); // Ejecutar por primera vez después de 10 segundos
}
 sergiosan118 *
public int onStartCommand(Intent intent, int flags, int startId) {

    dni = intent.getStringExtra( name: "dni");
    // Código para insertar
    SharedPreferences pref = getSharedPreferences( name: "datos_persona", Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = pref.edit();
    editor.putString("dni", dni);
    editor.commit();
    editor.apply();
    return Service.START_STICKY;
}
```

Figura 5.56- NotificarMensajesAsync - SharedPreferences

El atributo "leido_paciente" cambia de valor a 1 en el caso que el paciente entre en una consulta, indicando que esos mensajes ya han sido leídos. Esto se puede ver en el método "ponerMensajesComoLeídos" (punto 1, ver figura 5.57) pasándole como parámetro el dni de paciente y la consulta, para así marcar en leído los mensajes que pertenecen a dicho paciente y a dicha consulta. La información se envía al servidor y se modifica el atributo "leido_paciente" de los mensajes de dicho usuario.

El "recyclerView" está configurado para que se muestren los últimos mensajes escritos, no sólo al ver la consulta, sino también tras escribir un mensaje. Esto se hace para que el "recyclerView" actúe como un chat sencillo visualmente y cómodo (punto 2, ver figura 5.57).

Los mensajes son instantáneos, es decir, si el doctor escribe un mensaje, al cabo de dos segundos se podrá ver en el chat, ésto sucede por la clase "ActualizarMensajesAsync" (ver figura 5.58) en donde se van recogiendo todos los mensajes cada dos segundos y posteriormente pasarlos al adapter "MensajesListAdapter" para que se mantengan actualizados. De esta manera se consigue refrescar el *chat* para que se muestre dinámico (ver punto 3 figura 5.57).

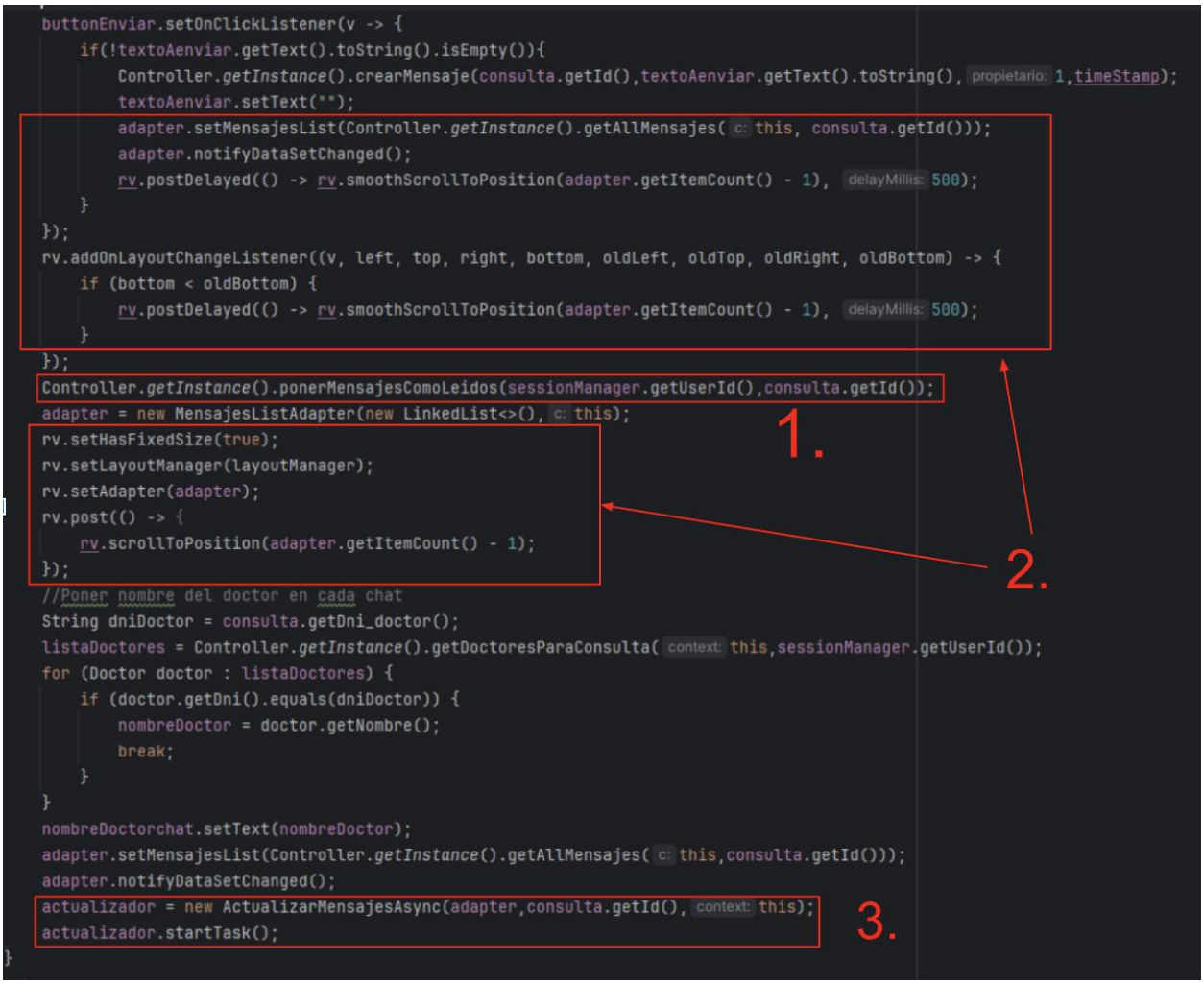

Figura 5.57- ChatViewActivity - Método initGui

```
public class ActualizarMensajesAsync {
    4 usages
    private Handler mHandler;
    2 usages
    private boolean mShouldStop = false;
    2 usages
    private MensajesListAdapter mAdapter;
    2 usages
    private Context context;
    2 usages
    private Long idConsulta;
    1 usage  Antoranz
    public ActualizarMensajesAsync(MensajesListAdapter adapter,Long idConsulta, Context context) {
        mAdapter = adapter;
        mHandler = new Handler(Looper.getMainLooper());
        this.idConsulta = idConsulta;
        this.context = context;
    }
    1 usage  Antoranz
    public void startTask() {
        mHandler.postDelayed(updateRunnable,  delayMillis: 2000); // Ejecutar la tarea cada 2 segundos
    }
    1 usage  Antoranz
    public void stopTask() { mShouldStop = true; }
    1 usage  Antoranz *
    private Runnable updateRunnable = new Runnable() {
        Antoranz *
        @Override
        public void run() {
            if (!mShouldStop) {
                LinkedList<Mensaje> newMensajes = obtenerMensajesDelServidor();
                mHandler.post(() -> mAdapter.setMensajesList(newMensajes));
                mHandler.postDelayed( r: this,  delayMillis: 2000);
            }
        }
    };
    1 usage  Antoranz
    private LinkedList<Mensaje> obtenerMensajesDelServidor() {
        return Controller.getInstance().getAllMensajes(context,idConsulta);
    }
```

Figura 5.58- ActualizarMensajesAsync

5.3.4.4 Enviar Mensajes Consultas

Para el envío de un mensaje, se rellena el cuadro de texto situado en la parte inferior de cada *chat* (el mensaje no se puede enviar vacío) y se presiona el botón que se encuentra al lado de este cuadro de texto.

Al pulsar el botón (llamado en el chatViewActivity "buttonEnviar"), si el texto no está vacío se ejecuta el método "crearMensaje" (ver figura 5.59) al que se le pasa como atributos el id de la consulta, el texto enviado, el propietario (que tiene el valor 1 indicando que es del paciente) y la fecha actual en la que el mensaje es enviado.

La información del mensaje es enviada como se ha visto en otras ocasiones, desde el service mediante el *controller* al servidor. El mensaje es insertado en la tabla de mensajes. Esto se vería reflejado en el *chat* a los pocos segundos debido a mencionado anteriormente sobre la actualización de este chat cada pocos segundos utilizando la clase "ActualizarMensajesAsync".

```
@SuppressLint({"WrongViewCast", "NotifyDataSetChanged"})
private void initUI() {
    nombreDoctorchat = findViewById(R.id.nombreDoctorTextView);
    textoAenviar = findViewById(R.id.textoAEnviar);
    buttonEnviar = findViewById(R.id.botonEnviar);
    RecyclerView rv = findViewById(R.id.recyclerViewChat);
    LinearLayoutManager layoutManager = new LinearLayoutManager( context: this);
    String timeStamp = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss", Locale.getDefault()).format(new Date());

    buttonEnviar.setOnClickListener(v -> {

        if(!textoAenviar.getText().toString().isEmpty()){
            Controller.getInstance().crearMensaje(consulta.getId(),textoAenviar.getText().toString(), propietario: 1,timeStamp);
            textoAenviar.setText("");

            adapter.setMensajesList(Controller.getInstance().getAllMensajes( c: this, consulta.getId()));
            adapter.notifyDataSetChanged();
            rv.postDelayed(() -> rv.smoothScrollToPosition(adapter.getItemCount() - 1), delayMillis: 500);
        }

    });


    rv.addOnLayoutChangeListener((v, left, top, right, bottom, oldLeft, oldTop, oldRight, oldBottom) -> {
        if (bottom < oldBottom) {
            rv.postDelayed(() -> rv.smoothScrollToPosition(adapter.getItemCount() - 1), delayMillis: 500);
        }
    });

    Controller.getInstance().ponerMensajesComoLeidos(sessionManager.getUserId(),consulta.getId());


    adapter = new MensajesListAdapter(new LinkedList<>(), c: this);
    rv.setHasFixedSize(true);
    rv.setLayoutManager(layoutManager);
    rv.setAdapter(adapter);
```

Figura 5.59- ChatViewActivity - Método crearMensaje

Capítulo 6 - Implementación, Estructura y Diseño de la Aplicación Web

En esta sección se exponen los detalles de la implementación, la estructura tanto del servidor como la estructura de la aplicación web ya que ambas se alojan en la página web y el diseño de la plataforma web, organizando las diversas funcionalidades en los siguientes módulos funcionales:

- Módulo Gestión de usuarios
- Módulo Gestión de asignaciones
- Módulo Gestión de Historial
- Módulo Gestión de Citas
- Módulo Gestión de tratamientos
- Módulo Gestión de consultas

6.1 Diseño de la Aplicación Web

La aplicación web tiene un diseño simple, que facilita la gestión de un listado de pacientes para los doctores.

La versión web de nuestra aplicación ha sido diseñada para reflejar los mismos principios de accesibilidad y sencillez que la aplicación móvil de Android. Nuestro objetivo es ofrecer una experiencia de usuario consistente y fácil de usar.

En el diseño de la web se han utilizado imágenes descriptivas para que la interfaz sea más comprensible y visualmente atractiva. Estas imágenes ayudan a guiar a los usuarios a través de las diferentes funcionalidades de la aplicación, mejorando la experiencia y accesibilidad.

6.2 Estructura del Servidor

La sección del servidor de la aplicación se organiza en un sistema de carpetas que comparte con el *front-end* de la parte web de la aplicación a través del sistema de

ficheros formado automáticamente por *node.js* como se puede observar en la *Figura 6.1)*.

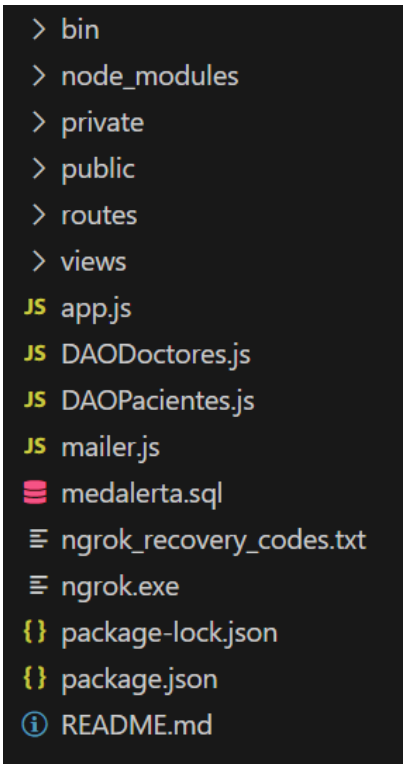

Figura 6.1- Estructura de ficheros del servidor.

El directorio "bin/" contiene un pequeño script llamado "*www*", generado automáticamente por *node.js*, cuya tarea es importar el fichero "app.js/" e inicializa el servidor *HTTP* sobre el cual funciona la aplicación.

La carpeta "node_modules/" almacena las dependencias del proyecto.

El directorio "private/" (ver figura 6.2) contiene datos privados que se almacenan en el servidor. Está compuesta por dos carpetas principales:

- "data/": Guarda un ".json" llamado "enfermedades.json" que contiene una lista de enfermedades que sirven para asignar automáticamente las enfermedades a los doctores al registrarse.
- "historiales/": En esta localización se guardan los historiales en dos formatos diferentes: ".pdf" y ".json".

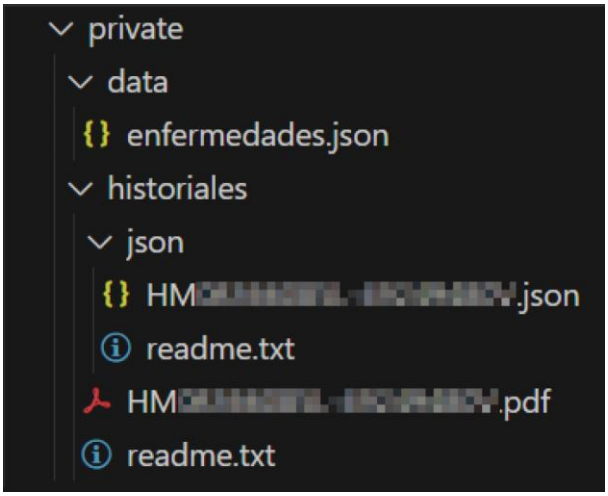

Figura 6.2- Estructura de carpeta private

La carpeta "public/" y "views/" contienen archivos que no se usan para el funcionamiento del servidor. Estas carpetas se tratarán más tarde al explicar el *front-end* de la web.

El directorio "routes/" está reservado para ficheros de tipo *JavaScript*(".js") que se encargan de definir las diferentes rutas de la aplicación (ver Figura 6.3).

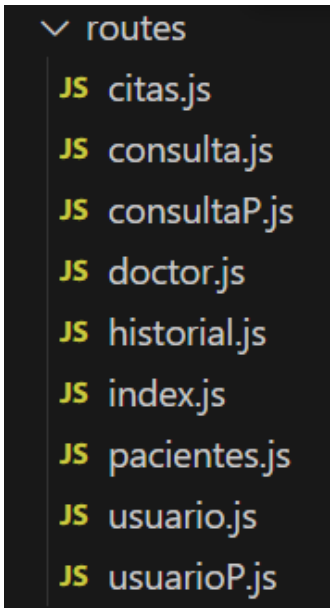

Figura 6.3- Estructura de directorio routes

El archivo "app.js" crea la instancia de la aplicación *express* y configura diferentes *middlewares*, módulos y rutas principales.

Los ficheros *JavaScript* "DAODoctores.js" y "DAOPacientes.js" guardan una clase con variedad de funciones que permiten realizar consultas a la base de datos *SQL* de sus respectivos usuarios (doctores y pacientes).

El archivo "mailer.js" posee información de configuración del correo electrónico que se usa para la verificación de usuarios.

El fichero "medAlerta.sql" almacena una copia de la base de datos.

Los archivos "ngrok.exe" y "ngrok_recovery_codes.txt" permiten exponer el servidor local de la aplicación a internet.

Los archivos "package.json" y "package-lock.json" almacenan meta información de la aplicación *node.js* como su nombre, versión, dependencias, entre otras cosas.

Finalmente cualquier documento de texto llamado "readme.txt" guarda meta información de los archivos de la aplicación.

6.3 Estructura de la Aplicación Web

La sección de la parte web de la aplicación es la que se encarga de la parte *front-end*. Comparte el directorio con el servidor pero solo utiliza la carpeta "public/" y "views/" *(ver figura 6.4)*.

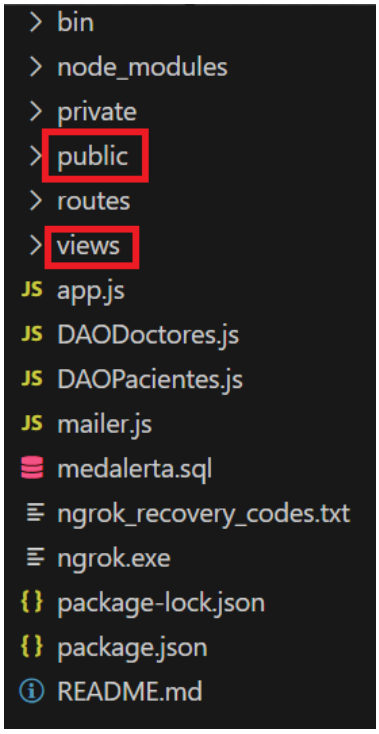

Figura 6.4- Estructura de ficheros de la aplicación web

La carpeta "public/" (ver figura 6.5) contiene recursos que utiliza la aplicación web para su funcionamiento. Esta está compuesta por otras tres carpetas:

- "images/" guarda imágenes de los tipos: "*.webp*", "*.png*" y "*.jpg*"
- "scripts/" almacena ficheros *JavaScript*("*.js*") con *scripts* para su posterior uso en las vistas.
- "stylesheets/" es donde se guarda un fichero "*.css*" que guarda información para el diseño de las páginas web.

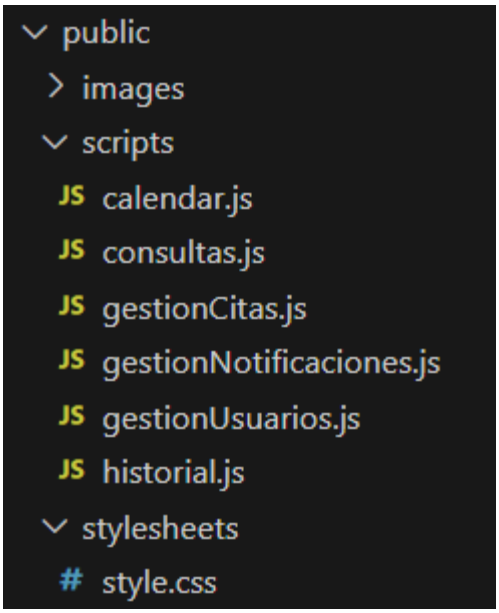

Figura 6.5- Estructura de la carpeta public

La carpeta "views/" (ver figura 6.6) posee todas las vistas de la página web en archivos de tipo *Embedded JavaScript* (".ejs")

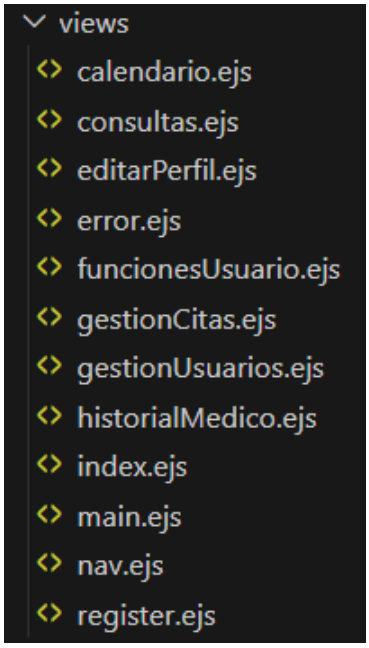

Figura 6.6- Estructura de la carpeta views

6.4 Implementación de la Aplicación Web

La implementación de la aplicación web se ha dividido en los módulos o funcionalidades previamente mencionados.

Las capturas de las pantallas se podrán visualizar en el manual de usuario doctor con el fin de no repetir las imágenes.

6.4.1 Módulo Gestión de Usuarios Doctor

6.4.1.1 Iniciar Sesión

Para iniciar sesión el usuario deberá rellenar un formulario con su dni y contraseña. Esta información recogida en el formulario (ver figura 6.7) se manda a través de un método *POST*, este método envía la información por el cuerpo del mensaje *HTTP*.

```
<form action="/doctor/usuario/signin/" method="post">
    <div class="form-group">
        <label for="dni">DNI</label>
        <input type="dni" class="form-control" id="dni" name="dni" placeholder="DNI" required>
    </div>

    <div class="form-group">
        <label for="idPassword">Contraseña</label>
        <input type="password" class="form-control" id="idPassword" name="password" placeholder="Contraseña" required>
    </div>

    <div class="form-group">
        <a href="/doctor/usuario/register" id="crearCuentaId" class="text-blue">Crear una cuenta</a>
    </div>
    <div class="form-group">
    </div>
    <div class="form-group">
        <button type="submit" class="btn btn-primary" id="editButton">Iniciar Sesión</button>
    </div>
```

Figura 6.7- Formulario inicio de sesión

Todas las peticiones al servidor son recogidas por "app.js" y dependiendo de la ruta, "app.js" enruta la llamada y la manda a otro *router*, en este caso a "doctorRouter" (ver Figura 6.8).

```
var indexRouter = require('./routes/index');
var pacientesRouter = require('./routes/pacientes');
var {router} = require('./routes/doctor');
var doctorRouter = router;

app.use('/', indexRouter);
app.use('/pacientes',pacientesRouter);
app.use('/doctor', doctorRouter);
```

Figura 6.8- Enrutamiento de app.js

Nuevamente el *router* "doctorRouter" enruta la llamada hacia el *router* "usuarioRouter" (ver figura 6.9)

```
var usuarioRouter = require('./usuario');
router.use('/usuario', usuarioRouter);

var citasRouter = require('./citas');
router.use('/citas', citasRouter);

var consultaRouter = require('./consulta');
router.use('/consulta', consultaRouter);

var historialRouter = require('./historial');
router.use('/historial', historialRouter);
```

Figura 6.9- Enrutamiento de doctor.js

Finalmente el "usuarioRouter" recibe la petición *POST*. Al recibirla se recoge el dni del doctor y su contraseña, la cual se cifra(ver Figura 6.10) antes de mandar ambos datos al *DAO* donde hace una llamada *SELECT* de la tabla doctores para comprobar si existe un usuario con esos datos. En caso de existir se crea la *session* de usuario para la web y se redirecciona a la página principal, habiéndose ya iniciado la sesión. Si no existe dicho usuario con contraseña en la base de datos se manda al usuario sin iniciar sesión a la página de inicio de sesión y se pasa un mensaje de error que la página detecta y muestra al usuario.

```
const crypto = require('crypto');

function cifrarContrasena(contrasena, salt) {

  const hash = crypto.createHash('sha256');

  hash.update(contrasena + salt);

  return hash.digest('hex');
}
```

Figura 6.10- Función de cifrado para contraseñas

6.4.1.2 Registrar

El registro de usuario se realiza al igual que el inicio de sesión con un formulario que te pide varios datos: dni, email, contraseña (2 veces para asegurar que se ha insertado la contraseña correcta), nombre, apellidos, fecha de nacimiento, domicilio, código postal y número de teléfono. El formulario hace una llamada *POST* que tras ser finalmente redirigida al "usuarioRouter", comprueba que los datos sean válidos (dni sea un dni existente, las contraseñas sean iguales, etc.). Tras comprobar que los datos son correctos se cifra la contraseña al igual que en el inicio de sesión y se hace una llamada al DAO donde se inserta a la tabla doctores. Finalmente se añaden al doctor las enfermedades por defecto guardadas en el fichero "private/data/ enfermedades.json" para que se generen por defecto algunos ejemplos para el doctor. Si durante el proceso de registro se detecta un error en los datos o no se permite añadir el doctor a la base de datos entonces se redirige a la página de registro con un mensaje de error.

6.4.1.3 Editar Perfil

Para editar el perfil de usuario, el doctor debe cambiar los datos de un formulario. Los datos que se permiten cambiar son: nombre, apellidos, fecha de nacimiento, domicilio, código postal y número de teléfono. Al igual que en los demás formularios de este apartado, se envían los datos con una llamada *POST* que viaja hasta "userRouter". Aquí se comprueba que los datos son válidos, si lo son, estos son actualizados en la base de datos a través de un dao que hace una llamada de UPDATE al usuario. Tras cambiar los datos en la base de datos se cambian también en la sesión actual. Durante la edición

del perfil, si ocurren errores se redirige a la página de edición del perfil y se muestra un mensaje de error.

6.4.1.4 Eliminar Cuenta

Para eliminar una cuenta el doctor debe pulsar un botón en el *navbar* que muestra un modal que pregunta por una confirmación para eliminar la cuenta. Si se confirma se genera una llamada *GET* que acaba en el "doctorRouter" y ejecuta múltiples llamadas al DAO (ver figura 6.11) que realizan operaciones de DELETE de las diferentes tablas donde se almacenan los distintos datos del doctor. Finalmente se redirecciona al inicio. Nótese que en las funciones del DAO se utilizan como parámetro de entrada el dni del usuario de la sesión actual ("req.session.currentUser.dni").

```
await dao.bajaDoctor_notificaciones(req.session.currentUser.dni);
await dao.bajaDoctor_consultas(req.session.currentUser.dni);
await dao.bajaDoctor_citas(req.session.currentUser.dni);
await dao.bajaDoctor_tratamientos(req.session.currentUser.dni)
await dao.bajaDoctor(req.session.currentUser.dni);
await dao.bajaDoctor_asignaciones(req.session.currentUser.dni);
```

Figura 6.11- Llamadas a funciones de bajas de doctor al dao

6.4.2 Módulo Gestión de Asignaciones

La gestión de asignaciones se centra en la tabla de la lista de usuarios, donde se muestran los usuarios que un doctor tiene asignado. En esta tabla se permite tanto asignar nuevos pacientes como eliminar antiguos pacientes. Inicialmente en la página web la tabla está vacía pero, al inicializarse o cuando ocurre algún cambio como filtrado, vinculación o desvinculación de un paciente, la tabla se actualiza con *JQUERY (ver figura 6.12)*.

```
$(document).ready(function () {
    var usuarios = [];
    var usuariosFiltrados = [];
```

Figura 6.12- Inicialización de página con script JQUERY

Como ya se ha especificado se declaran variables vacías. Para conseguir las variables también se llama a una función que ejecuta una función *AJAX* (ver figura 6.13)

```
function getUsuarios() {
    $.ajax({
        url: `/doctor/getUsuarios/`,
        type: 'GET',
        success: function(data) {
           usuarios = data;
           usuariosFiltrados = data;
           cargarBuscador();
           $("#filtro").text("Nombre");
           mostrarUsuarios();

        },
        error: function(error) {
            console.error('Error en la solicitud AJAX:', error);
        }
    });
}
```

Figura 6.13- Función AJAX de obtención y carga de usuarios

En esta función se realiza una llamada *GET* al "doctorRouter" donde se llama al DAO para obtener los pacientes. Luego estos pacientes se formatean para pasarlos como datos ".json" a la llamada *AJAX* (ver figura 6.14)

```
var pacientes = await dao.obtenerPacientes_doctor(req.session.currentUser.dni);

pacientes = pacientes.map(paciente => {
  return {
      Nombre: paciente.Nombre,
      Apellidos: paciente.Apellidos,
      email: paciente.email,
      dni: paciente.dni
  };
);
res.json(pacientes);
```

Figura 6.14- Obtención, formateo y envío de datos en formato .json

Tras hacer la llamada al *router* si se ha ejecutado con éxito se ejecuta una función definida en la parte "*success*" de la función *AJAX*. En esta parte se guardan los usuarios obtenidos, se carga el buscador y se llama a la función mostrar usuarios,que por cada usuario inserta una fila. Por cada fila se añaden los datos y botones de cada usuario (ver Figura 6.15)

```
$fila.find('td').eq(0).append(usuario.Nombre);
$fila.find('td').eq(1).append(usuario.Apellidos);
$fila.find('td').eq(2).append(usuario.email);
$fila.find('td').eq(3).append(usuario.dni);
$fila.find('td').eq(4).append($buttonFuncionesPaciente);
$fila.find('td').eq(5).append($buttonEliminar);


$('#tablaHistorial').append($fila);
```

Figura 6.15- Inserción de fila de datos de usuario en la lista de usuarios

Si ocurre algún fallo se ejecuta la parte "error" y se manda un mensaje por consola.

6.4.3 Vincular Médico Paciente

La vinculación de cada paciente se hace a través de un botón, que abre un modal con un formulario que pregunta por un dni o correo electrónico del paciente. Al confirmar hace una llamada *POST* al "doctorRouter". Este ejecuta una función que comprueba si el dato insertado contiene un carácter "@" perteneciente a un correo electrónico. Dependiendo de esto busca llamando a una función del DAO que comprueba que exista el paciente y obtenga sus datos. Si no se encuentra al paciente con dicho dato se lanza un error que se recoge en la vista y muestra un mensaje de error. Después de obtener los datos del paciente, cuando existe, se llama de nuevo al DAO y se añade el dni del paciente y del doctor a la tabla asignaciones.

Finalmente se redirecciona a la página web con la lista de usuarios, lo que los recarga. De esta manera se muestra al nuevo paciente asignado. Si ocurre algún fallo se redirecciona a la página con un mensaje de error.

6.4.4 Desvincular Médico Paciente

Al presionar el botón de eliminar al lado de un paciente se ejecuta una función que llama a un enlace con el dni del paciente a eliminar (ver figura 6.16). Dicha llamada llega al "doctorRouter" que a través del DAO elimina la asociación entre el doctor y el

paciente con una función *DELETE* de *SQL* con el dni de ambos. Si ocurre un error se redirige a la página de gestión de usuario con un mensaje de error y si no hay problemas se redirige pero con un mensaje de éxito.

```
$buttonEliminar.data('usuario', usuario);
$buttonEliminar.on('click', function(){
    window.location.href = `/doctor/eliminarAsociacion/${usuario.dni}`;
});
```

Figura 6.16- Llamada a la eliminación de asociaciones

6.4.5 Módulo Gestión de Historial Doctor

6.4.6 Crear Historial Médico

Para la creación del historial médico se usa un formulario con campos estáticos y dinámicos. Los campos dinámicos son *checkboxes* de las enfermedades que tiene el doctor que crea el historial. Estos campos se obtienen al cargar la página, el resto de campos son estáticos.

Cuando el doctor rellena los campos del formulario debe presionar un botón para crear el historial que hace una llamada al "doctorRouter" que redirige al *router* "historialRouter" desde el cual se ejecuta una función. En esta se llama a "aniadirJson()" el cual crea un fichero ".json" con todos los datos en la carpeta con la ruta "/private/historiales/json/". Adicionalmente también se ejecuta la función "crearPDF()" la cual a través de los recursos que otorga el módulo "PDFKit" crea un fichero de tipo ".pdf" con los datos del ".json" anterior y lo guarda en la ruta "/private/historial/". Finalmente se redirecciona a la página de las funciones de usuario.

6.4.7 Añadir Enfermedad para Historial Médico

Durante la creación del historial, en el formulario donde se guarda la información del paciente existe otro formulario que permite añadir enfermedades. Al hacer clic en el botón "Añadir enfermedad", se obtiene la enfermedad inscrita en el formulario, se añade un nuevo *checkbox* de la enfermedad y se llama al *router* con un método *POST* para guardar permanentemente la enfermedad en la tabla de enfermedades. Esta

llamada pasa por "doctorRouter" que la envía al "usuarioRouter" y este ejecuta una función *INSERT* del DAO que recibe como parámetro la enfermedad y el dni del doctor.

6.4.8 *Añadir Detalles al Historial Médico*

Para añadir detalles al historial médico se muestra un modal que inicialmente está escondido. Este modal (ver figura 6.17) contiene un formulario con varios tipos de datos escondidos que permiten su uso posteriormente . También contiene un campo de texto donde el doctor inserta los detalles a añadir. Si se decide añadir el detalle se realiza una llamada *POST* con los datos que finalmente llegan al "historialRouter".

```
<div class="modal-body">
  <form action="/doctor/historial/aniadirDetalles" method="post">
    <input type="hidden" name="dni" value="<%= dni %>">
    <input type="hidden" name="nombrePaciente" value="<%= nombrePaciente %>">
    <input type="hidden" name="apellidosPaciente" value="<%=apellidosPaciente%>">
    <label for="detalles">Detalles:</label>
    <textarea class="form-control" id="detalles" name="detalles" rows="3"></textarea>
    <div class="modal-footer">
      <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Cancelar</button>
      <button type="submit" class="btn btn-primary">Guardar</button>
    </div>
  </form>
</div>
```

Figura 6.17- Modal de añadir detalles

Para añadir el detalle se accede al fichero ".json" del historial médico (almacenado en la ruta "/private/historiales/json/") del paciente y se le añade al campo de detalles el nuevo detalle. Si no existe el fichero ".json" se redirige al doctor a la página de funciones de usuario con un mensaje de error. Si existe el fichero, posteriormente se crea un nuevo ".pdf" con el nuevo ".json" al igual que en el apartado anterior a través de la función "crearPDF()". Finalmente se redirecciona la página con un mensaje de error o de éxito, dependiendo de si ha habido un error en el servidor.

6.4.9 *Ver Historial Médico*

Al presionar el botón de ver historial se ejecuta una función *AJAX* que realiza un método *GET* que llega hasta el "historialRouter" el cual comprueba que existe el historial médico (en la ruta "/private/historiales/"). Si no existe manda un error que muestra un

mensaje de error en forma de un modal en la página de funciones de usuario y si existe manda la ruta. Tras mandar la ruta, la función *AJAX* crea un enlace a la ruta y hace clic en él automáticamente, lo cual descarga el ".pdf" del historial médico (ver figura 6.18).

```
$.ajax({
  url: '/doctor/historial/obtenerPDF',
  type: 'GET',
  contentType: 'application/json',
  data: { dniPaciente: dni, nombrePaciente:nombrePaciente, apellidosPaciente:apellidosPaciente },
  success: function(response) {

    const downloadLink = document.createElement('a');
    downloadLink.href = '/doctor/historial/obtenerPDF?dniPaciente=' + dni;
    downloadLink.download = 'Historial-' + dni + '.pdf';
    document.body.appendChild(downloadLink);
    downloadLink.click();
    document.body.removeChild(downloadLink);
  },
  error: function(response) {
    var errorModal = $('#errorModal');
    errorModal.find('.modal-body').text('El historial del paciente no se pudo encontrar');
    errorModal.modal('show');
  }
});
```

Figura 6.18- Función AJAX de obtención y descarga de historial médico

6.4.10 Eliminar Historial Médico

Cuando se quiere eliminar el historial se hace una llamada *AJAX* que a su vez hace una llamada *DELETE*. Esta acaba en "historialRouter" que ejecuta una función la cual busca el historial médico en ambos formatos (".pdf" y ".json") si no encuentra los ficheros en sus respectivas rutas ("/private/historiales/" y "/private/historiales/json/") se devuelve un error y la función *AJAX* muestra un modal con un mensaje de error. Si se encuentran los ficheros devuelve un estado de éxito y se muestra un modal con mensaje de éxito.

6.4.11 Módulo Gestión de Citas Doctor

6.4.11.1 Crear Cita

En esta funcionalidad se le muestra al doctor un formulario en un modal donde debe indicar la fecha, la hora, el asunto de la cita, y el tiempo dedicado. Estos datos

son recogidos en el router donde se adaptan las fechas para mandarlas a la base de datos (ver figura 6.19):

```
if (nuevoDate == "Invalid Date") {

  var fechaFormateada = formatearFecha(fecha,hora);
  fecha_hora = fechaFormateada;


}else{

  //Construir la fecha en el formato 'YYYY-MM-DD HH:MM:SS'
  const year = nuevoDate.getFullYear();
  const month = String(nuevoDate.getMonth() + 1).padStart(2, '0');
  const day = String(nuevoDate.getDate()).padStart(2, '0');
  const hour = String(nuevoDate.getHours()).padStart(2, '0');
  const minutes = String(nuevoDate.getMinutes()).padStart(2, '0');
  const seconds = String(nuevoDate.getSeconds()).padStart(2, '0');

  const fechaParaBD = `${year}-${month}-${day} ${hour}:${minutes}:${seconds}`;

  fecha_hora = fechaParaBD;

}
```

Figura 6.19- Adaptación de fechas para bases de datos

Después de adaptar estas fechas se realizan una serie de validaciones donde se comprueba el dni, si el paciente al que se le va asignar una cita existe, si la duración introducida es mayor a 0 e inferior a 60 y si el día de la cita es superior al actual(ver figura 6.20).

Una vez se han comprobado, el DAO llama a un método que ejecuta un proceso de la base de datos(ver figura 6.21) para comprobar si existe una cita en ese intervalo de tiempo, en caso de que exista se le muestra al usuario un mensaje de error indicando de que ya existe una cita en ese intervalo de tiempo.

```
try {
  if (!/^\d{8}[a-zA-Z]$/.test(dni)) {
    errorMessage = "Formato de DNI inválido"
    throw new Error("Formato de DNI inválido");
  }

  const paciente = await dao2.obtenerPaciente(dni);
  if (paciente.length === 0) {
    errorMessage = "El paciente no existe"
    throw new Error("El paciente no existe");
  }

  if (duracion <= 0 || duracion > 60) {
    errorMessage = "La duración de la cita debe estar entre 1 y 60 minutos"
    throw new Error("La duración de la cita debe estar entre 1 y 60 minutos");
  }

  if (nuevoDate <= fechaActual) {
    errorMessage = "La fecha y hora seleccionadas deben ser posteriores a la fecha y hora actual"
    throw new Error("La fecha y hora seleccionadas deben ser posteriores a la fecha y hora actual");
  }

  var citasCoincidentes = await dao.checkearCitasCoincidentes(req.session.currentUser.dni, fecha_hora, duracion);
  if (citasCoincidentes.length > 0) {
    errorMessage = "Ya existe una cita en ese intervalo de tiempo"
    throw new Error("Ya existe una cita en ese intervalo de tiempo");
  }

  await dao.asignarCita(req.session.currentUser.dni, dni, fecha_hora, duracion);

  res.render('calendario', { nombre: req.session.currentUser.nombre });
} catch (error) {
  errorMessage= "Error al asignar cita: "+ errorMessage
  res.render("funcionesUsuario",{ nombre:req.session.currentUser.nombre,nombrePaciente:nombrePaciente,apellidosPaciente:apellidosPaciente,dni:dni, correct:"", error:errorMessage});
}
```

```
connection.query('CALL ComprobarCita(?, ?, ?)', [fecha_hora_formateada, duracion, DNIDoctor], function (error, results, fields) {
    if (error) {
        console.error('Error al llamar al procedimiento almacenado:', error);
        reject(error);
    } else {
        console.log('Citas coincidentes:', results[0]);
        resolve(results[0]);
    }
});
```

Figura 6.20- Validaciones Citas

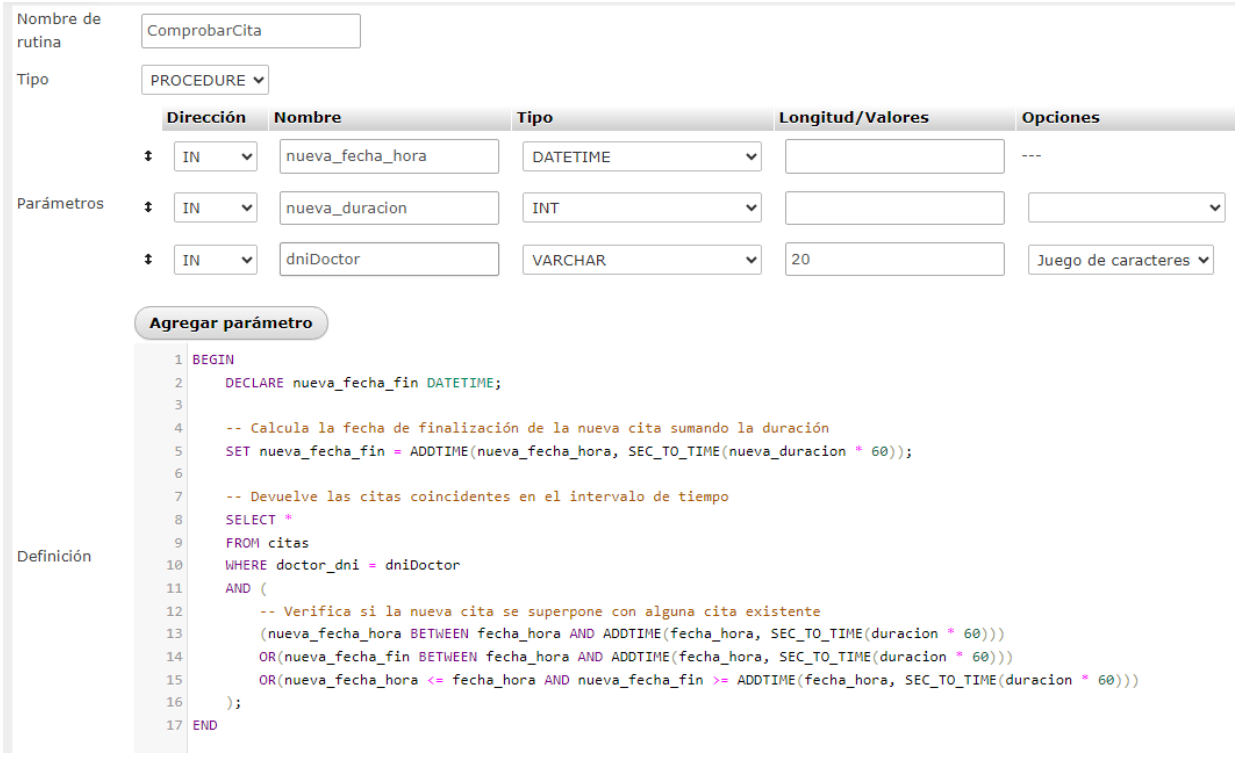

Figura 6.21- Procedimiento chequear Citas

6.4.11.2 Mostrar Citas

Para mostrar citas en la web se ha usado la librería de *JavaScript FullCalendar*, para mostrar las citas en un calendario con diferentes vistas mensuales,semanales y diarias, facilitando así la gestión al doctor.

A la hora de crear el calendario en el "calendar.js", se crea un objeto de tipo FullCalendar que recibe el div padre donde se va a mostrar el calendario y un JSON con una serie de configuraciones para la interfaz (ver figura 6.22):

1. "InitialView", define el modo de vista inicial para el calendario en este caso la vista mensual. "SlotDuration" y el "slotLabelInterval" establecen intervalos de quince minutos para mostrar los eventos o citas.
2. "BusinessHours", establece en la vista diferentes colores, un gris más oscuro cuando está fuera del horario y un gris más claro cuando está dentro del horario.
3. "HeaderToolbar", configura la barra de arriba. En este caso añade los botones: siguiente (">") o anterior("<") mes, día o semana; el botón "today" muestra la vista con el día de hoy y los diferentes botones para cambiar la vista.

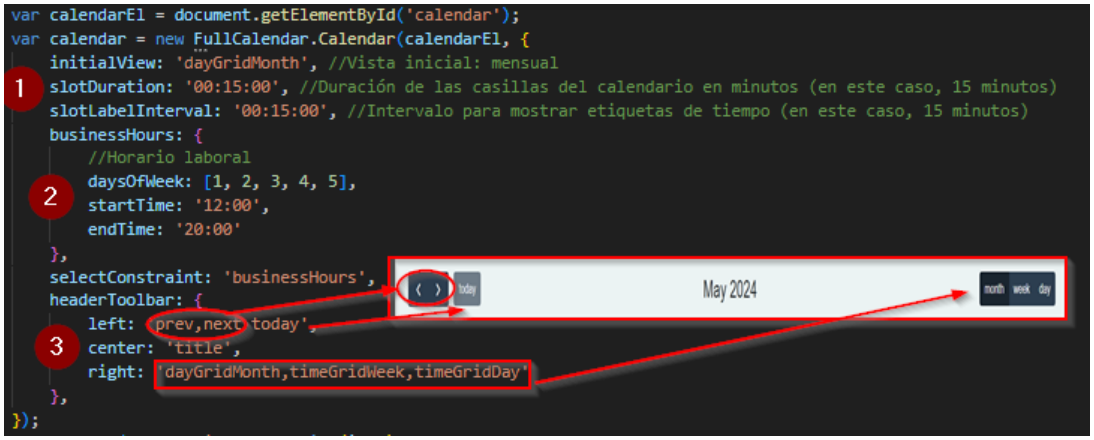

Figura 6.22- Código configuración FullCalendar

Al calendario se le añade un evento cuando se hace clic en una cita para que muestre un modal con la información de una cita y el botón para poder eliminarla (ver figura 6.23), en caso de que presione en eliminar, manda una petición *AJAX* al servidor pasándole el *id* de la cita. Si todo va bien recarga la vista de citas.

```
calendar.on('eventClick', function(info) {
    var evento = info.event;
    var modalContent = `
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title">${evento.title}</h5>
                <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
            </div>
            <div class="modal-body">
                <p>Inicio: ${evento.start}</p>
                <p>Fin: ${evento.end}</p>
                <!-- Otros detalles del evento aquí -->
            </div>
            <div class="modal-footer">
                <button id="eliminarCitaBtn" type="button" class="btn btn-danger">Eliminar Cita</button>
                <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Cerrar</button>
            </div>
        </div>
    `;
    var modalElement = document.getElementById('myModal');
    var modalBody = modalElement.querySelector('.modal-body');
    modalBody.innerHTML = modalContent;
    var modal = new bootstrap.Modal(modalElement);
    modal.show();

    document.getElementById('eliminarCitaBtn').addEventListener('click', function() {
        eliminarCita(evento.id);
        modal.hide();
    });
});
```

Figura 6.23- Modal eliminar

Además de esto, se ha configurado un evento para que cuando se haga clic en una casilla de un día se mande a la vista diaria de ese día facilitando mucho más la navegación por el calendario al doctor.

Después de configurar el calendario, una vez se ha cargado la página, se manda una solicitud *AJAX* al *router* de citas que llama al DAO para obtener todas las citas. En el JavaScript se guardan estas en un objeto eventos que es asignado al calendario (ver figura 6.24).

Figura 6.24- Cargar Citas

6.4.12 *Módulo Gestión de Tratamientos Doctor*

Cuando se quiere añadir un tratamiento a través del botón de la página de funciones de usuario se muestra un modal con un formulario. Este tiene varios campos incluyendo una tabla con varios *input.* Al final de esta tabla se puede presionar a un botón que agrega una fila a la tabla (ver figura 6.25), la cual te permite añadir un nuevo medicamento.

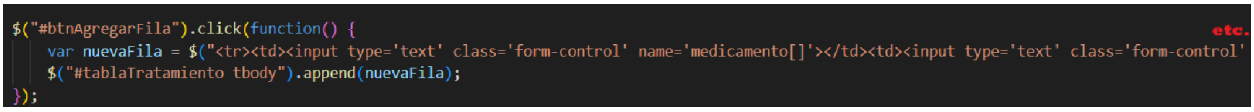

Figura 6.25- Botón para agregar filas

Al confirmar el formulario se hace una llamada *post* que llega al *router* "doctorRouter" que ejecuta una función. En esta primero se comprueba que los datos se han insertado, luego comprueba si hay un único medicamento o varios y dependiendo de esto se ejecuta el código una única vez o varias en un "for()" . El código comprueba que los datos de cada medicamento sean válidos (comprueba que las dosis y tomas al día sean mayor que 0 y que el tratamiento se realice en una fecha

posterior a la actual). Si algún dato es inválido se vuelve a la página y devuelve un mensaje de error.

Si todos los datos son válidos se hacen dos llamadas al DAO (si hay varios medicamentos se hace una llamada adicional por cada medicamento extra), las cuales ejecutan funciones INSERT en las tablas de tratamiento y alarmas, para su uso en la parte de la aplicación web.

Finalmente, si todo se ha ejecutado correctamente, se manda a la página de funciones de usuario con un mensaje de éxito, mientras que si ha ocurrido un error se manda un mensaje de error.

6.4.13 Módulo Gestión de Consultas Doctor

6.4.14 Carga de Consultas

Para cargar la página de las consultas se hace una llamada al DAO donde se obtienen las consultas del doctor con su información básica a través de una función SELECT. Cuando se hace clic en una consulta específica, se esconden las consultas y se hace una llamada *GET* con *AJAX*. De esta manera se llega al *router* de "consultaRouter" y se obtienen los mensajes al llamar a una función SELECT del DAO. Estos mensajes se mandan en formato *".json"* y se muestran en en un nuevo *div* que anteriormente estaba escondido. Adicionalmente al entrar en una consulta también se llama a una función *POST* con "fetch()" que borra las notificaciones de los mensajes del doctor, esto se hace también en la "consultaRouter" donde se actualiza el valor de las notificaciones del doctor a 0 a través de una función UPDATE en el DAO. De esta manera el doctor podrá saber qué mensajes son nuevos y qué mensajes ha leído ya.

6.4.15 Envío de Mensajes

Mediante un formulario se pueden enviar mensajes, al igual que en la carga de los mensajes se llama (con una llamada *POST*) a una función que acaba en el *router* "consultaRouter" y donde se obtienen al ejecutar un comando INSERT con el nuevo mensaje del doctor.

6.4.16 Actualización de Mensajes

Una vez se cargan los mensajes al presionar en una consulta concreta, a través de un *script* se eliminan los mensajes y se vuelven a obtener con una llamada *GET* en *AJAX* que llega al "consultaRouter". Desde aquí se obtienen los mensajes y se muestran nuevamente. De esta manera, el doctor puede verlos en tiempo real sin necesidad de cargar la página de nuevo.

Capítulo 7 - Conclusiones y Trabajo Futuro

7.1 Conclusiones

En conclusión, en este TFG se ha implementado una aplicación[1] en dos dispositivos diferentes (móvil y web) con el fin de cumplir el objetivo marcado: "elaborar una aplicación capaz de facilitar el uso a pacientes y doctores, mejorando la comunicación entre ambos y optimizando el proceso de atención médica."

Dentro de las marcas que se establecieron para lograr este objetivo, se ha investigado las dificultades y deseñado la aplicación. De tal forma que la aplicación se ha estructurado e implementado los módulos funcionales mencionados para lograrlo, dejando por otro lado para futuro la elaboración de pruebas, garantizar la seguridad, pruebas en un sector real y el análisis de resultados.

Referente a la experiencia del usuario se ha llevado a cabo un diseño sencillo y fácil de manipular con el objetivo que la aplicación sea accesible y manipulable por todo tipo de usuarios.

7.2 Trabajo a Futuro

En esta sección se van a abordar las funcionalidades o retoques que se han dejado como cambios a futuro después del desarrollo de estos últimos meses.

Se van a visualizar en la tabla 3 una serie de cambios a realizar tanto para el dispositivo móvil como para la aplicación web.

[1] Repositorio de la aplicación: https://github.com/Antoranz/MedAlerta.git

CAMBIOS	PLATAFORMA
Tema oscuro.	Ambos
Feedback a los usuarios sobre qué horarios ya han sido reservados para no solicitar una cita en ese intervalo.	Móvil
Añadir una posibilidad para que los pacientes puedan deshabilitar y habilitar alarmas.	Móvil
En los mensajes se indique la hora en la que han sido enviados	Ambos
Distinguir los mensajes en el chat por días.	Móvil
Buscar una alternativa con webSockets u otras herramientas para actualizar los mensajes.	Ambos
Subir el servidor a un host en la nube.	Ambos
Utilizar herramientas de seguridad para ver las vulnerabilidades del código y mejorarlas.	Ambos
Enviar confirmaciones de citas aceptadas al correo electrónico del paciente.	Web
Especificar chats no leídos en la notificación.	Móvil
Recuperar Contraseña.	Web
Validar correo.	Web
Hacer más bonito el PDF.	Web
Usuario Administrador.	Web
Multiplataforma, que los pacientes puedan realizar las funcionalidades en web y los doctores en móvil también.	Ambos
Tests automáticos.	Ambos
Añadir imágenes de perfil de usuario.	Ambos

Tabla 3- Tabla de cambios a futuro.de cambios a futuro.

Chapter 7. -Conclusions and future work

7.1 Conclusions

In conclusion, in this TFG an application[2] has been implemented on two different devices (mobile and web) in order to meet the objective set: "to develop an application capable of facilitating the use of patients and doctors, improving communication between them and optimizing the medical care process."

Within the brands that we set ourselves to achieve this goal, the difficulties have been investigated and the application designed. In such a way that the application has been structured and implemented the functional modules mentioned to achieve this, leaving for the future the elaboration of tests, guaranteeing security, tests in a real sector and the analysis of results.

Regarding the user experience, a simple and easy-to-use design has been carried out with the aim of making the application accessible and manipulable by all types of users.

7.2 Future Work

In this section we are going to address the functionalities or tweaks that have been left as future changes after the development of these last few months.

A series of changes to be made for both the mobile device and the web application will be displayed in Table 4.

CHANGES	PLATFORM
Dark theme.	Both
Feedback to users about which times have already been reserved so as not to request an appointment in that interval.	Mobile
Add a possibility for patients to disable and enable alarms	Mobile

[2] Application Repository: https://github.com/Antoranz/MedAlerta.git

Messages indicate the time they were sent.	Both
Distinguish messages in chat by days.	Mobile
Search an alternative with webSockets or other tools to update messages.	Both
Upload the server to a cloud host.	Both
Use security tools to see code vulnerabilities and improve them.	Both
Send confirmations of accepted appointments to the patient's email.	Web
Specify unread chats in the notification.	Mobile
Recover password.	Web
Validate email.	Web
Make the PDF more beautiful.	Web
User Administrator.	Web
Multiplatform, so that patients can perform the functionalities on the web and doctors can also perform the functions on mobile phones.	Both
Automatic tests.	Both
Add user profile images.	Both

Tabla 4- Table of future changes. f future changes.

CONTRIBUCIONES PERSONALES

Francisco Javier Antoranz Esteban

Durante los primeros dos meses el alumno estuvo investigando, documentando y registrándose en *firebase* para poder realizar una primera implementación de la web. Después de un proceso tedioso con una documentación desactualizada, se decidió por abandonar *firebase* y hacer el proyecto en local con *Xampp*.

El alumno participó en la elaboración del registro e inicio de sesión del móvil donde se investigó acerca del funcionamiento de cómo mantener una sesión iniciada, aunque se mantenga cerrada la aplicación (utilizando *sharedPreferences*).

Javier se implicó en la elaboración e investigación del uso de la librería de *mailer.js* para enviar correos electrónicos desde la cuenta "medalertatfg@gmail.com" a los usuarios de la aplicación. Se visualizaron una serie de videos que ayudaron a la configuración de la cuenta de *google* y a la creación del código *JavaScript* para poder configurarlo correctamente.

Ligado al funcionamiento del correo, se colaboró en la recuperación de la contraseña y la validación del correo al iniciar sesión en la aplicación móvil donde también se utilizó *mailer.js* para enviar códigos de seguridad al usuario.

En la aplicación web el alumno desarrolló: los tratamientos; la funcionalidad del listado de usuarios y sus búsquedas por sus diferentes atributos (nombre,DNI, apellidos,etc); el diseño de la vista de editar perfil; brevemente en una primera plantilla para la creación de un historial (vista ".ejs"), donde se estuvo investigando en una primera instancia en como guardar estos PDFs en *firebase*; y participó en una primera parte en el diseño global de la aplicación web.

Para poder realizar estas funcionalidades, se investigó acerca del funcionamiento de *Bootstrap* para lograr una aplicación *responsive*.

Francisco investigó acerca del funcionamiento de la librería de *JavaScript Full Calendar*, leyendo su documentación y aplicándolo para poder ver las citas que un doctor tenía al día, a la semana y al mes.

El alumno participó en el diseño y la elaboración de la creación y la aceptación de una cita donde debido a unos problemas que sucedieron con las validaciones de las fechas, el alumno tuvo que crear un procedimiento en *Xampp* para poder solucionarlo.

Con relación al móvil el alumno participó en el diseño general y la estructura de la aplicación. El alumno se basó en el modelo de vista controlador aprendido en la universidad para poder aportar una estructura al proyecto basado en este modelo. Javier se encargó de la separación de las funcionalidades en el móvil aislando el código situado en las actividades en otras carpetas o paquetes en función de donde le correspondiera, con el objetivo de llevar a cabo un buen desarrollo *software* y aislar las clases útiles, las peticiones del servidor y otras funcionalidades de las vistas.

Con relación al diseño Francisco se encargó de investigar el funcionamiento de los *fragments* y de la creación de los menús y botón *navbar* visualizando diferentes videos de *youtube* y mirando la teoría aprendida en la optativa de PAD. En este proceso adaptó todas las clases *activities* a *fragments* teniendo que cambiar el funcionamiento de la gestión de las pantallas.

Con respecto a las consultas el estudiante participó en el diseño de las consultas, donde se investigó acerca del funcionamiento de los "RecyclerView" y *Adapters* que tanto se usa en la aplicación. Participando en la implementación de estas clases para poder mostrar esta lista de mensajes y refrescar los mensajes cada poco tiempo.

El universitario desarrolló la vista de mostrar citas donde se investigó para poder mostrar un calendario parecido al de la web donde le aparecen al paciente los días que tiene una cita marcados. Después de una serie de problemas con compatibilidad de librerías el estudiante decidió hacer una lista de citas y mostrar un calendario.

El alumno participó en la investigación de las llamadas *API* desde la aplicación móvil y web con el objetivo de poder recibir y mandar información a la base de datos.

En relación a esto último y con el objetivo de probar la aplicación en dispositivos móviles el estudiante hizo una búsqueda profunda de como compilar la aplicación móvil y conectarse al servidor en *localhost* del ordenador. Después de una larga búsqueda se probó *ngrok*, permitiendo probar la aplicación en otros dispositivos móviles en vez de emuladores.

En relación a la base de datos se trabajó conjuntamente acerca de las propiedades y características que tenían que tener cada tabla compartiendo cualquier sugerencia o cambio que se hiciese sobre esta.

Sergio Sánchez Chamizo

Uno de los primeros avances en los que el estudiante trabajó, fue la investigación de la base de datos de *firebase* en donde se empezó investigando la gestión del funcionamiento de las sesiones.

El alumno se implicó en la creación de los registros e inicios de sesión tanto en la web como en el móvil, lo que incluye también la gestión de las sesiones en ambas aplicaciones.

Al inicio, el alumno participó en la primera instancia de la plantilla del historial médico del paciente, haciendo una investigación de cómo son los historiales médicos de los pacientes en los hospitales y adaptándolo al proyecto.

Sergio estuvo implicado en la creación del email de medAlerta llamado "medlatertatfg@gmail.com" y en la gestión necesaria para enviar correos a los pacientes mediante la librería *mailer.js*.

El estudiante participó de manera directa en la implementación y en el diseño de la funcionalidad para el cambio de contraseñas del móvil, la cual envía un código al correo para poder aprobar el cambio de contraseña y poder gestionar el cambio. Esta funcionalidad se encuentra tanto al iniciar sesión como en el apartado de editar perfil del paciente.

En la aplicación web, el alumno colaboró en el desarrollo de la gestión de citas del doctor, que corresponde al listado de peticiones solicitadas por el paciente al mismo tiempo que estuvo implicado en la comparación correcta de las fechas de las citas en el calendario.

Referente al correo también, Sergio participó tanto en el diseño como en la implementación de la validación de la cuenta del paciente al iniciar sesión, para confirmar su identidad.

En la creación de las tablas en la base de datos *MySQL* se iba participando en función de lo que se iba necesitando a medida que el proyecto avanzaba.

Sergio también estuvo trabajando en el diseño y en la implementación de la edición y guardado del perfil del paciente.

Respecto al móvil, el estudiante contribuyó en la gestión de las llamadas del móvil al servidor *node.js*, investigando la manera de enviar y recibir datos mediante llamadas *GET* y *POST*. También colaboró en el diseño y la implementación de la aplicación con la ayuda de los conocimientos aprendidos en la optativa PAD.

El estudiante estuvo investigando diferentes fuentes para la implementación de las alarmas en el móvil y todas las sub-alarmas en función de la fecha y la hora establecida por el doctor. En primer lugar se centró en la gestión sobre el cómo guardar y ajustar las alarmas con una hora y fecha determinadas, tras ello, el manejo del recibo de las mismas y por último la configuración de las notificaciones y del canal para que estas sean enviadas al móvil con un sonido y un mensaje determinados.

El alumno participó en la creación de la solicitud de una cita por parte del paciente, tanto en el diseño como en la implementación.

El estudiante estuvo implicado en el diseño por carpetas de los *service* y sus implementaciones los cuales se utilizaron para comunicarse con el servidor *node.js* para enviar y recibir datos. También participó en la organización y estructura de las carpetas para separar las actividades de los tipos, o los adaptadores de los *receivers*, además de la creación de un *controller* para facilitar la organización del código. Se basó en el modelo vista controlador que fue aplicado en varios proyectos de la universidad.

Referente a las consultas, el universitario colaboró en el diseño de la visualización de las consultas y la creación de éstas que es muy similar al formulario de las citas. También participó en el diseño y en la configuración del chat entre el doctor y el paciente, distinguiendo los mensajes de los usuarios y modificando el "recyclerView" para que la pantalla se inicie automáticamente al final y que se muestren los mensajes al estilo de un *chat* convencional.

Referente a las notificaciones de los mensajes, el alumno participó en la implementación de manera similar con las alarmas, sin embargo, tras investigar para que las notificaciones funcionasen, aunque la aplicación cerrase, hubo que guardar en *sharedPreferences* el dni del usuario para que las notificaciones sigan llegando.

Carlos Peña Pan

En el inicio del trabajo el estudiante se dedicó a familiarizarse con el entorno y decidió junto a sus compañeros los métodos y entornos de trabajo.

En la creación de tablas en la base de datos se iba participando en función de lo que se iba necesitando a medida que el proyecto avanzaba.

El alumno participó en la creación de los registros e inicios de sesión en la web, adicionalmente analizó diferentes maneras y estrategias para el almacenaje de contraseñas y se decidió por un método de cifrado.

Para la creación del historial médico, el estudiante participó en la creación del cuestionario del historial, añadiendo un sistema para que los doctores en el cuestionario obtuvieran las enfermedades por defecto a través de un ".json". Añadir la posibilidad de poder ingresar enfermedades nuevas y que estas se almacenen de manera persistente. También investigó diferentes métodos de creación de ficheros ".*pdf*". Tras la investigación se dedicó a la implementación de la gestión de historiales médicos a través de ficheros ".pdf" y ".json". Adicionalmente también fue encargado de almacenar de forma segura dichos ficheros críticos de la aplicación, lo que implicó alternar la estructura de almacenamiento del servidor con la creación de una carpeta privada.

Tras la creación del historial se dedicó a otorgar a los doctores la capacidad de descarga de los archivos de los historiales médicos a través de una ruta segura.

El universitario se encargó también de la modificación del historial médico mediante el uso de los detalles y la eliminación del documento.

El estudiante participó en la creación de la barra del navegador y el sistema principal de navegación en la aplicación web. También creó la página web de inicio de *MedAlerta*.

A lo largo del proyecto el alumno realizó una refactorización del código que implicó cambios en la estructura de enrutación del móvil, web y servidor, con la finalidad

de mejorar y aislar las diferentes llamadas, ya que anteriormente todas las rutas se conseguían a través de un único *router*. Para conseguir esto creó nuevas rutas y routers, dividiendo el trabajo en secciones dependiendo del uso que se le dio a cada ruta individual.

En cuanto las asignaciones entre doctores y pacientes el universitario contribuyó en la creación de la tabla dinámica de la lista de usuarios.

Carlos diseñó por la aplicación web el sistema de las consultas y mensajes entre doctor y paciente, donde realizó el sistema de envío y recepción de mensajes, incluyendo la actualización de los mensajes y la gestión de notificaciones de mensajes leídos del doctor y paciente para la vista de mensajes nuevos en las consultas.

A lo largo del desarrollo del trabajo el alumno participó en la gestión de errores del sistema web creando vistas y modales para mostrar a los usuarios el éxito o error de las funcionalidades del trabajo.

El usuario colaboró con la creación de las notificaciones de citas pendientes del doctor y creó la lista de citas como un método alternativo al calendario.

El universitario investigó e implementó un sistema de tests de prueba mediante la librería *Jest*, pero tras un corto estudio se decidió que los *tests* manuales serían más eficientes y el alumno se deshizo de los *tests automáticos* de prueba.

En los últimos días del trabajo el alumno se dedicó a realizar un rediseño final de la interfaz de usuario web. Esto incluye el uso de tecnologías como *Bootstrap* y otros métodos con la finalidad de hacer la aplicación web más sencilla, limpia y accesible para el público. Todas las vistas web fueron afectadas en este cambio

BIBLIOGRAFÍA

[1] Whatsapp: https://www.whatsapp.com/?lang=es_LA

[2] MyChart: https://www.mychart.org/

[3] HealthTap: https://www.healthtap.com/

[4] Zocdoc: https://www.zocdoc.com/

[5] Practice Fusion: https://www.practicefusion.com/

[6] JavaScript: https://es.wikipedia.org/wiki/JavaScript

[7] FS: https://nodejs.org/api/fs.html

[8] path: https://nodejs.org/api/path.html

[9] FullCalendar.js: https://fullcalendar.io

[10] PDFKit: https://pdfkit.org/

[11] CSS: https://es.wikipedia.org/wiki/CSS

[12] HTML: https://es.wikipedia.org/wiki/HTML

[13] JAVA: https://www.java.com/es/

[14] XML: https://www.w3.org/XML/

[15] Node.js: https://nodejs.org/en

[16] Bootstrap: https://getbootstrap.com/

[17] EJS: https://ejs.co/

[18] Visual Studio Code: https://code.visualstudio.com/

[19] Android Studio: https://developer.android.com/?hl=es-419

[20] MySql: https://www.mysql.com/

[21] Xampp: https://www.apachefriends.org/es/index.html

APÉNDICES

Apéndice A - Casos de Uso

Durante este capítulo, se expondrán los diferentes casos de uso que se han presentado en la aplicación, así como la forma de actuar de los actores en estos.

Actores

En el modo en el que se interactúa con la aplicación se distinguen tres tipos de usuarios: el paciente, el doctor y el usuario no registrado.

El paciente es un usuario que busca atención y asistencia médica continua. El paciente disfruta de la aplicación en formato *app* de móvil, e interactúa con esta solicitando citas, enviando consultas a doctores y recibiendo alarmas y notificaciones.

El doctor es un profesional de salud que ofrece sus servicios a través de la aplicación facilitando una comunicación cercana y continua con el paciente. El doctor usando la plataforma puede gestionar su agenda de citas, configurar tratamientos y proporcionar asesoramiento médico.

Diagrama de Casos de Uso

En este punto, se exponen los casos de uso que describen las diferentes interacciones que pueden realizar cada actor dentro de la aplicación. Se han organizado de tal manera que cada diagrama representa una funcionalidad en específico.

Gestión Usuarios

La Gestión de Usuarios se centra en las acciones que pueden realizar tanto los doctores como los pacientes a la hora de gestionar y administrar sus perfiles. En este módulo ambos actores pueden realizar las mismas acciones visibles en la figura A.1.

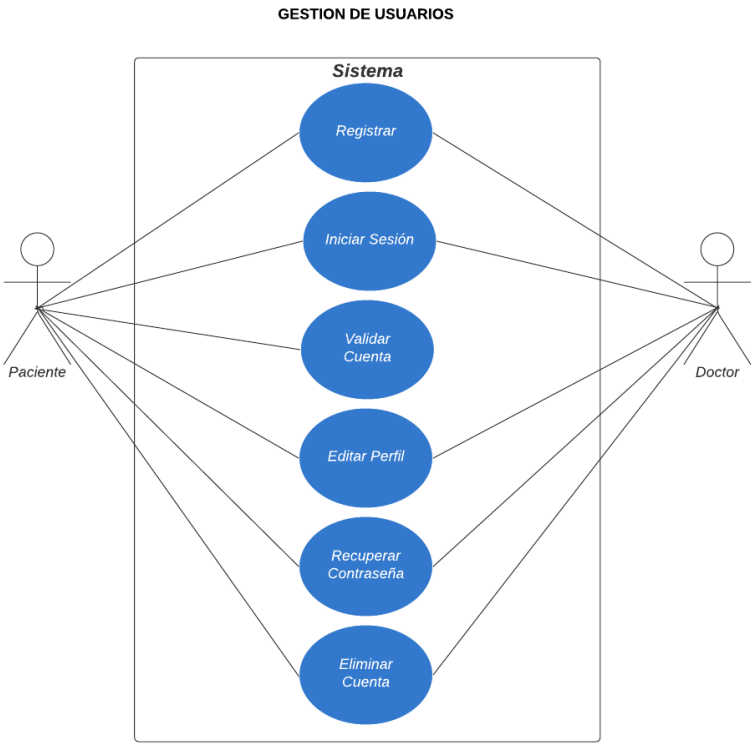

Figura Apéndice A-1- Diagrama Gestión de Usuarios

Gestión de Historial Médico

La Gestión de Historial Médico se basa en las acciones que tienen a su disposición los doctores para documentar y guardar un historial con datos médicos relevantes para el doctor. Desde la página web el doctor necesita estar vinculado a un paciente para poder crear un historial médico de dicho paciente. Adicionalmente el doctor podrá añadir detalles a dicho historial a lo largo del tiempo. Estas interacciones se ilustran en la figura A.2.

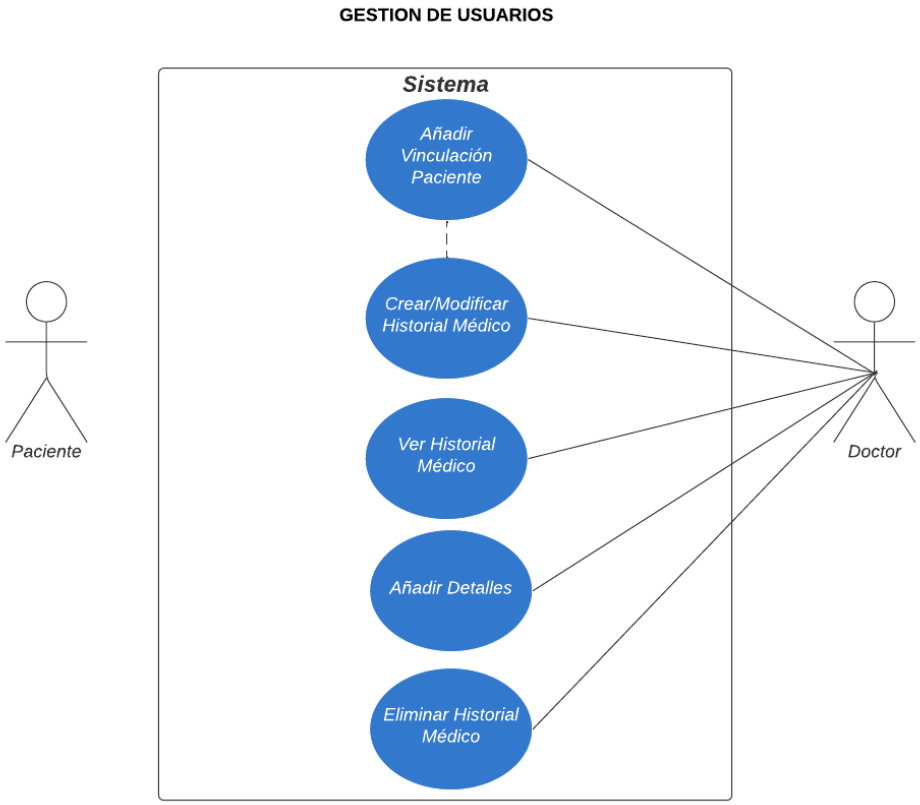

Figura Apéndice A-2- Diagrama de Gestión de Historial Médico

Gestión de Citas

La Gestión de Citas se enfoca en las acciones que pueden realizar tanto los doctores como los pacientes para programar y gestionar sus citas médicas. Dentro de la aplicación el usuario necesita estar vinculado a un doctor para poder solicitar una cita, y a su vez, el doctor necesita estar asociado a un paciente para poder crearle una cita. Estas interacciones se ilustran en la figura A.3

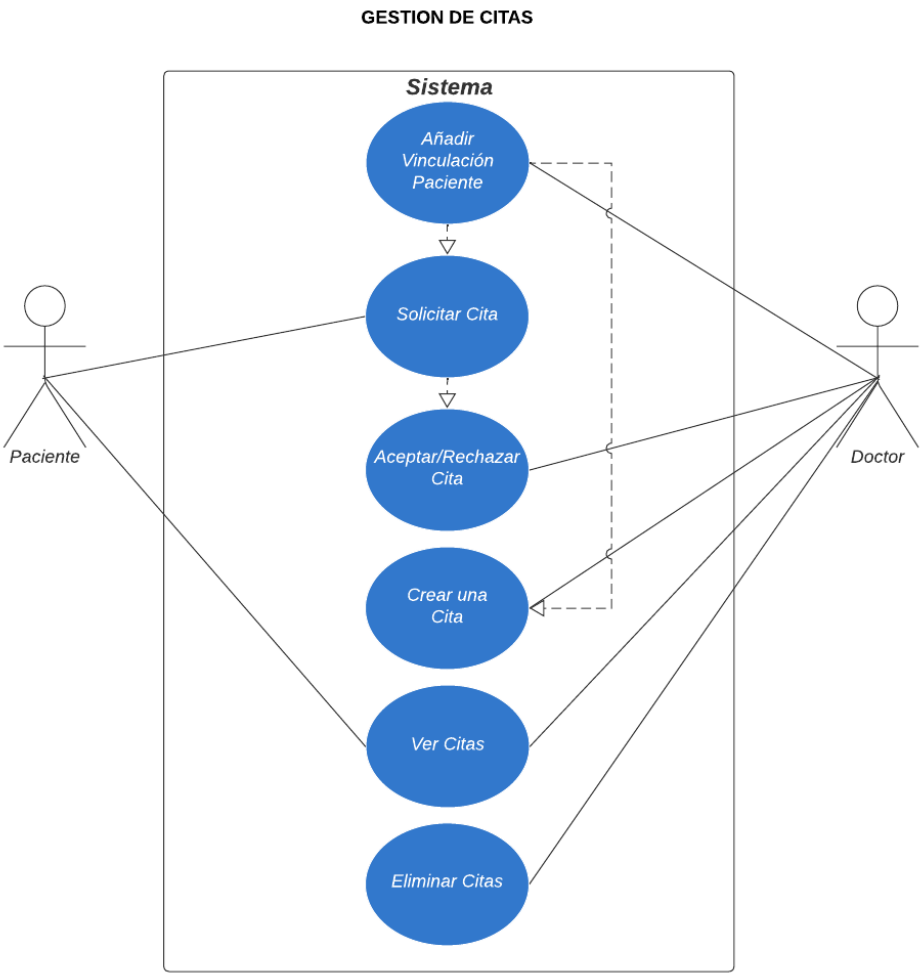

Figura Apéndice A-3- Diagrama de Gestión de Citas

Gestión de Tratamientos

La Gestión de Tratamientos pone énfasis en las acciones que pueden realizar los doctores para programar un tratamiento a un paciente y la gestión de ese tratamiento para configurar alarmas para las tomas de medicamentos a los pacientes. Dentro de la aplicación el doctor necesita estar asociado a un paciente para poder añadirle un tratamiento. Estas interacciones se ilustran en la figura A.4

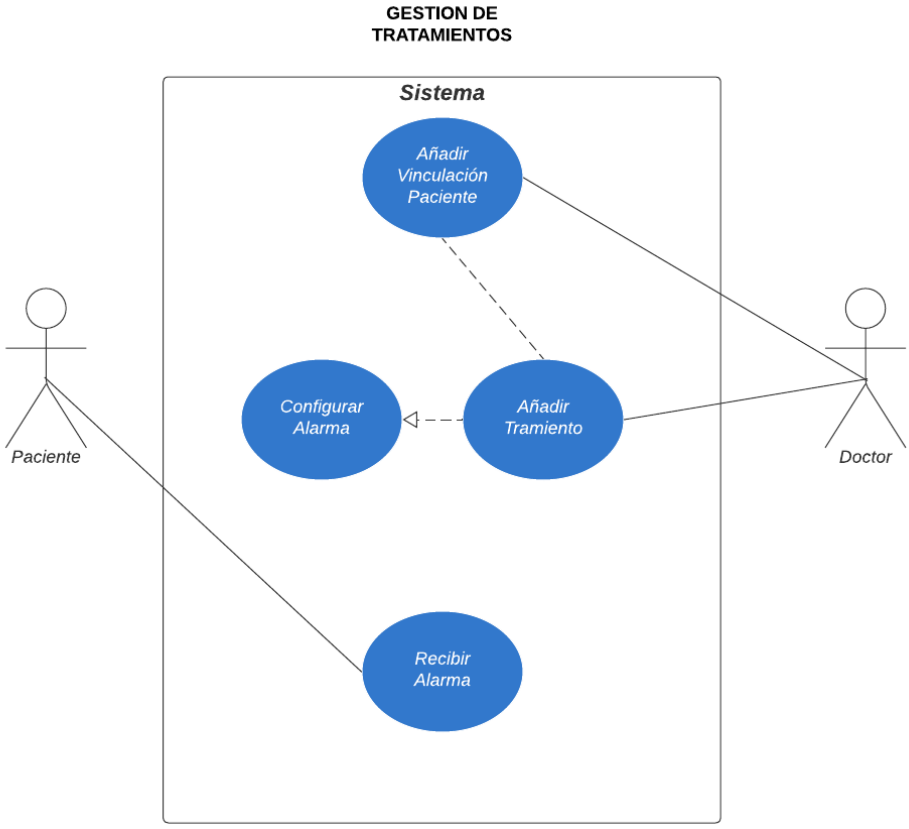

Figura Apéndice A-4- Diagrama de Gestión de Tratamientos

Gestión de Consultas

La Gestión de Consultas se focaliza en las acciones que pueden realizar tanto los doctores como los pacientes para poder recibir y enviar mensajes sobre una consulta o tema. Dentro de la aplicación el paciente necesita estar vinculado a un doctor para poder crear una consulta. Estas interacciones se ilustran en la figura A.5

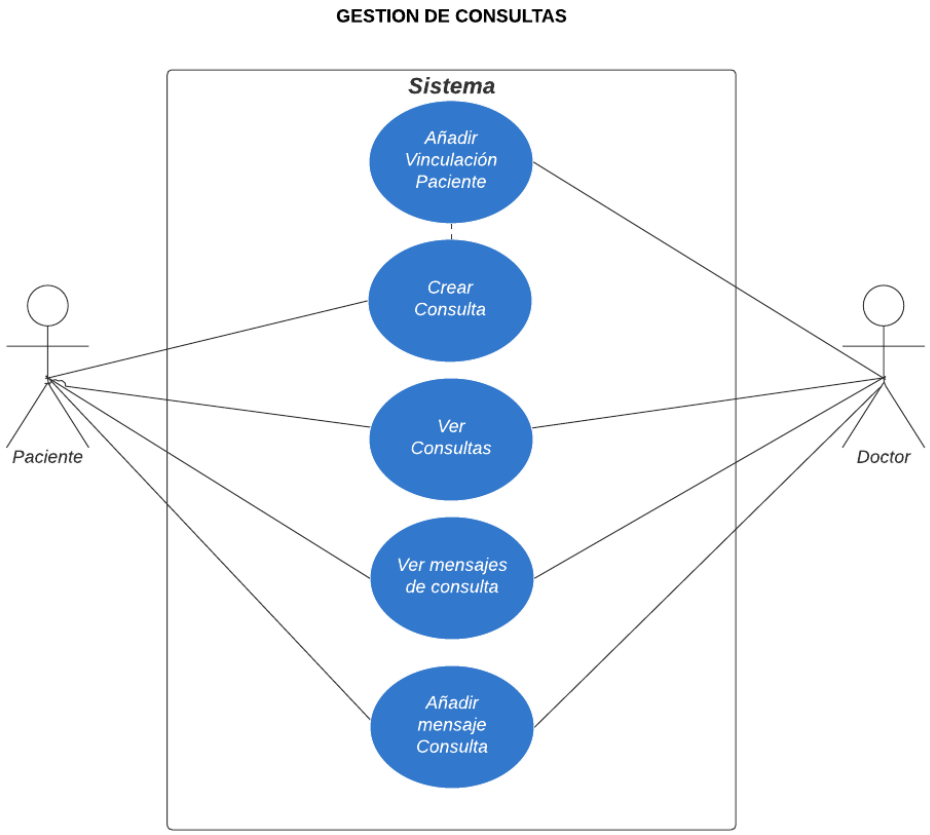

Figura Apéndice A-5- Diagrama Gestión de Consultas

Especificación de requisitos

En este punto, se exponen las diferentes especificaciones de requisitos que describen las diferentes interacciones que pueden realizar cada actor dentro de la aplicación.

Gestión Usuarios

La Gestión de Usuarios se centra en las acciones que pueden realizar tanto los doctores como los pacientes a la hora de gestionar y administrar sus perfiles. En este módulo se distinguen una serie de requisitos diferentes dependiendo de la plataforma donde se implementan, en web (doctores) y en móvil (pacientes)

Registro

La Tabla 5 muestra los requisitos para el registro de un doctor y la Tabla 6 para los de un paciente.

Requisito	Registro Doctor	
Identificador	1.1.1	
Prioridad	Alta	
Precondición	El usuario debe acceder al registro desde la plataforma web.	
Descripción	Los usuarios no registrados pueden registrarse, solo una cuenta por DNI.	
Entrada	DNI, nombre, apellidos, contraseña, repetir contraseña, correo electrónico, domicilio,código Postal, número de Teléfono, fecha de Nacimiento.	
Salida	Mensaje de error o redirige a Inicio de Sesión	
Secuencia normal	Paso	Acción
	1	El usuario accede a la página web y pulsa en el enlace: "Crear una cuenta".
	2	El sistema muestra una pantalla en la que le pide al usuario todos los campos necesarios para crear un usuario.
	3	El usuario rellena todos los campos y hace clic en registrar usuario.
	4	El sistema valida la información y la guarda.
	5	Muestra de nuevo la pantalla principal, o un error si no pasa las validaciones.
PostCondición	Se ha creado la cuenta de usuario tipo doctor.	
Excepciones	Paso	Acción
	3	Se muestra el mensaje correspondiente si las contraseñas no coinciden.

	3	Se muestra el mensaje correspondiente si el correo electrónico no cumple con los requisitos.
	3	Se muestra un mensaje de error si hay algún campo vacío.
	4	Se muestra el mensaje correspondiente si el correo electrónico o el ya hay una cuenta registrada con ese dni
	4	Se muestra un mensaje de error si no supera las validaciones (Formato DNI,fecha de nacimiento inferior a la fecha actual, número de teléfono).
Comentarios	NA.	
Actores	Usuario no registrado	

Tabla 5- Tabla de requisitos registrar doctor

Requisito	Registro Paciente	
Identificador	2.1.1	
Prioridad	Alta	
Precondición	El usuario debe acceder al registro desde la plataforma móvil.	
Descripción	Los usuarios no registrados pueden registrarse, solo una cuenta por DNI.	
Entrada	DNI, nombre, apellidos, contraseña, repetir contraseña, correo electrónico, domicilio,código Postal, número de Teléfono, fecha de Nacimiento.	
Salida	Mensaje de error o redirige a Inicio de Sesión	
Secuencia normal	Paso	Acción
	1	El usuario inicia el dispositivo móvil y pulsa en el enlace: "Crear una cuenta".
	2	El sistema muestra una pantalla en la que le pide al usuario todos los campos necesarios para crear un usuario.
	3	El usuario rellena todos los campos y hace clic en registrar

		usuario.
	4	El sistema valida la información y la guarda.
	5	Muestra de nuevo la pantalla de inicio de sesión, o un error si no pasa las validaciones.
PostCondición	Se ha creado la cuenta de usuario tipo paciente.	
Excepciones	Paso	Acción
	3	Se muestra el mensaje Toast correspondiente si las contraseñas no coinciden.
	3	Se muestra el mensaje Toast correspondiente si el correo electrónico no cumple con los requisitos.
	4	Se muestra el mensaje Toast correspondiente si el correo electrónico o el ya hay una cuenta registrada con ese dni
	4	Se muestra un mensaje Toast si hay algún campo vacío.
	4	Se muestra un mensaje TOAST si no supera las validaciones (Formato DNI,fecha de nacimiento inferior a la fecha actual, número de teléfono).
Comentarios	NA.	
Actores	Usuario no registrado	

Tabla 6- Tabla de requisitos registrar paciente

Iniciar Sesión

La Tabla 7 muestra los requisitos para el inicio de sesión de un doctor y la Tabla 8 para los de un paciente.

Requisito	Inicio sesión doctor
Identificador	1.1.2
Prioridad	Alta

Precondición	El usuario debe estar en la página de inicio de Sesión en l a plataforma web.	
Descripción	Los usuarios registrados deben poder iniciar sesión en la plataforma con su DNI y contraseña.	
Entrada	DNI, contraseña.	
Salida	Mensaje de error o llamada a la pantalla principal.	
Secuencia normal	Paso	Acción
	1	El usuario accede a la página de inicio de sesión.
	2	El usuario ingresa sus credenciales.
	3	El sistema verifica las credenciales y guarda la sesión.
PostCondición	El usuario ha iniciado sesión	
Excepciones	Paso	Acción
	3	Se muestra el mensaje correspondiente si las credenciales son incorrectas
Comentarios	NA	
Actores	Usuario iniciado como doctor	

Tabla 7- Tabla de requisitos inicio de sesión doctor

Requisito	Inicio sesión paciente
Identificador	2.1.2
Prioridad	Alta
Precondición	El usuario debe estar en la pantalla de inicio de sesión en el dispositivo móvil

Descripción	Los usuarios registrados deben poder iniciar sesión en la plataforma con su DNI y contraseña.	
Entrada	DNI, contraseña.	
Salida	Mensaje TOAST o cambio a validar cuenta.	
Secuencia normal	Paso	Acción
	1	El usuario accede a la pantalla de inicio de sesión.
	2	El usuario ingresa sus credenciales.
	3	El sistema verifica las credenciales y guarda la sesión en el dispositivo móvil.
PostCondición	El usuario ha iniciado sesión y no hará falta que vuelva a iniciar sesión hasta que se cierre.	
Excepciones	Paso	Acción
	3	Se muestra el mensaje TOAST correspondiente si las credenciales son incorrectas
Comentarios	NA	
Actores	Usuario iniciado como paciente	

Tabla 8- Tabla de requisitos inicio de sesión paciente

Validar Cuenta

Esta Tabla 9 muestra los requisitos para validar la cuenta de un paciente.

Requisito	Validar cuenta Paciente
Identificador	2.1.3

Prioridad	Baja	
Precondición	El usuario debe pulsar el botón de 'Iniciar sesión' en la pantalla de inicio de sesión con las credenciales del paciente correctas.	
Descripción	Los usuarios registrados tras iniciar sesión deben validar su cuenta con un mensaje enviado a su correo electrónico.	
Entrada	Código de validación enviado al correo.	
Salida	Mensaje de codigo erróneo o llamada a la pantalla principal.	
Secuencia normal	Paso	Acción
	1	Llegada a la pantalla de validar, mirar en el correo el código de validación.
	2	Si no se ha recibido el código, pulsar 'Reenviar correo'
	3	Una vez introducido el código,pulsar en 'Comprobar'.
PostCondición	El usuario ha validado su cuenta y accede a la página principal.	
Excepciones	Paso	Acción
	3	Se muestra el mensaje correspondiente si las credenciales son incorrectas
Comentarios	NA	
Actores	Usuario registrado como Paciente	

Tabla 9- Tabla de requisitos validar cuenta paciente

Editar Perfil

La Tabla 10 muestra los requisitos para editar el perfil de un doctor y la Tabla 11 para los de un paciente.

Requisito	Editar perfil doctor
Identificador	1.1.4

Prioridad	Media	
Precondición	El doctor debe iniciar sesión y acceder a la opción de editar perfil desde el menú desplegable en el nombre de usuario en la plataforma web.	
Descripción	Los doctores pueden editar sus datos personales.	
Entrada	Nombre, apellidos, domicilio,código postal, número de teléfono, fecha de nacimiento.	
Salida	Mensaje de error o redirige a la pantalla principal	
Secuencia normal	Paso	Acción
	1	El doctor accede a la opción de editar perfil desde el menú desplegable en el nombre de usuario en la plataforma web.
	2	El sistema muestra una pantalla en la que le muestra al doctor todos los campos rellenados por los datos actuales del usuario.
	3	El doctor modifica los campos deseados y hace clic en guardar cambios.
	4	El sistema valida la información y la guarda.
	5	Muestra de nuevo la pantalla principal, o un error si no pasa las validaciones.
PostCondición	Se han guardado los datos correspondientes del doctor.	
Excepciones	Paso	Acción
	3	Se muestra un mensaje de error si hay algún campo vacío.
	4	Se muestra el mensaje correspondiente si no se supera las validaciones.
Comentarios	NA.	

Actores	Doctor

Tabla 10- Tabla de requisitos editar perfil doctor

Requisito	Editar perfil paciente	
Identificador	2.1.4	
Prioridad	Media	
Precondición	El paciente debe iniciar sesión y acceder a la opción de editar perfil desde el menú desplegable en el *navbar* en la aplicación móvil.	
Descripción	Los pacientes pueden editar sus datos personales.	
Entrada	Nombre, apellidos, domicilio,código postal, número de teléfono, fecha de nacimiento.	
Salida	Mensaje Toast o redirige a la pantalla principal	
Secuencia normal	Paso	Acción
	1	El paciente accede a la opción de editar perfil desde el menú desplegable en el *navbar* en la aplicación móvil.
	2	El sistema muestra una pantalla en la que le muestra al paciente todos los campos rellenados por los datos actuales del usuario.
	3	El paciente modifica los campos deseados y hace clic en guardar información.
	4	El sistema valida la información y la guarda.
	5	Muestra de nuevo la pantalla principal, o un error si no pasa las validaciones.
PostCondición	Se han guardado los datos correspondientes del paciente.	

Excepciones	Paso	Acción
	4	Se muestra un mensaje Toast si hay algún campo vacío.
	4	Se muestra el mensaje Toast correspondiente si no se supera las validaciones.
Comentarios	NA.	
Actores	Paciente	

Tabla 11- Tabla de requisitos editar perfil paciente

Recuperar contraseña

La Tabla 12 muestra los requisitos para recuperar la contraseña de un paciente.

Requisito	recuperar contraseña paciente	
Identificador	2.1.5	
Prioridad	Baja	
Precondición	El usuario debe estar registrado	
Descripción	Si el paciente ha olvidado su contraseña, puede modificarla tras introducir un código que sería enviado a su correo.	
Entrada	Nueva contraseña, Código de validación	
Salida	Mensaje de código erróneo o llamada a la pantalla principal.	
Secuencia normal	Paso	Acción

	1	Llegada a la pantalla de recuperar contraseña, se debe introducir la nueva contraseña y el código enviado al correo.
	2	Si el código no se ha recibido, se deberá pulsar en 'Reenviar correo' para que se vuelva a enviar.
	3	Una vez introducido el código y la contraseña, se pulsa en 'Cambiar contraseña'.
PostCondición	El usuario ha cambiado su contraseña y es redirigido a la pantalla de inicio de sesión.	
Excepciones	Paso	Acción
	1	Se muestra el mensaje correspondiente si las credenciales son incorrectas
Comentarios	NA	
Actores	Usuario registrado como paciente	

Tabla 12- Tabla de requisitos validar cuenta paciente

Eliminar cuenta

La Tabla 13 muestra los requisitos para eliminar la cuenta de un doctor y la tabla 14 para los de un paciente.

Requisito	Eliminar doctor	
Identificador	1.1.6	
Prioridad	Baja	
Precondición	El doctor debe estar en la página web y logedo.	
Descripción	Los doctores registrados deben poder eliminar su cuenta en la web.	
Entrada	NA	
Salida	Redirigir a iniciar sesión.	
Secuencia normal	Paso	Acción

	1	El usuario accede a la página web logueada, hace clic en las opciones de gestión de usuario y le da a eliminar cuenta.
	2	Acepta el modal de eliminar cuenta
	3	El sistema redirige a la pantalla de inicio de sesión.
PostCondición	El doctor ha eliminado su cuenta	
Excepciones	Paso	Acción
	2	Si le da a rechazar el modal se mantiene el estado actual en la web.
Comentarios	NA	
Actores	Usuario iniciado como doctor	

Tabla 13- Tabla de requisitos eliminar cuenta doctor.

Requisito	Eliminar cuenta paciente	
Identificador	2.1.6	
Prioridad	Baja	
Precondición	El paciente debe estar en la aplicación móvil y logedo.	
Descripción	Los pacientes registrados deben poder eliminar su cuenta en la web.	
Entrada	NA	
Salida	Redirigir a iniciar sesión.	
Secuencia normal	Paso	Acción
	1	El paciente accede al móvil logueado, hace clic en el menú de opciones de arriba a la derecha y le da a eliminar cuenta.

	2	Acepta el *Dialog* de confirmación.
	3	El sistema redirige a la pantalla de inicio de sesión.
PostCondición	El paciente ha eliminado su cuenta	
Excepciones	Paso	Acción
	2	Si se da a rechazar el *Dialog* se mantiene el estado actual de la aplicación.
Comentarios	NA	
Actores	Usuario iniciado como paciente	

Tabla 14- Tabla de requisitos eliminar cuenta paciente

Gestión asignaciones

La Gestión de Asignaciones se encarga de la vinculación y desvinculación de un paciente con un doctor. Esta vinculación es realizada por el doctor. Si se desea realizar una funcionalidad sobre un paciente debe existir una vinculación.

Vincular médico paciente

La Tabla 15 muestra los requisitos para la vinculación de un doctor y un paciente.

Requisito	Vincular médico paciente
Identificador	1.2.1
Prioridad	Alta
Precondición	El doctor debe acceder a la página de gestión de usuarios.
Descripción	El doctor debe poder añadir un paciente a su lista.

Entrada	DNI o correo	
Salida	Mensaje de éxito o error.	
Secuencia normal	Paso	Acción
	1	El usuario accede a gestión de pacientes y hace clic en "Añadir Usuario".
	2	El sistema muestra un modal en el que se le pide al doctor que introduzca un DNI o un correo.
	3	El sistema valida la información y agrega el paciente a la lista.
PostCondición	Se ha agregado el paciente a la lista de usuarios.	
Excepciones	Paso	Acción
	3	Se muestra el mensaje correspondiente si no se ha encontrado el usuario con ese correo o DNI.
	3	Se muestra el mensaje correspondiente si el usuario ya está en la lista.
Comentarios	NA.	
Actores	Doctor	

Tabla 15- Tabla de requisitos para vincular médico paciente.

Desvincular médico paciente

La Tabla 16 muestra los requisitos para la desvinculación de un doctor y un paciente.

Requisito	Desvincular médico paciente
Identificador	1.2.2
Prioridad	Alta
Precondición	El doctor debe acceder a la página de gestión de usuarios.

Descripción	El doctor debe poder eliminar un paciente de su lista.	
Entrada	NA.	
Salida	Mensaje operación correcta	
Secuencia normal	Paso	Acción
	1	El usuario accede a gestión de pacientes, busca el paciente que desea y hace clic en "Eliminar".
	2	El sistema recibe la información y refresca la vista dando un mensaje de operación exitosa.
PostCondición	Se ha eliminado el paciente de la lista de usuarios.	
Excepciones	Paso	Acción
	2	Si se produce un error interno, se muestra un mensaje de error.
Comentarios	NA.	
Actores	Doctor	

Tabla 16- Tabla de requisitos para desvincular médico paciente.

Gestión historial médico

La Gestión de Historial Médico se basa en las acciones que tienen a su disposición los doctores para documentar y guardar un historial con datos médicos relevantes del paciente para determinar un mejor diagnóstico. Desde la página web el doctor necesita estar vinculado a un paciente para poder crear un historial médico de dicho paciente. Adicionalmente el doctor podrá añadir detalles a dicho historial a lo largo del tiempo.

Crear/Modificar historial médico

La Tabla 17 muestra los requisitos para la creación de un historial médico de un doctor.

Requisito	Crear/modificar historial médico	
Identificador	1.3.1	
Prioridad	Media	
Precondición	El doctor debe estar registrado y debe tener asignado un paciente	
Descripción	El doctor crea o modifica un historial médico a un paciente.	
Entrada	nombre, apellidos, sexo, fecha de nacimiento, peso, altura, marcar las casillas que corresponden a las enfermedades que padece el paciente si tuviera alguna, escribir alergias si tuviera, escribir notas si fueran necesarias y comentar detalles si fueran necesario	
Salida	Mensaje de error o redirige a la pantalla de gestión de usuarios del doctor	
Secuencia normal	Paso	Acción
	1	El usuario accede a la página web y en el menú pulsa el apartado 'Gestión Usuarios'
	2	El doctor selecciona el botón 'Funciones Paciente' del paciente que desee.
	3	En dicho menú, selecciona 'Crear historial médico'.
	4	A continuación se procede a rellenar todos los datos relevantes del paciente y se puede crear una nueva enfermedad en el caso de que la que padezca no se encuentre entre las opciones predeterminadas.
	5	Cuando esté rellenado el formulario, el doctor debe pulsar 'Generar PDF' y el doctor será redirigido al menú de gestión de usuarios
PostCondición	Se ha generado un PDF con la información del paciente en él.	
Excepciones	Paso	Acción
	4	Se muestra el mensaje correspondiente si algún campo está vacío

Comentarios	NA.
Actores	Doctor registrado

Tabla 17- Tabla de requisitos para crear/modificar historial médico

Ver historial médico

La Tabla 18 muestra los requisitos para que un doctor pueda visualizar un historial médico de un paciente.

Requisito	Ver historial médico	
Identificador	1.3.2	
Prioridad	Media	
Precondición	El doctor debe estar registrado y debe tener asignado un paciente	
Descripción	El doctor visualiza el historial médico de un paciente.	
Entrada	NA.	
Salida	Se muestra un pdf con el historial médico del paciente.	
Secuencia normal	Paso	Acción
	1	El usuario accede a la página web y en el menú pulsa el apartado 'Gestión Usuarios'
	2	El doctor selecciona el botón 'Funciones Paciente' del paciente que desee.
	3	En dicho menú, seleccione 'Ver historial médico'.
	4	A continuación se descarga y se muestra un pdf que contiene la información correspondiente al historial médico de dicho paciente.
PostCondición	Se muestra un PDF con la información del paciente en él.	

Excepciones	Paso	Acción
	4	Si se produce un error interno, se muestra un mensaje de error.
Comentarios	NA.	
Actores	Doctor registrado	

Tabla 18- Tabla de requisitos para ver historial médico

Añadir detalles al historial médico

La Tabla 19 muestra los requisitos para que un doctor pueda añadir detalles al historial médico de un paciente.

Requisito	Añadir detalles a un historial médico	
Identificador	1.3.3	
Prioridad	Media	
Precondición	El doctor debe estar registrado, debe tener asignado un paciente y haber creado un historial médico a dicho paciente.	
Descripción	El doctor puede añadir detalles al historial médico de un paciente.	
Entrada	Detalle que se desee añadir al historial médico.	
Salida	Se te redirigirá a la pantalla de gestión de usuarios al tener éxito.	
Secuencia normal	Paso	Acción
	1	El usuario accede a la página web y en el menú pulsa el apartado 'Gestión Usuarios'
	2	El doctor selecciona el botón 'Funciones Paciente' del paciente que desee.
	3	En dicho menú, selecciona 'Añadir detalles al historial médico'.

	4	A continuación, le das al botón 'guardar' para que se guarde dicho comentario en el historial médico, o a 'cancelar' si no deseas que se guarde.
PostCondición	Se añade un detalle al historial médico de dicho paciente.	
Excepciones	Paso	Acción
	4	Si se produce un error interno, se muestra un mensaje de error.
Comentarios	NA.	
Actores	Doctor registrado	

Tabla 19- Tabla de requisitos para añadir detalles al historial médico

Eliminar historial médico

La Tabla 20 muestra los requisitos para que un doctor pueda eliminar el historial médico de un paciente.

Requisito	Eliminar historial médico	
Identificador	1.3.4	
Prioridad	Media	
Precondición	El doctor debe estar registrado, debe tener asignado un paciente y haber creado un historial médico a dicho paciente.	
Descripción	El doctor puede eliminar el historial médico de un paciente	
Entrada	NA.	
Salida	Mensaje de historial médico eliminado exitosamente.	
Secuencia normal	Paso	Acción

	1	El usuario accede a la página web y en el menú pulsa el apartado 'Gestión Usuarios'
	2	El doctor selecciona el botón 'Funciones Paciente' del paciente que desee.
	3	En dicho menú, selecciona 'Eliminar historial médico'.
PostCondición	Se elimina el historial médico de un paciente.	
Excepciones	Paso	Acción
	3	Si se produce un error interno, se muestra mensaje de error.
Comentarios	NA.	
Actores	Doctor registrado	

Tabla 20- Tabla de requisitos para eliminar el historial médico de un paciente

Gestión de Citas

La Gestión de Citas se focaliza en las acciones que pueden realizar tanto los doctores como los pacientes a la hora de gestionar y administrar sus citas médicas.Se necesita una vinculación existente entre paciente y doctor para poder agregar una cita a un paciente con ese doctor.

Solicitar cita

La Tabla 21 muestra los requisitos para que un paciente pueda solicitar una cita.

Requisito	Solicitar cita
Identificador	2.4.1
Prioridad	Media

Precondición	El usuario debe acceder al *fragment* de solicitar una cita en el dispositivo móvil encontrado en el menú inferior a la derecha.	
Descripción	Los pacientes pueden solicitar una cita a un doctor, si al doctor le parece correcta puede ser aceptada.	
Entrada	Doctor, Día de la cita, Hora, Asunto.	
Salida	Mensaje Toast de error o redirige a la pantalla principal	
Secuencia normal	Paso	Acción
	1	El usuario accede al *fragment* de solicitar una cita en el dispositivo móvil encontrado en el menú inferior a la derecha.
	2	El sistema muestra una pantalla en la que le pide al usuario todos los campos necesarios para solicitar una cita.
	3	El usuario rellena todos los campos y hace clic en Solicitar cita.
	4	El sistema valida la información y la guarda.
	5	Muestra de nuevo la pantalla principal, o un mensaje Toast si no pasa las validaciones.
PostCondición	Se ha solicitado una cita para un doctor determinado.	
Excepciones	Paso	Acción
	3	Se muestra un mensaje Toast si la cita no es posterior a la actual
	4	Se muestra un mensaje Toast de error si hay algún campo vacío.
	4	Se muestra un mensaje Toast si hay un error interno en el servidor
Comentarios	NA.	
Actores	Paciente	

Tabla 21- Tabla de requisitos solicitar Cita

Aceptar/Rechazar cita

La Tabla 22 muestra los requisitos para que un doctor pueda aceptar o rechazar una cita solicitada por un paciente.

Requisito	Aceptar/Rechazar cita	
Identificador	2.4.2	
Prioridad	Media	
Precondición	El doctor debe acceder a la pantalla de citas desde el botón del navbar superior.	
Descripción	Los doctores pueden aceptar o rechazar una cita solicitada por un paciente.	
Entrada	Duración	
Salida	Modal de éxito o error	
Secuencia normal	Paso	Acción
	1	El doctor accede a la pantalla de citas desde el botón del navbar superior.
	2	El sistema muestra un botón para aceptar o rechazar cita, si se acepta se rellena el campo duración en minutos
	3	El sistema valida la información y agrega o elimina la solicitud.
	4	Muestra de nuevo la misma pantalla con un modal de éxito o error en la operación.
PostCondición	Se ha solicitado una cita para un doctor determinado.	
Excepciones	Paso	Acción
	3	En el caso de aceptar, se muestra un modal de error si ya existen citas coincidentes en la base de datos.
Comentarios	NA.	
Actores	Doctor	

Tabla 22- Tabla de requisitos aceptar o rechazar Cita

Crear cita

La Tabla 23 muestra los requisitos para que un doctor pueda concertar una cita con un paciente.

Requisito	Crear cita	
Identificador	1.4.3	
Prioridad	Alta	
Precondición	El doctor debe acceder a la pantalla de funciones de usuario dentro de gestión de usuarios y hacer clic en crear cita,	
Descripción	Un doctor puede crear una cita para un paciente.	
Entrada	día de la cita, hora, asunto y duración.	
Salida	Mensaje de error o éxito.	
Secuencia normal	Paso	Acción
	1	El doctor debe acceder a la pantalla de funciones de usuario dentro de gestión de usuarios y hacer clic en crear cita,
	2	El sistema muestra un modal en el que le pide al doctor todos los campos necesarios para crear una cita.
	3	El usuario rellena todos los campos y hace clic en crear cita.
	4	El sistema valida la información y la guarda.
	5	Muestra un mensaje de éxito o de error.
PostCondición	Se ha creado una cita para un paciente determinado.	
Excepciones	Paso	Acción
	3	Se muestra una advertencia de error si hay algún campo vacío.

	4	Se muestra un mensaje si la fecha de la cita no es posterior a la actual
	4	Se muestra un mensaje de error si hay alguna cita existente en ese periodo de tiempo.
Comentarios	NA.	
Actores	Doctor	

Tabla 23- Tabla de requisitos crear Cita

Ver citas

La Tabla 24 muestra los requisitos para mostrar las citas a un un doctor y la Tabla 25 para las de un paciente.

Requisito	Ver citas	
Identificador	1.4.4	
Prioridad	Alta	
Precondición	El doctor debe acceder a la pantalla de calendario o al apartado de citas.	
Descripción	Un doctor puede ver las citas que tiene al día, semana o mes.	
Entrada	NA	
Salida	Mensaje de error o éxito.	
Secuencia normal (Calendario)	Paso	Acción
	1	El doctor debe acceder a la pantalla de calendario
	2	El sistema les mostrará un calendario con vistas diarias, semanales y mensuales
Secuencia normal (Citas)	Paso	Acción

	1	El doctor debe acceder a la pantalla de citas
	2	El sistema le mostrará una lista con las citas que tiene.
Comentarios	NA.	
Actores	Doctor	

Tabla 24- Tabla de requisitos ver citas doctor

Requisito	Ver citas	
Identificador	2.4.4	
Prioridad	Alta	
Precondición	El paciente debe acceder al *fragment* de calendario en el dispositivo móvil haciendo clic en el botón del medio del *navbar* inferior.	
Descripción	Un paciente puede ver las citas que tiene con diferentes doctores.	
Entrada	NA	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El paciente debe acceder al *fragment* de calendario en el dispositivo móvil haciendo clic en el botón del medio del *navbar* inferior
	2	El sistema les mostrará un calendario y una lista de citas.
Comentarios	NA.	
Actores	Doctor	

Tabla 25- Tabla de requisitos ver citas paciente

Eliminar cita

La Tabla 26 muestra los requisitos para que un doctor pueda eliminar una cita existente.

Requisito	Eliminar cita	
Identificador	2.4.4	
Prioridad	Alta	
Precondición	El doctor debe acceder a la pantalla del calendario y hacer clic en una cita.	
Descripción	Un doctor puede eliminar una cita ya existente.	
Entrada	Cita seleccionada	
Salida	Refresca el calendario con la cita eliminada	
Secuencia normal	Paso	Acción
	1	El doctor accede a la pantalla del calendario.
	2	El doctor hace clic en una cita determinada.
	3	El sistema muestra un modal con la información de la cita y un botón para eliminar.
	4	El doctor selecciona eliminar
	5	El sistema obtiene el identificador de la cita y procede a eliminarla
PostCondición	Se ha eliminado una cita.	
Excepciones	Paso	Acción
	3	En el caso de no poderse eliminar la cita, se mantiene el estado actual del calendario.

Comentarios	NA.
Actores	Doctor

Tabla 26- Tabla de requisitos eliminar cita

Gestión de tratamientos

La Gestión de Tratamientos pone énfasis en las acciones que pueden realizar los doctores para programar un tratamiento a un paciente y la posterior configuración de las alarmas para las tomas de medicamentos a los pacientes. Dentro de la aplicación el doctor necesita estar asociado a un paciente para poder añadirle un tratamiento.

Añadir tratamiento

La Tabla 27 muestra los requisitos para la creación de tratamiento para un paciente.

Requisito	Añadir tratamiento	
Identificador	1.5.1	
Prioridad	Alta	
Precondición	El doctor debe estar registrado y estar vinculado a un paciente. El doctor debe acceder a la pantalla de funciones de usuario dentro de gestión de usuarios y hacer clic en iniciar tratamiento,	
Descripción	El doctor puede crea un tratamiento a un paciente, lo que conlleva que se le configuren las alarmas al paciente.	
Entrada	Descripción del tratamiento, se puede añadir los medicamentos que sean necesarios, y cada medicamento contiene lo siguiente: Nombre del medicamento, dosis, hora de la 1ra toma, número de tomas al día, fecha de inicio y fecha de fin de tratamiento.	
Salida	Mensaje de error o mensaje de confirmación de alarma creada.	
Secuencia normal	Paso	Acción

	1	El doctor accede a la pantalla de gestión de usuarios.
	2	El doctor pulsa el botón 'funciones paciente' del paciente al que desee crear el tratamiento.
	3	En la siguiente pantalla, el doctor pulsa el botón 'iniciar tratamiento'.
	4	El usuario rellena todos los campos y hace clic en guardar tratamiento.
	6	Se muestra mensaje de tratamiento creado con éxito.
PostCondición	El doctor ha creado un tratamiento para un pacienter.	
Excepciones	Paso	Acción
	5	Se muestra un mensaje de error si la fecha es anterior a la actual, también salta si la fecha de fin es anterior a la fecha de inicio
Comentarios	NA.	
Actores	Doctor registrado	

Tabla 27- Tabla de requisitos creación de un tratamiento.

Configurar alarma

La Tabla 28 muestra los requisitos para la configuración de las alarmas de un paciente.

Requisito	Configurar alarma
Identificador	1.5.2
Prioridad	Alta
Precondición	El doctor debe estar registrado y estar vinculado a un paciente. El doctor debe haber creado un tratamiento.
Descripción	En el momento que el botón inicia el tratamiento, se configuran las alarmas adecuadamente para que el paciente las reciba en el momento adecuado.

Entrada	NA.	
Salida	NA.	
Secuencia normal	Paso	Acción
	1	El doctor inicia el tratamiento.
	2	El tratamiento se guarda en la base de datos y se crean las alarmas en función de los medicamentos asignados.
	3	Cuando se crea una alarma, se crean todas las alarmas correspondientes a las horas y los días próximos.
	4	Cuando el paciente se loguea, si el doctor ha iniciado un tratamiento para él, se configurarán todas las alarmas para dicho paciente.
PostCondición	Se configuran las alarmas necesarias para completar el tratamiento iniciado por el doctor.	
Excepciones	Paso	Acción
	5	Si la alarma es posterior a la fecha actual, no sonará.
Comentarios	NA.	
Actores	NA.	

Tabla 28- Tabla de requisitos paral a configuración de una alarma.

Recibir alarma

La Tabla 29 muestra los requisitos para la recepción de las alarmas de un paciente.

Requisito	Recibir alarma
Identificador	2.5.3
Prioridad	Alta

Precondición	El doctor debe estar registrado y estar vinculado a un paciente. El doctor debe haber creado un tratamiento y el paciente debe de haberse logueado.	
Descripción	En el momento que el paciente inicia sesión en la app, se configuran las alarmas.	
Entrada	NA.	
Salida	NA.	
Secuencia normal	Paso	Acción
	1	El paciente debe loguearse.
	2	Una vez se configuran las alarmas y llegue la hora correspondiente, llegará una notificación con el nombre del medicamento y la dosis que el doctor ha preestablecido para dicho paciente.
	3	La notificación no parará de sonar hasta que el paciente interactúe con ella.
PostCondición	El paciente recibe una notificación con el nombre del medicamente y la dosis necesaria que deberá tomarse dicho paciente.	
Excepciones	Paso	Acción
	5	Si la alarma es posterior a la fecha actual, no sonará.
Comentarios	NA.	
Actores	NA.	

Tabla 29- Tabla de requisitos para la recepción de una alarma.

Gestión de consultas

La Gestión de Consultas hace énfasis en las acciones que pueden realizar tanto los doctores como los pacientes para poder recibir y enviar mensajes sobre una consulta o tema. Dentro de la aplicación el paciente necesita estar vinculado a un doctor para

poder crear una consulta. Estas interacciones se ilustran en la siguiente especificación de requisitos.

Crear consulta

La Tabla 30 muestra los requisitos para la creación de una consulta de un paciente.

Requisito	Crear consulta	
Identificador	2.6.1	
Prioridad	Alta	
Precondición	El paciente debe estar logueado en el dispositivo móvil y dirigirse a la pantalla de crear consultas haciendo clic en el botón de consultas del navbar inferior y presionando el botón de crear consulta en el dispositivo móvil.	
Descripción	Los pacientes pueden crear una consulta al doctor deseado.	
Entrada	Doctor, asunto o título	
Salida	Mensaje Toast de éxito o error	
Secuencia normal	Paso	Acción
	1	El paciente accede a la pantalla de consultas.
	2	El paciente hace clic en el botón de crear consulta
	3	El sistema muestra una pantalla en la que le pide al usuario los campos necesarios para crear un usuario.
	4	El usuario rellena todos los campos y hace clic en registrar usuario.
	5	El sistema valida la información y la guarda.
	6	Muestra de nuevo la pantalla principal, o un mensaje Toast de error si no pasa las validaciones.

PostCondición	Se ha creado una consulta para el paciente con un doctor.	
Excepciones	Paso	Acción
	5	Se muestra un mensaje de error si el título es un campo vacío.
Comentarios	NA.	
Actores	Paciente	

Tabla 30- Tabla de requisitos crear consulta

Ver consultas

La Tabla 31 muestra los requisitos para mostrar la lista de consultas de un doctor y la Tabla 32 las de un paciente.

Requisito	Ver consultas	
Identificador	1.6.2	
Prioridad	Alta	
Precondición	El doctor debe estar logueado en la plataforma web y dirigirse a la pantalla de consultas.	
Descripción	Si el doctor tiene alguna consulta con algún paciente se le mostrará una lista mostrando el título de cada una de ellas, el autor y la fecha	
Entrada	NA	
Salida	lista de consultas	
Secuencia normal	Paso	Acción
	1	El doctor accede a la pantalla de consultas.
	2	El sistema le muestra una lista con todas las consultas del doctor.

PostCondición	Se ha listado las consultas para un doctor	
Excepciones	Paso	Acción
	2	Si no se tienen consultas se mostrará un aviso con no existen consultas.
Comentarios	NA.	
Actores	Doctor	

Tabla 31- Tabla de requisitos ver consultas doctor

Requisito	Ver consultas	
Identificador	2.6.2	
Prioridad	Alta	
Precondición	El paciente debe estar logueado en el dispositivo móvil y dirigirse a la pantalla de consultas.	
Descripción	Si el paciente tiene alguna consulta con algún doctor se le mostrará una lista mostrando el título de cada una de ellas y una campanita si tiene algún mensaje sin leer.	
Entrada	NA	
Salida	lista de consultas	
Secuencia normal	Paso	Acción
	1	El paciente accede a la pantalla de consultas.
	2	El sistema le muestra una lista con todas las consultas del paciente.
	3	Si la consulta tiene mensajes sin leer aparecerá una campanita
PostCondición	Se ha listado las consultas para un paciente	

Excepciones	Paso	Acción
	2	Si no se tienen consultas se mostrará una lista vacía.
Comentarios	NA.	
Actores	Paciente	

Tabla 32- Tabla de requisitos ver consultas paciente

Ver mensajes consultas

La Tabla 33 muestra los requisitos para ver los mensajes de una consulta de un doctor y la Tabla 34 las de un paciente.

Requisito	Ver consultas	
Identificador	1.6.3	
Prioridad	Alta	
Precondición	El doctor debe estar logueado en la plataforma web y dirigirse a la pantalla de consultas, dentro de ella hacer clic en un elemento de la lista para mostrar el chat.	
Descripción	El doctor podrá ver todos los mensajes enviados y recibidos para una consulta en modo chat.	
Entrada	Consulta	
Salida	lista de mensajes	
Secuencia normal	Paso	Acción
	1	El doctor accede a la pantalla de consultas.
	2	El doctor hace clic en una consulta
	3	El sistema cargará el chat con los mensajes enviados y recibidos

PostCondición	Se ha listado las consultas para un doctor
Comentarios	NA.
Actores	Doctor

Tabla 33- Tabla de requisitos ver mensajes consulta doctor

Requisito	Ver consultas	
Identificador	2.6.3	
Prioridad	Alta	
Precondición	El paciente debe estar logueado en el dispositivo móvil y dirigirse a la pantalla de consultas, dentro de ella hacer clic en un elemento de la lista para mostrar el chat.	
Descripción	El paciente podrá ver todos los mensajes enviados y recibidos para una consulta en modo chat.	
Entrada	consulta	
Salida	lista de mensajes	
Secuencia normal	Paso	Acción
	1	El paciente accede a la pantalla de consultas.
	2	El paciente hace clic en una consulta
	3	El sistema cargará el chat con los mensajes enviados y recibidos
PostCondición	Se ha listado las consultas para un doctor	
Excepciones	Paso	Acción

	2	Si la consulta es nueva se mostrará un mensaje por defecto del doctor.
Comentarios	NA.	
Actores	Paciente	

Tabla 34- Tabla de requisitos ver mensajes consulta paciente

Enviar mensajes consultas

La Tabla 35 muestra los requisitos para enviar mensajes dentro del chat de consultas de un doctor y la Tabla 36 las de un paciente.

Requisito	Enviar mensajes consultas doctor	
Identificador	1.6.4	
Prioridad	Alta	
Precondición	El doctor debe estar logueado en la plataforma weby dirigirse a la pantalla de consultas, dentro de ella hacer clic en un elemento de la lista para mostrar el chat.	
Descripción	El doctor podrá añadir un mensaje al chat de la consulta.	
Entrada	Consulta	
Salida	se actualiza la lista de consultas	
Secuencia normal	Paso	Acción
	1	El doctor accede a la pantalla de consultas.
	2	El doctor hace clic en una consulta
	3	El sistema cargará el chat con los mensajes enviados y recibidos
	4	El doctor añade el texto que desea enviar en la casilla correspondiente y le da al botón de enviar.

	5	El mensaje se guarda en la base de datos y se actualiza la lista de mensajes correctamente
Excepciones	Paso	Acción
	5	Si da error interno en el servidor, la lista se refrescara eliminando el mensaje enviado.
PostCondición	Se ha añadido un mensaje al chat	
Comentarios	NA.	
Actores	Doctor	

Tabla 35- Tabla de requisitos enviar mensajes consulta doctor

Requisito	Enviar mensajes consultas paciente	
Identificador	2.6.4	
Prioridad	Alta	
Precondición	El paciente debe estar logueado en el dispositivo móvil y dirigirse a la pantalla de consultas, dentro de ella hacer clic en un elemento de la lista para mostrar el chat.	
Descripción	El doctor podrá añadir un mensaje al chat de la consulta.	
Entrada	consulta	
Salida	se actualiza la lista de consultas	
Secuencia normal	Paso	Acción
	1	El paciente accede a la pantalla de consultas.
	2	El paciente hace clic en una consulta

	3	El sistema cargará el chat con los mensajes enviados y recibidos
	4	El paciente añade el texto que desea enviar en la casilla correspondiente y le da al botón de enviar.
	5	El mensaje se guarda en la base de datos y se actualiza la lista de mensajes correctamente
Excepciones	Paso	Acción
	5	Si da error interno en el servidor, la lista se refrescara eliminando el mensaje enviado.
PostCondición	Se ha añadido un mensaje al chat	
Comentarios	NA.	
Actores	Paciente	

Tabla 36- Tabla de requisitos enviar mensajes consulta paciente

Apéndice B - Manual de Usuario Doctor

Este anexo tiene como objetivo dar al usuario una guía de cómo usar la aplicación si eres un doctor.

El usuario en primer lugar debe tener una cuenta, para ello seguir las figuras B.1, B.2 y B.3 que indican cómo acceder al registro desde la página principal de MedAlerta.

Figura Apéndice B-1- Pantalla Principal MedAlerta

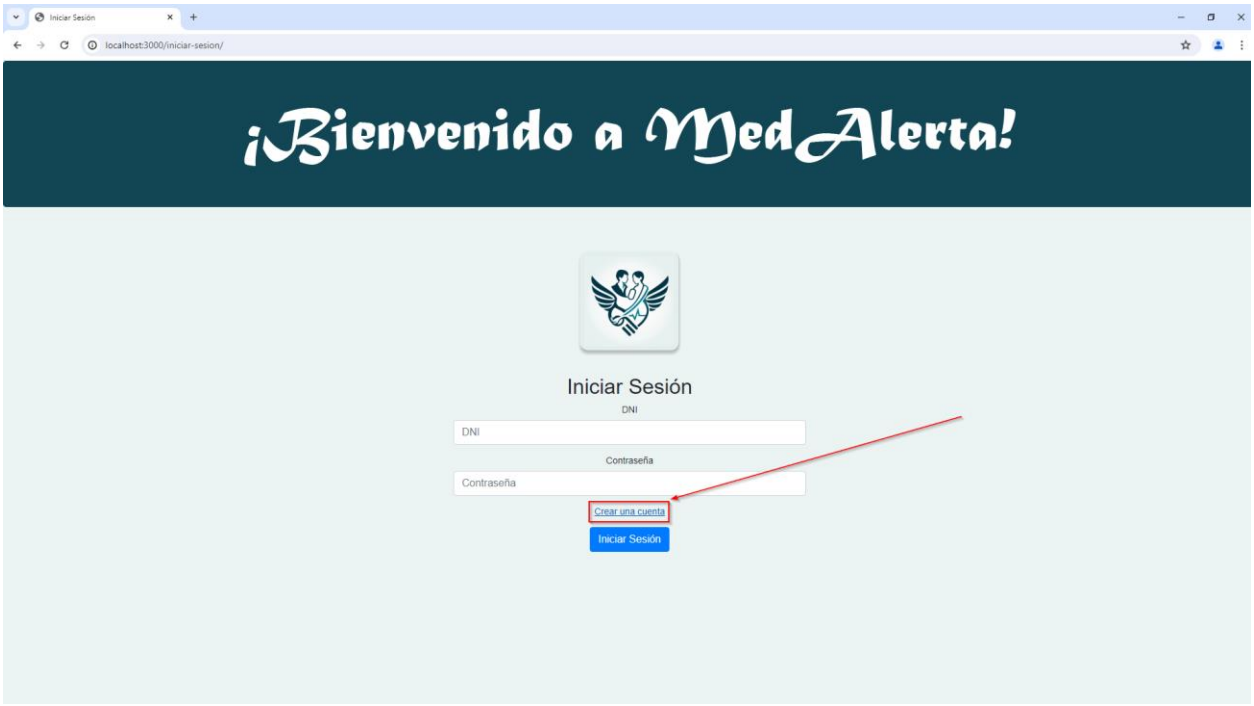

Figura Apéndice B-2- Pantalla Inicio Sesión MedAlerta

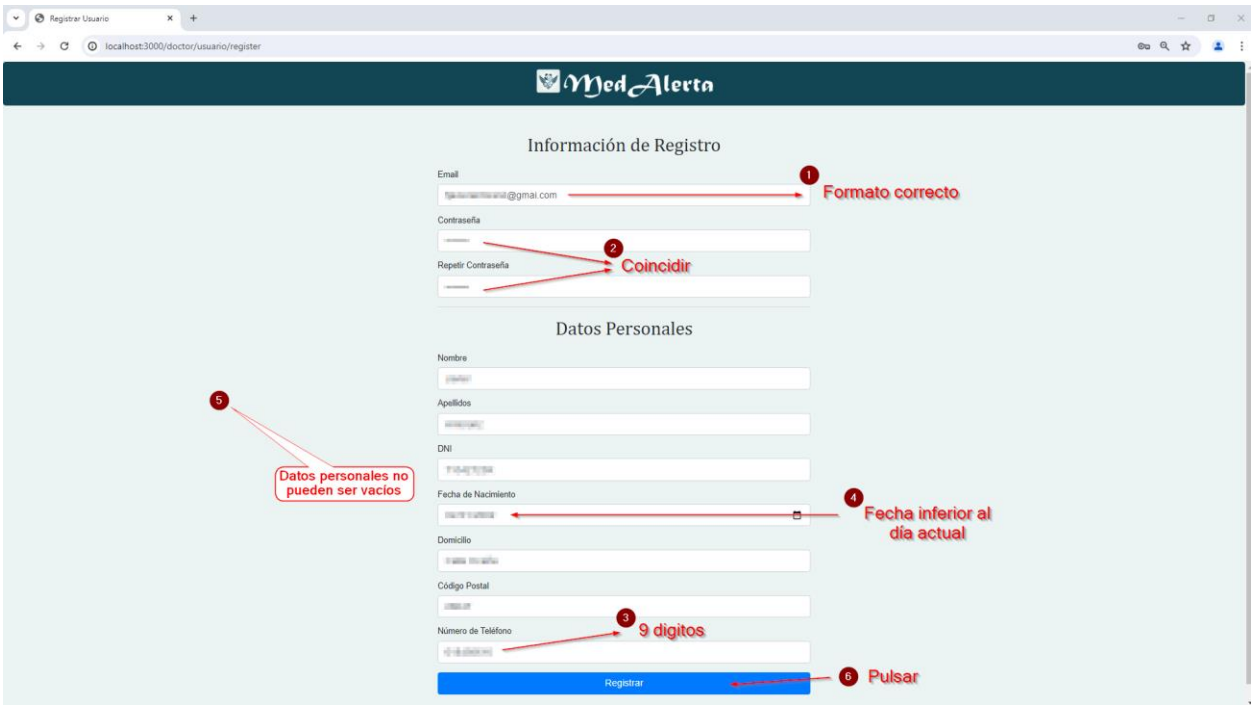

Figura Apéndice B-3- Pantalla Registro MedAlerta

Una vez creada una cuenta te redirige a la pantalla de inicio de sesión a no ser que se haya cometido algún error,es decir, no supera alguna de las validaciones indicadas en la figura B.3. Una vez en ella, se inicia sesión con la contraseña y dni registrado y se le da al botón de iniciar sesión (ver Figura B.2). Una vez pasado las credenciales de inicio de sesión te muestra la página principal con una serie de botones para acceder a los diferentes módulos de la aplicación. En la figura B.4 se piuede observar cómo acceder a los distintos módulos.

1. Gestión de Perfil. (Editar Perfil, Cerrar Sesión, Eliminar cuenta)
2. Gestión de Usuarios
3. Consultas
4. Calendario
5. Citas

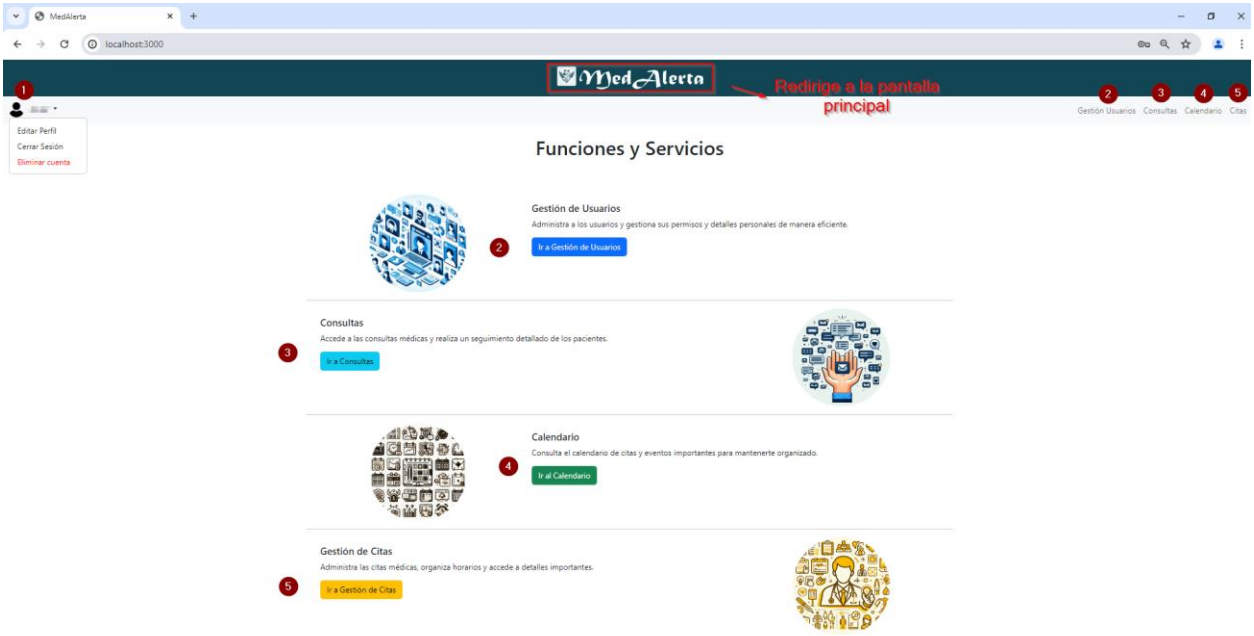

Figura Apéndice B-4- Pantalla Registro MedAlerta

Gestión de Perfil

Dentro de la gestión del perfil el usuario puede editar sus datos personales en editar Perfil (ver figura B.5).

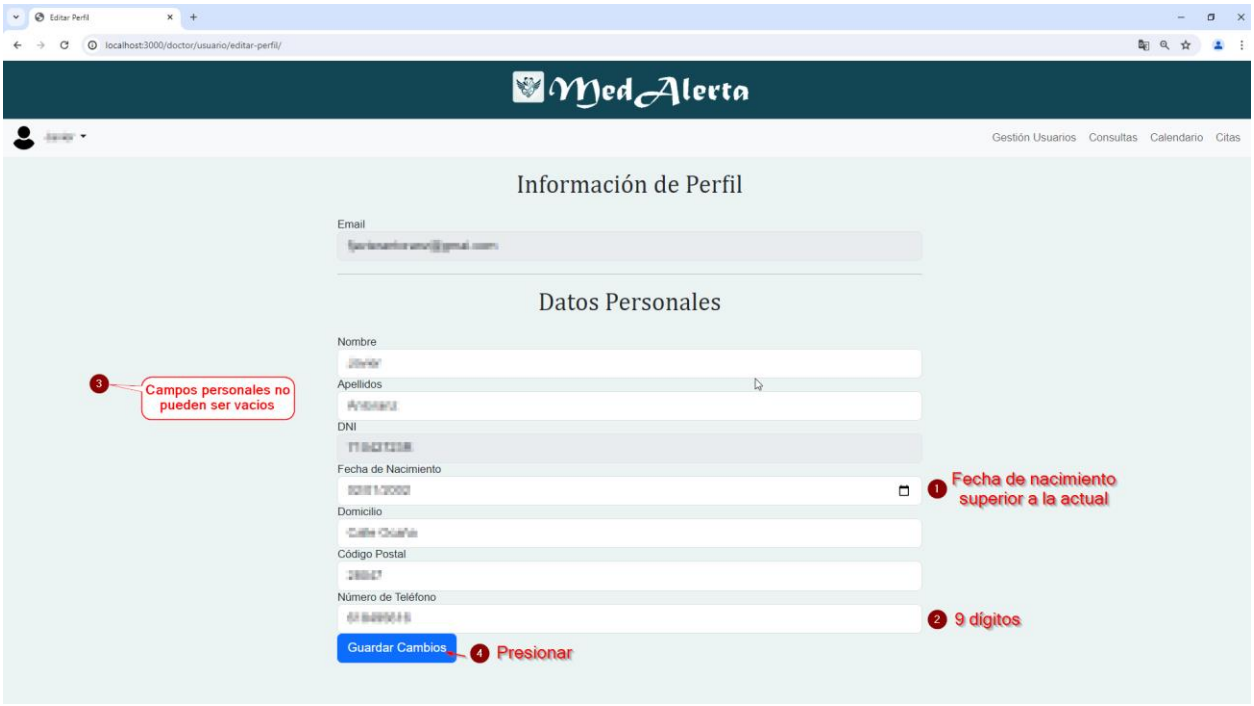

Figura Apéndice B-5- Pantalla Editar Perfil

Gestión de Usuarios

Dentro de este módulo el usuario puede realizar las siguientes acciones que pueden verse indicadas en la Figura B.6:

1. Búsquedas filtradas por Nombre, Apellidos, Correo y DNI.
2. Añadir un nuevo paciente a la lista de pacientes (ver Figura B.7).
3. Acceder a las diferentes funcionalidades para un paciente (ver Figura 8.8 y 8.9).
4. Eliminar un paciente de tu lista de pacientes.

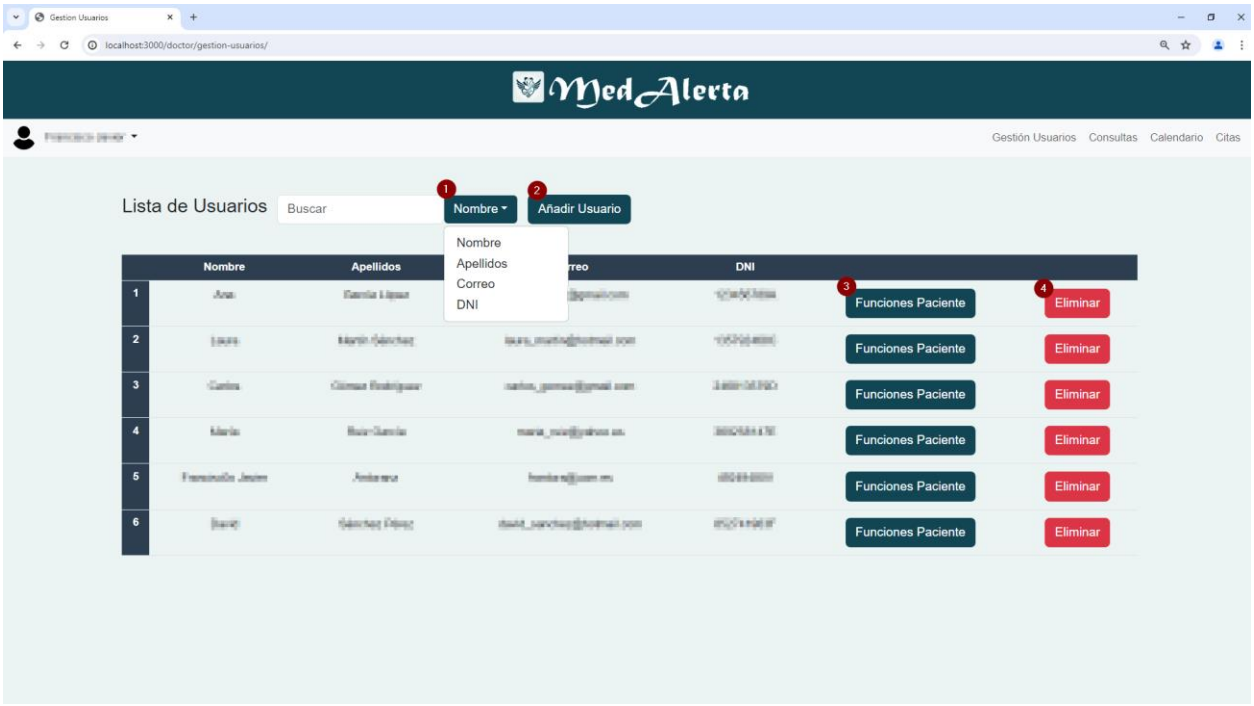

Figura Apéndice B-6- Gestión de usuarios

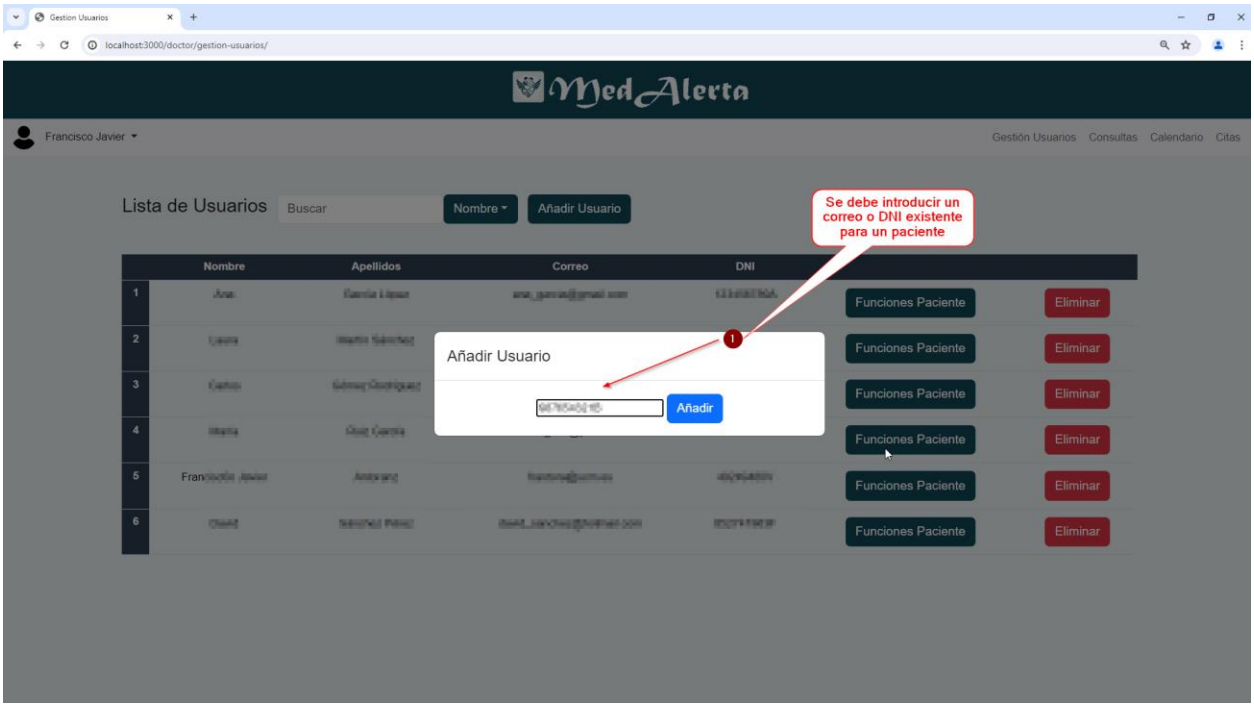

Figura Apéndice B-7- Gestión de usuarios - Añadir Usuario

Desde las funcionalidades de un usuario (ver Imagen H e I) el usuario podrá:

1. Iniciar un tratamiento
2. Pedir una cita
3. Crear Historial
4. Ver Historial
5. Ver Detalles de un historial
6. Eliminar un historial

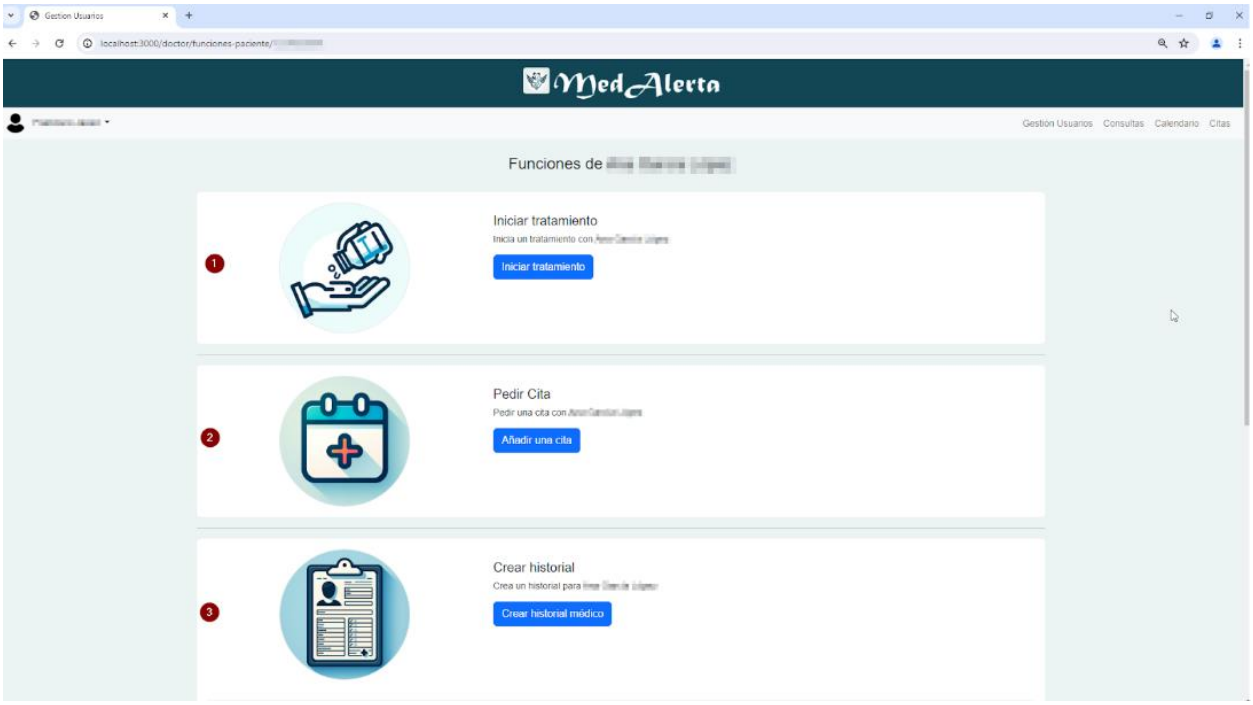

Figura Apéndice B-8- Gestión de usuarios - Funcionalidades de Usuario Parte 1

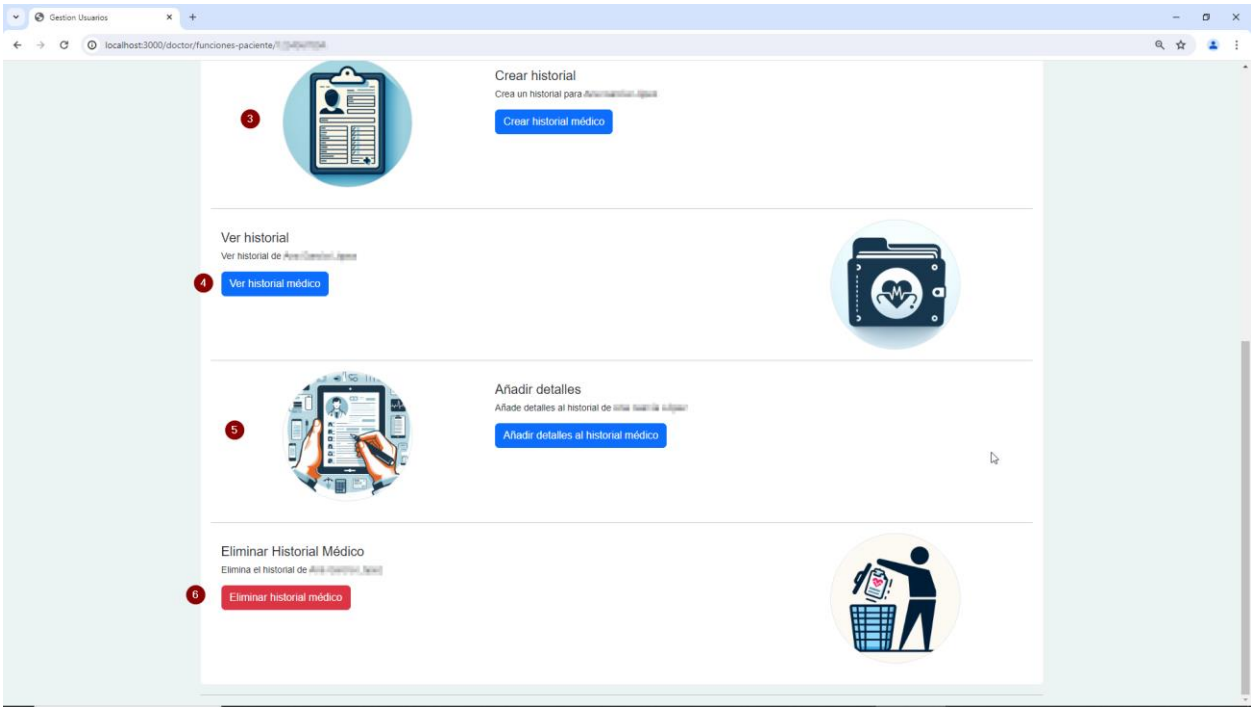

Figura Apéndice B-9.- Gestión de usuarios - Funcionalidades de Usuario Parte 21

Iniciar tratamiento

Para iniciar un tratamiento (Figura B.10) sobre un paciente el doctor debe introducir:

1. Descripción del tratamiento
2. Rellenar los datos para un medicamento
3. Pulsar para añadir un medicamento
4. Pulsar para eliminar un medicamento,
5. Pulsar para guardar el tratamiento.

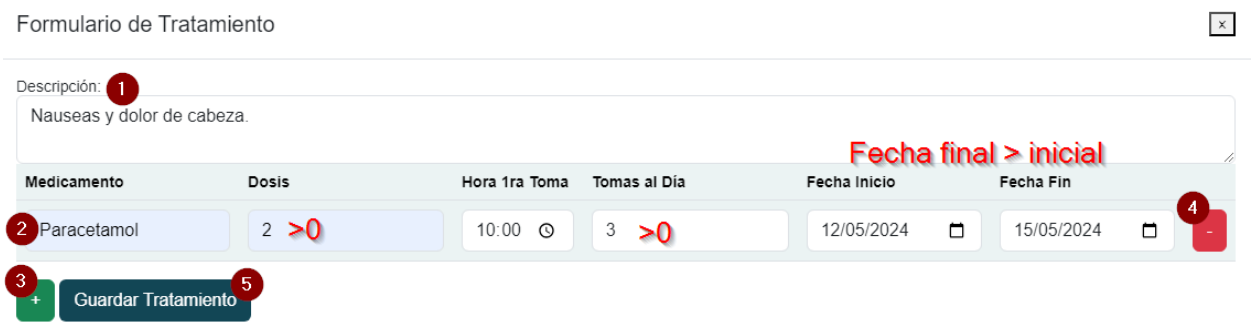

Figura Apéndice B-10- Gestión de usuarios - Iniciar Tratamiento

Asignar Cita

Para asignar una cita (Figura B.11) sobre un paciente el doctor debe introducir:

1. Descripción del tratamiento
2. Rellenar los datos para un medicamento
3. Pulsar para añadir un medicamento
4. Pulsar para eliminar un medicamento,
5. Pulsar para guardar el tratamiento.

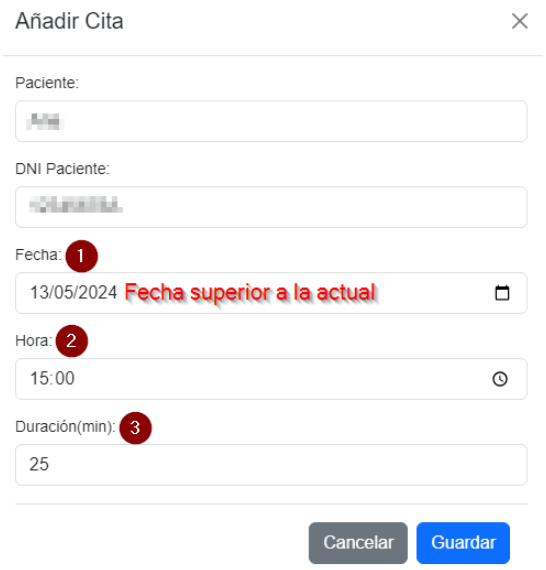

Figura Apéndice B-11- Gestión de usuarios - Añadir cita

Crear Historial

Para crear un historial (Figura B.12) sobre un paciente el doctor debe introducir:

1. Sexo masculino o femenino.
2. Fecha de nacimiento
3. Peso
4. Altura
5. Seleccionar una o más enfermedades.
6. Agregar una enfermedad a la lista de enfermedades
7. Alergias del paciente
8. Notas sobre el paciente, cosas a tener en cuenta
9. Detalles puntuales, se suelen anotar información relevante que se pueda añadir en una cita o consulta.
10. Todos los campos han de ser obligatorios exceptuando las notas y el detalle.

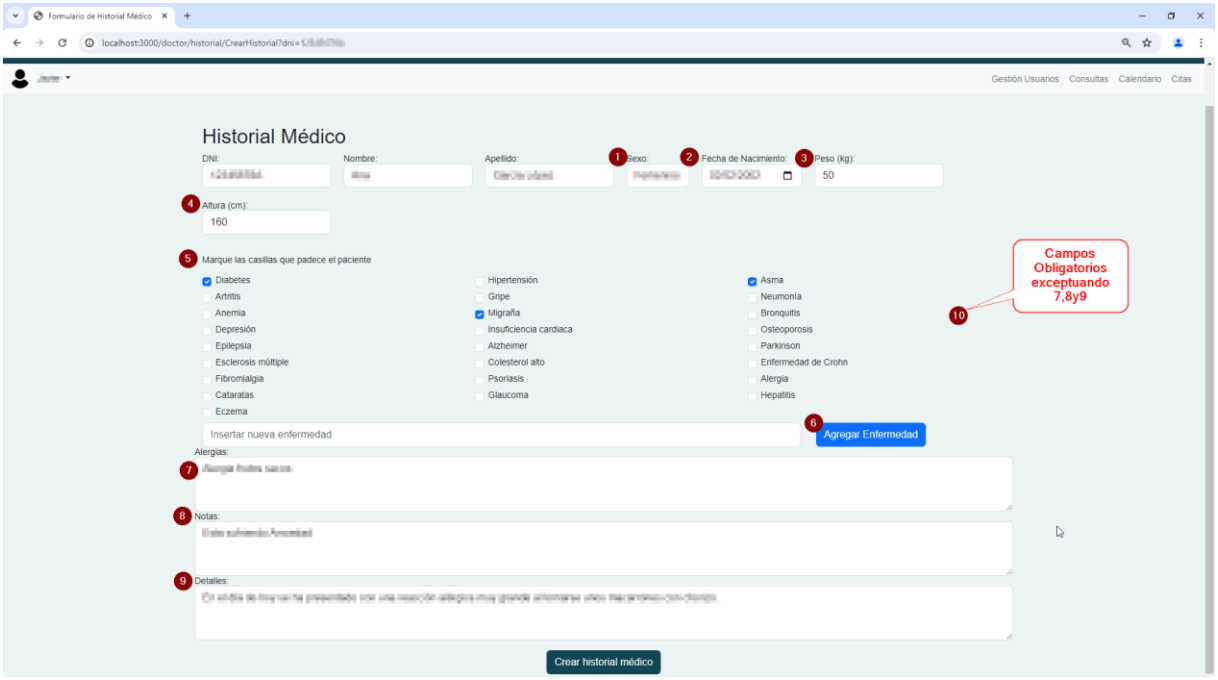

Figura Apéndice B-12- Gestión de usuarios - Crear Historial

Ver Historial

Al pulsar en el botón "Ver historial médico" se abre un fichero pdf(ver Figura B.13) que contiene el contenido de un Historial Médico(ver Figura B.14).

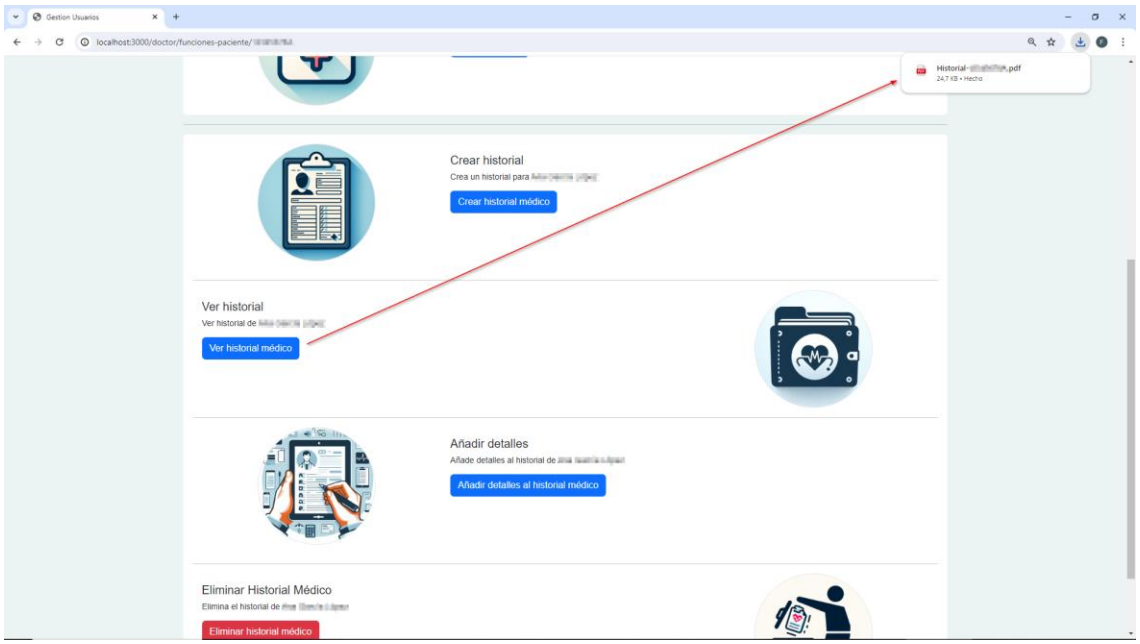

Figura Apéndice B-13Gestión de usuarios - Descargar Historial

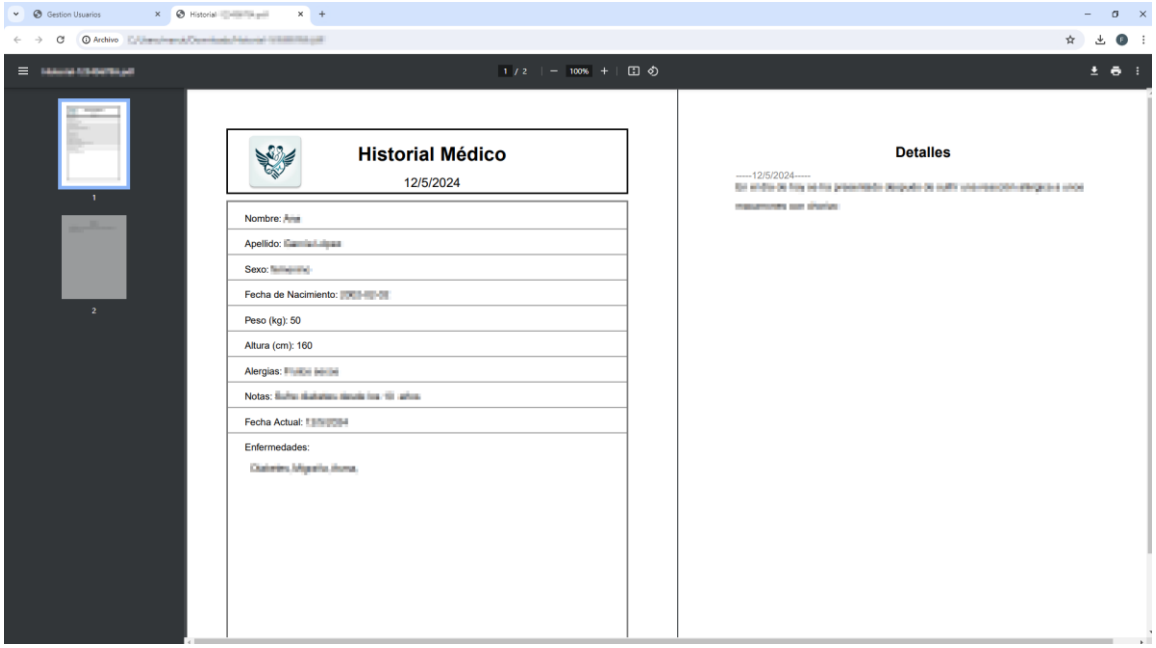

Figura Apéndice B-14- Gestión de usuarios - Pdf Historial

Añadir detalles

El usuario podrá añadir detalles al historial del paciente rellenando el formulario de la figura B.15. Estos datos deben ser relevantes para poder llevar una buena gestión del histórico de un paciente.

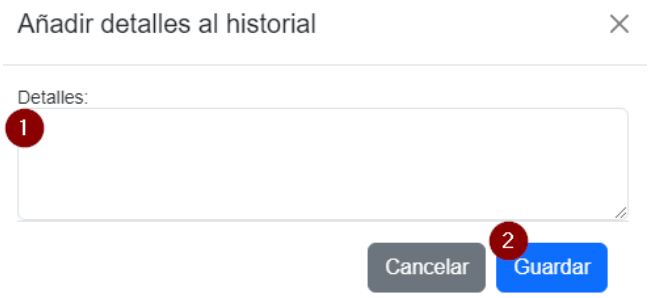

Figura Apéndice B-15- Gestión de usuarios - Añadir Detalles al Historial

Consultas

Si el usuario accede a la pantalla de consultas desde la pantalla principal o la barra superior, el usuario podrá (ver figura B.16):

1. Seleccionar una consulta que tenga con un paciente.
2. Ver el número de mensajes no leídos en un chat.

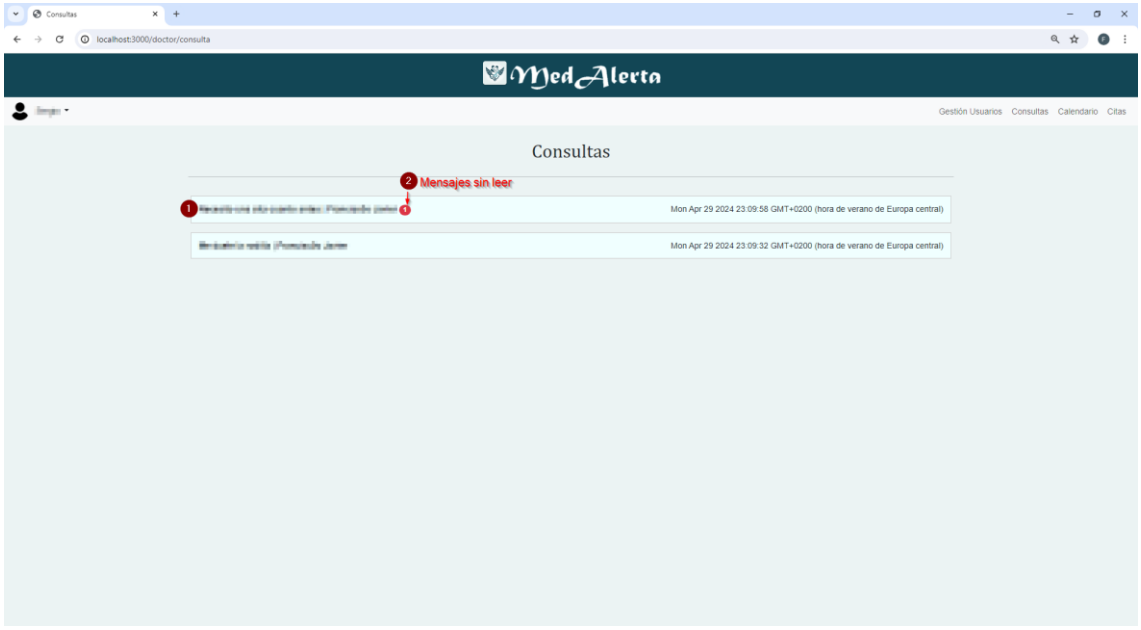

Figura Apéndice B-16- Consultas - Lista de consultas

Una vez el usuario haya hecho clic en una consulta podrá interactuar con ella a modo chat, recibiendo y enviando mensajes al instante (ver figura B.17).

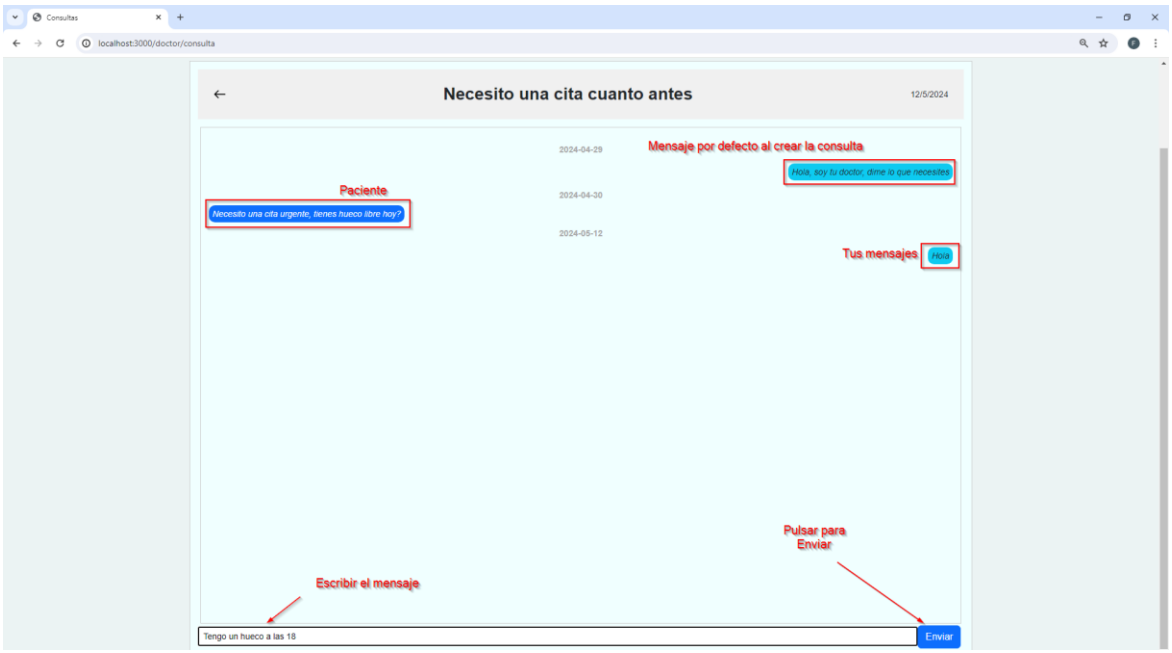

Figura Apéndice B-17- Consultas - Chat de consultas

Calendario

Si el usuario accede a la pantalla del calendario desde la pantalla principal o la barra superior, el usuario podrá ver las diferentes citas en tres modos:

1. Vista Mensual (ver Figura B.18).
2. Vista Semanal (ver Figura B.19).
3. Vista Diaria (ver Figura B.20).

Además, podrá eliminar una cita haciendo clic sobre una de ellas

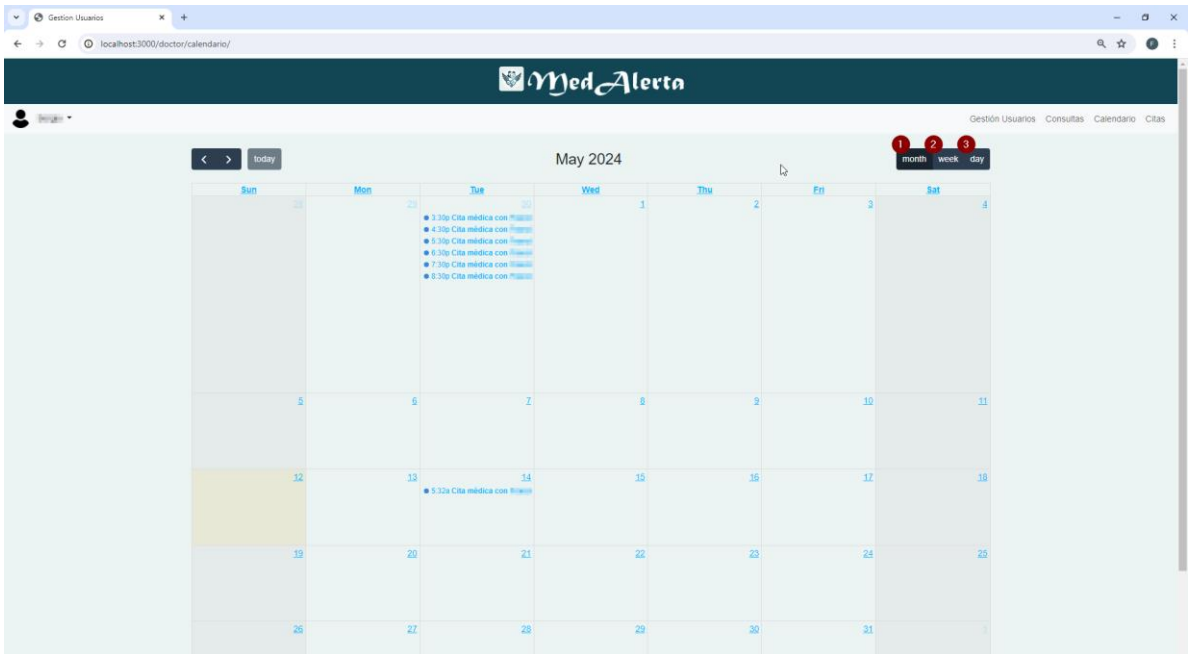

Figura Apéndice B-18- Calendario - Vista Mensual

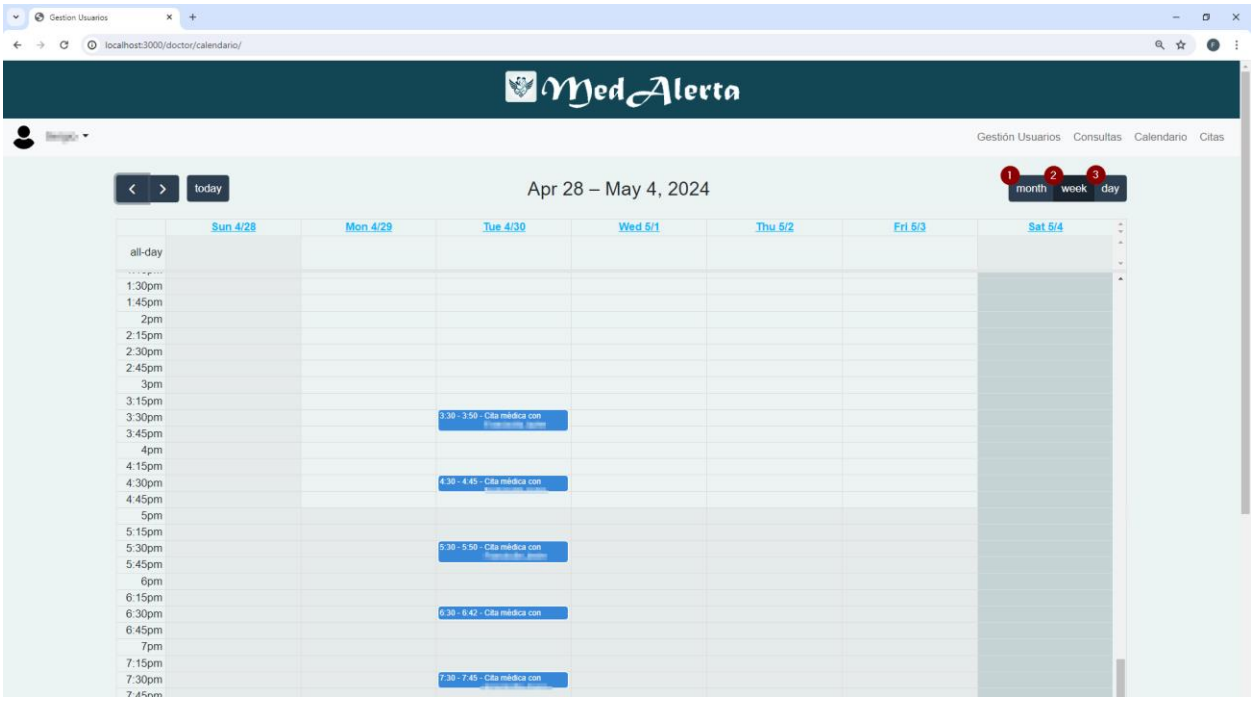

Figura Apéndice B-19- Calendario - Vista Semanal

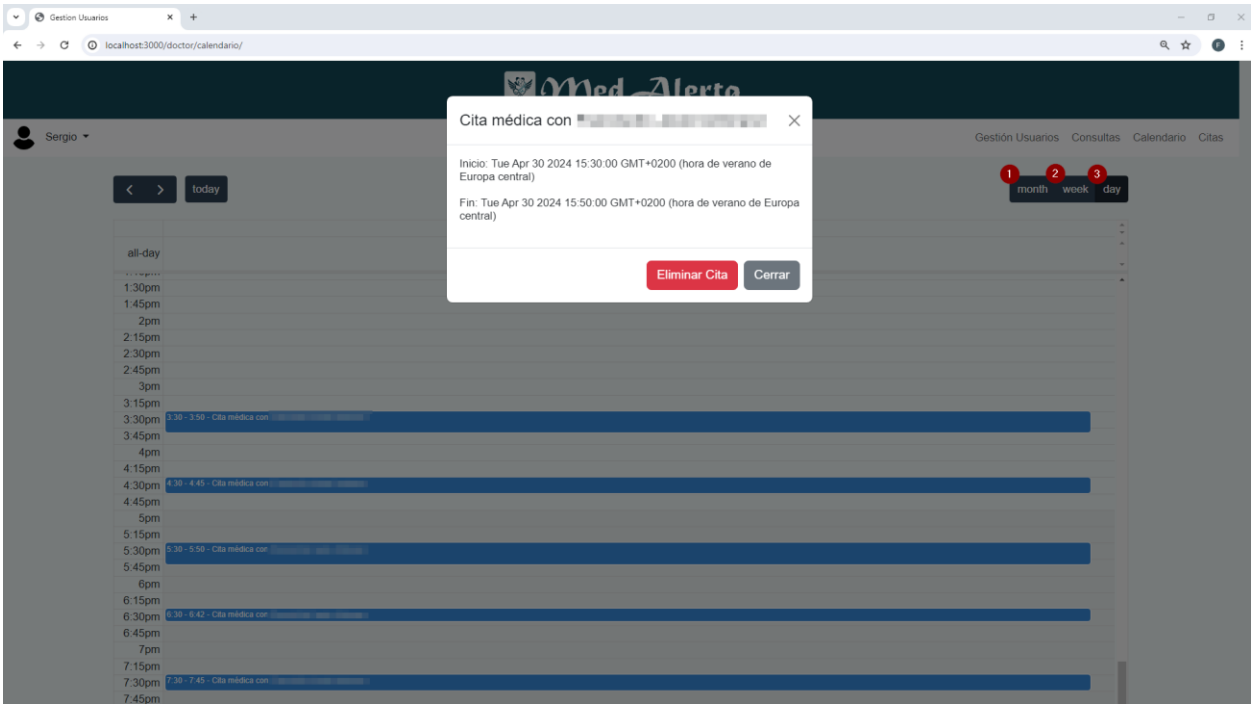

Figura Apéndice B-20- Calendario - Vista Diaria y cartel de detalle de la cita

Citas

Para acabar si el usuario accede a la pantalla de las citas desde la pantalla principal o la barra superior, el usuario podrá (ver figura B.21):

1. Ver las diferentes citas en una lista
2. Ver las diferentes peticiones de los pacientes.
3. Asignando el tiempo necesario para atender a la consulta de un paciente.
4. Aceptar petición.
5. Rechazar una petición.

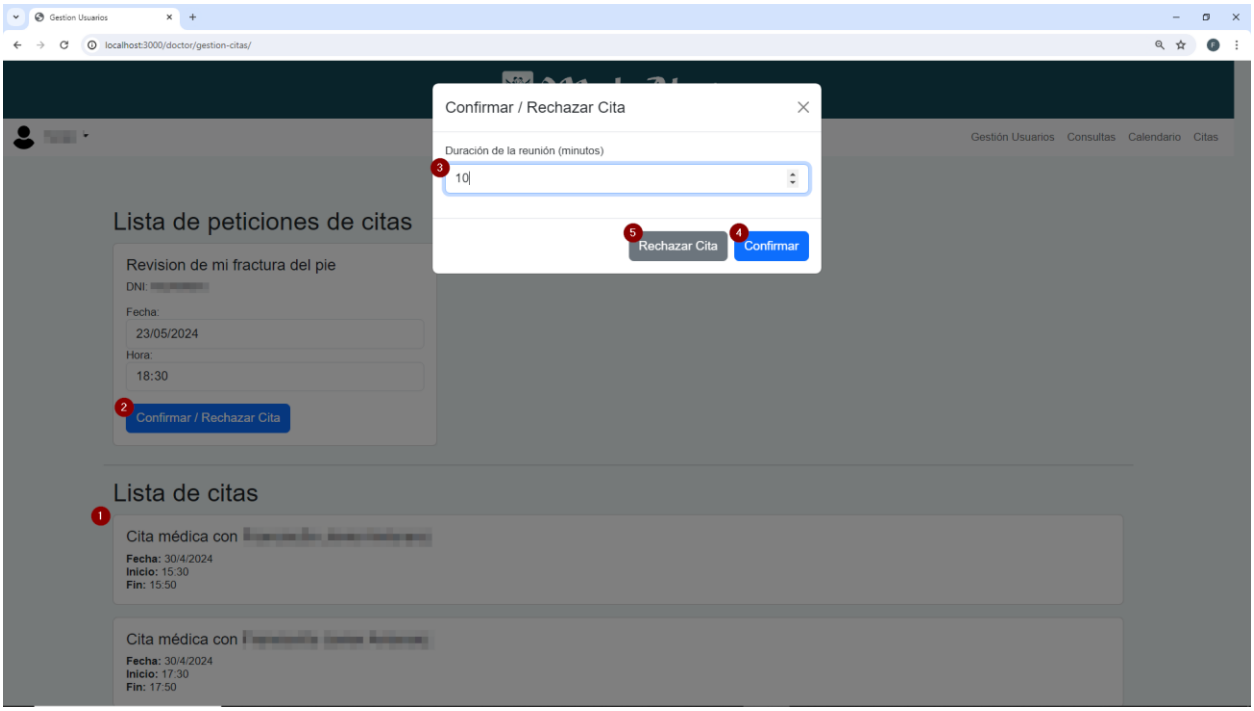

Figura Apéndice B-21- Citas- Ver/Confirmar/Rechazar peticiones de citas.

Apéndice C - Manual de Usuario Paciente

Este anexo tiene como objetivo dar al usuario una guía de cómo usar la aplicación móvil si eres un paciente.

El primer paso de un usuario es registrarse, para ello debe seguir los pasos mostrados en la figura C.1 y C.2 para el registro.

Figura Apéndice C-1- Inicio de sesión de la app

El botón 1 de la figura C.1, es para los usuarios que deseen cambiar su contraseña o la han olvidado, al hacer clic se muestra la pantalla para cambiar contraseña que aparece en la figura C.6, ya que también aparece en la pantalla de editar perfil del paciente.

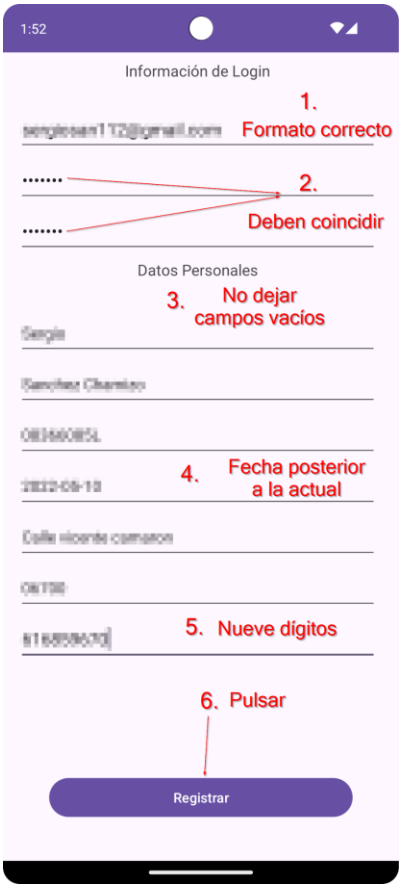

Figura Apéndice C-2- Registro de la app

Una vez presionado el botón de "Registrar", suponiendo que todos los campos son rellenados y sean válidos tal y como aparecen en la figura C.2, se habrá creado la cuenta y aparecerá de nuevo la pantalla de inicio de sesión. Esta vez se inicia sesión con el dni y contraseña dándole al botón de iniciar sesión (ver figura C.1).

Hay un paso intermedio entre el inicio de sesión y poder utilizar la aplicación llamado validar cuenta (ver figura C.3).

Figura Apéndice C-3- Validar cuenta

Tras validar la cuenta habiendo introducido el código que llega al correo, se mostrará pantalla principal de la app (ver figura C.4), aquí se pueden ver varias funciones separadas en módulos:

1. Gestión del perfil (Editar Perfil, Cerrar Sesión, Eliminar Cuenta)
2. Consultas
3. Calendario
4. Crear Cita
5. Interacción con alarmas

Figura Apéndice C-4- Pantalla principal de la aplicación móvil.

Gestión del Perfil

Dentro de la gestión del perfil el usuario puede editar sus datos personales en editar Perfil (ver figura C.5).

Figura Apéndice C-5- Gestión del perfil - Editar perfil.

El usuario tiene la posibilidad de cambiar la contraseña dando al botón que aparece representado con el número 4 en la figura C.5, una vez seleccionado, aparecerá la pantalla mostrada en la figura C.6.

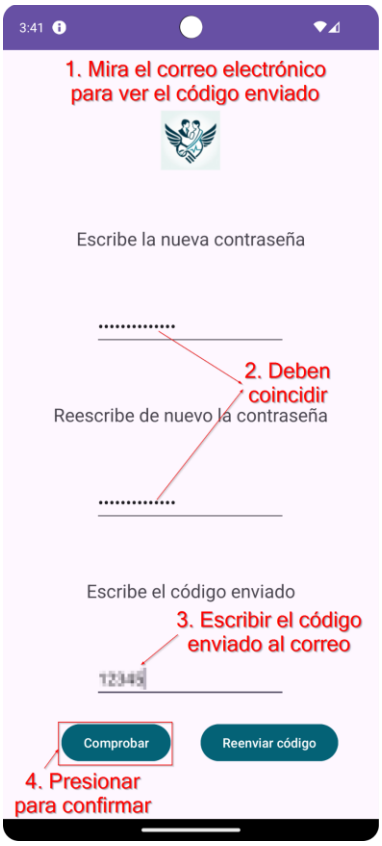

Figura Apéndice C-6- Gestión del perfil - Cambio de contraseña.

Consultas

Si un usuario presiona el botón Consultas desde la pantalla principal o desde cualquier pantalla, el usuario podrá (ver figura C.7):

1. Seleccionar una consulta para ver el chat con el doctor.
2. Ver si hay mensajes no leídos en una consulta.
3. Presionar el botón para crear una consulta.

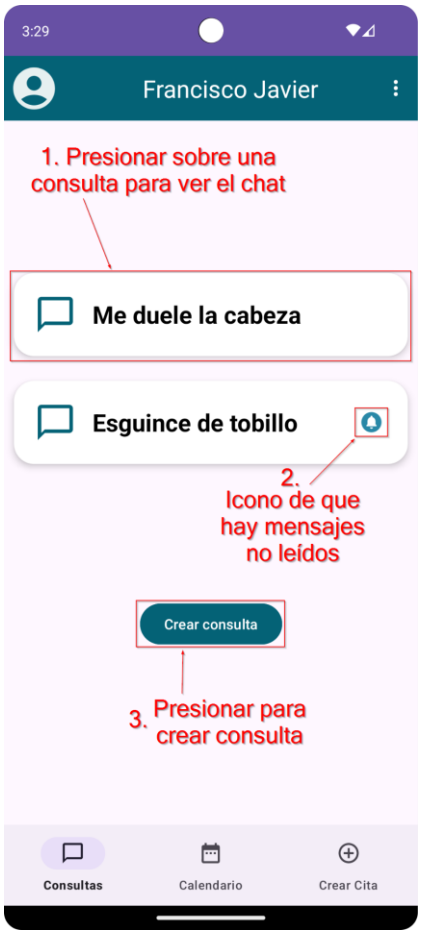

Figura Apéndice C-7- Consultas - Consultas del paciente

Al pulsar sobre una consulta, se te mostrará el chat y los mensajes que tienes con el doctor que corresponda (ver figura C.8).

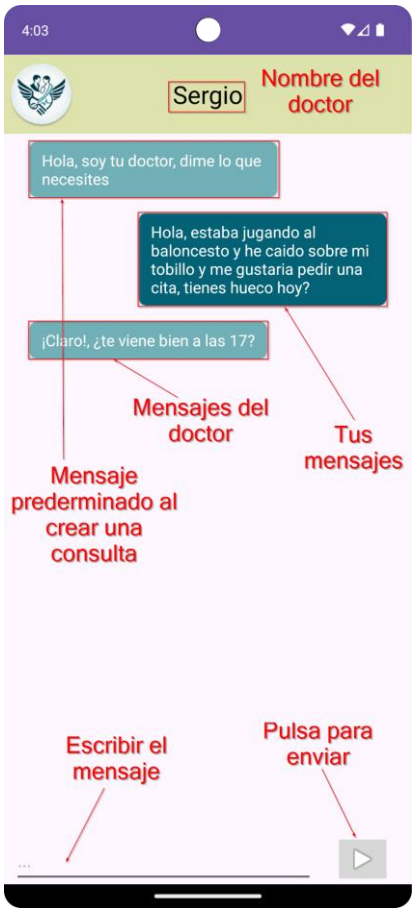

Figura Apéndice C-8- Consultas - Chat de consultas de la app

Al pulsar sobre el botón "Crear consulta", se podrá crear un nuevo chat con un doctor (ver figura C.9). En el momento que el doctor manda un mensaje, salta una notificación (ver figura C.10) al móvil con sonido en la que puedes pulsar para ir a la ventana de consultas vistas en la figura C.7.

Figura Apéndice C-9- Consultas - Crear consulta con un doctor

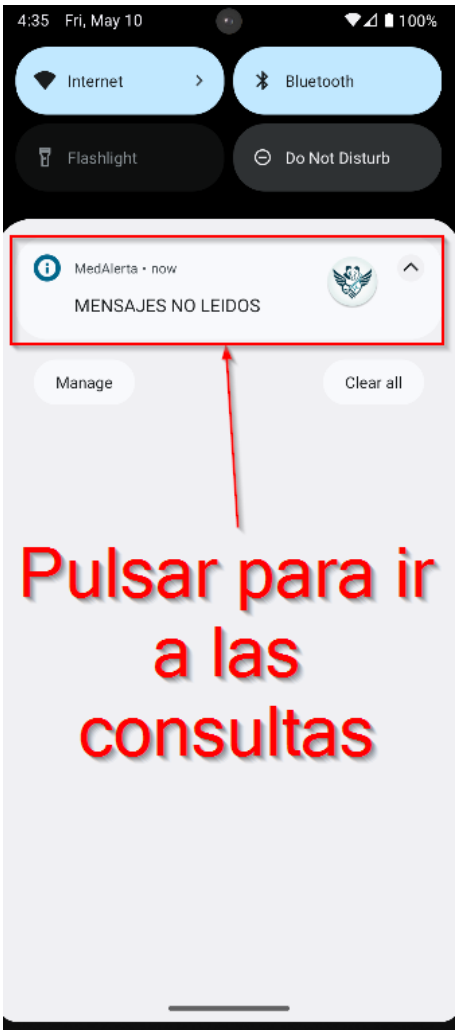

Figura Apéndice C-10- Consultas - Chat de consultas de la app

Calendario

Al presionar el botón "Calendario" del menú inferior podrás ver las citas que tiene el paciente (ver figura C.11). Además puedes ver la duración de la cita más cercana que tengas en ese momento haciendo clic en dicho día (ver figura C.12).

Figura Apéndice C-11- Calendario - Vista principal del calendario

Figura Apéndice C-12- Calendario - Detalles de la cita

Crear Cita

Para crear una cita, presiona sobre el botón "Crear Cita" del menú inferior, al hacerlo irás a una pantalla donde podrás rellenar un formulario (ver figura C.13) y solicitar una cita al doctor, el cuál éste podrá aceptar o cancelar en la web.

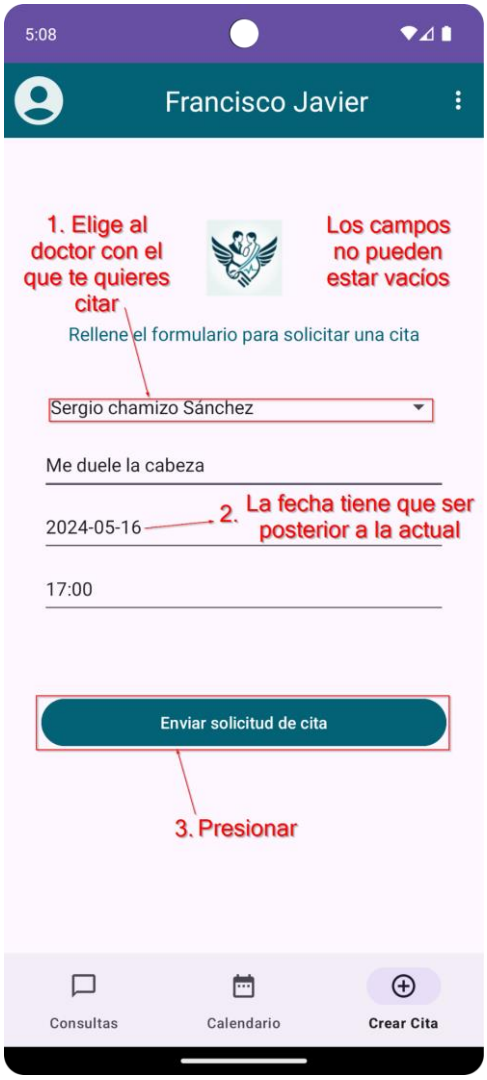

Figura Apéndice C-13- Crear Cita- Solicitud de cita

Interacción con alarmas

El doctor puede iniciar un tratamiento al paciente, en el momento que el móvil recibe la alarma y es la hora adecuada, te salta una notificación con el medicamento y la dosis recetada para ese momento (ver figura C.14) y la alarma no para de sonar hasta que la paras, la deslizas o la presionas, es decir hasta que interactúas con ella.

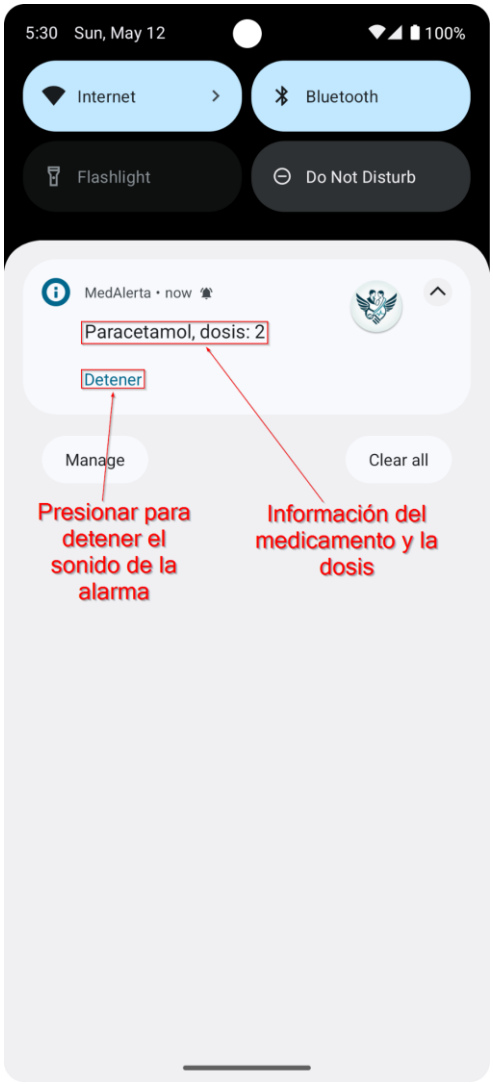

Figura Apéndice C-14- Alarma - Notificación de la alarma