Calificación: *9.5*

Desarrollo de una aplicación inteligente para predecir la validez de los riñones donados procedentes de donantes fallecidos tras una asistolia no controlada
Development of an intelligent application to predict the validity of donated kidneys from deceased donors after uncontrolled asystole

---

Trabajo de Fin de Grado
Curso 2024–2025

Autor
Laura Herrero Carranza

Director
Antonio Sarasa Cabezuelo

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Desarrollo de una aplicación inteligente para predecir la validez de los riñones donados procedentes de donantes fallecidos tras una asistolia no controlada
Development of an intelligent application to predict the validity of donated kidneys from deceased donors after uncontrolled asystole

Trabajo de Fin de Grado en Ingeniería Informática

Autor
Laura Herrero Carranza

Director
Antonio Sarasa Cabezuelo

Convocatoria: *Junio* 2025

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

24 de Mayo de 2025

Dedicatoria

A mis padres y a mi hermana. A mis amigas y a mis amigos y, en especial, a mis compañeras de piso y a Miguel.

Agradecimientos

En primer lugar, quiero agradecer a mi tutor, Antonio Sarasa Cabezuelo, la oportunidad de poder realizar este Trabajo Fin de Grado sobre un tema tan interesante. Gracias por haberme dado la libertad para explorar y desarrollar el proyecto a mi manera, pero al mismo tiempo guiarme y ayudarme cuando lo necesitaba.

En segundo lugar, quiero agradecer al doctor Carlos Rubio Chacón su comprensión y su colaboración, por resolver mis dudas médicas y por orientarme en los requisitos de la aplicación. Asimismo, agradezco a Amado Andrés Belmonte por haber hecho posible que este trabajo fuese una realidad.

Por último, quiero agradecer a mis padres, que siempre me han apoyado y alentado cuando lo necesitaba, a mi hermana, por sus críticas y su incondicional ayuda, y sobre todo a mis compañeras de piso (y amigas) y a Miguel, por haber soportado todas mis dudas y mis momentos durante este tiempo. Por haberme dicho qué color quedaba mejor y por haber realizado vuestras propias predicciones. Muchas gracias a todos.

Resumen

Desarrollo de una aplicación inteligente para predecir la validez de los riñones donados procedentes de donantes fallecidos tras una asistolia no controlada

En 2024, España superó los objetivos marcados por la Organización Nacional de Trasplantes realizando 6464 trasplantes, un 10 % más que el año anterior. En términos absolutos, el trasplante renal fue el más numeroso, con 4047 procedimientos realizados, lo que representa también un crecimiento del 10 % respecto a 2023.

El aumento de supervivencia y calidad de vida esperables tras el trasplante renal hacen necesario un estudio intensivo que identifique qué factores determinan si un donante será válido o no. Carlos Rubio Chacón, médico en Urgencias y Emergencias del SUMMA 112, ha publicado su tesis doctoral en la que identifica seis variables clave para determinar si un donante es válido o no. A partir de este hallazgo, surge la necesidad de desarrollar una aplicación móvil que, introduciendo estos valores, determine de forma rápida si el donante será o no válido.

Además, esta aplicación móvil se complementa con una aplicación web que, aprovechando la base de datos empleada en la tesis, servirá como plataforma para entrenar diversos modelos de Machine Learning, con el objetivo de investigar su eficacia en la predicción de la validez de los donantes.

Palabras clave

Trasplante, Órgano, Renal, Válido, Aplicación, Machine Learning

Abstract

Development of an intelligent application to predict the validity of donated kidneys from deceased donors after uncontrolled asystole

In 2024, Spain surpassed the targets set by the National Transplant Organization by performing 6464 transplants, 10% more than the previous year. In absolute terms, kidney transplants were the most numerous, with 4,047 procedures carried out, which also represents a 10% increase compared to 2023.

The expected increase in survival and quality of life after a kidney transplant highlights the need for an in-depth study to identify the factors that determine whether a donor is suitable. Carlos Rubio Chacón, a physician in Emergency and Urgent Care at SUMMA 112, published his doctoral thesis identifying six key values to determine donor suitability. Based on this finding, the need arises to develop a mobile application that, by providing these values, can quickly determine whether a donor is valid or not.

In addition, this mobile application is complemented by a web application that, using the database employed in the thesis, serves as a platform for training various Machine Learning models with the aim of investigating their effectiveness in predicting donor suitability.

Keywords

Transplant, Organ, Kidney, Suitable, Application, Machine Learning

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

Actualmente, tanto el sistema sanitario como el informático están en constante evolución, y la incorporación de uno en el otro es clave para avanzar en ambos ámbitos. Poder desarrollar una aplicación móvil que convierta los resultados de la tesis doctoral de Carlos Rubio Chacón en una herramienta real y práctica es especialmente gratificante, y es lo que motivó el proceso de trabajo durante todo el desarrollo del mismo.

En relación con el tema en concreto, el tiempo que transcurre desde que se recibe la llamada sobre una posible donación hasta que esta se lleva a cabo es crucial. Poder desarrollar una aplicación que reduzca este tiempo de manera fiable y eficiente no solo tiene un impacto médico, sino que también representa un reto en el campo de la informática.

Por otro lado, la idea de poder entrenar modelos de machine learning y analizar sus resultados resulta también un tema muy atractivo y actual.

Este trabajo refleja dos inquietudes principales: por un lado, la de crear una aplicación basada en un caso real, y por otro, explorar su evolución mediante el uso de modelos predictivos. Esto representa una oportunidad para contribuir al vínculo entre la tecnología, la IA y la medicina, con el objetivo de lograr un sistema sanitario más ágil, accesible y eficiente.

1.2. Objetivos

Este trabajo consta de dos objetivos claros: una aplicación sencilla, para sanitarios, que actualice una base de datos con donantes reales, y, por otro lado, una interfaz que permita entrenar modelos de Machine Learning y determinar cómo de fiables son estos para las predicciones de donantes válidos.

La aplicación consta tanto de una versión móvil como web, ambas conectadas a la misma base de datos, pero ofreciendo diferentes funcionalidades.

La aplicación móvil la pueden utilizar tanto médicos como administradores. Para los médicos, se trata de una aplicación donde la funcionalidad principal consiste en rellenar un formulario con los diferentes valores de un posible donante, y mostrar si

el donante es o no válido. Además, cada médico podrá ver su historial de predicciones realizas, así como una gráfica donde ver cuántas predicciones son válidas y no válidas. Para los administradores, se trata de una aplicación donde pueden ver un listado de todos los médicos que son usuarios de la aplicación y qué predicciones ha realizado cada uno.

La aplicación web es únicamente para administradores, ya que son los que pueden acceder a todas las predicciones realizadas por los médicos. El objetivo principal es que el administrador pueda añadir predicciones al conjunto de entrenamiento de diferentes modelos de Machine Learning. Así, posteriormente, se podrán estudiar los resultados de los modelos y comprobar si dan resultados similares entre ellos.

La comunicación entre el frontend (la aplicación web) y el backend (el entrenamiento de los modelos) se consigue a través de una API REST que envía el modelo seleccionado por el administrador y los nuevos datos de entrenamiento al backend, quien implementa toda la lógica de los modelos.

1.3. Plan de trabajo

La organización consiste en una serie de iteraciones con diferentes plazos para alcanzar los hitos establecidos. En la siguiente tabla se muestra la planificación seguida.

Objetivo	Fecha inicio	Fecha fin
Especificación de los requisitos	16/09/2024	29/09/2024
Estudio y obtención de las tecnologías empleadas	30/09/2024	02/10/2024
Planificación del proyecto	03/10/2024	07/10/2024
Creación de la base de datos	08/10/2024	08/10/2024
Módulo gestión de usuarios	09/10/2024	23/10/2024
Módulo administración	24/10/2024	03/11/2024
Cambio en los requisitos	20/11/2024	20/11/2024
Módulo gestión de usuarios en aplicación móvil	21/11/2024	12/12/2024
Módulo de Médico en aplicación móvil	13/12/2024	28/02/2025
Módulo de Administrador en aplicación móvil	01/03/2025	14/03/2025
Módulo gestión de usuarios en aplicación web	15/03/2025	25/03/2025
Módulo de Administrador en aplicación web	26/03/2025	08/04/2025
Estudio de los modelos de Machine Learning	09/04/2025	13/04/2025
Módulo para el entrenamiento de los modelos	14/04/2025	22/04/2025
Estudio de los resultados	23/04/2025	01/05/2025
Memoria	05/10/2024	24/05/2025

Tabla 1.1: Plan de trabajo

Introduction

Motivation

Currently, both the healthcare system and the IT field are in constant evolution, and integrating one into the other is key to advancing in both areas. Being able to develop a mobile application that turns the results of Carlos Rubio Chacón's doctoral thesis into a real and practical tool is particularly gratifying, and this is what motivated the work process throughout its development.

Regarding the specific topic, the time elapsed from when a call is received about a potential donation to when the donation is carried out is crucial. Developing an application that reliably and efficiently reduces this time has not only a medical impact but also represents a challenge in the field of computer science.

Additionally, the idea of training machine learning models and analyzing their results is a highly attractive and relevant topic.

This work reflects two main areas of interest: on the one hand, the desire to create an application based on a real-life case, and on the other, to explore its evolution through the use of predictive models. This represents an opportunity to contribute to the connection between technology, AI, and medicine, with the goal of achieving a more agile, accessible, and efficient healthcare system.

Goals

This project has two clear objectives: a simple application for healthcare professionals to update a database with real donors, and, on the other hand, an interface that allows training Machine Learning models and determining how reliable these models are for predicting valid donors.

The application includes both a mobile and a web version, both connected to the same database but offering different functionalities.

The mobile application can be used by both doctors and administrators. For doctors, it is an app where the main feature is filling out a form with the different values of a potential donor and displaying whether the donor is valid or not. Additionally, each doctor can view their prediction history, as well as a graph showing how many predictions are valid and invalid. For administrators, the application allows them to see a list of all doctors who are users of the app and the predictions each has made.

The web application is for administrators only, as they are the ones who can access all the predictions made by the doctors. The main goal is for the administrator to be able to add predictions to the training dataset of different Machine Learning

models. Later on, the results of the models can be studied to see if they produce similar outcomes.

The communication between the frontend (the web application) and the backend (the model training) is achieved through a REST API, which sends the model selected by the administrator and the new training data to the backend, which implements all the model logic.

Work plan

The organization consists of a series of iterations with different deadlines to achieve the established milestones. The following table shows the planning that was followed.

Objective	Start date	End date
Specification of requirements	16/09/2024	29/09/2024
Study and acquisition of the technologies used	30/09/2024	02/10/2024
Project planning	03/10/2024	07/10/2024
Database creation	08/10/2024	08/10/2024
User management module	09/10/2024	23/10/2024
Administration module	24/10/2024	03/11/2024
Change in requirements	20/11/2024	20/11/2024
User management module in mobile app	21/11/2024	12/12/2024
Doctor module in mobile app	13/12/2024	28/02/2025
Administrator module in mobile app	01/03/2025	14/03/2025
User management module in web application	15/03/2025	25/03/2025
Administrator module in web application	26/03/2025	08/04/2025
Study of Machine Learning models	09/04/2025	13/04/2025
Training module for the models	14/04/2025	22/04/2025
Study of results	23/04/2025	01/05/2025
Thesis	05/10/2024	24/05/2025

Tabla 1.2: Work Plan

Capítulo 2

Estado de la Cuestión

Para entender bien el contexto de este Trabajo Fin de Grado se debe tener en cuenta que, la raíz de todo esto, es la tesis doctoral publicada por el doctor Carlos Rubio Chacón con el título *Niveles de capnometría como indicador de validez de los donantes en asistolia no controlada (DANC) y de la evolución de sus injertos renales trasplantados*. (36) Esta tesis muestra, entre otras cosas, que hay seis variables que son las que determinan si un donante será válido o no. Tras este hallazgo surge la necesidad de una aplicación móvil para que, en el momento que se recibe información sobre un posible donante, se pueda comprobar si este será válido o no de forma rápida y eficaz.

A continuación se nombran diferentes proyectos que han servido de ayuda e inspiración a lo largo del desarrollo del proyecto.

2.1. UK Deceased Donor Kidney Transplant Outcome Prediction (UK-DTOP) Tool

Esta herramienta desarrollada en Reino Unido emplea inteligencia artificial para mejorar la predicción de los resultados en trasplantes renales de donantes fallecidos. Utiliza tres modelos de aprendizaje automático: XGBoost, Random Forest y Árboles de Decisión, y ha demostrado una notable capacidad para discriminar entre donantes de alto y bajo riesgo. Además, emplea k-means clustering, un algoritmo de aprendizaje no supervisado, para identificar distintos grupos de donantes y trasplantes. En el artículo se indica que el modelo que obtiene mejor resultados es el XGBoost. (37)

Gracias a este proyecto (y a otros), se tomó la decisión de implementar los modelos de Árbol de Decisión, Random Forest y Clustering. Se consideró la posibilidad de probar XGBoost, pero la falta de registros suficientes en la base de datos llevó a descartar esta opción, ya que funciona mejor ocn datasets grandes. (9)

2.2. Almacenamiento en la nube

Al buscar información sobre qué base de datos debía utilizarse para la aplicación en Android Studio, se encontraron muchas recomendaciones para usar Firebase, ya

que ofrece una integración sencilla con esta plataforma.

Al no haberse trabajado nunca con una base de datos en la nube, se procedió a informarse sobre las ventajas e inconvenientes que estas presentaban. En un informe de Google Scholar, escrito por Ángel Gutiérrez, se destacan varias ventajas e inconvenientes de usar estas bases de datos. Las ventajas que mejor se aplican a este proyecto son la ubicuidad de la nube (por si en algún momento se pierden los datos) y que se trata de un software centralizado y sin instalación (lo que hace más cómodo su uso).

Aunque existen debates sobre la seguridad de los datos en la nube, el uso Firebase también sirve como experiencia para trabajar con este tipo de herramientas.

2.3. SUMMA 112

El sector sanitario que emplearía la aplicación es el Servicio de Urgencia Médica de Madrid. Por ello, navegar en su web ha inspirado este trabajo sobre todo en cuanto al diseño de la interfaz.

Capítulo 3

Tecnologías empleadas

3.1. Entornos de Desarrollo

- Android Studio

  Android Studio (3) es un entorno de desarrollo integrado oficial empleado en el desarrollo de aplicaciones para Android. Cuenta con un sistema de compilación flexible basado en Gradle, que permite una gestión eficiente de dependencias. Además, incluye un emulador en el que puedes ver cómo se comporta la aplicación en tiempo real, incluyendo todas las funcionalidades de un dispositivo móvil. (4) Se ha utilizado para el desarrollo de la aplicación móvil.

- Visual Studio Code

  Visual Studio Code (26) es un editor de código fuente desarrollado por Microsoft, que destaca por ser ligero, rápido e intuitivo. (27) Se ha utilizado para el desarrollo de la aplicación web, aprovechando que en un mismo proyecto se puede programar en diferentes lenguajes (Javascript y Python).

- Anaconda Navigator

  Anaconda Navigator es una interfaz gráfica de usuario que permite administrar paquetes y entornos sin necesidad de escribir comandos Conda en una ventana de terminal. (2) Se ha creado un entorno virtual para la parte del backend den Python de la aplicación web, lo que permite instalar de forma eficiente las librerías específicas necesarias, como por ejemplo Pandas, Numpy o Flask.

- Firebase

  Firebase (11) es una plataforma para el desarrollo de aplicaciones web que pertenece a Google. Es una plataforma ubicada en la nube e integrada con Google Cloud Platform, que ofrece herramientas para facilitar el desarrollo del backend de las aplicaciones. En este caso, se ha utilizado el servicio de Firebase Authentication para permitir que los usuarios se identifiquen con su email y contraseña. Este servicio se encarga de gestionar de forma segura todo el proceso de autenticación, incluyendo el almacenamiento cifrado de las credenciales, la verificación de identidad y la gestión de sesiones. (12)

3.2. Bases de datos

- Firestore

  Firestore es una base de datos NoSQL flexible, escalable y en la nube, creada en la infraestructura de Google Cloud, para almacenar y sincronizar datos para el desarrollo tanto del lado del cliente como del servidor. (13) En esta base de datos se han creado dos colecciones, una para los usuarios (*users*) y otra para las predicciones (*predicciones*).

  Para interactuar con la base de datos, Firebase proporciona SDKs específicos para cada plataforma:

  - En la aplicación web se utiliza el SDK para JavaScript
  - En la aplicación móvil se usa un archivo de configuración

  El SDK de Firebase permite el acceso a sus servicios, como por ejemplo Firebase Authentication o Firestore.

  Se ha decidido utilizar una base de datos en la nube para tener una experiencia real trabajando con este tipo de tecnología. Actualmente existe un debate sobre el uso de bases de datos locales frente a soluciones en la nube, y esta era una buena oportunidad para acercarse a este debate desde una perspectiva práctica.

- Archivos CSV

  Para guardar las predicciones con las que se entrenan cada modelo se han utilizado archivos CSV locales por su simplicidad y facilidad de uso. Según se aumente la base de datos, se migrarán estas predicciones a una base de datos más robusta.

3.3. Lenguajes de programación

- Kotlin

  Kotlin (19) es un lenguaje de programación orientado a objetos que está diseñado para ser interoperable con Java (18). El código de la aplicación móvil está escrito en Kotlin, utilizando diferentes clases, como *Usuarios* o *Predicciones*, y actividades, como *LoginActivity* o *MainActivity*.

  Se ha utilizado Kotlin porque la página oficial de Android Studio incluye un tutorial de cómo aprender el lenguaje (6). Además, ses el lenguaje más utilizado para programar aplicaciones Android.

  En el proyecto se ha utilizado la versión 1.9.0

- JavaScript

  JavaScript es un lenguaje de programación ligero y compilado justo-a-tiempo (16). Se ha programado en JavaScript el frontend de la aplicación web, ya que la página oficial de Firebase proporciona una guía de cómo integrar ambos en una aplicación web. (1)

  En el proyecto se ha utilizado la versión 1.8.0

- Python

  Python (23) es un lenguaje de programación que hace hincapié en la legibilidad del código. Es el lenguaje más empleado según el índice TIOBE (24) y es uno de los más versátiles a la hora de programar modelos de Machine Learning. Se ha utilizado Python para el backend de la aplicación web y para el entrenamiento de diferentes modelos de Machine Learning.

  Para la comunicación entre el frontend (en JavaScript) y el backend he utilizado Flask (un framework para crear aplicaciones web en Python) y una API RESTful a través de la cual se gestionaron las solicitudes de entrenamiento del modelo de predicción.

  En el proyecto se ha utilizado la versión 2025.6.1

- HTML

  HyperText Markup Language (15) es un lenguaje de etiquetas de hipertexto que he utilizado en la aplicación web para estructurar el contenido. Para cada ventana se ha creado su correspondiente *.html*.

- XML

  Extensible Markup Language es un lenguaje que permite definir y almacenar datos de forma compartible. (29) En Android Studio, se utilizan los acrhivos *.xml* para definir los layouts donde se estructuran los componentes de la interfaz de usuario. Cada pantalla de la aplicación móvil está asociada a un archivo *.xml*.

- CSS

  CSS (8) es un lenguaje que describe cómo debe ser renderizado el elemento estructurado en la pantalla. Lo he utilizado para diseñar y dar estilo a la página web, asociando a cada *html* un archivo *.css*

Capítulo 4

Especificación de los requisitos

4.1. Actores

Tanto la aplicación móvil como la web cuenta con dos tipos de usuarios:

- **Médico**: personal médico registrado. Se registra desde la aplicación móvil y puede realizar diferentes funcionalidades. No tiene acceso a la aplicación web, mientras que en la aplicación móvil las funcionalidades más destacables son: realizar una predicción, ver su historial y importar y exportar varias predicciones a la vez.
- **Administrador**: actor destinado a la gestión de los Médicos y a entrenar modelos de Machine Learning. En la aplicación móvil puede acceder al historial de predicciones de los médicos. En la aplicación web puede seleccionar qué predicciones añadir al dataset y qué modelos entrenar. También puede introducir los datos de donantes para comparar los resultados que dan cada modelo y la fórmula de la tesis doctoral.

4.2. Casos de uso

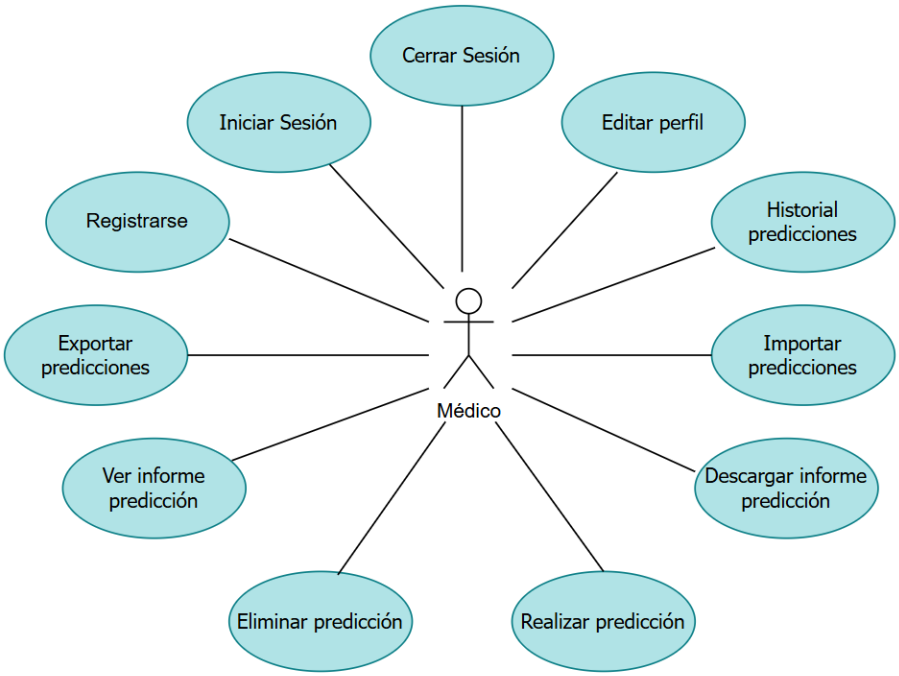


Figura 4.1: Diagrama de Médico en App Móvil

Figura 4.2: Diagrama de Administrador en App Móvil

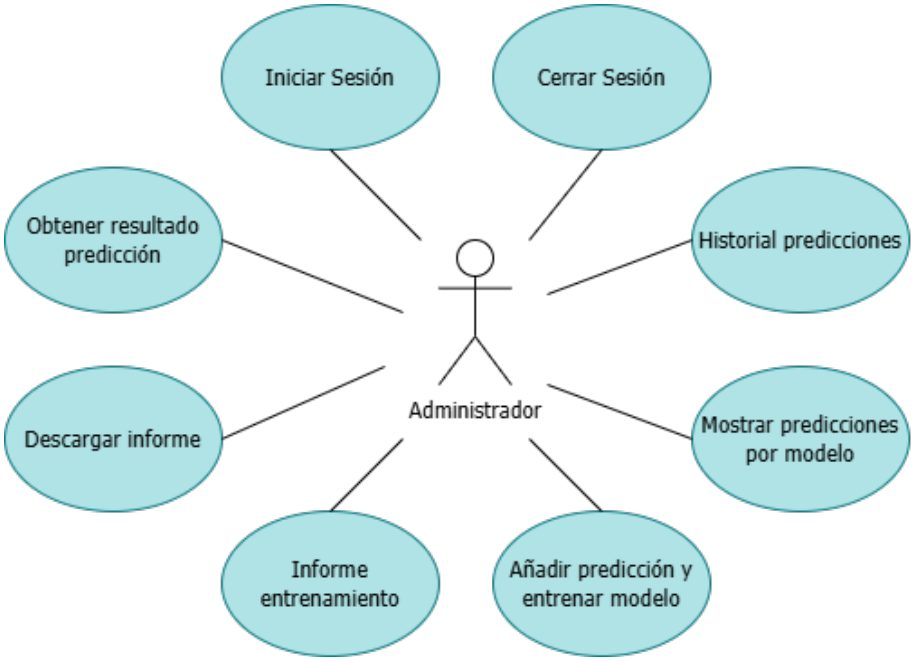


Figura 4.3: Diagrama de Administrador en App Web

4.2.1. Gestión de usuarios

ID	**ID01**
Nombre caso de uso	**Iniciar sesión**
Actores	Médico y Administrador
Descripción	El usuario accede introduciendo su nombre de usuario y contraseña.
Precondición	El usuario debe estar registrado
Secuencia normal	1. El usuario introduce su nombre de usuario y contraseña en el formulario 2. El usuario pulsa el botón Iniciar Sesión 3. El sistema comprueba los datos introducidos en la base de datos 4. Se muestra la página principal
Postcondición	Éxito: el usuario accede a la página principal Fallo: se muestra un mensaje de error indicando si el error está en el correo o en la contraseña y aparece de nuevo el formulario
Input	Nombre de usuario y contraseña
Output	Página principal
Excepciones	Si los datos introducidos no coinciden con ningún usuario de la base de datos, se muestra un mensaje de error y se vuelve al paso 1
Comentarios	El nombre de usuario es un correo electrónico

Tabla 4.1: Caso de uso: Iniciar sesión

ID	**ID02**
Nombre caso de uso	**Cerrar sesión**
Actores	Médico y Administrador
Descripción	El usuario pulsa el botón “Cerrar sesión” para volver a la página de Inicio de Sesión
Precondición	El usuario debe haber iniciado sesión
Secuencia normal	1. El usuario presiona el botón “Cerrar sesión” 2. Se muestra la página de Inicio de sesión
Postcondición	Se muestra la página para iniciar sesión
Input	-
Output	Página de Inicio de sesión
Excepciones	-
Comentarios	En la aplicación móvil se muestra un Toast de despedida

Tabla 4.2: Caso de uso: Cerrar sesión

ID	**ID03**
Nombre caso de uso	**Registro**
Actores	Médico y Administrador
Descripción	El usuario rellena un formulario con sus datos para acceder a la aplicación. Sus datos se guardan en la base de datos.
Precondición	El nombre de usuario no debe estar ya en la base de datos
Secuencia normal	1. El usuario pulsa el botón “No tengo cuenta” 2. Se abre la ventana con el formulario 3. El usuario rellena todos los campos con el formato adecuado 4. Presiona el botón Registrarse" 5. Se comprueba que los datos sean correctos 6. Se muestra un Toast con un mensaje de texto 7. Se muestra la página de Inicio de Sesión
Postcondición	Éxito: el usuario aparece en la base de datos y puede iniciar sesión con sus nuevas credenciales. Fallo: se muestra un mensaje de error indicando el tipo de este y se muestra el formulario de nuevo.
Input	Datos del usuario
Output	Toast con mensaje de éxito o error
Excepciones	-
Comentarios	Datos solicitados: Nombre, Apellidos, Fecha de nacimiento, Correo electrónico (nombre de usuario), Contraseña y Confirmación y Rol

Tabla 4.3: Caso de uso: Registro

ID	**ID04**
Nombre caso de uso	**Editar perfil**
Actores	Médico y Administrador
Descripción	Se cambian los datos del usuario.
Precondición	El usuario debe haber iniciado sesión.
Secuencia normal	1. Se presiona el botón “Editar perfil”. 2. Aparecen los datos del usuario en un formulario. 3. Se cambian los datos necesarios. 4. Se pulsa el botón Realizar cambios". 5. Se comprueba que los datos sean correctos. 6. Se actualiza la base de datos. 7. Aparece la Página principal.
Postcondición	Éxito: los datos se guardan en la base de datos y se muestra la Página principal. Fallo: se muestra un Toast de error y se muestra de nuevo el formulario.
Input	Datos a cambiar del usuario
Output	Página principal
Excepciones	-
Comentarios	Solo disponible en la aplicación móvil

Tabla 4.4: Caso de uso: Editar perfil

4.2.2. Módulo predicciones

ID	**ID05**
Nombre caso de uso	**Historial de predicciones**
Actores	Médico
Descripción	Se visualizan las predicciones realizadas por el médico
Precondición	El usuario debe haber iniciado sesión con rol de Médico
Secuencia normal	1. Se presiona el botón "Historial predicciones" 2. Aparece una tabla con las predicciones realizadas por el médico
Postcondición	-
Input	-
Output	Tabla con las predicciones
Excepciones	Si no hay predicciones asociadas al médico se muestra una tabla vacía
Comentarios	Se puede definir el orden de las predicciones con un spinner. Se puede eliminar una predicción o descargar el informe correspondiente

Tabla 4.5: Caso de uso: Historial de predicciones

ID	**ID06**
Nombre caso de uso	**Realizar predicción**
Actores	Médico
Descripción	Se rellena un formulario con los datos del donante y se obtiene el resultado
Precondición	El usuario debe haber iniciado sesión con rol de Médico
Secuencia normal	1. Se presiona el botón “Realizar predicción” 2. Aparece un formulario con las variables necesarias 3. Se presiona el botón Realizar predicción" 4. Se comprueba que todos los campos estén rellenados 5. Se calcula si el donante es válido o no y se redirige a la página correspondiente
Postcondición	Donante válido o no válido
Input	Datos del donante: Edad, Capnometría en el momento de la transferencia, Sexo, Recuperación del pulso en algún momento, Indicar si el donante ha fallecido por una causa cardiaca e Indicar si se ha realizado cardiocompresión al donante en algún momento
Output	Página correspondiente según si el donante es válido o no
Excepciones	-
Comentarios	-

Tabla 4.6: Caso de uso: Realizar predicción

ID	**ID07**
Nombre caso de uso	**Descargar informe predicción**
Actores	Médico
Descripción	Se muestra y se descarga un informe con los datos de la predicción
Precondición	El usuario debe haber iniciado sesión con rol de Médico. El usuario debe haber realizado una predicción o haber abierto su historial
Secuencia normal	1. Se presiona el botón "Descargar PDF" 2. Se comprueba que la versión de Android sea igual o superior a 10 3. Se crea el PDF 4. Se guarda el PDF en el dispositivo 5. Se muestra el PDF
Postcondición	-
Input	-
Output	Documento PDF
Excepciones	Si la versión Android no es correcta se muestra un Toast con el error
Comentarios	Se muestra el nombre del médico y la fecha, además de todos los datos de las variables del donante y el resultado final

Tabla 4.7: Caso de uso: Descargar informe predicción

ID	**ID08**
Nombre caso de uso	**Importar predicciones**
Actores	Médico
Descripción	Se selecciona un archivo CSV y se añaden las predicciones a la base de datos
Precondición	El usuario debe haber iniciado sesión con rol de Médico. El archivo debe tener formato CSV. El CSV debe tener el formato adecuado
Secuencia normal	1. Se presiona el botón "Importar predicciones" 2. Se solicita al usuario que seleccione el archivo 3. Se lee el archivo comprobando que tenga el formato correcto 4. Se actualiza la base de datos
Postcondición	Éxito: Base de datos ampliada Fallo: Toast con mensaje de error
Input	El formato del CSV debe ser: edad;capnometria;femenino;cardiomanual; recuperacionpulso;causacardiaca;valido. Todas son variables numéricas, y femenino, cardiomanual, recuperacionpulso, causacardiaca y valido son binarias
Output	-
Excepciones	-
Comentarios	-

Tabla 4.8: Caso de uso: Importar predicciones

ID	**ID09**
Nombre caso de uso	**Eliminar predicción**
Actores	Médico
Descripción	Se elimina la predicción indicada
Precondición	El usuario debe haber iniciado sesión con rol de Médico. El usuario debe haber navegado a su historial
Secuencia normal	1. Se presiona el icono de la papelera de la fila que se quiere eliminar 2. Se abre un diálogo de confirmación 3. Se actualiza la base de datos
Postcondición	Éxito: Base de datos actualizada Fallo: Toast con mensaje de error
Input	Objeto Predicción
Output	-
Excepciones	-
Comentarios	Se muestra un Toast con mensaje de éxito o fallo

Tabla 4.9: Caso de uso: Eliminar predicción

ID	**ID10**
Nombre caso de uso	**Exportar predicciones**
Actores	Médico
Descripción	Se crea un archivo CSV con las predicciones del médico
Precondición	El usuario debe haber iniciado sesión con rol de Médico. El usuario debe haber navegado a su historial
Secuencia normal	1. Se presiona el botón .ᴱˣportar" 2. Se crea un CSV en el que cada fila tiene los datos de cada predicción.
Postcondición	-
Input	Predicciones realizadas por el médico
Output	Archivo CSV
Excepciones	-
Comentarios	En la primera línea del CSV aparece el encabezado: Edad,Femenino,Capnometria,Causacardiaca, Cardiomanual,Recuperacionpulso,Valido, UIDmedico,Fecha

Tabla 4.10: Caso de uso: Exportar predicciones

4.2.3. Módulo administrador

ID	**ID11**
Nombre caso de uso	**Lista de médicos**
Actores	Administrador
Descripción	Se muestra una lista con todos los médicos registrados
Precondición	El usuario debe haber iniciado sesión con rol de Administrador
Secuencia normal	Al iniciar sesión se muestra en la página principal la lista de los médicos
Postcondición	-
Input	-
Output	Tabla con los datos de los médicos
Excepciones	-
Comentarios	Solo disponible en la aplicación móvil. Se muestran los apellidos, el correo electrónico y un icono para navegar al historial de predicciones de cada médico

Tabla 4.11: Caso de uso: Lista médicos

ID	**ID12**
Nombre caso de uso	**Lista de predicciones por médico**
Actores	Administrador
Descripción	Se visualizan las predicciones realizadas por el médico seleccionado
Precondición	El usuario debe haber iniciado sesión con rol de Administrador. El usuario debe haber presionado el icono de mostrar correspondiente a un médico
Secuencia normal	1. Se presiona el icono para mostrar las predicciones realizadas por un médico 2. Aparece una tabla con las predicciones realizadas por el médico
Postcondición	-
Input	Correo electrónico del médico seleccionado
Output	Tabla con las predicciones
Excepciones	Si no hay predicciones asociadas al médico se muestra una tabla vacía
Comentarios	Solo dispoible en la aplicación móvil. Se puede definir el orden de las predicciones con un spinner

Tabla 4.12: Caso de uso: Lista de predicciones asociadas a un médico

ID	**ID13**
Nombre caso de uso	**Historial predicciones (Administrador)**
Actores	Administrador
Descripción	Se visualizan las predicciones realizadas por todos los médicos
Precondición	El usuario debe haber iniciado sesión con rol de Administrador
Secuencia normal	1. Se inicia sesión y aparece en la página principal una tabla con todas las predicciones 2. Se filtran qué predicciones ver según el valor del spinner
Postcondición	El spinner contiene un valor que especifica qué predicciones ver
Input	-
Output	Tabla con las predicciones
Excepciones	Si no hay predicciones se muestra una tabla vacía
Comentarios	Solo disponible en la aplicación web

Tabla 4.13: Caso de uso: Lista de predicciones

ID	**ID14**
Nombre caso de uso	**Mostrar predicciones por modelo**
Actores	Administrador
Descripción	Se selecciona en el spinner el modelo y se muestran las predicciones qeu componen el dataset de entrerno de ese modelo
Precondición	El usuario debe haber iniciado sesión con rol de Administrador
Secuencia normal	1. Se selecciona un modelo en el spinner 2. Se muestra una tabla con las predicciones que componen el dataset de ese modelo
Postcondición	-
Input	Modelo del que se desea ver sus predicciones
Output	Tabla con predicciones que componen el dataset del modelo
Excepciones	-
Comentarios	Los modelos que se pueden seleccionar son: Árbol de decisión, Random Forest, Regresión Logística y Clustering

Tabla 4.14: Caso de uso: Mostrar predicciones por modelo

ID	**ID15**
Nombre caso de uso	**Añadir predicción y entrenar modelo**
Actores	Administrador
Descripción	Se selecciona las predicciones que añadir al dataset para entrenar el modelo definido en el spinner
Precondición	El usuario debe haber iniciado sesión con rol de Administrador
Secuencia normal	1. Se pulsa el botón *Añadir y entrenar* 2. Se muestra una tabla con las predicciones que se pueden añadir al modelo correspondiente 3. Se seleccionan las predicciones que se desean añadir 4. Se especifica el modelo en el spinner 5. Se pulsa el botón *Añadir* 6. Aparece un diálogo de confirmación 7. Se entrena el modelo correspondiente y se muestra un diálogo de advertencia 8. Se muestra un modal con los resultados tras entrenar el modelo
Postcondición	El spinner contiene un valor que especifica qué predicciones ver. Se muestra un modal con los resultados
Input	Modelo que se desea entrenar y predicciones nuevas
Output	Modelo entrenado e informe
Excepciones	Si no se selecciona ninguna predicción aparece mensaje informando de ello
Comentarios	-

Tabla 4.15: Caso de uso: Añadir predicciones al dataset

ID	**ID16**
Nombre caso de uso	**Informe entrenamiento**
Actores	Administrador
Descripción	Se muestra un modal con información sobre el modelo entrenado
Precondición	El usuario debe haber iniciado sesión con rol de Administrador. El usuario debe haber entrenado un modelo.
Secuencia normal	1. Aparece un modal con los resultados del entrenamiento del modelo correspondiente
Postcondición	-
Input	-
Output	Informe con los resultado del modelo entrenado
Excepciones	-
Comentarios	En el modal hay un botón para cerrar el mismo.

Tabla 4.16: Caso de uso: Informe entrenamiento

ID	**ID17**
Nombre caso de uso	**Descargar informe**
Actores	Administrador
Descripción	Se descarga un informe con la información sobre el modelo entrenado
Precondición	El usuario debe haber iniciado sesión con rol de Administrador. El usuario debe haber entrenado un modelo.
Secuencia normal	1. Se pulsa el botón de *Descargar .txt* del modal 2. Se descarga en la máquina el archivo
Postcondición	Archivo descargado en la carpeta *Descargas* de la máquina
Input	-
Output	Informe descargado con los resultado del modelo entrenado
Excepciones	-
Comentarios	-

Tabla 4.17: Caso de uso: Descargar informe

ID	**ID18**
Nombre caso de uso	**Obtener resultado predicción**
Actores	Administrador
Descripción	Se rellena un formulario con los datos del donante y se obtienen los resultados que dan cada modelo y la fórmula de la tesis doctoral
Precondición	El usuario debe haber iniciado sesión con rol de Administrador.
Secuencia normal	1. Se pulsa el botón *Predecir* de la página principal 2. Se navega a un formulario que solicita los datos del donante y los modelos de los que se quiere obtener el resultado 3. Se pulsa el botón *Realizar predicción* 4. Se navega a una página donde aparece el resultado de cada modelo seleccionado y de la fórmula, además de los datos del donante.
Postcondición	-
Input	Datos del donante y modelos
Output	Resultados de los modelos y de la fórmula
Excepciones	-
Comentarios	Si no se selecciona ningún modelo sale el resultado de la fórmula

Tabla 4.18: Caso de uso: Obtener resultados predicción

Capítulo 5

Arquitectura

5.1. Introducción a la arquitectura

La arquitectura consiste en un modelo cliente-servidor con tres capas: el backend, la aplicación web y la aplicación móvil.

El backend está desarrollado en Python y utiliza el framework Flask, que se pone en funcionamiento en el puerto 5000. Este contiene una API RESTful con dos rutas: una para entrenar modelos y otra para predecir. La primera recibe desde el frontend (la aplicación web) un conjunto de predicciones junto con el nombre del modelo que se desea entrenar. La segunda recibe, también del frontend, una lista con los modelos de los que se desea un resultado y los datos de la predicción. Para facilitar el acceso y la escritura de las predicciones en el dataset de cada modelo se han utilizado una serie de archivos CSV locales. Así, cada modelo (Árbol de decisión, Random Forest, Regresión logística y Clustering) tiene su propio flujo de entrenamiento y predicción.

Por otro lado, la aplicación web está gestionada por un servidor live-server que se ejecuta en el puerto 8080, lo que permite una visualización en tiempo real de los resultados y facilita la interacción con la API. Desde esta capa, la aplicación web también se conecta al backend mediante Firebase, en concreto para mostrar el contenido de las tablas de predicciones.

La tercera capa (la aplicación móvil) se ejecuta en el simulador de Android Studio y se conecta al backend mediante Firebase, lo que permite que los datos sean enviados y recibidos de manera eficiente entre la app y el backend.

5.2. Modelo C4

El modelo C4 permite representar un sistema de software con un conjunto de diagramas que describen cada uno un nivel diferente de detalle. (10) Así, podemos diseñar los siguientes diagramas:

- Contexto
- Contenedores
- Componentes
- Código

5.2.1. Diagrama de Contexto

El diagrama de contexto ofrece una vista global de todo el sistema, mostrando una descripción general del sistema: qué hace, quién lo utiliza y con qué otros sistemas externos interactúan. (10)

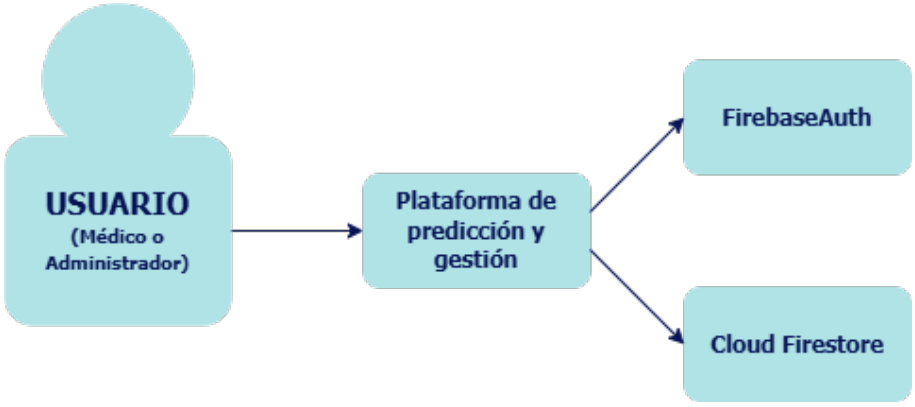


Figura 5.1: Diagrama de Contexto

5.2.2. Diagrama de Contenedores

El diagrama de contenedores muestra a alto nivel la arquitectura de software y cómo se distribuyen las responsabilidades entre los distintos componentes. También muestra la tecnología y cómo se comunican los contenedores entre sí. (10)

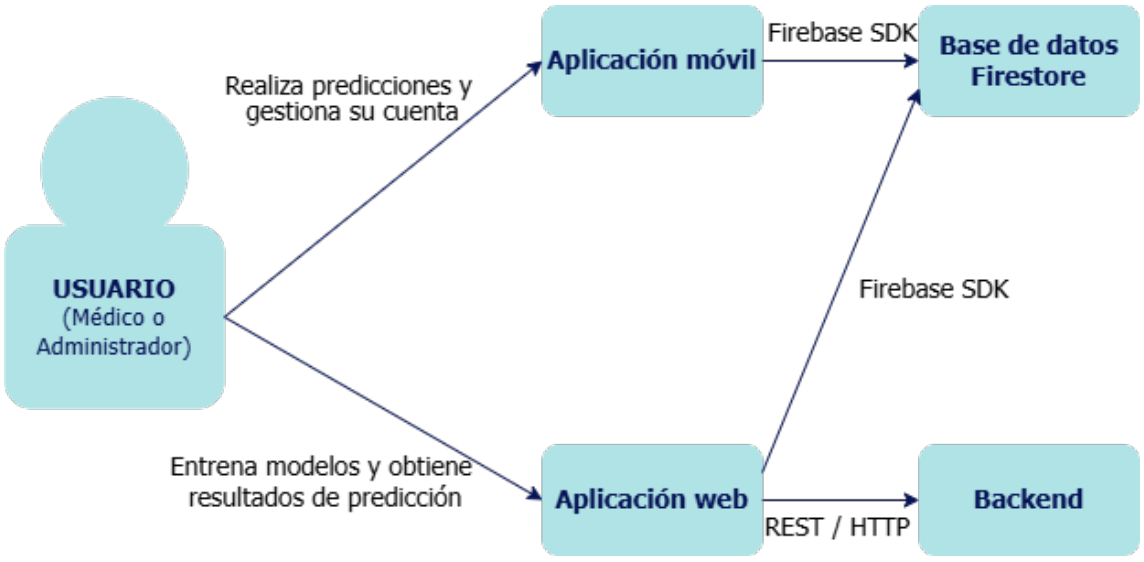


Figura 5.2: Diagrama de Contenedores

5.2.3. Diagrama de Componentes

El diagrama de componentes muestra cómo un contenedor está formado por una serie de componentes, cuáles son cada uno de ellos, sus responsabilidades y los detalles de tecnología o implementación. (10)

En este caso, se puede ver cómo está formado el contenedor del *Backend*, ya que es el más complejo y el que contiene la comunicación entre la aplicación, la base de datos y los modelos de *machine learning*.

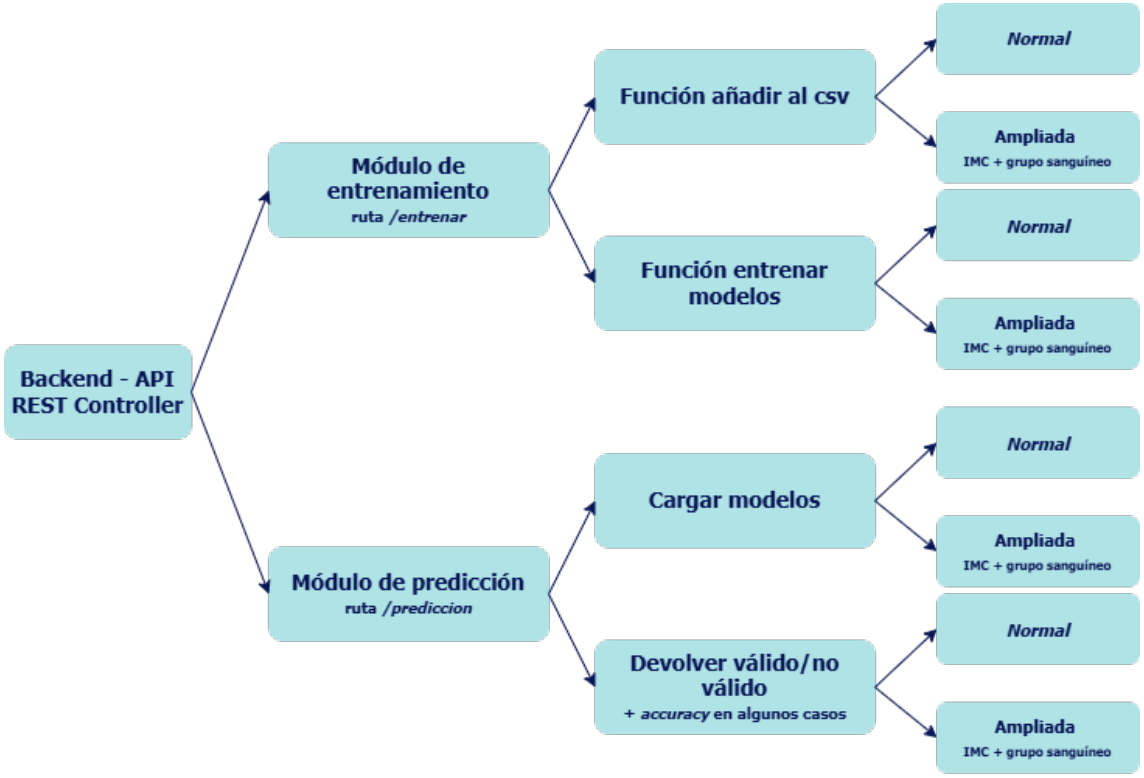


Figura 5.3: Diagrama de Componentes

5.2.4. Diagrama de Código

Por último, en el diagrama de código se puede ampliar cada componente para mostrar cómo se implementa como código.(10) En este caso vamos a ver cómo se esctructura el *backend* y las funciones principales.

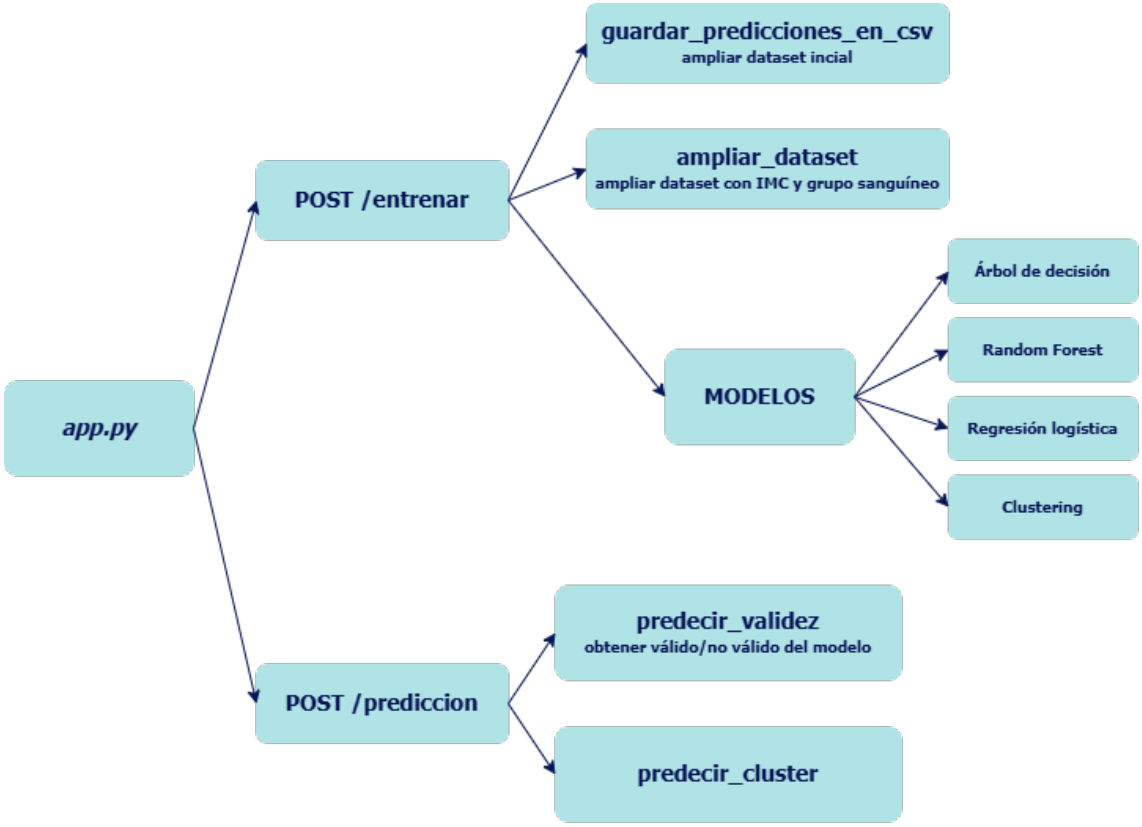


Figura 5.4: Diagrama de Código

Capítulo 6

Modelo de Datos

6.1. Introducción a los modelos de datos

En el proyecto se ha utilizado Cloud Firestore, una base de datos NoSQL de documentos basada en la nube. En esta se almacenan los datos simples como colecciones de documentos, que son muy similares a los JSON. Para comenzar a trabajar con esta base de datos, se creó una cuenta en Firebase y, a través de su consola web, se configuró un nuevo proyecto llamado *TFGPrediccion*.

Además de Firestore, también se ha empleado Firebase Authentication para gestionar el registro e inicio de sesión de los usuarios mediante correo electrónico y contraseña. Este servicio incluye una serie de funciones (como *createUserWithEmailAndPassword* o *signInWithEmailAndPassword*) que permite autenticar a los usuarios de forma segura, sin tener que implementar manualmente todo el sistema de registro y login.

Para integrar Firestore con la aplicación web y móvil se siguieron los pasos proporcionados por la consola de Firebase. Para la versión web, se utiliza un fragmento de código JavaScript proporcionado por la consola de Firebase que contiene la configuración del proyecto en una constante (*const firebaseConfig*). Este fragmento incluye las claves necesarias para establecer la conexión con el proyecto TFGPrediccion desde el entorno web. En el caso de la aplicación móvil, la configuración se realiza mediante el archivo *google-services.json*, también generado desde la consola de Firebase. Ambos métodos permiten que las dos plataformas accedan a la misma base de datos en la nube de forma sincronizada.

La base de datos consta de dos colecciones: *users* y *predicciones*. La primera contiene información básica que se proporciona a través del formulario de registro de los usuarios, así como contadores relacionados con las predicciones que se utilizan para la gráfica del menú principal. Por otro lado, la segunda colección contiene las variables necesarias para determinar si un donante es válido o no, además de datos sobre el médico que realiza la predicción.

6.2. Colección *users*

Esta colección es necesaria para poder acceder tanto a la aplicación móvil como a la web. A través del formulario de registro se crea automaticamente un nuevo

documento en la colección, el cual puede ser modificado a través de la actividad de *Editar perfil*. Además, los contadores se actualizan cada vez que el médico realiza o importa una predicción.

Los campos que forman esta colección son los siguientes.

Campo	Tipo	Descripción
email	string	Nombre de usuario único. Firebase Authentication valida que el email tenga un formato correcto y que no se repita.
lastname	string	Apellido/s del usuario.
name	string	Nombre del usuario.
birthdate	string	Fecha de nacimiento del usuario.
role	string	Puede ser Médico o Administrador.
uid	string	Identificador único generado por Firebase Authentication.
numeroPredicciones	number	Contador del número de predicciones que ha realizado el usuario.
predicciones_no_validas	number	Contador del número de predicciones no válidas que ha realizado el usuario.
predicciones_validas	number	Contador del número de predicciones válidas que ha realizado el usuario.

Tabla 6.1: Campos de la colección *users* en Firestore

6.3. Colección *predicciones*

Esta colección es esencial para tener un histórico de las predicciones realizadas por cada médico que, posteriormente, se utilizarán para entrenar diferentes modelos de Machine Learning. Cada documento contiene las variables requeridas por la fórmula propuesta en la tesis doctoral de Carlos Rubio, junto con la fecha de la predicción, el uid del médico y dos listas con los modelos de Machine Learning en las que la predicción ha sido incluida o no, respectivamente.

Los campos que forman esta colección son los siguientes.

Campo	Tipo	Descripción
capnometria	string	Valor de la capnometría en el momento de la transferencia. Es un número natural.
cardio_manual	string	Valor booleano (si/no) que indica si se le ha realizado al donante una cardiocompresión extrahospitalaria mecánica.
causa_cardiaca	string	Valor booleano (si/no) que indica si el donante ha fallecido por una causa cardiaca.
edad	string	Edad del donante. Número natural.
femenino	string	Valor booleano (si/no) que indica si el donante es de sexo femenino.
rec_pulso	string	Valor booleano (si/no) que indica si el donante recuperó el pulso en algún momento.
uid_medico	string	Identificador del médico que realiza la predicción.
valido	number	Valor booleano (si/no) que indica si la predicción ha resultado válida o no según la fórmula.
modelos	lista	Lista con los modelos de Machine Learning en los que la predicción está incluida.
no_modelos	lista	Lista con los modelos de Machine Learning en los que la predicción no está incluida.
fecha	marca de tiempo	Fecha de la predicción.

Tabla 6.2: Campos de la colección *predicciones* en Firestore

6.4. Firestore Authentication

Cada vez que un nuevo usuario se registra, Firebase Authentication se encarga de validar el formato del correo electrónico, comprobar que sea único, asignar un UID y crear un nuevo registro en la consola de Firebase. Así, podemos encontrar una tabla con información sobre los usuarios que incluye el email, la fecha de creación de la cuenta, la última fecha de acceso y el UID único asignado a cada usuario. En cuanto a la contraseña, Firebase Authentication comprueba que no sea una contraseña débil

y la encripta para garantizar la seguridad de la cuenta. Aunque la contraseña no es accesible desde la consola por cuestiones de seguridad, los usuarios sí pueden modificarla en cualquier momento a través de la actividad de *Editar perfil*.

6.5. Dataset de los modelos de Machine Learning

Para el almacenamiento y gestión del dataset utilizado en el entrenamiento de los modelos de Machine Learning, se ha optado por el uso de archivos CSV de manera local. Esta decisión se basa en el tamaño del dataset inicial (69 registros), la simplicidad y la flexibilidad que ofrecen los archivos CSV. Cada vez que se añade una nueva predicción, los datos se escriben en el correspondiente archivo CSV del modelo, permitiendo que la base de datos se actualice de manera dinámica. De la misma forma, cuando se entrenan los modelos, los datos se leen directamente desde su correspondiente archivo, asegurando que el conjunto de datos esté siempre disponible y actualizado.

Cada modelo tiene su correspondiente archivo CSV en dos versiones: una con las seis variables de la fórmula de la tesis doctoral de Carlos, y otra añadiendo el IMC y el grupo sanguíneo (además de la variable objetivo). Esto se ha hecho así para poder comparar los resultados a la hora de realizar predicciones entre la versión original y la ampliada.

A continuación, se indican los nombres de los archivos CSV creados:

- *predicciones_arbol_decision.csv*
- *predicciones_arbol_decision_ampliado.csv*
- *predicciones_random_forest.csv*
- *predicciones_random_forest_ampliado.csv*
- *predicciones_reg_log.csv*
- *predicciones_reg_log_ampliada.csv*
- *predicciones_cluster.csv*
- *predicciones_cluster_ampliado.csv*

Capítulo 7

Modelos de Machine Learning empleados

Esta parte del TFG consiste en explicar todo lo relativo a los modelos de Machine Learning empleados. La idea es que los Administradores puedan utilizar las predicciones realizadas por los Médicos en la aplicación móvil para entrenar diferentes modelos. Además, pueden comprobar qué resultado proporciona cada uno de los modelos y compararlo con el resultado de la fórmula de la tesis doctoral. De esta forma se podrá determinar, en un futuro, con qué tipo de donantes cada modelo funciona mejor y se podrán tomar decisiones sobre cuál emplear en función del contexto clínico.

Es importante destacar que, para cada modelo, se han empleado dos datasets:

- dataset = {*edad, capnometria, causa_cardiaca, cardiocompresion, recuperacion_pulso, sexo*}
- datasetAmpliado = {*edad, capnometria, causa_cardiaca, cardiocompresion, recuperacion_pulso, sexo, IMC, grupo_sanguineo*}

Cada variable se puede clasificar como se ve en la tabla 7.1.

Variable	Tipo	Posibles valores
capnometria	numérica	Número natural
cardio_manual	categórica binaria	1/0
causa_cardiaca	categórica binaria	1/0
edad	numérica	Número natural
sexo	categórica binaria	1/0
rec_pulso	categórica binaria	1/0
IMC	numérica continua	Valores decimales mayores que 0
grupo_sanguineo	categórica	A/B/AB/0
valido	categórica binaria	1/0

Tabla 7.1: Variables

Las dos variables nuevas (IMC y grupo sanguíneo) fueron seleccionadas a partir de una base de datos más extensa proporcionada por Carlos Rubio Chacón, autor de la tesis doctoral (36). Esta base de datos fue la que él empleó para identificar las seis variables principales del dataset inicial. De entre todas las variables posibles, se eligieron estas por su carácter independiente y su potencial para aportar información adicional. El objetivo de incorporarlas al *datasetAmpliado* es descubrir si contribuyen significativamente al resultado de la predicción en determinados perfiles de donantes que pudieran pasar desapercibidos mediante análisis tradicionales, así como facilitar una agrupación de pacientes en función de sus características comunes.

La razón por la que se usan dos conjuntos de datos para cada modelo (el inicial y el ampliado) es poder tener dos vías en el futuro. Si se pueden conocer los valores del IMC y del grupo sanguíneo, los modelos ampliados contendrán mucha más información a la que no se ha llegado aún. Sin embargo, si estos valores no se pueden conocer, los modelos ampliados se entrenarán con muchos valores *Desconocidos* y la utilidad de estos disminuirá según la base de datos vaya aumentando.

Para seleccionar los modelos adecuados se leyeron una serie de proyectos y webs de casos reales dentro del ámbito de la medicina. Entre otros, la herramienta UK-DTOP (37), un informe sobre Regresión logística binaria simple (39) y otro sobre aplicaciones de aprendizaje automático en salud(41) sirvieron de mucha ayuda para decidir que los tres modelos supervisados inciciales serían *Árbol de Decisión*, *Random Forest* y *Regresión Logística*. Estos modelos se adaptan bien al proyecto ya que funcionan bien con variables tanto categóricas como numéricas y son fáciles de interpretar. Por otro lado, se incluyó un modelo no supervisado basado en *Clustering*, con el objetivo de identificar posibles agrupaciones que puedan revelar patrones ocultos o perfiles de donantes nuevos.

A continuación, se describe el proceso de entrenamiento de cada modelo y se analizan sus resultados. Estos se comparan con el resultado de la fórmula de la tesis, que es la siguiente:

fórmula

$$7{,}5 - 0{,}096 \cdot \text{edad} - 1{,}013 \cdot \text{fem} + 0{,}08 \cdot \text{capnometria}$$
$$-0{,}517 \cdot \text{causa\_cardiaca} - 2{,}616 \cdot \text{cardio\_man} + 2{,}943 \cdot \text{rec}$$

Si el resultado de esta fórmula es mayor o igual que 4.366, entonces la predicción es válida. En caso contrario, sería inválido.

Para todos los modelos la variable objetivo es *Válido*, que es una variable categórica binaria que puede tomar el valor 1 o 0.

7.1. Árbol de Decisión

Un árbol de decisión en Machine Learning es una estructura de árbol donde un nodo interno representa una característica, la rama es una regla de decisión, cada nodo u hoja simboliza el resultado y el nodo superior es el nodo raíz. Este tipo de árbol se conoce como árbol de clasificación, donde cada ramificación contiene un conjunto de atributos o reglas de clasificación asociadas a una etiqueta de clase específica que se halla al final de la ramificación. El árbol de decisión clasificatorio aprende a particionar en función del valor del atributo, este se divide de manera recursiva buscando un mínimo local de acuerdo a medidas como la entropía en base a un conjunto de datos masivo de entrenamiento. (32)

Se ha decidido utilizar un Árbol de decisión ya que permite visualizar el árbol y ver en qué ramas se va descomponiendo. También proporciona el peso que ha estimado para cada variable, aspecto que resulta muy interesante a la hora de poder compararlo directamente con los pesos que asocia la fórmula de la tesis.

7.1.1. Implementación

El primer paso es dividir las predicciones entre entrenamiento y test, correspondiendo a cada conjunto un 70 % y un 30 % respectivamente. Esta división se hace con la función *train_test_split* de la biblioteca *scikit-learn* (25), una de las bibliotecas más populares en Python para aprendizaje automático. La llamada a esta función se realiza tal que:

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y),

siendo cada variable:

- *X_train*: predicciones para entrenar el modelo
- *X_test*: predicciones para testear el modelo
- *y_train*: valores objetivo correspondientes a X_train
- *y_test*: valores objetivo correspondientes a X_test

Y cada argumento:

- *X*: conjunto completo con los valores de cada predicción
- *y*: conjunto completo con los valores a predecir (1 o 0)

- *test_size*: tamaño del conjunto de testeo (0.3 en este caso)
- *random_state*: fija una semilla para que cada vez que se ejecute el código se obtenga la misma partición
- *stratify*: garantiza que la proporción de clases en y se mantenga igual tanto en el conjunto de entrenamiento como en el de prueba. Al tener una base de datos pequeña viene bien para que se tengan en cuenta tanto las predicciones válidas como las no válidas.

Tras esta divisón, se crea una instancia del modelo con *model = DecisionTreeClassifier()* (también de la biblioteca *scikit-learn*). A esta función se le llama con el argumento *random_state=42* de nuevo para asegurar que el modelo se construya de forma reproducible. A continuación, se entrena el modelo con *model.fit(X_train, y_train)*.

El siguiente paso es evualar el modelo entrenado con el conjunto de predicciones para el testeo, que se hace llamando a la función: *model.predict(X_test)*. Una vez evaluado, se guarda el modelo entrenado utilizando *joblib.dump(model, 'modelo_arbol_decision.pkl')*, para así poder utilizarlo cuando el usuario quiera realizar una predicción y comparar los resultados entre modelos. *joblib* es una biblioteca de Python especializada en guardar y cargar objetos grandes y complejos, perfecto para guardar modelos entrenados (17).

Para terminar, se visualiza y se guarda el árbol utilizando *plot_tree*, también de la biblioteca *sklearn.tree*, y se genera el archivo *info_arbol_decision.txt* con la siguiente información:

- **Importancia de las características:** muestra qué variables fueron más relevantes para tomar decisiones en el árbol. Se obtiene con *feature_importances_*. Estos sirven para comparar con los pesos que se asignaron a cada variable en la fórmula de la tesis y para ver si las dos variables añadidas al dataset inicial muestran que tienen significancia.
- **Estructura del árbol:** representa cómo se organizan los nodos y las ramas de decisión en formato textual. Se obtiene con *export_text(model, feature_names)*
- **Matriz de confusión:** tabla que muestra cuántas predicciones se clasificaron correctamente o incorrectamente en cada clase. Se calcula con *confusion_matrix(y_test, y_pred)*.
- **Reporte de clasificación:** incluye métricas como precisión, recall, F1-score y soporte para cada clase. Se genera con *classification_report(y_test, y_pred)*
- **Valor de accuracy:** proporción total de predicciones correctas. Se obtiene con *accuracy_score(y_test, y_pred)*

7.2. Random Forest

Random Forest es una técnica de regresión que combina múltiples algoritmos de árboles de decisión para clasificar o predecir el valor de una variable. Consiste en construir un número de árboles de regresión a partir de diferentes subconjuntos del conjunto de datos original, generados mediante una técnica de muestreo con reemplazo (bagging). Cada árbol se construye utilizando una selección aleatoria de

características para dividir los datos, lo que reduce la correlación entre árboles y, por tanto, el error de generalización. Además, Random Forest no poda sus árboles, lo que los hace computacionalmente ligeros. También permite medir la importancia relativa de cada característica en la predicción, lo cual es útil en contextos con alta dimensionalidad de datos. (40)

Se ha decidido utilizar un Random Forest ya que, al proporcionar una predicción promediando entre todos los árboles, la respuesta será más precisa y estable. Además, al igual que el Árbol de decisión, asocia una importancia a cada variable que se puede comparar directamente con los pesos asociados a cada variable de la fórmula de la tesis.

7.2.1. Implementación

Se siguen los mismos pasos que para el Árbol de Decisión:

1. Se dividen las predicciones entre entrenamiento y test (70 % y 30 % respectivamente)
2. Se crea una instancia del modelo con

   model = RandomForestClassifier(random_state=42, class_weight=balanced)

   Este es un objeto que proviene de la biblioteca *sklearn.ensemble*. Se establece el parámetro *class_weight* en *balanced* para que el modelo ajuste automáticamente los pesos de las clases, de modo que las clases minoritarias reciban más atención durante el proceso de entrenamiento. Por ejemplo, si hubiese muchas más predicciones *válidas* que *no válidas* en los datos de entrenamiento, el modelo no aprendería correctamente ya que tendería a predecir mayoritariamente la clase con más ejemplos.
3. Se entrena el modelo con *model.fit(X_train, y_train)*
4. Se evalúa el modelo con *model.predict(X_test)*
5. Se guarda el modelo con *joblib.dump(model, modelo_random_forest.pkl')*
6. Se genera el archivo *info_random_forest.txt* con la misma información que los árboles de decisión, quitando la estructura del árbol que ocuparía mucho espacio.

7.3. Regresión Logística Binaria

La Regresión Logística Binaria es un modelo que se usa cuando se desea conocer la relación entre una variable dependiente cualitativa dicotómica y una o más variables independientes o explicativas, que pueden ser cualitativas y/o cuantitativas, con el objetivo de obtener una estimación ajustada de la probabilidad de ocurrencia de un evento a partir de una o más variables independientes. (33)

Se ha elegido este modelo ya que se ajusta perfectamente a los tipos de las variables. Para empezar, se tiene una variable objetivo que es cualitativa dicotómica,

que es *Válido*, ya que solo puede tomar el valor 1 o 0. Por otro lado, este modelo permite que el resto de las variables sean tanto cualitativas como cuantitativas, lo que se ajusta perfectamente al dataset.

7.3.1. Implementación

Para empezar, en la versión del dataset ampliado, se modifican los valores del IMC y grupo sanguíneo en el caso de que sean NaN. Para el IMC, se cambia a -1 ya que el modelo de regresión logística en scikit-learn no acepta estos valores. Se ha elegido el valor -1 ya que está fuera de rango y permite que el modelo aprenda que esa entrada representa un caso especial. Para el grupo sanguíneo, se ha cambiado a *Desconocido* para que el modelo lo trate como una categoría más durante el entrenamiento. La razón de marcar estos valores como *desconocidos* es que, al tener pocos registros, no podemos permitirnos eliminar los que estén incompletos.

Una vez se tienen ya los datos desconocidos preparados, se aplica *one-hot-encoding* para el grupo saguineo. Esta estrategia consiste en crear una columna binaria (que solo puede contener los valores 0 o 1) para cada valor único que exista en la variable categórica que se está codificando, y marcar con un 1 la columna correspondiente al valor presente en cada registro, dejando las demás columnas con un valor de 0. (21) Así, se crean las siguiente categorías para el grupo sanguíneo:

- grupo_sanguineo_A
- grupo_sanguineo_0
- grupo_sanguineo_B
- grupo_sanguineo_AB

A continuación, se divide el conjunto en entrenamiento y test, al igual que para el Árbol de decisión y el Random forest, con *train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)*. Después, se escalan las variables numéricas (IMC, edad y capnometría) para asegurar que todas las variables numéricas estén en la misma escala, lo que ayuda a que el modelo de la misma importancia a todas las variables. Esto se consigue aplicando primero *fit_transform* al conjunto de entrenamiento, que calcula la media y desviación estándar y luego transforma los datos. Posteriormente, se usa *transform* sobre el conjunto de test para aplicar esa misma escala, sin recalcular los parámetros, evitando así introducir información del test en el entrenamiento. (28)

El siguiente paso es crear una instancia del modelo con: *model = LogisticRegression(max_iter=1000, class_weight=balanced)*, que es un objeto de la biblioteca *sklearn.linear_model*. El parámetro max_iter establece el número máximo de iteraciones necesarias para que los solucionadores converjan. (20)

Después, se entrena el modelo con *model.fit(X_train, y_train)* y se guarda con *joblib.dump(model, modelo_regresion_logistica_ampliado.pkl')*.

Por último, con `y_pred = model.predict(X_test)` se realizan predicciones sobre el conjunto de predicciones de testeo utilizando el modelo entrenado. Luego, estas predicciones se comparan con los valores reales para evaluar los resultados y se escriben en el informe *info_regresion_logistica.txt* métricas como la matriz de

confusión, el informe de clasificación y los coeficientes del modelo. Esto último sirve para identificar qué variables tienen mayor influencia en las predicciones del modelo.

7.4. Clustering

El agrupamiento en clústeres es una técnica de aprendizaje automático no supervisado diseñada para agrupar ejemplos sin etiquetar en función de su similitud entre sí. (30)

Aplicar este modelo al proyecto resulta muy interesante ya que permite visualizar y agrupar a los donantes en grupos según sus características. Esto permitiría crear perfiles de donantes y, en un futuro, permitiría predecir la validez de la donación según sus características.

Antes de aplicar la clusterización, se implementó el Algoritmo del Codo para determinar cuál es el número óptimo de clusters en el que se deben dividir las predicciones.(35) Este algoritmo utiliza a su vez el algoritmo *K-Prototypes*, que encaja perfectamente con las variables ya que es un algoritmo que se aplica cuando hay variables tanto numéricas como categóricas, y la mayoría de las columnas de los datos son categóricas. Este algoritmo asigna un peso a cada característica según su tipo y utiliza esto para calcular la distancia entre los puntos de datos. (34) Las gráficas que proporciona este algoritmo son las mostradas en la figura 7.1 y 7.2 (para la base de datos inicial). Con estas gráficas determinamos el número óptimo de clústers en 4.

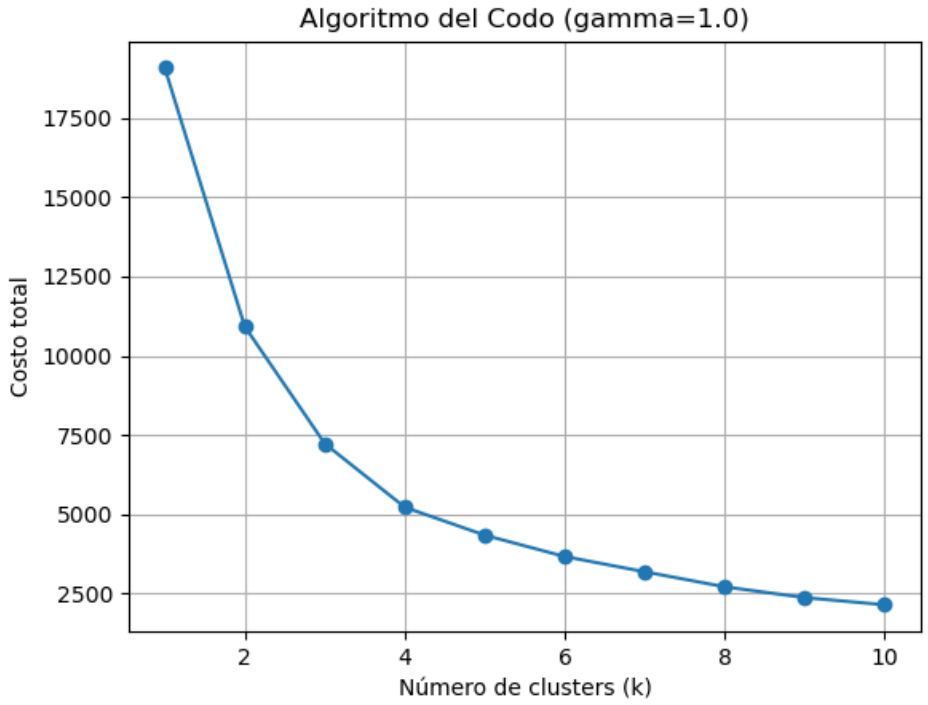


Figura 7.1: Gráfica Algoritmo del Codo

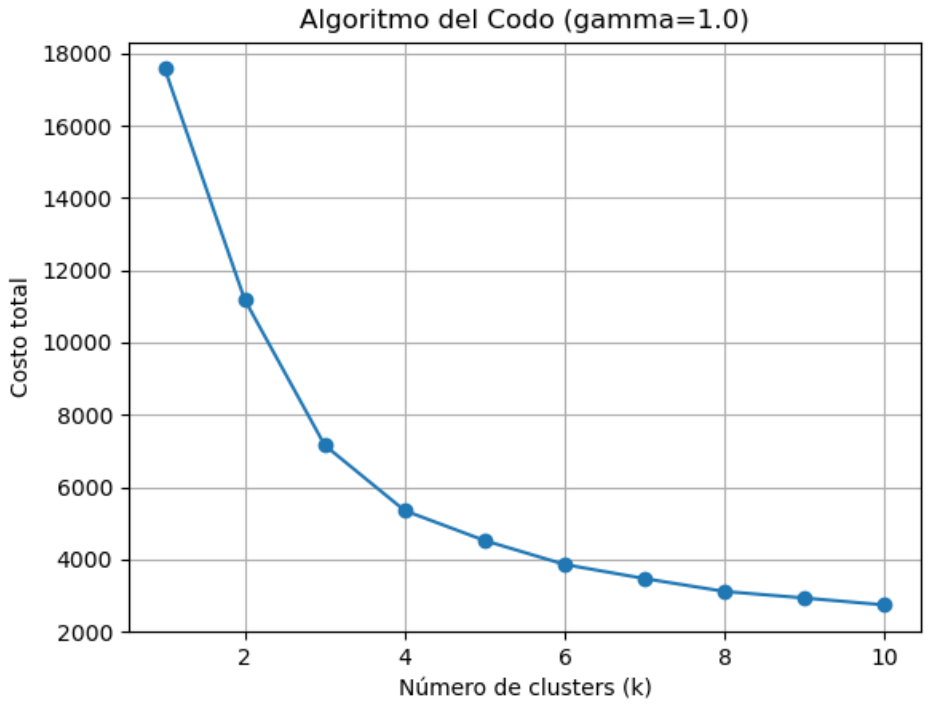


Figura 7.2: Gráfica Algoritmo del Codo (ampliado)

El siguiente paso después de saber en cúanto clústers se dividirán las predicciones es entrenar el modelo de la siguiente forma: *kp = KPrototypes(n_clusters=4, init='Huang', random_state=None, verbose=1) clusters = kp.fit_predict(X, categorical=cat_cols)*

Esto se ejecuta 10 veces para conseguir el modelo de clustering que genera menor costo de desimilitud. Después, en el archivo *info_clustering.txt* se guardan los resultados del modelo y con *pickle* (que es un módulo de Python para guardar el estado de objetos) se guarda el modelo haciendo *pickle.dump()*.

7.5. Resultados y Análisis

En esta sección se muestran los resultados obtenidos para los diferentes modelos habiendo sido entrenados con los 69 registros de la base de datos inicial.

7.5.1. Árbol de Decisión, Random Forest y Regresión Logística

info_arbol_decision.txt

```
INFORME ARBOL DE DECISION
Importancia de las caracteristicas:
edad: 0.3799
capnometria: 0.5515
femenino: 0.0000
causa_cardiaca: 0.0685
cardio_manual: 0.0000
rec_pulso: 0.0000

Estructura del arbol:
|--- capnometria <= 16.50
|   |--- causa_cardiaca <= 0.50
|   |   |--- class: 0
|   |--- causa_cardiaca >  0.50
|   |   |--- edad <= 52.50
|   |   |   |--- class: 1
|   |   |--- edad >  52.50
|   |   |   |--- class: 0
|--- capnometria >  16.50
|   |--- edad <= 54.00
|   |   |--- edad <= 45.50
|   |   |   |--- capnometria <= 43.00
|   |   |   |   |--- edad <= 19.00
|   |   |   |   |   |--- causa_cardiaca <= 0.50
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- causa_cardiaca >  0.50
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- edad >  19.00
|   |   |   |   |   |--- class: 1
|   |   |   |--- capnometria >  43.00
|   |   |   |   |--- class: 0
|   |   |--- edad >  45.50
|   |   |   |--- edad <= 51.50
|   |   |   |   |--- capnometria <= 28.00
|   |   |   |   |   |--- capnometria <= 24.50
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- capnometria >  24.50
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |--- capnometria >  28.00
|   |   |   |   |   |--- class: 0
|   |   |   |--- edad >  51.50
|   |   |   |   |--- capnometria <= 26.50
|   |   |   |   |   |--- capnometria <= 21.00
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- capnometria >  21.00
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- capnometria >  26.50
|   |   |   |   |   |--- class: 1
```

info_arbol_decision.txt

```
|   |--- edad >  54.00
|   |   |--- capnometria <= 22.50
|   |   |   |--- class: 1
|   |   |--- capnometria >  22.50
|   |   |   |--- class: 0


--- Evaluacion del modelo ---
Matriz de confusion:
[[12  1]
 [ 3  5]]

Reporte de clasificacion:
              precision    recall  f1-score   support

           0       0.80      0.92      0.86        13
           1       0.83      0.62      0.71         8

    accuracy                           0.81        21
   macro avg       0.82      0.77      0.79        21
weighted avg       0.81      0.81      0.80        21

Accuracy: 0.8095
```

info_arbol_decision_ampliado.txt

```
INFORME ARBOL DE DECISION AMPLIADO
Importancia de las caracteristicas:
edad: 0.3135
capnometria: 0.3713
femenino: 0.0000
causa_cardiaca: 0.0451
cardio_manual: 0.0000
rec_pulso: 0.0698
IMC: 0.0778
grupo_sanguineo_0: 0.0593
grupo_sanguineo_A: 0.0632
grupo_sanguineo_AB: 0.0000
grupo_sanguineo_B: 0.0000
```

info_arbol_decision_ampliado.txt

```
Estructura del arbol:
|--- capnometria <= 16.50
|   |--- rec_pulso <= 0.50
|   |   |--- class: 0
|   |--- rec_pulso >  0.50
|   |   |--- grupo_sanguineo_0 <= 0.50
|   |   |   |--- class: 0
|   |   |--- grupo_sanguineo_0 >  0.50
|   |   |   |--- class: 1
|--- capnometria >  16.50
|   |--- edad <= 54.00
|   |   |--- grupo_sanguineo_A <= 0.50
|   |   |   |--- edad <= 46.50
|   |   |   |   |--- class: 1
|   |   |   |--- edad >  46.50
|   |   |   |   |--- edad <= 49.00
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- edad >  49.00
|   |   |   |   |   |--- capnometria <= 24.00
|   |   |   |   |   |   |--- capnometria <= 21.00
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |--- capnometria >  21.00
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- capnometria >  24.00
|   |   |   |   |   |   |--- class: 1
|   |   |--- grupo_sanguineo_A >  0.50
|   |   |   |--- causa_cardiaca <= 0.50
|   |   |   |   |--- class: 0
|   |   |   |--- causa_cardiaca >  0.50
|   |   |   |   |--- rec_pulso <= 0.50
|   |   |   |   |   |--- edad <= 49.00
|   |   |   |   |   |   |--- edad <= 45.00
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |--- edad >  45.00
|   |   |   |   |   |   |   |--- capnometria <= 28.00
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- capnometria >  28.00
|   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- edad >  49.00
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- rec_pulso >  0.50
|   |   |   |   |   |--- class: 1
|   |--- edad >  54.00
|   |   |--- IMC <= 24.73
|   |   |   |--- class: 1
|   |   |--- IMC >  24.73
|   |   |   |--- class: 0
```

info_arbol_decision_ampliado.txt

```
--- Evaluacion del modelo ---
Matriz de confusion:
[[12  1]
 [ 4  4]]

Reporte de clasificacion:
              precision    recall  f1-score   support

           0       0.75      0.92      0.83        13
           1       0.80      0.50      0.62         8

    accuracy                           0.76        21
   macro avg       0.78      0.71      0.72        21
weighted avg       0.77      0.76      0.75        21

Accuracy: 0.7619
```

info_random_forest.txt

```
INFORME RANDOM FOREST
Importancia de las caracteristicas:
edad: 0.3719
capnometria: 0.4248
femenino: 0.0214
causa_cardiaca: 0.0843
cardio_manual: 0.0590
rec_pulso: 0.0386


--- Evaluacion del modelo ---
Matriz de confusion:
[[11  2]
 [ 3  5]]

Reporte de clasificacion:
              precision    recall  f1-score   support

           0       0.79      0.85      0.81        13
           1       0.71      0.62      0.67         8

    accuracy                           0.76        21
   macro avg       0.75      0.74      0.74        21
weighted avg       0.76      0.76      0.76        21

Accuracy: 0.7619
```

info_random_forest_ampliado.txt

```
INFORME RANDOM FOREST AMPLIADO
Importancia de las caracteristicas:
edad: 0.2372
capnometria: 0.2559
femenino: 0.0147
causa_cardiaca: 0.0419
cardio_manual: 0.0382
rec_pulso: 0.0524
IMC: 0.2126
grupo_sanguineo_0: 0.0520
grupo_sanguineo_A: 0.0733
grupo_sanguineo_AB: 0.0031
grupo_sanguineo_B: 0.0188


--- Evaluacion del modelo ---
Matriz de confusion:
[[11  2]
 [ 4  4]]

Reporte de clasificacion:
              precision    recall  f1-score   support

           0       0.73      0.85      0.79        13
           1       0.67      0.50      0.57         8

    accuracy                           0.71        21
   macro avg       0.70      0.67      0.68        21
weighted avg       0.71      0.71      0.70        21

Accuracy: 0.7143
```

info_regresion_logistica.txt

```
INFORME REGRESION LOGISTICA
[[7  2]
 [3  2]]
              precision    recall  f1-score   support

           0       0.70      0.78      0.74         9
           1       0.50      0.40      0.44         5

    accuracy                           0.64        14
   macro avg       0.60      0.59      0.59        14
weighted avg       0.63      0.64      0.63        14

causa_cardiaca     1.260733
edad              -0.905774
capnometria        0.905129
rec_pulso          0.851284
cardio_manual     -0.722268
femenino          -0.347072
dtype: float64
```

info_regresion_logistica_ampliado.txt

```
INFORME REGRESION LOGISTICA AMPLIADO
[[8  5]
 [5  3]]
              precision    recall  f1-score   support

           0       0.62      0.62      0.62        13
           1       0.38      0.38      0.38         8

    accuracy                           0.52        21
   macro avg       0.50      0.50      0.50        21
weighted avg       0.52      0.52      0.52        21

rec_pulso            1.133049
causa_cardiaca       0.972019
grupo_sanguineo_A   -0.873955
edad                -0.840322
cardio_manual       -0.692039
capnometria          0.673339
grupo_sanguineo_0    0.570157
IMC                 -0.446965
grupo_sanguineo_B    0.438595
femenino            -0.138062
grupo_sanguineo_AB  -0.136783
dtype: float64
```

La comparación de las importancias que da cada modelo y la fórmula de la tesis a cada variable se puede ver en la siguiente tabla. Los modelos siguen el siguiente orden: *Árbol de decisión, Árbol de decisión ampliado, Random Forest, Random Forest ampliado, Regresión Logística y Regresión Logística ampliada.*

Variable \Modelo	ÁD	ÁD Amp	RF	RF Amp	RegLog	RegLog Amp	Fórmula
capnometria	0.5515	0.3713	0.4248	0.2559	0.905129	0.673339	0.08
cardio_manual	0.0000	0.0000	0.0590	0.0382	-0.722268	-0.692039	-2.616
causa_cardiaca	0.0685	0.0451	0.0843	0.0419	1.260733	0.972019	-0.517
edad	0.3799	0.3135	0.3719	0.2372	-0.905774	-0.840322	-0.096
sexo_femenino	0.0000	0.0000	0.0214	0.0147	-0.347072	-0.138062	-1.013
rec_pulso	0.0000	0.0698	0.0386	0.0524	0.851284	1.133049	2.943
IMC	-	0.0778	-	0.2126	-	-0.446965	-
grupo_sanguineo_0	-	0.0593	-	0.0520	-	0.570157	-
grupo_sanguineo_A	-	0.0632	-	0.0733	-	-0.873955	-
grupo_sanguineo_AB	-	0.0000	-	0.0031	-	-0.136783	-
grupo_sanguineo_B	-	0.0000	-	0.0188	-	0.438595	-

Tabla 7.2: Comparación pesos

Árbol de Decisión y Random Forest

Hay que tener en cuenta que los valores que se muestran para el Árbol de Decisión y para el Random Forest no pueden ser negativas. Por lo tanto, los valores son realmente la importancia de las características, no el peso como es en el caso de la fórmula. Así, poniendo como orden en la fórmula sus pesos en valor absoluto, podemos determinar que el orden de relevancia que dan estos dos modelos a las variables es el mostrado en la tabla 7.3.

Ránking importancia	Árbol de Decisión	Árbol de Decisión Ampliado	Random Forest	Random Forest Ampliado	Fórmula
1	capnometria	capnometria	capnometria	capnometria	rec_pulso
2	edad	edad	edad	edad	cardio_manual
3	causa_cardiaca	IMC	causa_cardiaca	IMC	sexo_femenino
4	cardio_manual	rec_pulso	cardio_manual	grupo_sanguineo_A	causa_cardiaca
5	sexo_femenino	grupo_sanguineo_A	rec_pulso	rec_pulso	edad
6	rec_pulso	grupo_sanguineo_0	sexo_femenino	grupo_sanguineo_0	capnometria
7	-	causa_cardiaca	-	causa_cardiaca	-
8	-	grupo_sanguineo_AB	-	cardio_manual	-
9	-	grupo_sanguineo_B	-	sexo_femenino	-
10	-	cardio_manual	-	grupo_sanguineo_B	-
11	-	sexo_femenino	-	grupo_sanguineo_AB	-

Tabla 7.3: Ránking de importancia de las variables

Los valores con el fondo amarillo significan que tienen valor 0. De esta forma vemos claramente cómo el modelo Random Forest ofrece un resultado más sólido que el Árbol de decisión, ya que para todas las variables ha podido asignar una importancia.

Para la versión inicial, un resultado sorprendente es que ambos modelos colocan en sus tres primeras posiciones las mismas variables, mientras que para la fórmula de la tesis estas variables son justo las tres con menos peso.

Para la versión ampliada, podemos ver cómo los modelos dan una importancia significativa para el IMC (posicionándola como la tercera variable), mientras que para el grupo sanguíneo parece que los grupos 0 y A tienen mayor relevancia que los grupos AB y B.

En cuanto a la matriz de confusión, en el Árbol de decisión podemos ver cómo predice correctamente 17 casos: 12 negativos y 5 positivos. Sin embargo, hay 3 falsos negativos y 1 falso positivo. Esta diferencia en este caso resulta beneficiosa, ya que se busca minimizar los falsos positivos y aceptar solo los donantes que el modelo está seguro que son válidos. En el Árbol de decisión ampliado, vemos que se predicen correctamente 16 casos: 12 negativos y 4 positivos. En cuanto a los falsos negativos son 4 las predicciones que dieron como resultado *no válido* cuando sí que lo era, y una predicción ha resultado ser un falso positivo. Así podemos apreciar cómo, al haber añadido el IMC y el grupo sanguíneo, el modelo tiene un menor valor de *accuracy* para el modelo inicial que para el ampliado (0.8095 frente a 0.7619). No osbtante, un valor que nos interesa más comparar es el de la precisión, que indica el porcentaje de las predicciones que el modelo predijo como válidas que realmente lo eran. Para esta métrica tenemos que el Árbol de decisión tiene una precisión de 0.81, mientras que para la versión ampliada la precisión es de 0.77.

Para el Random Forest, la matriz de confusión nos indica que ha predicho correc-

tamente 16 casos: 11 negativos y 5 positivos. También ha obtenido 3 falsos negativos y 2 falsos positivos. Para la versión ampliada, se han predicho correctamente 15 casos, de los cuales 11 son negativos y 4 positivos. Además, el número de falsos positivos y negativos coinciden con el de la versión inicial. Estos modelos muestran un menor valor de precisión que los árboles de decisión (0.76 y 0.71 respectivamente), pero al haber asignado una importancia a cada variable su interpretabilidad es mucho mayor. No obstante, su rendimiento en términos de aciertos aún es inferior en esta primera fase de entrenamiento.

Regresión Logística

En el caso de la Regresión Logística sí que se puede comparar directamente con los pesos de la fórmula. Como se ve en la tabla 7.2, los signos de cada variable coinciden tanto para el modelo inicial, el ampliado y la fórmula. Un valor negativo indica que la variable influye negativamente y, por lo tanto, reduce la probabilidad de que una predicción sea válida. En cambio, un signo positivo aumenta la probabilidad de que una predicción sea positiva. Por lo tanto, que los signos coincidan indica un aspecto positivo en relación a la fiabiliad de los modelos y de la fórmula.

En cuanto a los valores negativos, vemos que según la fórmula las variables que más influyen al resultado de forma negativa son la realización de la cardiocompresión y el sexo femenino. En cambio, para los modelos de Regresión Logística (tanto el inicial como el ampliado), las variables con mayor impacto negativo son la edad y la realización de la cardio compresión. No obstante, al haber entrenado los modelos con una base de datos reducida, el hecho de que los resultados se asemejen bastante a la fórmula sugiere que el modelo sigue un comportamiento adecuado.

En cuanto a la matriz de confusión, el modelo inicial de Regresión Logística predice correctamente 9 casos, de los cuales 7 son negativos y 2 positivos. Por otro lado, obtiene 3 falsos negativos y 2 falsos positivos. El modelo ampliado predice correctamente 11 casos, de los cuales, 8 son negativos y 3 positivos. En cuanto a los falsos negativos obtiene 5 casos y en cuanto a los falsos positivos obtiene 3. Estos números nos indican también una precisión baja, de 0.63 y 0.52 respectivamente. Para que estos valores mejoren se deben entrenar los modelos con una base de datos mucho mayor, que se conseguirá según se vaya utilizando la aplicación móvil.

7.5.2. Clustering

Para ambos conjuntos de predicciones (el inicial y el ampliado) hemos obtenido los siguientes clústers.

info_clustering.txt

```
Cluster 0:
    edad: 49.70
    capnometria: 43.90
    femenino: 0
    causa_cardiaca: 0
    cardio_manual: 0
    rec_pulso: 0
    valido: 0

Cluster 1:
    edad: 50.48
    capnometria: 25.07
    femenino: 0
    causa_cardiaca: 1
    cardio_manual: 0
    rec_pulso: 0
    valido: 0

Cluster 2:
    edad: 26.67
    capnometria: 20.67
    femenino: 0
    causa_cardiaca: 1
    cardio_manual: 0
    rec_pulso: 0
    valido: 1

Cluster 3:
    edad: 49.83
    capnometria: 10.91
    femenino: 0
    causa_cardiaca: 1
    cardio_manual: 0
    rec_pulso: 0
    valido: 0
```

Animación 3D para clusters del conjunto inicial (solo disponible al abrir con Adobe Acrobat):

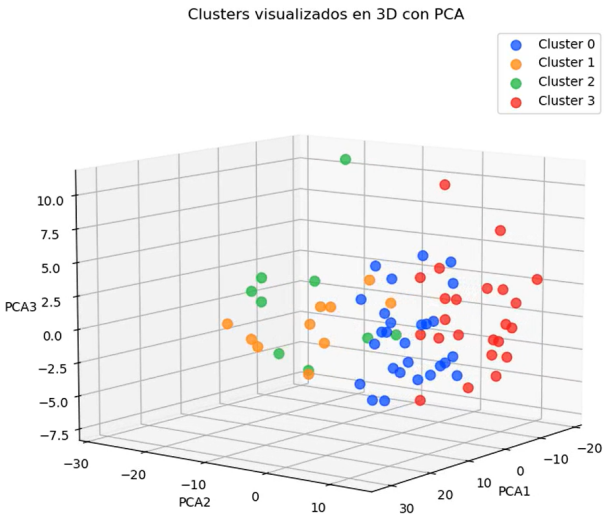


Figura 7.3: Animación 3D clusters

info_regresion_clustering_ampliado.txt

```
Cluster 0:
    edad: 50.48
    capnometria: 25.07
    IMC: 26.86
    femenino: 0
    causa_cardiaca: 1
    cardio_manual: 0
    rec_pulso: 0
    valido: 0
    grupo_sanguineo: A

Cluster 1:
    edad: 49.83
    capnometria: 10.91
    IMC: 27.82
    femenino: 0
    causa_cardiaca: 1
    cardio_manual: 0
    rec_pulso: 0
    valido: 0
    grupo_sanguineo: A

Cluster 2:
    edad: 49.70
    capnometria: 43.90
    IMC: 28.45
    femenino: 0
    causa_cardiaca: 0
    cardio_manual: 0
    rec_pulso: 0
    valido: 0
    grupo_sanguineo: A

Cluster 3:
    edad: 26.67
    capnometria: 20.67
    IMC: 26.78
    femenino: 0
    causa_cardiaca: 1
    cardio_manual: 0
    rec_pulso: 0
    valido: 1
    grupo_sanguineo: 0
```

Animación 3D para clusters del conjunto ampliado (solo disponible al abrir con Adobe Acrobat):

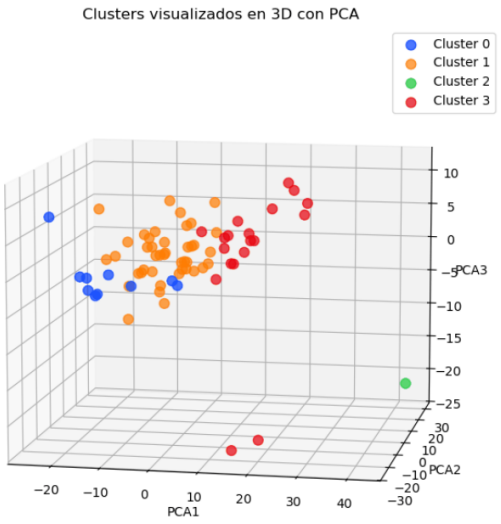


Figura 7.4: Animación 3D clusters ampliado

Para ambos conjuntos, solo hay un cluster que contenga la variable *válido* a 1, lo cual sugiere que este grupo concentra los casos con predicción válida. Para el conjunto inicial se trata del clúster 2 que tiene las siguientes características:

- edad: 26.67
- capnometria: 20.67
- femenino: 0
- causa_cardiaca: 1
- cardio_manual: 0
- rec_pulso: 0
- valido: 1

Para el conjunto ampliado se trata del clúster 3:

- edad: 26.67
- capnometria: 20.67
- IMC: 26.78
- femenino: 0
- causa_cardiaca: 1
- cardio_manual: 0
- rec_pulso: 0
- valido: 1

- grupo_sanguineo: 0

Ambos agrupan a varones jóvenes, a los que no se les ha realizado una cardiocompresión, que no han recuperado el pulso y que han muerto por una causa cardiaca. Además, el índice de la capnometría es bajo (lo normal es 35–45 mmHg).

En cuanto al grupo sanguíneo, el grupo 0 está asociado a una predicción válida, mientras que el IMC que muestra de media el clúster está asignado a un valor bajo dentro de la categoría *Sobrepeso*. (31)

Una conclusión relevante que se puede sacar de la animación 3D del clúster ampliado, es que el clúster 2 se trata de un grupo separado. Este grupo se distingue del resto por tener de media a donantes que no han muerto por causa cardiaca y un valor de capnometría mucho más elevado que el resto (43.90). Esto indica que se trata de un subgrupo con condiciones particulares, lo que sugiere que merece una atención especial en futuros análisis.

Capítulo 8

Diseño

El objetivo principal de la aplicación es proporcionar al personal médico del Servicio de Urgencia Médica de la Comunidad de Madrid una aplicación sencilla e intuitiva que les permita determinar (a partir de información que mayoritariamente es recibida por llamada) si un donante es válido o no. Dado que se trata de una situación que puede resultar muy estresante, contar con una interfaz clara y ágil resulta esencial. Por ello, la aplicación móvil ha sido diseñada con secciones básicas y funcionales, y utiliza una paleta de colores acorde a la unidad de SUMMA: azul y amarillo.

Los principales códigos de HTML empleados han sido, tanto para la aplicación móvil como la web, los siguientes: (22)

- #042939: color principal para la fuente
- #417f9a: color de fondo principal tanto en pantallas como en contenedores
- #fcfe82: utilizado en botones y como delimitadores de campos de entrada en formularios
- #7cc873: predicciones válidas
- #e56f66: predicciones no válidas

El tipo de fuente utilizado en toda la aplicación web es *Roboto Condensed, sans-serif* y en la móvil *sans-serif-condensed-medium*, ambas fuentes son condensadas y contribuyen al diseño moderno de la interfaz.

Todos los iconos de la aplicación han sido descargados de la web *falticon*. (14)

8.1. Aplicación móvil

Al iniciar sesión, el usuario con rol *Médico* se encuentra con una gráfico circular donde aparecen el número de predicciones realizadas, cuántas son válidas y cuántas no. Debajo de este gráfico aparecen botones con las funcionalidades básicas y principales de estos usuarios, que se describen más adelante.

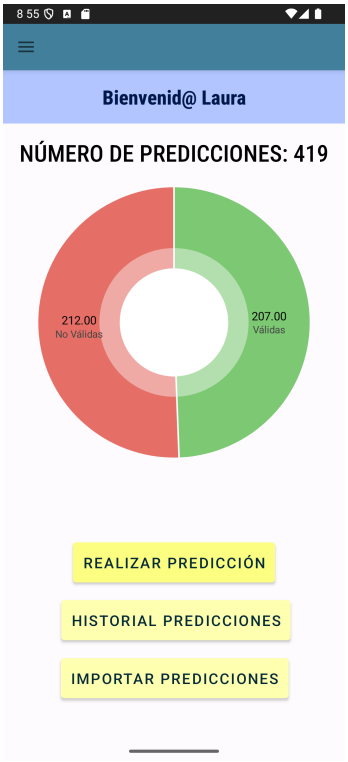


Figura 8.1: Captura de pantalla: Gráfico circular menú principal Médico

Por otro lado, tanto para los Médicos como para los Administradores, la aplicación móvil cuenta con un menú lateral desplegable donde aparece el nombre del usuario que ha iniciado sesión y dos items: uno para editar el perfil y otro para cerrar sesión.

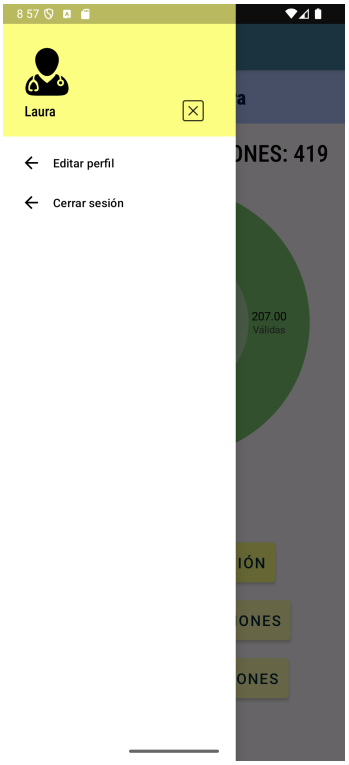
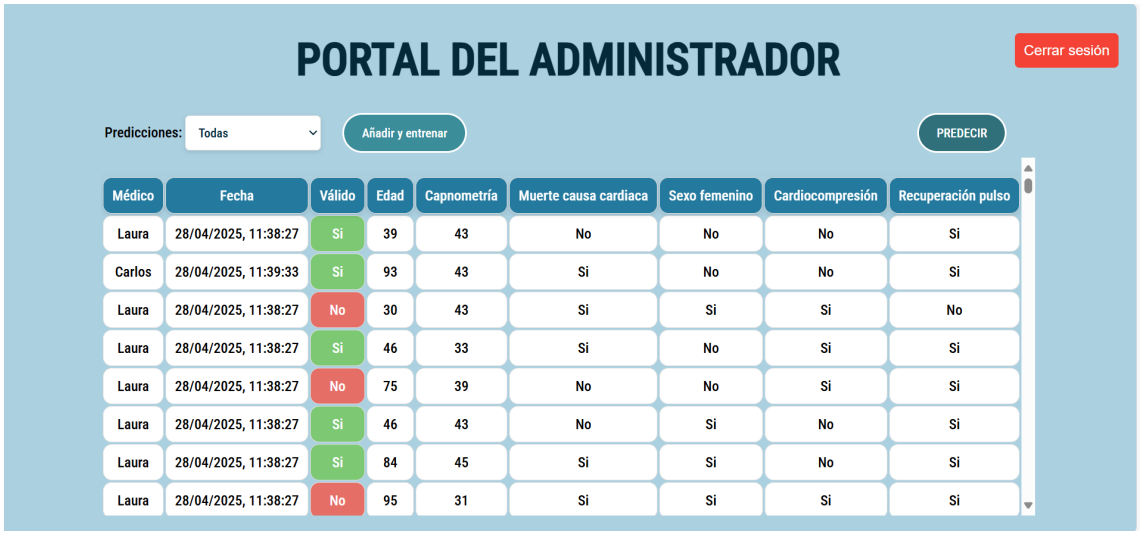


Figura 8.2: Captura de pantalla: Menú desplegable

También se usan algunas animaciones para hacer que la experiencia sea más dinámica. Cuando el usuario rellena el formulario con los datos de la predicción, aparece el resultado en un contenedor que aparece empezando más pequeño y va creciendo hasta alcanzar su tamaño normal en un segundo. Este tipo de detalles ayudan a que la interfaz se sienta más viva y la interacción sea más agradable. También el gráfico circular se actualiza de manera animada, completándose el círculo cada vez que se carga la página principal.

8.2. Aplicación web

En la aplicación web, las tablas han sido diseñadas con un estilo moderno y funcional. Tienen un encabezado fijo que se mantiene visible al hacer scroll, lo que permite visualizar el contenido de forma clara en todo momento.



Médico	Fecha	Válido	Edad	Capnometría	Muerte causa cardiaca	Sexo femenino	Cardiocompresión	Recuperación pulso
Laura	28/04/2025, 11:38:27	Si	39	43	No	No	No	Si
Carlos	28/04/2025, 11:39:33	Si	93	43	Si	No	No	Si
Laura	28/04/2025, 11:38:27	No	30	43	Si	Si	Si	No
Laura	28/04/2025, 11:38:27	Si	46	33	Si	No	Si	Si
Laura	28/04/2025, 11:38:27	No	75	39	No	No	Si	Si
Laura	28/04/2025, 11:38:27	Si	46	43	No	Si	No	Si
Laura	28/04/2025, 11:38:27	Si	84	45	Si	Si	No	Si
Laura	28/04/2025, 11:38:27	No	95	31	Si	Si	Si	Si

Figura 8.3: Captura de pantalla: Ejemplo de tabla

Por otro lado, para mostrar los resultados de los diferentes modelos se ha hecho hincapié en el uso de los colores rojos y verdes, de forma que sea más visual ver si el porcentaje de concordancia es alto o bajo.

Figura 8.4: Captura de pantalla: Resultados Modelos

Capítulo 9

Implementación

Este capítulo diferencia entre el código de ambas aplicaciones para explicar la estructura de cada una y mostrar su interfaz.

9.1. Aplicación móvil

9.1.1. Estructura

En Android Studio se tiene el proyecto *tfg_prediccion*. Al crearlo, aparecen una serie de carpetas y archivos automáticamente, de los cuales los principales son:

- **directorio *app*:** directorio principal que contiene el código fuente, los recursos visuales y la configuración específica de la app. En él se ha incluido el archivo *google-services.json* con la configuración de la base de datos.
- ***app/src/main*:** subdirectorio principal
- ***app/src/main/java*:** directorio con los archivos de código escritos en Kotlin (se llama Java por defecto). Aquí se encuentran una serie de paquetes:
  - ***data/model*:** contiene tres clases de datos: LoggedInUser, Medico y Prediccion. En ellas se definen los atributos necesarios para cada objeto.
  - ***ui*:** contiene seis paquetes para separar en módulos las funcionalidades de la aplicación: *edit*, *historial*, *login*, *main*, *prediction* y *register*. Cada una incluye las actividades necesarias con la lógica de la aplicación.
- ***app/src/main/res*:** directorio que contiene todos los recursos utilizados por la aplicación
- ***res/anim*:** contiene los archivos de animación
- ***res/drawable*:** contiene todos los iconos y recursos gráficos como por ejemplo, los campos a rellenar de los formularios o las flechas de los spinners.
- ***res/layout*:** contiene todos los archivos XML que definen las interfaces de usuario de cada pantalla (por ejemplo, para la actividad *EditProfileActivity* tenemos *activity_editprofile.xml*)
- ***app/src/main/AndroidManifest.xml*:** archivo donde se declaran las actividades de la aplicación, indicando cuál es por la que se debe comenzar, y otros parámetros como el icono de la aplicación.

- ***app/src/build.gradle.kts*:** script de configuración del módulo app. En él se incluyen las dependencias necesarias para instalar librerías, como las de los servicios de Firebase o iText7 para generar documentos PDF.

En Android Studio cada pantalla es una actividad, que corresponde a un archivo *.kt*. Estos scripts contienen una clase de la pantalla correspondiente, que hereda de *AppCompatActivity*. Esta es una clase base que permite utilizar funcionalidades nuevas en el proyecto, pese a haberlo configurado con una versión Android más antigua. (5) Por ejemplo, para la pantalla de Inicio de Sesión tenemos el siguiente código de la figura 9.1, donde podemos ver como se declara la clase LoginActivity que hereda de AppCompatActivity. A continuación, siempre se declara la variable *binding*, que sirve para acceder a los elementos definidos en el *xml*, como por ejemplo los botones o los campos a rellenar. También se define siempre la función *onCreate*, en la que se inicializan los componentes necesarios y se escribe la base de la lógica de la actividad. Esta estructura se sigue para todas las pantallas de la aplicación.

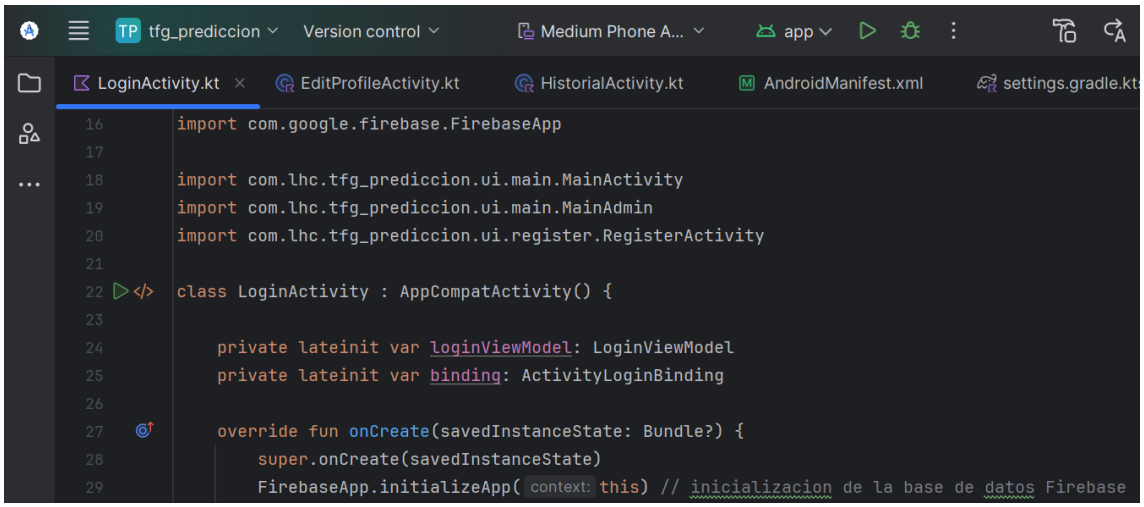


Figura 9.1: Captura de pantalla: Código AppCompatActivity

Otro aspecto que comparten las pantallas es el modo en que aparecen notificaciones brevemente en la pantalla para informar al usuario sobre alguna acción o resultado. Esto se consigue utilizando *Toast*, una clase de Android, que se llama de la siguiente forma mostrada en la figura 9.2. A esta clase se la llama pasándole como argumentos la propia actividad, el texto que se desea mostrar y el tiempo que debe mostrarse el Toast (Toast.LENGTH_SHORT o Toast.LENGTH_LONG). El ejemplo coincide con la notificación mostrada en la figura **??**.

```
// todos los campos deben estar llenos
if (name.isEmpty() || lastname.isEmpty() || birthdate.isEmpty() || email.isEmpty() || password.isEmpty()
    || confirmPassword.isEmpty()) {
    Toast.makeText( context: this,  text: "Todos los campos son obligatorios", Toast.LENGTH_SHORT).show()
    return@setOnClickListener
}
```

Figura 9.2: Captura de pantalla: Código Toast

En este proyecto se han configurado los aspectos técnicos de la aplicación en el

archivo *build.gradle.kts*. En este se especificaron las diferentes versiones que se iban a emplear de la siguiente forma:

- *compileSdk* = 35: la app se compila con la versión 35 del SDK de Android
- *targetSdk* = 35: se espera que la app se ejecute en dispositivos con hasta la versión 35 del SDK
- *minSdk* = 24: la app es compatible con dispositivos Android desde la versión 7.0 en adelante

Esto se corresponde a lo mostrado por la captura de la figura

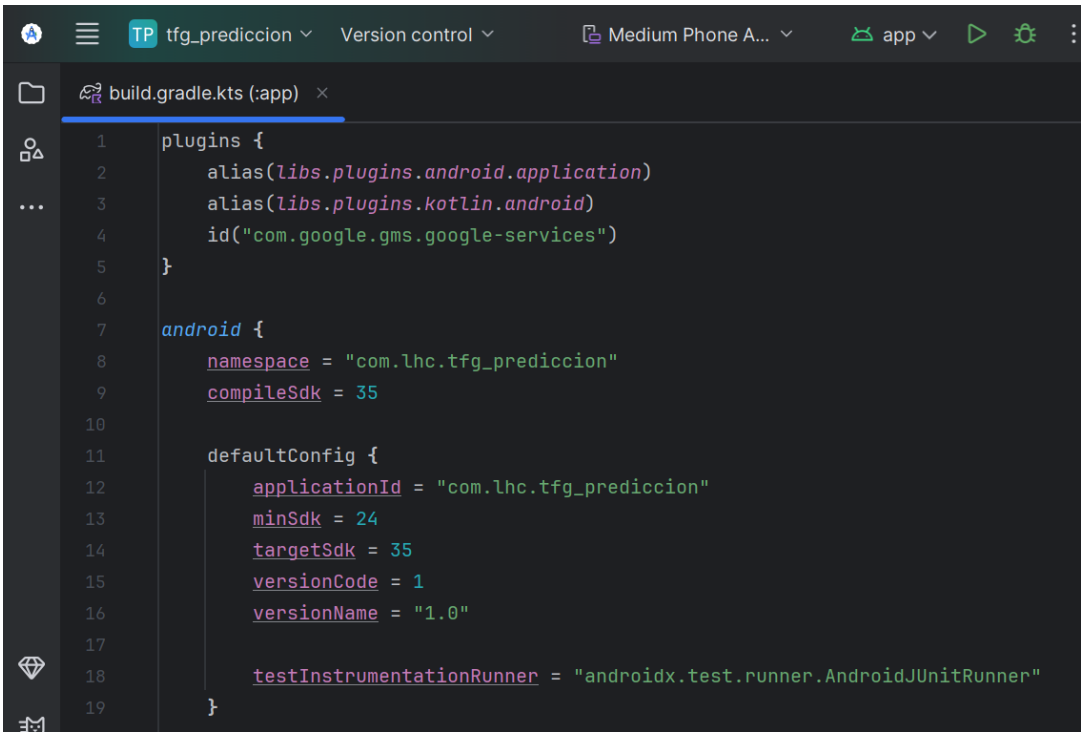


Figura 9.3: Captura de pantalla: versiones

9.1.2. Módulo inicial

9.1.2.1. Iniciar Sesión

Al abrir la aplicación nos encontramos con una pantalla de inicio de sesión como la mostrada en la figura 9.4. Si tratamos de iniciar sesión con un usuario y/o contraseña no válidos, saldrá una notificación como en la figura 9.5.

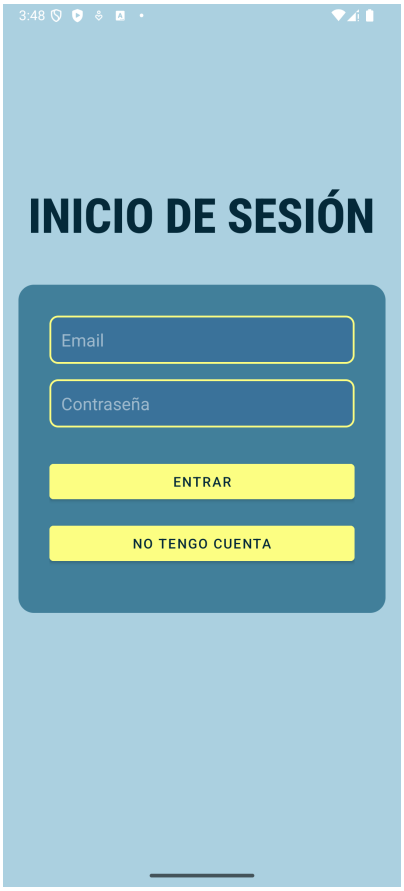


Figura 9.4: Captura de pantalla: Login

Figura 9.5: Captura de pantalla: Login fallido

Al crear el proyecto en Android Studio, se seleccionó la opción de *Login Activity* como plantilla inicial. De esta forma se creó de forma automática toda la base de la lógica del inicio de sesión. Como se ve en la figura 9.1, se define junto con el binding una variable *loginViewModel*. Esta se encarga de validar los datos introducidos por el usuario, de gestionar la autenticación conectándose a la base de datos y de separar la lógica de la vista.

La función que se encarga de comprobar si los datos introducidos por el usuario son válidos o no es la mostrada en la figura 9.6 y 9.7. Esta función utiliza *auth*, que es un objeto que representa el servicio de autenticación definido por Firebase. Con este objeto se llama al método *SignInWithEmailAndPassword*, que se utiliza para iniciar sesión con un correo y contraseña. Este es un método asíncrono y devuelve un objeto Task, que representa una operación que puede completarse con éxito o fallar. Para saber cuando termina, se añade un *listener* donde se comprueba si la autenticación ha resultado exitosa o no. Si la autenticación es correcta, se obtiene el usuario actual con *val user = task.result?.user* y se obtiene su *uid*. Con este se hace una consulta a la base de datos Firestore para conocer el rol y el nombre del usuario. Esta consulta se realiza con *db.collection(users).document(userId).get()*, que coge de la colección *users* el documento que tenga ese *uid*. Finalmente se devuelve un objeto LoggedInUser con los datos necesarios y se invoca el callback para avisar de que el login fue exitoso. En el caso de que la autenticación no haya resultado exitosa, se utilizan con-

diciones para distinguir el tipo de excepción que ha lanzado Firebase Authentication, como por ejemplo *FirebaseAuthInvalidCredentialsException* (cuando las credenciales que el usuario ingresó son inválidas) o *FirebaseAuthInvalidUserException* (cuando el usuario no existe o ha sido eliminado). Al igual que en el caso anterior, se invoca el callback para avisar de que la autenticación no fue exitosa.

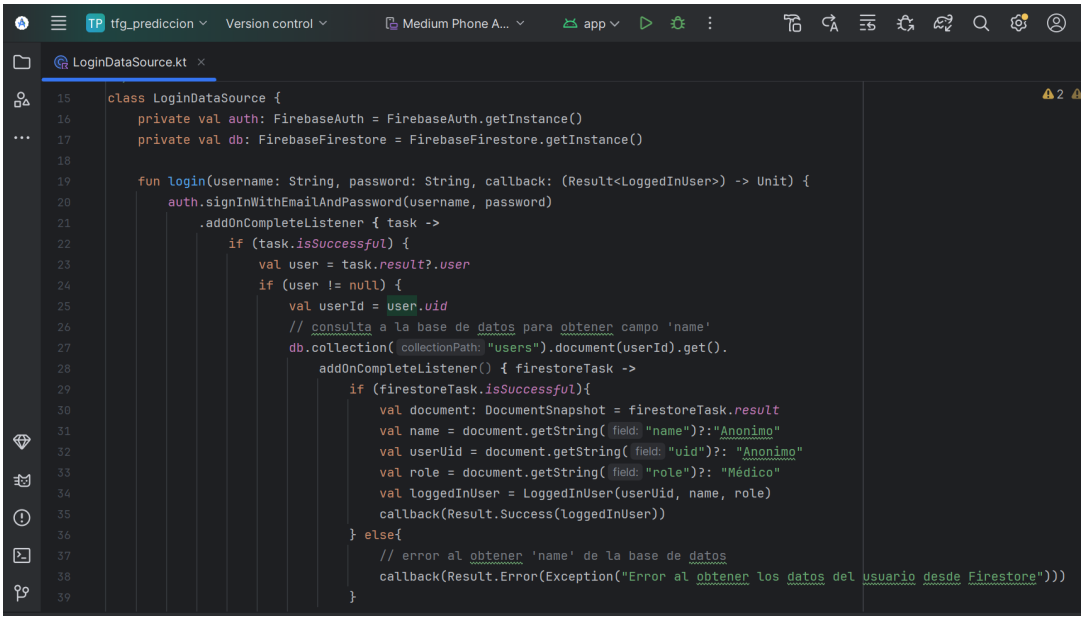


Figura 9.6: Captura de pantalla: Lógica de Login

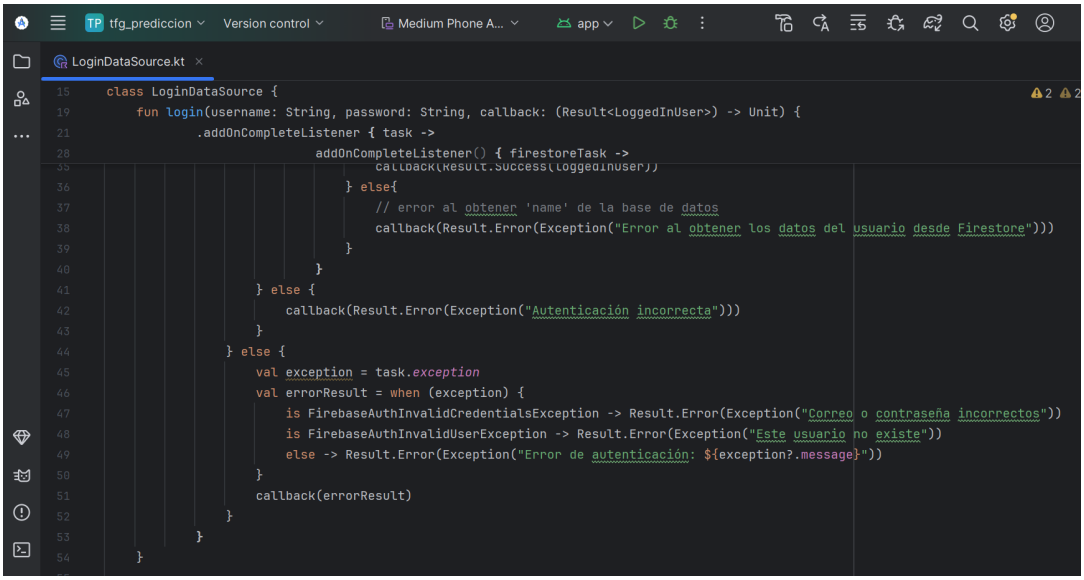


Figura 9.7: Captura de pantalla: Lógica de Login (2)

9.1.2.2. Registro

En la pantalla de Inicio de sesión, si pulsamos el botón de *No tengo cuenta*, nos lleva a la pantalla de registro como la que nos muestra la figura 9.8. Es necesario

rellenar todos los campos para poder crear una cuenta. En el caso de que haya ya un usuario con ese correo electrónico o las contraseñas no coincidan, aparecen notificaciones especificando el problema en concreto.

Figura 9.8: Captura de pantalla: Pantalla de registro

Para mostrar este formulario, en la actividad de Registro, en la función *onCreate*, se utiliza de nuevo una variable *binding* para acceder de a los elementos de la interfaz gráfica. Esto incluye: configurar los valores del spinner, el listener para el botón de volver al inicio de sesión, el diálogo para mostrar un calendario cuando eliges la fecha de nacimiento, etc.

El más importante es el mostrado en la figura 9.9, que contiene el listener del botón que se pulsa tras haber rellenado todos los campos con los nuevos datos del usuario. En este listener se cogen los datos del formulario utilizando el binding y se comprueba que la contraseña coincide con la confirmación. Después, llama a la función *registerUser* (figura 9.10), que utiliza la función de Firebase Authentication: *createUserWithEmailAndPassword* para crear una cuenta con ese usuario y contraseña. Después, esta función llama a *saveUserToFirestore* (figuras 9.11 y 9.12) para guardar los datos del usuario como el rol, el nombre y la fecha de nacimiento en la base de datos.

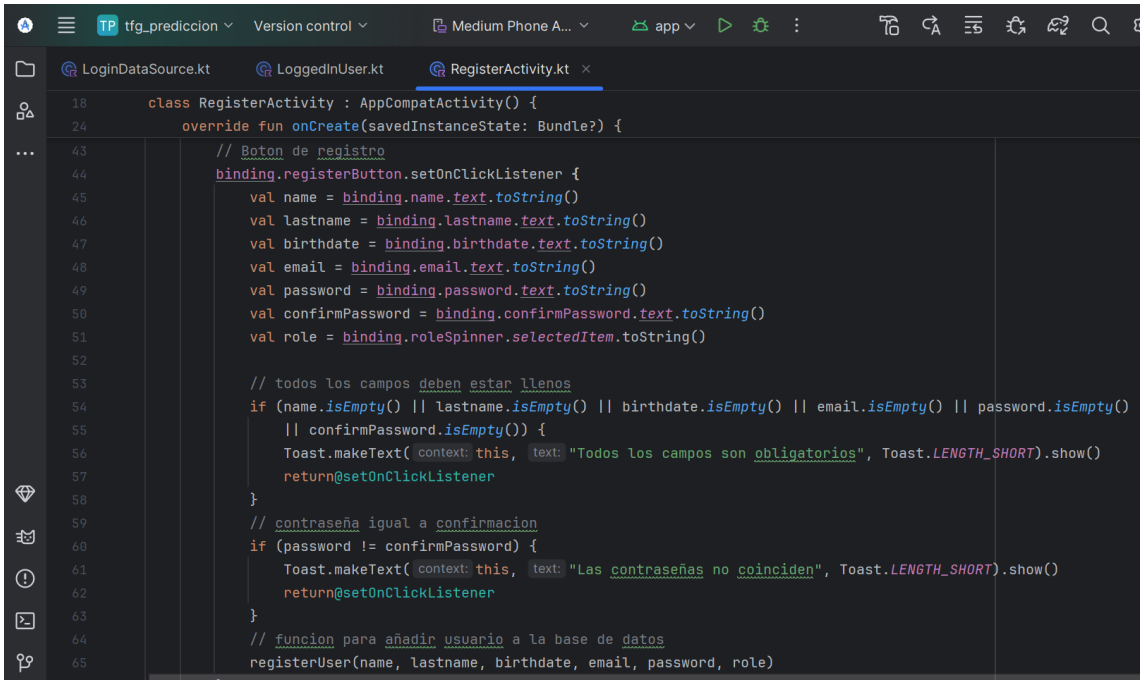


Figura 9.9: Captura de pantalla: Código del registro (1)

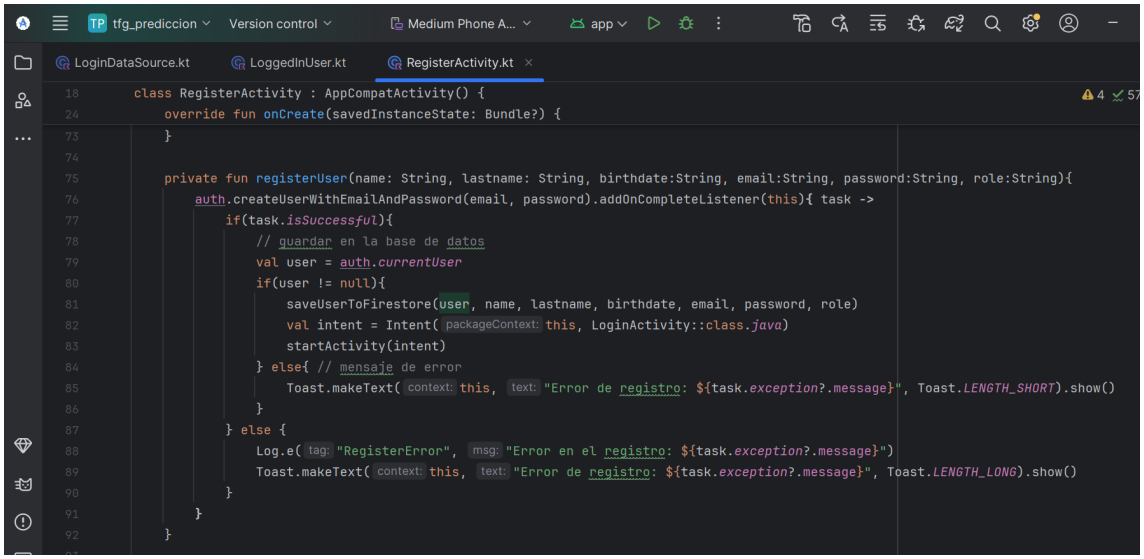


Figura 9.10: Captura de pantalla: Código del registro (2)

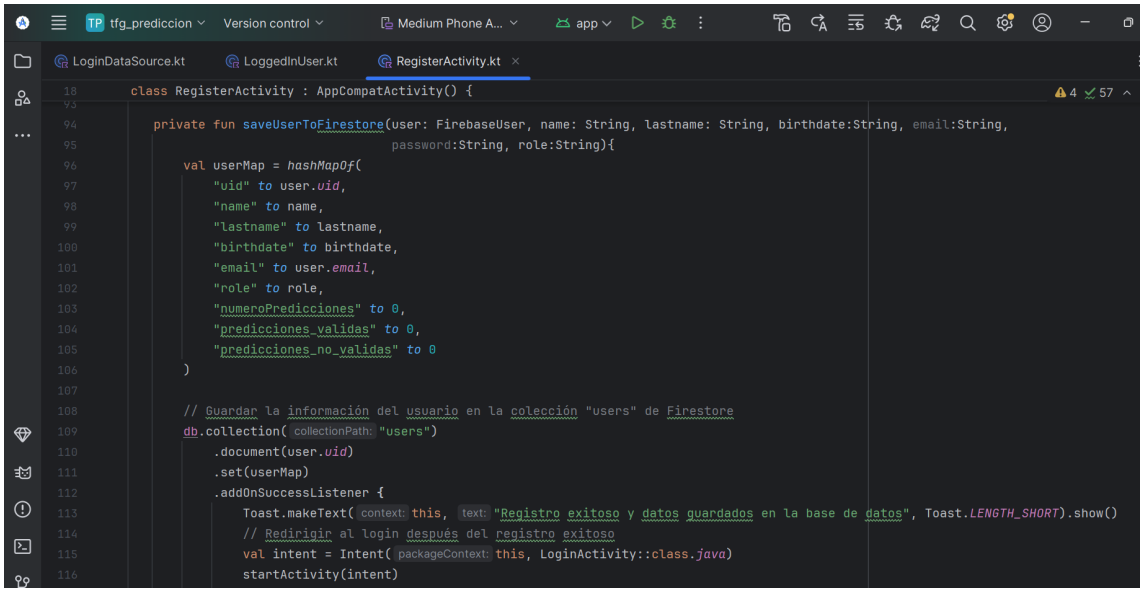


Figura 9.11: Captura de pantalla: Código del registro (3)

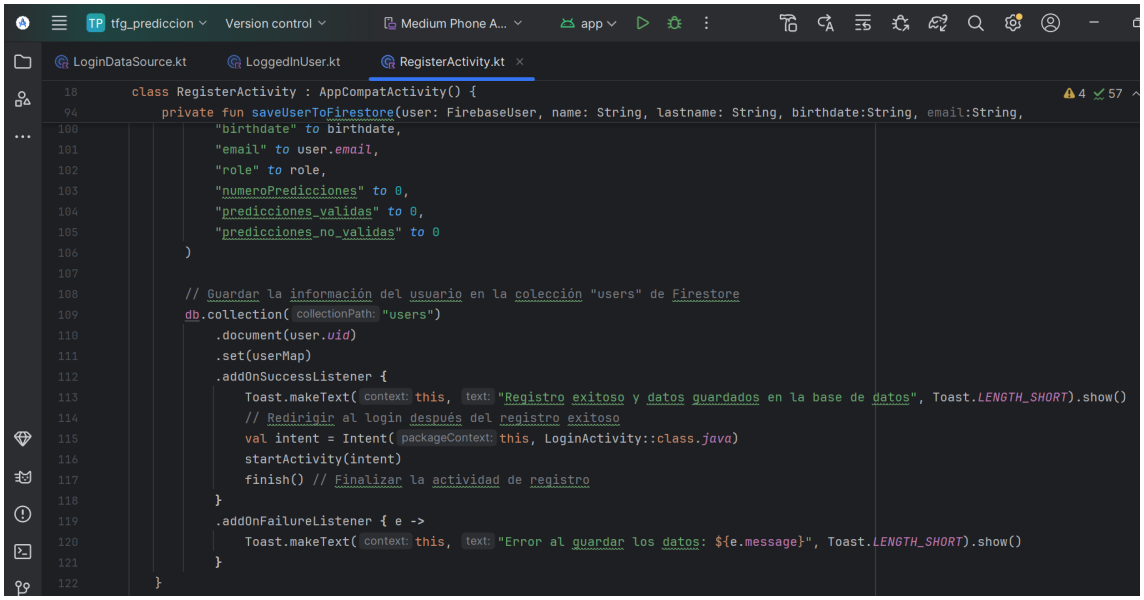


Figura 9.12: Captura de pantalla: Código del registro (4)

9.1.2.3. Menú principal: menú lateral

Una vez se ha iniciado sesión, tanto los Médicos como los Administradores pueden abrir un menú lateral desplegable como el de la figura 8.2 mostrado anteriormente.

Para conseguir este menú lateral, la clases de las actividades principales del médico y del administrador heredan de AppCompatActivity e implementan la interfaz *NavigationView.OnNavigationItemSelectedListener*. Esta interfaz permite gestionar las opciones del menú lateral mediante el método *onNavigationItemSelected*, que se encarga de detectar qué opción ha sido seleccionada por el usuario y ejecutar la acción correspondiente, como navegar a la pantalla de editar perfil o cerrar sesión.

El menú se inicializa en el método onCreate, donde se configura el *DrawerLayout* junto con un *ActionBarDrawerToggle* que sincroniza el icono del menú en la barra superior con el estado abierto o cerrado del cajón lateral. Además, se asigna el listener para recibir los eventos de selección de items del menú y así poder reaccionar adecuadamente.

Finalmente, tras la selección de una opción, el menú lateral se cierra automáticamente para pasar a la pantalla correspondiente. Toda esta funcionalidad se puede ver en el código de las figuras 9.13, 9.14 y 9.15.

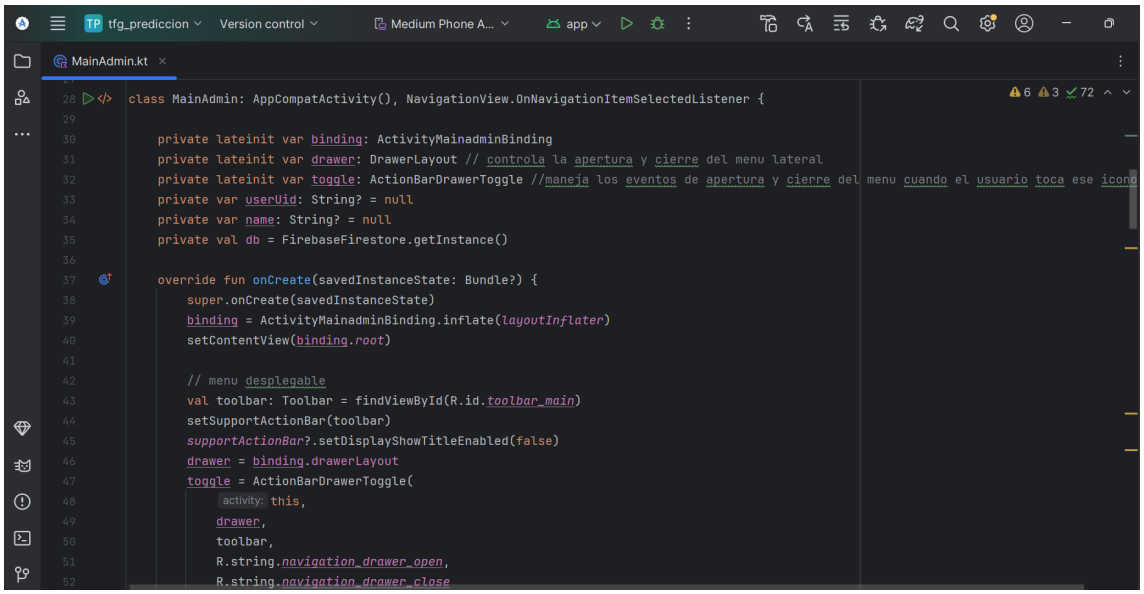


Figura 9.13: Captura de pantalla: Código del Menú desplegable (1)

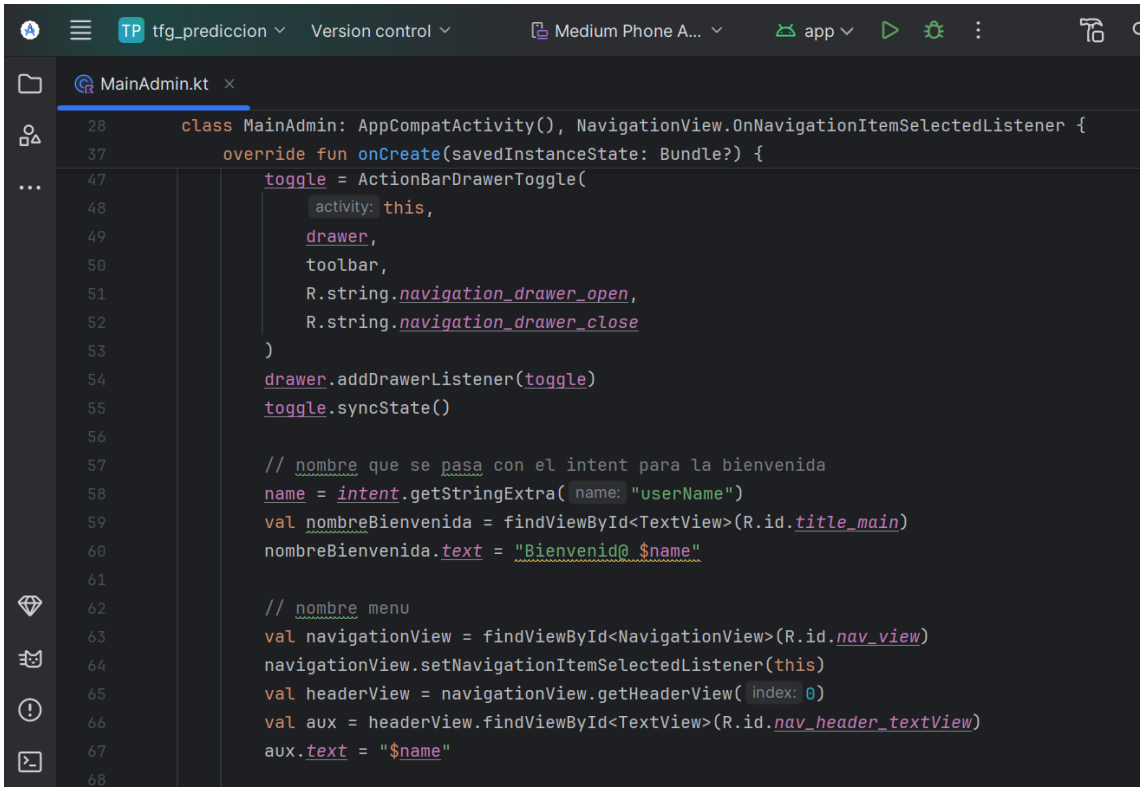


Figura 9.14: Captura de pantalla: Código del Menú desplegable (2)

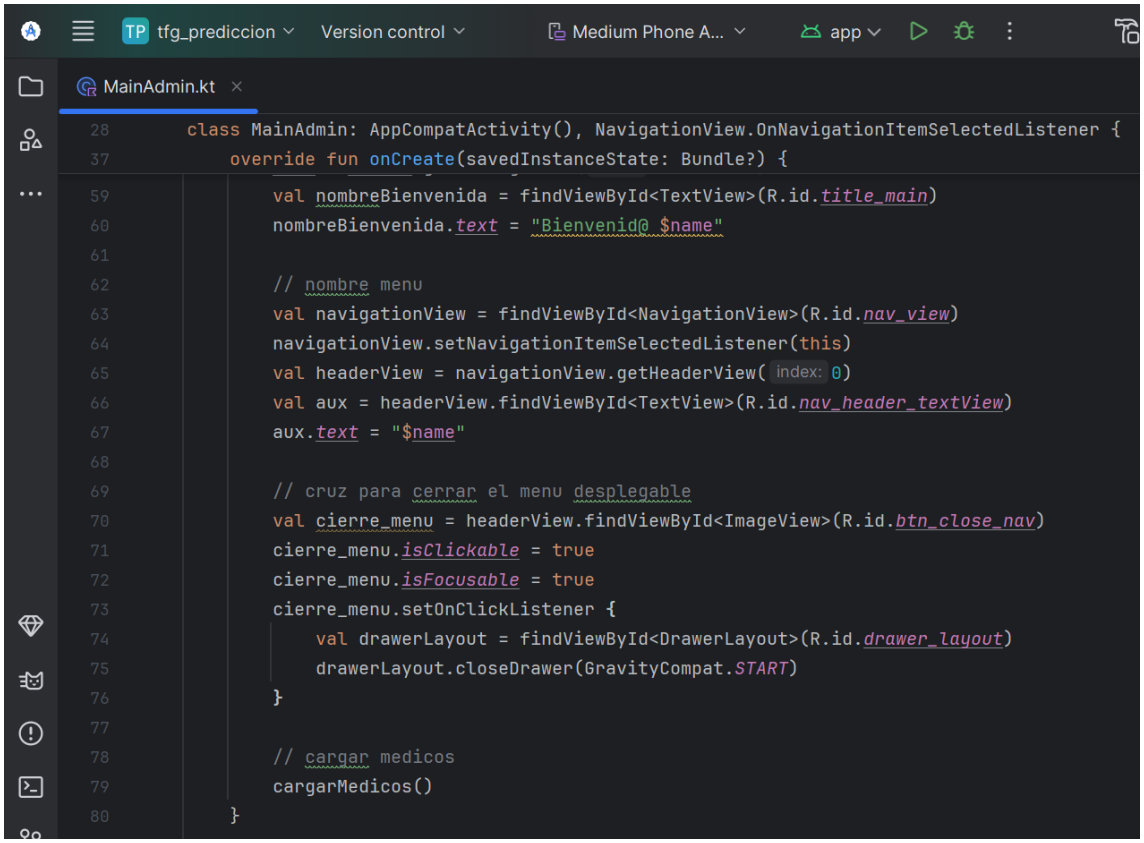


Figura 9.15: Captura de pantalla: Código del Menú desplegable (3)

En la siguiente figura se puede ver la función a través de la cual se navega a las actividades correspondientes según la opción seleccionada en el menú.

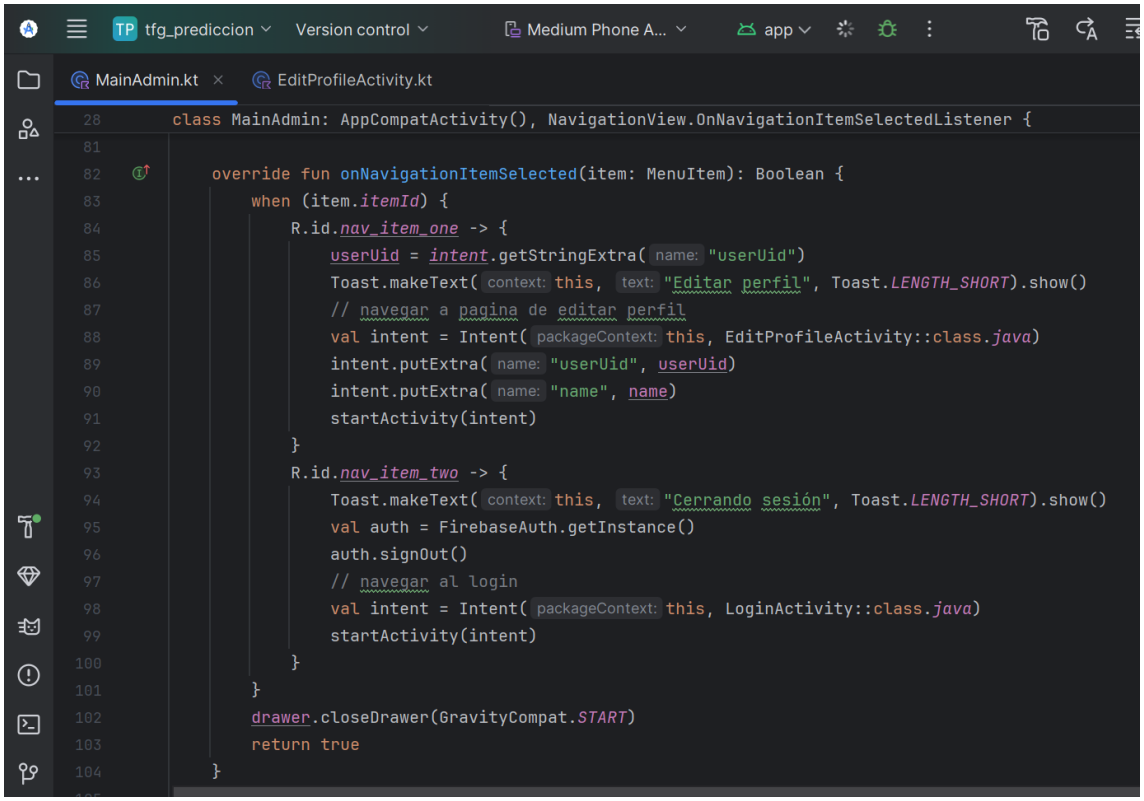


Figura 9.16: Captura de pantalla: Código del selección del Menú desplegable

9.1.2.4. Editar perfil

El menú lateral consta de dos opciones: Editar perfil y Cerrar sesión. Si pulsamos sobre *Editar perfil*, navegamos a un formulario similar al del registro, pero con los campos ya completados y sin la posibilidad de modificar el rol ni el nombre de usuario. Una vez se realizan los cambios, se muestra una notificación de éxito y se navega automáticamente a la pantalla principal.

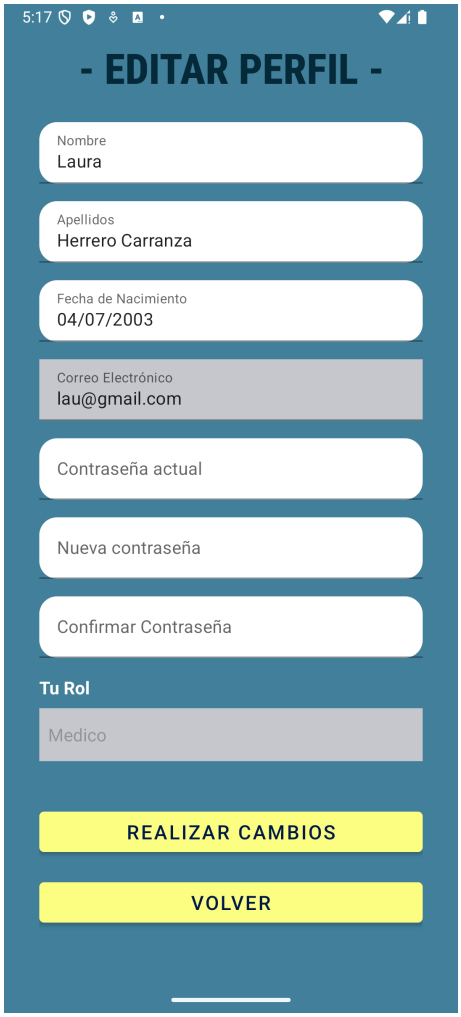


Figura 9.17: Captura de pantalla: Editar perfil

En cuanto al código, la lógica de mostrar el formulario y los elementos es igual que la de la pantalla de registro. Sin embargo, a la hora de actualizar los datos en la base de datos se siguen los siguientes pasos descritos a continuación.

Como se ve en la figura 9.18, primero se comprueba que los campos de la contraseña actual, la nueva contraseña y la confirmación no estén vacíos (lo que significa que se desea modificar). Después se verifica que la nueva contraseña y la confirmación coincidan, asegurando que el usuario haya escrito correctamente su nueva contraseña.

Para actualizar la contraseña, Firebase requiere que el usuario esté autenticado recientemente, como medida de seguridad. Para realizar esta autenticación, se utiliza el método *EmailAuthProvider.getCredential(email, password)*. Este método devuelve una credencial basada en el correo y la contraseña actual del usuario, que se usa posteriormente al llamar a *currentUser.reauthenticate*. Esta función permite reautenticar al usuario antes de hacer cambios sensibles, como es actualizar la contraseña.

Si la reautenticación es exitosa, se llama al método *updatePassword* (figura 9.19) para actualizar la contraseña del usuario en Firebase. Si esta actualización también

es exitosa, se llama a la función updateUser(), que actualiza los otros datos del usuario en la base de datos por si hubiera algunos cambios en el formulario.

Si ocurre algún error durante todo el proceso, se muestran mensajes de error mediante diferentes Toasts.

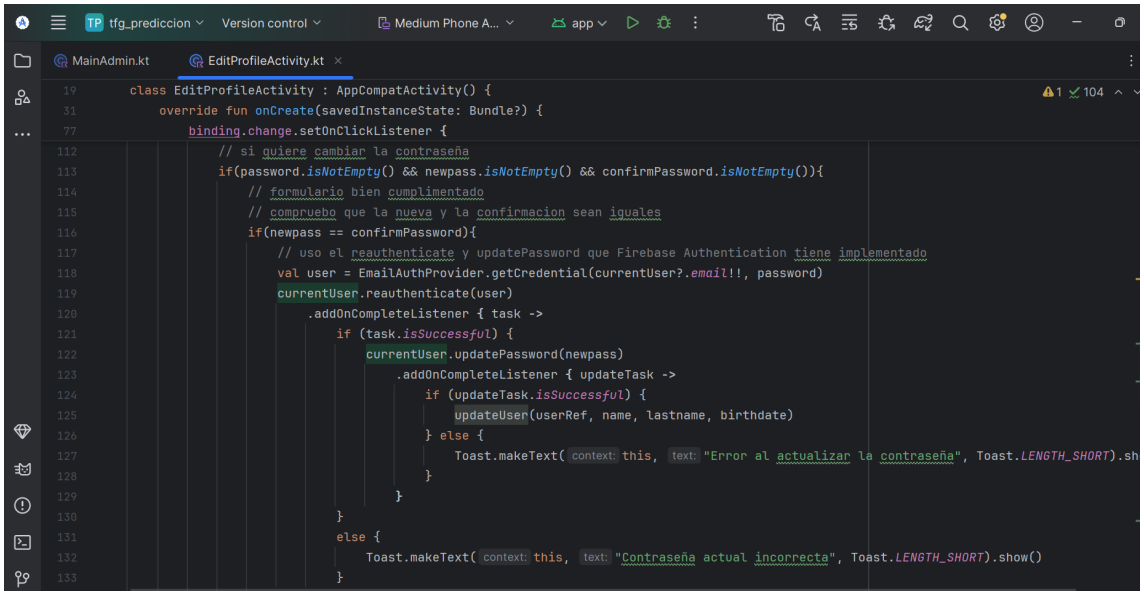


Figura 9.18: Captura de pantalla: Código actualizar datos usuario (1)

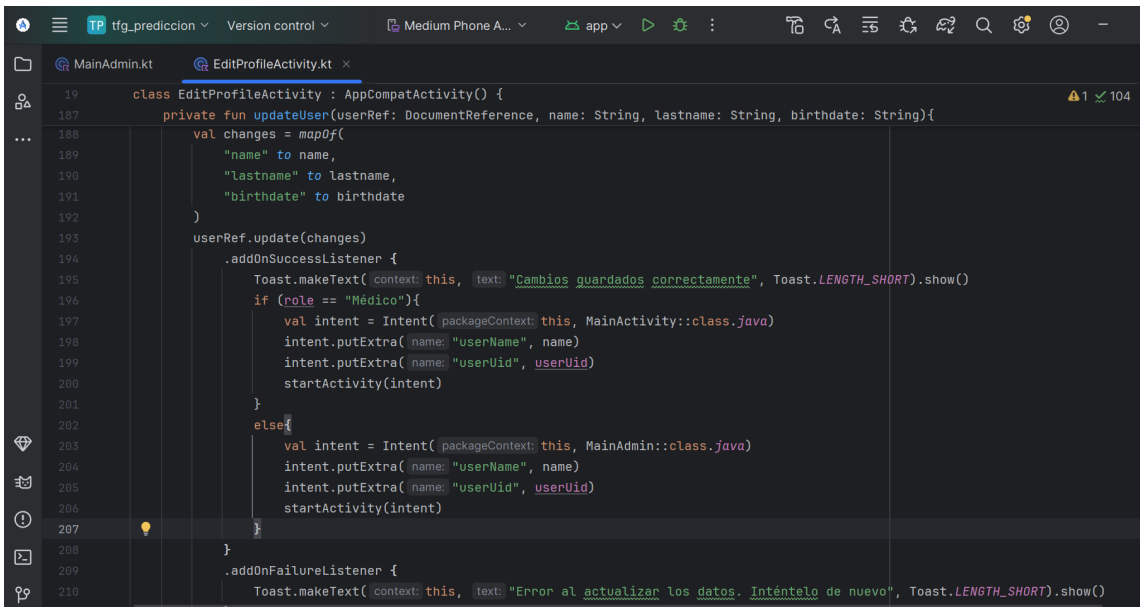


Figura 9.19: Captura de pantalla: Código actualizar datos usuario (2)

9.1.2.5. Cerrar sesión

Como se puede ver en la figura 9.16 mostrada anteriormente, al pulsar el botón de cerrar sesión se muestra un Toast notificándolo, se cierra sesión a través del método de Firebase Authentication, y se navega a la pantalla de inicio de sesión.

9.1.3. Módulo de Usuario: Médico

Al entrar como Médico se abre la pantalla principal, como se muestra en la figura 9.20.

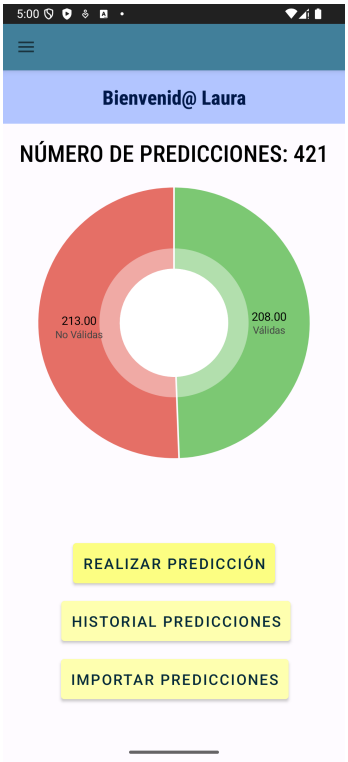


Figura 9.20: Captura de pantalla: Pantalla principal Médico

Este gráfico de sectores se consigue gracias a la biblioteca *MPAndroidChart*. Como se muestra en la figura 9.21, primero se ha obtenido la referencia del componente desde el archivo *xml* de la actividad mediante *findViewById*. Este método permite acceder al gráfico y modificar los datos que se muestran en él. A continuación se ha accedido a la base de datos para obtener el número de predicciones válidas y el número de predicciones no válidas que el médico en concreto ha realizado. Si esta consulta resulta exitosa, se llama a la la función *configurarPieChart* donde se espcifican de forma detallada todos los detalles sobre la gráfica (espacios, colores, leyenda, etc).

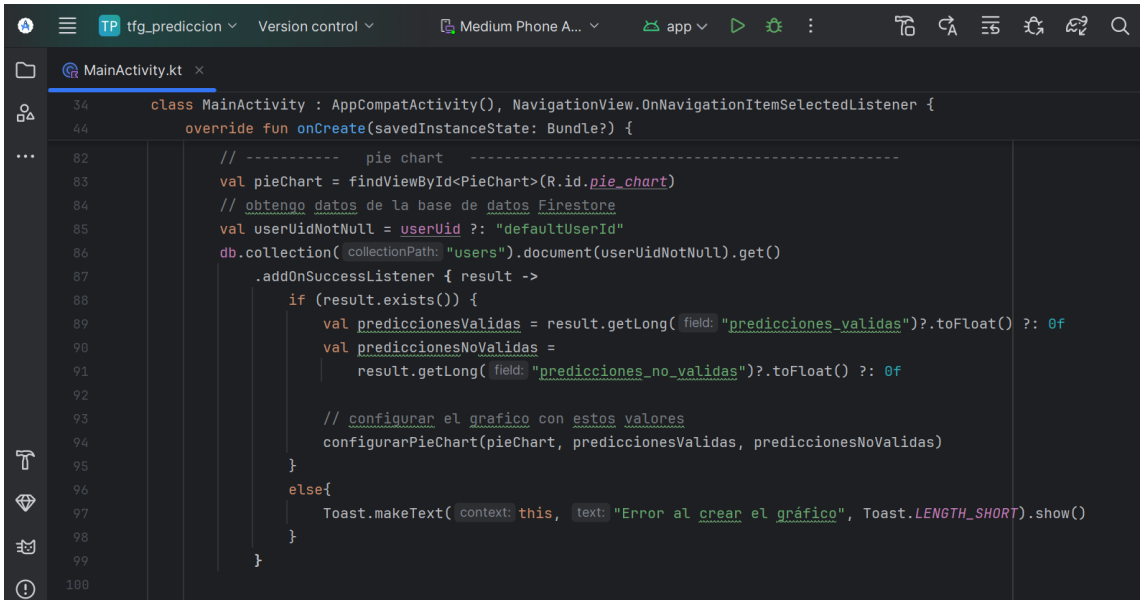


Figura 9.21: Captura de pantalla: Código Pie Chart

9.1.3.1. Realizar predicción

Si en el menú principal pulsamos el botón de *Realizar predicción*, navegamos al formulario para introducir los datos del donante, como se ve en la figura 9.22. En este formulario todos los campos son obligatorios, como se puede ver en la notificación de la figura 9.23. Además, este formulario solo permite que los campos de *Edad* y *Capnometría* sean números, lo que se consigue gracias a *android:inputType="number"* en su correspondiente campo del archivo XML (figura 9.24).

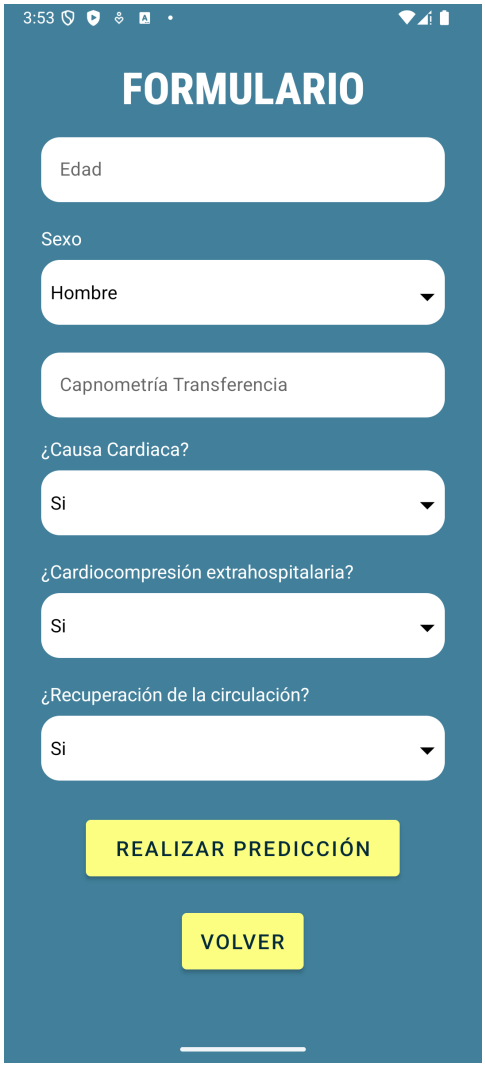


Figura 9.22: Captura de pantalla: Formulario de donante

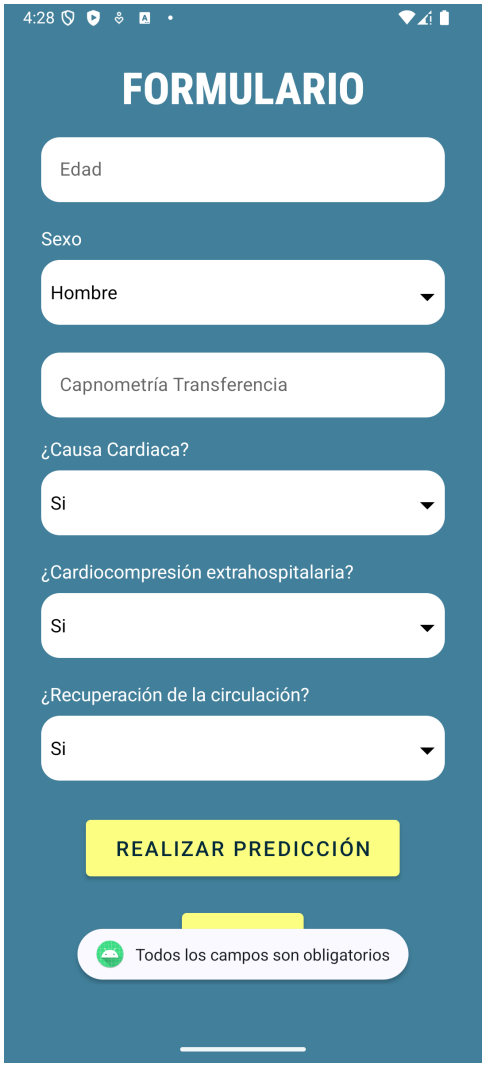


Figura 9.23: Captura de pantalla: Campos obligatorios

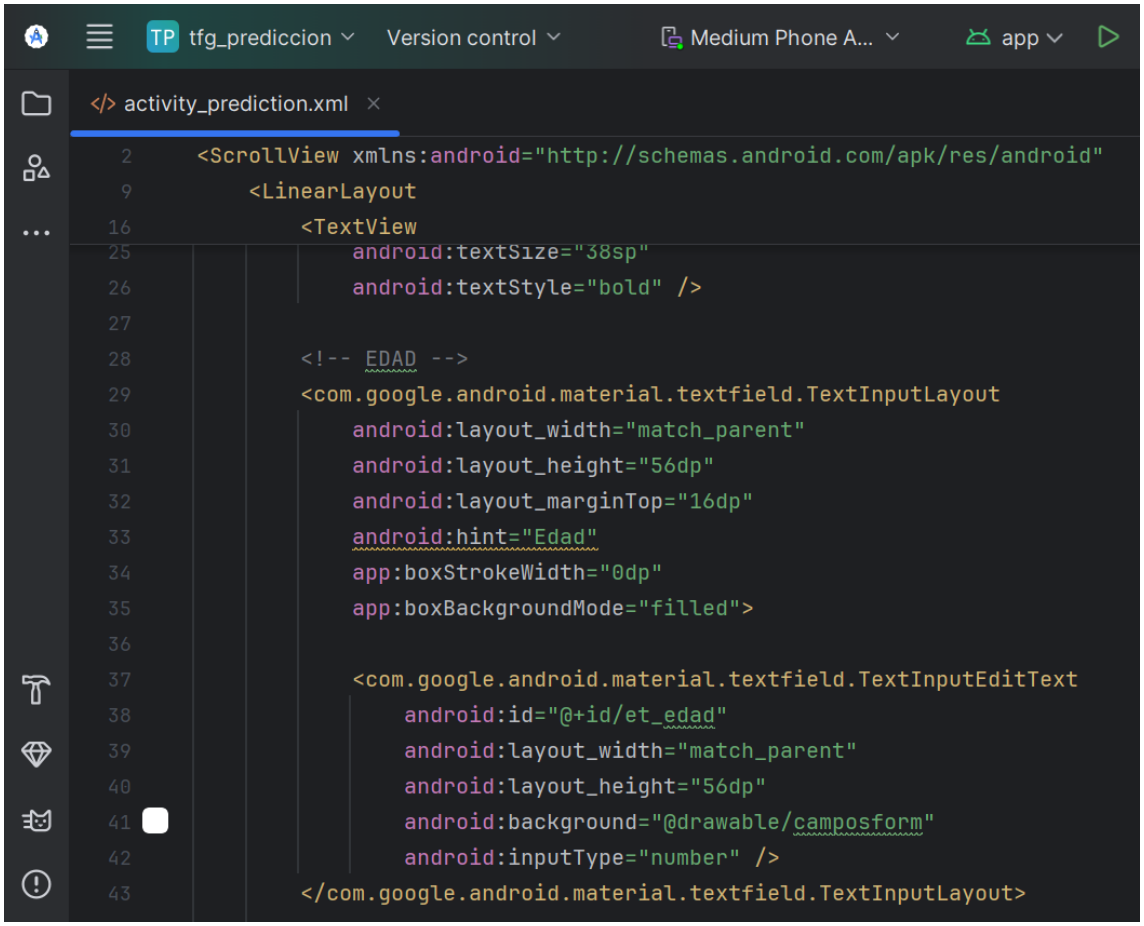


Figura 9.24: Captura de pantalla: Código XML input type

La lógica para decidir si un donante es válido o no sigue los siguientes pasos:

1. Coger los valores de los campos con el binding (igual que en el registro y en editar perfil)
2. Comprobar que no son Null. Si no es así se muestra un Toast indicando que todos los campos deben ser obligatorios
3. Calcular el resultado con la fórmula matemática de la tesis doctoral
4. Dependiendo del resultado, se navega a una pantalla u otra

Estos pasos se pueden ver en las figuras 9.25, 9.26 y 9.27.

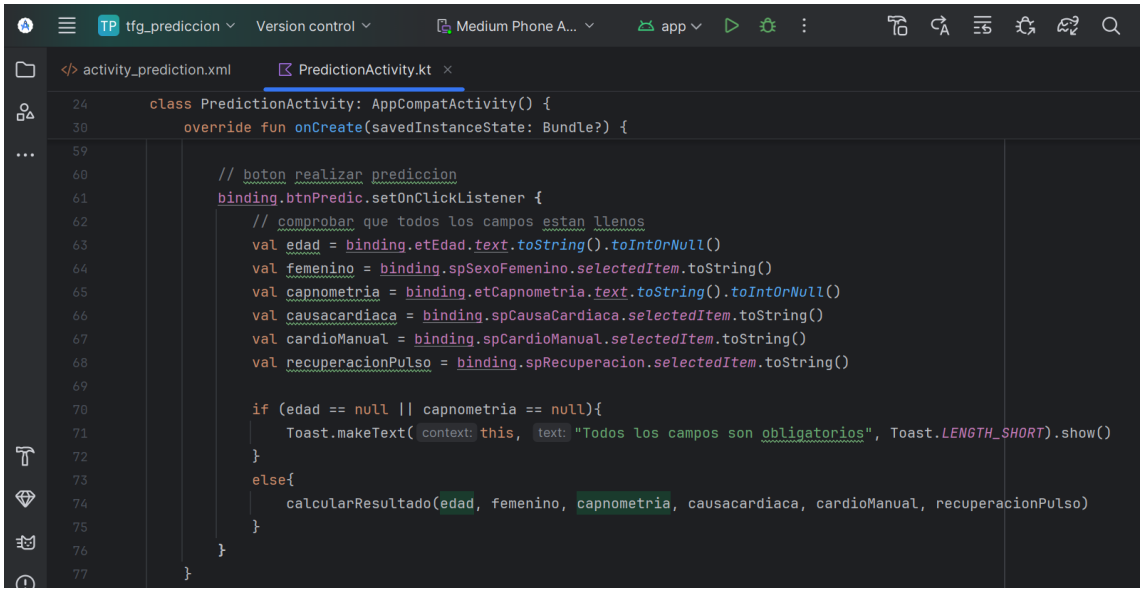


Figura 9.25: Captura de pantalla: Código realizar predicción (1)

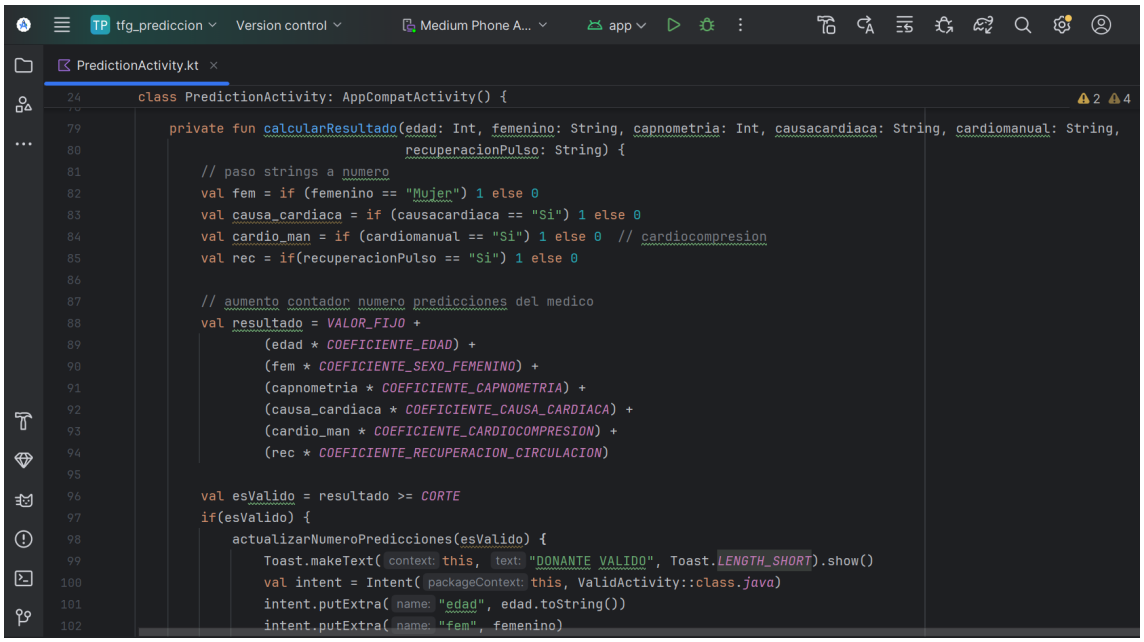


Figura 9.26: Captura de pantalla: Código realizar predicción (2)

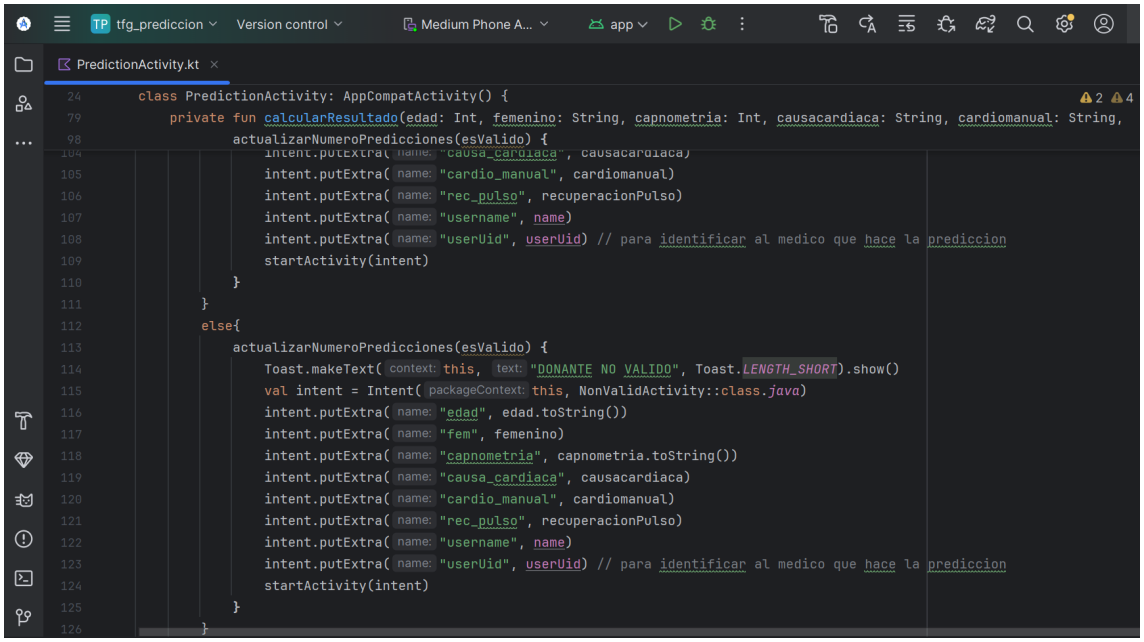


Figura 9.27: Captura de pantalla: Código realizar predicción (3)

Tras realizar la predicción, se navega a una pantalla diferente dependiendo de si el resultado es Válido o No válido. En ambas aparece una notificación y el resultado con su color correspondiente: verde para las válidas y rojo para las no válidas. Además, aparecen dos botones, uno para descargar un PDF con el informe de la predicción (figura 9.30) y otro para volver al menú principal.

Figura 9.28: Captura de pantalla: Donante válido

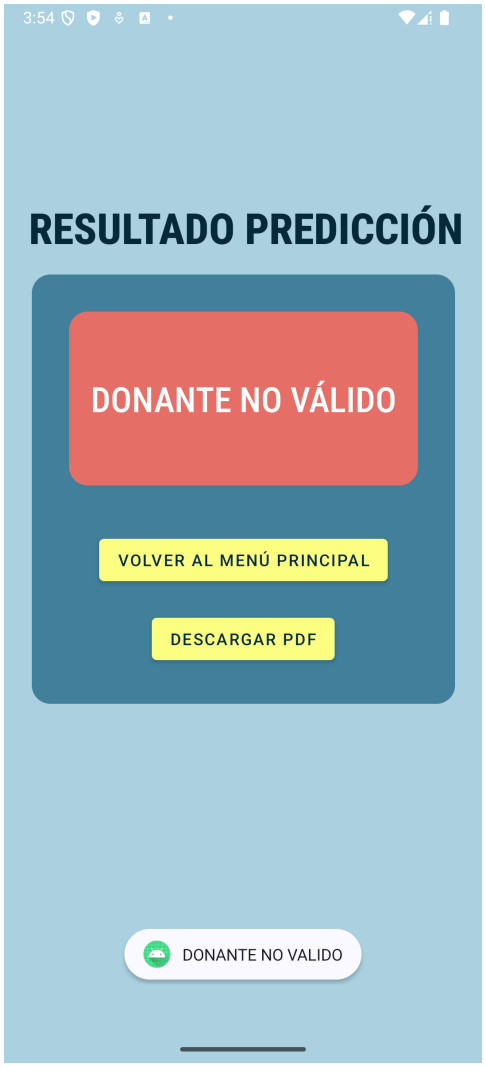


Figura 9.29: Captura de pantalla: Donante no válido

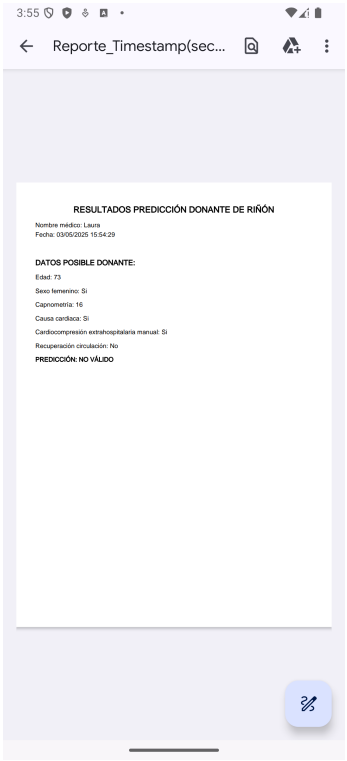


Figura 9.30: Captura de pantalla: PDF generado

La lógica para la pantalla de un donante válido y no válido es la misma. En la función *onCreate* se implementan las funciones básicas de los componentes. La más relevante en este caso es la de crear el PDF con los resultados de la predicción. Para conseguir esto, primero se compueba que la versión Android del dispositivo sea igual o superior a 10. Después se llama a la función *generarPDF* que se muestra en la captura de las figuras 9.31 y 9.32. Esta función crea un nombre único para el archivo (que incluye la fecha) y prepara los metadatos para que el PDF se guarde automáticamente en la carpeta *Descargas* del dispositivo. Para esto, se usa la clase *ContentValues* junto con el *ContentResolver* de Android, que permite crear el archivo de manera compatible con el sistema. Una vez creado el archivo y su uri, se abre un flujo de salida hacia ese archivo. A través de la librería *iText PDF*, se crea un objeto *PdfDocument* y un *Document* asociado. Dentro del PDF se estructura el contenido añadiendo todos los elementos necesarios (nombre del médico, fecha, valores del donante, etc) y, finalmente, se cierra el documento, se muestra un mensaje con un Toast indicando que el PDF se ha guardado correctamente, y se llama a un método *abrirPDF* para abrir el archivo automáticamente. En caso de que se produzca algún error durante el proceso, se captura la excepción y se informa al usuario mediante un Toast.

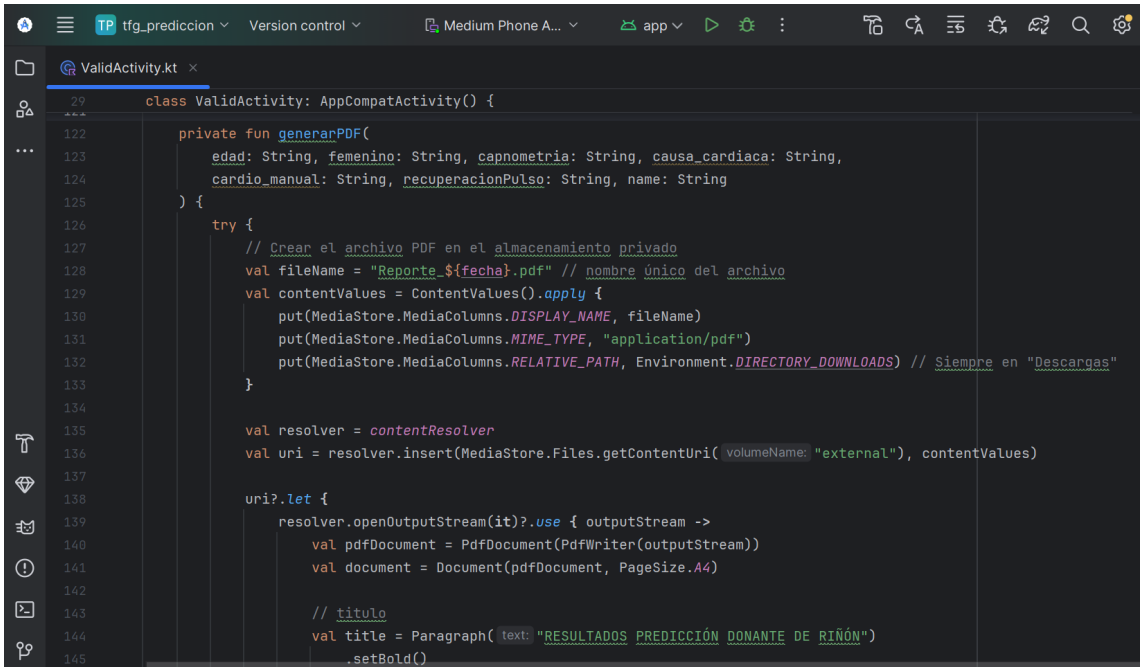


Figura 9.31: Captura de pantalla: Código generar PDF (1)

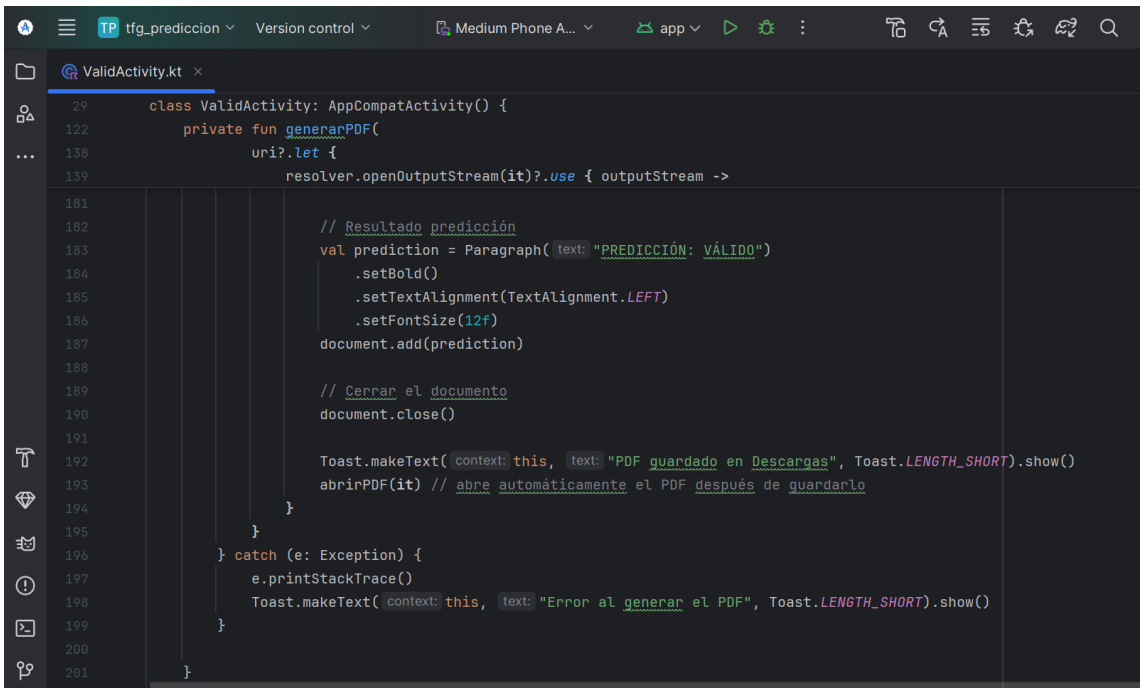


Figura 9.32: Captura de pantalla: Código generar PDF (2)

9.1.3.2. Historial

Volviendo al menú principal, el usuario puede pulsar el botón de *Historial de predicciones*. Este botón navega a la pantalla que muestra la figura 9.33, donde aparecen las predicciones realizadas por el usuario ordenadas según su fecha. Podemos

determinar el orden de las predicciones utilizando el spinner, en el que las opciones son: Fecha, Edad y Capnometría. Para cada predicción encontramos las siguientes columnas: *Edad*, *Sexo Femenino*, *Capnometría*, *Muerte por causa cardiaca*, *Cardiocompresión*, *Recuperación del pulso*, *Resultado*, *Informe* y *Eliminar*. Pulsando el icono de *Informe* se descarga un nuevo PDF con los datos de la predicción. Por otro lado, pulsando el icono de *Eliminar* aparece un diálogo de confirmación que, si es aceptado, eliminará la predicción de la base de datos y actualizará los contadores.

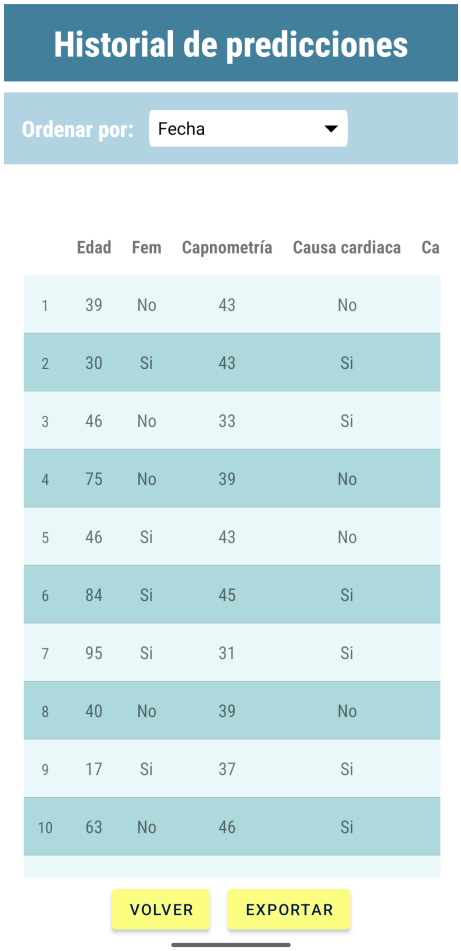


Figura 9.33: Captura de pantalla: Historial

En cuanto a la lógica, el aspecto más relevante en esta pantalla recae en que se puede eliminar la predicción seleccionada. Para conseguir esto se crea un *event listener* por cada icono de basura que, al activarse, muestra un diálogo de confirmación como se ve en la figura 9.34. Al confirmar el diálogo se elimina la predicción de la base de datos y se actualizan los contadores del médico (el número de predicciones que lleva).

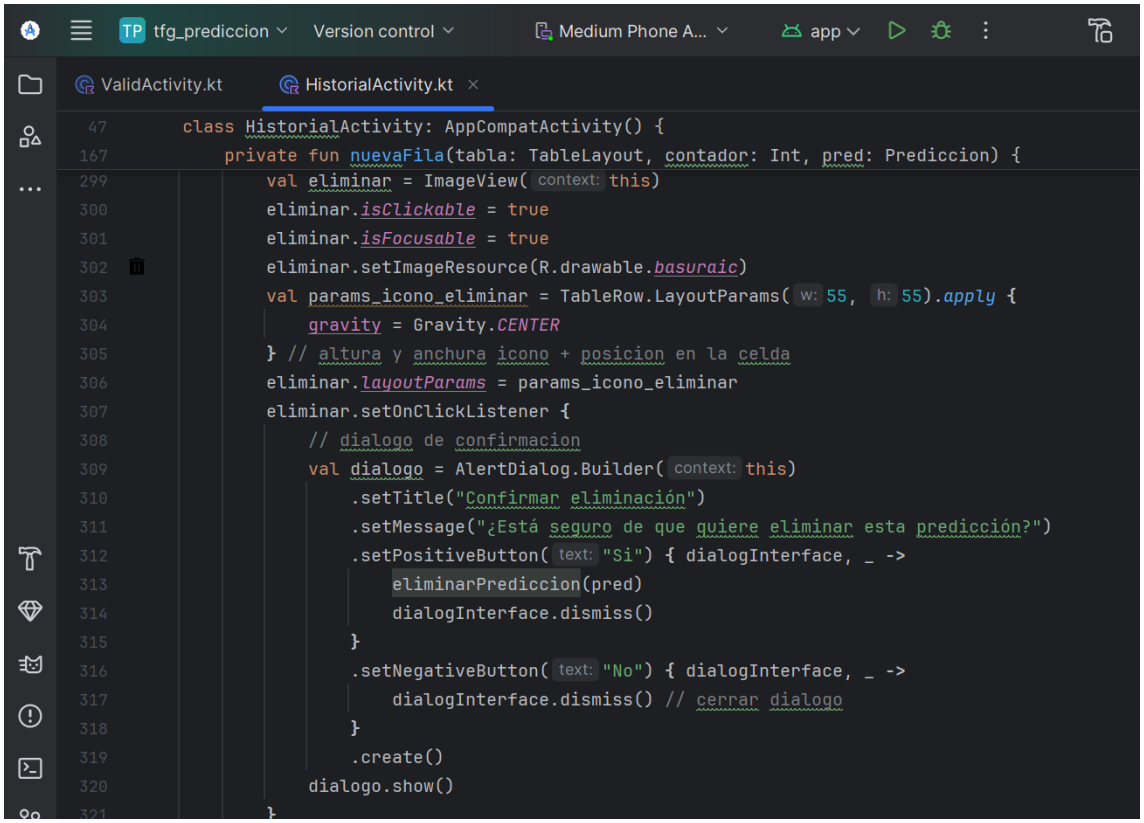


Figura 9.34: Captura de pantalla: Código para eliminar una predicción

9.1.3.3. Importar predicciones

Para terminar, el tercer botón del menú principal es el de *Importar predicciones*. Este botón navega a una pantalla simple que pide al usuario seleccionar un archivo CSV con el formato indicado. Tras seleccionar el archivo, la lógica de la actividad lee cada línea y añade la predicción a la base de datos. Si se ha completado la tarea con éxito, aparece un Toast de éxito y la barra se completa, como se ve en la figura B.21.

Figura 9.35: Captura de pantalla: Importar archivo CSV

Figura 9.36: Visualización del CSV importado correctamente

En el código podemos destacar el siguiente fragmento de código mostrado en la figura 9.37. En este fragmento se puede ver la función *selectorArchivos* que utiliza la API moderna de AndroidX llamada *registerForActivityResult*, que permite lanzar una actividad externa y recibir un resultado. En concreto, esta función está configurada para abrir un selector de archivos en el dispositivo, que inicia una actividad esperando que el usuario elija un archivo y devuelva ese resultado. Cuando el usuario selecciona un archivo y confirma la acción, se ejecuta la función lambda que maneja el resultado. Si el resultado indica que la operación fue exitosa, se muestra un mensaje Toast en la pantalla para informar que el archivo CSV ha sido importado correctamente. A continuación, se obtiene el Intent con los datos devueltos por la actividad, de donde se extrae la URI que identifica al archivo CSV seleccionado dentro del sistema de archivos. Si la URI es válida, se llama a la función leerCSV, que lee línea a línea el archivo y añade la predicción a la base de datos.

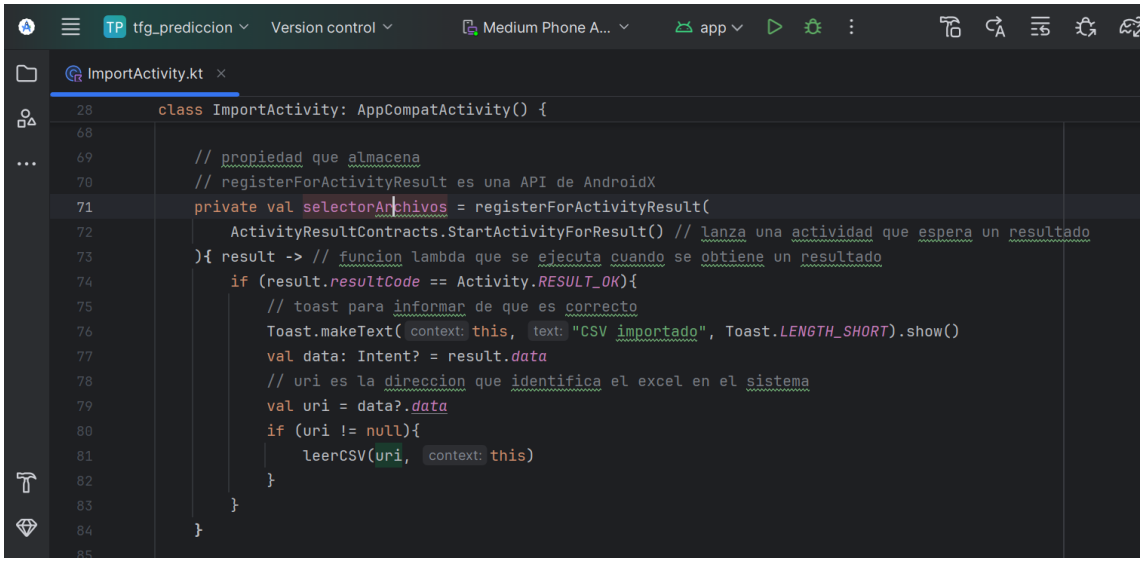


Figura 9.37: Captura de pantalla: Código para que el usuario seleccione el archivo que desea importar

9.1.4. Módulo de Usuario: Administrador

Al iniciar sesión como Administrador aparece la pantalla principal correspondiente, como se muestra en la figura 9.38. Esta consta de una tabla con todos los médicos loggeados en la aplicación, mostrando sus apellidos y correo electrónico. Al presionar sobre el icono de la columna *Preds*, se navega a la pantalla mostrada en la figura 9.39 donde aparecen todas las predicciones realizadas por el correspondiente médico.

Figura 9.38: Captura de pantalla: Pantalla principal Administrador

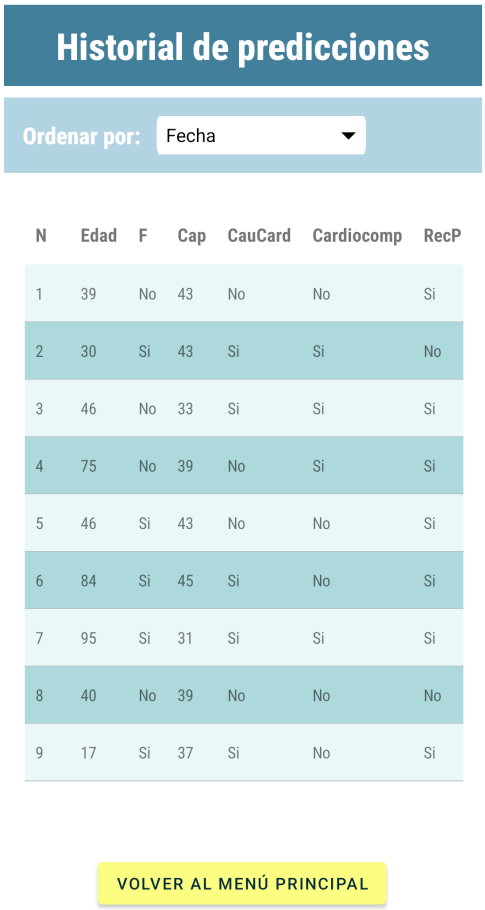


Figura 9.39: Captura de pantalla: Predicciones realizadas por Médico

En cuanto al código, este no tiene nada relevante en comparación con los apartados anteriores. Tan solo se hace una consulta a la base de datos para conseguir los datos de cada médico y sus predicciones.

Los administradores no tienen más funcionalidades en la aplicación móvil ya que su rol está enfocado a las funciones de la aplicación web, donde pueden acceder a la base de datos para entrenar modelos y visualizar sus resultados.

9.2. Aplicación web

9.2.1. Estructura

En Visual Studio Code se tiene el proyecto *app_ web* que se divide en dos carpetas principales: *backend* y *frontend*.

Backend contiene los archivos en Python destinados al entrenamiento y obtención de resultados de los modelos de Machine Learning. Su estructura se divide en tres

scripts:

- *app.py*: contiene dos rutas POST (entrenar y prediccion) creadas con Flask que reciben datos JSON desde el frontend.

  Cada vez que se desean añadir predicciones nuevas al dataset de cierto modelo, se llama desde el frontend a la función *entrenar*, que recibe el modelo que se desea entrenar y una lista con las predicciones nuevas. Esta función añade las predicciones al CSV correspondiente y entrena el modelo.

  Por otro lado, cada vez que se desean obtener los resultados de ciertos modelos dados unos valores de entrada, se llama desde el frontend a la función *prediccion*, que recibe una lista de los modelos de los que se devolverá un resultado y los valores de donante.

- *regresion_logistica.py*: contiene la lógica del entrenamiento del modelo de regresión logística.

- *clustering.py*: contiene la lógica para agrupar las predicciones en clusters. Se implementa el código necesario para aplicar el Algoritmo del Codo para determinar el número óptimo de clusters y, a partir de este resultado, aplica K-Prototypes.

Frontend contiene los archivos Javascript necesarios para mostrar las interfaces de inicio de sesión, visualizar las tablas y presentar los resultados de los modelos. También se incluyen en esta carpeta los archivos *.html* y *.css* que acompañan a cada pantalla para estructurar el contenido y definir el estilo de la interfaz.

También en esta carpeta se incluye el archivo *configuraciondb.js* que contiene las credenciales necesarias para conectarse a la base de datos Firestore.

9.2.2. Módulo de Usuario: Administrador

9.2.2.1. Iniciar Sesión

Al abrir la aplicación web nos encontramos la pantalla de inicio de sesión como la mostrada en la figura 9.40.

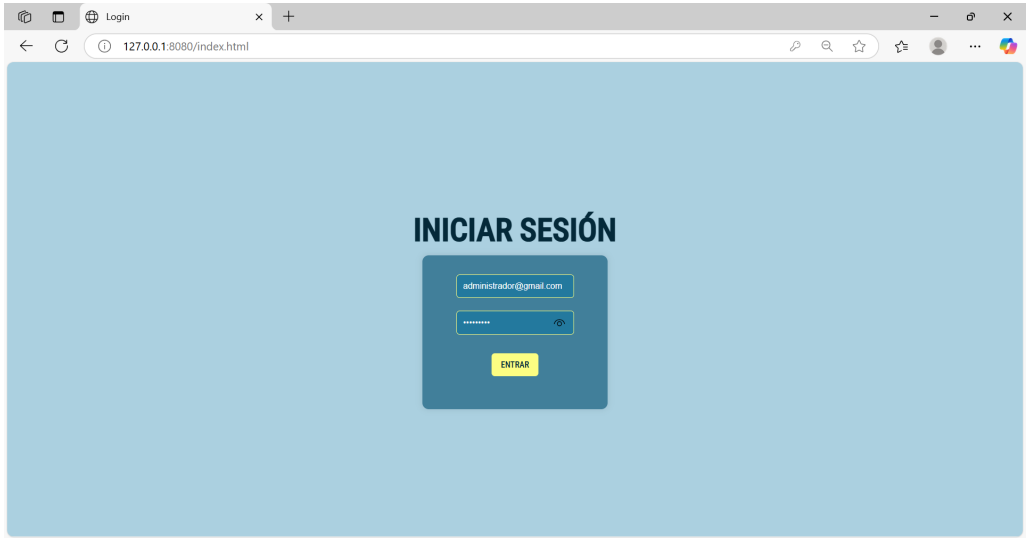


Figura 9.40: Captura de pantalla: Login web

Este formulario se define en el archivo *index.html* como *loginForm* (figura 9.41). También se define un botón de typo submit, cuya lógica tras ser pulsado se programa en el archivo *index.js* (figura 9.42). En este listener se comprueba con el método *signInWithEmailAndPassword* de Firebase Authentication que la cuenta existe en la base de datos y devuelve una credencial de la misma. Con esta, se accede al *uid* del usuario y se obtiene el documento de Firestore que corresponde al usuario. Si este existe, se comprueba que su rol es de Administrador (ya que es el único que puede acceder), y se navega a la pantalla principal.

Al realizar las consultas en la base de datos se utiliza *await*, una palabra clave en Javascript que se utiliza para esperar a que se complete una operación asíncrona antes de continuar ejecutando el código. (7)

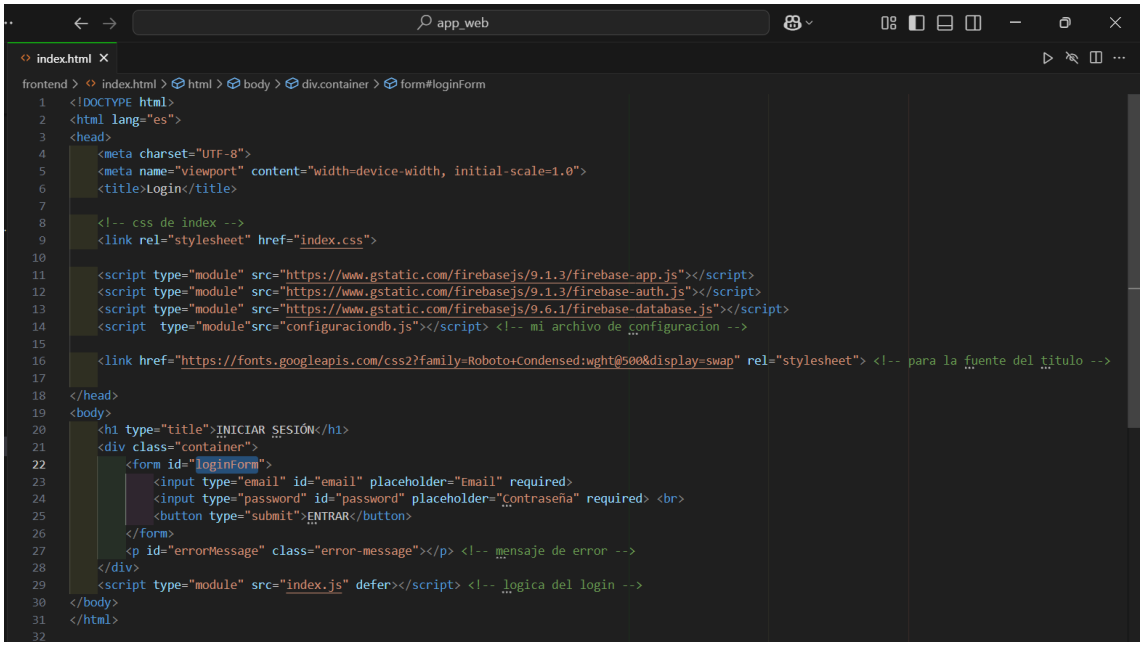


Figura 9.41: Captura de pantalla: Código html de index

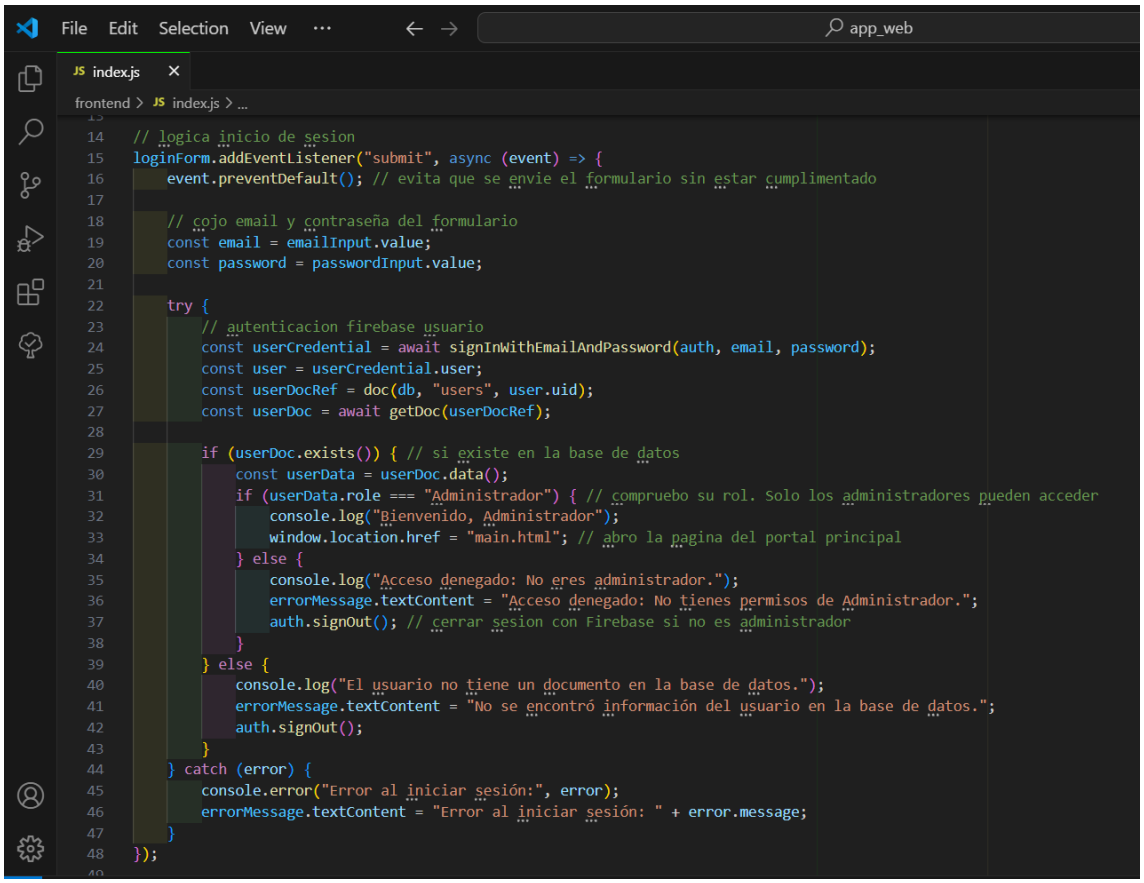


Figura 9.42: Captura de pantalla: Código Javascript de index

9.2.2.2. Menú principal

Una vez se inicia sesión, se navega a la pantalla principal. En esta aparece una tabla con las predicciones realizadas por todos los médicos. Cambiando de opción con el spinner, se pueden ver las predicciones que forman parte del dataset del modelo seleccionado. Las opciones del spinner son: *Todas*, *Árbol de decisión*, *Random Forest*, *Regresión Logística* y *Clustering*.

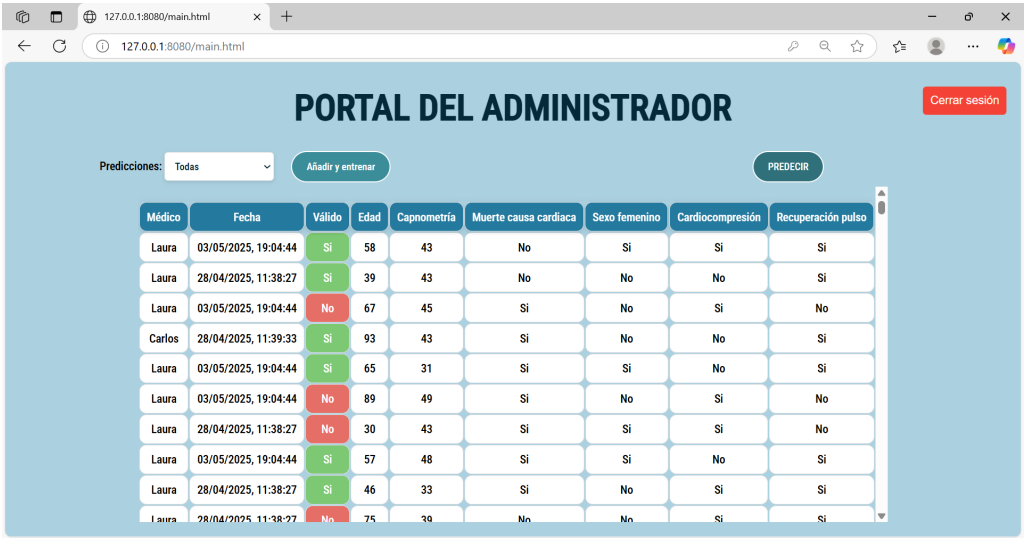


Figura 9.43: Captura de pantalla: página principal con todas las predicciones

El código de esta pantalla es bastante sencillo ya que tan solo muestra las predicciones en un formato de tabla. No obstante, en la figura 9.44 podemos ver la función más importante del script: *cargar_predicciones*. Esta es una función asíncrona ya que depende de Firestore para obtener los datos necesarios, por lo que el código debe esperar a recibir estos datos para continuar. Estas consultas van precedidas, al igual que en el scrip anterior, por la palabra clave *await*. Sin incluir estas dos palabras: *async* y *await*, la función no respondería correctamente.

En la captura se puede ver que lo primero que se hace es comprobar el valor seleccionado en el spinner para así filtrar según lo especificado. Cuando se selecciona un modelo en concreto, se utiliza en la consulta a la base de datos la función *arraycontains*, que devuelve de forma eficiente si una predicción se encuentra en ese modelo o no. Si corresponde, el documento se incluye en el *querySnapshot*, que luego se procesa para generar una lista que se pasa a la función encargada de mostrar las predicciones en la interfaz.

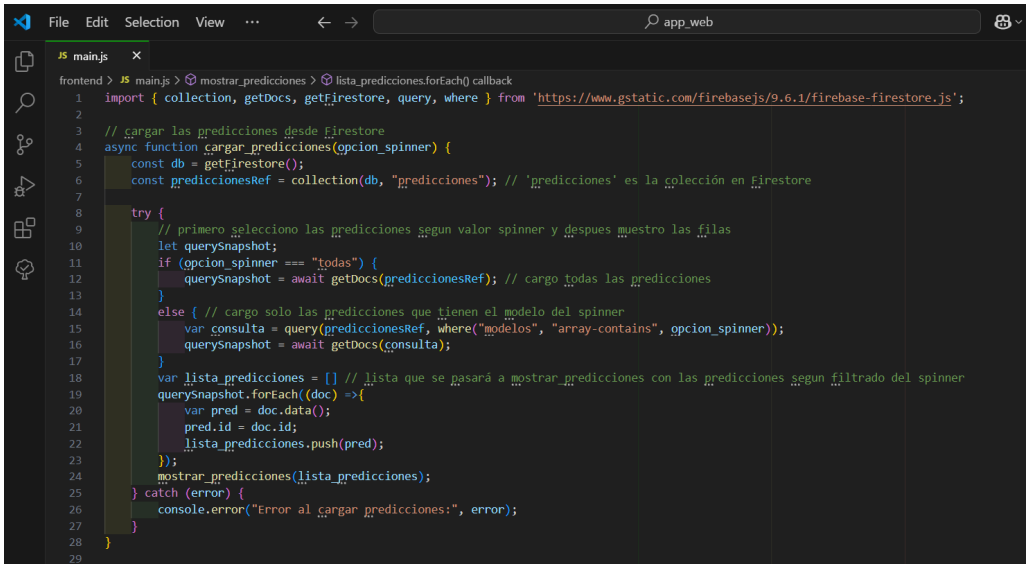


Figura 9.44: Captura de pantalla: código del main

9.2.2.3. Añadir y Entrenar

Desde la página principal, si pulsamos el botón *Añadir y entrenar*, navegamos a la pantalla mostrada en la figura 9.45, donde se muestran las predicciones que forman parte de la base de datos pero no pertenecen al dataset del modelo seleccionado por el spinner.

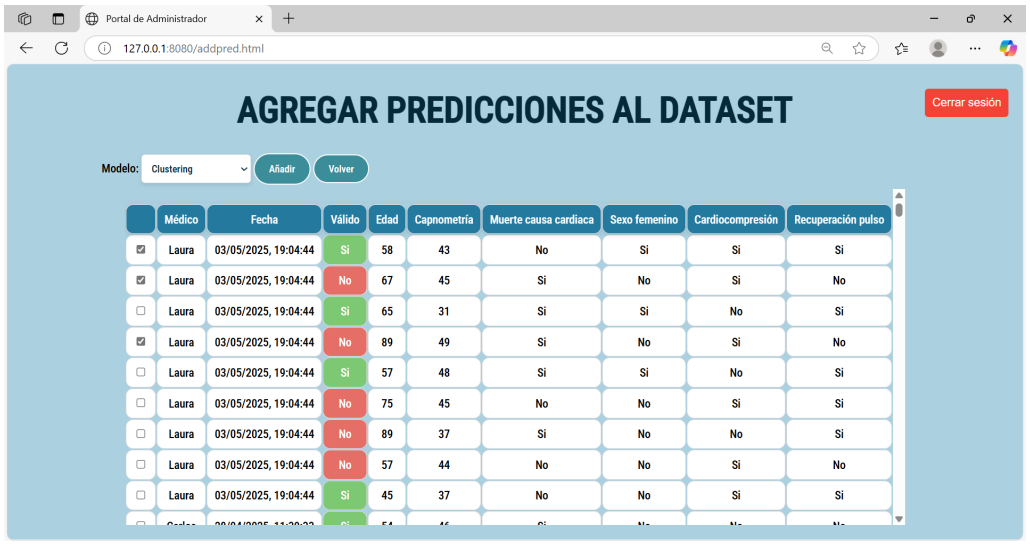


Figura 9.45: Captura de pantalla: tabla añadir predicciones

El usuario debe seleccionar las predicciones que desea añadir al dataset de entrenamiento del modelo. Cuando se pulsa el botón *Añadir*, se comprueba que haya predicciones seleccionadas y, si así es, se muestra un diálogo de confirmación y se añaden las predicciones al dataset del modelo correspondiente.

En el código se sigue la misma dinámica que en el apartado anterior, realizando consultas a la base de datos y mostrando las predicciones. Un vez se pulsa el botón

de Añadir, se debe añadir el nuevo modelo al array *modelos* de la base de datos de la predicción correspondiente y eliminarlo del array *no_ modelos*. Esto se consigue de forma eficiente gracias a que, en vez de llamar una vez a la base de datos por cada predicción, se utiliza un batch. Esto es una forma de agrupar múltiples operaciones de escritura para que se ejecuten todas juntas de forma atómica (figura 9.46).

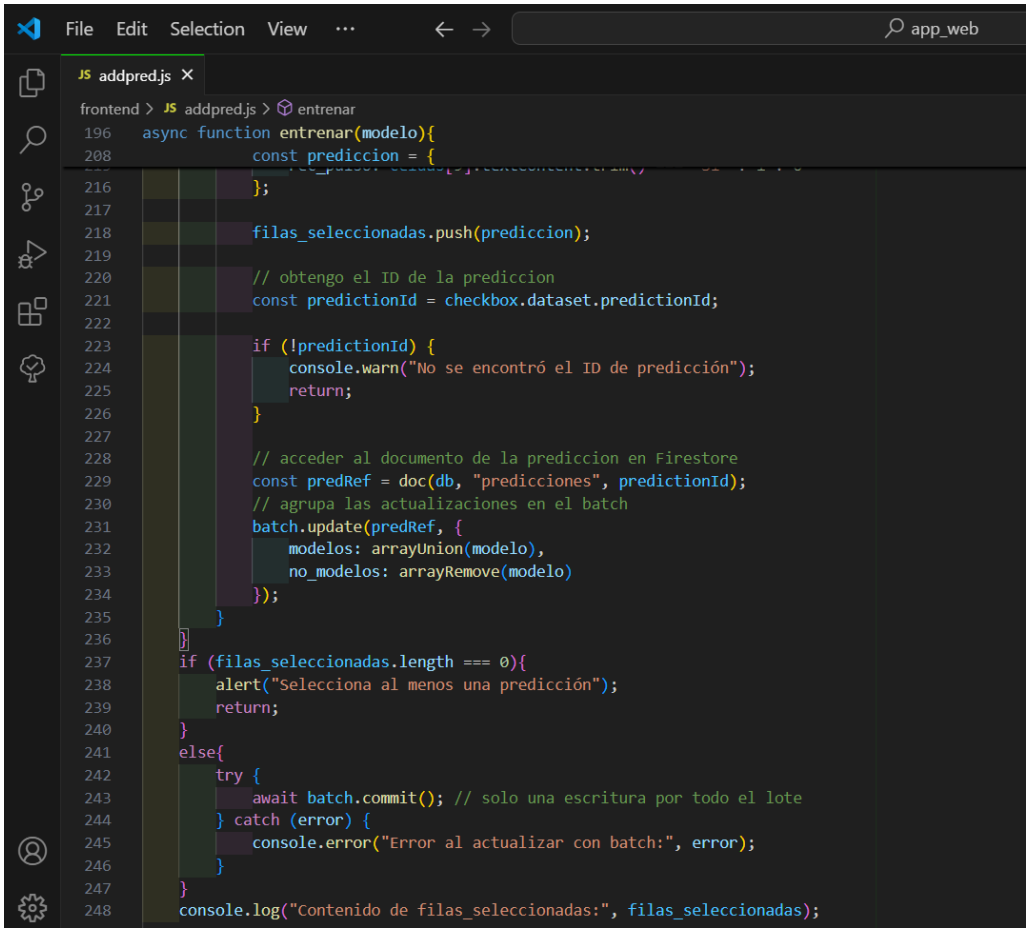


Figura 9.46: Captura de pantalla: código batch

A continuación, se envía el modelo y la lista de predicciones al backend para entrenar el modelo. Esto se consigue realizando una petición HTTP de tipo POST a la API alojada en localhost en el puerto 5000, enviando los datos en formato JSON a la función de Flask */entrenar*. La función fetch se utiliza junto con await para esperar de forma asíncrona la respuesta del servidor. Cuando la API responde, se convierte la respuesta en un objeto JSON y se comprueba si la respuesta contiene un mensaje de éxito. En caso afirmativo, se muestra una alerta informando que el modelo ha sido entrenado correctamente y se muestra un informe detallado en la interfaz mediante la función *mostrarModalInforme*. Esta lógica se puede ver en la siguiente captura de pantalla.

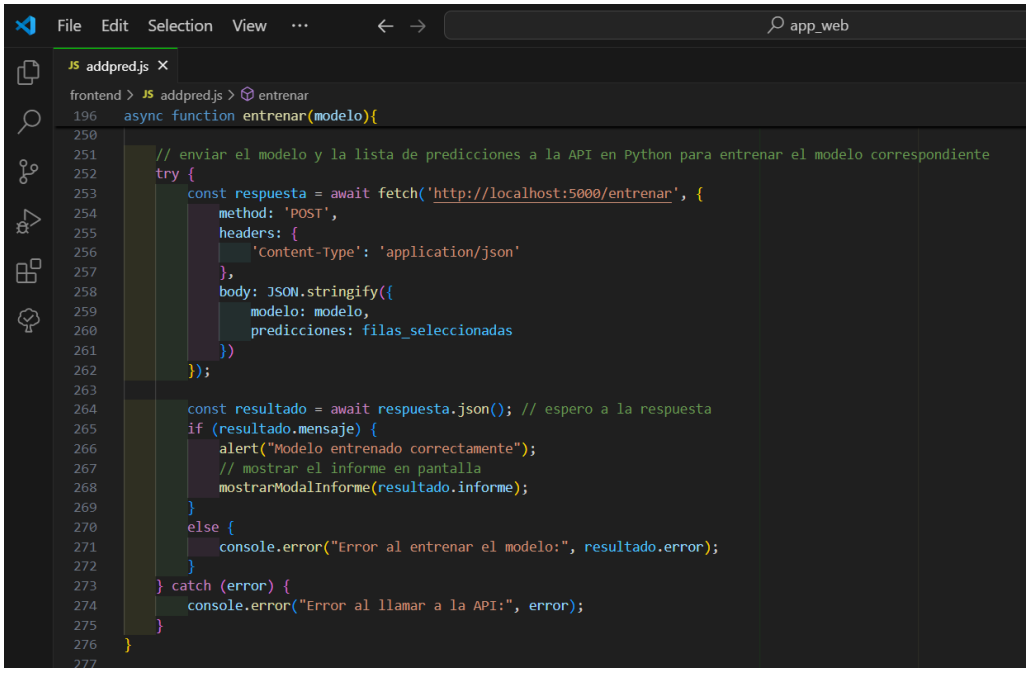


Figura 9.47: Captura de pantalla: código envío backend

Por otro lado, el código en el backend es el mostrado en la figura 9.48. En este obtenemos los datos en forma de JSON y, dependiendo del modelo, llamamos a la función correspondiente para entrenarlos. Al terminar, se devuelve en formato JSON el mensaje de éxito, el modelo y el informe con los resultados a mostrar al frontend. Para poder aceptar la solicitud del frontend en javascript, se ha utilizado *CORS(app)* que permite aceptar solicitudes de cualquier origen.

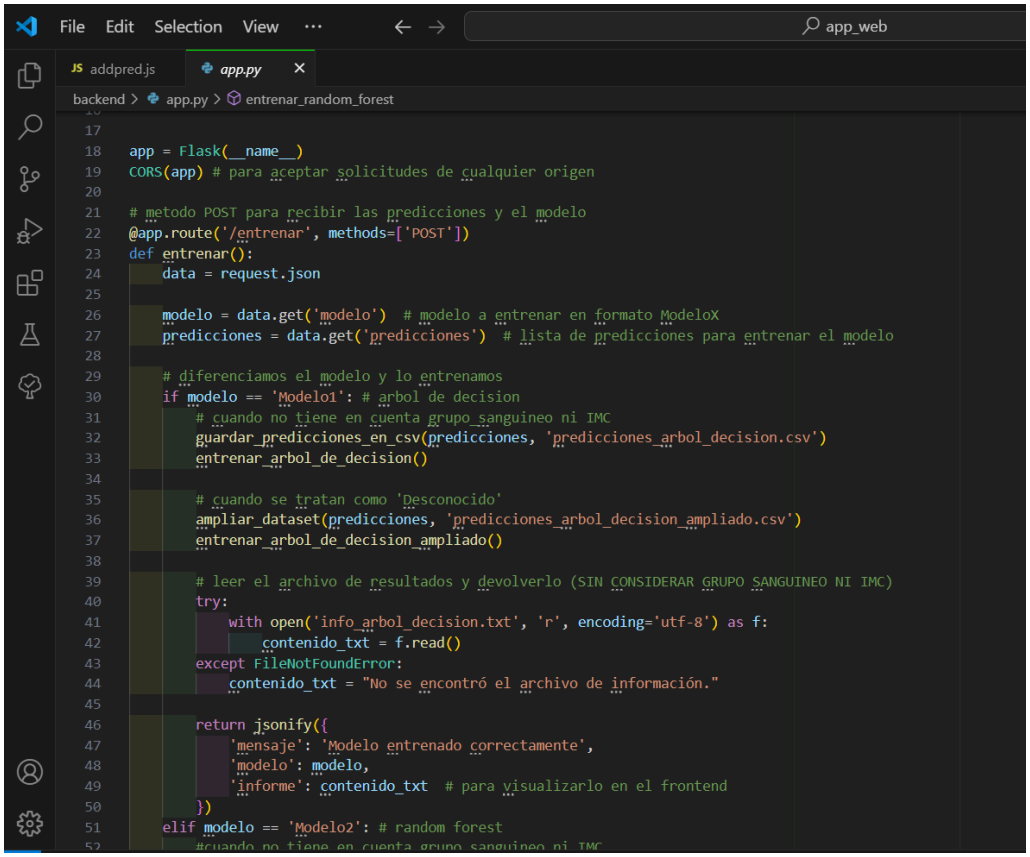


Figura 9.48: Captura de pantalla: código en backend para entrenar modelos

En el modal que se muestra con la información se puede descargar el informe en formato *.txt* pulsando *descargar .txt*. Se pueden encontrar ejemplos de los cuatro modelos de archivos .txt en textiteste enlace.

Para conseguir descargar este informe, se implementa el código mostrado en la figura 9.49. Este código obtiene el contenido del informe utilizando el id del modal correspondiente al html. Después, se genera un objeto Blob, que es un tipo de objeto que representa datos como archivos. En este caso, el tipo se especifica como *text/plain*, indicando que es un archivo de texto. A continuación, se utiliza *URL.createObjectURL()* para crear una URL especial que permite acceder al Blob recién creado (esta URL es válida de forma temporal en el navegador). El siguiente paso es crear un enlace al que se le asigna la URL generada y el nombre del archivo a descargar. Luego, se simula un click en este enlace para iniciar la descarga del archivo en el navegador. Por último, se elimina el enlace y se libera la URL generada usando *URL.revokeObjectURL()* para evitar problemas de memoria.

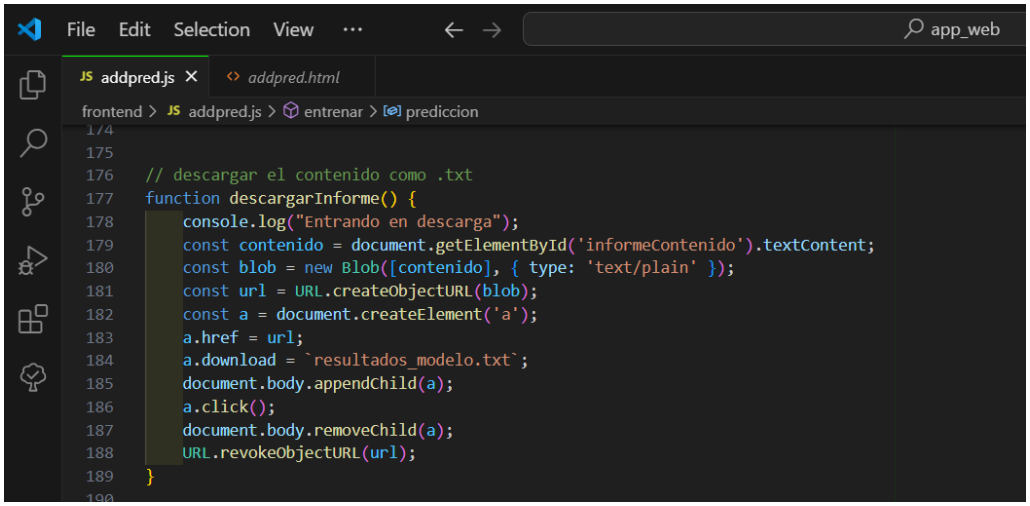


Figura 9.49: Captura de pantalla: código descargar informe de entrenamiento

9.2.2.4. Predecir

La última funcionalidad del usuario en la aplicación web es poder realizar predicciones y comparar resultados. Para ello, el usuario puede pulsar el botón *PREDECIR* de la página principal (figura 9.43 mostrada anteriormente) para rellenar un formulario con los datos de un posible donante y poder comparar los resultados de los diferentes modelos.

En la figura 9.50 se puede ver la interfaz con el formulario. Además, el usuario puede seleccionar los modelos cuyos resultados desea visualizar, con el fin de compararlos con los obtenidos con la fórmula de la tesis doctoral.

Figura 9.50: Captura de pantalla: formulario

Una vez se rellenan los campos y se seleccionan los modelos, la aplicación envía al backend esta información, se calcula el resultado de cada modelo y se envían los resultados al frontend. Esto se consigue de la misma forma que en el apartado an-

terior: realizando una petición HTTP de tipo POST a la API alojada en localhost en el puerto 5000, enviando los datos en formato JSON a la función de Flask */prediccion*. En esta función del backend se recibe el modelo seleccionado y los datos de la predicción. Dependiendo del modelo se obtienen los diferentes resultados, que se guardan en un JSON que se devolverá al frontend. Esta lógica se puede ver en la siguiente figura.

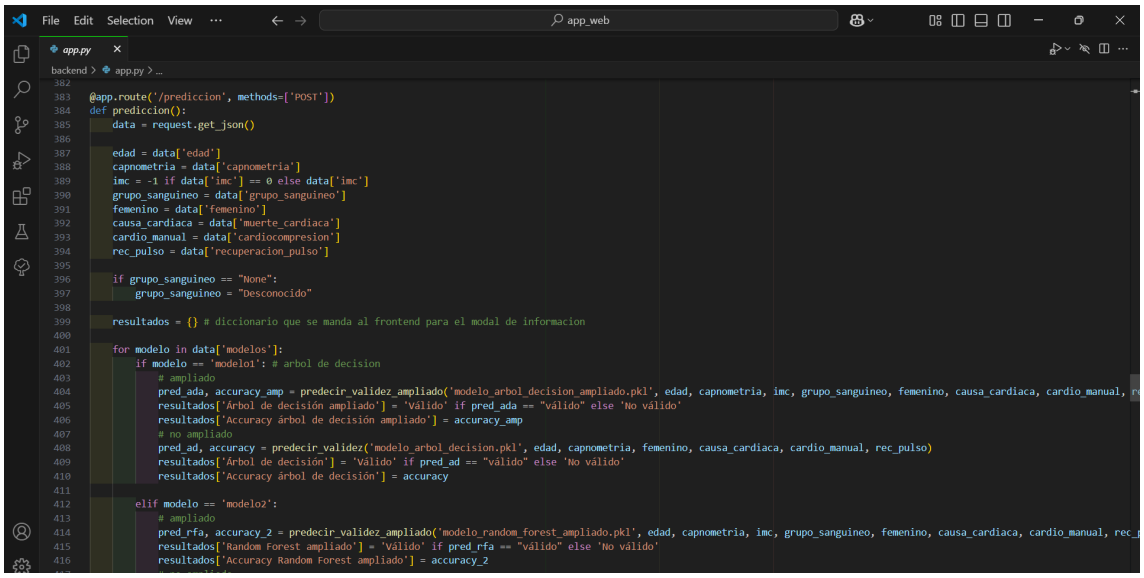


Figura 9.51: Captura de pantalla: código en backend para obtener resultados

Con estos resultados en el frontend, se carga la pantalla mostrada en las figuras 9.52 y 9.53, donde para cada modelo se indican los resultados obtenidos en el JSON del backend. También aparece el resultado que da la fórmula de la tesis doctoral.

Esta funcionalidad tiene como propósito ver con qué tipo de predicciones los modelos coinciden en sus resultados, para así en un futuro poder determinar qué modelo va mejor según los datos de entrada.

Figura 9.52: Captura de pantalla: resultados 1

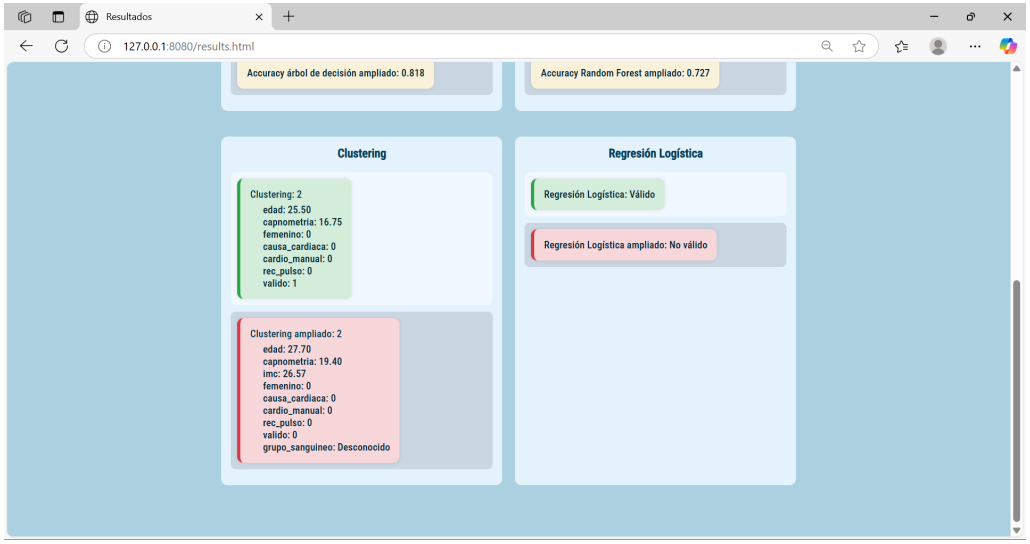


Figura 9.53: Captura de pantalla: resultados 2

Capítulo 10

Ética: Inteligencia Artificial aplicada a la Medicina

En 2021, la Organización Mundial de la Salud publicó el primer informe mundial sobre inteligencia artificial aplicada a la salud. Este informe incluye seis principios rectores relativos a su concepción y utilización, con el fin de limitar los riesgos y aumentar al máximo las oportunidades que conlleva la utilización de la IA en el ámbito de la salud. Estos principios son los siguientes: (38)

1. Preservar la autonomía del ser humano
2. Promover el bienestar y la seguridad de las personas y el interés público
3. Garantizar la transparencia, la claridad y la inteligibilidad
4. Promover la responsabilidad y la rendición de cuentas
5. Garantizar la inclusividad y la equidad
6. Promover una IA con capacidad de respuesta y sostenible

Este trabajo ha sido desarrollado siempre con la idea de cumplir con estos principios éticos. Además, los resultados de los modelos implementados no buscan sustituir la decisión humana, sino servir de herramienta adicional de apoyo para el personal sanitario. Este siempre será el encargado de tomar la última decisión sobre la viabilidad de un donante. Asimismo, en ningún momento se almacenan datos personales que identifiquen al donante, ya que solo se guarda la información necesaria para entrenar los modelos, tratando de respetar así la privacidad de los donantes.

Al estar tratando un tema tan delicado como es la salud, y enfocándonos más en los trasplantes, es imprescindible no olvidarse de que la tecnología no avance sin tener en cuenta la reflexión ética. No hay duda de que la inteligencia artificial puede ser una herramienta muy útil, pero se debe usar siempre con responsabilidad, transparencia y de forma justa. Es importante destacar que la aplicación web desarrollada en este trabajo debe utilizarse, en este momento, con fines de investigación y por curiosidad académica. La base disponible con la que se han entrenado los modelos es muy pequeña para poder ser utilizada correctamente. Así, su objetivo principal es permitir analizar cómo evolucionan las predicciones de los modelos y la importancia de las diferentes variables a medida que se incorporan más datos.

Capítulo 11

Conclusiones y Trabajo Futuro

11.1. Conclusiones del trabajo

El objetivo de este Trabajo Fin de Grado era desarrollar una aplicación móvil que cumpliese con los requisitos establecidos. Estos requisitos engloban, principalmente, la idea de una aplicación a la que se pueda acceder de forma rápida para poder comprobar si un donante es válido o no. A partir de esta aplicación se llegó a la idea de desarrollar un aplicación web paralela que permitiese entrenar diferentes modelos de Machine Learning para comparar los resultados con las predicciones realizadas en la aplicación móvil. De esta forma, se trata de acercar al personal sanitario a nuevas herramientas tecnológicas que faciliten la exploración de datos y realizar diagnósticos con métodos modernos.

En cuanto a la aplicación web, al tener un dataset tan pequeño (69 registros), no se han podido entrenar los modelos predictivos con un nivel de fiabilidad alto. Sin embargo, los resultados obtenidos permiten tener una primera aproximación al comportamiento de los donantes que, en un futuro, evolucionará hacia un modelo mucho más fiable. Lo más relevante ha sido el análisis de clustering, que ha logrado agrupar a los donantes de una forma significativa y que no se había considerado hasta ahora. Aunque no podemos asegurar que estos resultados sean 100 % correctos o representativos debido al tamaño limitado de la muestra, sí constituyen una base útil para explorar patrones iniciales y guiar futuros desarrollos con datasets más amplios.

Tras haber finalizado la aplicación, se han cumplido con los propósitos establecidos. Se puede encontrar el código de ambas aplicaciones en el siguiente enlace:

https://TFGLauraHerrero

Para terminar, es importante enfatizar que a medida que se utilice la aplicación web, las predicciones que se utilizarán para entrenar los modelos estarán sesgados según el resultado de la fórmula de la tesis. Esto no supone un problema importante ya que esta fórmula ha sido el resultado de una tesis doctoral cuidada y precisa, y de la que tenemos la certeza de que es correcta.

11.2. Líneas de trabajo futuro

11.2.1. Ampliar base de datos

Como se indicó en secciones anteriores, el dataset inicial empleado para el entrenamiento de los modelos cuenta con pocos registros. Es esencial que, según se vaya utilizando la aplicación móvil, se vayan entrenando los modelos con estas predicciones. Cuantos más registros haya, más fiable será el modelo y mejores conclusiones se podrán sacar según los resultados.

11.2.2. Añadir resultados reales

Según se vaya utilizando la aplicación, los resultados estarán sesgados por la fórmula de la tesis doctoral. Aunque estos resultados sean fiables, sería conveniente añadir un apartado para las predicciones que se pudiese modificar con el resultado real sobre si la donación fue exitosa o no. De esta forma, los modelos se entrenarían con estas predicciones y no habría ningún tipo de sesgo.

11.2.3. Aplicación móvil para iOS

Al haber desarrollado la aplicación con Android Studio, esta está solo disponible para móviles con este sistema operativo. Así, sería conveniente desarrollar la aplicación para que esté disponible en otros sistemas operativos, como iOS, para ampliar el alcance de la herramienta y facilitar su uso a un mayor número de usuarios. Para esto sería necesario reescribir la aplicación utilizando un framework multiplataforma, o desarrollar una versión específica para iOS.

11.2.4. Mejorar la interfaz web

Al igual que se puede ver qué resultado da cada modelo según el donante, se podrían añadir una serie de gráficas que muestren la evolución de la importancia de las variables según se entrenen los modelos con más datos. De esta forma, se podría observar cómo varía la relevancia que los modelos dan a cada variable a lo largo del tiempo, así como analizar la evolución de los clústeres generados y la forma en que se agrupan los donantes.

11.2.5. Añadir más modelos

A medida que aumente la base de datos, se podrían añadir nuevos modelos de Machine Learning más complejos, como por ejemplo Redes neuronales o Support Vector Machines. Estos permitirían tener un nuevo enfoque más sotisficado para el análisis de los resultados.

11.2.6. Mejorar calidad del código y seguridad de la aplicación

Pese a que se ha tratado desarrollar la aplicación de la forma más eficiente y segura, es cierto que siempre se pueden implementar herramientas que mejoren estos aspectos. Por ejemplo, a la hora de guardar las predicciones de cada modelo en archivos CSV, se podría migrar a una base de datos más robusta y escalable, como puede ser Firebase, donde ya están los datos de los usuarios. Asimismo, se podrían implementar sistemas de control de errores más avanzados, validación de entrada más estricta por parte del usuario, y mecanismos de respaldo para evitar pérdidas de datos.

Conclusions and Future Work

Conclusions

The goal of this Bachelor's thesis was to develop a mobile application that met the established requirements. These requirements mainly focus on the idea of an application that can be quickly accessed to verify whether a donor is valid or not. From this application, the idea of developing a parallel web application emerged, which would allow training different Machine Learning models to compare the results with the predictions made in the mobile application. This approach aims to bring healthcare professionals closer to new technological tools that facilitate data exploration and enable diagnostics using modern methods.

Regarding the web application, due to having such a small dataset (69 records), it was not possible to train predictive models with a high level of reliability. However, the results obtained provide an initial approximation of donor behavior, which will evolve into a much more reliable model in the future. The most relevant aspect was the clustering analysis, which successfully grouped the donors in a meaningful way that had not been considered before. Although we cannot guarantee that these results are 100% accurate or representative due to the limited sample size, they do provide a useful foundation for exploring initial patterns and guiding future developments with larger datasets.

After completing the application, the established goals have been met. The code for both applications can be found at the following link:

https://TFGLauraHerrero

Future Work Lines

Expand the Database

As mentioned in previous sections, the initial dataset used for training the models contains few records. It is essential that, as the mobile application is used, the models are trained with these predictions. The more records there are, the more reliable the model will become, and better conclusions can be drawn from the results.

Adding a new feature

As the application is used, the results will be biased by the formula of the doctoral thesis. Although these results are reliable, it would be advisable to add a section for

predictions that could be modified with the real outcome, whether the donation was successful or not. In this way, the models would be trained with these predictions and there would be no bias.

Mobile App for iOS

Since the application was developed with Android Studio, it is available only for devices with this operating system. Therefore, it would be advisable to develop the application so that it is available on other operating systems, such as iOS, in order to expand the tool's reach and make it more accessible to a larger number of users. To achieve this, it would be necessary to rewrite the application using a cross-platform framework or to develop a specific version for iOS.

Web Interface Improvement

As it is possible to see the result of each model based on the donor, a set of graphs could be added to show the evolution of the importance of the variables as the models are trained with more data. In this way, it would be possible to observe how the relevance that the models assign to each variable changes over time, as well as to analyze the evolution of the clusters generated and how the donors are grouped.

Adding new models

As the database grows, new, more complex Machine Learning models could be added, such as Neural Networks or Support Vector Machines. These would provide a more sophisticated approach to analyzing the results.

Code quality and security improvement

Although efforts have been made to develop the application in the most efficient and secure way, it is true that there are always tools that could improve these aspects. For example, when saving the predictions of each model in CSV files, it could be migrated to a more robust and scalable database, such as Firebase, where the user data is already stored. In addition, more advanced error control systems, stricter user input validation, and backup mechanisms could be implemented to prevent data loss.

Bibliografía

[1] Agrega firebase al proyecto de javascript. URL: https://firebase.google.com/docs/web/setup?hl=es-419.

[2] Anaconda navigator. URL: https://www.anaconda.com/products/navigator.

[3] Android studio - official ide for android development. Accedido el 21 de noviembre de 2024. URL: https://developer.android.com/studio?hl=es-419.

[4] Android studio - official ide for android development. URL: https://developer.android.com/studio/intro?hl=es-419.

[5] Appcompatactivity. URL: https://developer.android.com/reference/androidx/appcompat/app/AppCompatActivity.

[6] Aprende el lenguaje de programación kotlin. URL: https://developer.android.com/kotlin/learn?hl=es-419#:~:text=Kotlin%20es%20un%20lenguaje%20de,y%20comenzar%20a%20utilizarlo%20r%C3%A1pidamente.

[7] await. URL: https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/await.

[8] Css. URL: https://developer.mozilla.org/es/docs/Web/CSS.

[9] Cómo funciona el algoritmo xgboost. URL: https://pro.arcgis.com/es/pro-app/latest/tool-reference/geoai/how-xgboost-works.htm.

[10] Diagramas de arquitectura con c4 model. URL: https://adictosaltrabajo.com/2023/05/06/diagramas-de-arquitectura-con-c4-model/.

[11] Firebase. URL: https://firebase.google.com/?hl=es-419.

[12] Firebase authentication. URL: https://firebase.google.com/docs/auth?hl=es-419.

[13] Firestore database. URL: https://firebase.google.com/docs/firestore?hl=es-419.

[14] Flaticon. URL: http://flaticon.es/.

[15] Html. URL: https://developer.mozilla.org/es/docs/Web/HTML.

[16] Javascript. URL: https://developer.mozilla.org/es/docs/Web/JavaScript.

[17] Joblib: running python functions as pipeline jobs. URL: https://joblib.readthedocs.io/en/stable/.

[18] Kotlin (lenguaje de programación). URL: https://es.wikipedia.org/wiki/Kotlin_(lenguaje_de_programaci%C3%B3n)#:~:text=%E2%80%8B-,Sem%C3%A1ntica,y%20el%20uso%20de%20funciones.

[19] Kotlin programming language. URL: https://kotlinlang.org/.

[20] Logisticregression. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[21] One hot encoding. URL: https://interactivechaos.com/es/manual/tutorial-de-machine-learning/one-hot-encoding.

[22] Paleta de colores html. URL: https://htmlcolorcodes.com/es/.

[23] Python. URL: https://www.python.org/.

[24] Tiobe index for april 2025. URL: https://www.tiobe.com/tiobe-index/.

[25] train_test_split. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[26] Visual studio code. URL: https://code.visualstudio.com/.

[27] Visual studio code: Editor de código para desarrolladores. URL: https://openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece/.

[28] What is data leakage in machine learning? URL: https://www.ibm.com/think/topics/data-leakage-machine-learning#:~:text=Data%20leakage%20happens%20when%20data,production%2C%20it%20becomes%20entirely%20inaccurate.

[29] Xml. URL: https://aws.amazon.com/es/what-is/xml/.

[30] ¿qué es el agrupamiento en clústeres? URL: https://developers.google.com/machine-learning/clustering/overview?hl=es-419.

[31] ¿qué es el índice de masa corporal? URL: https://www.cun.es/escuela-salud/indice-masa-corporal.

[32] Árbol de decisión en aprendizaje automático. URL: https://ojs.umsa.bo/ojs/index.php/revistavarianza/article/view/433/365.

[33] Anónimo. Modelo de regresión logística para estimar la dependencia según la escala de lawton y brody. *Elsevier*, Septiembre 2010.

[34] O. Aran. k-prototypes clustering. URL: https://www.kaggle.com/code/orkunaran/detailed-eda-k-prototypes-clustering.

[35] A. Barrios. Tutorial del algoritmo k-means en python. URL: https://medium.com/latinxinai/tutorial-del-algoritmo-k-means-en-python-d8055751e2f3.

[36] C. R. Chacón. Niveles de capnometría como indicador de validez de los donantes en asistolia no controlada (danc) y de la evolución de sus injertos renales trasplantados. Master's thesis, Universidad Francisco de Vitoria, 2025.

[37] K. S. M. Z. M. A. S.-B. B. S. S. D. B. Hatem Ali, Arun Shroff and N. Krishnan. Improved survival prediction for kidney transplant outcomes usingartificial intelligence-based models: development of the uk deceaseddonor kidney transplant outcome prediction (uk-dtop) tool. URL: https://www.tandfonline.com/doi/epdf/10.1080/0886022X.2024.2373273?needAccess=true.

[38] OMS. La oms publica el primer informe mundial sobre inteligencia artificial aplicada a la salud y seis principios rectores relativos a su concepción y utilización. *Organización Mundial de la Salud*, 2021.

[39] M. A. Ortega Páez, Ochoa Sangrador. Fundamentos de medicina basada en la evidencia. URL: https://evidenciasenpediatria.es/files/41-14126-RUTA/11_Fundamentos_Regresion_logistica.pdf.

[40] M. C.-O. M. C.-R. V. Rodriguez-Galiano, M. Sanchez-Castillo. Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees and support vector machines. *ScienceDirect*, January 2016.

[41] M. Álvarez Vega. Inteligencia artificial y aprendizaje automático en medicina. URL: https://dialnet.unirioja.es/servlet/articulo?codigo=7605021.

Apéndice A

Guía de Instación y Uso

Para poder abrir y probar la aplicación, se deben seguir los siguientes pasos descritos. Es importante destacar que este proyecto se ha desarrollado desde una máquina Windows.

A.1. Aplicación móvil

A.1.1. Herramientas a instalar

A continuación se decriben las diferentes aplicaciones y herramientas que se deben tener instaladas:

- **Código:** En el enlace proporcionado en el capítulo 11, descargar la carpeta *tfg_predicción*.
- **Android Studio Ladybug:** version 2024.2.1 Patch 2

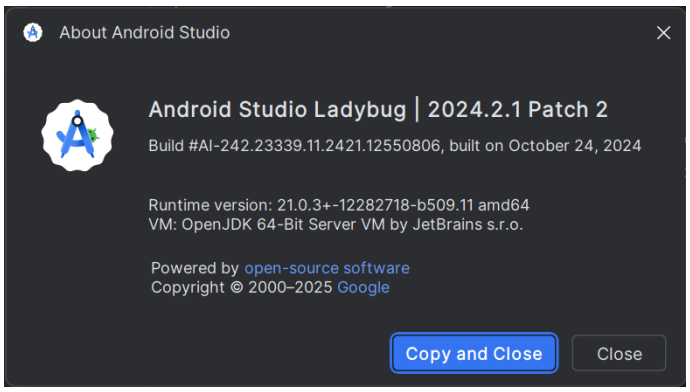


Figura A.1: Información Android Studio

- **Dispositivo emulado:** a la derecha, en *Device Manager*, añadir un dispositivo del tipo: *Medium Phone API 35*. Si ya está por defecto, no hace falta añadirlo.

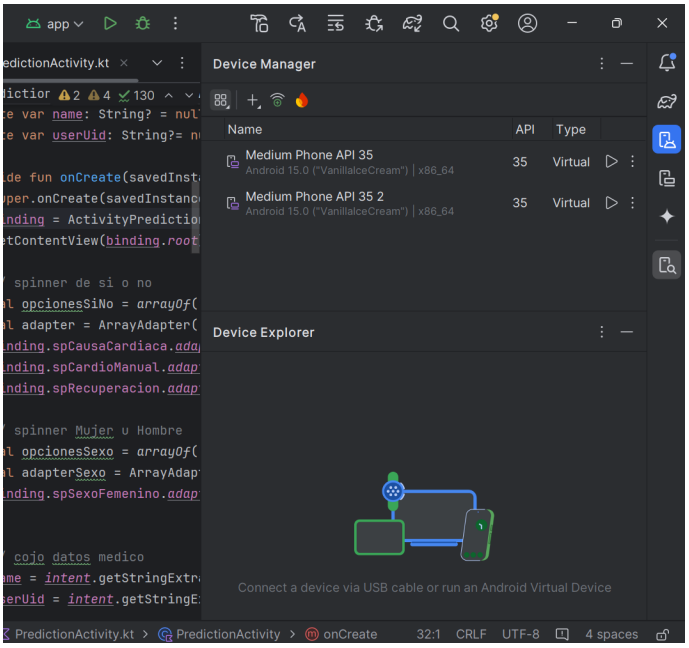


Figura A.2: Añadir dispositivo emulado

A.1.2. Pasos para importar y ejecutar

Una vez se tiene todo lo necesario instalado, se deben seguir los siguientes pasos:

1. Abrir Android Studio
2. En la barra de herramientas superior, hacer click en el icono de las tres barras.
3. Hacer click en *Open* y seleccionar la carpeta descargada *tfg_prediccion*. Se abrirá automáticamente el proyecto.
4. Antes de ejecutarlo, hacer click en la parte derecha en el icono de *Device Manager*. Si por defecto ya aparece un *Medium Phone API 35*, pasar al siguiente paso. En caso contrario, hacer click sobre el icono + y seleccionar *Create virtual device*. En la sección *Select Hardware* (figura A.3) seleccionar la categoría *Phone* y el nombre *Medium Phone*. Hacer click en *Next*, se abrirá la sección *System Image*. Seleccionar *VanillaIceCream* con API 35 (figura A.4). Hacer click en *Next* y, en la nueva sección de Android Virtual Device (figura A.5), hacer click en *Finish*.

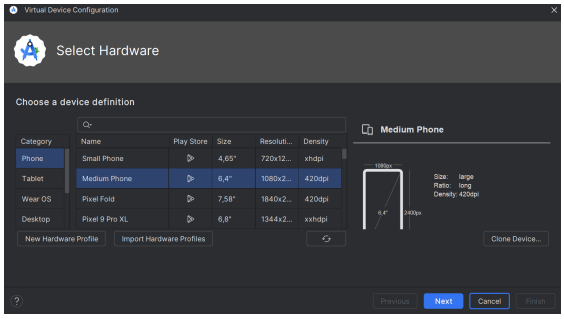


Figura A.3: Sección Seleccionar hardware

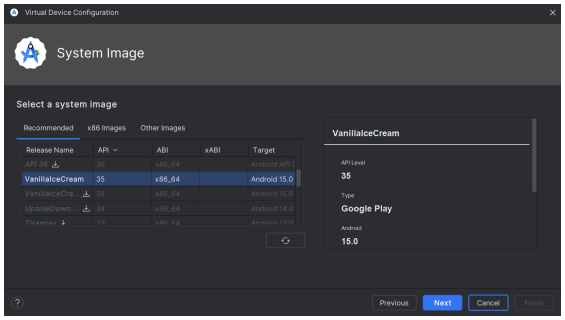


Figura A.4: Sección Imagen de sistema

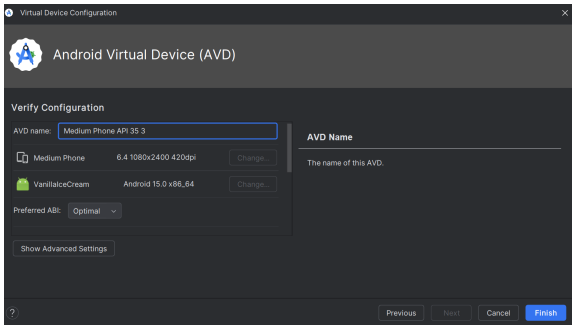


Figura A.5: Sección Android Virtual Device

5. Para ejecutarlo, hacer click en el icono de *Run app* en la barra de herramientas superior. Es un icono de triángulo verde apuntando a la derecha. El emulador se cargará con la aplicación.

6. Ahora ya se puede utilizar la aplicación móvil. Cuando se haya terminado, se debe hacer click en *Stop app*, un icono de un cuadrado rojo, ubicado a la derecha del *Run app* en la barra de herramientas superior.

Una vez la aplicación está ejecutándose, el usuario puede crear su propia cuenta o utilizar una que ya exista.

Para acceder como médico puede introducir los siguientes datos:

- Usuario: medico@gmail.com
- Contraseña: 123123123

Para acceder como administrador puede introducir los siguientes datos:

- Usuario: administrador@gmail.com
- Contraseña: 123123123

A.2. Aplicación web

A.2.1. Herramientas a instalar

A continuación se decriben las diferentes aplicaciones y herramientas que se deben tener instaladas:

- **Código:** En el enlace proporcionado en el capítulo 11, descargar la carpeta *app_ web*.
- **Visual Studio Code:** instalar la versión 1.100.0
- **Anaconda Navigator:** instalar la versión 2.6.3.
- **Python:** en las extensiones de Visual Studio Code (en la barra lateral izquierda) instalar la extensión de Python (versión 2025.6.0)
- **JavaScript:** en las extensiones de Visual Studio Code (en la barra lateral izquierda) instalar la extensión de Javascript (versión 1.8.0)
- **Archivo requirements:** En el enlace proporcionado en el capítulo 11, descargar la carpeta *requirements.txt*.

A.2.2. Pasos para importar y ejecutar

Una vez se tiene todo lo necesario instalado, se deben seguir los siguientes pasos:

1. En Anaconda Navigator, para crear un nuevo entorno, hacer click sobre *Create* y poner como nombre, por ejemplo, *tfg*. Seleccionar Python, la versión por defecto.

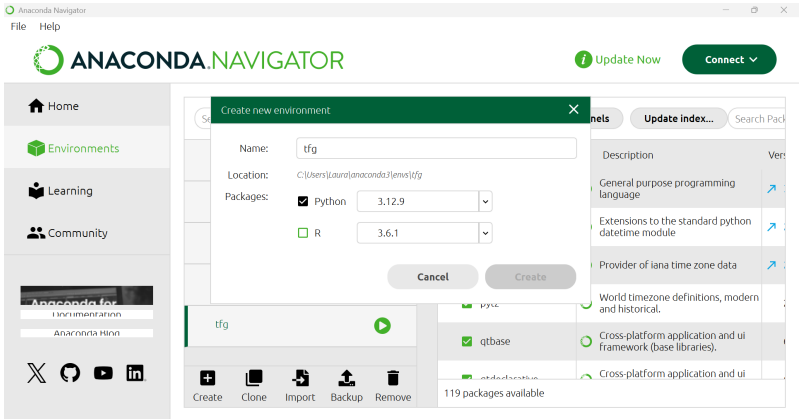


Figura A.6: Nuevo entorno virtual

2. Abrir en VS Code la carpeta descargada *app_ web*. Esto se realiza haciendo click en *File*, que se encuentra en la barra de herramientas superior. Después, hacer click en *Open Folder* y seleccionar la carpeta. El proyecto se abrirá automáticamente.

3. En VS Code, abrir la paleta de comandos con *Ctrl + Shift + P* y escribir *Python: Select Interpreter*. Seleccionar el entorno creado en el paso 1.

4. Abrir una terminal en VS Code (Terminal ->New Terminal en la barra superior) y ejecutar el siguiente comando: pip install -r requirements.txt (para que funcione se debe navegar a la carpeta donde se encuentre este archivo)

5. Ya está todo preparado para poder ejecutar la aplicación. Para ello se deben abrir dos terminales en VS Code. En una se debe navegar a la carpeta *backend* y en la otra a *frontend*. En backend, se debe escribir el siguiente comando: *python app.py*, y en frontend se debe escribir: *live-server*. De esta forma, se abrirá automáticamente en el navegador la web correspondiente.

Una vez la aplicación está ejecutándose, el usuario debe utilizar una cuenta de administrador creada en la aplicación móvil.

Para acceder, puede introducir los siguientes datos:

- Usuario: administrador@gmail.com
- Contraseña: 123123123

Apéndice B

Manual de Usuario

A continuación se muestran los pasos que debe seguir cualquier persona que desee utilizar la aplicación. Antes de todo, se deben haber completado los pasos indicados en la guía de instalación y uso, del apéndice A.

B.1. Aplicación móvil

B.1.1. Módulo inicial

Al abrir la aplicación nos encontramos con una pantalla de inicio de sesión como la mostrada en la figura B.1. Si tratamos de iniciar sesión con un usuario y/o contraseña no válidos, saldrá una notificación como en la figura B.2.

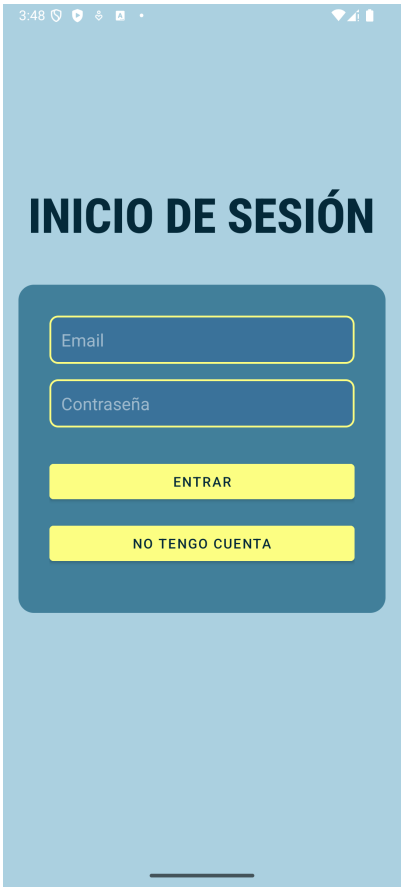


Figura B.1: Captura de pantalla: Login

Figura B.2: Captura de pantalla: Login fallido

Si pulsamos el botón de *No tengo cuenta*, nos lleva a la pantalla de registro como la que nos muestra la figura B.3. Es necesario rellenar todos los campos para poder crear una cuenta. En el caso de que haya ya un usuario con ese correo electrónico o las contraseñas no coincidan, aparece una notificación como las de las figuras B.4, B.5 y B.6, respectivamente.

Figura B.3: Captura de pantalla: Pantalla de registro

Figura B.4: Captura de pantalla: notificación campos obligatorios

Figura B.5: Captura de pantalla: Registro fallido, correo electrónico ya existente

Figura B.6: Captura de pantalla: Registro fallido, contraseñas no coinciden

Una vez se ha iniciado sesión, tanto los Médicos como los Administradores pueden abrir un menú lateral desplegable como el de la figura 8.2 mostrado anteriormente. Si pulsamos sobre *Editar perfil*, navegamos a un formulario similar al del registro, pero con los campos ya completados y sin la posibilidad de modificar el rol ni el nombre de usuario. Una vez se realizan los cambios, se muestra una notificación de éxito y se navega automáticamente a la pantalla principal (figura B.10). En el caso de que se quiera cambiar la contraseña, se comprueba que la actual coincida con la de la base de datos (figura B.8) y que la nueva y la confirmación coincidan también. En caso de que no coincidan se muestra una notificación de error, como se ve en la figura B.9.

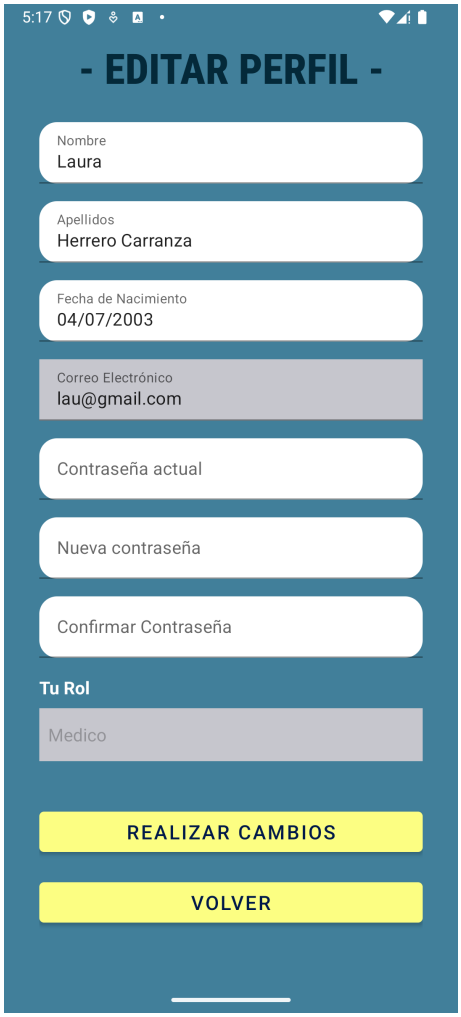


Figura B.7: Captura de pantalla: Editar perfil

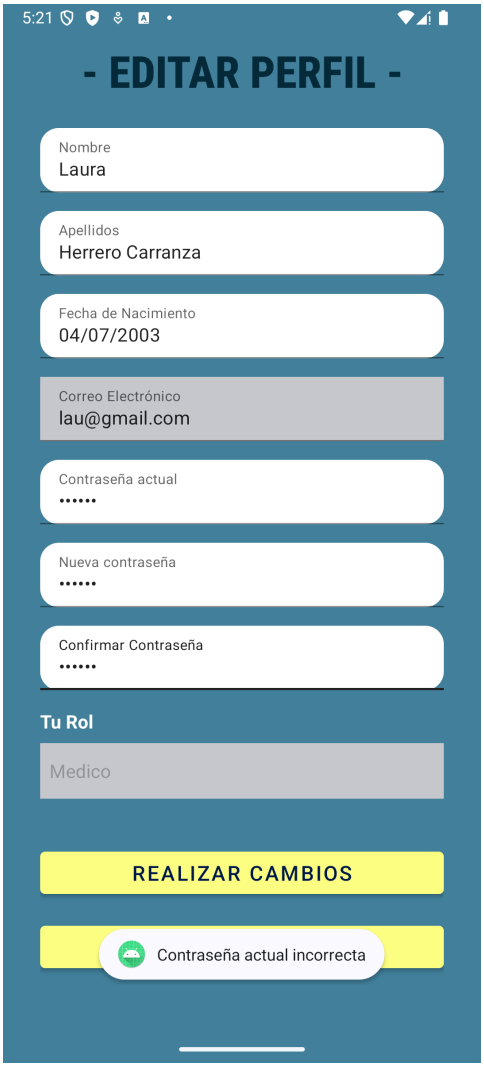


Figura B.8: Captura de pantalla: Contraseña inválida

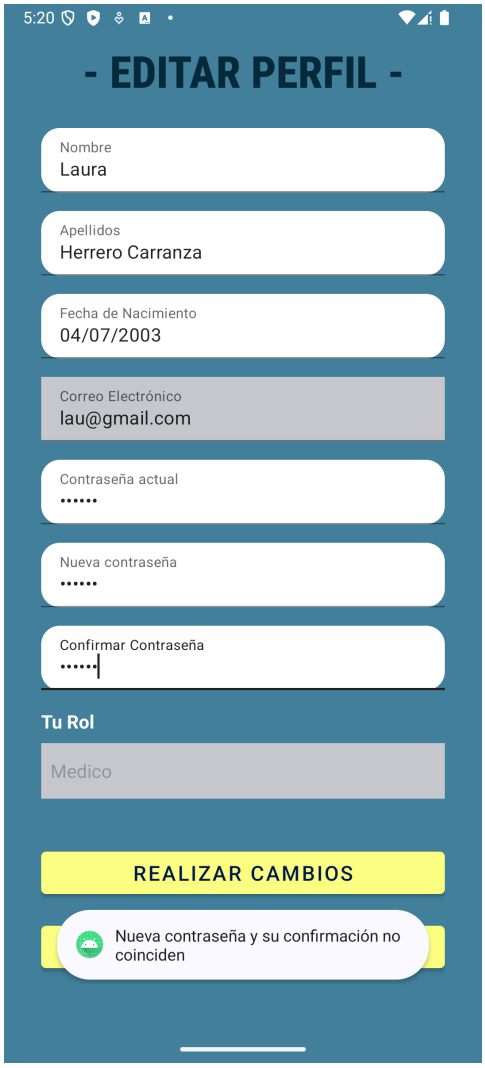


Figura B.9: Captura de pantalla: Contraseñas no coinciden

B.1.2. Módulo de Usuario: Médico

Al entrar como Médico se abre la pantalla principal, como se muestra en la figura B.10.

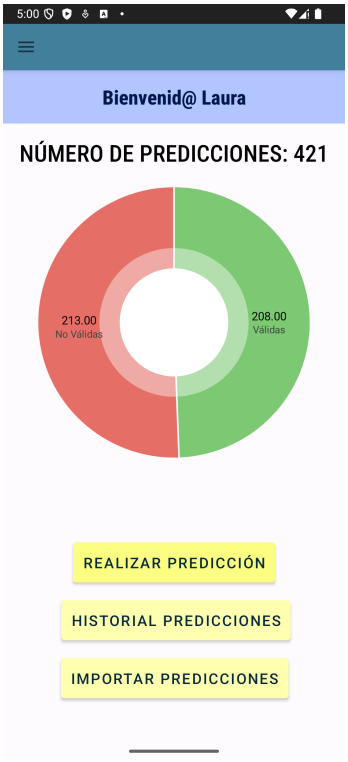


Figura B.10: Captura de pantalla: Pantalla principal Médico

Si pulsamos el botón de *Realizar predicción*, navegamos al formulario para introducir los datos del donante, como se ve en la figura B.11. En este formulario todos los campos son obligatorios, como se puede ver en la notificación de la figura B.12. Además, este formulario solo permite que los campos de *Edad* y *Capnometría* sean números, lo que se consigue gracias a *android:inputType="number"* en su correspondiente campo del archivo XML.

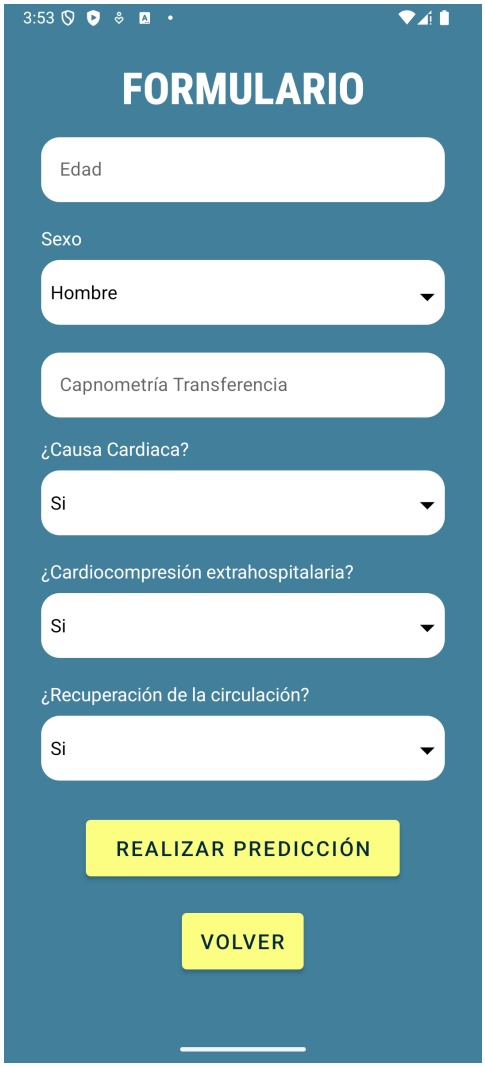


Figura B.11: Captura de pantalla: Formulario de donante

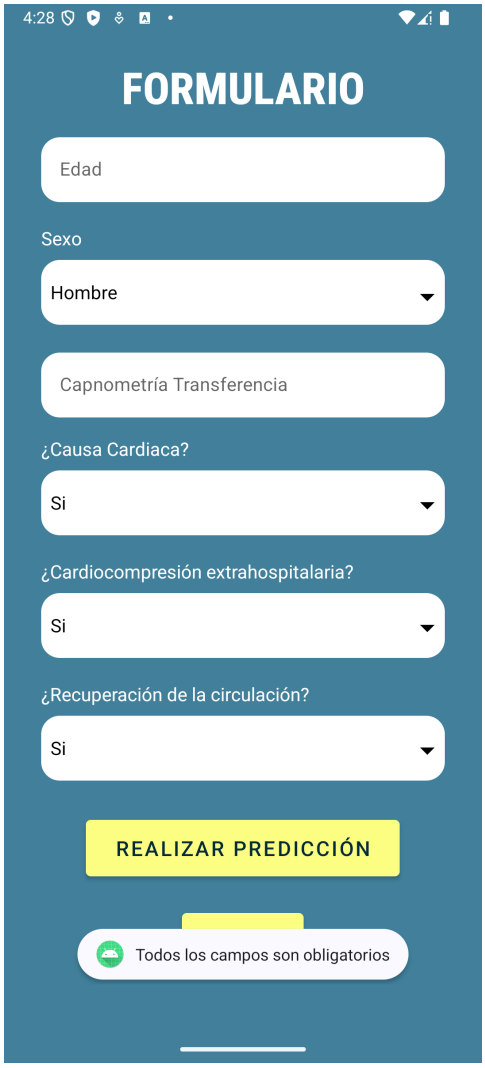


Figura B.12: Captura de pantalla: Campos obligatorios

Tras realizar la predicción, se navega a una pantalla diferente dependiendo de si el resultado es Válido o No válido. En ambas aparece una notifiación y el resultado con su color correspondiente: verde para las válidas y rojo para las no válidas. Además, aparecen dos botones, uno para descargar un PDF con el informe de la predicción (figura B.15) y otro para volver al menú principal.

Figura B.13: Captura de pantalla: Donante válido

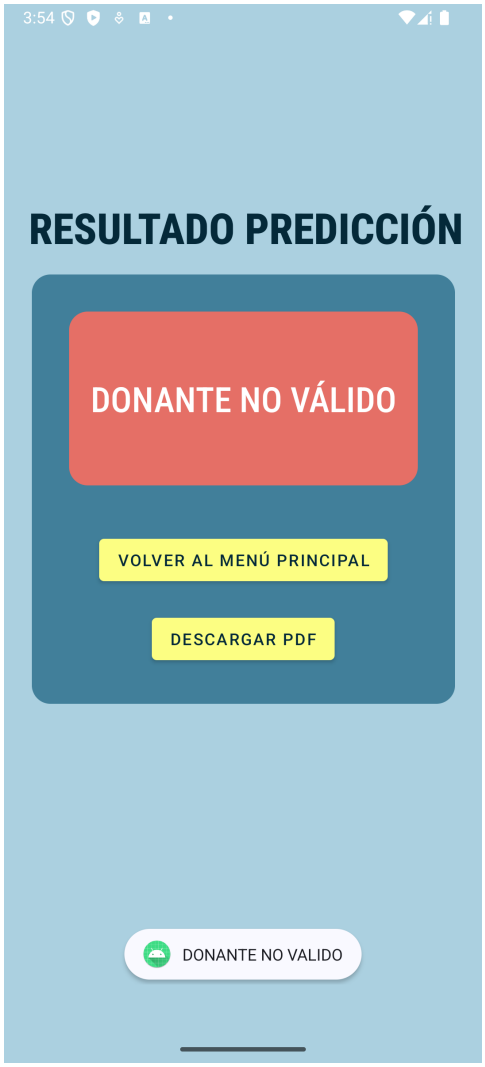


Figura B.14: Captura de pantalla: Donante no válido

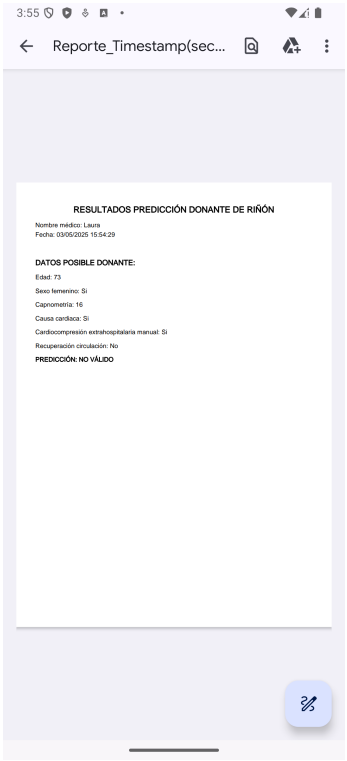


Figura B.15: Captura de pantalla: PDF generado

Volviendo al menú principal, el usuario puede pulsar el botón de *Historial de predicciones*. Este botón navega a la pantalla que muestra la figura B.16, donde aparecen las predicciones realizadas por el usuario ordenadas según su fecha. Podemos determinar el orden de las predicciones utilizando el spinner, en el que las opciones son: Fecha, Edad y Capnometría. Para cada predicción encontramos las siguientes columnas: *Edad*, *Sexo Femenino*, *Capnometría*, *Muerte por causa cardiaca*, *Cardiocompresión*, *Recuperación del pulso*, *Resultado*, *Informe* y *Eliminar*. Pulsando el icono de *Informe* se descarga un nuevo PDF con los datos de la predicción, como en la figura B.15 mostrada anteriormente. Por otro lado, pulsando el icono de *Eliminar* aparece un diálogo de confirmación que, si es aceptado, eliminará la predicción de la base de datos y actualizará los contadores.

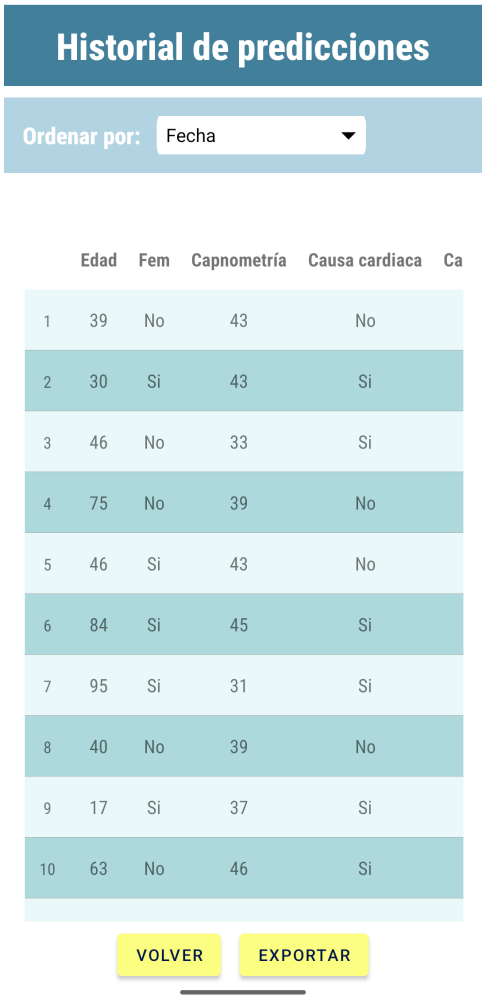


Figura B.16: Captura de pantalla: Historial

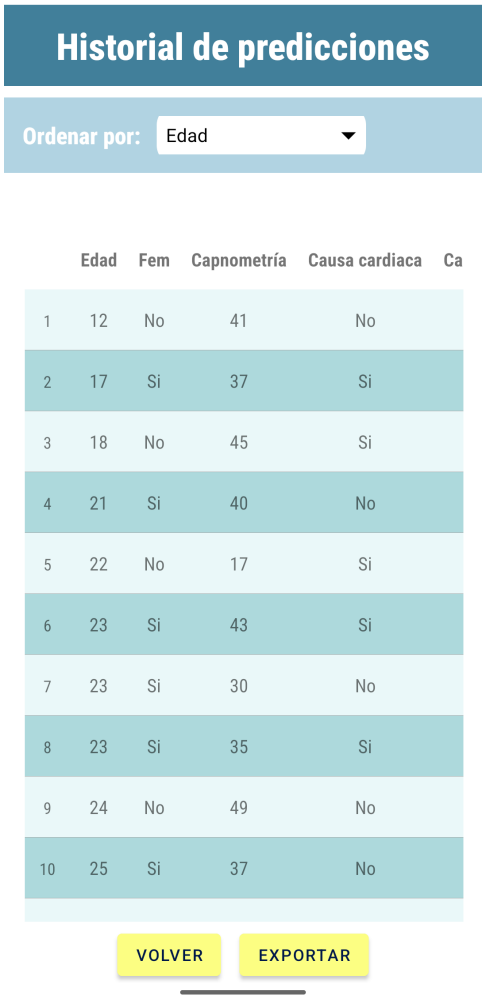


Figura B.17: Captura de pantalla: Historial ordenado por edad

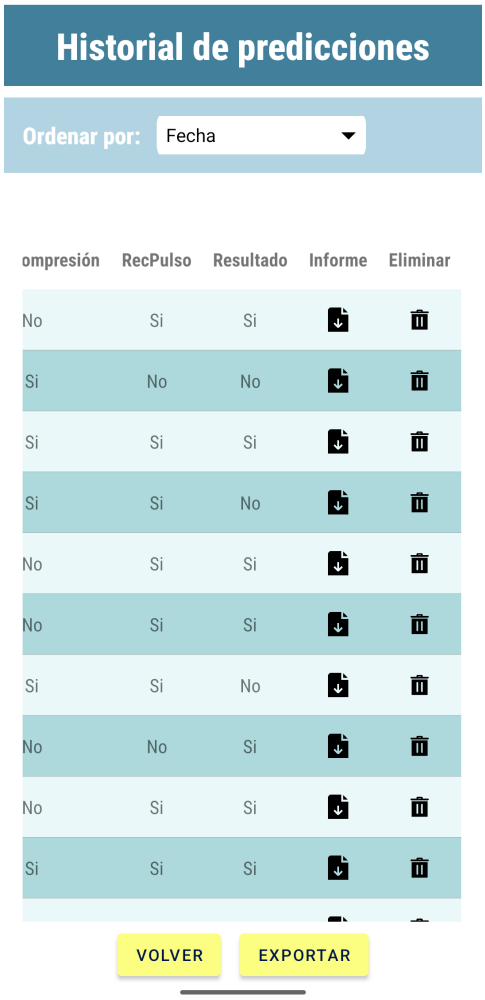


Figura B.18: Captura de pantalla: Historial desplazado hacia la derecha

Figura B.19: Captura de pantalla: confirmación de eliminación

Para terminar, el tercer botón del menú principal es el de *Importar predicciones*. Este botón navega a una pantalla simple que pide al usuario seleccionar un archivo CSV con el formato indicado. Tras seleccionar el archivo, la lógica de la actividad lee cada línea y añade la predicción a la base de datos. Si se ha completado la tarea con éxito, aparece una notificación de éxito y la barra se completa, como se ve en la figura B.21.

Figura B.20: Captura de pantalla: Importar archivo CSV

Figura B.21: Visualización del CSV importado correctamente

B.1.3. Módulo de Usuario: Administrador

Al iniciar sesión como Administrador aparece la pantalla principal correspondiente, como se muestra en la figura B.22. Esta consta de una tabla con todos los médicos loggeados en la aplicación, mostrando sus apellidos y correo electrónico. Al presionar sobre el icono de la columna *Preds*, se navega a la pantalla mostrada en la figura B.23 donde aparecen todas las predicciones realizadas por el correspondiente médico.

Figura B.22: Captura de pantalla: Pantalla principal Administrador

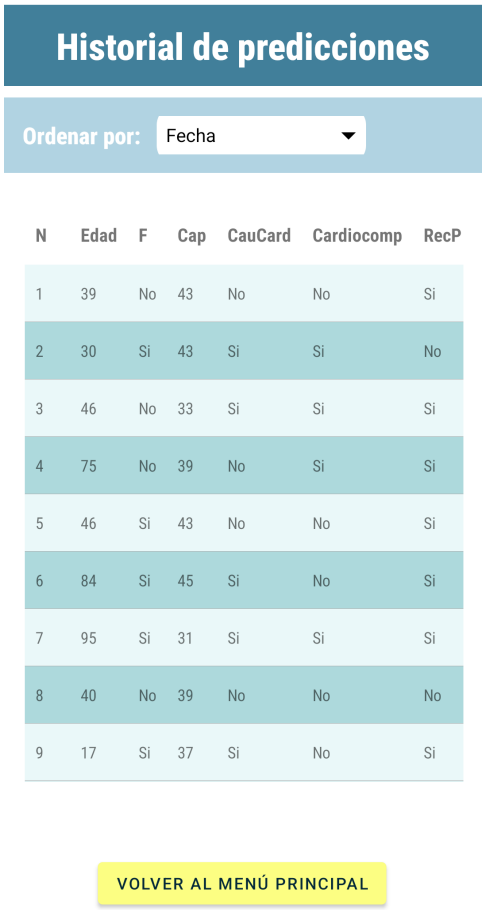


Figura B.23: Captura de pantalla: Predicciones realizadas por Médico

Los administradores no tienen más funcionalidades en la aplicación móvil ya que su rol está enfocado a las funciones de la aplicación web, donde pueden acceder a la base de datos para entrenar modelos y visualizar sus resultados.

B.2. Aplicación web

B.2.1. Módulo de Usuario: Administrador

Al abrir la aplicación móvil nos encontramos la pantalla de inicio de sesión. Si tratamos de iniciar sesión con un usuario y/o contraseña no válidos, sadrá una notificación como en la figura B.25. Si tratamos de iniciar sesión con un usuario de rol Médico, saldrá una notificación como en la figura B.26.

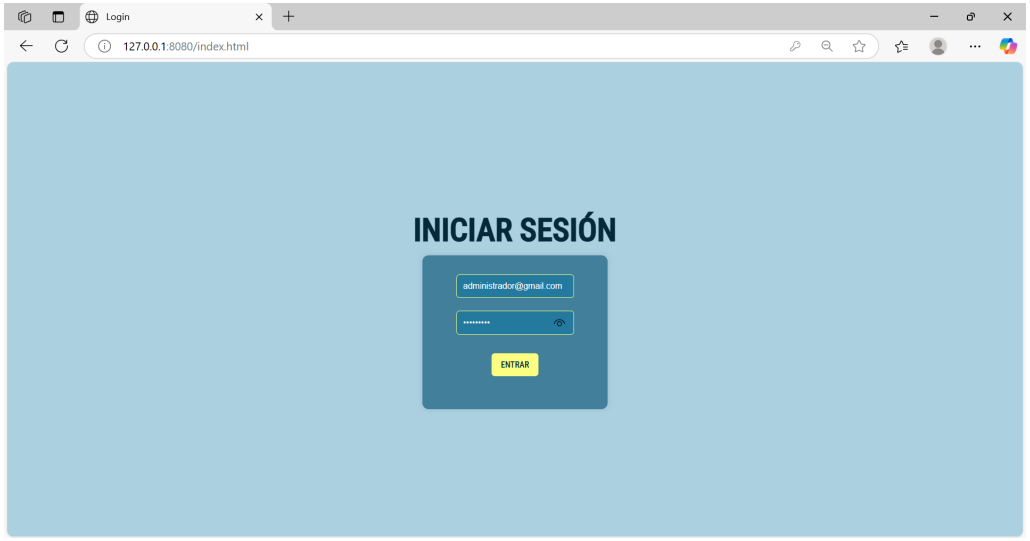


Figura B.24: Captura de pantalla: Login web

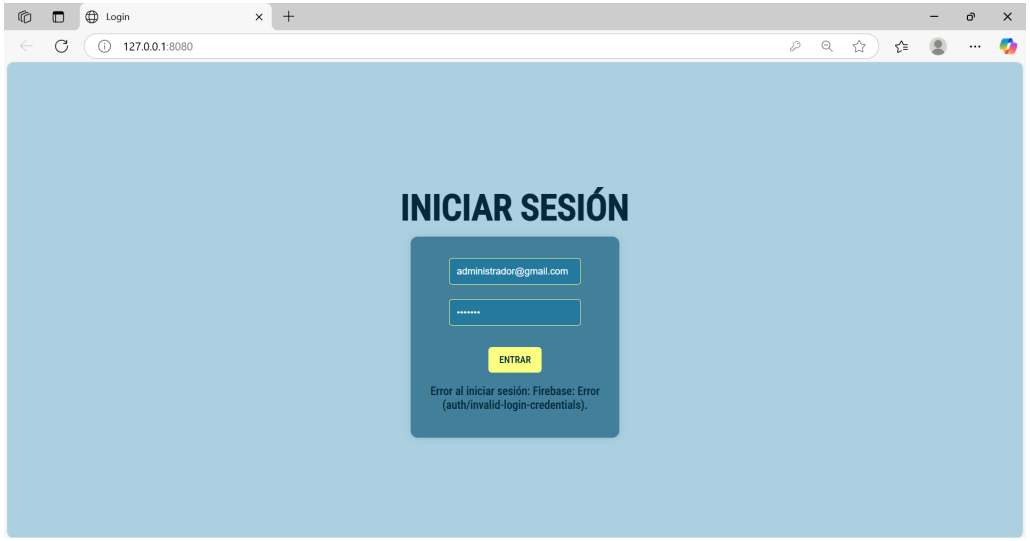


Figura B.25: Captura de pantalla: usuario y/o contraseña no válidos

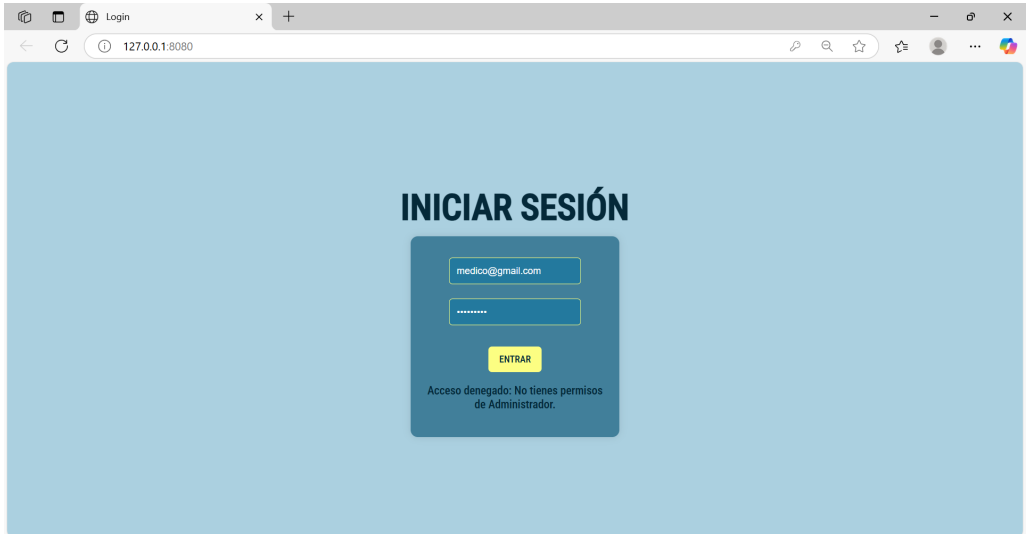


Figura B.26: Captura de pantalla: Login no válido para Médicos

Una vez se inicia sesión, se navega a la pantalla principal. En esta aparece una tabla con las predicciones realizadas por todos los médicos. Cambiando de opción con el spinner, se pueden ver las predicciones que forman parte del dataset del modelo seleccionado. Por ejemplo, en la figura B.28 se ven las predicciones con las que el Árbol de decisión ha sido entrenado. Las opciones del spinner son: *Todas*, *Árbol de decisión*, *Random Forest*, *Regresión Logística* y *Clustering*.

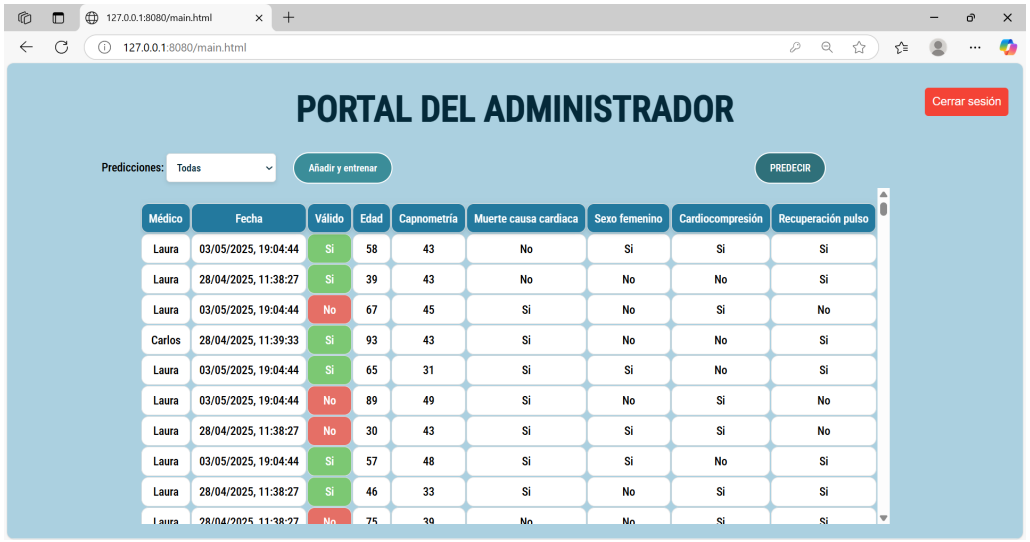


Figura B.27: Captura de pantalla: página principal con todas las predicciones

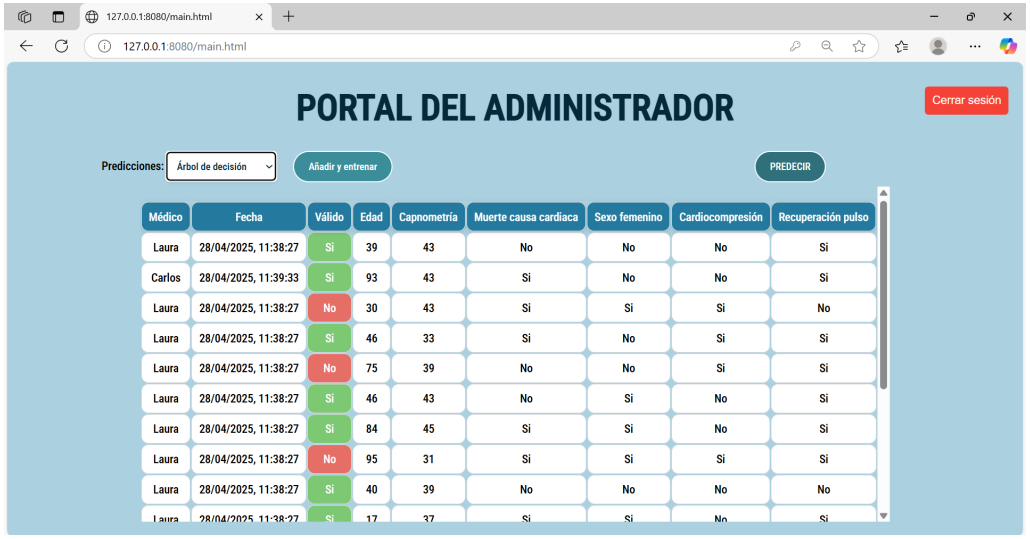


Figura B.28: Captura de pantalla: página principal con el dataset del modelo Árbol de decisión

Si pulsamos el botón *Añadir y entrenar*, navegamos a la pantalla mostrada en la figura B.29, donde se muestran las predicciones que forman parte de la base de datos pero no pertenecen al dataset del modelo seleccionado por el spinner.

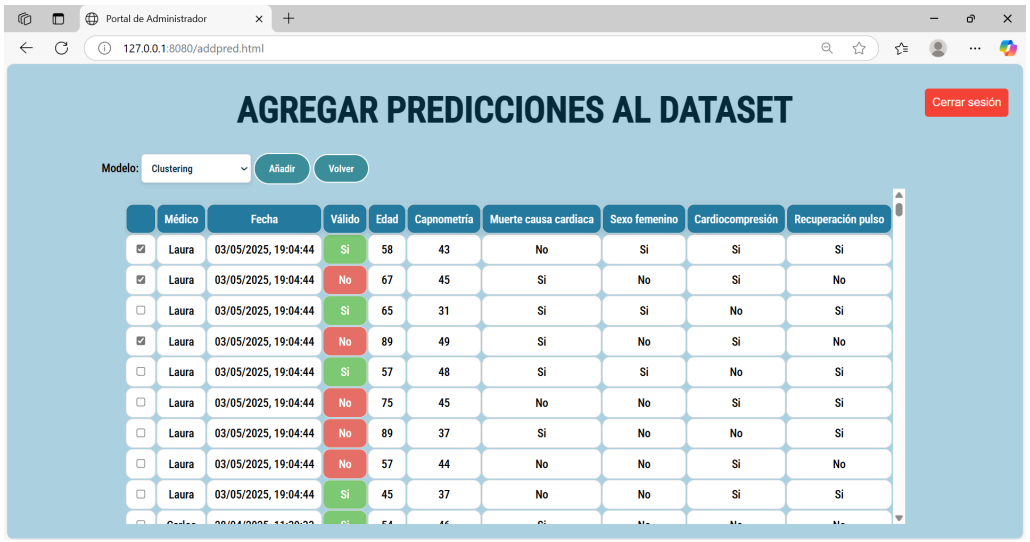


Figura B.29: Captura de pantalla: tabla añadir predicciones

El usuario debe seleccionar las predicciones que desea añadir al dataset de entrenamiento del modelo. Cuando se pulsa el botón *Añadir*, se comprueba que haya predicciones seleccionadas. De no ser el caso, aparece un mensaje de alerta notificando sobre ello (figura B.30). En caso de que sí se hayan seleccionado al menos una predicción, se muestra un diálogo de confirmación como el de la figura B.31.

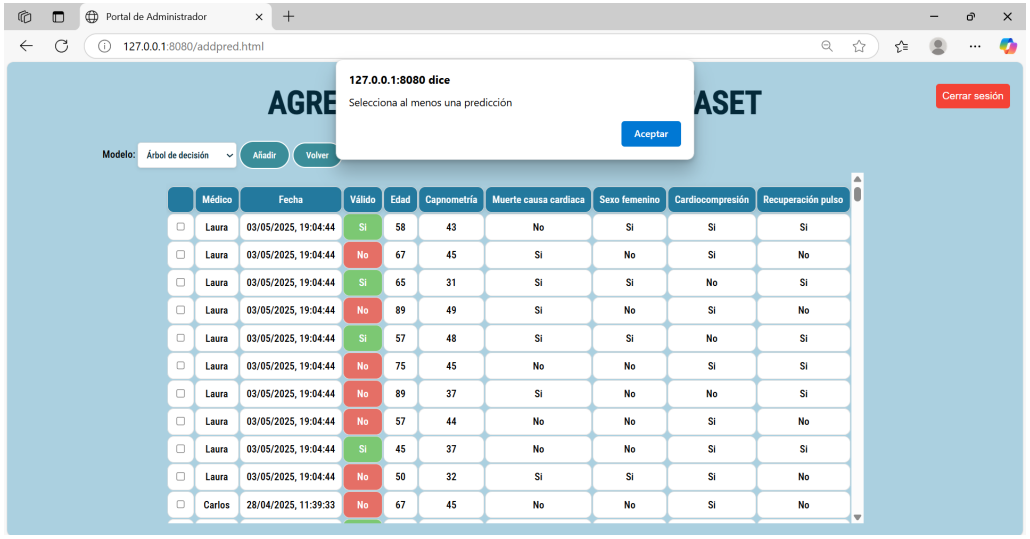


Figura B.30: Captura de pantalla: seleccionar al menos una predicción

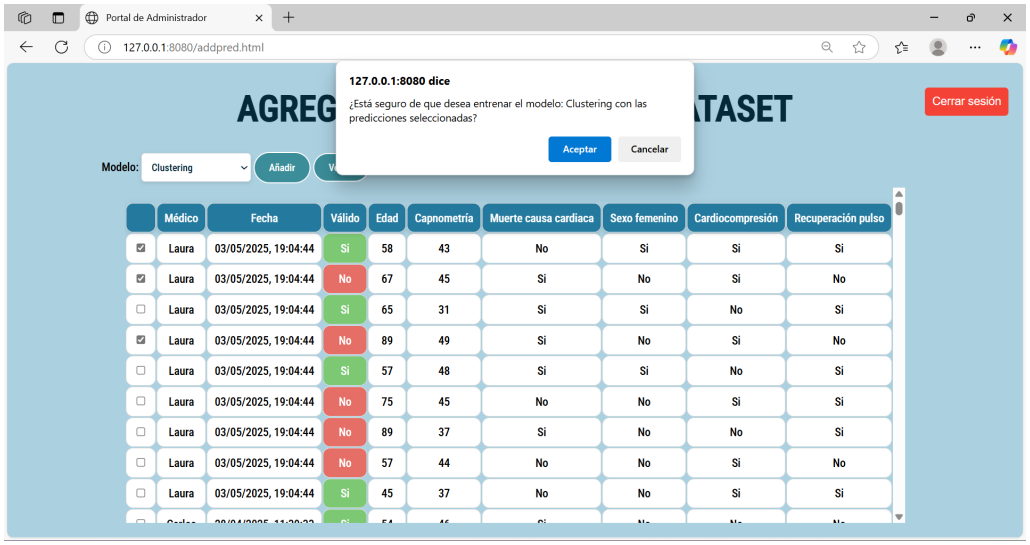


Figura B.31: Captura de pantalla: diálogo de confirmación

Una vez se pulsa la confirmación del diálogo, el usuario debe esperar a que aparezca un mensaje indicando que el modelo se ha entrenado correctamente (figura B.32).

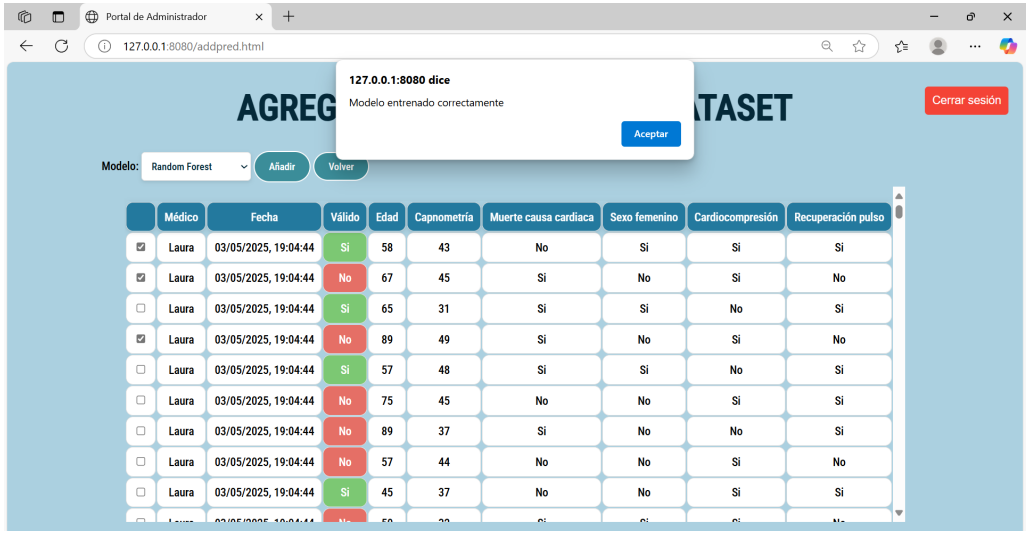


Figura B.32: Captura de pantalla: modelo entrenador correctamente

Dependiendo de cuál sea el modelo entrenado, se abrirá un modal con información sobre el entrenamiento. Esta información se puede descargar en formato .txt si el usuario lo desea, pulsando *descargar .txt*. Se pueden encontrar ejemplos de los cuatro modelos de archivos .txt en ***este enlace***. No obstante, un ejemplo de cómo se ve el modal con la información son las figuras B.33 y B.34, donde aparece un informe con los datos del entrenamiento del Árbol de decisión. Al pulsar la cruz del modal, se redirige a la pantalla principal, con todas las predicciones en la tabla.

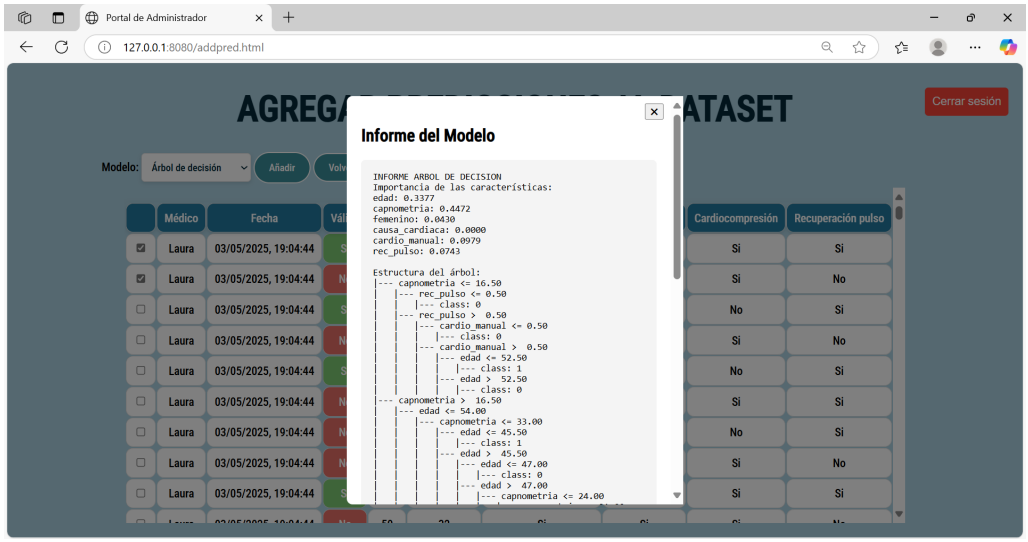


Figura B.33: Captura de pantalla: modal 1

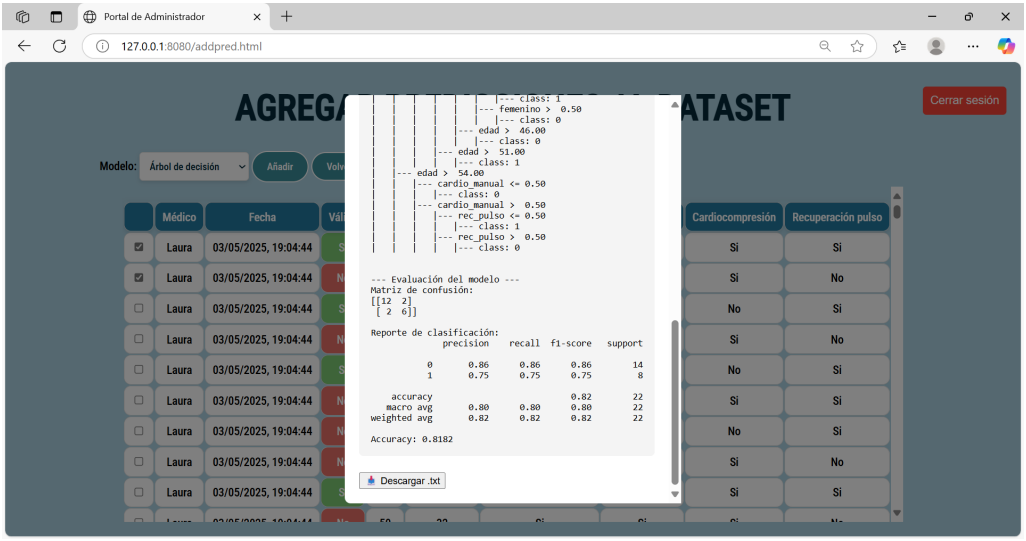


Figura B.34: Captura de pantalla: modal 2

Por último, el usuario puede pulsar el botón *PREDECIR* de la página principal (figura B.27 mostrada anteriormente) para rellenar un formulario con los datos de un posible donante y poder comparar los resultados de los diferentes modelos.

En la figura B.35 se puede ver la interfaz con el formulario. Los datos que se solicitan son: *Edad*, *Capnometría*, *IMC*, *Muerte por causa cardiaca*, *Realización de cardiocompresión*, *Recuperación del pulso*, *Grupo sanguíneo* y *Sexo*. Conviene señalar que los valores del Índice de Masa Corporal y del grupo sanguíneo pueden ser desconocidos, más adelante se explicará cómo se han tratado este tipo de variables en los modelos. Además, el usuario puede seleccionar los modelos cuyos resultados desea visualizar, con el fin de compararlos con los obtenidos con la fórmula de la tesis doctoral. En caso de no rellenar todos los campos aparece un cuadro de alerta como el de la figura B.36.

Figura B.35: Captura de pantalla: formulario

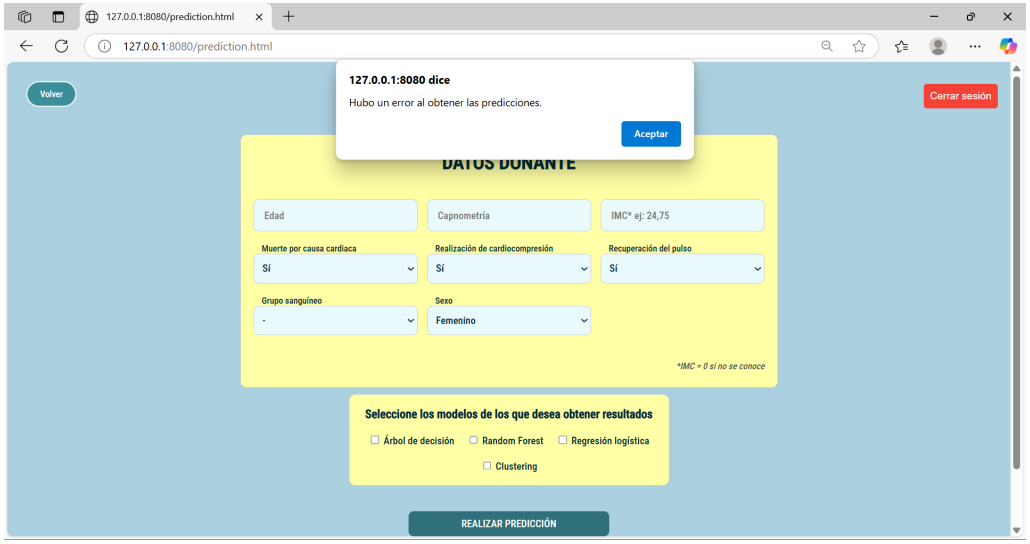


Figura B.36: Captura de pantalla: cuadro de alerta

Una vez se rellenan los campos y se seleccionan los modelos, la aplicación envía al backend esta información, se calcula el resultado de cada modelo y se envían los resultados al frontend. Inmediatamente después se carga la pantalla mostrada en las figuras B.37 y B.38, donde para cada modelo se indica si el modelo devuelve un resultado válido o no, tanto para la versión ampliada que contiene el IMC y el grupo sanguíneo como para la versión estándar. También aparece el resultado que da la fórmula de la tesis doctoral. Asimismo, para el Árbol de Decisión y el Random Forest, se muestra el valor de *Accuracy* obtenido durante el entrenamiento con el mismo conjunto de datos utilizado que para realizar la predicción. De igual forma, en el apartado de *Clustering*, aparecen los valores medios de las variables en el cluster al que ha sido asignada la predicción indicada en el formulario.

En este ejemplo podemos ver que, para los valores introducidos, coincide el resultado de la fórmula con el del modelo de Regresión Logística y el de K-Prototypes del Clustering (ambas para la versión estándar, que no incluye el IMC ni el grupo sanguíneo). Esta herramienta tiene como propósito ver con qué tipo de predicciones los modelos coinciden en sus resultados, para así en un futuro poder determinar qué modelo va mejor según los datos de entrada.

Figura B.37: Captura de pantalla: resultados 1

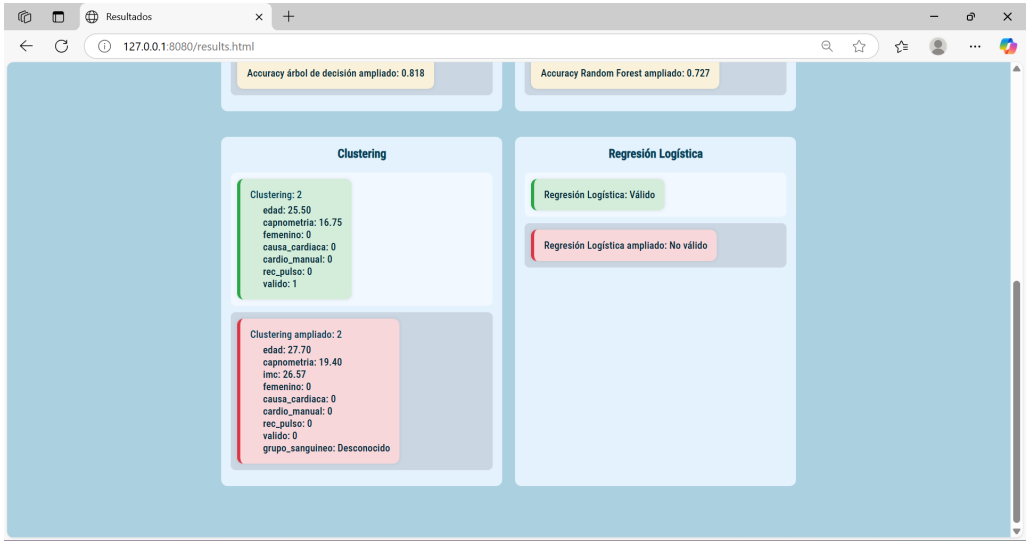


Figura B.38: Captura de pantalla: resultados 2