Neiva, 31 de septiembre de 2022

Señores

CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN

UNIVERSIDAD SURCOLOMBIANA

Neiva

El (Los) suscrito(s):

Maicol Andres Garcia Rodriguez , con C.C. No 1075310787,

Gustavo Andres Medina Diaz, con C.C. No. 1075292739

Autor(es) de la tesis y/o trabajo de grado titulado SISTEMA EXPERTO PARA EL APOYO AL DIAGNÓSTICO MÉDICO DE HIPERTENSIÓN Y DIABETES A TRAVÉS DE MACHINE LEARNING, presentado y aprobado en el año 2022 como requisito para optar al título de INGENIERO ELECTRÓNICO;

Autorizo (amos) al CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN de la Universidad Surcolombiana para que, con fines académicos, muestre al país y el exterior la producción intelectual de la Universidad Surcolombiana, a través de la visibilidad de su contenido de la siguiente manera:

- Los usuarios puedan consultar el contenido de este trabajo de grado en los sitios web que administra la Universidad, en bases de datos, repositorio digital, catálogos y en otros sitios web, redes y sistemas de información nacionales e internacionales “open access” y en las redes de información con las cuales tenga convenio la Institución.

- Permita la consulta, la reproducción y préstamo a los usuarios interesados en el contenido de este trabajo, para todos los usos que tengan finalidad académica, ya sea en formato Cd-Rom o digital desde internet, intranet, etc., y en general para cualquier formato conocido o por conocer, dentro de los términos establecidos en la Ley 23 de 1982, Ley 44 de 1993, Decisión Andina 351 de 1993, Decreto 460 de 1995 y demás normas generales sobre la materia.

- Continúo conservando los correspondientes derechos sin modificación o restricción alguna; puesto que, de acuerdo con la legislación colombiana aplicable, el presente es un acuerdo jurídico que en ningún caso conlleva la enajenación del derecho de autor y sus conexos.

De conformidad con lo establecido en el artículo 30 de la Ley 23 de 1982 y el artículo 11 de la Decisión Andina 351 de 1993, “Los derechos morales sobre el trabajo son propiedad de los autores”, los cuales son irrenunciables, imprescriptibles, inembargables e inalienables.

EL AUTOR/ESTUDIANTE:

Maicol Andres Garcia Rodriguez

Firma: _____

Maicol A. Garcia

EL AUTOR/ESTUDIANTE:

Gustavo Andres Medina Diaz

Firma: _____

TÍTULO COMPLETO DEL TRABAJO: SISTEMA EXPERTO PARA EL APOYO AL DIAGNÓSTICO MÉDICO DE HIPERTENSIÓN Y DIABETES A TRAVÉS DE MACHINE LEARNING

AUTOR O AUTORES:

Primero y Segundo Apellido	Primero y Segundo Nombre
GARCIA RODRIGUEZ	MAICOL ANDRES
MEDINA DIAZ	GUSTAVO ANDRES

DIRECTOR Y CODIRECTOR TESIS:

Primero y Segundo Apellido	Primero y Segundo Nombre
QUINTERO POLANCO	JESÚS DAVID

ASESOR (ES):

Primero y Segundo Apellido	Primero y Segundo Nombre

PARA OPTAR AL TÍTULO DE: Ingeniero Electrónico

FACULTAD: Ingeniería

PROGRAMA O POSGRADO: Ingeniería Electrónica

CIUDAD: Neiva **AÑO DE PRESENTACIÓN:** 2022 **NÚMERO DE PÁGINAS:** 108

TIPO DE ILUSTRACIONES (Marcar con una **X**)**:**

Diagramas X Fotografías X Grabaciones en discos___ Ilustraciones en general X Grabados___ Láminas___ Litografías___ Mapas___ Música impresa___ Planos___ Retratos___ Sin ilustraciones___ Tablas o Cuadros X

SOFTWARE requerido y/o especializado para la lectura del documento:

MATERIAL ANEXO:

PREMIO O DISTINCIÓN *(En caso de ser LAUREADAS o Meritoria)*:

PALABRAS CLAVES EN ESPAÑOL E INGLÉS:

Español	Inglés	Español	Inglés
1. API	API	**6.** Predicciones	Predictions
2. Framework	Framework	**7.** Aplicación móvil	Mobile application
3. Django	Django		
4. Aprendizaje de máquina	Machine learning		
5. Algoritmos	Algorithms		

RESUMEN DEL CONTENIDO: (Máximo 250 palabras)

Se realizó la implementación de un producto mínimo viable del sistema experto para el diagnóstico de hipertensión y diabetes con Machine Learning desarrollado en Python donde se probaron diferentes métodos tradicionales para determinar cuál se ajustaba mejor a los datos y entregaba un mayor porcentaje de predicción.

Se desarrolló una aplicación web con el framework de Django que permite a los usuarios realizar un proceso de diagnóstico a través de una interfaz gráfica con diferentes formularios que la hacen fácil y agradable de utilizar, así como también llevar un historial de las consultas realizadas mediante la utilización del gestor de bases de datos PostgreSQL. Además, se

utilizó la librería Django Rest Framework para crear una API que interactúa e intercambia información con una aplicación móvil desarrollada en Android Studio.

La aplicación móvil permite a los usuarios registrarse, hacer un proceso de login y realizar el proceso de diagnóstico de hipertensión y diabetes haciendo uso de la librería Retrofit para la comunicación con la API.

ABSTRACT: (Máximo 250 palabras)

The implementation of a minimum viable product of the expert system for the diagnosis of hypertension and diabetes was carried out with Machine Learning developed in Python where different traditional methods were tested to determine which one best fit the data and delivered a higher percentage of prediction.

A web application was developed with the Django framework that allows users to carry out a diagnostic process through a graphical interface with different forms that make it easy and pleasant to use, as well as keeping a history of the queries made through the use of the PostgreSQL database manager. In addition, the Django Rest Framework library was used to create an API that interacts and exchanges information with a mobile application developed in Android Studio.

The mobile application allows users to register, perform a login process and carry out the hypertension and diabetes diagnosis process using the Retrofit library for communication with the API.

APROBACION DE LA TESIS

Nombre Jurado: Martín Diomedes Bravo Obando

Firma:

Nombre Jurado: Johan Julián Molina Mosquera

Firma:

SISTEMA EXPERTO PARA EL APOYO AL DIAGNÓSTICO MÉDICO DE HIPERTENSIÓN Y DIABETES A TRAVÉS DE MACHINE LEARNING

MAICOL ANDRES GARCIA RODRIGUEZ
GUSTAVO ANDRÉS MEDINA DIAZ

UNIVERSIDAD SURCOLOMBIANA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
NEIVA - HUILA
2022

SISTEMA EXPERTO PARA EL APOYO AL DIAGNÓSTICO MÉDICO DE HIPERTENSIÓN Y DIABETES A TRAVÉS DE MACHINE LEARNING

MAICOL ANDRES GARCIA RODRIGUEZ
GUSTAVO ANDRES MEDINA DIAZ

Trabajo de Grado Para Optar al Título de Ingeniero Electrónico

Director
Jesús David Quintero Polanco
Msc. TIC, Profundización en Telecomunicaciones

UNIVERSIDAD SURCOLOMBIANA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
NEIVA - HUILA
2022

Nota de aceptación

_____

_____

_____

_____

_____

_____

_____

_____

Firma del presidente del jurado

_____

Firma del jurado

_____

Firma del jurado

Neiva, 29 de Junio de 2022

DEDICATORIA

Este trabajo de grado quiero dedicárselo a mis papás, a mi hermano, y novia quienes me apoyaron moral y económicamente, también a esas personas que me dieron su voz de aliento para seguir adelante.

Maicol Andrés García Rodríguez

A Dios por bendecir mi camino, a mis padres que con mucho esfuerzo me apoyaron y guiaron incondicionalmente para crecer personal y profesionalmente, a mis hermanas. A familiares y amigos con quienes conté durante todo este proceso.

Gustavo Andrés Medina Díaz

AGRADECIMIENTOS

Al ingeniero electrónico Jesús David Quintero quien nos guió como director de tesis para desarrollar el proyecto y culminar de la mejor manera posible.

A la Facultad de Ingeniería de la Universidad Surcolombiana nuestra alma máter y a todos los docentes quienes nos brindaron las herramientas necesarias para prepararnos académicamente y adquirir los conocimientos que tenemos actualmente.

Finalmente a nuestros padres, familiares, amigos y a todas aquellas personas que nos apoyaron y pusieron un granito de arena para que este proyecto saliera adelante.

CONTENIDO

Pág

LISTA DE FIGURAS

Pág

LISTA DE ANEXOS

Pág

GLOSARIO

API: Interfaz de programación de aplicaciones que contiene un conjunto de funciones y procedimientos para realizar una o varias funciones con el fin de ser utilizadas por otro software.

APP: abreviatura de la palabra inglesa Application. Es una aplicación de software que realiza funciones específicas y trabaja en diversos dispositivos como teléfonos, relojes inteligentes, tablets e incluso televisores.

FRAMEWORK: esquema de trabajo para desarrollar e implementar software en diversos lenguajes de programación.

HTTP: protocolo de transferencia de hipertexto el cual permite la transferencia de datos y navegación en la web.

JSON: notación de objetos de JavaScript, formato de intercambio de datos que permite transferir información entre cliente y servidor.

MÉTODO: función que contiene un conjunto de instrucciones, las cuales se ejecutan cuando se llame a dicha función.

Middleware: software que provee un mecanismo para filtrar las peticiones HTTP de una aplicación.

Postgresql : Es un sistema de gestión de bases de datos relacional orientado a objetos y de código abierto, publicado bajo la licencia PostgreSQL.

Django: Es un framework de desarrollo web de código abierto que se encuentra escrito en Python y que respeta el patrón de diseño conocido como modelo–vista–controlador.

REST: transferencia de estados representacional, es una arquitectura que permite realizar la comunicación entre cliente y servidor basado en el protocolo HTTP, lo cual permite obtener y generar datos u operaciones que se entregan en formatos como el JSON.

Retrofit: Retrofit es un cliente de servidores REST para Android y Java desarrollado por Square, muy simple y muy fácil de aprender. Permite hacer peticiones al servidor tipo GET, POST, PUT, PATCH, DELETE y HEAD.

Docker: Docker es un sistema operativo (o runtime) para contenedores. Proporciona un conjunto sencillo de comandos que puede utilizar para crear, iniciar o detener contenedores.

Kubernetes: Kubernetes es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Kubernetes facilita la automatización y la configuración declarativa.

RESUMEN

Se realizó la implementación de un producto mínimo viable del sistema experto para el diagnóstico de hipertensión y diabetes con Machine Learning desarrollado en Python donde se probaron diferentes métodos tradicionales para determinar cuál se ajustaba mejor a los datos y entregaba un mayor porcentaje de predicción.

Se desarrolló una aplicación web con el framework de Django que permite a los usuarios realizar un proceso de diagnóstico a través de una interfaz gráfica con diferentes formularios que la hacen fácil y agradable de utilizar, así como también llevar un historial de las consultas realizadas mediante la utilización del gestor de bases de datos PostgreSQL. Además, se utilizó la librería Django Rest Framework para crear una API que interactúa e intercambia información con una aplicación móvil desarrollada en Android Studio.

La aplicación móvil permite a los usuarios registrarse, hacer un proceso de login y realizar el proceso de diagnóstico de hipertensión y diabetes haciendo uso de la librería Retrofit para la comunicación con la API.

PALABRAS CLAVE

API, framework, aprendizaje de máquina, Django, algoritmos, predicciones, aplicación móvil.

ABSTRACT

The implementation of a minimum viable product of the expert system for the diagnosis of hypertension and diabetes was carried out with Machine Learning developed in Python where different traditional methods were tested to determine which one best fit the data and delivered a higher percentage of prediction.

A web application was developed with the Django framework that allows users to carry out a diagnostic process through a graphical interface with different forms that make it easy and pleasant to use, as well as keeping a history of the queries made through the use of the PostgreSQL database manager. In addition, the Django Rest Framework library was used to create an API that interacts and exchanges information with a mobile application developed in Android Studio.

The mobile application allows users to register, perform a login process and carry out the hypertension and diabetes diagnosis process using the Retrofit library for communication with the API.

KEYWORDS:

API, framework, Machine Learning, Django, algorithms, predictions, mobile application.

1. INTRODUCCIÓN

En la última década, las enfermedades de hipertensión y diabetes han sido una de las principales causas de muertes a nivel mundial, ya que son pocas las personas que poseen estas enfermedades y son conscientes de padecerlas, esto se debe a deficiencias en el sistema de salud en cuanto a no llevar a cabo campañas de prevención, información y acciones de autocuidado para conservar la salud y vida. Tanto la hipertensión y la diabetes son enfermedades que generan múltiples complicaciones como: problemas cardiovasculares, daño a las arterias, nervios, daño en los órganos, en los ojos, extremidades, entre otros. Por lo que es de gran importancia tomar medidas de autocuidado.

Teniendo en cuenta lo anterior, en la actualidad algunas organizaciones y universidades, han desarrollado diferentes proyectos que facilitan el diagnóstico médico de enfermedades de hipertensión y diabetes. Proyectos que se han enfocado en diagnosticar solo una de estas enfermedades con porcentajes de precisión bajas.

Debido a esto, se desarrolló un sistema experto que facilita el diagnóstico de las dos enfermedades en una misma aplicación, utilizando métodos de pre-procesamientos y algoritmos de Machine Learning que permiten mejorar el porcentaje de precisión de diagnóstico. De igual forma se usó el framework Django, que permite simplificar el desarrollo de sitios web complejos y facilita la implementación de la Api Rest Full Django (DRF), logrando la comunicación entre los pacientes mediante la App Android creada y las peticiones HTTP con la página web.

2. OBJETIVOS

2.1. OBJETIVO GENERAL

Desarrollo de un sistema experto para el apoyo al diagnóstico médico de enfermedades de hipertensión y diabetes a través de Machine Learning.

2.2. OBJETIVOS ESPECÍFICOS

- Investigar el estado del arte de los diferentes algoritmos de Machine Learning y pre-procesamiento para la predicción de enfermedades de hipertensión y diabetes.
- Realizar el algoritmo de Machine Learning para la predicción de enfermedades de hipertensión y diabetes.
- Desarrollar el sistema experto mediante el diseño del modelo vista controlador MVC de la aplicación nativa y la aplicación web.
- Realizar la validación de los aplicativos web y móvil, junto con la precisión del algoritmo de aprendizaje desarrollado, y elaborar el documento final.

3. MARCO TEÓRICO

3.1. HIPERTENSIÓN

La hipertensión, muy conocida como presión arterial alta, es un trastorno muy frecuente que afecta a un tercio de la población adulta, donde los vasos sanguíneos presentan una tensión permanentemente alta, lo que puede dañarlos. La presión arterial es la fuerza que produce la sangre contra las paredes de los vasos(arterias) al ser bombeadas por el corazón. Una de las características de la hipertensión es que no presenta síntomas claros, por ello se le conoce como el “asesino silencioso” y puede tardar mucho tiempo en manifestarse, de igual forma presenta uno de los factores de riesgos cardiovasculares más prevalentes.

En adultos la presión arterial normal cuando el corazón late (presión sistólica) es de 120 mm Hg y cuando el corazón esta relajado (presión diastólica) es de 80 mm Hg. La presión arterial se considera alta cuando la presión sistólica es mayor o igual a 140 mm Hg y/o la presión diastólica es mayor o igual a 90 mm Hg. La hipertensión es la causa prevenible más importante de enfermedades cardiovasculares del mundo y si no es controlada puede provocar un ensanchamiento en el corazón, infarto de miocardio, insuficiencia cardiaca, daños en las arterias, dilatación en el ventrículo izquierdo, accidente isquémico transitorio, insuficiencia renal, daño en los vasos sanguíneos de los ojos, coroidopatía y neuropatía óptica. Los efectos de la hipertensión para la salud se pueden empeorar por otros factores como lo es una dieta poco saludable, el consumo de tabaco, la inactividad física, el uso nocivo de alcohol, la diabetes, la obesidad y el colesterol alto [8].

3.2. DIABETES

La diabetes es una enfermedad incurable que se produce porque el páncreas no genera suficiente insulina o el cuerpo no utiliza de manera eficiente la insulina que produce. La insulina es una hormona generada por el páncreas. La cual se encarga del mantenimiento de los valores exactos de glucosa en la sangre. La insulina permite que la glucosa sea capaz de ingresar al organismo y llegue al interior de las células, para transformarse en energía y funcionen de manera adecuada los músculos y tendones.

La diabetes se puede presentar en diferentes tipos: diabetes tipo 1, diabetes tipo 2 y diabetes gestacional. La diabetes tipo 1, se le diagnostica a niños y jóvenes. Se produce debido a que el cuerpo no genera insulina porque el sistema inmunitario destruye cada una de las células del páncreas. Por lo tanto, estas personas tienen que usar insulina todos los días para poder vivir. La diabetes tipo 2, puede aparecer en

cualquier edad y es el tipo más común de diabetes. Se produce porque el cuerpo no genera o no usa la insulina adecuadamente. La diabetes gestacional, se presenta en algunas mujeres en embarazo y desaparece en la mayoría de los casos cuando nazca el bebé [9].

3.3. SISTEMA EXPERTO

Es un sistema basado en computadoras, interactivas y confiables. Que usa conocimiento de un área de aplicación compleja y actúa como un consultor experto para los usuarios finales. También, Proporcionan soluciones a problemas muy específicos al hacer inferencias iguales a la de los humanos sobre los conocimientos específicos. Los sistemas expertos se identifican por el alto nivel de experiencia que proporciona precisión, eficiencia y resolución imaginativas de problemas [10].

3.4. SISTEMA OPERATIVO ANDROID

Android es una plataforma de software para dispositivos móviles que incorpora un sistema Operativo y aplicaciones de base. También mezcla herramientas y aplicaciones que están vinculadas a una distribución Linux para dispositivos móviles y es de código abierto, donde se pueden crear aplicaciones para plataformas usando el SDK de Android para dispositivos móviles [11].

3.5. MODELO VISTA CONTROLADOR (MVC)

El modelo vista controlador (MVC) es un patrón de arquitectura de software que separa una aplicación en tres componentes fundamentales, los cuales son el Modelo, la Vista y el Controlador, que se trabajan de forma independiente, permitiendo que la aplicación sea mucho más fácil de desarrollar y más organizada. Muchos frameworks modernos implementan MCV para la arquitectura, de los cuales podemos mencionar AngularJS, Ruby on Rails y Django [12].

En el caso específico de Django, a pesar de que sigue en teoría el patrón MVC, en éste el Controlador, "C" es manejada por el mismo framework y la parte más importante se produce en los modelos, las plantillas y las vistas, Django es conocido como un Framework MTV [13]. Donde MTV significa:

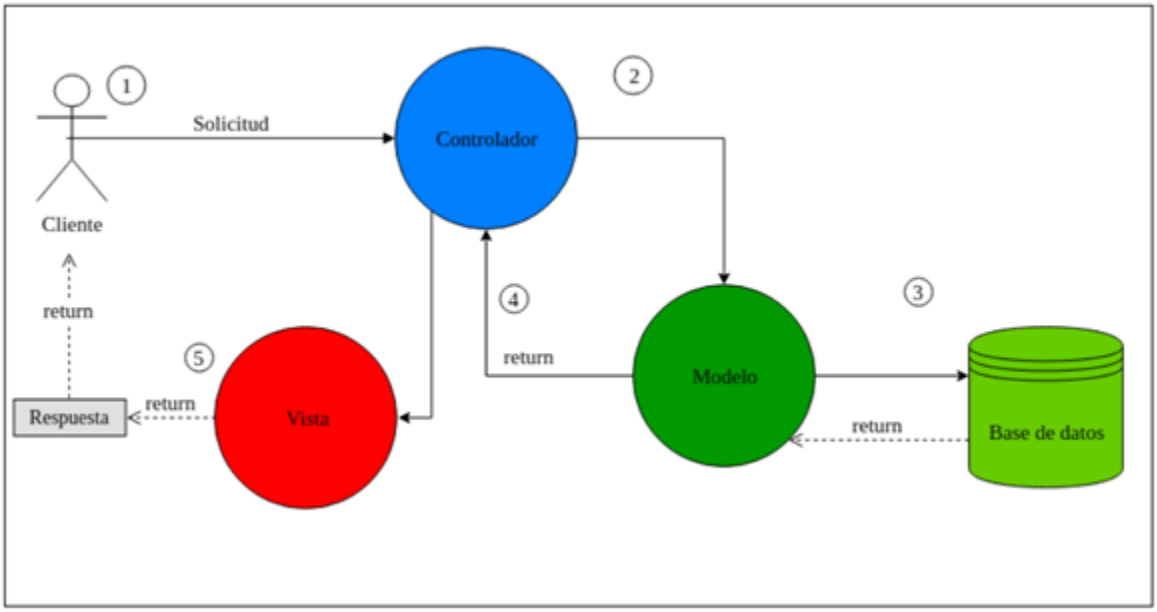
M de "Model" (Modelo), la capa de acceso a la base de datos.

T significa "Template" (Plantilla), la capa de presentación.

V significa "View" (Vista), la capa de la lógica de negocios. Esta capa contiene la lógica que accede al modelo y la delega a la plantilla apropiada.

La razón de que se utilice MVC es que permite separar los componentes de una aplicación, esto quiere decir que, al momento de modificar una parte del código, esto no afecta a otra parte del mismo. Por ejemplo, si se llega a modificar la base de datos, solo se debe modificar el modelo ya que representa la estructura lógica de los datos en una aplicación. Siendo un puente de comunicación entre el controlador, la base de datos y la vista. El controlador es el encargado de aceptar y controlar las solicitudes que hace el usuario, encargándose de solicitar los datos al modelo y, por último, toma la vista adecuada para mostrar los datos al usuario. La vista en cambio es la representación visual de los datos, esto quiere decir que va todo lo que tenga que ver con la interfaz gráfica, todo esto se puede observar de una forma más clara en la **figura 1** [14].

Figura 1. Diagrama vista controlador (MVC). (imagen de [14])



3.6. MACHINE LEARNING

Machine Learning o aprendizaje automático, es una de las ramas de la inteligencia artificial que se encarga de crear sistemas que permitan a las máquinas o computadores aprender, sin la necesidad de estar expresamente programadas para ello. Lo que resulta una habilidad indispensable para hacer, no solo sistemas inteligentes, sino también autónomos, y capaces de identificar patrones entre los datos para poder hacer predicciones. Machine Learning tiene un aspecto iterativo importante porque a medida que los modelos son expuestos a nuevos datos, éstos pueden adaptarse de forma independiente. Aprenden de cálculos previos para producir decisiones y resultados confiables y repetibles. Es una ciencia que no es nueva – pero que ha cobrado un nuevo impulso.

El aprendizaje automático tiene una amplia gama de aplicaciones, incluyendo motores de búsqueda, diagnósticos médicos, detección de fraude en el uso de tarjetas de crédito, análisis del mercado de valores, clasificación de secuencias de ADN, reconocimiento del habla y del lenguaje escrito, juegos y robótica.

¿Por qué es importante Machine Learning?, Con el aprendizaje automático es posible producir modelos de manera rápida y automática que puedan analizar datos más grandes y complejos y producir resultados más rápidos y precisos – incluso en una escala muy grande. Y con la construcción de modelos precisos, una organización tiene una mejor oportunidad de identificar oportunidades rentables – o de evitar riesgos desconocidos [15].

Métodos de Machine Learning: Dos de los métodos de aprendizaje basado en máquina más ampliamente adoptados son aprendizaje supervisado y aprendizaje no supervisado – pero existen también otros métodos de machine Learning.

Los algoritmos de aprendizaje supervisado producen una función que establece una correspondencia entre las entradas y las salidas deseadas del sistema. Un ejemplo de este tipo de algoritmo es el problema de clasificación, donde el sistema de aprendizaje trata de etiquetar (clasificar) una serie de vectores utilizando una entre varias categorías (clases). La base de conocimiento del sistema está formada por ejemplos de etiquetados anteriores [16].

El aprendizaje no supervisado se utiliza contra datos que no tienen etiquetas históricas. No se da la "respuesta correcta" al sistema. El algoritmo debe descubrir lo que se muestra. El objetivo es explorar los datos y encontrar alguna estructura en su interior [16]. El sistema tiene que ser capaz de reconocer patrones para poder etiquetar las nuevas entradas.

Técnicas de clasificación: existen diferentes técnicas o algoritmos que nos permiten clasificar unos datos de entrada de acuerdo a unas etiquetas de salida, entre las técnicas se encuentran árboles de decisión, máquinas de soporte vectorial, bosques aleatorios, redes bayesianas, entre otros.

3.6.1. K-NEAREST-NEIGHBOR

K vecinos más cercanos es un algoritmo de clasificación de tipo supervisado de machine learning. El cual simplemente busca en las observaciones más cercanas a la que se está tratando de predecir y clasifica el punto de interés basado en la mayoría de datos que le rodean. Este algoritmo por ser uno de los más básicos y esenciales en machine learning es muy implementando en el reconocimiento de patrones, sistema de recomendación, minería de datos, plataforma de contenido digital y detección de intrusos. [23]

3.6.2. ÁRBOLES DE DECISIÓN

Árboles de decisión es un algoritmo de clasificación de tipo supervisado de machine learning. El cual está estructurado como un diagrama de flujo que permite la toma de decisiones. El algoritmo está constituido por diferentes tipos de nodos, donde un nodo interno representa una característica, la rama representa una regla de decisión y cada nodo hoja representa el resultado. [24]

3.6.3. MÁQUINAS DE VECTORES DE SOPORTE (SVM)

Máquinas de vectores de soporte es un algoritmo que se puede usar para regresión y clasificación de tipo supervisado. Este algoritmo con dos o más clases de datos etiquetados, actúa como un clasificador trazando un hiperplano óptimo que separa todas las clases, siendo el hiperplano el límite de decisión [25].

3.6.4. NAIVE BAYES

Naive Bayes es un algoritmo de clasificación de tipo supervisado de machine learning. Este algoritmo proporciona una forma en la que se puede calcular la probabilidad de una hipótesis dado un conocimiento previo [26].

3.6.5. BOSQUES ALEATORIOS

Bosques Aleatorios es un algoritmo de clasificación de tipo supervisado de machine learning. Este algoritmo implementa conjuntos de árboles de decisión los cuales permiten que distintos árboles vean diferentes porciones de datos, logrando que ningún árbol vea todos los datos de entrenamiento [27]. Esto proporciona que cada árbol se entrene con una gran variedad de datos de un mismo problema. De esta forma, al combinar todos los resultados, unos errores se compensan con otros, logrando una predicción que generaliza mejor.

3.7. PREPROCESAMIENTO DE DATOS

El preprocesamiento de datos es un paso fundamental en la minería de datos. Siendo una etapa esencial del proceso de descubrimiento de información. Ya que muchos de

los datos no están limpios, presentando ausencia de información clave, valores atípicos, entre otros.

La imperfección en la limpieza de datos es uno de los grandes problemas que permiten que se presenten resultados erróneos en los algoritmos de clasificación. En el preprocesamiento se realizan algunas de las siguientes tareas:

- Corregir inconsistencias
- Identificar y eliminar datos que se pueden considerar un ruido
- Rellenar valores faltantes
- Resolver redundancia

3.8. ANÁLISIS DE COMPONENTES PRINCIPALES (PCA)

El análisis de componentes principales es una técnica para disminuir la dimensión del conjunto de datos, esto quiere decir que sirve para encontrar las causas de la variabilidad de un conjunto de datos y ordenarlas por importancia. El ACP permite representar los datos originales, en una especie de dimensión inferior del espacio original, mientras limita al máximo la pérdida de información [29]

3.9. POSTGRESQL

PostgreSQL es un sistema de gestión de base de datos relacional y está orientado a objetos, siendo multiplataforma y open source.

3.10. API REST

API REST es un método de arquitectura de desarrollo Web que se apoya en HTTP, y que a través de los métodos HTTP (GET, POST, PUT, DELETE) se puede implementar para ejecutar operaciones con los datos de cualquier aplicación, permitiendo gozar de las ventajas como lo es: facilitar el desarrollo del frontend, crear arquitecturas orientadas a servicios y exponer datos a otros programas.

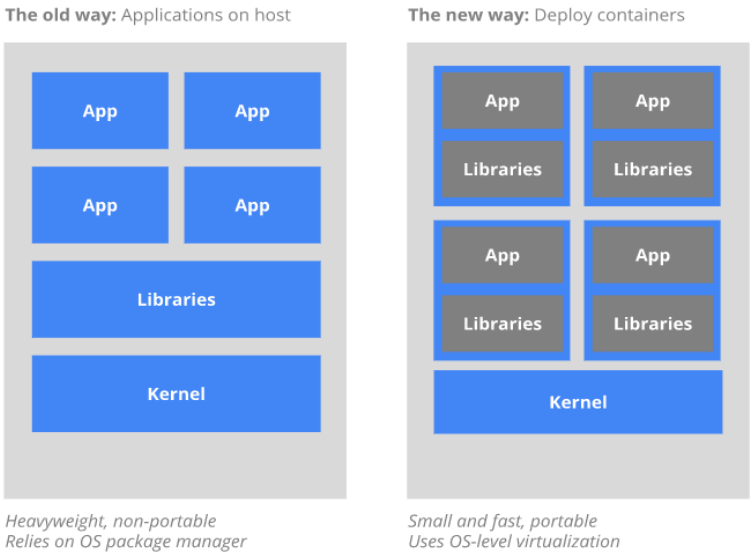
3.11. DOCKER

El sistema de software de TI llamado "Docker" es la tecnología de organización en contenedores que posibilita la creación y el uso de los contenedores de Linux®.

Con Docker, podrá utilizar los contenedores como máquinas virtuales muy livianas y modulares, y obtendrá la flexibilidad necesaria para crearlos, implementarlos, copiarlos y trasladarlos de un entorno a otro, lo cual le permite optimizar sus aplicaciones para la nube.

Existen dos diferentes maneras de desplegar aplicaciones, una antigua y una nueva.

Figura 2. Maneras de desplegar aplicaciones.



Fuente: Tomado de la página oficial de Kubernetes [33]

La *Manera Antigua* de desplegar aplicaciones era instalarlas en un servidor usando el administrador de paquetes del sistema operativo y la *Manera Nueva* es desplegar contenedores basados en virtualización a nivel del sistema operativo, en vez del hardware. Estos contenedores están aislados entre ellos y con el servidor anfitrión: tienen sus propios sistemas de archivos, no ven los procesos de los demás y el uso de recursos puede ser limitado [33].

Entre los beneficios de usar contenedores están:

- Ágil creación y despliegue de aplicaciones: Mayor facilidad y eficiencia al crear imágenes de contenedor en vez de máquinas virtuales
- Desarrollo, integración y despliegue continuo: Permite que la imagen de contenedor se construya y despliegue de forma frecuente y confiable, facilitando los rollbacks pues la imagen es inmutable
- Aislamiento de recursos: Hace el rendimiento de la aplicación más predecible
- Utilización de recursos: Permite mayor eficiencia y densidad

3.12. KUBERNETES

Kubernetes es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Kubernetes facilita la automatización y la configuración declarativa. Tiene un ecosistema grande y en rápido crecimiento. El soporte, las

herramientas y los servicios para Kubernetes están ampliamente disponibles. Es muy usado para la orquestación de contenedores [33].

3.13. AMAZON ELASTIC KUBERNETES SERVICE (EKS)

Amazon Elastic Kubernetes Service (Amazon EKS) es un servicio de contenedores administrado para ejecutar y escalar aplicaciones Kubernetes en la nube o en las instalaciones [35].

4. DESARROLLO ALGORITMO DE MACHINE LEARNING

En este capítulo se explicará el desarrollo del algoritmo de Machine Learning que se encargará de la predicción de hipertensión y diabetes, esto mediante el uso de la interfaz de escritorio Anaconda Navigator que permite iniciar aplicaciones y administrar fácilmente paquetes, entornos como Spyder y JupyterLab, y canales de conda sin usar líneas de comandos, y bajo el lenguaje de programación Python 3 que cuenta con licencia de código abierto y permite su utilización en cualquier situación y costo.

4.1. BASES DE DATOS

Para el desarrollo del algoritmo, se utilizaron dos bases de datos de uso libre con fines académicos, que contienen datos que son necesarios para la predicción tanto de hipertensión cómo de diabetes, la base de datos utilizada para la predicción de hipertensión fue obtenida de la página web Datos Abiertos de Colombia, y la base de datos utilizada para diabetes es la Pima Indians Diabetes Database (PIDD) la cual ha sido ampliamente utilizada y estudiada en el diagnóstico de diabetes.

Entre las columnas necesarias para la predicción de hipertensión están la edad (Años), presión arterial sistólica (mm Hg), Índice de masa corporal (IMC), Peso (Kg), y los Atributos necesarios para la predicción de diabetes están la edad (años), presión arterial sistólica (mm Hg), espesor de piel (mm), insulina (mg/dl), índice de masa corporal (IMC) y la función de pedigrí de diabetes (FPD). La función de pedigrí de diabetes es como su nombre lo dice una función creada por un grupo de investigadores Smith et al. en su trabajo Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus [21], la cual utiliza datos de los antecedentes familiares del paciente cómo el parentesco del familiar, la edad a la cual fue diagnosticado el familiar entre otros.

4.1.1. Función de Pedigrí de Diabetes

Consiste en una función que puntúa la probabilidad de diabetes según los antecedentes familiares de acuerdo a la historia y relación genética con el paciente. La función de pedigrí de diabetes (DPF en inglés) fue desarrollada por Smith et al. [21] y proporciona una medida de la influencia genética esperada de los parientes afectados y no afectados sobre el eventual riesgo de diabetes del paciente. La función del pedigrí de la diabetes se representa de la siguiente forma:

$$DPF = \frac{\sum_i K_i\left(88 - ADM_i\right) + 20}{\sum_j K_j\left(ALC_j - 14\right) + 50}$$

Donde,

i hace referencia a todos los parientes, que han desarrollado diabetes hasta la fecha del examen del paciente.

j hace referencia a todos los parientes, que no han desarrollado diabetes hasta la fecha del examen del paciente.

Kx es el porcentaje de genes compartidos por el pariente. Y equivale a:

0.500 cuando el pariente es padre o hermano completo, 0.250 cuando el pariente es medio hermano, abuelo, tía o tío, y 0.125 cuando el pariente es mitad tía o mitad tío.

ADM_i es la edad en años del familiar cuando se le diagnosticó diabetes.

ALC_j es la edad en años del familiar que no ha sido diagnosticado con diabetes.

Las constantes 88 y 14 representan las edades máxima y mínima en

qué familiares de los pacientes generalmente desarrollan diabetes. Y 50 y 20 son valores para ajustar los resultados de la función

4.2. LIMPIEZA Y PREPROCESAMIENTO DE LOS DATOS

Para comenzar el desarrollo nuestro algoritmo de Machine Learning es necesario realizar una limpieza y preprocesamiento de los datos con los que vamos a trabajar. Para realizar la limpieza primero se debe importar todas las librerías con las que se trabajará para realizar este proceso, entre las librerías que usamos están Pandas, y numpy las cuales nos sirven para manipular y analizar los datos como arreglos y dataframes, Otras librerías utilizadas están la librería para visualizar los resultados de mediante gráficas como matplotlib y seaborn, y librerías para Machine Learning como Scikit-learn.

Luego de importar las librerías se procede a cargar los datos con los que se va a trabajar a nuestro entorno de trabajo el cual es JupyterLab, y ya que nuestros datos se encuentran en un archivo CSV la tarea de cargar los datos resulta bastante fácil, solo es necesario utilizar el comando de la librería pandas “read_csv(nombrearchivo.csv)”. Luego de haber cargado la base de datos, podemos proceder a analizar el tipo de dato de cada columna y una breve descripción sobre ellos mediante el comando dtypes y decribe().

Figura 3. Tipos de datos de cada columna de base de datos

```
[25]: data_h.dtypes

[25]: Edad (años)                                                  int64
      Grupo de edad                                               object
      Genero                                                       int64
      Etnia                                                       object
      Zona                                                        object
      Escolaridad                                                  int64
      Fumador Activo                                              object
      ¿Diabetes?                                                  object
      Hipertensión Arterial Sistemica                             object
      HTA + DM                                                    object
      Clasificacion de Diabetes o del ultimo estado de Glicemia    int64
      Complicaciones  y Lesiones en Organo Blanco                 object
      Antecedentes Fliar  Enfermedad Coronaria                    object
      Tension SISTOLICA                                            int64
      Tension DIASTOLICA                                           int64
      HTA COMPENSADOS                                             object
      Colesterol Total                                             int64
      Colesterol HDL                                              object
      Triglicéridos                                               object
      Colesterol LDL                                              object
      CALCULO DE RIESGO DE Framingham (% a 10 años)               object
      Clasificación de RCV Global                                 object
      Glicemia de ayuno                                          float64
      Perímetro Abdominal                                        float64
      Clasificación perímetro abdominal                           object
      Peso                                                         int64
      Talla                                                        int64
      IMC                                                          int64
```

Fuente: elaboración propia

Se comprueba el tipo de datos de cada columna y se separan para poder observar la cantidad de columnas de tipo numérico y categórico. Estos primeros pasos se realizan para ambos datasets, el de hipertensión y diabetes, y a continuación se pueden observar cómo funcionan:

Ahora vamos a comenzar a realizar la limpieza de los datos, primero comprobando la cantidad de valores nulos presentes entre los datos, analizando cada columna, eliminando las columnas con mayor número de valores nulos, ya que no aportan valores suficientes con los cuales trabajar, y también se procede a eliminar todas las columnas que no aportan nada a nuestro desarrollo, pero se encuentran presentes en el dataset. Para eliminar los valores nulos se utiliza el comando *dropna* que permite eliminar valores de archivos dataframe. Aquellas columnas que tengan un número pequeño de datos nulos fueron rellenados con el valor medio para cada columna, y luego se puede comprobar si aún existen valores nulos y suma la cantidad mediante el código *data_h.isnull().any().sum()*, como se muestra a continuación:

Figura 4. Sumatoria de valores null por columna.

```
[34]: data_h.isnull().sum()

[34]: Num.                                0
      subject_ID                          0
      Sex(M/F)                            0
      Age(year)                           0
      Height(cm)                          0
      Weight(kg)                          0
      Systolic Blood Pressure(mmHg)       0
      Diastolic Blood Pressure(mmHg)      0
      Heart Rate(b/m)                     0
      BMI(kg/m^2)                         0
      Hypertension                        0
      Diabetes                          181
      cerebral infarction               199
      cerebrovascular disease           194
      dtype: int64
```

Fuente: elaboración propia

Luego, se completarán los campos nulos hallados utilizando la media de cada columna respectivamente, y para los datos de tipo categóricos se tomó la decisión de eliminar las filas que contengan datos nulos para evitar que estos tengan un efecto negativo en nuestro algoritmo.

Utilizando el componente preprocessing de la librería Sklearn se importó LabelEncoder(), la cual es una función de esta librería que nos permite realizar una codificación de las variables categóricas presentes en la tabla por variables numéricas , por ejemplo para la salida de hipertensión será 0 si la variable es NO y 1 si la variable es SI. Este proceso se realiza para las variables categóricas necesarias para la realización del algoritmo. Luego de esto, se procede a observar la correlación presente en todos los atributos en relación con los datos de salida y determinar qué variables tienen una mayor relación. Por lo cual los atributos útiles y que fueron elegidos para la realización de nuestro algoritmo fueron la edad, la presión arterial sistólica, altura, peso, Índice de Masa Corporal (IMC).

Se observaron los datos de la salida de nuestra base de datos los cuales se utilizaran para realizar el entrenamiento, y se construyó una gráfica con python que nos permitirá observar la cantidad filas que presentan hipertensión, ya sea prehipertensión,

hipertensión en etapa 1, o hipertensión en etapa 2, y las filas que no presentan hipertensión en nuestra base de datos.

4.2.1. Algoritmo de preprocesamiento PCA

Luego de la limpieza de los datos de entrada se utilizará el algoritmo de preprocesamiento Principal Component Analysis o PCA, por sus siglas en inglés, que nos permite combinar variables de entrada de una manera específica y eliminar algunas variables menos importantes. Este algoritmo se puede importar en python mediante la librería de sklearn, mediante el comando from sklearn.decomposition import PCA, después de esto se llama el número de componentes que utilizará el algoritmo para entregar el resultado, y para finalizar utilizamos fit_transform para transformar nuestros datos mediante PCA, y quedando todo listo para realizar el entrenamiento de nuestro algoritmo de detección de hipertensión o diabetes.

4.3. ALGORITMO DE PREDICCIÓN

Después de realizar todo el preprocesamiento anterior, los datos se encuentran listos para poder entrenar el algoritmo, y lo primero que vamos a hacer será crear un arreglo que tendrá los atributos de entrada y un arreglo que tendrá los valores de salida, luego se procede a dividir los datos en datos de entrenamiento y en datos de test. Los datos de entrenamiento serán utilizados para entrenar nuestro algoritmo de machine learning y de esa manera obtener un resultado, por otra parte, los datos de test serán utilizados para comprobar el correcto funcionamiento del algoritmo entrenado y la precisión correspondiente.

El comando para dividir los datos es train_test_split, importado de la librería sklearn de la siguiente manera, los parámetros utilizados en este comando son el arreglo de entrada, el arreglo de salida, test_size que indica el tamaño de los datos de test que retorna el comando, y random_state, que es un valor que se usa básicamente para que cada vez que se ejecute el comando test_train_split se obtenga el mismo conjunto de datos de entrenamiento y de test.

Luego, se procede a importar el algoritmo a utilizar desde la librería sklearn, se crea el algoritmo y se varían sus parámetros de entrada para poder obtener la mejor respuesta posible, y luego se entrena el algoritmo mediante el arreglo de datos de entrenamiento.

Después, se utiliza la función predict, que utilizará el algoritmo entrenado y los datos de entrada de test para predecir la salida, la cual será hipertensión y diabetes. los datos que entrega como resultado este comando se utilizara para obtener el porcentaje de precisión del algoritmo, se utilizará la salida predecida y la salida de test, y se evalúa en qué no se cumple que ambas salidas tengan igual valor.

Todo este proceso se realizará para diferentes algoritmos de machine learning, y así obtener el mejor resultado para nuestro conjunto de datos, los algoritmos evaluados son Arboles de decisión, Random Forest, KNN, Maquina de vectores de soporte y Naive Bayes, y las respuestas obtenidas para cada algoritmo son resumidas en la siguiente tabla.

Tabla 1. Resultado Entrenamiento Algoritmos de predicción

Métodos de Machine Learning	% Predicción Hipertensión	% Predicción Diabetes
Árbol de decisión	0.94	0.89
Random Forest	0.96	0.88
KNN	0.98	0.90
Máquina de vectores de soporte (SVC)	0.99	0.91
Naive Bayes	0.94	0.88

Luego de tener el algoritmo con mejor respuesta, se encapsula el algoritmo y se exporta mediante la librería pickle de python, los cual mediante unos pocos comandos nos permite generar un archivo con extensión pkl que contendrá nuestro algoritmo de detección. Todo este proceso se hace con la intención de poder utilizar este algoritmo generado en nuestra interfaz de usuario web, la cual tendrá un formulario para poder ingresar los datos de entrada del algoritmo mediante un método POST, luego procesar estos datos con el algoritmo y entregar el resultado del diagnóstico.

5. DESARROLLO DEL SISTEMA EXPERTO

En este capítulo se explicará el desarrollo del sistema experto, el cual consta de una aplicación web y una aplicación móvil, la aplicación web se desarrolló bajo el framework Django debido a su versatilidad, facilidad de trabajo y que su lenguaje de programación base es Python 3, y se utilizó el entorno de desarrollo Visual Studio Code para construir el proyecto. La aplicación móvil se desarrolló mediante el uso del entorno de desarrollo integrado (IDE) ya que es el entorno oficial para el desarrollo de apps para dispositivos Android.

5.1. DESARROLLO APLICACIÓN WEB EN DJANGO

5.1.1. Configuración inicial

Para trabajar con Django se debe tener instalado Python 3 con anterioridad, y luego instalar la librería de Django mediante la línea de comando *pip install Django==3.2.5*

Esta línea depende de la versión que quieras instalar. Luego de eso, se crea el proyecto mediante el comando *django-admin startproject mysite*, donde mysite es el nombre del proyecto y luego se puede comprobar su funcionamiento mediante la ejecución de un servidor local que viene con la librería, el comando a utilizar es *python manage.py runserver.* Cuando se crea el proyecto se crea una carpeta de archivos que se puede abrir en cualquier editor de código, en nuestro caso utilizamos Visual Studio Code, y donde se puede empezar a programar nuestro proyecto.

5.1.2. Creación de apps para el proyecto

Luego de crear el proyecto es necesario crear aplicaciones para desarrollar cada una de las partes importantes del proyecto, todo esto mediante el comando *python manage.py startapp myapp* donde myapp es el nombre de la app. Para nuestro proyecto creamos 3 apps, una para la creación de registro e inicio de sesión de usuarios, otra para la parte del sistema que tiene que ver con Hipertensión y otra para la parte de Diabetes.

5.1.3. Conexión base de datos

Para la base de datos utilizamos el gestor de base de datos Postgresql, el cual es fácil de utilizar y frecuentemente utilizado con Django. Para la configuración y establecer una conexión con la base de datos es necesario instalar la librería psycopg2 en Python

mediante el comando *pip install psycopg2 y* modificar el archivo *settings.py* del proyecto de la siguiente manera:

Figura 5. Configuración base de datos en Django

Fuente: elaboración propia

Donde ENGINE equivale al motor o librería que se utiliza para realizar la conexión (psycopg2), NAME es el nombre de la base de datos creada en PostgreSQL mediante PgAdmin, USER y PASSWORD son credenciales para poder gestionar la base de datos en PgAdmin, HOST equivale a la dirección IP del servidor, DATABASE_PORT es el puerto de conexión con el gestor de base de datos.

5.2. CREACIÓN DE TABLAS DE DATOS EN DJANGO

Para la creación de las tablas necesarias para el proyecto se debe modificar el archivo del proyecto models.py de cada app, y luego de eso realizar las migraciones en el CMD mediante el comando *python manage.py makemigrations* y luego ejecutar el comando *python manage.py migrate.* El archivo manage.py es el que contiene toda la información de la ejecución del proyecto.

5.2.1. **Creación de tabla usuarios**

Hace referencia a los campos registrados en la tabla *usuarios*. Esta tabla contiene información básica del usuario o paciente: id de tipo incremental (servirá como identificador para relacionar la información con las otras tablas), nombre, apellido, nombre de usuario, correo, contraseña, y confirmación de contraseña. Esta tabla se crea en base a una clase predefinida en Django que se extiende de django.contrib.auth.models la cual tiene por nombre User, y para importar esta clase se

escribe *from django.contrib.auth.models import User*. Está clase contiene otros campos aparte de los mencionados anteriormente, los cuales son last_login que es un campo tipo datetime autogenerado que indica la fecha del último login del usuario, is_stack que indica si el usuario puede entrar al admin site de Django, is_active indica si el usuario es considerado activo, is_superuser que indica si el usuario tiene todos los permisos.

Tabla 2. Campos por defecto del modelo Login

Campo:	Tipo de dato:
username	Charfield
First_name	Charfield
Last_name	Charfield
Email	Emailfield
password	Passwordfield
Is_stack	Booleano
Is_active	Booleano
Is_superuser	Booleano
Last_login	Datefield
Date_joined	Datefield

5.2.2. Creación de tabla hipertensión

Hace referencia a los campos registrados en la tabla hipertensión. Esta tabla contiene información de los datos necesarios para el diagnóstico de hipertensión de cada paciente, donde el campo id de tipo incremental servirá como identificador para relacionarla con la información de las demás tablas, edad en años del paciente, el peso en kilogramos , la presión arterial sistólica que presente el paciente, el índice de masa corporal el cual se haya con la altura y peso del paciente, y por último el usuario_id el cual es el indicador del paciente el cual esté realizando el diagnóstico.

La información del servicio se divide en dos tablas relacionadas para manejar la información más ordenada y segmentada: rutas e imágenes.

Figura 6. Código tabla hipertensión

```
# Create your models here.
class Hipertension(models.Model):
    usuario= models.ForeignKey(User, null=True, blank=True, on_delete=models.CASCADE)
    presion_s = models.DecimalField(max_digits=5, decimal_places=2)
    edad= models.IntegerField(null= True)
    peso= models.IntegerField(null= True)
    imc= models.DecimalField(max_digits=5, decimal_places=2, null= True)
    resultado = models.CharField(max_length=24, null = True)
```

Fuente: elaboración propia

5.2.3. Creación tabla diabetes

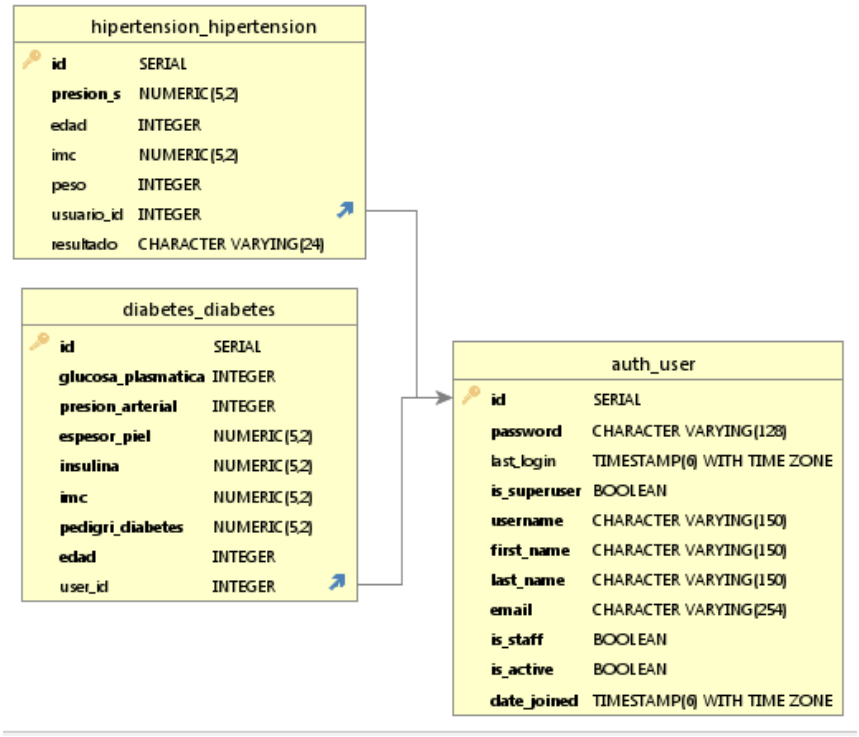
Hace referencia a los campos registrados en la tabla *diabetes*. Esta tabla contiene información de los datos necesarios para el diagnóstico de diabetes de cada paciente, donde el campo id de tipo incremental servirá como identificador para relacionarla con la información de las demás tablas, edad en años del paciente, la presión arterial sistólica que presente el paciente, el espesor de la piel, la insulina, el índice de masa corporal el cual se haya con la altura y peso del paciente, la función de pedigrí de diabetes mencionada anteriormente, y por último el usuario_id el cual es el indicador del paciente el cual esté realizando el diagnóstico.

Figura 7. Código tabla diabetes

```
# Create your models here.
class Diabetes(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE, null=True)
    glucosa_plasmatica = models.IntegerField('Glucosa plasmatica')
    presion_arterial = models.IntegerField('Presion Arterial')
    espesor_piel = models.DecimalField('Espesor de la piel', max_digits=5, decimal_places=2)
    insulina = models.DecimalField('insulina', max_digits=5, decimal_places=2)
    imc = models.DecimalField('Indice masa muscular', max_digits=5, decimal_places=2)
    pedigri_diabetes = models.DecimalField('Pedigri diabetes', max_digits=5, decimal_places=2)
    edad = models.IntegerField('Edad')
```

Fuente: elaboración propia

Figura 8. Relación de tablas base de datos



Fuente: elaboración propia

5.3. ENVÍO Y RECEPCIÓN DE INFORMACIÓN DE BASES DE DATOS

Mediante la gestión de los modelos y controladores que provee Django se realizan consultas y se insertan registros en la base de datos. Para realizar la comunicación con la base de datos en el caso del sistema de registro y autenticación de usuarios se realiza con la ayuda de Django y su libreria auth, la cual permite crear usuarios en la base de datos y realizar las validaciones correspondientes a la autenticación de usuario en el proceso de login.

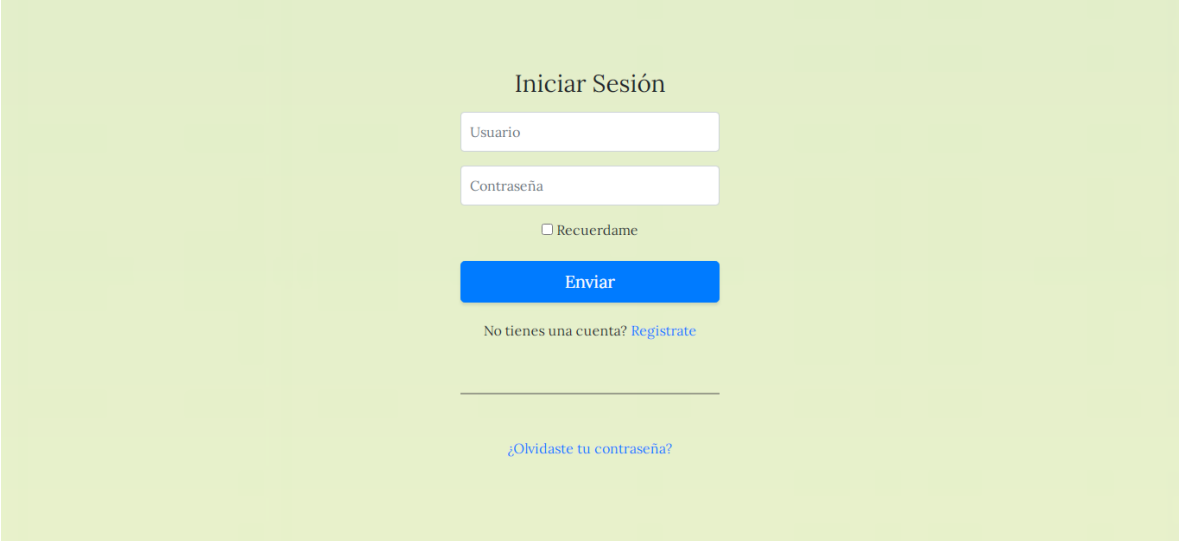
5.4. CREACIÓN DE VISTAS EN DJANGO

Se realizó la creación de las vistas para la aplicación web, comenzando por la vista de login, la cual contiene un formulario para ingresar los datos de usuario y contraseña, una vista de registro con su respectivo formulario y botón, una vista principal llamada home, la cual nos muestra un menú con las opciones de login, registro, diagnóstico de hipertensión y diabetes, y por último las vistas necesarias para realizar el diagnóstico de hipertensión y diabetes.

5.4.1. Vista Login

Para la vista login, se realizó un formulario que contiene los datos de usuario el cual hace referencia al id de usuario registrado al crear la cuenta y la respectiva contraseña. luego se tiene el botón de recuerdame que permite almacenar los datos del usuario para cuando intente ingresar de nuevo y un botón para enviar los datos y realizar la respectiva validación de credenciales, en caso de que las credenciales sean incorrectas se generará un error que aparecerá en pantalla. al final aparecerá un enlace que permitirá registrarse en caso de no tener una cuenta, que nos redirigirá a la vista Registro.

Figura 9. Vista login de la aplicación web

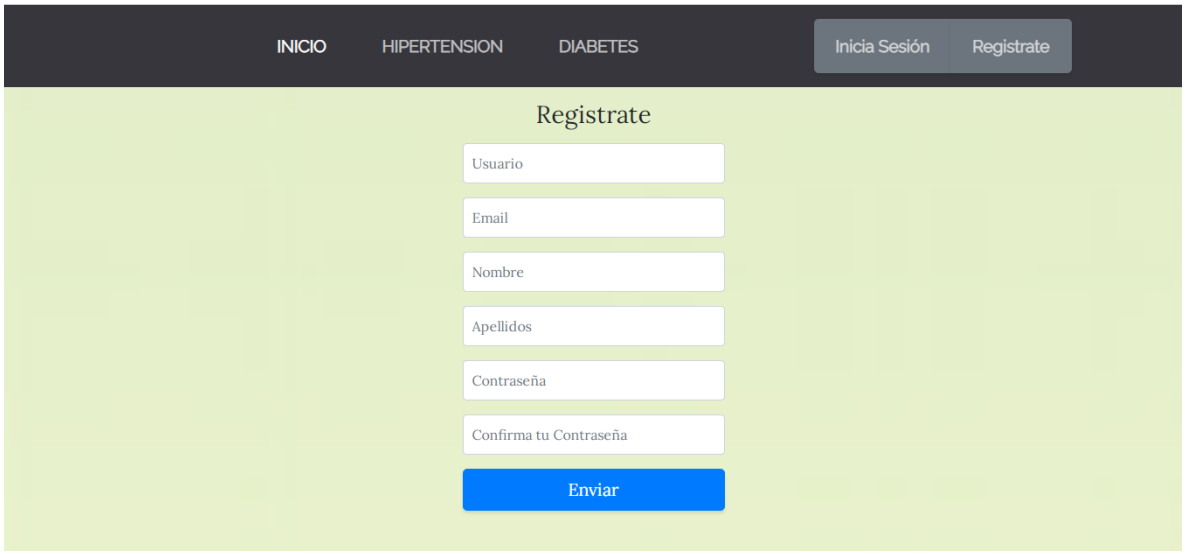


Fuente: elaboración propia

5.4.2. Vista Registro

La vista registro contiene un formulario con los diferentes campos necesarios para que una persona pueda registrarse en la página web, los campos son usuario, email, nombres, apellidos, contraseña y el campo confirmar contraseña, campos típicos utilizados en cualquier sistema de registros. Al final de la página se encuentra el botón enviar, que enviará los datos para su respectiva validación cómo por ejemplo que el id no se encuentre ya registrado en el sistema, el email sea correcto y la contraseña sea adecuada.

Figura 10. Vista registro de la aplicación web

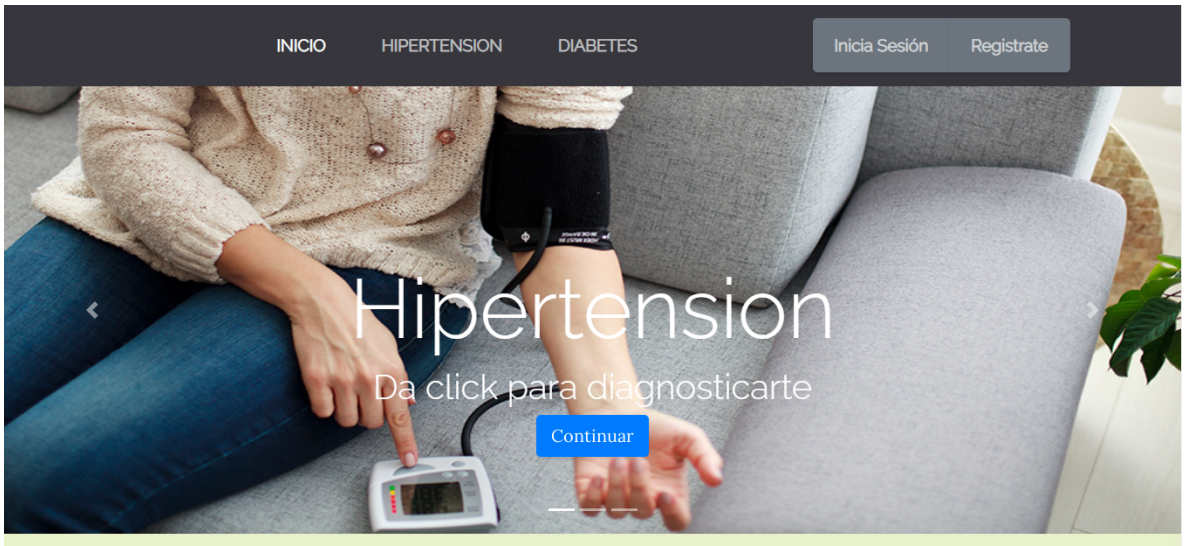


Fuente: elaboración propia

5.4.3. Vista Inicio

La página de inicio contiene la barra de menú superior con sus respectivas opciones cómo inicio, hipertensión, diabetes y los botones de iniciar sesión y registrate. luego se trabaja un poco el diseño de la página con un slide y botones que llevan a diferentes vistas.

Figura 11. Vista Inicio de la aplicación web



Fuente: elaboración propia

5.4.4. Vista diagnóstico hipertensión

En esta vista se encuentra el formulario con todos los datos requeridos para realizar el diagnóstico de hipertensión y un botón de enviar para realizar la validación de los datos y la redirección a los resultados.

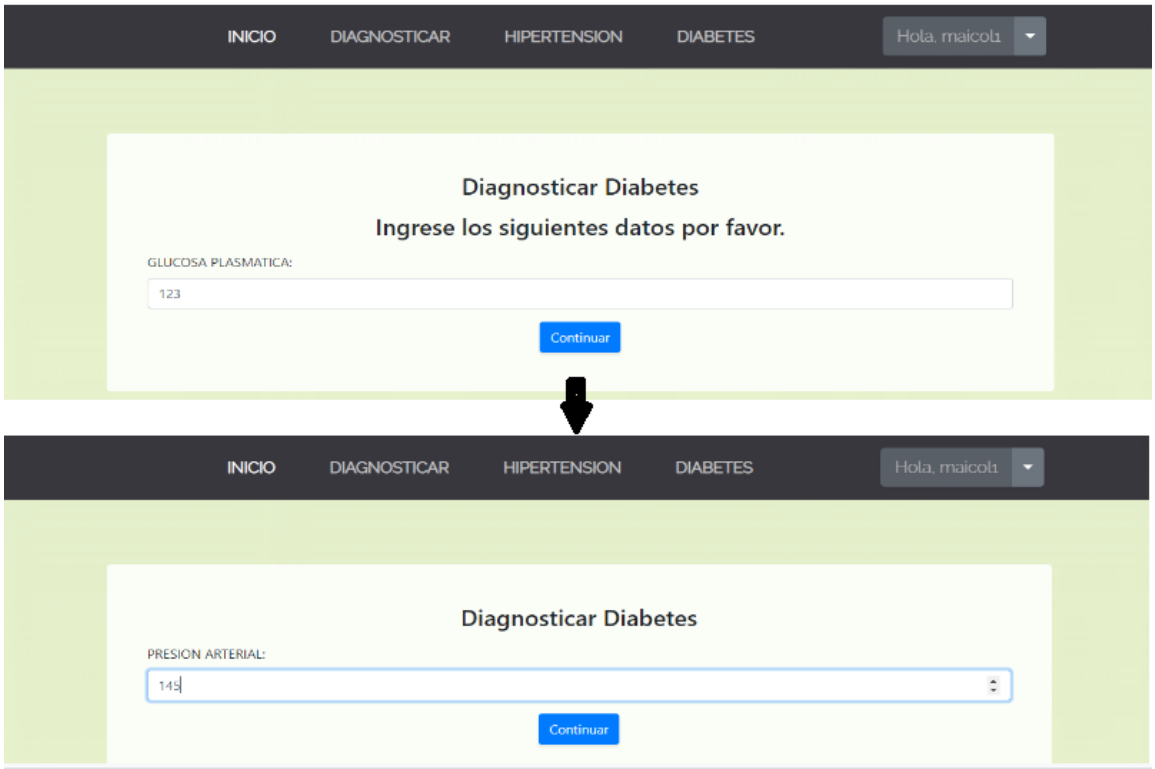
Figura 12. Vista diagnóstico hipertensión de la aplicación web

Fuente: elaboración propia

5.4.5. Vista diagnóstico Diabetes

Para diabetes se crearon varias vistas con diferentes formularios que contienen todos los datos requeridos para realizar el diagnóstico de diabetes y un botón de enviar para realizar la validación de los datos y la redirección a los resultados.

Figura 13. Vista diagnostico diabetes de la aplicación web



Fuente: elaboración propia

Figura 14. Vista resultado diabetes de la aplicación web

Fuente: elaboración propia

5.4.6. Vista Historial

La vista historial mostrará todos los diagnósticos hechos por un usuario, ya sea de hipertensión o diabetes, los datos se mostrarán en una tabla con todos los campos registrados. Para mostrar estos datos se realizó una consulta a la base de datos.

Figura 15. Vista login de la aplicación web

Fuente: elaboración propia

5.5. CREACIÓN DE CONTROLADORES EN DJANGO

Contienen la lógica para el manejo de las peticiones del sistema, y la logica utilizada para la visualización de las vistas y demás procesos que se realicen en éstas. Los controladores se encuentran o se realizan en el archivo views.py creado automáticamente a la hora de crear el proyecto de Django.

5.5.1. Controlador Registro

- Maneja la lógica proveniente de la vista Registro, valida los datos del formulario y se comunica con la base de datos para realizar el guardado de los datos, además de realizar la renderización del código en HTML para visualizar el contenido en la vista.

Figura 16. Controlador registro usuario página web

```
# Create your views here.
class SignUpView(CreateView):
    model = Perfil
    form_class = SignUpForm

    def form_valid(self, form):
        '''
        En este parte, si el formulario es valido guardamos lo que se obtiene de él y usar
        '''
        form.save()

        usuario = form.cleaned_data.get('username')
        password = form.cleaned_data.get('password1')
        usuario = authenticate(username=usuario, password=password)
        login(self.request, usuario)
        return redirect('/')
```

Fuente: elaboración propia

El método *form_valid* recibe los valores del formulario y realiza una validación de ellos para comprobar que los datos ingresados sean correctos, si la validación falla, se muestran mensajes de error en pantalla, además de eso autentica al usuario y lo redirige a la página de inicio.

5.5.2. Controlador Login

Para este controlador se extiende de la clase LoginView de la librería de autenticación de Django que permite obtener el formulario de login y realizar el proceso de autenticación del usuario, y también se encarga de renderizar el template con código HTML de la vista Login.

Figura 17. Controlador login usuario página web

```
31  from django.contrib.auth.views import LoginView
32
33  class SignInView(LoginView):
34      form_class=LoginForm
35      template_name = 'usuarios/iniciar_sesion.html'
```

Fuente: elaboración propia

5.5.3. Controlador Inicio

El controlador Inicio se extiende de la clase TemplateView de la librería django.views.generic, la cual se encarga de renderizar el template con código HTML de la vista de Inicio asignando el valor de template_name.

5.5.4. Controlador Historial

El controlador Historial maneja la lógica para poder traer los registros de la base de datos para el usuario donde mostrará sus consultas realizadas y resultados, para ello es necesario importar los modelos en el controlador y luego realizar la correspondiente consulta para traer los registros filtrando de acuerdo al usuario y ordenando los datos, además de realizar el proceso de renderización del template con código HTML de la vista Historial y enviar los datos obtenidos en la consulta para su visualización.

Figura 19. Controlador historial página web

```
44  def historial(request):
45      usuariop=User.objects.get(username=request.user)
46      consulta=Hipertension.objects.filter(usuario=usuariop).order_by("-id")
47
48      return render(request, "usuarios/historial.html", {"consultas":consulta})
```

Fuente: elaboración propia

5.5.5. Controlador Hipertensión

El controlador Hipertensión cuenta con dos métodos, uno se encarga de renderizar el template con código HTML de la vista de diagnóstico hipertensión, el cual contiene el formulario con todos los datos necesarios para el diagnóstico, y el otro método se encargará de implementar la lógica para cuando se haga click en el botón diagnosticar de la vista, los datos del formulario serán enviados mediante un método POST y este método recibirá los datos del formulario, transformara el dato al tipo de dato correspondiente, luego de eso se utilizará el algoritmo realizado en machine learning, el cual se importará mediante la sentencia joblib de la librería sklearn, para realizar el proceso de diagnóstico ingresando los datos al algoritmo y con el comando predict arrojar el resultado.Luego se procede a guardar los resultados y datos en la base de datos y se renderiza un nuevo template donde se mostrarán los resultados obtenidos y los datos mediante una tabla, además de las consultas anteriormente realizadas por el usuario.

Figura 20. Controlador diagnóstico hipertensión página web

```
def buscar(request):
    if (request.POST ["edad"] and request.POST ["presions"] and request.POST ["imc"] and request.POST [ "peso"]):

        usuariop=User.objects.get(username=request.user)

        presions=request.POST ['presions']
        #genero=request.POST ['select']
        edad=request.POST ['edad']
        imc=request.POST ['imc']
        peso=request.POST ['peso']
        #usuario=User.objects.username()
        #if(genero=="1"):
        #    sexo="Masculino"
        #elif(genero=="2"):
        #    sexo="Femenino"

        modelo= load('hipertension/modelo/modelo_entrenado_hipertension.pkl')
        pca= load("hipertension/modelo/modelo_pca_hipertension.pkl")
        presionsp=int(presions)
        pesop=int(peso)
        edadp=int(edad)
        imcp=int(imc)

        #X=np.array([[-90.4469, -14.529,    1.09817,    -1.40543,   -4.05994]])
        X=np.array([[edadp, pesop, presionsp ,imcp]])
        X_pca=pca.transform(X)
        y = modelo.predict(X_pca)
        if (y == 0):
            Resultado="Normal"
        elif(y==1):
            Resultado="Prehipertenso"
        elif(y==2):
            Resultado="Hipertension en estado 1"
        elif(y==3):
            Resultado="Hipertension en estado 2"


        datos=Hipertension(usuario=usuariop,presion_s=presions, edad=edad,  imc=imc, peso=pesop, resultado=Resultado)
        datos.save()
        consulta=Hipertension.objects.filter(usuario=usuariop).order_by("-id")
```

Fuente: elaboración propia

5.5.6. Controlador Diabetes

El controlador de Diabetes cuenta con varios métodos que se encargan de renderizar los templates con código HTML de las vistas de diabetes que contienen los formularios con todos lo campos necesarios para el diagnóstico y un botón para pasar a una nueva vista hasta llegar a la vista resultados la cual contiene el código para hacer uso del algoritmo realizado en machine learning, el cual se importará mediante la sentencia joblib de la librería sklearn, para realizar el proceso de diagnóstico ingresando los datos al algoritmo y con el comando predict arrojar el resultado. Luego se procede a guardar los resultados y datos en la base de datos y se renderiza un nuevo template donde se mostrarán los resultados obtenidos y los datos mediante una tabla, además de las consultas anteriormente realizadas por el usuario.

Figura 21. Controlador diagnóstico diabetes página web

```
def resultado(request):
    if (request.POST ["glucosa-plasmatica"] and request.POST ['presion-arterial'] and request.POST ['espesor-piel'] and request.POST ['insulina'] and request.POST ["imc"]):
        glucosap = request.POST ['glucosa-plasmatica']
        presionp = request.POST ['presion-arterial']
        espesorp = request.POST ['espesor-piel']
        insulinap = request.POST ['insulina']
        imcp = request.POST ['imc']
        pedigrep = request.POST ['pedigree']
        edadp = request.POST ['edad']
        #usuariop=User.objects.get(email=request.user)
        usuariop=User.objects.get(username=request.user)
        datos=Diabetes(user=usuariop, glucosa_plasmatica=glucosap, presion_arterial=presionp,espesor_piel=espesorp, insulina=insulinap, imc=imcp, pedigri_diabetes=pedigrep, edad=edadp)
        datos.save()

        glucosap = int(glucosap)
        presionp = int(presionp)
        espesorp = int(espesorp)
        insulinap = int(insulinap)
        imcp = float(imcp)
        pedigrep = float(pedigrep)
        edadp = int(edadp)
        modelo= joblib.load("usuarios/static/ProyectoTesisApp/modelo/entreno_diabetes.pkl")
        pca= joblib.load("usuarios/static/ProyectoTesisApp/modelo/pca_diabetes.pkl")
        X=np.array([[0, glucosap, presionp, espesorp, insulinap,imcp, pedigrep, edadp ]])
        X_pca=pca.transform(X)
        y = modelo.predict(X_pca)
        if (y == 0):
            Resultado="No diabetico"
        elif(y==1):
            Resultado="Diabetico"

        consulta=Diabetes.objects.filter(user=usuariop).order_by("-id")
        recomendaciones = []
        estados = []
        s = 0
```

Fuente: elaboración propia

5.6. RUTAS PAGINA WEB

Las rutas se encargan de hacer el direccionamiento cuando se realiza una petición HTTP, y se encargan de llamar al controlador en cada ruta.

En esta sección se encuentra el direccionamiento que se relaciona con cada controlador para dar respuestas a las peticiones realizadas por el sistema.

5.6.1. Rutas usuario

Las rutas contenidas en *usuario* llaman al controlador de usuario y realizan las peticiones de la página web para registrar nuevo usuario, inicio de sesión de usuario, obtener su información, guardar un nuevo registro, el historial y todas las rutas necesarias para actualizar la contraseña del usuario.

Figura 22. Rutas usuario página web

```
urlpatterns = [
    re_path(r'^$', views.BienvenidaView.as_view(), name='bienvenida'),
    re_path(r'^registrate/$', views.SignUpView.as_view(), name='sign_up'),
    re_path(r'^incia-sesion/$', views.SignInView.as_view(), name='sign_in'),
    re_path(r'^cerrar-sesion/$', views.SignOutView.as_view(), name='sign_out'),
    re_path('historial/',views.historial, name="Historial"),
    re_path('', include(router.urls)),
    re_path('api/create_user/', UserAPI.as_view(), name = "api_create_user"),
    path("password_reset", views.password_reset_request, name="password_reset"),
    path('reset/password_reset_done', PasswordResetDoneView.as_view(template_name='usuarios/restore-pass/password_reset_done.html'), name= 'password_reset_done'),
    re_path(r'^reset/(?P<uidb64>[0-9A-za-z_\-]+)/(?P<token>.+)/$', PasswordResetConfirmView.as_view(template_name='usuarios/restore-pass/password_reset_confirm.html',
        form_class=PasswordResetConfirmForm), name= 'password_reset_confirm'),
    path('reset/password_reset/done', PasswordResetCompleteView.as_view(template_name='usuarios/restore-pass/password_reset_complete.html'), name= 'password_reset_complete')
]
```

Fuente: elaboración propia

5.6.2. Rutas Hipertensión

Las rutas contenidas en *Hipertensión* son todas las necesarias para poder realizar el diagnóstico de hipertensión, de las cuales tenemos el formulario donde se ingresan los datos y la ruta del resultado del diagnóstico.

Figura 23. Rutas diagnóstico hipertensión página web

```
urlpatterns = [
    path('modelo/', views.modelo, name='Modelo'),
    path('diagnosticar/',views.buscar, name="Diagnosticar"),
    path('api/create_hipe/', HipertensionAPI.as_view(), name = "api_create_hipe"),
]
```

Fuente: elaboración propia

5.6.3. Rutas Diabetes

Las rutas contenidas en *Diabetes* llaman al controlador de diabetes y realizan las peticiones de la página web para realizar el diagnóstico de diabetes, que incluyen las peticiones del formulario para cada campo y la ruta del resultado del diagnóstico.

Figura 24. Rutas diagnóstico diabetes página web

```
urlpatterns = [
    path('listar-diabetes/',views.ListDiabetes,name='diabetes'),
    path('presionarterial/',views.PresionArterial, name="presion"),
    path('espesorpiel/',views.Espesorpiel, name="espesor"),
    path('insulina/',views.Insulina, name="insulina"),
    path('imc/',views.Imc, name="imc"),
    path('pedigree1/',views.pedigree1, name="pedigre1"),
    path('pedigree2/',views.pedigree2, name="pedigree2"),
    path('pedigree3/',views.pedigree3, name="pedigre3"),
    path('pedigree4/',views.pedigree4, name="pedigre4"),
    path('pedigree5/',views.pedigree5, name="pedigree5"),
    path('edad/',views.edad, name="edad"),
    path('resultado/',views.resultado, name="Resultado"),
    path('api/create_diabe/', DiabetesAPI.as_view(), name = "api_create_diabe"),
]
```

Fuente: elaboración propia

4.7. CONFIGURACIÓN PARA LA CREACIÓN API

La creación APIs REST es fundamental en nuestra aplicación WEB, ya que permite que se ejecuten operaciones con nuestra aplicación móvil, por medio de los métodos HTTP(GET, POST, PUT, DELETE). Para crear nuestra APIs REST debemos descargar e instalar un framework llamado Django Rest Framework.

Figura 25. Librerías para la creación del API

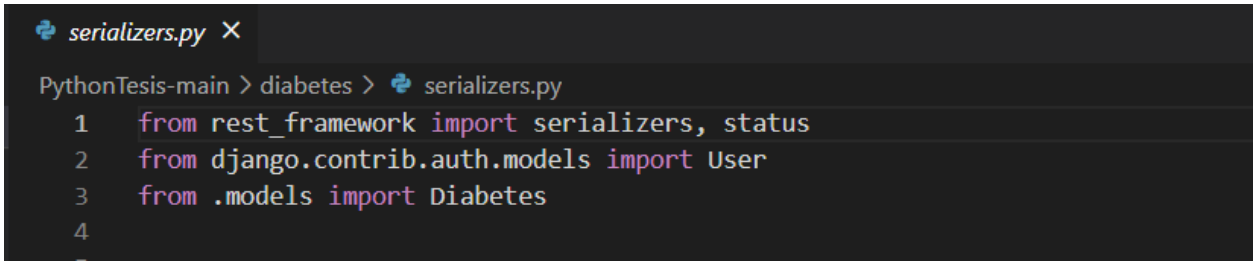
```
1  from rest_framework.response import Response
2  from .serializers import DiabetesSerializer
3  from rest_framework.views import APIView
4  from rest_framework import status, viewsets
5  from joblib import load
6  import numpy as np
```

Fuente: elaboración propia

Para poder ejecutar los datos debemos serializarlos creando un archivo llamado serializer.py en la carpeta de la aplicación que vamos a utilizar, en este caso en la

carpeta de Diabetes, es importante importar del framework de REST los serializers y los modelos de los cuales se van a serializar nuestros datos.

Figura 26. Archivo Serializers para la creación del API



Luego se crea la Clase DiabetesSerializer en la cual se colocan todos los campos que se van a serializar en este caso son los siguientes:

Figura 27. Serializer diabetes para la API

```
4
5
6   class DiabetesSerializer(serializers.Serializer):
7       id = serializers.ReadOnlyField()
8       resultado = serializers.ReadOnlyField()
9       user = serializers.CharField()
10      glucosa_plasmatica = serializers.CharField()
11      presion_arterial = serializers.CharField()
12      espesor_piel = serializers.CharField()
13      insulina = serializers.CharField()
14      imc = serializers.CharField()
15      pedigri_diabetes = serializers.CharField()
16      edad= serializers.CharField()
17
18
19      def create(self, validate_data):
20          instance = Diabetes()
21          usuariop = validate_data.get('user')
22          instance.user=User.objects.get(username=usuariop)
23          instance.glucosa_plasmatica = validate_data.get('glucosa_plasmatica')
24          instance.presion_arterial = validate_data.get('presion_arterial')
25          instance.espesor_piel = validate_data.get('espesor_piel')
26          instance.insulina = validate_data.get('insulina')
27          instance.imc = validate_data.get('imc')
28          instance.pedigri_diabetes = validate_data.get('pedigri_diabetes')
29          instance.edad = validate_data.get('edad')
30
31          instance.save()
32          return instance
33
```

Fuente: elaboración propia

Después se crean varias instancias las cuales si todas están correctas se guardan.

En el archivo llamado api.py de la carpeta diabetes se realiza el código encargado de la predicción en este caso de diabetes. Utilizando el método POST el cual permite crear recursos, luego se serializa los datos requeridos y se llaman.

Figura 28. Método para la crear una petición POST y poder diagnosticar diabetes

```
11      def post(self,request):
12          serializer = DiabetesSerializer(data = request.data)
13          if serializer.is_valid():
14              diabetes = serializer.save()
15
16              glucosap = diabetes.glucosa_plasmatica
17              presionp = diabetes.presion_arterial
18              espesorp = diabetes.espesor_piel
19              insulinap = diabetes.insulina
20              imcp = diabetes.imc
21              pedigrep = diabetes.pedigri_diabetes
22              edadp = diabetes.edad
23              glucosap = int(glucosap)
24              presionp = int(presionp)
25              espesorp = int(espesorp)
26              insulinap = int(insulinap)
27              imcp = float(imcp)
28              pedigrep = float(pedigrep)
29              edadp = int(edadp)
```

Fuente: elaboración propia

Para lograr realizar la predicción de diabetes se carga el método realizado en Machine Learning, método que se puede observar en la linea de codigo numero 30, en la linea de codigo número 31 se percibe como se carga el algoritmo utilizado para el preprocesamiento llamado Análisis de Componentes Principales, por medio de la línea 34 podemos realizar la predicción de diabetes siendo un cero no diabetes y un uno que si presenta diabetes.

Figura 29. Lógica de respuesta de la API para el método de diagnosticar diabetes

```
            modelo= load("diabetes/modelo/entreno_diabetes.pkl")
            pca= load("diabetes/modelo/pca_diabetes.pkl")
            X=np.array([[0, glucosap, presionp, espesorp, insulinap,imcp, pedigrep, edadp ]])
            X_pca=pca.transform(X)
            y = modelo.predict(X_pca)
            if (y == 0):
                Resultado="No diabetico"
            elif(y==1):
                Resultado="Diabetico"

            diabetes.resultado=Resultado


            return Response(serializer.data, status = status.HTTP_201_CREATED)

        else:
            return Response(serializer.errors, status = status.HTTP_400_BAD_REQUEST)
```

Fuente: elaboración propia

5.7. CONTENERIZACIÓN Y ALOJAMIENTO WEB MEDIANTE EKS DE AWS

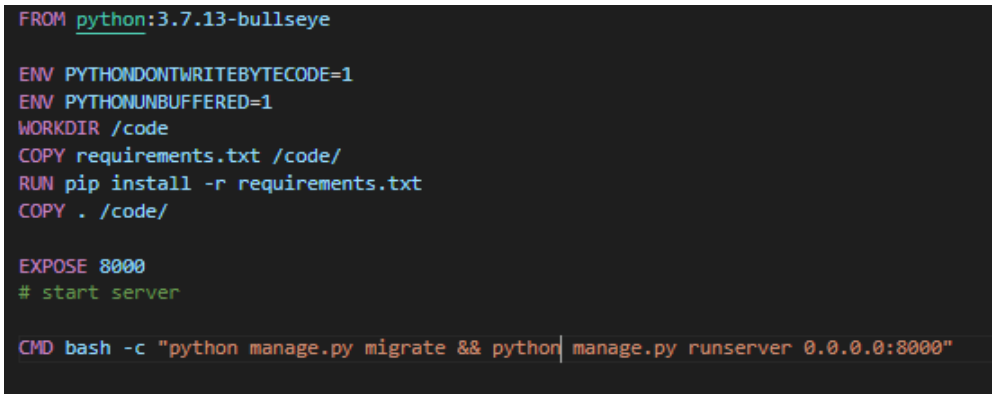
El siguiente paso a seguir una vez se haya terminado el desarrollo de la aplicación web, es realizar realizar la publicación o alojamiento web de la misma, para lo cual se hará uso una tecnología relativamente nueva que se llama docker y se encarga de realizar la contenerización o encapsulamiento de esta aplicación sin la necesidad de tener en cuenta factores externos cómo el sistema operativo y demás , sino que solo se tiene en cuenta las librerías y código de aplicación que se van a contenerizar. La aplicación se encapsula en una imagen, la cual será publicada en docker-hub, un repositorio de imágenes docker, y luego se utilizará Kubernetes para realizar la orquestación del contenedor creado para la aplicación, se hará uso del servicio de Amazon Web Services (AWS) que se llama Amazon Elastic Kubernetes Service (EKS) para poder desplegar los kubernetes en la infraestructura de AWS y así poder acceder a través de una url a la aplicación mediante internet.

5.7.1. Contenerización de aplicación web

Para realizar la contenerización de la aplicación se debe tener instalado el software de Docker en el computador como requisito previo, se debe crear un archivo Dockerfile en el cual se debe escribir los comandos necesarios empezando por importar la imagen

base sobre la cual se va construir nuestra aplicación que en nuestro caso será python. Luego se setean algunas variables de entorno necesarias para que la imagen funcione correctamente, se define el directorio donde almacenarán los archivos en el contenedor y también se ejecuta el comando run para instalar todas la librerías que son necesarias para la aplicación. Por último se expone el puerto por el cual se podrá acceder a la aplicación y se ejecuta el comando para ejecutar el servidor de Django.

Figura 30. Código archivo Dockerfile para la contenerización de la aplicación



```
FROM python:3.7.13-bullseye

ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1
WORKDIR /code
COPY requirements.txt /code/
RUN pip install -r requirements.txt
COPY . /code/

EXPOSE 8000
# start server

CMD bash -c "python manage.py migrate && python manage.py runserver 0.0.0.0:8000"
```

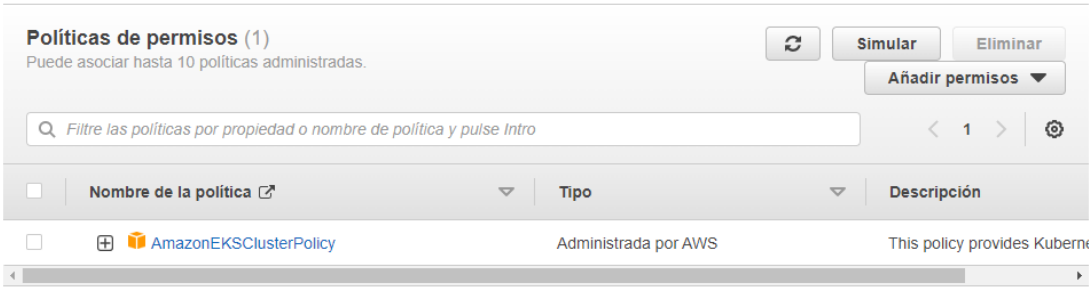
Fuente: elaboración propia

Para crear la imagen de docker, se debe ejecutar este archivo Dockerfile mediante el comando *Docker build . -t nombre:versión*, luego de crear la imagen se debe subir al repositorio en Docker-hub, para lo cual se debe tener primeramente una cuenta en Docker-hub y estar logueado, luego con el comando *docker push nombre:versión* se sube la imagen al repositorio.

5.7.2. Alojamiento web mediante EKS de AWS

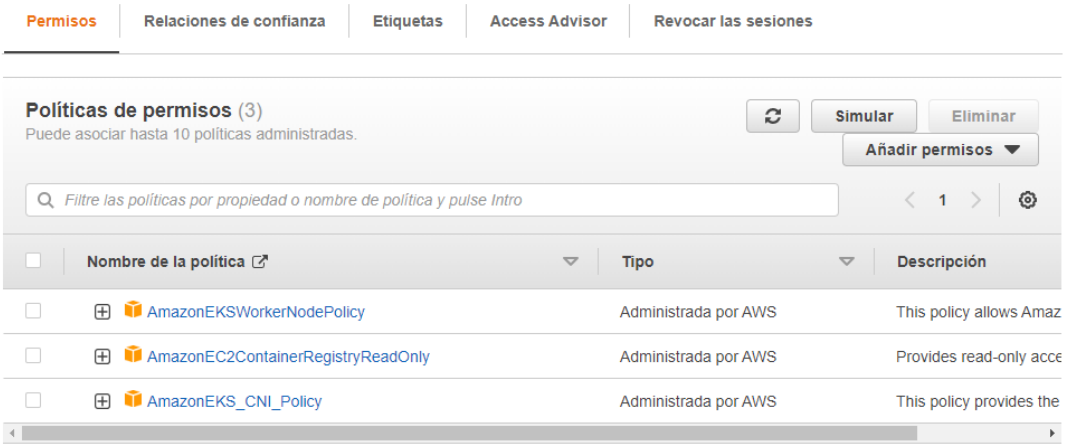
Luego de haber creado la imagen docker de la aplicación se procede a realizar el despliegue mediante Kubernetes en EKS de AWS. Primero, se tiene que tener una cuenta en AWS, luego se deben crear los roles de IAM que vamos a utilizar para realizar el manejo del cluster que vamos a crear en EKS. Se crean dos roles, uno que se encargará de realizar el manejo del cluster y otro que se encargará del manejo de los worker-nodes que se usarán en el despliegue, se le deben asignar los siguientes permisos en los roles.

Figura 31. Políticas o permisos para el Rol que maneja el cluster



Fuente: elaboración propia

Figura 32. Políticas o permisos para el Rol que maneja los worker-nodes



Fuente: elaboración propia

Luego de crear los roles se procede a crear el cluster y los nodos de trabajo necesarios para poder desplegar la aplicación. Se debe ejecutar el siguiente comando *aws eks update-kubeconfig --region region-code --name my-cluste*r en el cmd para poder emparejar de manera local con el cluster creado.

Figura 33. Imagen Cluster EKS creado

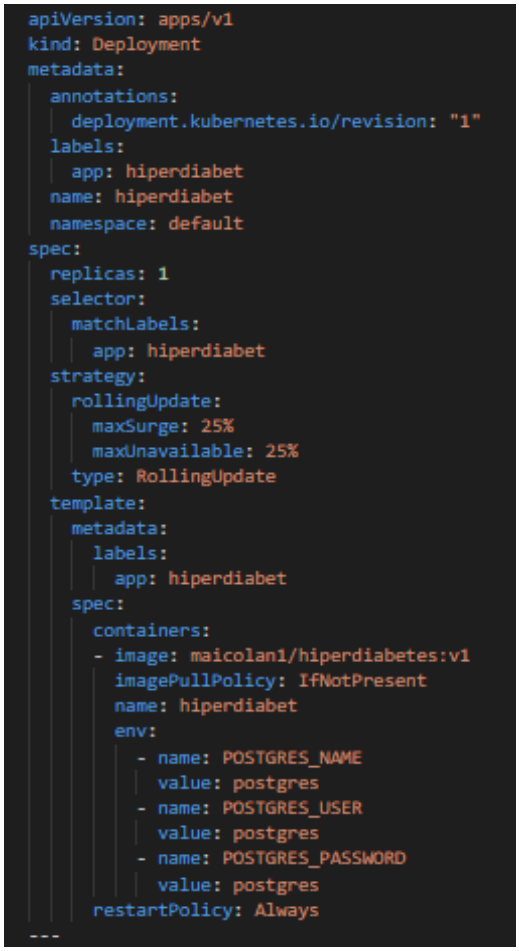
Fuente: elaboración propia

Una vez emparejado el cluster, se deben crear los archivos YAML que se ejecutarán para poder realizar el despliegue de la aplicación en kubernetes. Es importante mencionar que es necesario crear un despliegue de kubernetes también para la base de datos de Postgresql. En los archivos YAML es necesario colocar el nombre de la imagen que se utilizará en el despliegue, el puerto que se expondrá, las variables de entorno y el tipo de servicio que puede ser NodePort, LoadBalancer, entre otros.

Para el despliegue se tiene que crear un deployment y un service, ambos son necesarios para el correcto funcionamiento de la aplicación desplegada, el deployment se encarga de crear el pod o contenedor de la aplicación y la reiniciará mediante un componente que se llama replicaset en casó de que llegue a fallar. El service es el encargado de exponer la aplicación a través de un puerto.

Para ejecutar el archivo YAML y realizar el despliegue se debe ejecutar el comando *kubectl apply -f nombrearchivo.yaml.*

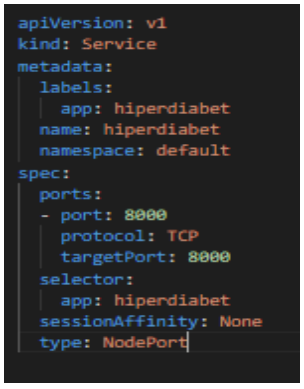
Figura 34. Código YAML para el despliegue del deployment en kubernetes



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  labels:
    app: hiperdiabet
  name: hiperdiabet
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hiperdiabet
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: hiperdiabet
    spec:
      containers:
      - image: maicolan1/hiperdiabetes:v1
        imagePullPolicy: IfNotPresent
        name: hiperdiabet
        env:
          - name: POSTGRES_NAME
            value: postgres
          - name: POSTGRES_USER
            value: postgres
          - name: POSTGRES_PASSWORD
            value: postgres
      restartPolicy: Always
---
```

Fuente: elaboración propia

Figura 35. Código YAML para el despliegue del servicio en kubernetes

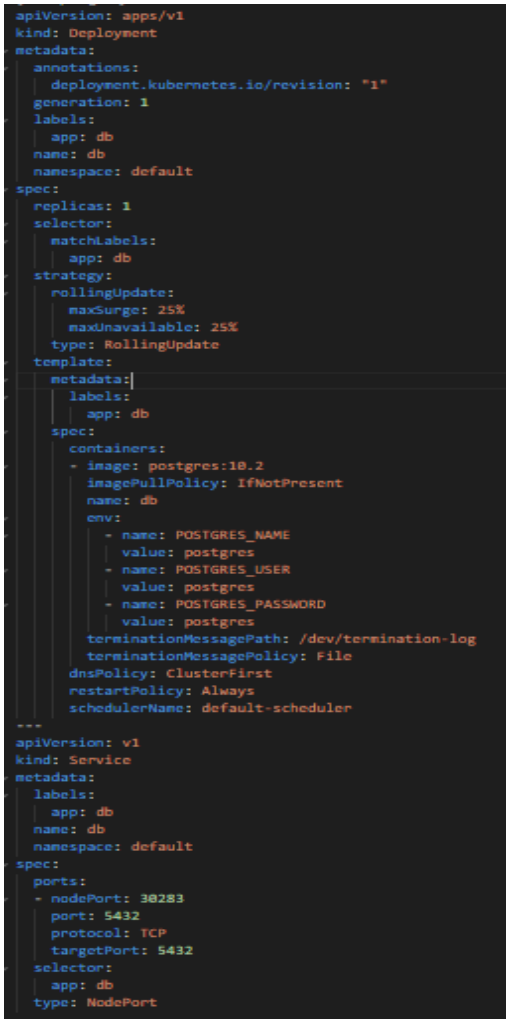


```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: hiperdiabet
  name: hiperdiabet
  namespace: default
spec:
  ports:
  - port: 8000
    protocol: TCP
    targetPort: 8000
  selector:
    app: hiperdiabet
  sessionAffinity: None
  type: NodePort
```

Fuente: elaboración propia

El archivo YAML para el despliegue de la base de datos postgres es el siguiente:

Figura 36. Código YAML para el despliegue de base de datos en kubernetes



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  generation: 1
  labels:
    app: db
  name: db
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: db
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: db
    spec:
      containers:
      - image: postgres:10.2
        imagePullPolicy: IfNotPresent
        name: db
        env:
          - name: POSTGRES_NAME
            value: postgres
          - name: POSTGRES_USER
            value: postgres
          - name: POSTGRES_PASSWORD
            value: postgres
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: db
  name: db
  namespace: default
spec:
  ports:
  - nodePort: 30283
    port: 5432
    protocol: TCP
    targetPort: 5432
  selector:
    app: db
  type: NodePort
```

Fuente: elaboración propia

6. DESARROLLO APLICACIÓN MÓVIL

En este capítulo se explicará la creación de la aplicación móvil, la cual se realizó en el entorno de desarrollo Android Studio (véase anexo F) ya que es la herramienta oficial de Google para desarrollar aplicaciones para su sistema operativo.

6.1. CONFIGURACIÓN

6.1.1. Build.grade

Se importan las librerías del proyecto, tales como *retrofit* para realizar el consumo de la api mediante http, *gson* para convertir *json* a objetos y viceversa y demás librerías predeterminadas de java para android studio. Además de recalcar que la aplicación es desarrollada en la versión 26 de SDK.

6.1.2. AndroidManifest.xml

Se realiza la configuración principal del proyecto, como la declaración de las Activities (login, registro, y la actividad de diagnóstico que consta de las dos operaciones hipertensión y diabetes), se configuran los permisos de acceso a internet y además se ajusta el nombre y logo de la aplicación.

6.2. VISTAS

6.2.1. Vista login

Corresponde al activity principal para el logueo del taxista. Esta vista se muestra en pantalla completa, contiene el logo de la aplicación, los campos usuario y contraseña, un botón para iniciar sesión y un botón para ir hacía la vista de registro.

Figura 37. Vista login app

Fuente: elaboración propia

Se debe llenar los dos campos que aparecen en la vista y luego presionar el botón iniciar sesión, el cual llamará a un evento que primeramente realizará la validación de los campos y luego intentará realizar el proceso de logueo del usuario llamando a la api realizada en django con la ayuda de la librería de retrofit. Si los campos se envían vacíos se muestra un mensaje informando indicando que son requeridos.

Figura 38. Validación de ingreso app

Fuente: elaboración propia

El proceso autenticación se realiza mediante la comunicación con la api, está validará si las credenciales son correctas o incorrectas, en caso de ser correctas enviará los datos del usuario logueado y un token de autenticación que se guardaran en la aplicación para cuando el usuario entre después de cerrar la aplicación la sesión permanezca activa y se procederá a lanzar la actividad principal del diagnostico que tiene una primera vista de home. En caso de que las credenciales sean incorrectas la API enviará un mensaje de error que se mostrará en forma de pop-up en la vista de login.

6.2.2. Vista Home

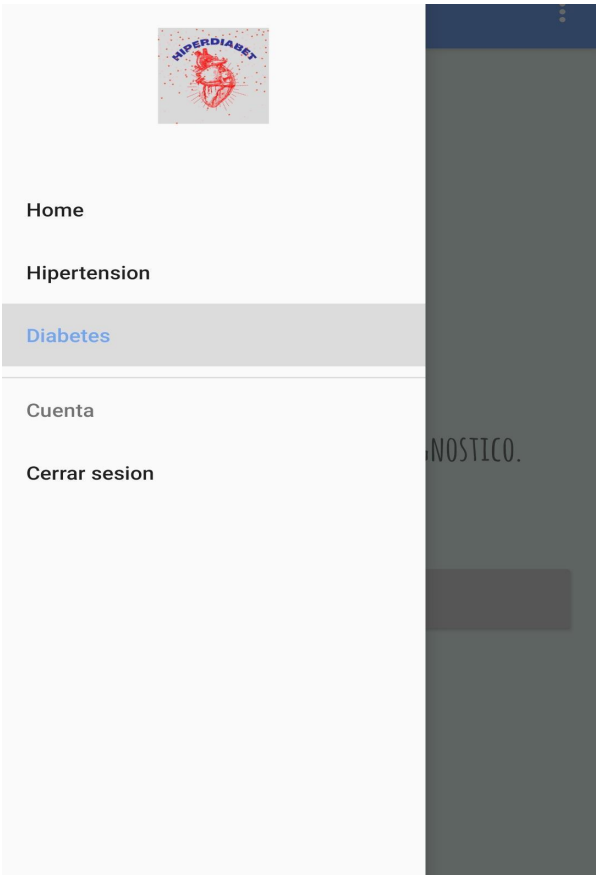
Esta vista hace parte del activity diagnóstico que muestra una pantalla de inicio junto con un menú desplegable que contiene las demás vistas. Esta vista muestra un mensaje que invita a la persona a hacer uso de la aplicación para poder diagnosticarse.

Figura 39. Vista servicio app

Fuente: elaboración propia

En el lado superior izquierdo de la vista se encuentra el botón del menú desplegable que contiene las opciones de hipertensión, diabetes para proceder al diagnóstico, y la opción de cerrar sesión, que finaliza la sesión del usuario y nos retorna a la vista del login.

Figura 40. Menú desplegable

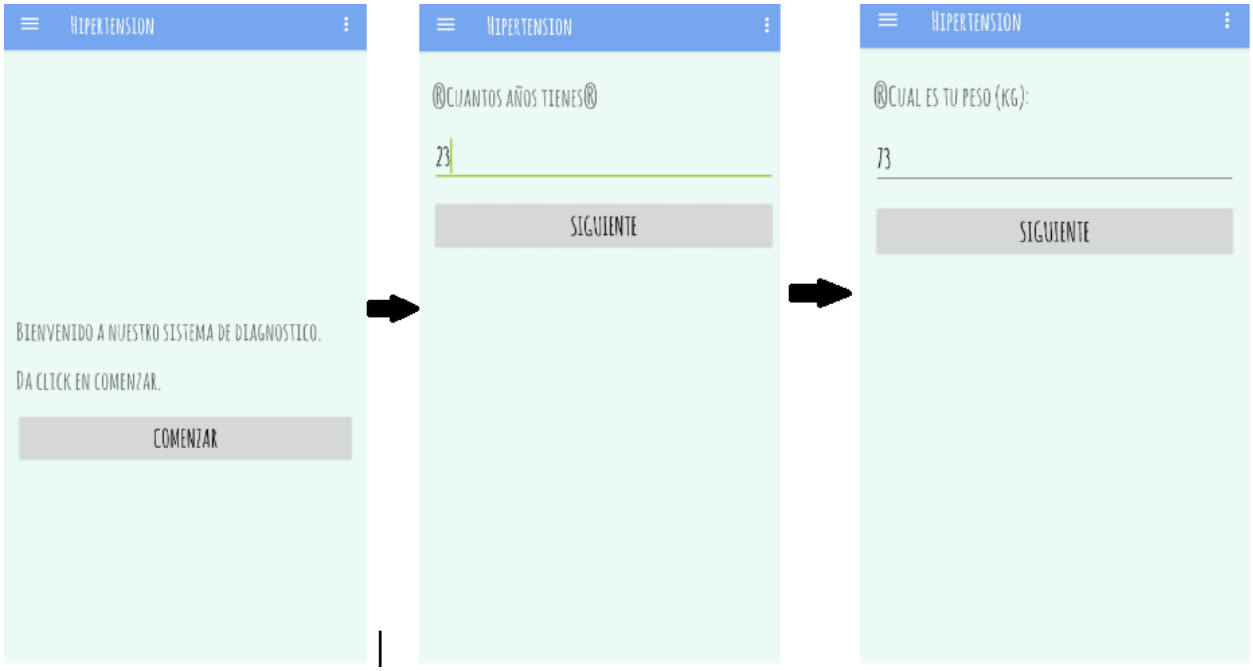


Fuente: elaboración propia

6.2.3. Vista Hipertensión

Esta vista se muestra cuando se presiona la opción hipertensión en el menú, contiene un botón que llamará una secuencia de vistas, cada una con un formulario en el cual el usuario deberá ingresar un dato pedido en un input para poder continuar, en caso de enviar el campo vacío se desplegará un error que indicará que el campo es necesario para poder continuar.

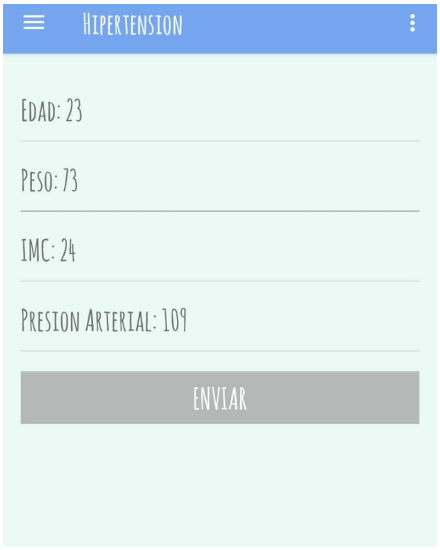
Figura 41. Ejemplo vistas diagnóstico hipertensión App Móvil



Fuente: elaboración propia

Una vez se hayan llenado todos los formularios en el flujo, se presenta una vista que recopila todos los datos ingresados para que el usuario pueda validar si son correctos y preparar estos datos para enviarlos a la API con ayuda de la librería de retrofit. La vista contiene un botón que llama al evento que realizará la comunicación con el API, que en caso de responder exitosamente recibirá el resultado del diagnóstico y llamará a una nueva vista que contendrá este resultado. En caso de que se presente un error en la conexión con el API se retornará el error y se informará al usuario mediante un pop-up.

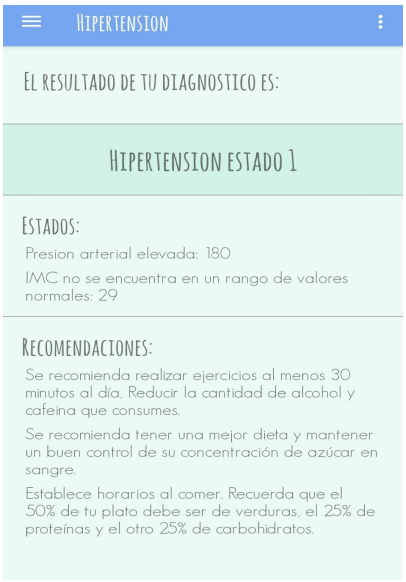
Figura 42. Vista recopilación datos hipertensión



Fuente: elaboración propia

En la respuesta del diagnóstico se muestra el resultado, el estado de los valores presentes en el usuario cómo por ejemplo si el paciente presenta una presión arterial elevada o valor de imc irregular, además de algunas recomendaciones que pueden ser útiles para el usuario con la finalidad de ayudar a mejorar la condición de salud en la que se encuentra.

Figura 43. Vista resultado diagnóstico hipertensión

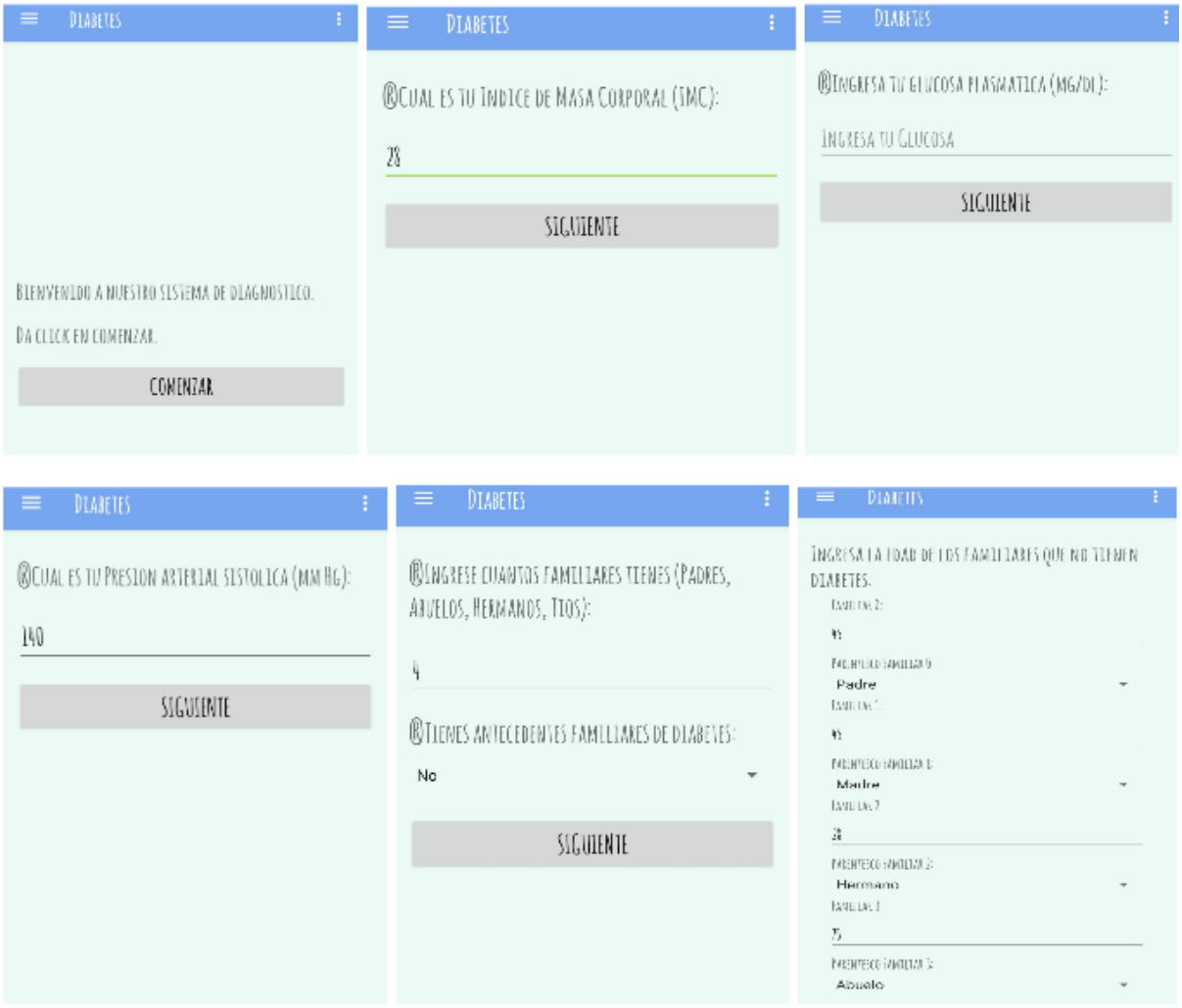


Fuente: elaboración propia

6.2.4. Vista Diabetes

Esta vista hace parte del activity del diagnóstico y se muestra cuando se presiona la opción diabetes en el menú, contiene un botón que llamará una secuencia de vistas, cada una con un formulario en el cual el usuario deberá ingresar un dato pedido en un input para poder continuar y que son necesarios para realizar el diagnóstico de diabetes, en caso de enviar el campo vacío se desplegará un error que indicará que el campo es necesario para poder continuar.

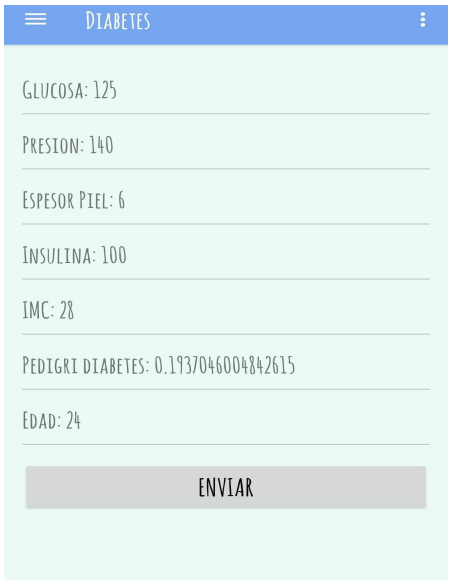
Figura 44. Ejemplo vistas diagnóstico diabetes App Móvil



Fuente: elaboración propia

Una vez se hayan llenado todos los formularios en el flujo, se presenta una vista que recopila todos los datos ingresados para que el usuario pueda validar si son correctos y preparar estos datos para enviarlos a la API con ayuda de la librería de retrofit. La vista contiene un botón que llama al evento que realizará la comunicación con el API, que en caso de responder exitosamente recibirá el resultado del diagnóstico y llamará a una nueva vista que contendrá este resultado. En caso de que se presente un error en la conexión con el API se retornará el error y se informará al usuario mediante un pop-up.

Figura 45. Vista recopilación datos diabetes



Fuente: elaboración propia

En la respuesta del diagnóstico se muestra el resultado, el estado de los valores presentes en el usuario cómo por ejemplo si el paciente presenta un valor elevado de glucosa plasmática, presión arterial elevada o un valor de insulina irregular, además de algunas recomendaciones que pueden ser útiles para el usuario con la finalidad de ayudar a mejorar la condición de salud en la que se encuentra.

Figura 46. Vista resultado diagnóstico diabetes

Fuente: elaboración propia

6.2.5. Vista Registro

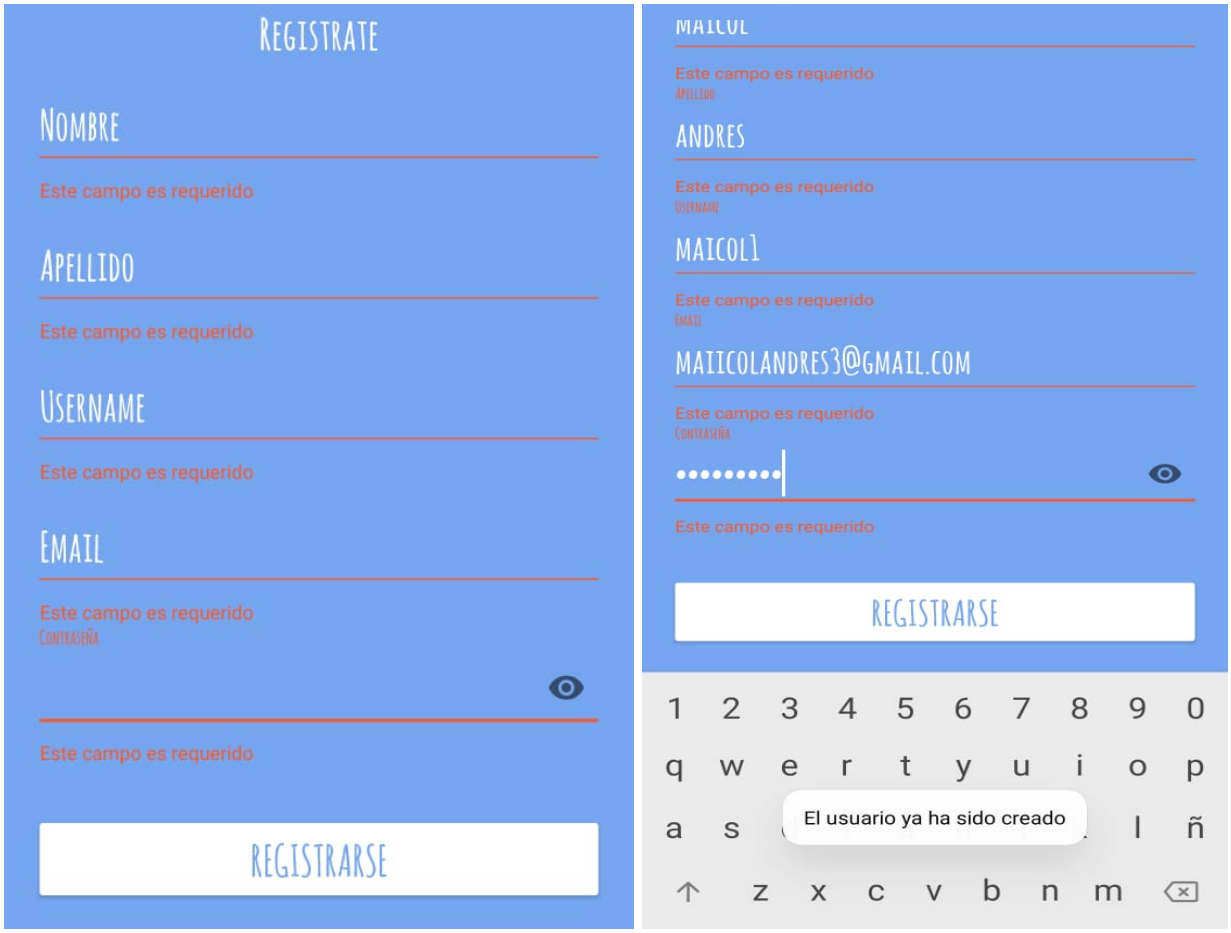
Corresponde al activity para realizar el registro de un nuevo usuario en la aplicación. Esta vista se muestra en pantalla completa, contiene los campos usuario, nombre, apellidos, email y contraseña y un botón para enviar los datos y registrarse.

Figura 47. Vista registro de la aplicación móvil

Fuente: elaboración propia

Cuando se presione el botón de registrarse, se llamará un evento el cual primero validará que los campos enviados no se encuentren vacíos, de lo contrario enviará un mensaje de error indicando que los campos son requeridos, luego de eso, enviará los campos a la API creada en django con la ayuda de la librería de retrofit, donde se validará que no haya un usuario con el mismo nombre ya creado en la base de datos. Si el usuario ya ha sido creado con anterioridad, el API responde un mensaje de error indicando que ese usuario ya existe en la base de datos y que por favor ingrese uno diferente. Si el usuario no existe, se guardan los campos en la base de datos y se envía un mensaje al usuario indicando que la cuenta fue creada con éxito.

Figura 48. Validación de campos del registro



Fuente: elaboración propia

6.3. COMUNICACIÓN POR RETROFIT PARA CONSUMIR API

Para la comunicación con la API se utilizó la librería retrofit que funciona como cliente HTTP para Android y Java, para hacer uso de la librería es necesario importarla en el gradle y luego crear un cliente de retrofit que recibe cómo parámetro de entrada la url base de la API.

Figura 49. Código creación de cliente retrofit

```
public class RetrofitClient {
    private static Retrofit retrofit = null;

    public static Retrofit getClient(String baseUrl) {
        if (retrofit==null) {
            retrofit = new Retrofit.Builder()
                    .baseUrl(baseUrl)
                    .addConverterFactory(GsonConverterFactory.create())
                    .build();
        }
        return retrofit;
    }
```

Fuente: elaboración propia

Además de eso se debe crear una interfaz, donde se debe estipular los métodos HTTP para poder realizar el consumo de cada uno de los endpoints de la API, y también se deben crear funciones con los campos que van en cada petición hacía la API.

Figura 50. Código interfaz para la comunicación con los endpoints del API

```
    public static final String BASE_URL = "http://192.168.1.6:8000/";

    @POST("http://192.168.1.6:8000/users/login/")
    Call<Affiliate> login(@Body LoginBody loginBody);

    @POST("http://192.168.1.6:8000/api/create_user/")
    @FormUrlEncoded
    Call<Post> savePost(@Field("first_name") String firstname,
                        @Field("last_name") String lastname,
                        @Field("username") String username,
                        @Field("email") String email,
                        @Field("password") String password);

    @POST("http://192.168.1.6:8000/api/create_hipe/")
    @FormUrlEncoded
    Call<Hipertension> savehipertension(@Field("usuario") String usuario,
                                        @Field("presion_s") String presion_s,
                                        @Field("edad") String edad,
                                        @Field("peso") String peso,
                                        @Field("imc") String imc);

    @POST("http://192.168.1.6:8000/api/create_diabe/")
    @FormUrlEncoded
    Call<Diabetes> savediabetes(@Field("user") String user,
                                @Field("glucosa_plasmatica") String glucosa_plasmatica,
                                @Field("presion_arterial") String presion_arterial,
                                @Field("espesor_piel") String espesor_piel,
                                @Field("insulina") String insulina,
                                @Field("imc") String imc,
                                @Field("pedigri_diabetes") String pedigri_diabetes,
                                @Field("edad") String edad);

}
```

Fuente: elaboración propia

7. CONCLUSIONES

Con este proyecto se logró diseñar e implementar un sistema experto que permite predecir si una persona presenta diabetes y/o hipertensión a través de un algoritmo de machine learning con un porcentaje de predicción de 92,4% para diabetes y 98,9% para hipertensión, y a través de una interfaz gráfica fácil de utilizar e intuitiva, la cual fue hecha en django y en android studio, permite a sus usuarios registrarse, llevar un historial de sus diagnósticos y varios formularios que permiten ingresar datos y recibir resultados fácil y rápidamente. Esta aplicación puede tener diferentes usos, ya sea para llevar un control de tus datos y diagnósticos teniendo en cuenta que existe un porcentaje de error en la aplicación, cómo también ser utilizada con ayuda de un experto de la salud y que funcione como un apoyo para poder llegar a un resultado confiable y rápido.

La creación de la API rest es una implementación muy importante para cualquier sistema ya que se pueden hacer peticiones HTTP desde el servidor hacia el cliente o viceversa sin importar el lenguaje de programación utilizado. En este caso, por ejemplo, se realizan peticiones desde JAVA (Android) y Python (Django) utilizando una sola estructura de control para realizar el intercambio de datos mediante la librería Django Rest Framework y la librería de android retrofit para realizar el consumo fácilmente.

De acuerdo a los resultados obtenidos relacionados con el algoritmo de predicción, el método que mejor se ajustó a los datos de entrada utilizados en la creación del algoritmo fue Máquinas de Vectores de Soporte a comparación de los demás métodos utilizados. El porcentaje de predicción obtenido para los algoritmos de hipertensión y diabetes fue aceptable, aunque se debe tener en cuenta que el algoritmo no es 100% confiable, sino que se tiene un porcentaje de error y se debe tener en cuenta a la hora de ejecutar el diagnóstico.

La aplicación móvil permitió realizar las operaciones de diagnóstico, inicio de sesión y registro desde la comodidad de un dispositivo móvil al que fácilmente cualquier persona tiene acceso hoy en día, con una interfaz gráfica sencilla e intuitiva que permite registrar información que será intercambiada con la API hecha en django mediante el protocolo de comunicación http y la librería retrofit. La aplicación hace un buen uso API y de estas herramientas de integración que nos permiten la comunicación entre servicios con el fin de brindar un producto útil para sus usuarios.

La página web permite al usuario acceder a la información del usuario de una forma sencilla y visualmente agradable. Se puede consultar el historial del usuario donde se muestran los datos y el resultado del diagnóstico, así cómo también realizar el proceso

de diagnóstico de hipertensión y diabetes. La página web fue hecha en el Framework Django ya que éste se encuentra basado en el Lenguaje de programación python, lo cual nos facilitó el tema de añadir el algoritmo hecho con machine learning con la librería de sk-learn de python.

8. RECOMENDACIONES

Aunque el proyecto es ambicioso, se pueden realizar varias mejoras que permiten optimizar tanto la interfaz gráfica cómo el algoritmo de predicción para obtener un producto mucho más estructurado, tales como:

Se realizaron las pruebas de construcción del algoritmo en diferentes métodos de machine learning que se consideran clásicos, sin embargo, existen varios tipos de métodos de machine learning que deberían ser considerados para obtener un mejor ajuste de los datos y porcentaje de predicción como lo son las redes neuronales y los Modelos Ensemble que presentan mejoras en la respuesta de predicción.

Para la aplicación móvil se pueden realizar varias mejoras en tema de diseño y estética de la aplicación, cómo por ejemplo hacer uso de tecnologías relativamente nuevas cómo el SDK Flutter que permite desarrollar interfaces de una forma sencilla y con una mayor estética.

Otro tema a mejorar para un futuro es mejorar la estructura responsive de la aplicación web que aún no se encuentra ajustada para que sea atractiva para los usuarios que acceden desde un dispositivo móvil a la aplicación web.

Los temas relacionados a seguridad de la aplicación no fueron estudiados a fondo para este proyecto y pueden ser mejorados en el futuro, se realizó un sistema de inicio de sesión y todos los datos son enviados mediante métodos post que utilizan un token CSRF que brinda seguridad a la hora de enviar datos en el formulario HTML, pero al ser datos de tipo sensible y de carácter médico en necesario tener un nivel alto de seguridad de los datos que se ajusten a los parámetros y leyes internacionales de protección de datos. Se propone como mejora implementar un sistema de doble factor de autenticación para poder acceder a la aplicación.

Además de la seguridad de las aplicaciones, también es necesario mejorar la seguridad de la API creada, para que no cualquiera pueda realizar una conexión con nuestra API, sino que también exista un factor de autorización para realizar el envío de peticiones

9. REFERENCIAS

[1] Las 10 principales causas de defunción. (s. f.). Recuperado 26 de abril de 2020, de https://www.who.int/es/news-room/fact-sheets/detail/the-top-10-causes-of-death

[2] Morales, J., (2017, mayo 19). OPS/OMS Colombia - Día Mundial de la Hipertensión 2017: Conoce tus números | OPS/OMS. Pan American Health Organization / World Health Organization.

https://www.paho.org/col/index.php?option=com_content&view=article&id=2752:dia-mundial-de-la-hipertension-2017-conoce-tus-numeros&Itemid=487

[3] Diabetes. (s. f.). Recuperado 26 de abril de 2020, de https://www.who.int/es/news-room/fact-sheets/detail/diabetes

[4] Cayon, A. (2017, mayo 11). OPS/OMS | Día Mundial de la Hipertensión 2017: Conoce tus números. Pan American Health Organization / World Health Organization.

https://www.paho.org/hq/index.php?option=com_content&view=article&id=13257:dia-mundial-de-la-hipertension-2017-conoce-tus-numeros&Itemid=42345&lang=es

[5] Ministerio de salud. (17 de Mayo de 2017). Minsalud. Dia mundial de la hipertensión arterial. Obtenido de Minsalud: https://www.minsalud.gov.co/Paginas/Colombia-enfrenta-epidemia-de-enfermedades-cardiovasculares-y-diabetes.aspx

[6] OMS | Informe mundial sobre la diabetes. (s/f). WHO; World Health Organization. Recuperado el 23 de abril de 2020, de http://www.who.int/diabetes/global-report/es/

[7] Sociedad Colombiana de Cardiología y Cirugía Cardiovascular (SCC), Capítulo de Hipertensión arterial, ¿cuáles son las cifras normales?. Recuperado el 23 de abril de 2020, de http://scc.org.co/wp-content/uploads/2018/04/Hipertensi%C3%B3n-arterial-cu%C3%A1les-son-las-cifras-normales-Dr-Luis-Moya.pdf

[8] OMS | Preguntas y respuestas sobre la hipertensión. (s. f.). WHO. Recuperado 21 de febrero de 2020, de http://www.who.int/features/qa/82/es/

[9] Diabetes: Tratamiento, síntomas, causas y prevención. (2009, febrero 18). CuidatePlus. https://cuidateplus.marca.com/enfermedades/digestivas/diabetes.html

[10] Los sistemas expertos—Inteligencia Artificial. (s. f.). Recuperado 26 de abril de 2020, de https://sites.google.com/site/proyectointeligenciaartificial/indice/los-sistemas-expertos

[11] Diego, E. A. (12 de Junio de 2019). Monografias. Obtenido de Monografias : https://www.monografias.com/trabajos101/sistema-operativo-android/sistema-operativo-android.shtml

[12] MVC (Model, View, Controller) explicado. (s. f.). Recuperado 19 de septiembre de 2019, de CódigoFacilito website: https://codigofacilito.com/articulos/mvc-model-view-controller-explicado

[13] El patrón de diseño MTV (El libro de Django 1.0). (s/f). Recuperado el 26 de abril de 2020, de https://uniwebsidad.com/libros/django-1-0/capitulo-5/el-patron-de-diseno-mtv

[14] Qué es el modelo vista controlador (MVC) y como funciona. (s. f.). Recuperado 19 de septiembre de 2019, de https://articulosvirtuales.com/articles/educacion/que-es-el-modelo-vista-controlador-mvc-y-como-funciona

[15] “Machine learning”: ¿qué es y cómo funciona? (s/f). Recuperado el 21 de febrero de 2020, de https://www.bbva.com/es/machine-learning-que-es-y-como-funciona/

[16] Aprendizaje automático: Qué es y por qué es importante | SAS. (s/f). Recuperado el 21 de febrero de 2020, de https://www.sas.com/es_co/insights/analytics/machine-learning.html#machine-learning-importance

[17] May, O. A. C., Koo, J. J. P., Kinani, J. M. V., & Encalada, M. A. Z. (2018). CONSTRUCCIÓN DE UN MODELO DE PREDICCIÓN PARA APOYO AL DIAGNÓSTICO DE DIABETES (CONSTRUCTION OF A PREDICTION MODEL TO SUPPORT THE DIABETES DIAGNOSIS). Pistas Educativas, 40(130). http://www.itc.mx/ojs/index.php/pistas/article/view/1805

[18] LaFreniere, D., Zulkernine, F., Barber, D., & Martin, K. (2016). Using machine learning to predict hypertension from a clinical dataset. 2016 IEEE Symposium Series on Computational Intelligence (SSCI), 1–7. https://doi.org/10.1109/SSCI.2016.7849886

[19] Rawat, V., & Suryakant. (2019). A Classification System for Diabetic Patients with Machine Learning Techniques. https://doi.org/10.33889/ijmems.2019.4.3-057

[20] Chatrati, S. P., Hossain, G., Goyal, A., Bhan, A., Bhattacharya, S., Gaurav, D., & Tiwari, S. M. (2020). Smart home health monitoring system for predicting type 2 diabetes and hypertension. Journal of King Saud University - Computer and Information Sciences. https://doi.org/10.1016/j.jksuci.2020.01.010

[21] Smith, J. W., Everhart, J. E., Dickson, W. C., Knowler, W. C., & Johannes, R. S. (1988). Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus. Proceedings of the Annual Symposium on Computer Application in Medical Care, 261-265. Tomado de https://www.personal.kent.edu/~mshanker/personal/Zip_files/sar_2000.pdf

[22] ANACONDA NAVIGATOR. (s.f.). Anaconda Navigator (Versión 1.6.9) [Entorno de desarrollo Python]. [Consultado: 12 de Febrero de 2019]. https://www.anaconda.com/products/individual

[23] Algoritmo k-Nearest Neighbor | Aprende Machine Learning. (s. f.). Recuperado 30 de junio de 2022, de https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/

[24] Árbol de decisión en Machine Learning (Parte 1)—Sitiobigdata.com. (s. f.). Recuperado 30 de junio de 2022, de https://sitiobigdata.com/2019/12/14/arbol-de-decision-en-machine-learning-parte-1/#

[25] Máquinas de Vector Soporte (Support Vector Machines, SVMs). (s. f.). Recuperado 30 de enero de 2022, de https://www.cienciadedatos.net/documentos/34_maquinas_de_vector_soporte_support_vector_machines

[26] Brownlee, J. (2016, abril 10). Naive Bayes for Machine Learning. Machine Learning Mastery. https://machinelearningmastery.com/naive-bayes-for-machine-learning/

[27] Random Forest (Bosque Aleatorio): Combinando árboles - IArtificial.net. (2019, junio 10). https://www.iartificial.net/random-forest-bosque-aleatorio/

[28] Calidad de datos en minería de datos a través del preprocesamiento. (s. f.). Recuperado 30 de junio de 2022, de https://blog.powerdata.es/el-valor-de-la-gestion-de-datos/calidad-de-datos-en-mineria-de-datos-a-traves-del-preprocesamiento

[29] Análisis de Componentes Principales (ACP). (s. f.). XLSTAT, Your data analysis solution. Recuperado 30 de junio de 2022, de

https://www.xlstat.com/es/soluciones/funciones/analisis-de-componentes-principales-acp

[30] ¿Qué es PostgreSQL? - Para qué sirve, Características e Instalación. (s. f.). Recuperado 30 de junio de 2022, de https://blog.infranetworking.com/servidor-postgresql/

[31] Sistema Operativo Android. (s. f.). Recuperado 30 de junio de 2022, de https://www.monografias.com/trabajos101/sistema-operativo-android/sistema-operativo-android

[32] ¿Qué es y como crear un API REST en Django? (s. f.). Recuperado 30 de junio de 2022, de https://platzi.com/clases/26-backend-online/1062-que-es-y-como-crear-un-api-rest-en-django/?utm_source=google&utm_medium=cpc&utm_campaign=17418244234&utm_adgroup=&utm_content=&gclid=Cj0KCQjwntCVBhDdARIsAMEwACnn213VsqV88Z_vYPCr4B27QOZejPrOXWu1HjEDTBpRy9v8sx-AFt8aAqYPEALw_wcB&gclsrc=aw.ds

[33] ¿Qué es Kubernetes? (s. f.). Kubernetes. Recuperado 30 de junio de 2022, de https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/

[34] ¿Qué es Docker? (s. f.). Recuperado 30 de junio de 2022, de https://www.redhat.com/es/topics/containers/what-is-docker

[35] Servicio de Kubernetes administrado – Amazon EKS – Amazon Web Services. (s. f.). Amazon Web Services, Inc. Recuperado 30 de junio de 2022, de https://aws.amazon.com/es/eks/

[36] Pima Indians Diabetes Database. (s. f.). Recuperado 30 de junio de 2022, de https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

10. ANEXOS

Debido a la cantidad de código que contiene el proyecto Python, Django y Android Studio, se recomienda dirigirse al siguiente enlace de Google Drive y descargar la totalidad del código fuente: https://drive.google.com/drive/folders/1IEmS3-FWOdu4C-nDwC8_khKZWRX7Mynb?usp=sharing. A continuación, se detalla una lista de anexos que contiene el código fuente básico y a grandes rasgos del proyecto.

Anexo A. Código Algoritmo en Machine Learning en Python

Figura 51. Código Algoritmo en Machine Learning en Python parte 1

```
#Se importan la librerias a utilizar
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import decomposition
from sklearn.metrics import accuracy_score, roc_auc_score

#datos=pd.read_csv("LEARNING_procesado5.csv", header=0 )
datos=pd.read_csv("hipertension_dataset_procesado.csv", header=0 )
datos.dtypes

########## PREPARAR LA DATA ##########
X = np.array(datos[['Age(year)','Weight(kg)', 'Systolic Blood Pressure(mmHg)',
                    'BMI(kg/m^2)']])
########## ENTENDIMIENTO DE LA DATA ##########
#Verifico la información contenida en el dataset
print('Información en el dataset:')
print(datos.keys())
#Defino los datos correspondientes a las etiquetas
y = np.array(datos[['Hypertension']])

########## IMPLEMENTACIÓN DE MAQUINAS VECTORES DE SOPORTE ##########

# Using PCA from sklearn PCA
pca = decomposition.PCA(n_components=4)
pca.fit(X)
#from sklearn.externals import joblib
#joblib.dump(pca4, 'modelo_pca.pkl')
#print(pca4)
#np.savetxt('matrizpca.txt',pca,delimiter = ",")
X_pca = pca.transform(X)

from sklearn.model_selection import train_test_split

resultado=[0.0]
```

Figura 52. Código Algoritmo en Machine Learning en Python parte 2

```
#Separo los datos de "train" en entrenamiento y prueba para probar los algoritmos
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.3,stratify=y,random_state=18)

#Defino el algoritmo a utilizar
from sklearn.svm import SVC
algoritmo = SVC(kernel = 'linear')

#Entreno el modelo
algoritmo.fit(X_train, y_train)

#Realizo una predicción
y_pred = algoritmo.predict(X_test)

#Verifico la matriz de Confusión
from sklearn.metrics import confusion_matrix

matriz = confusion_matrix(y_test, y_pred)
print('Matriz de Confusión:')
print(matriz)

#Calculo la precisión del modelo
from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred, average='macro')
print('Precisión del modelo:')
print(precision)
```

Figura 53. Código Python preprocesamiento datos parte 1

[1]:
```
#importando librerias
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import decomposition
from sklearn.preprocessing import LabelEncoder
```

[28]:
```
# importando el dataset a un Dataframe de Pandas
data_h = pd.read_csv('database.csv', ';', header=1)
```

[29]:
```
data_h.describe()
```

[29]:

	Num.	subject_ID	Age(year)	Height(cm)	Weight(kg)	Systolic Blood Pressure(mmHg)	Diastolic Blood Pressure(mmHg)	Heart Rate(b/m)	BMI(kg/m^2)
count	219.000000	219.000000	219.000000	219.000000	219.000000	219.000000	219.000000	219.000000	219.000000
mean	110.000000	156.598174	57.168950	161.228311	60.191781	127.945205	71.849315	73.639269	23.107215
std	63.364028	101.604347	15.874327	8.202871	11.886301	20.377779	11.111203	10.738874	4.004302
min	1.000000	2.000000	21.000000	145.000000	36.000000	80.000000	42.000000	52.000000	14.690000
25%	55.500000	85.500000	48.000000	155.000000	52.500000	113.500000	64.000000	66.000000	20.550000
50%	110.000000	152.000000	58.000000	160.000000	60.000000	126.000000	70.000000	73.000000	22.600000
75%	164.500000	214.500000	67.500000	167.000000	66.500000	139.000000	78.000000	80.000000	25.000000
max	219.000000	419.000000	86.000000	196.000000	103.000000	182.000000	107.000000	106.000000	37.460000

Figura 54. Código Python preprocesamiento datos parte 2

```
[35]: # Agrupando columnas por tipo de datos
      tipos = data_h.columns.to_series().groupby(data_h.dtypes).groups

      # Armando lista de columnas categóricas
      ctext = tipos[np.dtype('object')]
      len(ctext)  # cantidad de columnas con datos categóricos.
```

```
[35]: 5
```

```
[36]: # Armando lista de columnas numéricas
      columnas = data_h.columns  # lista de todas las columnas
      cnum = list(set(columnas) - set(ctext))
      len(cnum)
```

```
[36]: 9
```

```
[37]: # Completando valores faltantas datos cuantititavos
      for c in cnum:
          mean = data_h[c].mean()
          data_h[c] = data_h[c].fillna(mean)
```

```
[38]: # Completando valores faltantas datos categóricos
      for c in ctext:
          mode = data_h[c].mode()[0]
          data_h[c] = data_h[c].fillna(mode)
```

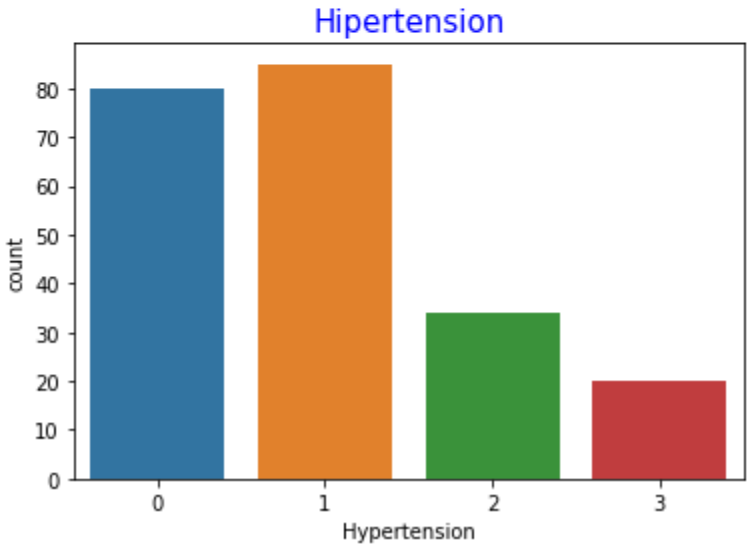
```
[39]: # Controlando que no hayan valores faltantes
      data_h.isnull().any().any()
```

```
[39]: False
```

Figura 55. Código Python preprocesamiento datos parte 3

```
[43]: data_h = data_h.replace('Female',0)
      data_h = data_h.replace('Male',1)
      data_h = data_h.replace('Normal',0)
      data_h = data_h.replace('Prehypertension',1)
      data_h = data_h.replace('Stage 1 hypertension',2)
      data_h = data_h.replace('Stage 2 hypertension', 3)
      data_h = data_h.replace('Diabetes', 1)
      data_h = data_h.replace('Type 2 Diabetes', 2)
      data_h = data_h.replace(',','.')
```

```
[44]: sns.countplot(data_h.Hypertension)
      #sns.countplot(kill.manner_of_death)
      plt.title("Hipertension",color = 'blue',fontsize=15)
      plt.show()
```



Anexo B. Código Django

Figura 56. Código Settings.py Django parte 1

```
ALLOWED_HOSTS = ['*']


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'usuarios',
    'hipertension',
    'diabetes',
    'diagnostico',
    'rest_framework',
    'rest_framework.authtoken',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'Tesis1.urls'
```

Figura 57. Código Settings.py Django parte 2

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'Tesis1.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': os.environ.get('POSTGRES_NAME'),
        'USER': os.environ.get('POSTGRES_USER'),
        'PASSWORD': os.environ.get('POSTGRES_PASSWORD'),
        'HOST': 'db',
        'DATABASE_PORT': '5433',
    }
}
```

Figura 58. Código Archivo Views.py (controlador) para diagnosticar hipertensión parte 1

```
def buscar(request):
    if (request.POST ["edad"] and request.POST ["presions"] and request.POST ["imc"] and request.POST [ "peso"]):

        usuariop=User.objects.get(username=request.user)

        presions=request.POST ['presions']
        #genero=request.POST ['select']
        edad=request.POST ['edad']
        imc=request.POST ['imc']
        peso=request.POST ['peso']
        #usuario=User.objects.username()
        #if(genero=="1"):
        #    sexo="Masculino"
        #elif(genero=="2"):
        #    sexo="Femenino"

        modelo= load('hipertension/modelo/modelo_entrenado_hipertension.pkl')
        pca= load("hipertension/modelo/modelo_pca_hipertension.pkl")
        presionsp=int(presions)
        pesop=int(peso)
        edadp=int(edad)
        imcp=int(imc)

        #X=np.array([[-90.4469, -14.529,    1.09817,    -1.40543,   -4.05994]])
        X=np.array([[edadp, pesop, presionsp ,imcp]])
        X_pca=pca.transform(X)
        y = modelo.predict(X_pca)
        if (y == 0):
            Resultado="Normal"
        elif(y==1):
            Resultado="Prehipertenso"
        elif(y==2):
            Resultado="Hipertension en estado 1"
        elif(y==3):
            Resultado="Hipertension en estado 2"


        datos=Hipertension(usuario=usuariop,presion_s=presions, edad=edad,  imc=imc, peso=pesop, resultado=Resultado)
        datos.save()
        consulta=Hipertension.objects.filter(usuario=usuariop).order_by("-id")
        #mensaje= "Tus datos : %r" %request.POST ["nombre"]
        #mensaje=usuario

        recomendaciones = []
        estados = []
        s = 0
```

Figura 59. Código Archivo Views.py (controlador) para diagnosticar hipertensión parte 2

```
    if(int(datos.presion_s) >120):
        estados.append("Presion arterial elevada: "+datos.presion_s)
        s=s+1

    if(float(datos.imc) > 25):
        estados.append("IMC no se encuentra en un rango de valores normales: "+datos.imc)
        s=s+1


    if(int(datos.presion_s) >120):
        recomendaciones.append(settings.RE_HIPERTENSION_ALTA)
        recomendaciones.append(settings.RE_GLUCOSA_ALTA)


    if(float(datos.imc) > 25):
        recomendaciones.append(settings.RE_IMC)

    if(int(datos.presion_s) <120 and float(datos.imc) < 25):
        recomendaciones.append('Sus valores se encuentran en un rango normal, sin embargo se recomienda realizar ejercicio al menos 3 veces por semana y comer saludable.')


else:

    Resultado="Error, llena todas las lineas"
    datos=0
    sexo=0
    usuariop=User.objects.get(username=request.user)
    consulta=Hipertension.objects.filter(usuario=usuariop).order_by("-id")


return render(request, "hipertension/resultado.html", {"resultados":Resultado,"estados":estados,"recomendaciones":recomendaciones, "datos":datos, "consultas":consulta})
```

Figura 60. Código Archivo Views.py (controlador) para el login y registro parte 1

```python
class SignUpView(CreateView):
    model = Perfil
    form_class = SignUpForm

    def form_valid(self, form):
        '''
        En este parte, si el formulario es valido guardamos lo que se obtiene de él y usamos authenticate para que el
        '''
        form.save()

        usuario = form.cleaned_data.get('username')
        password = form.cleaned_data.get('password1')
        usuario = authenticate(username=usuario, password=password)
        login(self.request, usuario)
        return redirect('/')

class BienvenidaView(TemplateView):
    template_name = 'usuarios/bienvenida.html'


from django.contrib.auth.views import LoginView

class SignInView(LoginView):
    form_class=LoginForm
    template_name = 'usuarios/iniciar_sesion.html'


from django.contrib.auth.views import LogoutView

class SignOutView(LogoutView):
    pass


def historial(request):
    usuariop=User.objects.get(username=request.user)
    consulta=Hipertension.objects.filter(usuario=usuariop).order_by("-id")

    consulta_diabetes = Diabetes.objects.filter(user=usuariop).order_by("-id")

    return render(request, "usuarios/historial.html", {"consultas":consulta, "consultadiabetes":consulta_diabetes})
```

Figura 61. Código Archivo Views.py (controlador) para el login y registro parte 2

```python
from django.core.mail import send_mail, BadHeaderError
from django.http import HttpResponse
from django.contrib.auth.forms import PasswordResetForm
from django.contrib.auth.models import User
from django.template.loader import render_to_string
from django.db.models.query_utils import Q
from django.utils.http import urlsafe_base64_encode
from django.contrib.auth.tokens import default_token_generator
from django.utils.encoding import force_bytes

def password_reset_request(request):
    if request.method == "POST":
        password_reset_form = UserForgotPasswordForm(request.POST)
        if password_reset_form.is_valid():
            data = password_reset_form.cleaned_data['email']
            associated_users = User.objects.filter(Q(email=data))
            if associated_users.exists():
                for user in associated_users:
                    subject = "Password Reset Requested"
                    email_template_name = "usuarios/restore-pass/password_reset_email.html"
                    c = {
                    "email":user.email,
                    'domain':'127.0.0.1:8000',
                    'site_name': 'Website',
                    "uid": urlsafe_base64_encode(force_bytes(user.pk)),
                    "user": user,
                    'token': default_token_generator.make_token(user),
                    'protocol': 'http',
                    }
                    email = render_to_string(email_template_name, c)
                    try:
                        send_mail(subject, email, 'hiperdiabet1@gmail.com' , [user.email], fail_silently=False)
                    except BadHeaderError:
                        return HttpResponse('Invalid header found.')
                    return redirect ("reset/password_reset_done")
    password_reset_form = UserForgotPasswordForm()
    return render(request=request, template_name="usuarios/restore-pass/password_reset_form.html", context={"password_reset_form":password_reset_form})
```

Anexo C. Lógica frontend (Django)

Figura 62. Código HTML base de la página web

```
<nav class="navbar navbar-expand-lg navbar-dark py-lg-4" id="mainNav">
  <div class="container">
    <a class="navbar-brand text-uppercase text-expanded font-weight-bold d-lg-none" href="{% url 'bienvenida' %}">Ayuda al diagnostico</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarResponsive" aria-controls="navbarResponsive" aria-expanded="false" aria-label="Toggl
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarResponsive">
      <ul class="navbar-nav mx-auto">
        <li class="nav-item active px-lg-4">
          <a class="nav-link text-uppercase text-expanded" href="{% url 'bienvenida' %}">Inicio</a>
        </li>
        <li class="nav-item px-lg-4">
          <a class="nav-link text-uppercase text-expanded" href="{% url 'Diagnostico' %}">Diagnosticar</a>
        </li>
        <li class="nav-item px-lg-4">
          <a class="nav-link text-uppercase text-expanded" href="{% url 'Modelo' %}">Hipertension</a>
        </li>
        <li class="nav-item px-lg-4">
          <a class="nav-link text-uppercase text-expanded" href="{% url 'diabetes' %}">Diabetes</a>
        </li>

      </ul>
      <ul class="navbar-nav mx-left">
          {% if user.is_authenticated %}
            <li class="nav-item px-lg-2 ">
              <div class="btn-group">
                <button type="button" class="btn btn-secondary btn-lg disabled">Hola, {{ user.username }}</button>
                <button type="button" class="btn btn-secondary btn-lg dropdown-toggle dropdown-toggle-split" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
                  <span class="sr-only">Toggle Dropdown</span>
                </button>
                <div class="dropdown-menu">
                  <a class="dropdown-item" href="{% url 'Historial' %}">Historial</a>
                  <div class="dropdown-divider"></div>
                  <a class="dropdown-item" href="{% url 'sign_out' %}">Cerrar Sesión</a>
                </div>
              </div>
            </li>
          {% else %}
            <li class="nav-item px-lg-2">
              <div class="btn-group">
                <button type="button" class="btn btn-secondary btn-lg "><a class="nav-link" href="{% url 'sign_in' %}">Inicia Sesión</a></button>
                <button type="button" class="btn btn-secondary btn-lg "><a class="nav-link" href="{% url 'sign_up' %}">Registrate</a></button>
              </div>
            </li>

          {% endif %}
```

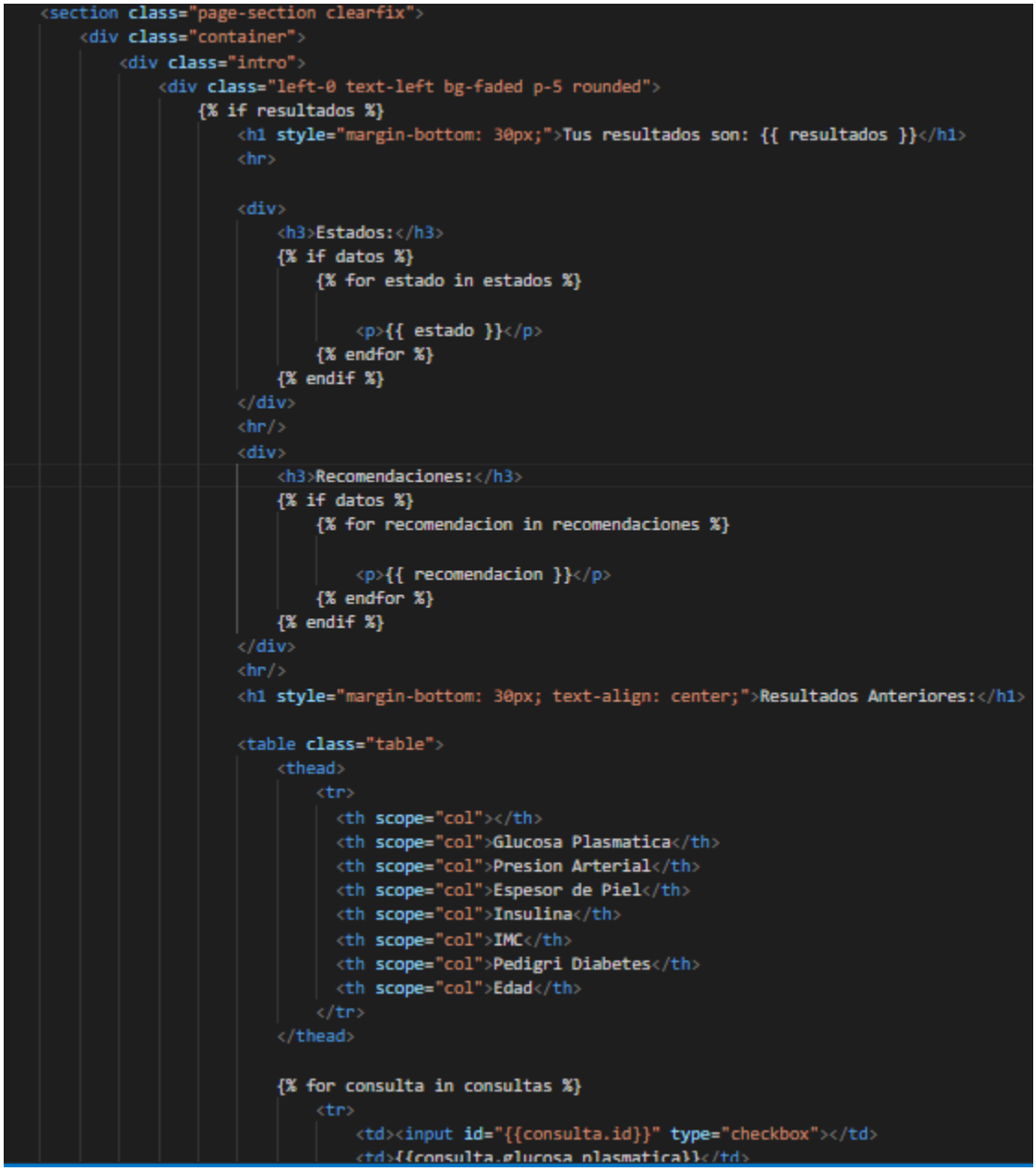
Figura 63. Código para crear el formulario que se muestra en la página web parte 1

```
class SignUpForm(UserCreationForm):
    first_name = forms.CharField(max_length=140, required=True, widget=forms.TextInput(
            attrs= {
                'placeholder':'Nombre',
                'class': 'form-control'
                } ))
    last_name  = forms.CharField(max_length=140, required=False,widget=forms.TextInput(
            attrs= {
                'placeholder':'Apellidos',
                'class': 'form-control'
                } ))
    email = forms.EmailField(required=True,widget=forms.EmailInput(
            attrs= {
                'placeholder':'Email',
                'class': 'form-control'
                } ))

    def __init__(self, *args, **kwargs):
        super(SignUpForm, self).__init__(*args, **kwargs)

        for fieldname in ['username', 'password1', 'password2']:
            self.fields[fieldname].help_text = None
            self.fields['username'].widget=forms.TextInput(
            attrs= {
                'placeholder':'Usuario',
                'class': 'form-control'
                } )

            self.fields['password1'].widget=forms.PasswordInput(
            attrs= {
                'placeholder':'Contraseña',
                'class': 'form-control'
                } )

            self.fields['password2'].widget=forms.PasswordInput(
            attrs= {
                'placeholder':'Confirma tu Contraseña',
                'class': 'form-control'
                } )

    class Meta:
        model = User
        fields = (
            'username',
            'email',
            'first_name',
            'last_name',
            'password1',
            'password2'
```

Figura 64. Código para crear el formulario que se muestra en la página web parte 2

```python
class LoginForm(AuthenticationForm):
    username = forms.CharField(widget=forms.TextInput(attrs={'placeholder':'Usuario','class':'form-control'}))
    password = forms.CharField(widget=forms.PasswordInput(attrs={'placeholder':'Contraseña','class':'form-control'}))

    def __init__(self, *args, **kwargs):
        super(LoginForm, self).__init__(*args, **kwargs)


class UserForgotPasswordForm(PasswordResetForm):
    email = forms.EmailField(required=True,max_length=254, widget=forms.EmailInput(attrs={'placeholder':'Ingresa Email','class':'form-control'}))
    class Meta:
        model = User
        fields = ("email")

class PasswordResetConfirmForm(SetPasswordForm):

    new_password1 = forms.CharField(widget=forms.TextInput(attrs={'placeholder':'Contraseña','class':'form-control',
                'type':'password'}))
    new_password2 = forms.CharField(widget=forms.PasswordInput(attrs= {'placeholder':'Confirma tu contraseña',
                'class': 'form-control', 'type':'password'} ))


    class Meta:
        model = User
        fields = (
            'new_password1',
            'new_password2',
        )
```

Figura 65. Código HTML de vista donde se muestra el resultado en página web



```
<section class="page-section clearfix">
    <div class="container">
        <div class="intro">
            <div class="left-0 text-left bg-faded p-5 rounded">
                {% if resultados %}
                    <h1 style="margin-bottom: 30px;">Tus resultados son: {{ resultados }}</h1>
                    <hr>

                    <div>
                        <h3>Estados:</h3>
                        {% if datos %}
                            {% for estado in estados %}

                                <p>{{ estado }}</p>
                            {% endfor %}
                        {% endif %}
                    </div>
                    <hr/>
                    <div>
                        <h3>Recomendaciones:</h3>
                        {% if datos %}
                            {% for recomendacion in recomendaciones %}

                                <p>{{ recomendacion }}</p>
                            {% endfor %}
                        {% endif %}
                    </div>
                    <hr/>
                    <h1 style="margin-bottom: 30px; text-align: center;">Resultados Anteriores:</h1>

                    <table class="table">
                        <thead>
                            <tr>
                              <th scope="col"></th>
                              <th scope="col">Glucosa Plasmatica</th>
                              <th scope="col">Presion Arterial</th>
                              <th scope="col">Espesor de Piel</th>
                              <th scope="col">Insulina</th>
                              <th scope="col">IMC</th>
                              <th scope="col">Pedigri Diabetes</th>
                              <th scope="col">Edad</th>
                            </tr>
                        </thead>

                        {% for consulta in consultas %}
                            <tr>
                                <td><input id="{{consulta.id}}" type="checkbox"></td>
                                <td>{{consulta.glucosa_plasmatica}}</td>
```

Anexo D. Lógica aplicación móvil

Figura 66. Código vista login aplicación móvil parte 1

```
private void attemptLogin() {

    // Reset errors.
    mFloatLabelUserId.setError(null);
    mFloatLabelPassword.setError(null);

    // Store values at the time of the login attempt.
    String userId = mUserIdView.getText().toString();
    String password = mPasswordView.getText().toString();

    boolean cancel = false;
    View focusView = null;

    // Check for a valid password, if the user entered one.
    if (TextUtils.isEmpty(password)) {
        mFloatLabelPassword.setError(getString(R.string.error_field_required));
        focusView = mFloatLabelPassword;
        cancel = true;
    } else if (!isPasswordValid(password)) {
        mFloatLabelPassword.setError(getString(R.string.error_invalid_password));
        focusView = mFloatLabelPassword;
        cancel = true;
    }

    // Verificar si el ID tiene contenido.
    if (TextUtils.isEmpty(userId)) {
        mFloatLabelUserId.setError(getString(R.string.error_field_required));
        focusView = mFloatLabelUserId;
        cancel = true;
    } else if (!isUserIdValid(userId)) {
        mFloatLabelUserId.setError(getString(R.string.error_invalid_user_id));
        focusView = mFloatLabelUserId;
        cancel = true;
    }
```

Figura 67. Código vista login aplicación móvil parte 2

```
if (cancel) {
    // There was an error; don't attempt login and focus the first
    // form field with an error.
    focusView.requestFocus();
} else {
    // Mostrar el indicador de carga y luego iniciar la petición asíncrona.
    showProgress(true);

    Call<Affiliate> loginCall = mSaludMockApi.login(new LoginBody(userId, password));
    loginCall.enqueue(new Callback<Affiliate>() {
        @Override
        public void onResponse(Call<Affiliate> call, Response<Affiliate> response) {
            // Mostrar progreso
            showProgress(false);

            // Procesar errores
            if (!response.isSuccessful()) {
                String error = "Ha ocurrido un error. Contacte al administrador";
                if (response.errorBody()  ResponseBody
                        .contentType()  MediaType
                        .subtype()  String
                        .equals("json")) {
                    ApiError apiError = ApiError.fromResponseBody(response.errorBody());

                    error = apiError.getMessage();
                    //Log.d("LoginActivity", apiError.getDeveloperMessage());
                } else {
                    try {
                        // Reportar causas de error no relacionado con la API
                        Log.d( tag: "LoginActivity", response.errorBody().string());
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }

                showLoginError(error);
                return;
```

Figura 68. Código vista login aplicación móvil parte 3

```
                // Guardar afiliado en preferencias
                SessionPrefs.get(LoginActivity.this).saveAffiliate(response.body());
                usuario = response.body().toString();
                // Ir a la citas médicas
                showAppointmentsScreen();

            }

            @Override
            public void onFailure(Call<Affiliate> call, Throwable t) {
                showProgress(false);
                showLoginError(t.getMessage());
            }
        });
    }
}

private boolean isUserIdValid(String userId) { return userId.length() > 4; }

private boolean isPasswordValid(String password) { return password.length() > 4; }

private void showProgress(boolean show) {
    mProgressView.setVisibility(show ? View.VISIBLE : View.GONE);

    int visibility = show ? View.GONE : View.VISIBLE;

    mLoginFormView.setVisibility(visibility);
}

private void showAppointmentsScreen() {
    Bundle bundle = new Bundle();
    Intent intent = new Intent( packageContext: this, MainActivity.class);
    bundle.putString("usuario", usuario );
    intent.putExtras(bundle);
    startActivity(intent);
    finish();
}
```

Figura 69. Código vista Registro aplicación móvil parte 1

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_register);

    final EditText firstnameEt = (EditText) findViewById(R.id.et_firstname);
    final EditText lastnameEt = (EditText) findViewById(R.id.et_lastname);
    final EditText usernameEt = (EditText) findViewById(R.id.et_username);
    final EditText emailEt = (EditText) findViewById(R.id.et_email);
    final EditText passwordEt = (EditText) findViewById(R.id.et_password);
    mFloatLabelFirstname = (TextInputLayout) findViewById(R.id.float_label_firstname);
    mFloatLabelLastname = (TextInputLayout) findViewById(R.id.float_label_lastname);
    mFloatLabelUsername = (TextInputLayout) findViewById(R.id.float_label_username);
    mFloatLabelEmail = (TextInputLayout) findViewById(R.id.float_label_email);
    mFloatLabelPassword2 = (TextInputLayout) findViewById(R.id.float_label_password_2);

    Button submitBtn = (Button) findViewById(R.id.btn_submit);
    mResponseTv = (TextView) findViewById(R.id.tv_response);

    mAPIService = ApiUtils.getAPIService();

    submitBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            if (!isOnline()) {
                showLoginError("Conexión de red no disponible");
                return;
            }

            String first_name = firstnameEt.getText().toString().trim();
            String last_name = lastnameEt.getText().toString().trim();
            String username = usernameEt.getText().toString().trim();
            String email = emailEt.getText().toString().trim();
            String password = passwordEt.getText().toString().trim();

            boolean cancel = false;
            View focusView = null;
```

Figura 70. Código vista Registro aplicación móvil parte 2

```
if (TextUtils.isEmpty(first_name)) {
    mFloatLabelFirstname.setError("Este campo es requerido");
    focusView = mFloatLabelFirstname;
    cancel = true;
} else if (!isFirstnameValid(first_name)) {
    mFloatLabelFirstname.setError("Contraseña inválida");
    focusView = mFloatLabelFirstname;
    cancel = true;
}

// Verificar si el ID tiene contenido.
if (TextUtils.isEmpty(last_name)) {
    mFloatLabelLastname.setError("Este campo es requerido");
    focusView = mFloatLabelLastname;
    cancel = true;
} else if (!isLastnameValid(last_name)) {
    mFloatLabelLastname.setError("Número de identificación inválido");
    focusView = mFloatLabelLastname;
    cancel = true;
}
if (TextUtils.isEmpty(username)) {
    mFloatLabelUsername.setError("Este campo es requerido");
    focusView = mFloatLabelUsername;
    cancel = true;
} else if (!isUsernameValid(username)) {
    mFloatLabelUsername.setError("Número de identificación inválido");
    focusView = mFloatLabelUsername;
    cancel = true;
}
if (TextUtils.isEmpty(email)) {
    mFloatLabelEmail.setError("Este campo es requerido");
    focusView = mFloatLabelEmail;
    cancel = true;
} else if (!isEmailValid(email)) {
    mFloatLabelEmail.setError("Número de identificación inválido");
    focusView = mFloatLabelEmail;
    cancel = true;
```

Figura 71. Código vista Registro aplicación móvil parte 3

```
public void sendPost( String first_name, String last_name, String username, String email, String password) {
    mAPIService.savePost(first_name,last_name,username,email,password).enqueue(new Callback<Post>() {
        @Override
        public void onResponse(Call<Post> call, Response<Post> response) {

            if (!response.isSuccessful()) {
                String error = "Ha ocurrido un error. Contacte al administrador";
                if (response.errorBody() ResponseBody
                        .contentType() MediaType
                        .subtype() String
                        .equals("json")) {
                    ApiError apiError = ApiError.fromResponseBody(response.errorBody());

                    error = apiError.getMessage();
                    //Log.d("LoginActivity", apiError.getDeveloperMessage());
                } else {
                    try {
                        // Reportar causas de error no relacionado con la API
                        Log.d( tag: "RegisterActivity", response.errorBody().string());
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }

                showLoginError(error);
                return;
            }
            if(response.isSuccessful()) {
                showResponse(response.body().toString());
                showLoginScreen();
                Log.d( tag: "good",  msg: "post submitted to API." + response.body().toString());
            }
        }

        @Override
        public void onFailure(Call<Post> call, Throwable t) {
            showLoginError(t.getMessage());
```

Figura 72. Código vista Menú aplicación móvil parte 1

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Redirección al Login
    if (!SessionPrefs.get(this).isLoggedIn()) {
        startActivity(new Intent( packageContext: this, LoginActivity.class));
        finish();
        return;
    }

    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    toolbar.setTitleTextAppearance( context: this, R.style.RobotoBoldTextAppearance);
    getSupportActionBar().setTitle("Home");


    DrawerLayout drawer = findViewById(R.id.drawer_layout);
    ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
            activity: this, drawer, toolbar, "Open navigation drawer", "Close navigation drawer");
    drawer.addDrawerListener(toggle);
    toggle.syncState();

    NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view); //Este es el menu del lado
    navigationView.setNavigationItemSelectedListener(this);

    //startActivity(new Intent(this, HomeActivity.class));
    //finish();
    HomeFragment fragment = new HomeFragment();
    getSupportFragmentManager().beginTransaction().replace(R.id.conteiner,fragment).addToBackStack(null).commit();


}
```

Figura 73. Código vista Menú aplicación móvil parte 2

```
public void onBackPressed() {
    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    if (drawer.isDrawerOpen(GravityCompat.START)) {
        drawer.closeDrawer(GravityCompat.START);
    } else {
        super.onBackPressed();
    }
    if (getFragmentManager().getBackStackEntryCount()>0){
        getFragmentManager().popBackStack();
    }
}

/StatementWithEmptyBody/
@Override
public boolean onNavigationItemSelected(MenuItem item) {
    // Handle navigation view item clicks here.
    int id = item.getItemId();

    boolean fragmentTraslacion = false;
    Fragment fragment = null;


    if (id == R.id.nav_home) {
        fragment = new HomeFragment();
        fragmentTraslacion = true;
        getSupportActionBar().setTitle(item.getTitle());
        //return true;
    }
    else if (id == R.id.nav_hipertension) {
        fragment = new Hipertension2Fragment();
        fragmentTraslacion = true;
        getSupportActionBar().setTitle(item.getTitle());

    }
    else if (id == R.id.nav_diabetes) {
        fragment = new DiabetesFragment();
        fragmentTraslacion = true;
        getSupportActionBar().setTitle(item.getTitle());
```

Figura 74. Código vista Menú aplicación móvil parte 3

```
    else if (id == R.id.cerrarsesion) {
        SessionPrefs.get(MainActivity.this).logOut();
        startActivity(new Intent( packageContext: this, LoginActivity.class));
        finish();
        return true;
    }

    if (fragmentTraslacion){
        getSupportFragmentManager().beginTransaction().replace(R.id.conteiner,fragment).addToBackStack(null).commit();
    }

    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    drawer.closeDrawer(GravityCompat.START);
    return true;
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_appointments, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        SessionPrefs.get(MainActivity.this).logOut();
        startActivity(new Intent( packageContext: this, LoginActivity.class));
        finish();
        return true;
    }
```

Figura 75. Código vista formulario Edad aplicación móvil

```
public View onCreateView(@NonNull LayoutInflater inflater,
                         ViewGroup container, Bundle savedInstanceState) {

    View view = inflater.inflate(R.layout.fragment_edad, container,  attachToRoot: false);
    EditText edad= (EditText) view.findViewById(R.id.edad_in);
    Button button = (Button) view.findViewById(R.id.btnedad);
    Bundle datosRecuperados = getArguments();


    button.setOnClickListener(new View.OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            String nombre2 = edad.getText().toString();
            if (!TextUtils.isEmpty(nombre2)) {
                Bundle datosEnviar = new Bundle();
                //datosEnviar.putString("nombre",nombre);
                datosEnviar.putString("edad",nombre2);
                PesoFragment nuevoFragmento = new PesoFragment();
                nuevoFragmento.setArguments(datosEnviar);
                FragmentTransaction transaction = getFragmentManager().beginTransaction();
                transaction.replace(R.id.conteiner, nuevoFragmento);
                transaction.addToBackStack(null);
                // Commit a la transacción
                transaction.commit();
            }else{
                edad.setError("Error, Por favor ingresa tu edad");
            }


        }
    });
    return view;
}
```