UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Desarrollo de una plataforma de consultoría médica online

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Esteban Blasco, Pablo

Tutor/a: López Patiño, José Enrique

CURSO ACADÉMICO: 2023/2024

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

PORTADA GENERADA AUTOMÁTICAMENTE EN EBRON

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS
VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS
Gestión ambiental verificada
REG.NO. ES-CV- 000030

Resumen

Este proyecto consiste en el planteamiento, creación y despliegue de una aplicación web para la salud. La aplicación consistirá en una plataforma web en la que pacientes y médicos podrán ponerse en contacto, reduciendo así la distancia entre estos. La plataforma permitirá que tanto clientes como especialistas se registren. Por una parte, los médicos serán capaces de anunciarse, especificando la especialidad a la que pertenecen y los horarios de sus consultas. Por el otro lado, los pacientes, podrán elegir la consulta del especialista que deseen y el horario. Para la consulta paciente y médico, podrán comunicarse tanto por un chat de mensajes de texto como por videoconferencia, y todo esto sin salir de la aplicación.

Resum

Aquest projecte consisteix en el plantejament, creació i desplegament d'una aplicació web per a la salut. L'aplicació consistirà en una plataforma web on pacients i metges podran posar-se en contacte, reduint així la distància entre ells. La plataforma permetrà que tant clients com a especialistes es registren. Per una banda, els metges seran capaços de publicitar-se, especificant l'especialitat a la qual pertanyen i els horaris de les seves consultes. D'altra banda, els pacients podran triar la consulta de l'especialista que desitgin i l'horari. Per a la consulta pacient i metge, podran comunicar-se tant per un xat de missatges de text com per videoconferència, i tot això sense sortir de l'aplicació.

Abstract

This project involves the design, development, and deployment of a web application for healthcare. The application will consist of a web platform where patients and doctors can connect, reducing the distance between them. The platform will allow both clients and specialists to register. On one hand, doctors will be able to advertise themselves, specifying their specialty and consultation hours. On the other hand, patients will be able to choose the specialist's consultation and schedule. For the patient-doctor consultation, they will be able to communicate through a text message chat and video conferencing, all within the application.

Índice

Contenido

Índice de Figuras

Capítulo 1. Introducción

1.1 Presentación del proyecto y su contexto

Vivimos en un mundo cada vez más digitalizado, en el que prácticamente cualquier individuo cuenta con un dispositivo con acceso a Internet. Este fenómeno no solo refleja una penetración masiva de la tecnología en la sociedad, sino que también subraya la oportunidad de aprovechar estas herramientas para superar barreras geográficas y mejorar significativamente el acceso a los servicios de salud. La tecnología puede desempeñar un papel fundamental en la mejora de la accesibilidad y la calidad de la atención médica, mejorando la comunicación entre pacientes y profesionales de la salud.

En el actual escenario, después de una pandemia que impuso restricciones a la movilidad ciudadana, llevando a la población a permanecer en sus hogares y limitando el acceso a la atención médica personal. Restringiéndola a casos de extrema urgencia o a aquellos afectados por la COVID-19. Con esto se evidenció una tendencia en la que las personas solo buscaban asistencia médica en situaciones críticas. En numerosas ocasiones, cuando se buscaba ayuda, estas situaciones ya habían alcanzado niveles avanzados, resultando en complicaciones adicionales. Un diagnóstico más temprano habría contribuido significativamente a simplificar el abordaje de los problemas de salud.

Las crisis sanitarias, han destacado la importancia de contar con soluciones que permitan a pacientes y médicos conectarse de manera rápida y efectiva. Esta aplicación busca abordar estos desafíos, proporcionando una plataforma integral que facilita la interacción entre profesionales de la salud y usuarios, sin importar la ubicación física.

Si bien es cierto que en el panorama actual existen diversas aplicaciones que integran funciones similares, como por ejemplo “Doki”, es relevante destacar que estas no suelen estar disponibles de manera pública. En su mayoría, estas aplicaciones son propiedad de centros de salud o aseguradoras privadas que ofrecen servicios en línea exclusivamente a sus clientes. Sin embargo, esta exclusividad plantea limitaciones para aquellos que no son miembros de dichos centros o que buscan acceder a servicios de consulta de forma puntual.

Adicionalmente, muchas de estas aplicaciones presentan funcionalidades e interfaces de usuario complejas, lo cual puede resultar confuso para personas de mayor edad que no están familiarizadas con tecnologías avanzadas.

Este proyecto se centra en el desarrollo de una aplicación web que facilita la comunicación entre ciudadanos y profesionales médicos de cualquier especialidad, permitiéndoles agendar citas con el objetivo de abordar sus inquietudes de salud en la medida de lo posible. La aplicación ha sido concebida y diseñada con la premisa de maximizar la simplicidad y la intuición, garantizando así su accesibilidad incluso para aquellos menos familiarizados con la tecnología, incluyendo específicamente a la población de edad avanzada.

1.2 Justificación y relevancia del proyecto

La implementación de una aplicación web para la salud se justifica en varias dimensiones. En primer lugar, la tecnología puede reducir las barreras de acceso a la atención médica, especialmente para aquellos que enfrentan limitaciones geográficas, movilidad reducida o situaciones de emergencia. Además, en situaciones de pandemia o crisis sanitaria, donde la distancia física es crucial, contar con una herramienta que permita la atención médica a distancia se vuelve esencial.

El enfoque en la simplicidad y la intuitividad responde a la necesidad de crear una herramienta que no solo cumpla su función principal de facilitar la interacción entre ciudadanos y médicos, sino que también elimine posibles obstáculos tecnológicos para usuarios que puedan tener una menor familiaridad con plataformas digitales.

Asimismo, la aplicación contribuirá a optimizar la gestión del tiempo tanto para médicos como para pacientes al ofrecer la posibilidad de programar consultas de manera eficiente. Esto no solo beneficia a los usuarios al proporcionar flexibilidad en la elección de especialistas y horarios, sino que también permite a los médicos organizar sus agendas de manera más efectiva.

La relevancia de este proyecto radica en su potencial para mejorar la eficiencia del sistema de atención médica, fortalecer la relación médico-paciente y proporcionar una solución innovadora que se adapte a las demandas cambiantes de la sociedad actual y resuelva los problemas existentes en aplicaciones similares.

1.3 Metodología

La implementación de una metodología en un proyecto es crucial por varias razones, ya que proporciona una estructura organizada y guía para la planificación, ejecución y control del proyecto.

Para el desarrollo de este proyecto se han contemplado diversas metodologías de trabajo, para finalmente optar por la metodología Waterfall. Por sus características, esta metodología es la más optima ya que se trata de una metodología simple, que se adapta muy bien a proyectos individuales.

1.3.1 Metodología Waterfall

La metodología Waterfall, también conocida como método de desarrollo en cascada, se clasifica como una metodología secuencial que se utiliza para representar y gestionar proyectos a través de distintas fases sucesivas. Estas fases guían el funcionamiento del proyecto de manera secuencial, desde el principio hasta el final.

Si bien la metodología Waterfall se utiliza comúnmente en el desarrollo de software, también es aplicable a cualquier proyecto que requiera un sistema de funcionamiento para su ejecución. Aunque se le atribuye la invención de esta metodología a Winston Royce, es importante destacar que lo que él hizo fue una reflexión crítica sobre los procedimientos lineales de gestión de software, en su ensayo "*Managing the Development of Large Software Systems*" en 1970. Es relevante señalar que el término "*Waterfall*" no aparece en su ensayo y se aplica como una traducción del término "cascada", que visualiza gráficamente el funcionamiento de este método.

A raíz de esta crítica, Royce propuso como alternativa un modelo iterativo incremental, en el cual cada fase se basa en la anterior y verifica los resultados obtenidos en dicha fase. Esta propuesta surge como una respuesta a las limitaciones identificadas en los procedimientos lineales de gestión de software. [1]

1.3.1.1 Funcionamiento

La metodología Waterfall, propuesta por Royce, se estructura en un modelo de siete fases a ejecutar en iteraciones: 1- Requisitos del sistema; 2- Requisitos de software; 3- Análisis; 4- Diseño; 5- Implementación; 6- Prueba; 7- Servicio. [21]

No obstante, en la práctica, se emplean diversas versiones del modelo, siendo las más comunes las que consolidan las fases 1, 2 y 3 en la etapa de análisis de los requisitos, como se detalla a continuación:

1. Análisis: Incluye planificación, análisis y especificación de los requisitos.

2. Diseño: Se centra en el diseño y especificación del sistema.

3. Implementación: Comprende la programación y las pruebas unitarias.

4. Verificación: Involucra la integración de sistemas, pruebas de sistema y de integración.

5. Mantenimiento: Engloba la entrega, mantenimiento y mejora.

1.3.1.2 Fases

Las fases de la Metodología Waterfall desempeñan un papel fundamental en la conducción y viabilidad del desarrollo del proyecto, ya que el progreso de una fase a otra solo es posible si la fase precedente se ha aplicado correctamente. Estas fases son las siguientes:

Fase 1: Análisis

En esta etapa se realiza un estudio de viabilidad y se define claramente los requisitos del proyecto. El análisis de viabilidad evalúa costos, rentabilidad y factibilidad, proporcionando una descripción general de los requisitos, un plan y una estimación financiera. Además, se elabora una propuesta para el cliente si es necesario. Esta fase implica un análisis detallado de la definición de los requisitos, donde los problemas complejos se desglosan en tareas secundarias más pequeñas, acompañadas de estrategias de resolución correspondientes.

Esta fase corresponde con el capítulo 2 de la memoria.

Fase 2: Diseño

La fase de diseño tiene como objetivo formular una solución específica basada en las exigencias, tareas y estrategias definidas en la fase anterior. Se traduce en un borrador preliminar con el plan de diseño del proyecto.

Esta fase corresponde con el capítulo 3 de la memoria.

Fase 3: Implementación

Aquí, se ejecuta la arquitectura del proyecto concebida en la fase de diseño. Incluye programación, búsqueda de errores y pruebas unitarias. Esta etapa culmina en un producto que se somete a la primera comprobación como producto final en la siguiente fase.

Esta fase corresponde con el capítulo 4 de la memoria.

Fase 4: Prueba

Esta fase abarca la integración del proyecto en el entorno seleccionado. Las pruebas de aceptación desarrolladas en la fase de análisis permiten determinar si el proyecto cumple con las exigencias predefinidas. Los productos que superan con éxito estas pruebas están listos para ser lanzados.

Esta fase corresponde con el capítulo 5 de la memoria.

Fase 5: Servicio

Una vez que la fase de prueba concluye exitosamente, se autoriza la aplicación fructífera del proyecto. Esto incluye la entrega, el mantenimiento y la mejora continua del producto.

Esta fase corresponde con el capítulo 6 de la memoria.

La Metodología Waterfall, al igual que otras metodologías tradicionales basadas en etapas secuenciales, garantiza el desarrollo correcto y la finalización de cada etapa. Dado que estas etapas están interrelacionadas, cualquier inconveniente en una fase impide avanzar a la siguiente. Este tipo de desarrollo permite un control efectivo del proceso por parte de todos los responsables del proyecto y sus superiores.

1.3.1.3 Ventajas e inconvenientes

La metodología Waterfall ofrece varias ventajas, destacando la facilidad para controlar el progreso del proyecto y la opción para que el cliente no se involucre si algo no le resulta convincente. Sin embargo, su principal ventaja radica en un presupuesto cerrado acordado con el proveedor desde el principio. Una vez que el proyecto está definido, se vuelve “estático” y el precio no cambia al no introducir cambios en el proceso de desarrollo.

A pesar de esto, esta ventaja puede convertirse en un contrapunto, ya que la falta de cambios en el desarrollo del software puede generar mayores gastos en etapas posteriores. Después de probar el programa, los trabajadores pueden requerir funcionalidades que no se consideraron en la fase inicial de definición de características, lo que conlleva la necesidad de solicitar un nuevo presupuesto para abordar estos requisitos adicionales.

La metodología Waterfall se suele emplear, especialmente, en procedimientos estrictamente lineales. Este enfoque se adapta especialmente bien a proyectos pequeños, sencillos y claramente estructurados.

Capítulo 2. Objetivos del proyecto

La primera fase de la metodología "Waterfall" consiste en la recopilación y documentación de los objetivos del proyecto. Por esa razón a continuación se recogen los objetivos y las principales características de este proyecto.

2.1 Objetivo

El objetivo de este proyecto consiste en la creación de una aplicación WEB para la realización de consultas médicas. Existirán dos tipos de usuarios Doctores y Pacientes, los cuales tendrán propiedades y capacidades distintas, pero que podrán realizar consultas por medio de videoconferencias sin hacer uso de aplicaciones externas. Los pacientes tendrán la capacidad filtrar por consultas en base a especialidades y horarios y solicitar estas con los doctores que deseen.

La aplicación web será desplegada en un proveedor de servicios en la nube que asegure accesibilidad global.

2.2 Características

A continuación, se hace una recopilación de las principales características que deberá de cumplir la aplicación. En la fase de prueba se verificará que el producto cumple cada una de ellas.

Interfaz WEB

- Toda interacción con la aplicación se hará por medio de la aplicación WEB. No ser requerirá uso de aplicaciones externas.
- La aplicación ha de funcionar en los principales navegadores: Chrome, Firefox, Opera, Microsoft Edge y Safari.
- La aplicación deberá de tener al menos las siguientes interfaces
    - Interfaz principal desde la que se podrán consultar la disponibilidad de los distintos doctores, así como programar una cita.
    - Perfiles de usuario con la información pública de este.
    - Interfaz para editar el perfil de usuario.
    - Panel con las futuras consultas.
    - Interfaz en la que se podrá consultar la información de cada cita médica, así como acceder a la videoconferencia
    - Videoconferencia

Registro de usuarios

- Capacidad de registro para nuevos pacientes.
- Capacidad de registro de nuevos doctores.
    - El registro de los doctores requerirá de la activación del perfil por parte de un administrador. Esto se debe a que se ha de confirmar la capacidad de ejercer su especialidad.
    - Capacidad de doctores de registro en más de una especialidad.

Usuarios anónimos

- Podrán acceder a la página principal y consultar la disponibilidad de los doctores. No podrán coger citas con estos.

- Capacidad de iniciar sesión como cliente o doctor.

Doctores

- Propiedades de usuario:
  - Email
  - Usuario
  - Nombre y apellidos
  - Especialidades
  - Descripción personal
- Capacidad de editar su perfil.
- Capacidad de especificar su horario semanal.
  - Podrán especificar los intervalos de horas en los que desean trabajar.
  - Podrán especificar la duración de sus consultas.
- Panel para consultar sus futuras citas.
- Capacidad de realizar videoconferencias con pacientes.

Pacientes

- Propiedades de usuario:
  - Email
  - Usuario
  - Nombre y apellidos
  - Fecha de nacimiento
- Capacidad de editar su perfil.
- Capacidad de coger citas con los doctores.
- Panel para consultar sus futuras citas.
- Capacidad de realizar videoconferencias con doctores.

Videoconferencia

- Se tendrá la capacidad de realizar videoconferencias sin uso de aplicaciones externas.
- Capacidad de encendido y apagado de micrófono.
- Capacidad de encendido y apagado de cámara.
- Capacidad de terminar videoconferencia

Administración

- Usuarios administradores
  - Podrán administrar a los usuarios.
  - Podrán verificar a los doctores y activarlos en la aplicación.
- Panel de administración solo accesible por los administradores.

Servicio

- 100% de disponibilidad
- Bajo tiempo de respuesta

Capítulo 3. Diseño

En esta fase se analizarán los objetivos documentados en la fase anterior para diseñar un producto que cumpla con todos estos. Durante esta fase se llevará a cabo el análisis y selección de las tecnologías a emplear, se propondrá la estructura final del proyecto en la que se explicaran como los distintos componentes interaccionaran entre sí y por último se diseñaran los diagramas de flujo, modelos e interfaces de la aplicación.

3.1 Herramientas, tecnologías o lenguajes utilizados

Durante el diseño del proyecto se han analizado diversas tecnologías que podría suplir las necesidades de este, algunas de ellas fueron descartadas porque no cumplían las características requeridas. Otras en cambio han sido aceptadas y utilizadas durante la realización del proyecto. En los siguientes puntos se mencionan todas ellas.

3.1.1 Python

Python es un lenguaje de programación interpretado de alto nivel y de propósito general. Su diseño enfatiza la legibilidad del código, utilizando una sintaxis que permite a los programadores expresar conceptos en menos líneas de código.

Python surgió a finales de los años ochenta, con el propósito de suceder al lenguaje ABC. Fue creado por Guido van Rossum en Stichting Mathematisch Centrum (CWI) en los Países Bajos, y debe su nombre a la afición de su creador por los humoristas británicos Monty Python. [2] [3]

3.1.1.1 Características

Algunas de sus características más importantes son las siguientes:

Legibilidad y Comprensión: Los programas escritos en Python son fácilmente legibles y comprensibles para los desarrolladores, gracias a su sintaxis básica, la cual guarda similitudes con el idioma inglés.

Productividad Mejorada: Python contribuye a la productividad de los desarrolladores al permitir la redacción de programas con una cantidad reducida de líneas de código en comparación con diversos lenguajes, facilitando así la eficiencia en el desarrollo.

Amplia Biblioteca Estándar: La extensa biblioteca estándar de Python alberga códigos reutilizables para una amplia gama de tareas, eliminando, en ocasiones, la necesidad de crear código desde cero.

Interoperabilidad con Otros Lenguajes: Python se integra sin dificultades con otros lenguajes de programación notables, como Java, C y C++, proporcionando a los desarrolladores flexibilidad en sus elecciones tecnológicas.

Apoyo de una Comunidad Activa: La comunidad activa de Python, conformada por millones de desarrolladores a nivel mundial, ofrece un respaldo significativo. Ante cualquier inconveniente, los desarrolladores pueden obtener un soporte ágil y eficaz de esta comunidad comprometida.

Abundancia de Recursos en Internet: Abundan recursos útiles en línea para aprender Python, incluyendo videos, tutoriales, documentación y guías para desarrolladores, facilitando el proceso de adquisición de conocimientos.

Portabilidad entre Sistemas Operativos: Python exhibe la capacidad de trasladarse con facilidad entre diferentes sistemas operativos de computadora, tales como Windows, macOS, Linux y Unix, brindando versatilidad en su implementación.

3.1.1.2 Motivo de elección

Como se puede apreciar Python es un leguaje de alto nivel que permite ha sido utilizado en creación de aplicaciones profesionales como, por ejemplo: Instagram, Netflix y Spotify. Además, cabe destacar que se dispone conocimiento previo por parte del autor en este lenguaje. Es por esto y por todas las características mencionadas anteriormente que se ha elegido Python como el lenguaje de programación que se empleará para la realización del proyecto.

3.1.2 Django

Django constituye un framework web de alto nivel, concebido para agilizar el desarrollo eficiente de sitios web seguros y sostenibles. Elaborado por profesionales experimentados, Django aborda de manera exhaustiva las complejidades inherentes al desarrollo web, posibilitando que los desarrolladores se centren en la creación de aplicaciones sin redundar en soluciones ya existentes. Este framework, de naturaleza gratuita y de código abierto, se destaca por su comunidad activa, documentación extensa y opciones variadas de soporte, tanto gratuito como con opciones de pago. [4]

3.1.2.1 Características

Completo: Django sigue la filosofía de "Baterías incluidas", proporcionando una integración exhaustiva de prácticamente todos los elementos necesarios para el desarrollo web. Esto garantiza una funcionalidad homogénea, principios de diseño coherentes y documentación detallada y actualizada.

Versátil: Este framework se erige como una herramienta versátil, apta para la construcción de una amplia gama de sitios web, desde sistemas de gestión de contenidos hasta plataformas de redes sociales. Su compatibilidad con diversos frameworks del lado del cliente y su capacidad para presentar contenido en múltiples formatos lo hacen sumamente flexible.

Seguro: Django contribuye a la seguridad de manera proactiva, ofreciendo un framework diseñado para salvaguardar automáticamente los sitios web. Proporciona métodos seguros para la gestión de cuentas de usuario y contraseñas, evitando vulnerabilidades como el almacenamiento de información de sesión en cookies y asegurando contraseñas mediante técnicas de hash.

Escalable: Con una arquitectura basada en el principio “shared-nothing”, Django exhibe escalabilidad al gestionar eficazmente el aumento de tráfico mediante la incorporación de hardware en diferentes niveles, como servidores de cache, bases de datos o servidores de aplicación. Ejemplos notables, como Instagram, respaldan su eficacia en entornos de alta concurrencia.

Mantenible: El código de Django se adhiere a principios y patrones de diseño que promueven la creación de código mantenible y reutilizable. El principio “No te repitas” (DRY) se aplica para evitar redundancias innecesarias, mientras que la organización en aplicaciones reutilizables simplifica la gestión de la funcionalidad relacionada.

Portable: Al ser desarrollado en Python, Django posee portabilidad y puede ejecutarse en diversas plataformas. Esta característica confiere al framework independencia de plataformas específicas, permitiendo su implementación en distintas distribuciones de Linux, Windows y Mac OS X. Además, el respaldo de múltiples proveedores de alojamiento web facilita la adopción de Django en diversos entornos.

3.1.2.2 Motivo de elección

Django ha sido seleccionado como el framework para el proyecto debido a su enfoque integral, versatilidad, seguridad proactiva, escalabilidad y la facilidad de mantenibilidad del código. Además, la movilidad que ofrece Django, respaldada por la sólida presencia y apoyo de Python, da al proyecto la flexibilidad necesaria para su implementación en diversas plataformas.

3.1.3 Entorno virtual de Python

Un entorno virtual en Python funciona como un entorno aislado donde el intérprete de Python, las bibliotecas y los scripts están encapsulados, diferenciados de los instalados en otros entornos virtuales o del entorno Python predeterminado del sistema.

venv es un módulo incorporado en Python que facilita la creación y gestión de entornos virtuales. Estos entornos virtuales permiten a los desarrolladores encapsular las dependencias específicas de un proyecto, incluyendo las versiones de las bibliotecas y el intérprete de Python. [5] [6]

3.1.3.1 Características

Cuando se activa un entorno virtual para un proyecto, se convierte en una entidad autocontenida con sus propias características únicas, las características de estos entornos virtuales:

Aislamiento: El entorno de desarrollo queda confinado dentro del proyecto, evitando interferencias con el Python instalado en el sistema u otros entornos virtuales.

Múltiples Versiones de Python: Los entornos virtuales permiten la creación de entornos aislados para diferentes versiones de Python, adaptándose a diversos requisitos del proyecto.

Compartir y Replicar Fácilmente: Las aplicaciones dentro de entornos virtuales pueden empaquetarse y compartirse fácilmente con otros desarrolladores, facilitando la replicación.

Gestión de Dependencias: La creación de una lista de dependencias y subdependencias en un archivo para el proyecto simplifica el proceso de replicar e instalar todas las dependencias necesarias.

3.1.3.2 Motivo de elección

En este proyecto se creará y trabajará en un entorno virtual. En él se instalarán las dependencias y versiones deseadas de todas las librerías necesarias. Se ha decidido trabajar de esta forma ya que permite el aislamiento del proyecto y dependencias con el resto de los proyectos del dispositivo en el que se trabajará.

3.1.4 Video Calling de agora.io

Agora.io es una empresa americana que ofrece una Plataforma de Comunicaciones como Servicio (CPaaS) que proporciona comunicaciones en tiempo real para desarrolladores y empresas a nivel global.

Ofrecen la funcionalidad llamada "*Video Calling*", que permite incorporar experiencias de video en tiempo real personalizadas a diversas aplicaciones. Gracias a su robusto SDK de Video, *Agora* proporciona un control total sobre una variedad de funciones, desde grabación hasta moderación de contenido. Además, ofrecen la capacidad de integrar extensiones del mercado, como eliminación de fondo y filtros faciales. [7]

3.1.4.1 Características

Algunas de las características del Producto *Video Calling* de *Agora.io* son:

Grabación de Llamadas: Permite la grabación de llamadas de video en la nube o en instalaciones locales, ofreciendo control sobre el formato, la ruta de almacenamiento y la calidad.

Compartir Pantalla y Colaboración: Facilita la función de compartir pantalla o utilizar pizarras interactivas, permitiendo a los usuarios dibujar, anotar y compartir contenido desde varios dispositivos de manera simultánea.

Múltiples Pistas de Audio y Video: Posibilita la publicación de múltiples pistas de audio y video en uno o más canales desde una única instancia. Admite cámaras y micrófonos de captura multicanal.

Video de Alta Calidad a Escala: Garantiza video de alta calidad y consistente, desde llamadas individuales (1:1) hasta miles de usuarios concurrentes, incluso en condiciones de red desafiantes.

Mejora de Audio Impulsada por IA: Ofrece soporte para audio de alta calidad mediante características como audio espacial 3D, supresión de ruido impulsada por inteligencia artificial y control de ganancia, proporcionando así una experiencia de audio inmersiva.

Cobertura Global: La red en tiempo real definida por software de Agora (SD-RTN) respalda a usuarios de video en más de 200 países y regiones, asegurando una cobertura global efectiva.

3.1.4.2 Motivo de elección

Se ha optado por esta tecnología para la implementación de las videoconferencias porque, agora.io ofrece una versión gratuita de su producto en la que dispondremos de 10.000 minutos gratis de videoconferencia cada mes, que será más que suficiente. Además, existe una gran comunidad en torno a los productos de *agora.io* la cual ha creado una gran cantidad de tutoriales y guías que serán muy útiles para la realización del proyecto.

3.1.5 FullCalendar

FullCalendar es una biblioteca de JavaScript altamente personalizable y potente utilizada para crear calendarios de eventos en aplicaciones web. Es de código abierto, lo que significa que está disponible de forma gratuita para que los desarrolladores lo utilicen y lo modifiquen según sus necesidades.

FullCalendar admite la integración con varios marcos de JavaScript populares, como React, Vue y Angular, lo que lo hace versátil para diferentes entornos de desarrollo. [8] [9]

3.1.5.1 Características

FullCalendar ofrece a los desarrolladores web las siguientes características:

Versatilidad: *FullCalendar* es compatible con varios marcos de JavaScript populares, como React, Vue y Angular. Esto significa que se puede integrar fácilmente en diferentes entornos de desarrollo.

Altamente personalizable: Con más de 300 configuraciones, *FullCalendar* puede adaptarse a una amplia gama de requisitos. Permite la personalización de la apariencia del evento, como establecer el color de fondo, el color del borde y las clases CSS para cada registro de calendario.

Interactivo: *FullCalendar* cuenta con capacidades de arrastrar y soltar, así como ganchos para eventos activados por el usuario. También puede buscar eventos sobre la marcha para cada mes utilizando AJAX.

Internacionalización: *FullCalendar* admite la localización, lo que significa que puede adaptarse a diferentes idiomas y zonas horarias.

Versión premium: Además de la versión gratuita de código abierto, *FullCalendar* también ofrece una versión premium con características adicionales, como Vista de Línea de Tiempo, Vista de Recursos Verticales y renderizado para impresora.

3.1.5.2 Motivo de elección

La elección de FullCalendar para nuestra aplicación web se debe a su versatilidad y capacidad de personalización. Además, al tratarse de una aplicación de código abierto gratuita, no requeriremos de licencia para su utilización y podrá ser integrado sin gastos adicionales.

3.1.6 Heroku

Heroku es una plataforma como servicio (PaaS) que permite a los desarrolladores construir, ejecutar y escalar aplicaciones de manera eficiente y rápida. Esta plataforma, pionera en el ámbito de la nube, ha estado en desarrollo desde 2007. [10] [11] [12]

3.1.6.1 Características

Alguna de las características de Heroku son las siguientes:

Soporte de múltiples lenguajes de programación: Heroku es compatible con diversos lenguajes de programación, como Node.js, Ruby, Java, Python, Scala, PHP y más. Esto facilita a los desarrolladores escalar sus aplicaciones sin importar su lenguaje de desarrollo preferido.

Modelo de contenedor gestionado: Heroku se basa en un modelo de contenedor gestionado llamado "*dynos*". Este enfoque permite a los desarrolladores escalar sus aplicaciones fácilmente, ya sea añadiendo *dynos* o cambiando el tipo de *dyno* que ejecuta la aplicación.

Despliegue y configuración simplificados: Heroku simplifica procesos como el despliegue, configuración, escalado, ajuste y gestión de aplicaciones. Esto permite que los desarrolladores se centren en lo más importante: construir aplicaciones excepcionales.

Add-ons incorporados: Heroku ofrece una variedad de add-ons incorporados, desde alarmas hasta herramientas analíticas y servicios de seguridad para monitoreo, caché, correo y red.

Heroku Teams: Heroku Teams proporciona un entorno colaborativo para desarrolladores y sus colaboradores. Facilita la colaboración entre usuarios de Heroku, permitiendo la modificación de permisos, la gestión administrativa, la configuración de recursos y más.

Documentación y tutoriales completos: Heroku ofrece tutoriales y documentación exhaustivos y útiles. Los usuarios pueden encontrar ejemplos en todos los lenguajes de programación compatibles con Heroku en su repositorio de GitHub.

Compatibilidad con Linux: Además de los lenguajes oficialmente compatibles, Heroku permite el uso de cualquier lenguaje que funcione en Linux a través de un *buildpack* de terceros.

3.1.6.2 Motivo de elección

Elegimos Heroku como proveedor de hosting para nuestro proyecto debido a su destacada posición como servicio PaaS en la nube, con soporte para Django. Heroku dispone de un nivel gratuito el cual es suficiente para cubrir nuestras necesidades, además de una amplia comunidad con tutoriales y guías que serán de gran ayuda.

3.1.7 Git

Git, un sistema de control de versiones distribuido de código abierto, se destaca por su capacidad para gestionar proyectos de cualquier escala con velocidad y eficiencia. [19]

3.1.7.1 Características

Entre sus características principales se encuentran:

Control de versiones distribuido: A diferencia de los sistemas centralizados, Git permite a cada desarrollador tener una copia completa del repositorio en su máquina local, facilitando el trabajo sin conexión o de forma remota.

Soporte para múltiples ramas: Con excepcionales capacidades de gestión de ramas, Git facilita el flujo de trabajo de ramas de características, garantizando que la rama principal siempre contenga código de calidad de producción.

Velocidad y eficiencia: Git ha sido diseñado para ser rápido y eficiente en comparación con otras alternativas de control de versiones.

Compatibilidad con la mayoría de los sistemas operativos: Git se puede instalar en sistemas operativos comunes como Windows, Mac y Linux, y a menudo viene preinstalado en máquinas Mac y Linux.

Comunidad de usuarios activa: Como la herramienta más ampliamente adoptada en su categoría, Git cuenta con una comunidad activa que proporciona recursos para formación y soporte.

3.1.7.2 Motivo de elección

Se ha elegido *Git* para por su flexibilidad, el amplio respaldo de la comunidad y su capacidad para gestionar proyectos de cualquier tamaño.

3.2 Estructura del proyecto

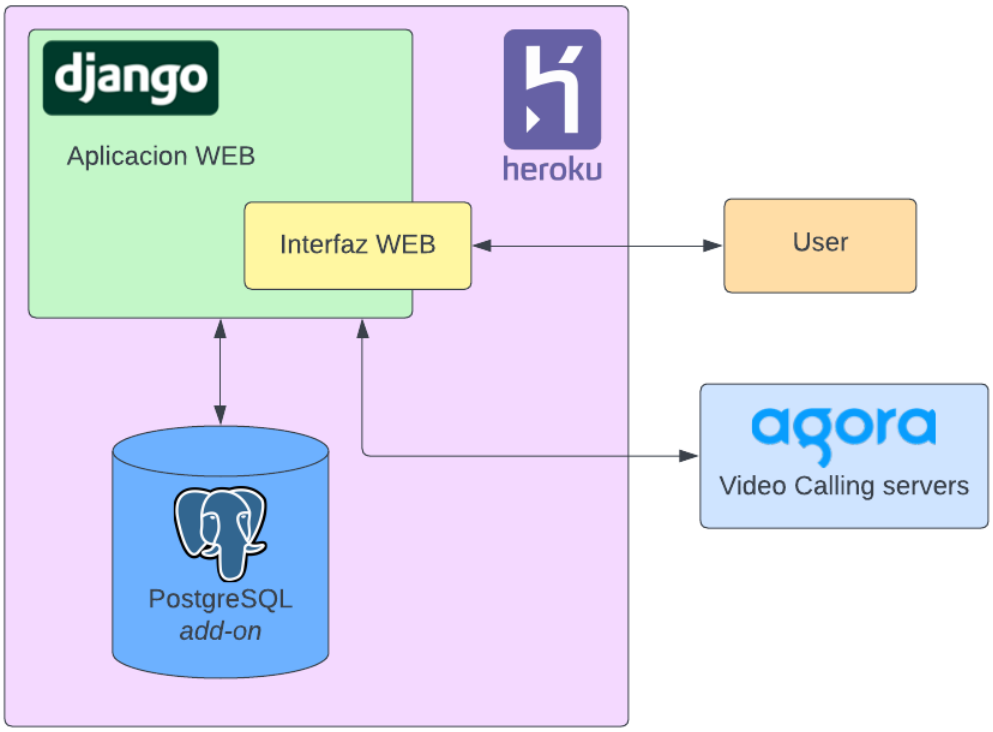

Figura 1. Vista general de la estructura de los componentes.

El diagrama anterior muestra una visión simple y general de la estructura de este proyecto. Como se puede apreciar el proyecto consistirá en una aplicación WEB programada en el framework Django. Esta aplicación tendrá un interfaz WEB a través de la que accederán los usuarios. Además de esta interfaz la aplicación será capaz de comunicarse con una base de datos PostgreSQL en la que se almacenará todos los datos de esta. La aplicación también será capaz de comunicarse con los servidores de la plataforma "*agora*" que se usarán para llevar a cabo las videoconferencias.

La aplicación será desplegada en los servidores de Heroku, donde se comportará como un dyno. Además, se incorporará con un add-on para la base de datos PostgreSQL, asegurando una sólida y eficiente gestión de datos.

3.3 Diagramas de Flujo

Con intención de diseñar los procesos más importantes de la aplicación se han diseñado diversos diagramas de flujo en los que se representan las interacciones de los usuarios con el sistema, así como los procesos internos de este.

La intención interacciones es que sean lo más simples e intuitivas posibles. Se pretende que la aplicación pueda ser utilizada por usuarios de cualquier edad. Por norma general, las personas de avanzada edad no están muy familiarizadas con las tecnologías, así que un simplificar al máximo las interacciones ayudara a que puedan hacer un manejo correcto de la aplicación y no se vean abrumados por opciones innecesarias.

3.3.1 Registro de usuarios

Todo usuario tendrá la capacidad de registrarse en la aplicación como paciente o como médico. Los pacientes simplemente tendrán que rellenar un formulario con sus datos, si estos son correctos y el usuario no existe, tendrán acceso completo a la aplicación. Los doctores, en cambio, requerirán de un paso extra. Un administrado deberá verificar que disponen las acreditaciones necesarias para ejercer la especialidad que acreditan.

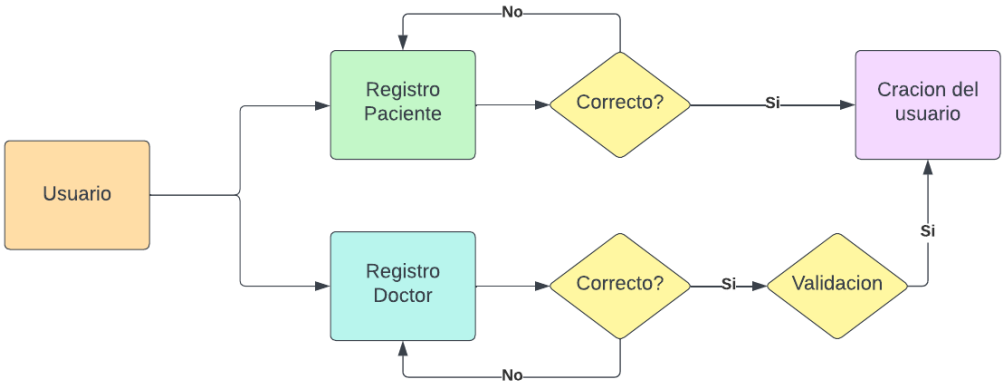

Figura 2. Diagrama de flujo para el registro de usuarios.

3.3.2 Inicio de sesión del usuario

Para iniciar sesión, todo usuario rellenara el mismo formulario para el inicio de la sesión. El formulario es el mismo tanto para pacientes como doctores. El usuario estará vinculado a una cuenta de paciente o doctor, así que la sesión se asociará con esta.

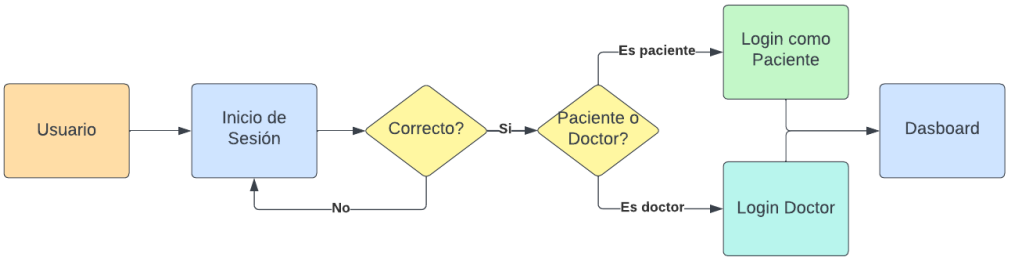

Figura 3. Diagrama de flujo para el inicio de sesión.

3.3.3 Vista general sesión de paciente

Los pacientes serán capaces de realizar diversas acciones dentro de la aplicación. Todas estas acciones serán accesibles desde la vista "*dashboard*". Esto no quiere decir que las puedan hacer desde allí, sino que desde allí podrán dirigirse a la vista correspondiente para llevar a cabo estas. Estas acciones son: actualizar su información de usuario, pedir una nueva cita o acceder a una cita ya concedida.

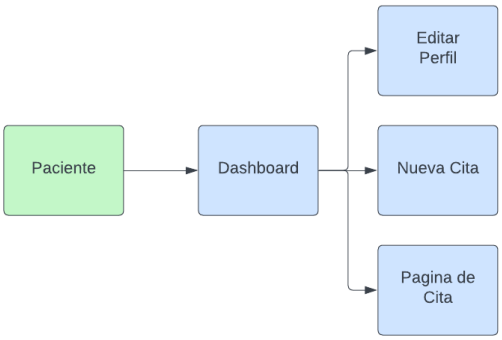

Figura 4. Diagrama de flujo de una sesión de paciente.

3.3.4 Paciente escoge un doctor y pide una cita

Para la petición de la cita por parte del usuario, el flujo consistirá en dos partes. Una primera en la que el paciente podrá filtrar por especialidades, día de la semana y disponibilidad horaria y donde seleccionará el doctor con el que desea tener la consulta. Será redirigido a una segunda vista en la que seleccionará la fecha y hora en la que desea tener la consulta. Por último, se creará la cita solicitada.

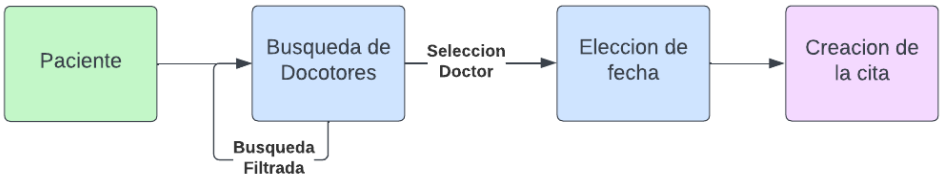

Figura 5. Diagrama de flujo en el que un paciente pide una cita con un doctor de su elección.

3.3.5 Paciente accede a videoconferencia de una cita

Una vez se ha solicitado la cita con el doctor y esta ha sido creada en la base de datos, el paciente deberá poder acceder a esta. Para ello, desde el "*dashboard"*, podrá ver todas las citas en la que se encuentra inscrito y podrá seleccionar la que le interese en ese momento. Será redirigido a la página de la videoconferencia donde podrá ver la información básica de esta como el doctor con el que se llevará a cabo, la hora, la duración o la especialidad. Desde esta interfaz, podrá acceder a la página de la videoconferencia donde paciente y medico se reunirán para llevar a cabo la consulta.

Figura 6. Diagrama de flujo del acceso a la videoconferencia por el paciente.

3.3.6 Vista general sesión de doctor

Los doctores tendrán sus propios flujos aun también compartirán una parte con los pacientes. Entre otras cosas, los doctores serán capaces de: modificar sus datos de usuario, así como planificar su horario semanal en el que impartirán consultas y la duración de estas; consultar los perfiles de los pacientes donde podrán ver las anotaciones que han hecho sobre estos en caso de que lo necesiten; y al igual que los pacientes, podrán acceder a las páginas de las citas.

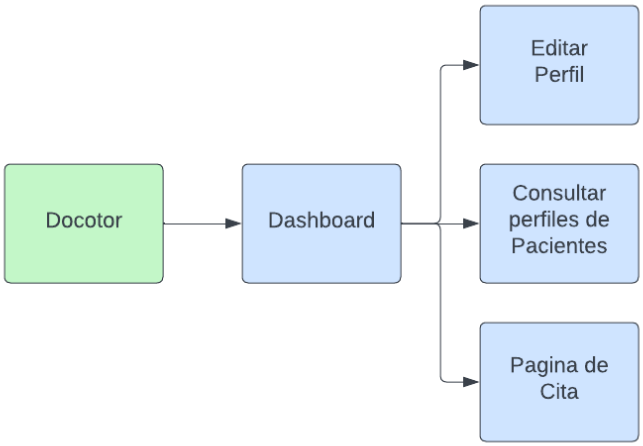

Figura 7. Diagrama de flujo de una sesión de doctor.

3.3.7 Doctor modifica sus datos de usuario

Los doctores tendrán la capacidad de configurar sus perfiles, modificando sus datos de usuario y especificando su horario semanal. Todo esto se realizará desde la página para la edición del perfil. Para la configuración del horario semanal podrán especificar los intervalos de horas que desean trabajar cada día de la semana, así como la duración de estas consultas.

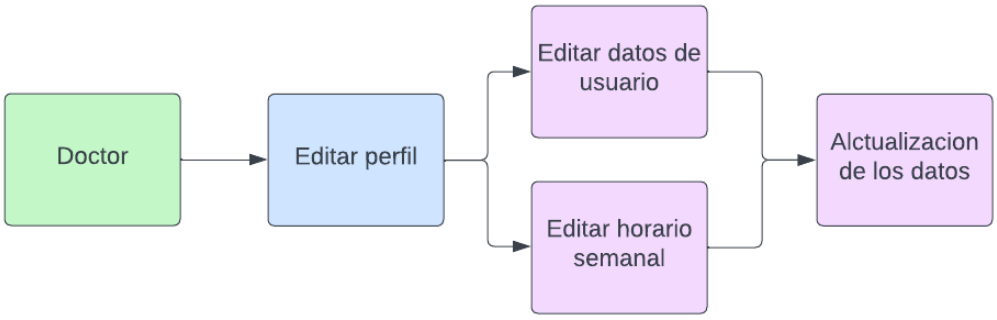

Figura 8. Diagrama de flujo de la actualización de los datos de usuario.

3.3.8 Doctor accede a la cita

Al igual que el paciente el flujo para acceder a la videoconferencia será el mismo, a diferencia que el doctor tendrá una funcionalidad extra. El doctor podrá crear notas sobre el paciente, apuntando todo aquello que crea conveniente. Estas anotaciones solo serán visibles por él.

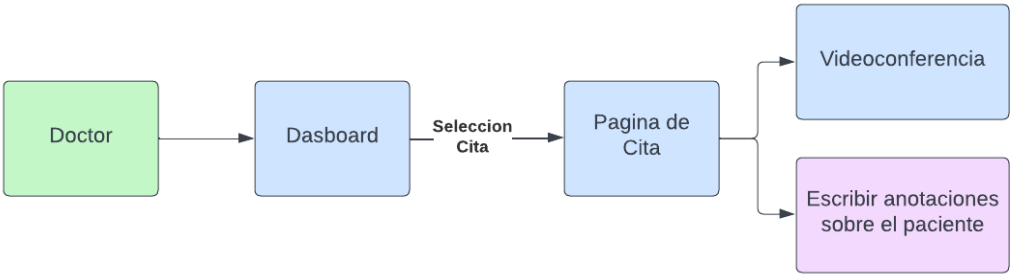

Figura 9. Diagrama de flujo del acceso a la cita por parte del doctor.

3.3.9 Vista general sesión de administrador

Los administradores. Para ello, dispondrán de su propia interfaz desde donde podrán llevar a cabo sus funciones. Están funciones incluyen: activación o desactivación de los usuarios de doctores tras la verificación manual de que estos disponen de las acreditaciones necesarias para ejercer su especialidad; y administración de los datos de usuarios, así como los datos de la aplicación, por poner un ejemplo, serán ellos los que puedan crear nuevos tipos de especialidades.

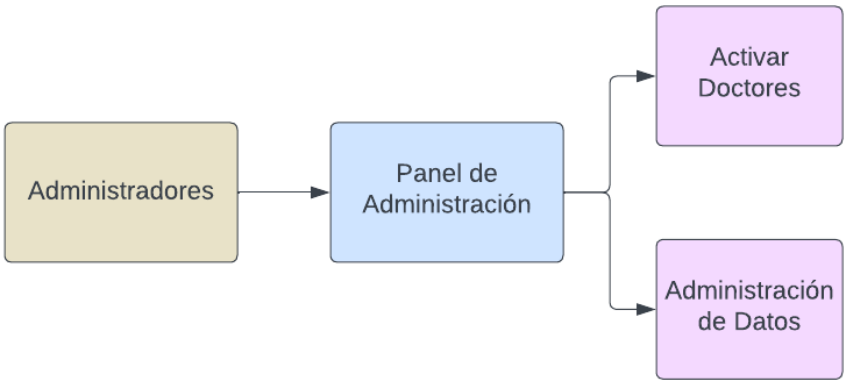

Figura 10. Diagrama de flujo de una sesión de administrador.

3.4 Modelos

Analizando las especificaciones principales de la aplicación y teniendo en cuenta las características que esta debe cumplir, se han diseñado los modelos que deberán ser creados como mínimo durante el desarrollo. Estos modelos se implementarán utilizando la característica de *models* de Django.

A continuación, se presenta un diagrama con los modelos donde se representa los atributos y como se relacionan entre sí.

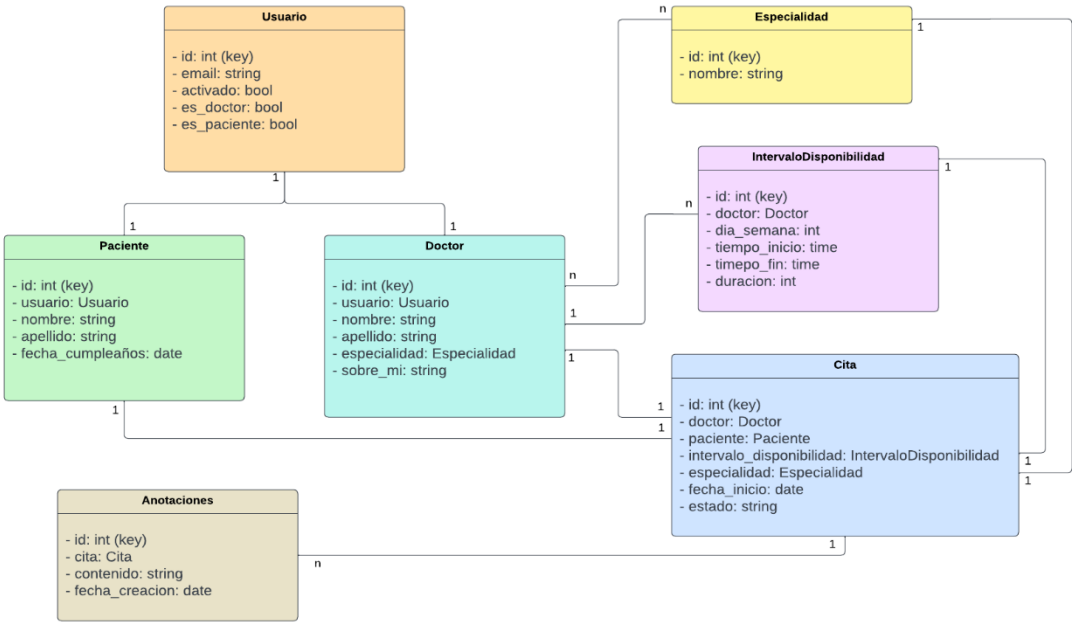

Figura 11. Vista general de los modelos de la aplicación y sus relaciones.

A modo de explicación del diagrama, se comentan los siguientes puntos:

Como se ha mencionado antes, se empleará el framework de Django para la realización de la aplicación. Django permite la opción de usar un modelo de usuario con la librería *django.contrib.auth* el cual permite implementar la autenticación de los usuarios de una forma más simple y segura.

Para poder diferenciar entre dos tipos de usuarios paciente y doctor se propone vincular ambos modelos con un usuario en concreto. Esto permitirá utilizar los métodos predefinidos por Django pero a su vez poder especificar atributos únicos para cada clase.

A la clase usuario predefinida por Django le han sido añadidos otros atributos que se consideran necesarios y que son comunes tanto para el modelo doctor como el paciente. Estos atributos son:

- email: especificará la dirección de correo electrónico
- activado: flag que indicara si el usuario ha sido activado por los administradores. Los pacientes serán activados automáticamente, pero los doctores tendrán que ser activados de forma manual por los administradores.
- es_doctor y es_paciente: indican si los usuarios están asociados con un doctor o un paciente. Esto simplificará el desarrollo de la aplicación.

El modelo “Especialidad” constituirá un listado de todas las especialidades disponibles dentro de la aplicación y que irán asociadas a doctores y citas. Estas solo podrán ser creadas por los administradores.

Los “intervalos de disponibilidad” serán creados por los doctores, en ellos se podrá especificar la hora de inicio y fin del intervalo, el día de la semana al que estará asociado y la duración de sus citas.

La “citas” irán asociadas a un doctor, un paciente, un intervalo de disponibilidad y una especialidad. Además, tendrán otros atributos como la fecha de inicio de esta o el estado. El resto de los datos interesantes de esta podrán ser obtenidos a través sus asociaciones con otros modelos. Por ejemplo, para obtener la duración de esta podríamos emplear la relación con “Intervalo de disponibilidad”.

La “anotaciones” serán las notas que los doctores puedan tomar a cerca de los pacientes. Estas serán únicamente asociadas a las citas desde donde se podrá obtener toda la información. Además, dispondrán de un atributo con el comentario y otro que guardará automáticamente la fecha en la que fue creada.

3.5 Interfaces

Una parte importante del diseño trata de diseñar como van a ser las interfaces de usuario. Esta parte es con la que interacciona el usuario y es muy importante que las interfaces sean intuitivas y los usuarios no se sientan perdidos entre estas. Además de esto, un diseño bonito mejorará la experiencia de usuario.

La idea principal es que todas las interfaces mantengan un estilo común que no variará entre ellas. Este estilo consistirá en un fondo de color verde claro y una barra de navegación en la parte superior. En esta barra aparecerán: a la izquierda, el logo de la aplicación junto con el nombre de esta. A la derecha el icono del usuario junto con su nombre y nombre de usuario si el usuario no está autenticado todavía, aparecerá “login”. Además de esto, si el usuario esta autenticado y hace “*click*”, aparecerá un menú desde el que se podrá dirigir a las interfaces de “dashboard”, “editar perfil” o cerrar sesión.

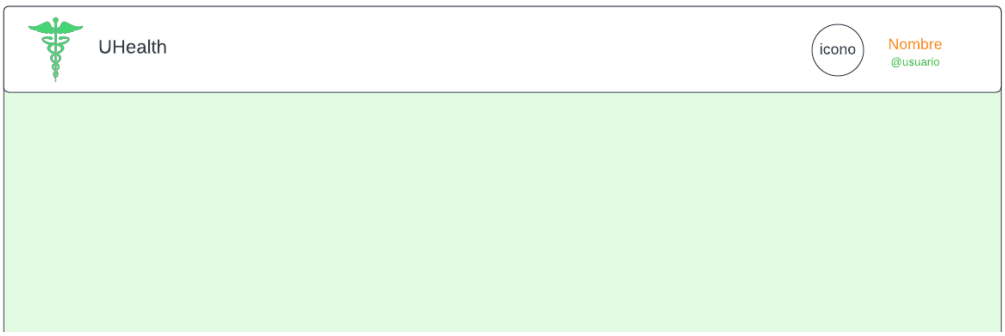

Figura 12. Diseño de la barra de navegación.

Además, de esto muchas de las interfaces dispondrán de un panel a la derecha desde el que se podrán dirigir a las interfaces más importantes como son: el “dashboard”, la interfaz para buscar doctores desde donde comienza el flujo para crear una nueva cita y la interfaz para editar el perfil.

A partir de ahora nos referiremos a este panel como “panel de movilidad”.

Figura 13. Diseño del panel de movilidad.

3.5.1 Dashboard

Esta será la interfaz base, una vez el usuario se haya autenticado. Desde esta interfaz, el usuario, será capaz de dirigirse a cualquier parte de la aplicación. En la parte izquierda aparecerán las siguientes citas del usuario, así como las citas pasadas. El usuario podrá acceder a cualquiera de estas citas a través del enlace de "Ir a la cita" que aparecerá en la parte de abajo a la derecha. En la parte derecha de esta interfaz, aparece el *panel de movilidad*.

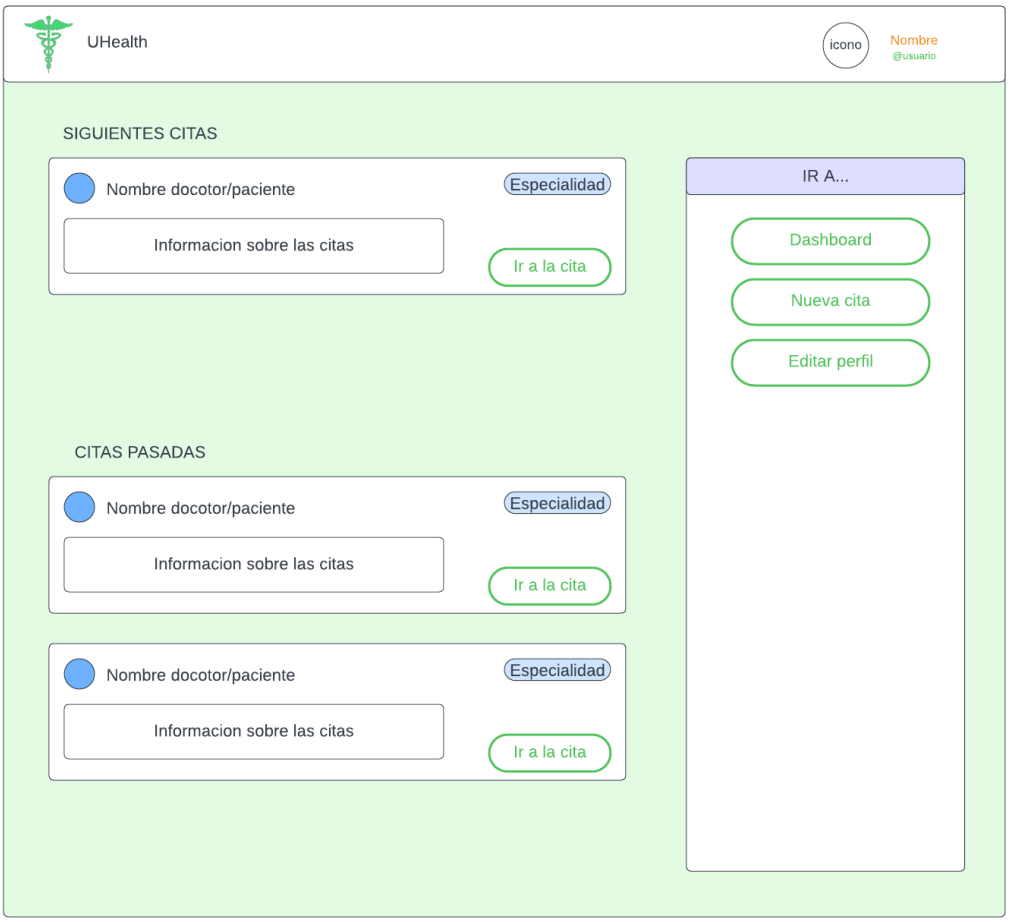

Figura 14. Diseño de la interfaz "dashboard"

3.5.2 Búsqueda de doctores

Desde esta interfaz comienza el flujo para pedir una cita. Es en esta interfaz donde los clientes podrán filtrar por especialidad, día de la semana y fecha de inicio y fin para seleccionar un doctor.

En la parte izquierda aparecerán la lista de doctores que cumplen los filtros especificados por el cliente. Junto a estos se muestra la información sobre los doctores, así como las especialidades en las que están dados de alta.

En la parte de la derecha aparecerán los filtros y el *panel de movilidad*.

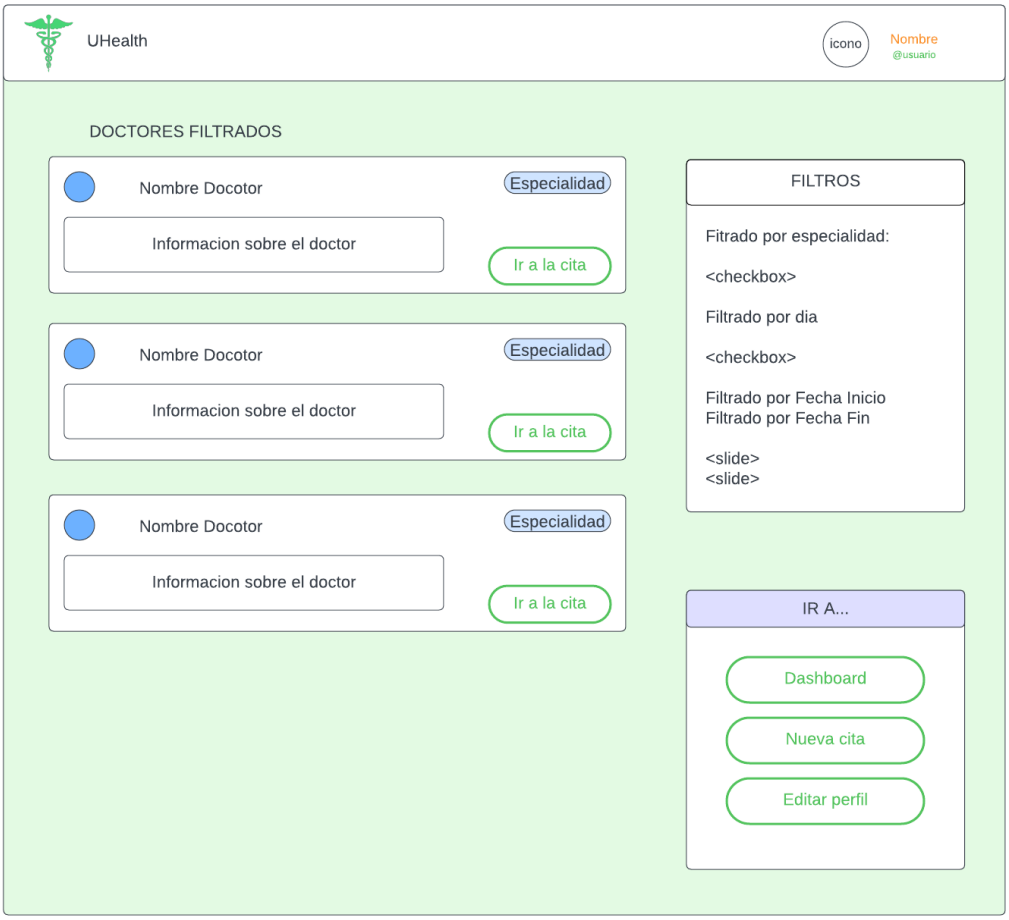

Figura 15. Diseño de la interfaz de búsqueda de doctores.

3.5.3 Selección del horario de consulta

Una vez el paciente ha escogido el doctor con el que desea pedir una consulta, es redirigido al este panel. En el deberá de seleccionar la hora a la que desea tener la consulta, así como la especialidad en la que desea, dentro de la lista de especialidades del doctor.

El selector de la fecha consistirá en un calendario en el que aparecerán eventos. Cada uno de estos corresponderá con una cita disponible con ese doctor. Si el paciente hace pulsa en el reservará automáticamente la cita y esta quedará deshabilitada para el resto de los usuarios.

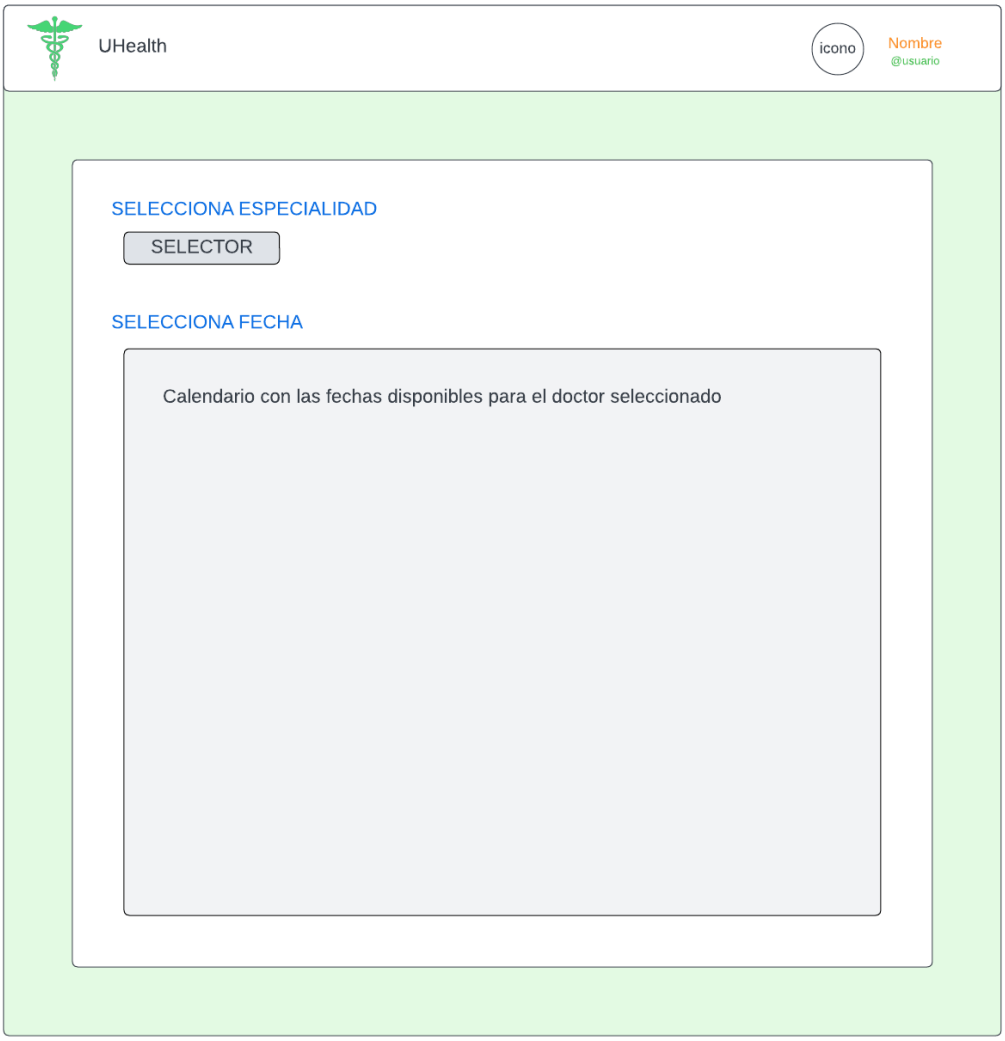

Figura 16. Diseño de la interfaz para la selección del horario de consulta.

3.5.4 Perfil

La interfaz de los perfiles de usuario tendrá una parte común que consiste en el icono de usuario, nombre, apellidos y nombre de usuario. El resto de la interfaz dependerá de si se trata de un perfil de paciente o de doctor.

Si se trata del perfil de un doctor, entonces, aparecerá un campo con información sobre el doctor, esta información es la que se encuentra en el atributo "sobre_mi" del modelo doctor.

Si se trata del perfil de un paciente, y el usuario autenticado es un doctor, entonces, aparecerán todas las notas que el doctor haya tomado sobre ese paciente en específico en cualquier consulta anterior.

En la parte de la derecha aparecerán los filtros y el *panel de movilidad*.

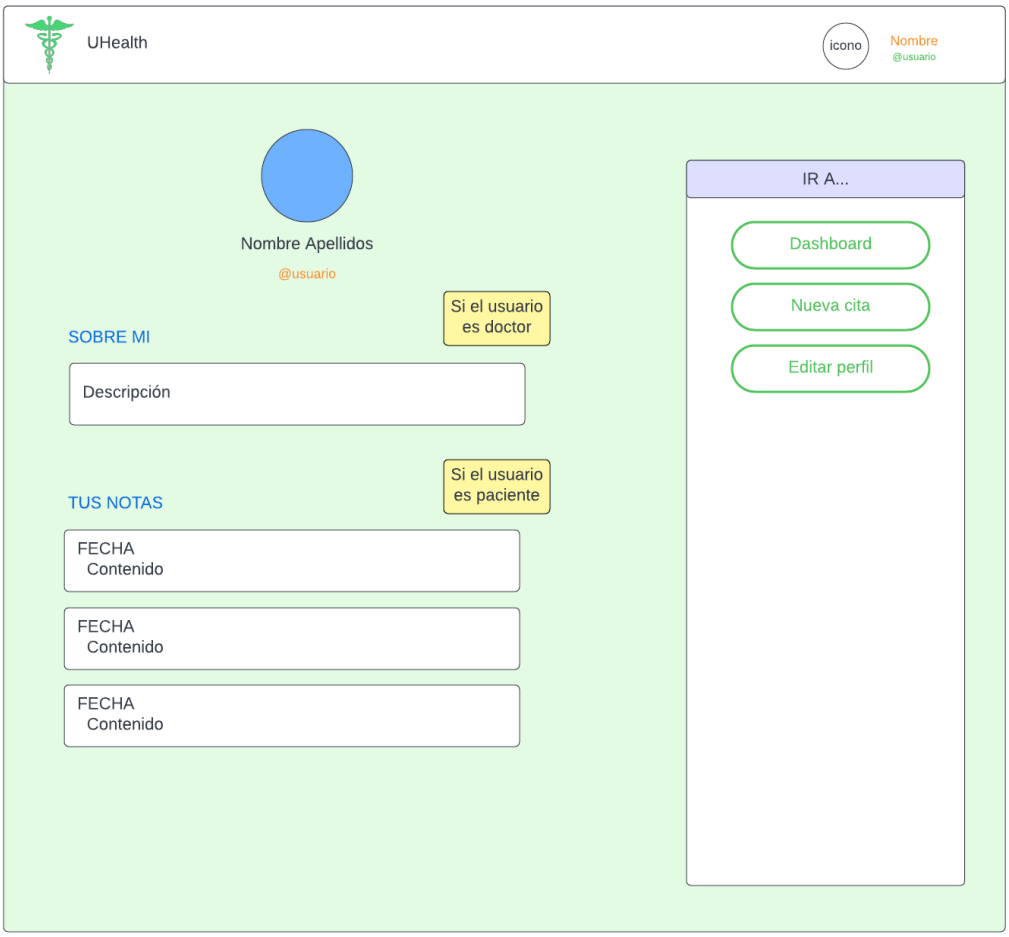

Figura 17. Diseño de la interfaz del perfil de usuario.

3.5.5 Editar perfil

Esta interfaz permite a los usuarios editar sus perfiles. En ella se muestran:

En la parte superior un formulario con los datos básicos del perfil de usuario. Estos campos podrán ser modificados y posteriormente actualizados pulsando el botón de actualizar.

Si el usuario autenticado es un doctor aparecerá un formulario adicional en el que el doctor podrán especificar su horario semanal. Este formulario constara de dos partes. Una primera parte que consistirá en una visión semanal de los intervalos ya establecidos. Una segunda parte con un formulario para definir nuevos intervalos.

En la parte de la derecha aparecerán los filtros y el *panel de movilidad*.

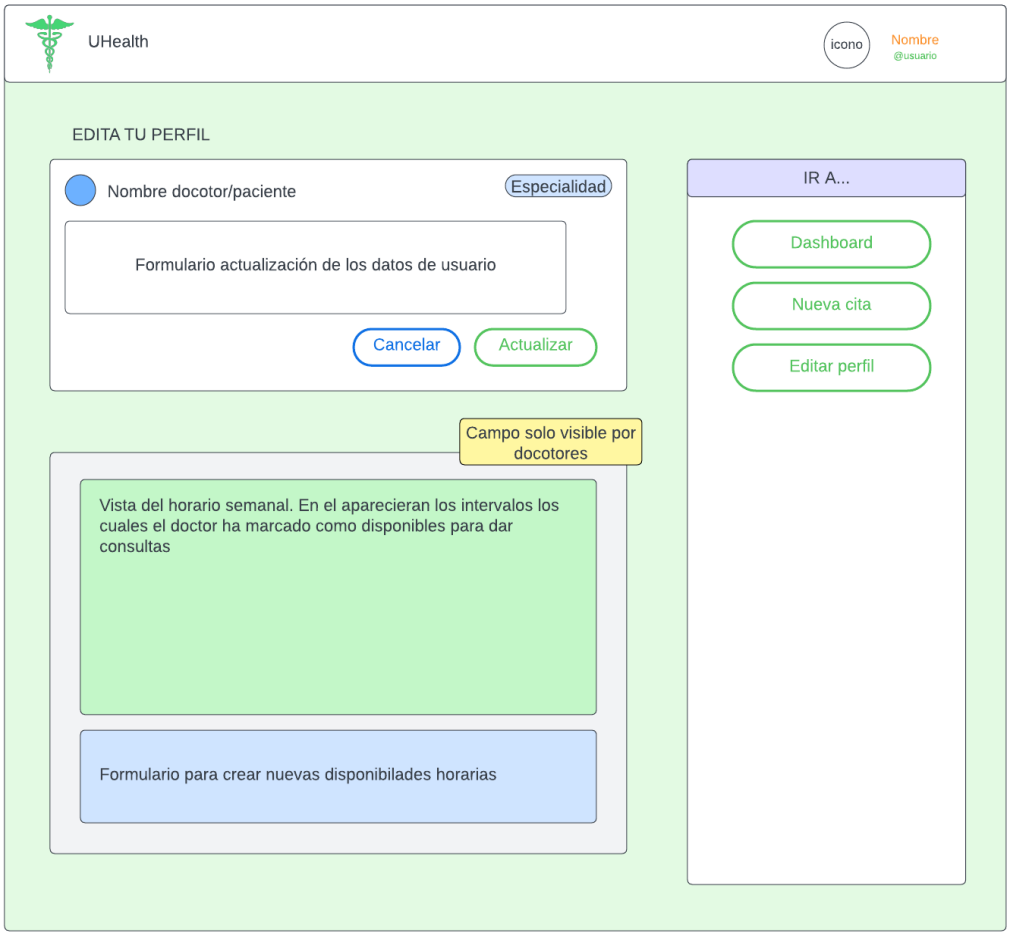

Figura 18. Diseño de la interfaz para la edición del perfil.

3.5.6 Citas

Esta interfaz muestra información sobre una cita en concreto. En la parte superior aparece el nombre del doctor o paciente asociado a la consulta (si el usuario autenticado es el doctor, aparecerá el paciente y viceversa), información básica sobre la consulta como la hora o la duración y un botón para unirse a la videoconferencia asociada a esta.

Si el usuario autenticado es un doctor y mas concretamente es el doctor asociado a esta consulta. Aparecerá una sección en la que podrá tomar notas de consulta sobre los pacientes.

En la parte de la derecha aparecerán los filtros y el *panel de movilidad*.

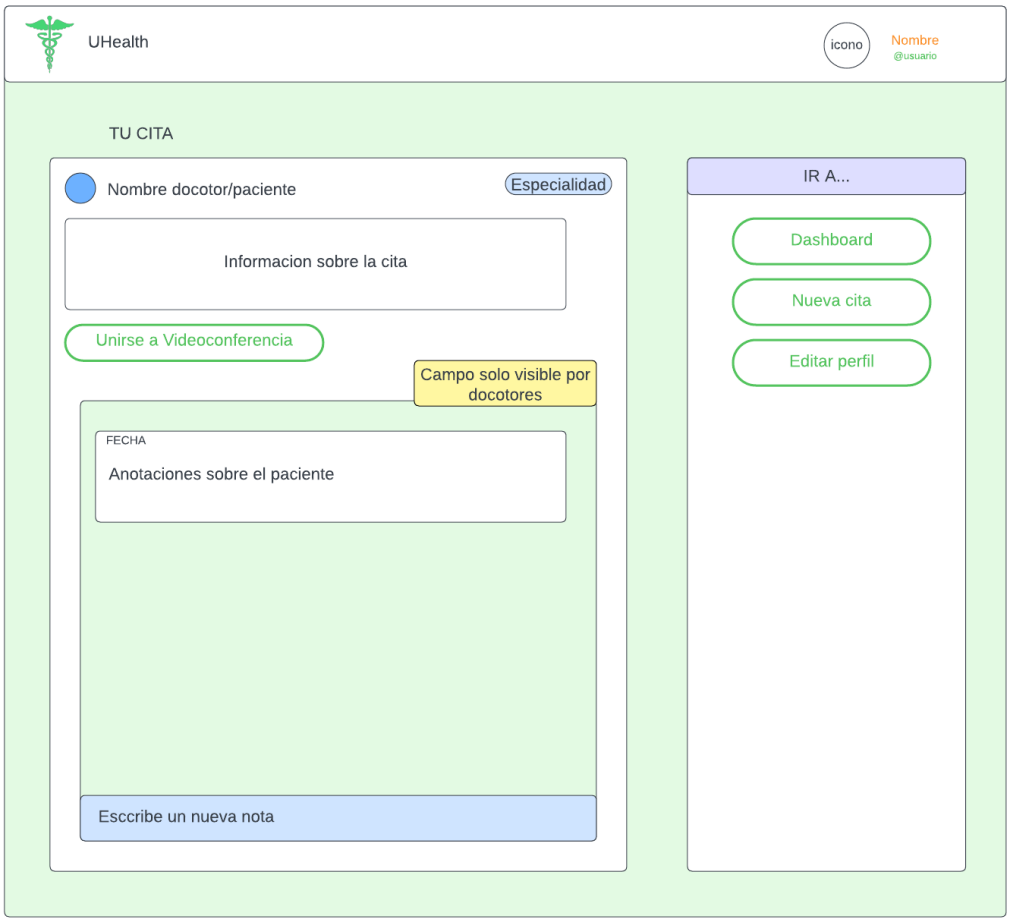

Figura 19. Diseño de la interfaz de las citas.

3.5.7 Videoconferencias

Se trata de una simple interfaz donde se realizarán las videoconferencias. Paciente y doctor aparecerán a la derecha e izquierda de la ventana respectivamente.

En la parte inferior se dispondrán tres simples botones que controlarán la videoconferencia. Estos botones son: activar/desactivar cámara; activar/desactivar micrófono; y colgar.

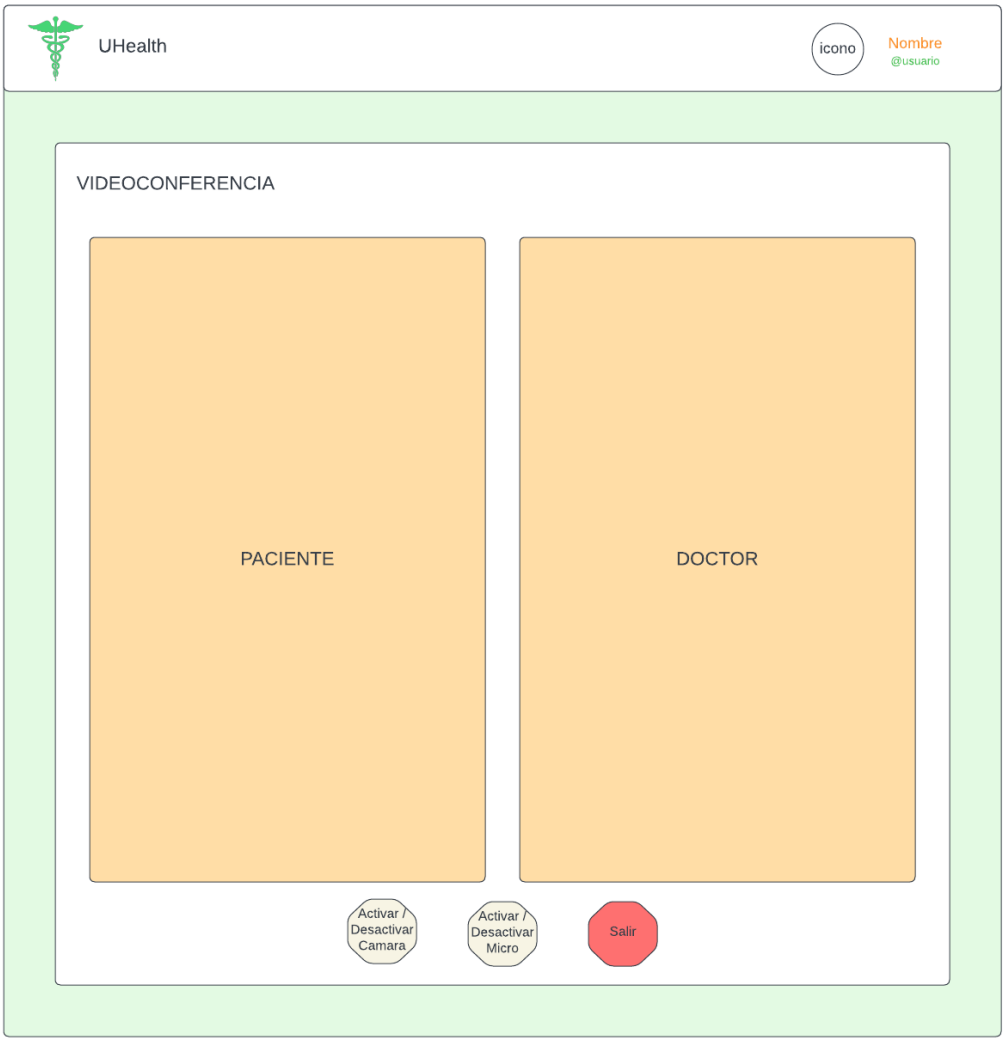

Figura 20. Diseño de la interfaz de la videoconferencia.

3.5.8 Formularios de sesión

Esta interfaz será la encargada de mostrar los formularios para el inicio de sesión y el registro de tanto pacientes como doctores.

Gracias a Django podremos implementar estas tres interfaces empleando una única plantilla, solo deberemos el formulario que se pasara como contexto.

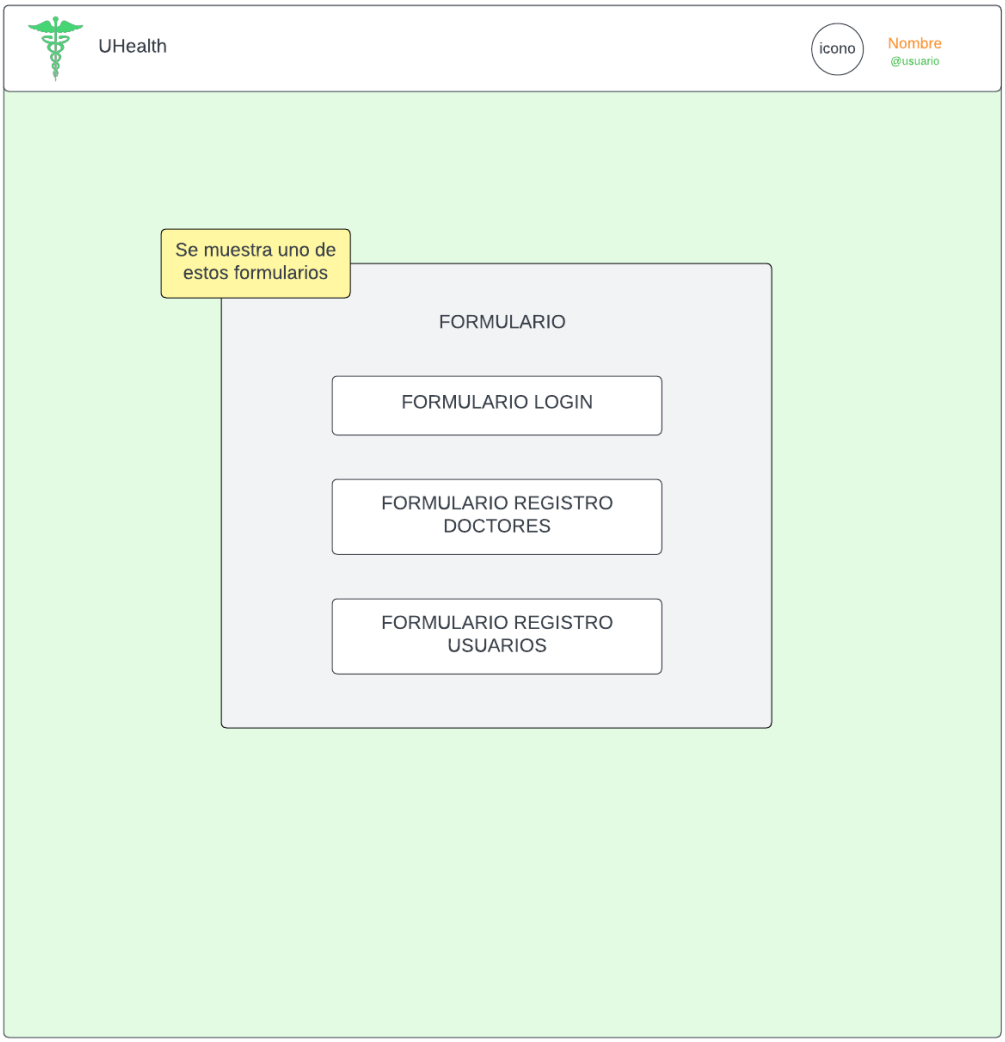

Figura 21. Diseño de la interfaz con los formularios de inicio de sesión y registro.

3.6 Flujo de la aplicación

En el siguiente diagrama se muestra el flujo general de la aplicación. En él se muestran cuáles son las opciones que tanto pacientes como doctores pueden tomar para navegar entre las distintas interfaces. En el diagrama se si las interfaces también se indica si el usuario ha de estar autenticado o no para acceder a la interfaz.

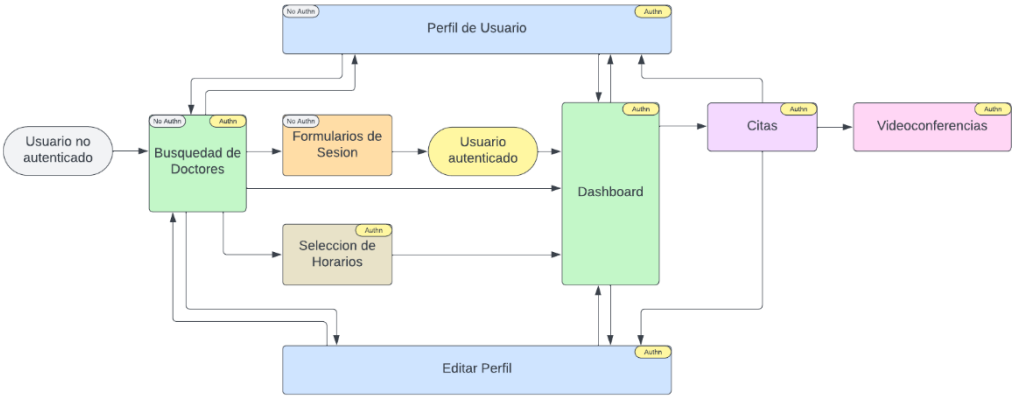

Figura 22. Flujograma general de un usuario en la aplicación.

Capítulo 4. Desarrollo

En este capítulo se detalla la implementación de la aplicación web diseñada en capítulos anteriores. En este capítulo donde se materializarán todos los conceptos explicados con anterioridad. Este apartado se estructura con el propósito de proporcionar una visión integral del proceso de desarrollo, abordando aspectos cruciales.

Como ya he mencionado anteriormente este proyecto se utilizará utilizando el framework de Django.

4.1 Configuración del Entorno:

Indudablemente, esta fase del proceso conlleva la configuración meticulosa del entorno, un requisito fundamental para iniciar el desarrollo con Django. A continuación, se proporciona una explicación minuciosa de esta etapa:

4.1.1 Instalación de Python:

Dado que Django es un marco web diseñado para Python, el primer paso implica asegurarse de tener Python instalado en el sistema. La versión más reciente de Python puede descargarse del sitio oficial. Para este proyecto estaremos usando la versión 3.11.5

4.1.2 Configuración del Entorno Virtual

El siguiente paso consiste en la instalación de un entorno virtual. Este entorno virtual, esencial para evitar conflictos potenciales, sirve para aislar las dependencias del proyecto del entorno global del sistema. La creación de un entorno virtual se realiza mediante el siguiente comando:

```
$ python -m venv env
```

El anterior comando creará un carpeta en el directorio actual con el nombre *env* y donde se almacenará el entorno virtual.

A continuación, hemos de activar este entorno. Esta activación se logra de la siguiente manera:

```
$ env\Scripts\activate
```

Con esto cualquier comando ejecutado en la terminal se estará ejecutando en el marco del entorno virtual.

4.1.3 Instalación de Django

Una vez instalado Python y dentro de nuestro entorno virtual, Django puede incorporarse mediante el administrador de paquetes de Python, conocido como pip. Este proceso se inicia mediante la ejecución del siguiente comando desde la terminal o la línea de comandos:

```
$ pip install django
```

Este comando facilita la instalación de la versión más reciente y estable de Django en el sistema. Para nuestro caso estaremos utilizando la versión 4.2

4.1.4 Inicio de un Proyecto Django

Con la última versión instalada en el sistema, se procede a iniciar un nuevo proyecto Django. Para ello, se navega a la ubicación deseada para el proyecto y se ejecuta el siguiente comando:

```
$ django-admin startproject project
```

Este comando genera un directorio con el nombre del proyecto y establece la estructura básica correspondiente a un proyecto Django. Para nuestro caso el nombre del proyecto es “project”.

Ahora si ejecutamos el siguiente comando podremos iniciar el servidor de desarrollo, permitiendo visualizar la aplicación en el navegador mediante la dirección `http://127.0.0.1:8000/`.

```
$ python manage.py runserver
```

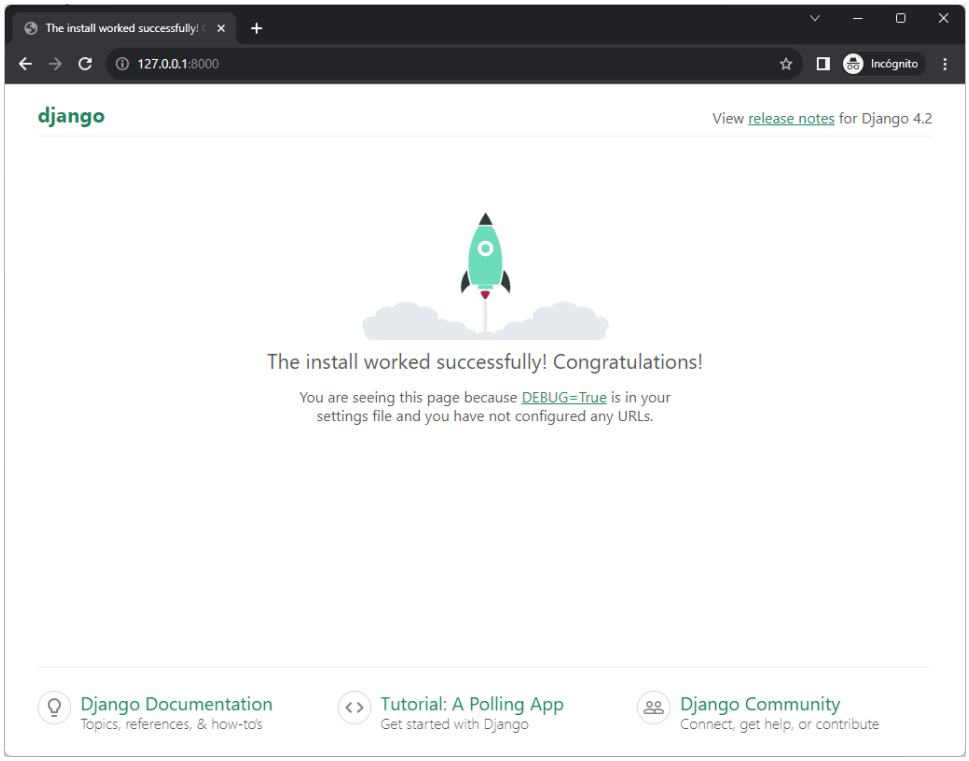

Figura 23. Página de inicio de proyecto de Django.

4.1.5 Creación de la aplicación base

La arquitectura de Django organiza la funcionalidad en aplicaciones para fomentar el modularidad, la reutilización de código y una estructura organizada en proyectos web. En nuestro caso, al tratarse de un proyecto pequeño, implementaremos las distintas funcionalidades dentro de la misma aplicación. La creación de una aplicación dentro del proyecto se realiza ejecutando el siguiente comando:

```
python manage.py startapp base
```

Este comando genera una estructura básica de archivos destinada a la aplicación dentro del proyecto.

4.2 Estructura del Proyecto:

La implementación de la aplicación WEB se ralizara mediante un proyecto de Django llamado "project" y una única aplicación principal llamada "base". Durante esta fase se emplearan la base de datos y el servidor web de testeo proporcionado por el framework Django.

El framework de Django presenta una estructura fija que ha de respetarse en la medida de lo posible en sus proyectos. En esta estructura, cada archivo y directorio cumple un papel específico en el marco de desarrollo. A continuación, se ofrece una explicación formal de cada uno de ellos:

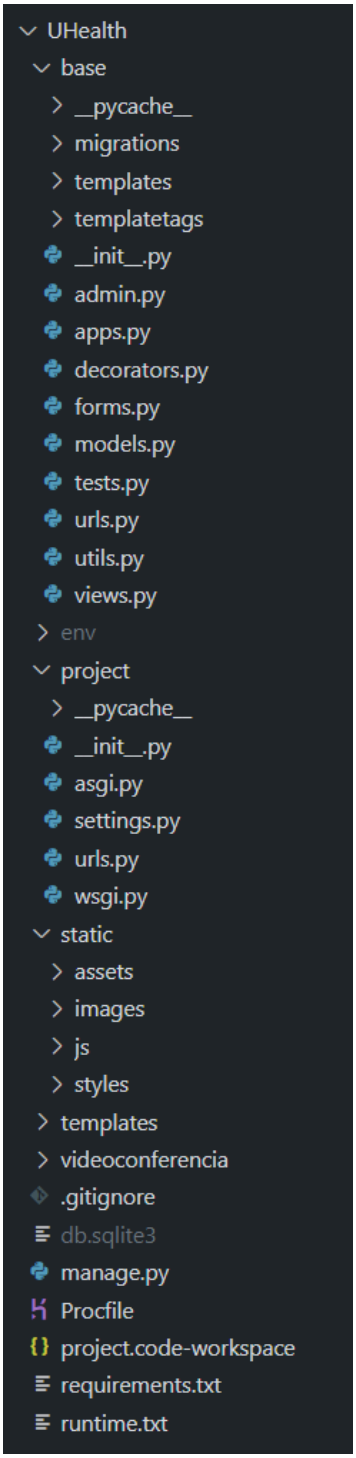

Figura 24. Captura de pantalla realizada a la estructura de carpetas y archivos del proyecto.

La carpeta “base” contiene todos los archivos de la aplicación base en la cual se desarrollado toda la aplicación.

La carpeta “migrations/” almacena las migraciones de la base de datos generadas por el comando “makemigrations”

La carpeta “templates.py” contiene las plantillas de aplicación base que especifican el diseño de las distintas interfaces de la aplicación.

En el fichero “admin.py” se registran los modelos de la aplicación para ser administrados a través de la interfaz de administración de Django.

El fichero “apps.py” se utiliza para configurar la aplicación y proporcionar información adicional sobre ella.

El fichero “decorators.py” contiene definiciones de decoradores específicos de esta aplicación.

En el fichero “models.py” contiene las definiciones de los modelos de la aplicación, que representan las tablas de la base de datos.

En el archivo “tests.py” se pueden escribir pruebas para verificar la funcionalidad de la aplicación. (No se empleará)

El archivo “url.py” define las rutas URL del proyecto, especificando cómo se manejarán las solicitudes entrantes.

En el fichero “utils.py” se definirán todas aquellas funciones que deseemos y que no forman parte de framework.

En el archivo “views.py” se definen las vistas que gestionan las solicitudes del usuario y controlan la lógica de la aplicación.

La carpeta “env.py” contiene todos los ficheros y configuraciones del entorno virtual de Python

La carpeta “project/” contiene la configuración del proyecto.

El archivo “asgi.py” es específico para proyectos que utilizan el estándar ASGI (Asynchronous Server Gateway Interface). Define la aplicación ASGI para el proyecto.

En el archivo “settings.py” se configuran las configuraciones del proyecto, como la base de datos, las aplicaciones instaladas y las claves secretas.

El archivo “urls.py” define las rutas URL del proyecto, especificando cómo se manejarán las solicitudes entrantes. Se importarán las URL de la aplicación base.

El archivo “wsgi.py” se utiliza para configurar el servidor WSGI (Web Server Gateway Interface) que sirve la aplicación en un entorno de producción.

La carpeta “statics/” contiene todos aquellos ficheros estáticos del proyecto. Tanto las plantillas de Django comunes para todas las aplicaciones como las imágenes, scripts en JavaScript y archivos de definición de estilos CSS.

El fichero “db.sqlite3” representa la base de datos SQLite predeterminada utilizada por Django en el entorno de desarrollo. Almacena los datos de la aplicación.

El script “manage.py” se utiliza para realizar diversas tareas administrativas relacionadas con el proyecto, como la ejecución de servidores, la creación de migraciones y la gestión de la base de datos.

En el fichero “Procfile” Especifica los comandos que deben ejecutarse para iniciar la aplicación en Heroku. Define los procesos y la forma en que deben ser ejecutados.

El fichero “requirements.txt” enumera todas las dependencias (bibliotecas y versiones) que la aplicación necesita para ejecutarse correctamente en python.

El fichero “runtime.txt” especifica la versión del entorno de ejecución. En el contexto de Heroku, se utiliza para indicar la versión de Python que se debe utilizar.

Cada uno de estos archivos y directorios desempeña un papel crucial en la estructura y funcionalidad general de un proyecto Django, siguiendo el paradigma del diseño de Django y permitiendo un desarrollo modular y organizado.

4.3 Proyecto en Django

Como ya se ha mencionado anteriormente, todo el desarrollo se realizará en una única aplicación de Django. Se ha decidido trabajar de esta manera mantener la simplicidad del proyecto.

4.3.1 Modelos

Django permite la ceración de modelos, los cuales definen la estructura de los datos en la aplicación. Cada modelo en Django corresponde con una única tabla en la base de datos. En este contexto, se destaca que cada modelo se configura como una clase en Python que hereda de la clase *django.db.models.Model*, y cada atributo de dicho modelo representa un campo en la base de datos. Este enfoque enriquece la funcionalidad de Django al proveer una interfaz de acceso a la base de datos generada automáticamente. Se pasa a comentar cada uno de los modelos desarrollados para la aplicación [3]

4.3.1.1 Usuario

Este modelo extiende la clase *AbstractUser* de Django. Además de los campos comunes de un usuario (como nombre de usuario y contraseña), se añaden campos personalizados, como email, que debe ser único. También se introducen los campos booleanos *is_doctor* e *is_patient* para identificar si el usuario tiene el rol de doctor o paciente, respectivamente.

```
class User(AbstractUser):
    email = models.EmailField(unique=True)
    is_doctor = models.BooleanField(default=False)
    is_patient = models.BooleanField(default=False)
```

4.3.1.2 Especialidad

Este modelo representa las diversas especialidades médicas disponibles. Cada instancia tiene un campo *name* para almacenar el nombre de la especialidad.

```
class Speciality(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name
```

4.3.1.3 Doctor

Este modelo establece una relación uno a uno con el modelo *User*, representando así a un usuario con el rol de doctor. Incluye campos adicionales como *first_name* y *last_name* para el nombre del doctor, así como una relación de muchos a muchos con el modelo *Speciality* para indicar las especialidades médicas asociadas al doctor. El campo *about_me* permite proporcionar información adicional sobre el doctor.

```
class Doctor(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE,
primary_key=True, related_name='doctor')
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    specialities = models.ManyToManyField(Speciality)
    about_me = models.TextField()

    def __str__(self):
        return self.user.username
```

4.3.1.4 Paciente

Similar al modelo Doctor, este modelo establece una relación uno a uno con el modelo User y representa a un usuario con el rol de paciente. Incluye campos básicos como *first_name*, *last_name* y *date_of_birth* para

```
class Patient(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE,
primary_key=True, related_name='patient')
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    date_of_birth = models.DateField()

    def __str__(self):
        return self.user.username
```

4.3.1.5 Intervalo de disponibilidad

Este modelo gestiona la disponibilidad horaria de un doctor durante la semana. Almacena información como el día de la semana, la hora de inicio y fin de la disponibilidad, así como la duración de las citas médicas en minutos.

```
class DoctorAvailability(models.Model):
    doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE)
    day_of_week = models.IntegerField(choices=[
        (0, 'Monday'),
        (1, 'Tuesday'),
        (2, 'Wednesday'),
        (3, 'Thursday'),
        (4, 'Friday'),
        (5, 'Saturday'),
        (6, 'Sunday'),
    ])
    start_time = models.TimeField()
    end_time = models.TimeField()
    appointment_duration = models.PositiveIntegerField()

    def __str__(self):
```

```
        return f"{self.doctor.user.username}'s Availability on
{self.get_day_of_week_display()}"
```

4.3.1.6 Citas

Representa una cita médica y establece relaciones con los modelos *Doctor*, *DoctorAvailability*, *Patient*, y *Speciality*. Incluye información como la hora de inicio de la cita, el estado de la cita (pendiente, confirmada, completada o cancelada) y la especialidad asociada.

```
class Appointment(models.Model):
    doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE)
    doctor_availability = models.ForeignKey(DoctorAvailability,
on_delete=models.CASCADE)
    patient = models.ForeignKey(Patient, on_delete=models.CASCADE)
    speciality = models.ForeignKey(Speciality, on_delete=models.CASCADE)
    start_time = models.DateTimeField()
    STATUS_CHOICES = (
        ('Pending', 'Pending'),
        ('Confirmed', 'Confirmed'),
        ('Completed', 'Completed'),
        ('Canceled', 'Canceled'),
    )
    status = models.CharField(max_length=20, choices=STATUS_CHOICES)
```

4.3.1.7 Anotaciones

Este modelo permite a los doctores agregar notas relacionadas con una cita médica específica. Almacena el contenido de la nota y la fecha de creación. Está vinculado a la cita médica a través de la relación con el modelo *Appointment*.

```
class DoctorNote(models.Model):
    appointment =
models.ForeignKey(Appointment,on_delete=models.CASCADE,related_name='doctor_n
otes')
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Doctor Note for Appointment {self.appointment.id}"
```

4.3.2 Formularios

Los formularios de Django proporcionan una forma elevada de manejar la entrada y validación de datos en el lado del servidor. Se dividen principalmente en dos tipos: "*forms.Form*" para el manejo de formularios de propósito general y "*forms.ModelForm*" para interactuar con modelos de bases de datos. Estos formularios simplifican el proceso de recopilación y procesamiento de la entrada del usuario, encargándose de tareas como la validación de datos, la presentación y el almacenamiento.

Los formularios de Django permiten definir su estructura, personalizar el comportamiento de los campos e integrarlos fácilmente en las vistas para crear aplicaciones web interactivas. [14]

4.3.2.1 Actualizar Paciente

Este formulario utiliza la clase *ModelForm* de Django y está vinculado al modelo *User*. Define los campos a mostrar en el formulario, que son "*username*" y "*email*". Este formulario se utiliza para la actualización de información básica del paciente.

```
class UserForm(ModelForm):
    class Meta:
        model = User
        fields = ['username', 'email']
```

4.3.2.2 Actualizar Doctor

Similar al *UserForm*, este formulario está vinculado al modelo Doctor y define los campos que se pueden actualizar, como "*first_name*", "*last_name*" y "*about_me*". Se utiliza para la actualización de información específica de los doctores.

```
class UpdateDoctorForm(ModelForm):
    class Meta:
        model = Doctor
        fields = ['first_name', 'last_name', 'about_me']
```

4.3.2.3 Registro de pacientes

Este formulario se utiliza para el registro de nuevos pacientes. Hereda de *UserCreationForm*, que es proporcionado por Django para la creación de usuarios. Agrega campos adicionales como "*first_name*", "*last_name*" y "*date_of_birth*" para recopilar información específica del paciente. Incluye lógica adicional para crear tanto el usuario como el paciente asociado.

```
class PatientRegisterForm(UserCreationForm):
    email = forms.EmailField(widget=forms.EmailInput())
    password1 = forms.CharField(widget=forms.PasswordInput())
    password2 = forms.CharField(widget=forms.PasswordInput())

    first_name = forms.CharField(widget=forms.TextInput())
    last_name = forms.CharField(widget=forms.TextInput())
    date_of_birth = forms.DateField(
        widget=forms.DateInput(attrs={'type': 'date'}),
        label='Select a Date of Birth'
    )

    class Meta(UserCreationForm.Meta):
        model = User
        fields = ('username', 'email', 'password1', 'password2')

    @transaction.atomic
    def save(self, commit=True):
        user = super().save(commit=False)
        user.is_patient = True
        if commit:
            user.save()
        patient = Patient.objects.create(user=user,
             first_name=self.cleaned_data.get('first_name'),
             last_name=self.cleaned_data.get('last_name'),
             date_of_birth=self.cleaned_data.get('date_of_birth'))
        return user
```

4.3.2.4 Registro de doctores

Similar al formulario anterior, este formulario se utiliza para el registro de nuevos doctores. También hereda de *UserCreationForm* y agrega campos específicos para doctores como "*first_name*", "*last_name*" y "*specialities*" (lista de especialidades). Al igual que el formulario de paciente, incluye lógica adicional para crear tanto el usuario como el doctor asociado. Por ejemplo, en este se formulario se especifica que el doctor estará desactivado por defecto y deberá ser activado por un administrador.

```
class DoctorRegisterForm(UserCreationForm):
    email = forms.EmailField(widget=forms.EmailInput())
    password1 = forms.CharField(widget=forms.PasswordInput())
    password2 = forms.CharField(widget=forms.PasswordInput())

    first_name = forms.CharField(widget=forms.TextInput())
    last_name = forms.CharField(widget=forms.TextInput())
    specialities = forms.ModelMultipleChoiceField(
        queryset=Speciality.objects.all(),
        widget=forms.CheckboxSelectMultiple,
    )

    class Meta(UserCreationForm.Meta):
        model = User
        fields = ('username', 'email', 'password1', 'password2')

    @transaction.atomic
    def save(self, commit=True):
        user = super().save(commit=False)
        user.is_doctor = True
        user.is_active = False
        if commit:
            user.save()
        doctor = Doctor.objects.create(
            user=user,
            first_name=self.cleaned_data.get('first_name'),
            last_name=self.cleaned_data.get('last_name'),
        )
        doctor.specialities.set(self.cleaned_data.get('specialities'))
        return user
```

4.3.2.5 Inicio de session

Este formulario se utiliza para la autenticación de usuarios y hereda de *AuthenticationForm* proporcionado por Django. Personaliza los widgets para los campos de "*username*" y "*password*", añadiendo *placeholders*.

```
class LoginForm(AuthenticationForm):
    username = forms.CharField(
widget=forms.TextInput(attrs={'placeholder':'Enter your username'}))
    password =
forms.CharField(widget=forms.PasswordInput(attrs={'placeholder':'•••••••••'})
)
```

4.3.2.6 Formulario para la creación de intervalos de disponibilidad

Este formulario está vinculado al modelo *DoctorAvailability* y se utiliza para ingresar la disponibilidad horaria de un doctor. Define campos como "*day_of_week*", "*start_time*", "*end_time*" y "*appointment_duration*". Además, incluye lógica adicional en el método *clean*() para verificar que no haya solapamientos de intervalos de disponibilidad para un mismo doctor en un mismo día.

```
class DoctorAvailabilityForm(forms.ModelForm):

    class Meta:
        model = DoctorAvailability
        fields = ['doctor','day_of_week', 'start_time', 'end_time',
'appointment_duration']

        widgets = {
            'doctor': forms.HiddenInput(),
            'day_of_week': forms.HiddenInput(),

            'start_time': forms.TimeInput(
                attrs={'type': 'time', 'class': 'form-control'},
                format='%H:%M'
            ),
            'end_time': forms.TimeInput(
                attrs={'type': 'time', 'class': 'form-control'},
                format='%H:%M'
            ),
            'appointment_duration': forms.NumberInput(
                attrs={'min': 15, 'step': 5}
            ),
        }

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

    def clean(self):
        cleaned_data = super().clean()
        doctor = cleaned_data.get('doctor')
        day_of_week = cleaned_data.get('day_of_week')
        start_time = cleaned_data.get('start_time')
        end_time = cleaned_data.get('end_time')
        appointment_duration = cleaned_data.get('appointment_duration')

        # Perform interval overlap check for the same doctor on the same day
        overlapping_availabilities = DoctorAvailability.objects.filter(
            doctor=doctor,
            day_of_week=day_of_week,
            start_time__lt=end_time,
            end_time__gt=start_time
        )

        if overlapping_availabilities.exists():
            raise forms.ValidationError('There is already an overlapping time
slot for this doctor on the same day.')

        return cleaned_data
```

Además de todos estos formularios, existen otros que han sido definidos directamente en las plantillas por simplicidad. Estos formularios has sido definidos empleando la sintaxis HTML básica.

4.3.3 Plantillas

Las plantillas son una parte esencial del framework Django ya que facilita la presentación de datos en las aplicaciones web. El sistema de plantillas de Django sigue el principio de separación entre la lógica de la aplicación y la presentación, permitiendo que los diseñadores y desarrolladores trabajen de manera eficiente en sus respectivas áreas.

En Django, una plantilla es esencialmente un documento HTML con marcadores de posición que se llenan con datos específicos cuando la página se renderiza. Estos marcadores de posición se conocen como "variables", "etiquetas" y "filtros".

Las variables son delimitadas por "{{ }}" y sirven para imprimir valores del "contexto" de la vista.

Las etiquetas "{% %}" proporcionan lógica a la plantilla, por ejemplo, una plantilla puede imprimir contenido o servir como una estructura de control actuando como una condición o un bucle.

Por último, los filtros, sirven para transformar los valores de las variables y etiquetas y se de finen de la siguiente manera "|".

La asignación de los valores a renderizar se realiza desde la definición de la vista. Al renderizar una vista, se proporciona un "contexto". El contexto es un diccionario de datos que contiene la información que se insertará en la plantilla. Cuando una vista de Django es llamada, la plantilla asociada se carga y se renderiza. Durante el renderizado, las etiquetas y filtros son evaluados y reemplazados por los datos correspondientes del contexto. [16]

Para una mejor comprensión se propone el siguiente ejemplo:

```
VISTAS

def vistaEjemplo(request):
    user_data = {'nombre': 'Juan'}
    return render(request, 'mi_plantilla.html', {'usuario': user_data})


PLANTILLAS

<h1>Bienvenido, {{ ususario.username }}!</h1>
```

En este caso, el valor de "usuario.nombre" en la plantilla será "Juan". Como se puede observar, este valor es guardado en el contexto de la vista y es renderizado utilizando una variable.

Django también admite la herencia de plantillas, lo que permite definir plantillas base con elementos comunes y luego extenderlas en plantillas más específicas.

Esta capacidad se ha empleado para la realización del proyecto. Todas las interfaces de la aplicación comparten un aparte común, la barra superior. Esta parte común ha sido definida en una plantilla de la cual se extienden el resto de las plantillas de la aplicación.

Todas las interfaces diseñadas en la fase de diseño has sido implementadas utilizando plantillas, a continuación, se explicará el desarrollo de una de ellas para que sirva como ejemplo para el resto.

```
{% extends 'main.html' %}
{% load static %}

{% block content %}
<main class="layout layout--2">
 <div class="container">

  <!-- Room List Start -->
  <div class="roomList">

   <div class="roomList__header">
    <div>
     <h2>Next appointments</h2>
     <p>You have {{next_appointments.count}} appointments</p>
    </div>
   </div>

   {% for appointment in next_appointments %}
   <div class="roomListRoom">

    <div class="roomListRoom__header">
     <a href="{% url 'user-profile' appointment.doctor.user.id %}" class="roomListRoom__author">
      <div class="avatar avatar--small">
       <img src="{%static 'images/avatars/avatar' %}{{appointment.doctor.user.avatar_id}}.svg"/>
      </div>
      <span>{{appointment.doctor.first_name}} {{appointment.doctor.last_name}}</span>
     </a>
     <div>
      <p class="roomListRoom__topic">{{appointment.speciality.name}}</p>
     </div>
    </div>
    <div class="roomListRoom__content">
    </div>
    <div class="roomListRoom__meta">
    </div>

    <a class="roomListRoom__joined">
     <b>Starts at:</b>
     {{appointment.start_time}}
    </a>
    <a class="roomListRoom__joined">
     <b>Duration:</b>
     {{appointment.doctor_availability.appointment_duration}}
    </a>

    <div class="roomListRoom__meta">
     <p class="roomListRoom__topic">{{appointment.status}}</p>
     <a href="{% url 'appointment' appointment.id %}">Go to appointment page</a>
    </div>

   </div>
   {% endfor %}

   <div class="roomList__header">
    <div>
     <h2>Past appointments</h2>
    </div>
   </div>

   {% for appointment in past_appointments %}
   <div class="roomListRoom">


    <div class="roomListRoom__header">
     {% if request.user.is_doctor%}
     <a href="{% url 'user-profile' appointment.patient.user.id %}"
class="roomListRoom__author">
      <div class="avatar avatar--small">
```

```
       <img src="{% static 'images/avatars/avatar' %}{{ appointment.patient.user.avatar_id
}}.svg" />
      </div>
      <span>{{appointment.patient.first_name}} {{appointment.patient.last_name}}</span>
     </a>
     {% elif request.user.is_patient%}
     <a href="{% url 'user-profile' appointment.doctor.user.id %}" class="roomListRoom__author">
      <div class="avatar avatar--small">
       <img src="{% static 'images/avatars/avatar' %}{{ appointment.doctor.user.avatar_id
}}.svg" />
      </div>
      <span>{{appointment.doctor.first_name}} {{appointment.doctor.last_name}}</span>
     </a>
     {% endif%}
     <div>
      <p class="roomListRoom__topic">{{appointment.speciality.name}}</p>
     </div>
    </div>
    <div class="roomListRoom__content">
    </div>
    <div class="roomListRoom__meta">
    </div>

    <a class="roomListRoom__joined">
     <b>Starts at:</b>
     {{appointment.start_time}}
    </a>
    <a class="roomListRoom__joined">
     <b>Duration:</b>
     {{appointment.doctor_availability.appointment_duration}}
    </a>

    <div class="roomListRoom__meta">
     <p class="roomListRoom__topic">{{appointment.status}}</p>
     <a href="{% url 'appointment' appointment.id %}">Go to appointment page</a>
    </div>

   </div>
   {% endfor %}
  </div>


  {% include 'base/component_activities.html' %}

</main>
{% endblock %}
```

{% extends 'main.html' %} indica que esta plantilla extiende la plantilla base llamada `main.html`. Esto significa que hereda el diseño y la estructura de `main.html`, y luego puedes personalizar el contenido específico de esta plantilla.

{% load static %} carga los archivos estáticos en la plantilla, permitiendo el uso de etiquetas como *{% static 'ruta/al/archivo.css' %}* para incluir archivos estáticos como hojas de estilo CSS.

{% block content %} y *{% endblock %}* definen un bloque llamado “content”. Los bloques en Django se utilizan para permitir que las plantillas secundarias (que extienden esta plantilla) proporcionen su propio contenido en ese espacio.

Dentro del bloque content se diferencian varias secciones:

- Sección de próximas citas:

  Muestra la cantidad de próximas citas (*{{next_appointments.count}}*) y la información de cada cita en un bucle *{% for appointment in next_appointments %}*.

  Muestra detalles como el médico, la especialidad, la hora de inicio, la duración y el estado de cada cita.

- Sección de citas pasadas:

  Muestra la información de las citas pasadas en un bucle *{% for appointment in past_appointments %}*.

  Dependiendo de si el usuario actual es un médico o un paciente, muestra la información del paciente o del médico respectivamente.

- *{% include 'base/component_activities.html' %}*: Incluye otro archivo de plantilla llamado "component_activities.html" que probablemente contenga componentes o información adicional.

A continuación, se comenta como han sido desarrolladas todas y cada una de las interfaces diseñadas para este proyecto:

4.3.3.1 Dashboard

- *dashboard.html*: constituye la interfaz general, en ella se definen la sección de citas siguientes y pasadas y se importa el panel de movilidad.
- *component_activities.html:* Define el panel de movilidad.

4.3.3.2 Búsqueda de doctores

- *home.html:* Define la estructura general e importa el resto de los componentes.
- *home_component_doctors.html:* Este componente lista todos los doctores pasados en el contexto de la vista, los cuales habrán sido filtrados en esta.
- *home_component_search.html:* Muestra el formulario con los filtros.
- *component_activities.html:* Define el panel de movilidad.

4.3.3.3 Selección del horario de consulta

- *appointment_form.html:* Define la estructura principal de la interfaz. Importa el componente del calendario.
- *component_calendar.html:* Componente con el calendario. Importa la librería de *Full Calendar* y configura y despliega el calendario para seleccionar la cita deseada.

4.3.3.4 Perfil

- *profile.html:* Define todos y cada uno de los componentes de la vista de perfil e importa el panel de movilidad.
- *component_activities.html:* Define el panel de movilidad.

4.3.3.5 Editar perfil

- *update_user.html:* Contiene la estructura general de la interfaz. Importa el componente del selector de disponibilidad y lo muestra solo si el usuario es un doctor. Además, importa el panel de movilidad.
- *component_availability_selector.html:* Define el componente con el selector de movilidad.
- *component_activities.html:* Define el panel de movilidad.

4.3.3.6 Citas

- *appointment.html:* Define la interfaz le la cita e importa el panel de movilidad. En esta plantilla, la sección para la creación de las notas de los doctores solo será visible si el usuario es un doctor.
- *component_activities.html:* Define el panel de movilidad.

4.3.3.7 Videoconferencias

- *room.html:* Define la interfaz de la videoconferencia, importa la librería de *agora.io* y define los botones para la interacción con esta.

4.3.3.8 Formularios de sesión

- *login_register.html*: Define la estructura de los formularios de inicio de sesión y registro de pacientes y doctores.

4.3.4 Decoradores

En Django, un decorador es una función especial que se utiliza para modificar el comportamiento de otra función o método. En el contexto de las vistas de Django, los decoradores son comúnmente empleados para añadir funcionalidades como restricciones de acceso, almacenamiento en caché o registro de información. Por ejemplo, el decorador "*@login_required*" se utiliza para garantizar que solo los usuarios autenticados puedan acceder a una vista específica. Los decoradores permiten controlar el acceso a las vistas y encapsular comportamientos específicos de manera modular y reutilizable, mejorando la funcionalidad de las vistas en Django.

Para este proyecto de han definido dos decoradores, ambos tienen la finalidad de comprobar si el usuario que intenta a ceder a la vista es un paciente o un doctor. [17]

4.3.4.1 Paciente requerido

Este decorador verifica que el usuario que intenta acceder a la vista sea un paciente activo. Utiliza el decorador más genérico *user_passes_test* proporcionado por Django, que evalúa una función dada para determinar si se permite o no el acceso. En este caso, la función lambda verifica si el usuario está activo y es un paciente. Si el usuario no cumple con estos requisitos, se redirige a la página de inicio de sesión.

```
def patient_required(function=None, redirect_field_name=REDIRECT_FIELD_NAME,
login_url='login'):
    '''
    Decorator for views that checks that the logged in user is a patient,
    redirects to the log-in page if necessary.
    '''
    actual_decorator = user_passes_test(
        lambda u: u.is_active and u.is_patient,
        login_url=login_url,
        redirect_field_name=redirect_field_name
    )
    if function:
        return actual_decorator(function)
    return actual_decorator
```

4.3.4.2 Doctor requerido

Similar al decorador para pacientes, este decorador verifica que el usuario que intenta acceder a la vista sea un doctor activo. También utiliza el decorador *user_passes_test* con una función lambda que evalúa si el usuario está activo y es un doctor. Al igual que en el caso anterior, si el usuario no cumple con estos requisitos, se redirige a la página de inicio de sesión

```
def doctor_required(function=None, redirect_field_name=REDIRECT_FIELD_NAME,
login_url='login'):
    '''
```

```
    Decorator for views that checks that the logged in user is a doctor,
    redirects to the log-in page if necessary.
    '''
    actual_decorator = user_passes_test(
        lambda u: u.is_active and u.is_doctor,
        login_url=login_url,
        redirect_field_name=redirect_field_name
    )
    if function:
        return actual_decorator(function)
    return actual_decorator
```

4.3.5 Vistas

En Django, una vista es una función de Python que toma una solicitud web y devuelve una respuesta web. Las vistas son responsables de procesar la lógica de negocio y determinar qué información se mostrará al usuario. En el contexto de Django, las vistas se utilizan para manejar las solicitudes HTTP, como las solicitudes de un navegador web.

Una vista en Django es simplemente una función de Python que toma una solicitud HTTP como argumento y devuelve una respuesta HTTP como resultado.

Después de procesar la lógica de negocio, las vistas deben devolver una respuesta HTTP. Puedes utilizar la función “render” para combinar una plantilla con datos y devolver una respuesta completa.

```
from django.shortcuts import render

def mi_vista_con_template(request):
    contexto = {'nombre': 'Usuario'}
    return render(request, 'mi_template.html', contexto)
```

Para implementar todas y cada una de las funciones de la aplicación se han empleado un total de catorce vistas distintas que pasaran a comentarse a continuación. [15]

loginPage:

- Esta vista maneja el inicio de sesión. Muestra un formulario de inicio de sesión (*LoginForm*) en la plantilla *login_register.html*.
- Verifica las credenciales del usuario y redirige al usuario a la página de inicio (*dashboard*) si el inicio de sesión es exitoso.
- Si el usuario ya está autenticado, lo redirige directamente al panel de control (*dashboard*).

logoutUser:

- Esta vista maneja la función de cierre de sesión. Desconecta al usuario y lo redirige a la página de inicio (*home*).

doctorRegisterView:

- Esta vista maneja el registro de nuevos doctores.

- Utiliza un formulario (*DoctorRegisterForm*) para recopilar la información necesaria del usuario.
- Después de un registro exitoso, redirige al usuario a la página de inicio (*home*).

patientRegisterView:

- Similar a *doctorRegisterView*, pero específico para pacientes.
- Utiliza el formulario *PatientRegisterForm*.

home:

- Filtra los doctores según varios criterios, como días disponibles, especialidades, etc.
- Muestra la lista de doctores disponibles en la página principal.
- Carga la interfaz de Búsqueda de doctores.

dashboard:

- Página del panel de control para usuarios autenticados.
- Carga la interfaz del dashboard.
- Muestra los detalles de las citas pasadas y futuras del usuario.
- También proporciona información sobre especialidades y el recuento total de citas.

appointment:

- Muestra detalles sobre una cita específica, incluidas las notas del médico.
- Permite a los usuarios agregar notas.
- Carga la interfaz de las citas

userProfile:

- Muestra el perfil del usuario y, si es un médico, también muestra las notas del médico asociadas a las citas con pacientes.
- Carga la interfaz de los perfiles de usuario.

createAppointment:

- Permite a los pacientes programar citas con médicos.
- Se indican las horas disponibles basándose en la disponibilidad del médico y otros criterios proporcionados en la solicitud.
- Carga la interfaz de la selección del horario de consulta.

createDoctorAvailability, ***updateDoctorAvailability***, ***deleteDoctorAvailability***:

- Vistas que manejan la creación, actualización y eliminación de los intervalos de disponibilidad de los doctores.
- Recoge el formulario para indicar la disponibilidad de los doctores dentro de la interfaz del perfil.

updateUser:

- Permite a los usuarios actualizar su perfil, y a los médicos, su información y disponibilidad.
- Carga la interfaz para la edición del perfil.

chatRoom:

- Vistas relacionadas con la funcionalidad de la sala de chat.
- Carga la interfaz de la videoconferencia.

getToken:

- Vista para la obtención de tokens para la videoconferencia.

4.3.6 URLs

En Django, una URL se refiere a la dirección única de un recurso en la web. En el contexto de un proyecto Django, las URLs se utilizan para mapear las solicitudes del navegador a funciones o vistas específicas en el código Python. Esto quiere decir que a través de ellas se tendrán acceso a los distintos recursos y vistas de la aplicación. [18]

El enrutamiento de URL en Django se define en el archivo urls.py. Este archivo contiene patrones de URL y especifica qué funciones o clases de vista deben manejar las solicitudes que coincidan con esos patrones. El fichero urls.py de nuestra aplicación se ve de la siguiente forma:

```
from django.urls import path
from . import views

from django.contrib.auth import views as auth_views

urlpatterns = [
    path('login/', views.loginPage, name="login"),
    path('logout/', views.logoutUser, name="logout"),
    path('register/doctor/', views.doctorRegisterView.as_view(),
name='doctor-register'),
    path('register/patient/', views.patientRegisterView.as_view(),
name='patient-register'),
    path('', views.home, name="home"),
    path('dashboard/', views.dashboard, name="dashboard"),
    path('appointment/<str:pk>/', views.appointment, name="appointment"),
    path('profile/<str:pk>/', views.userProfile, name="user-profile"),
    path('create-appointment/', views.createAppointment, name="create-
appointment"),
    path('update-user/', views.updateUser, name="update-user"),
    path('create-doctor-availability/', views.createDoctorAvailability,
name="create-doctor-availability"),
    path('update-doctor-availability/<str:pk>',
views.updateDoctorAvailability, name="update-doctor-availability"),
    path('delete-doctor-availability/<str:pk>',
views.deleteDoctorAvailability, name="delete-doctor-availability"),
    path('room/<str:pk>', views.chatRoom, name="room"),
    path('get_token/', views.getToken, name="get_token"),

]
```

Cada patrón relaciona una URL con una vista de la aplicación. El parámetro "*name*" asocia estas vistas con un nombre que podrá ser utilizado para referirse a ellas desde las plantillas.

A continuación, se comentarán cada uno de estos patrones:

Inicio de sesión:
```
path('login/', views.loginPage, name="login")
```

Cierre de sesión:
```
path('logout/', views.logoutUser, name="logout")
```

Registro de doctores:
```
path('register/doctor/', views.doctorRegisterView.as_view(), name='doctor-
register')
```

Registro de usuarios:
```
path('register/patient/', views.patientRegisterView.as_view(), name='patient-
register')
```

Página de búsqueda de doctores:
```
path('', views.home, name="home")
```

Dashboard:
```
path('dashboard/', views.dashboard, name="dashboard")
```

Página de cita, *pk* indica el ID de la cita:
```
path('appointment/<str:pk>/', views.appointment, name="appointment")
```

Visualización de perfiles de usuario, *pk* indica el ID del usuario:
```
path('profile/<str:pk>/', views.userProfile, name="user-profile")
```

Creación de citas:
```
path('create-appointment/', views.createAppointment, name="create-
appointment")
```

Actualización de la información de usuario:
```
path('update-user/', views.updateUser, name="update-user")
```

Creación de la disponibilidad de los doctores:
```
path('create-doctor-availability/', views.createDoctorAvailability,
name="create-doctor-availability")
```

Actualización de la disponibilidad de los doctores, *pk* indica el ID del objeto de disponibilidad:
```
path('update-doctor-availability/<str:pk>', views.updateDoctorAvailability,
name="update-doctor-availability")
```

Eliminación de la disponibilidad de los doctores, *pk* indica el ID del objeto de disponibilidad:
```
path('delete-doctor-availability/<str:pk>', views.deleteDoctorAvailability,
name="delete-doctor-availability")
```

Asociado a la vista para la sala de la videoconferencia, *pk* indicara el ID de la sala:
```
path('room/<str:pk>', views.chatRoom, name="room")
```

Asociado a la vista para obtener un token de autenticación:
```
path('get_token/', views.getToken, name="get_token")
```

4.4 Videoconferencia

Para la realización de las videoconferencias se empleará el servicio de "Video Calling" de *agora.io* que permite conexiones de chat de video individuales o en grupos pequeños con transmisión de video suave y sin fluctuaciones. *agora.io* ofrece un SDK que facilita la incorporación de *Video Calling* en tiempo real en aplicaciones web, móviles y nativas. [7]

Para iniciar una videoconferencia, hay que realizar los siguientes pasos:

1. Obtener un token: El token autentica a un usuario cuando la aplicación se une a un canal. En nuestro proyecto la obtención del token se realiza mediante una petición HTTP a la URL "*get_token/*". Esta URL está asociada con una vista que emplea la librería *agora_token_builder* para la obtención del token.

2. Unirse a un canal: Llama a métodos para crear y unirte a un canal; las aplicaciones que pasan el mismo nombre de canal se unen al mismo canal.

3. Enviar y recibir video y audio en el canal: Todos los usuarios envían y reciben flujos de video y audio de todos los usuarios en el canal.

El siguiente diagrama proporcionado por agora.io muestra una definición más detallada de este flujo.

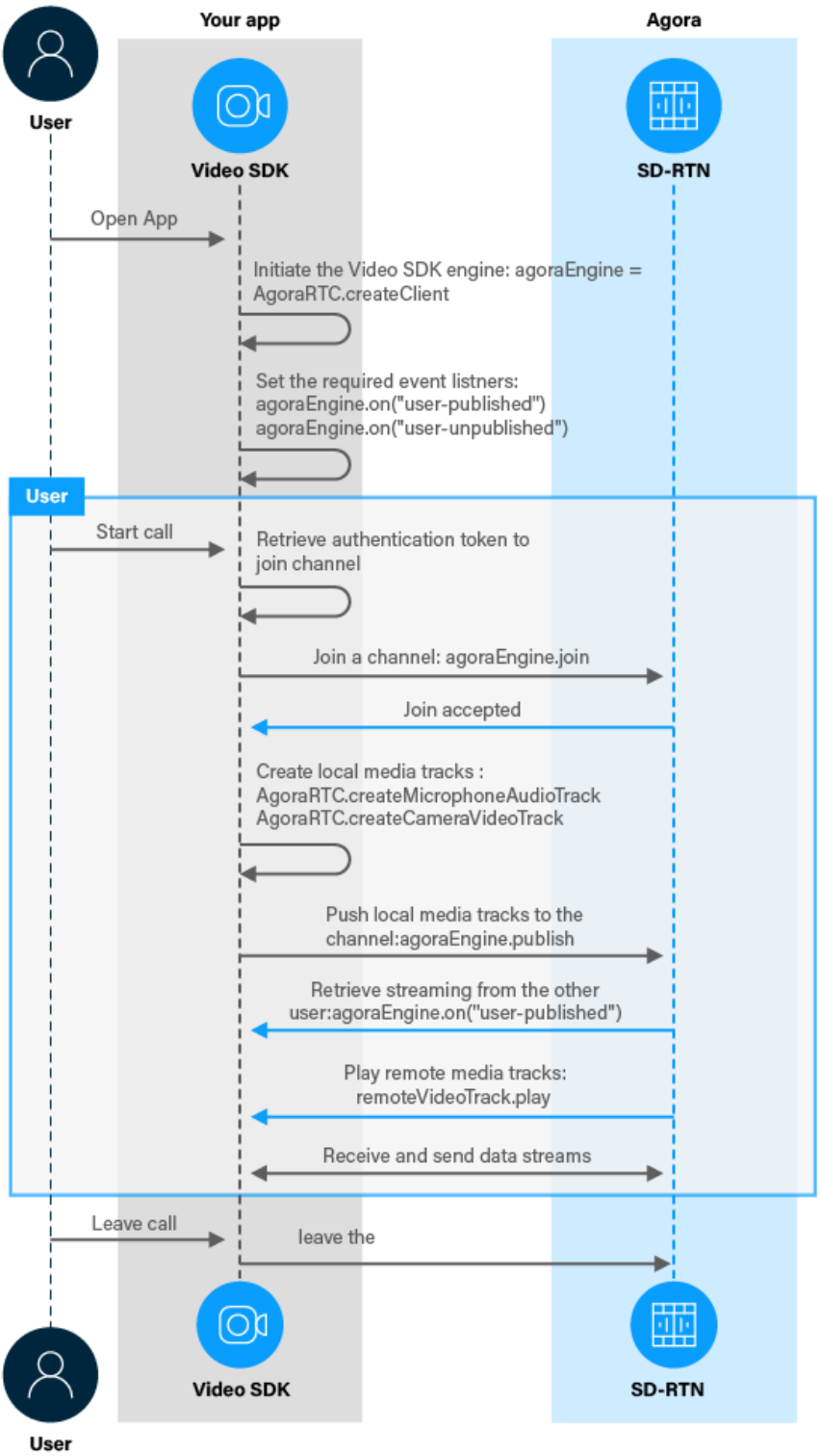

Figura 25. Flujo para la inicialización de una videoconferencia ofrecido por agora.io.

Capítulo 5. Pruebas

Las pruebas de este proyecto han sido realizadas de forma manual, para ello se ha comprobado que cada una de las funciones y características definidas en la fase de diseño haya sido implementada con éxito.

Para la realización de las pruebas se ha utilizado un servidor de pruebas ofrecido por Django. Para desplegara este servidor se debe ejecutar el siguiente comando:

```
$ python.exe .\manage.py runserver
```

Una vez desplegado podemos dirigirnos a la URL http://127.0.0.1:8000/ donde podremos acceder a nuestra aplicación.

Nada más acceder a la WEB apareceremos en la interfaz de búsqueda de doctores donde podremos filtrarlos para elegir el que deseemos. Pero para poder realizar esto deberemos estar dados de alta como un paciente.

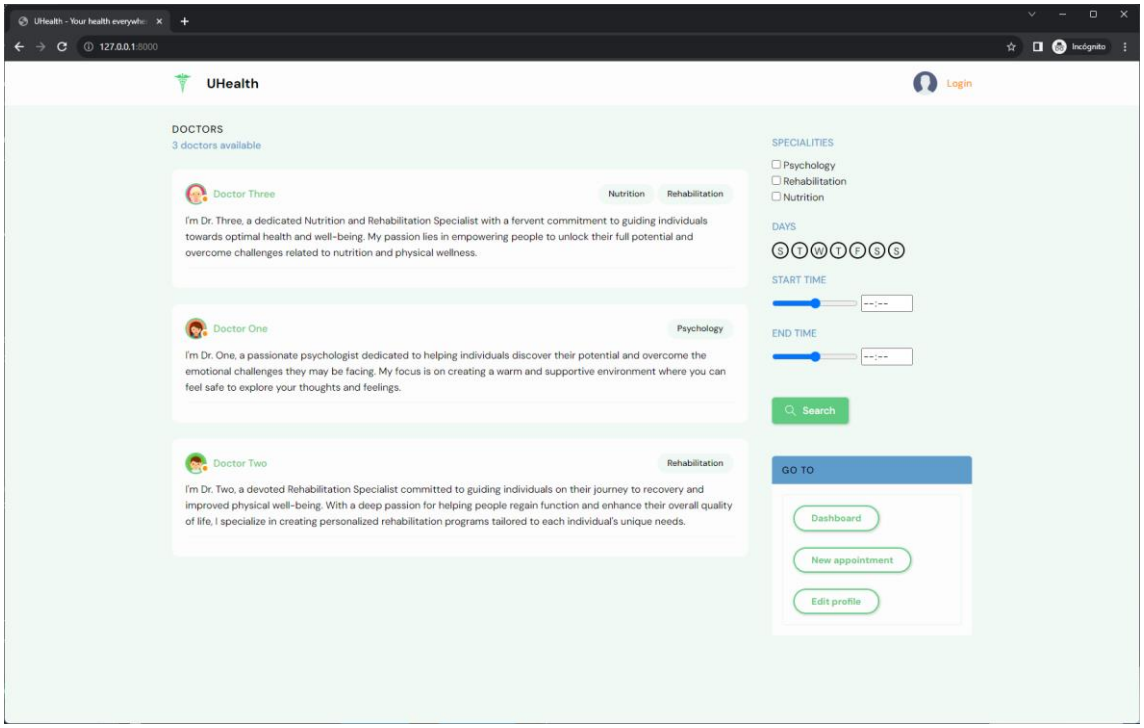

Figura 26. Vista inicial de la aplicación por un usuario no autenticado. Corresponde con la interfaz de búsqueda de doctores.

Una vez registrados como pacientes y seleccionaremos el doctor con que doctor deseamos tener una consulta. Accederemos a la siguiente interfaz en la que seleccionaremos el horario de la consulta y la especialidad (en caso de que el doctor disponga de más de una).

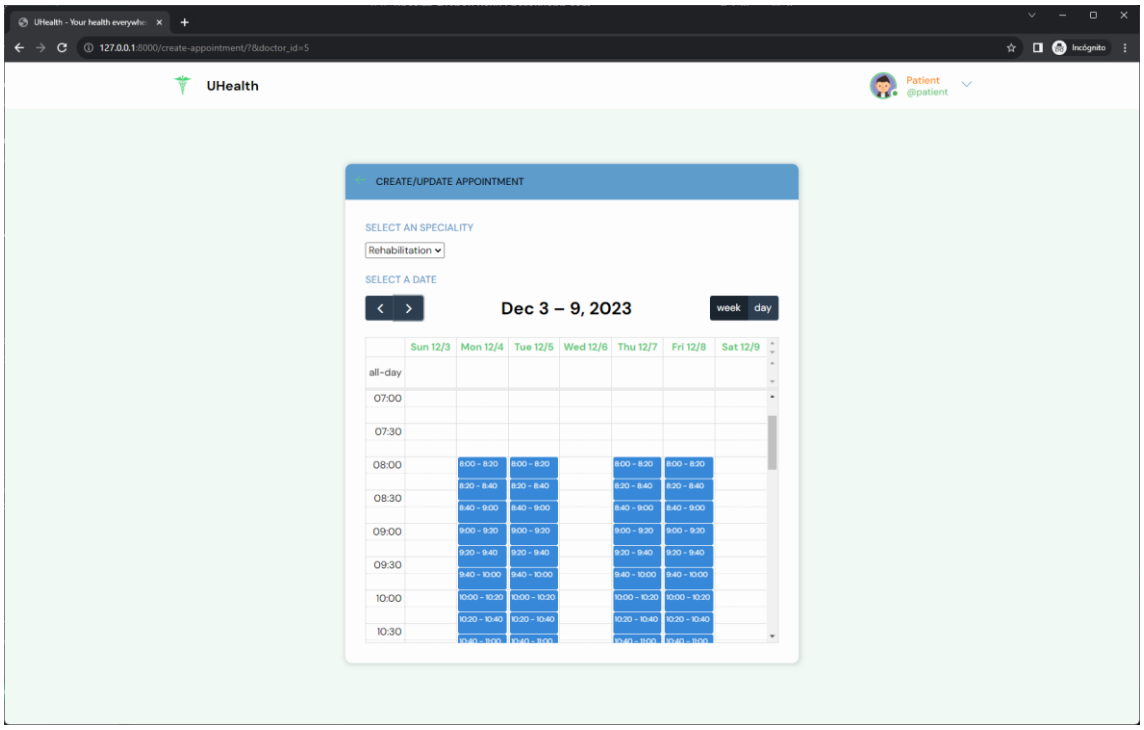

Figura 27. Captura de pantalla de la interfaz para la selección de un horario.

Una vez seleccionada la hora en la que deseamos tener la cita, esta nos aparecerá en nuestro dashboard.

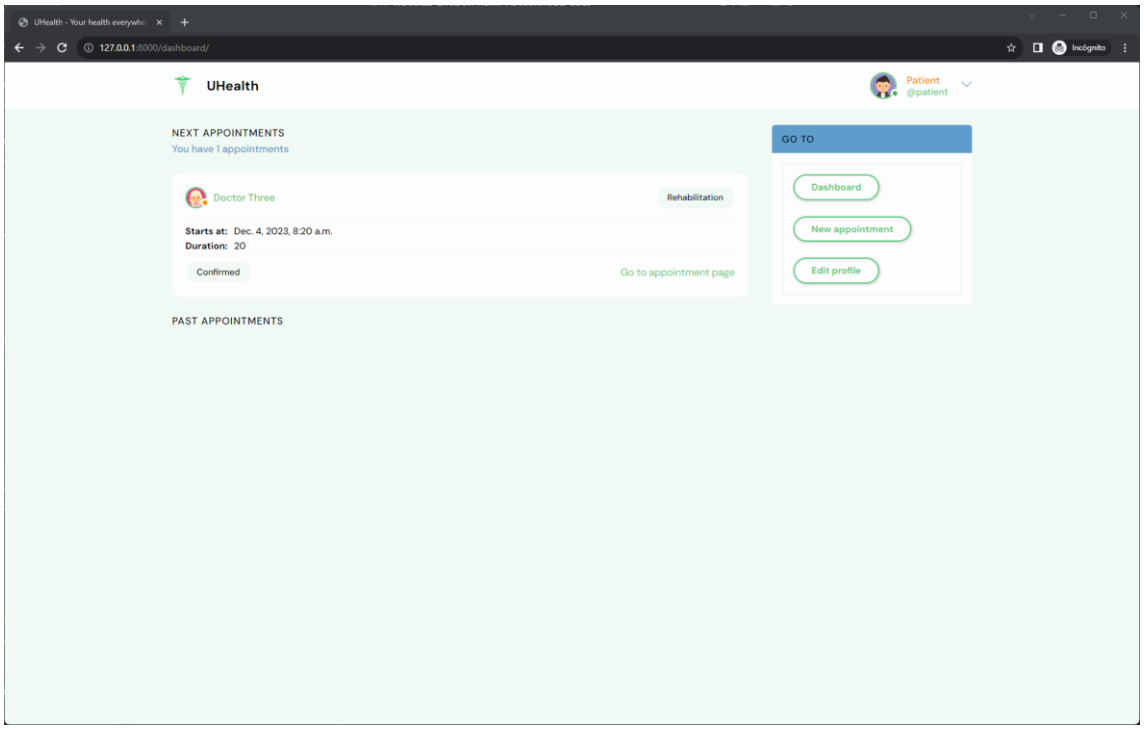

Figura 28. Captura de pantalla del dashboard

Si clicamos en el botón “Go to appointment page” podremos acceder a la pagina de la cita donde podremos ver la información de esta y acceder a la videoconferencia. Los doctores serán capaces de ver y apuntar notas del paciente.

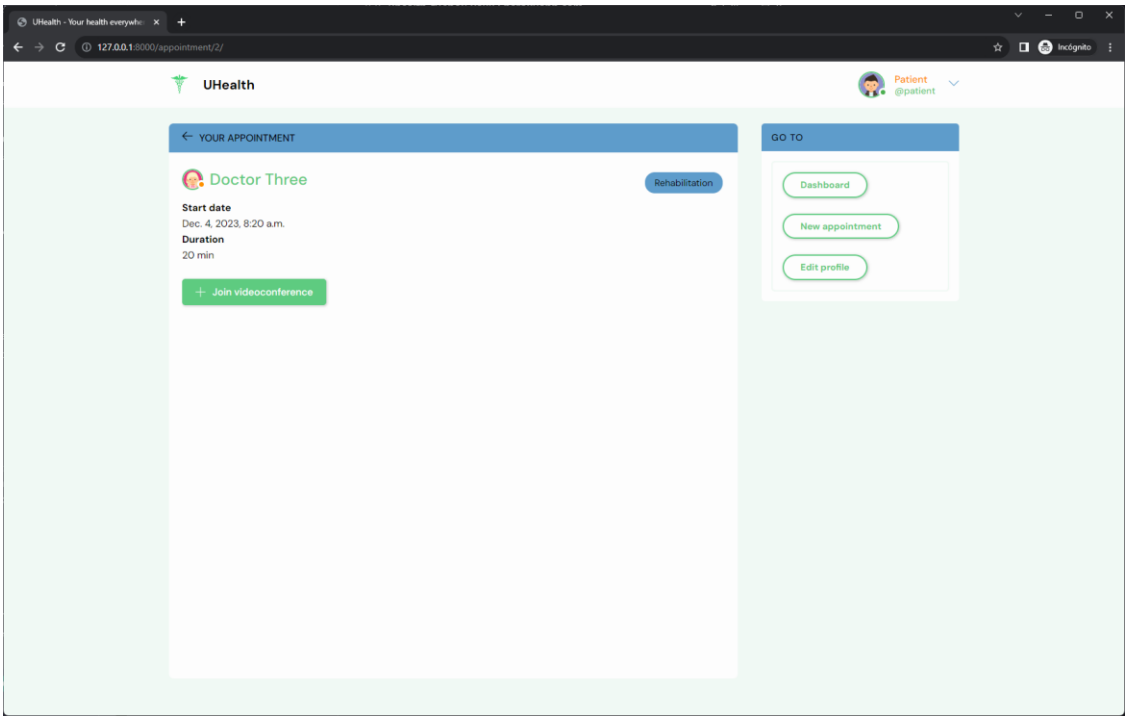

Figura 29. Captura de pantalla de la vista de cita por un paciente.

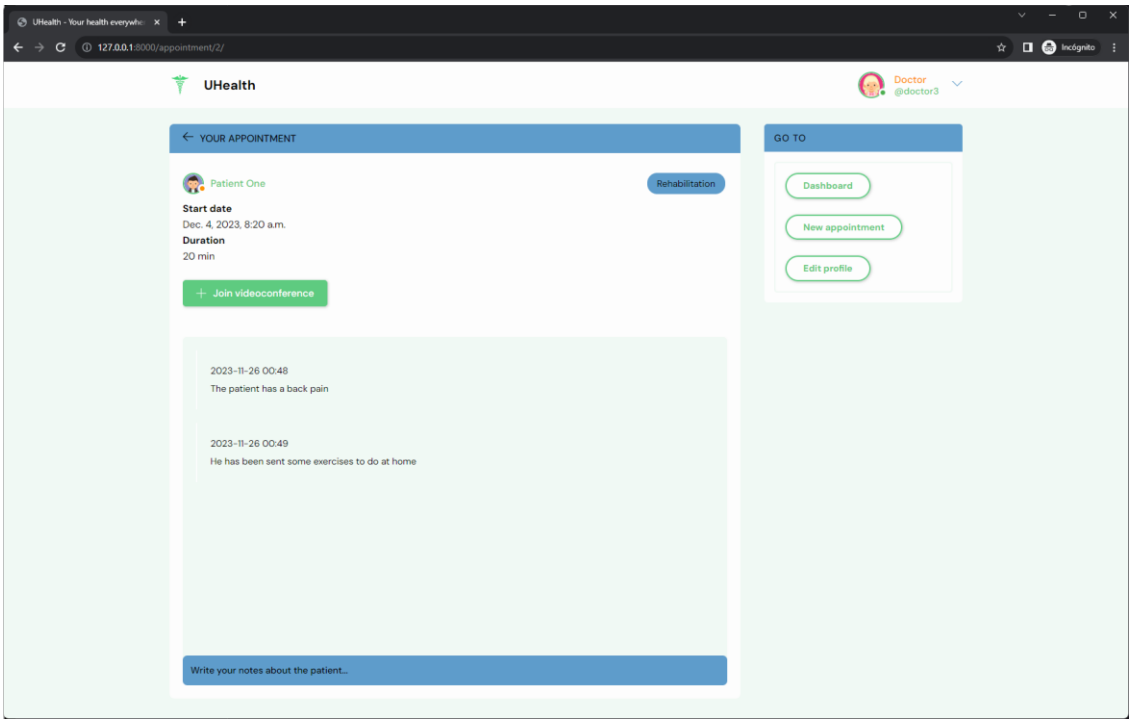

Figura 30. Captura de pantalla de la vista de cita por parte de un doctor.

Si pulsamos en “Join videoconference” accederemos directamente a la videoconferencia donde podremos activar y desactivar la cámara y el micrófono.

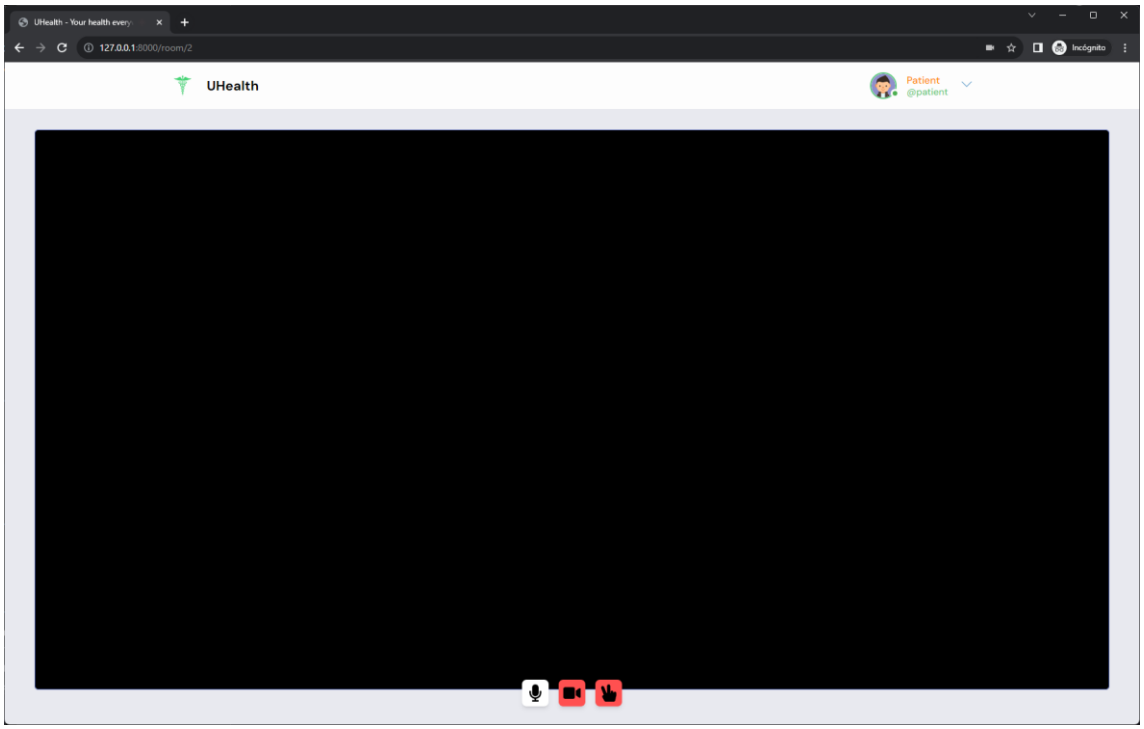

Figura 31. Captura de pantalla de una videoconferencia (cámara desactivada)

Si en lugar de ir a la videoconferencia pulsamos sobre el doctor para ver su perfil accederemos a la siguiente página, donde se mostrará la información básica del doctor. Si nuestro usuario fuese un doctor y estuviésemos accediendo al perfil de un paciente. Junto a su información básica, podríamos ver las notas que el doctor ha tomado sobre él, junto a la fecha de estas.

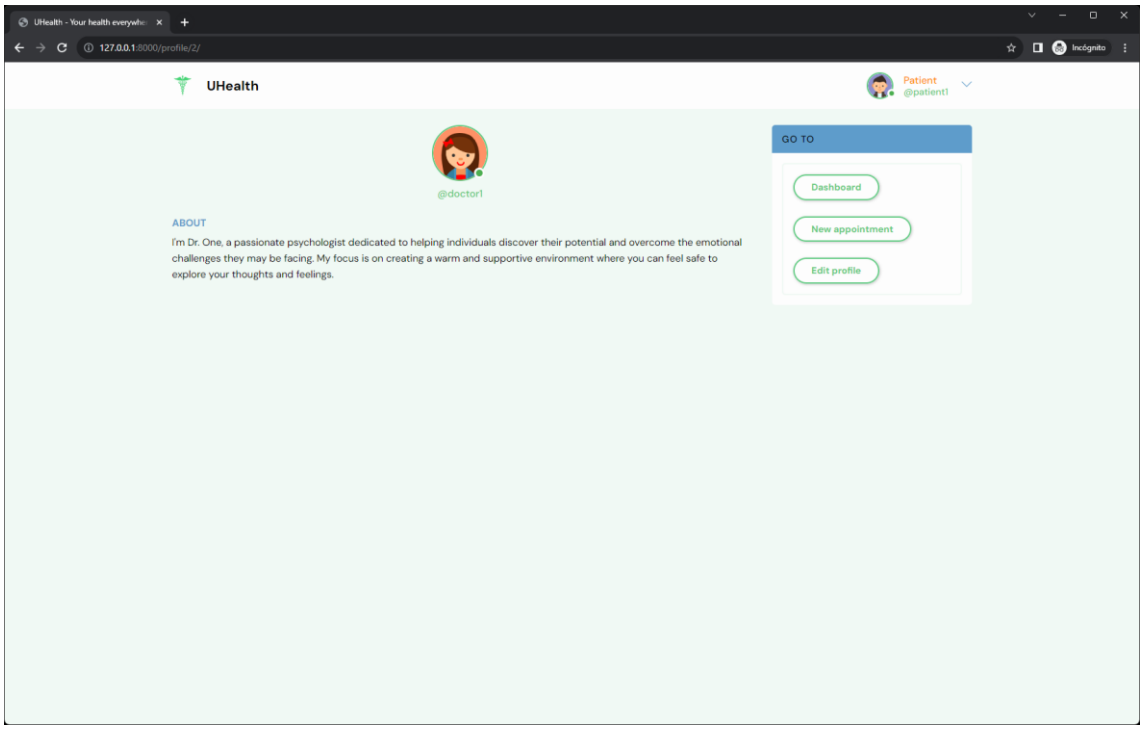

Figura 32. Captura de pantalla del perfil de un doctor.

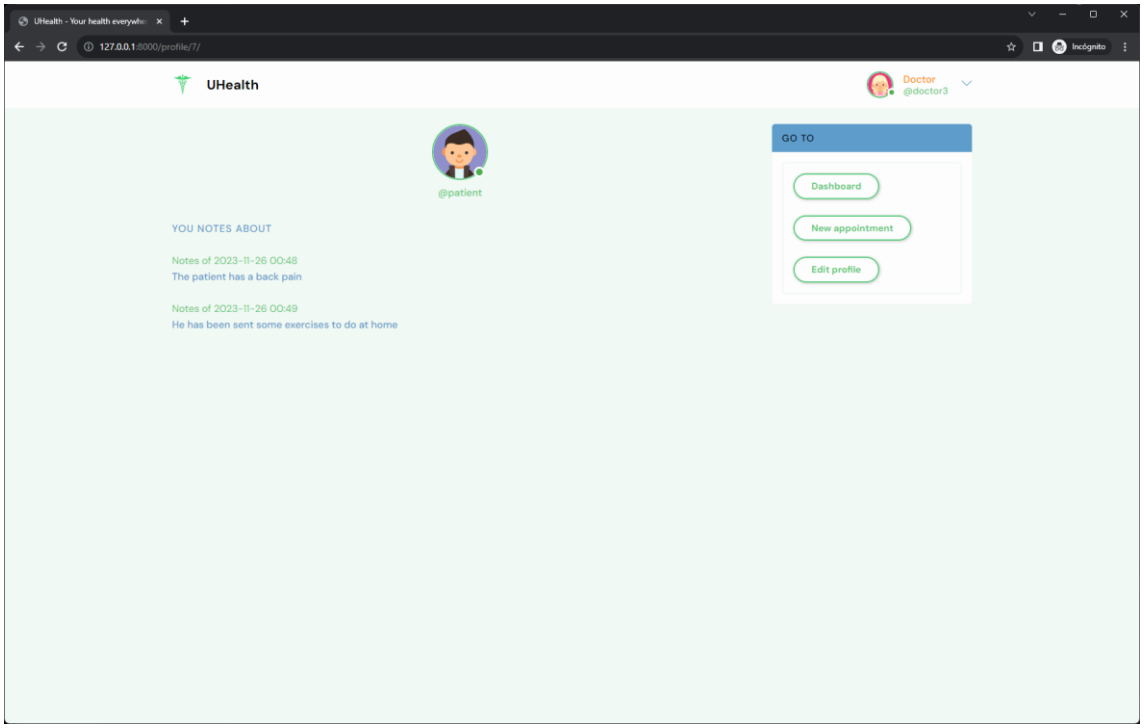

Figura 33. Captura de pantalla del perfil de un paciente.

Por último, podremos editar la información básica de nuestro perfil. Los doctores, además, pueden especificar su disponibilidad desde la interfaz de edición de perfil.

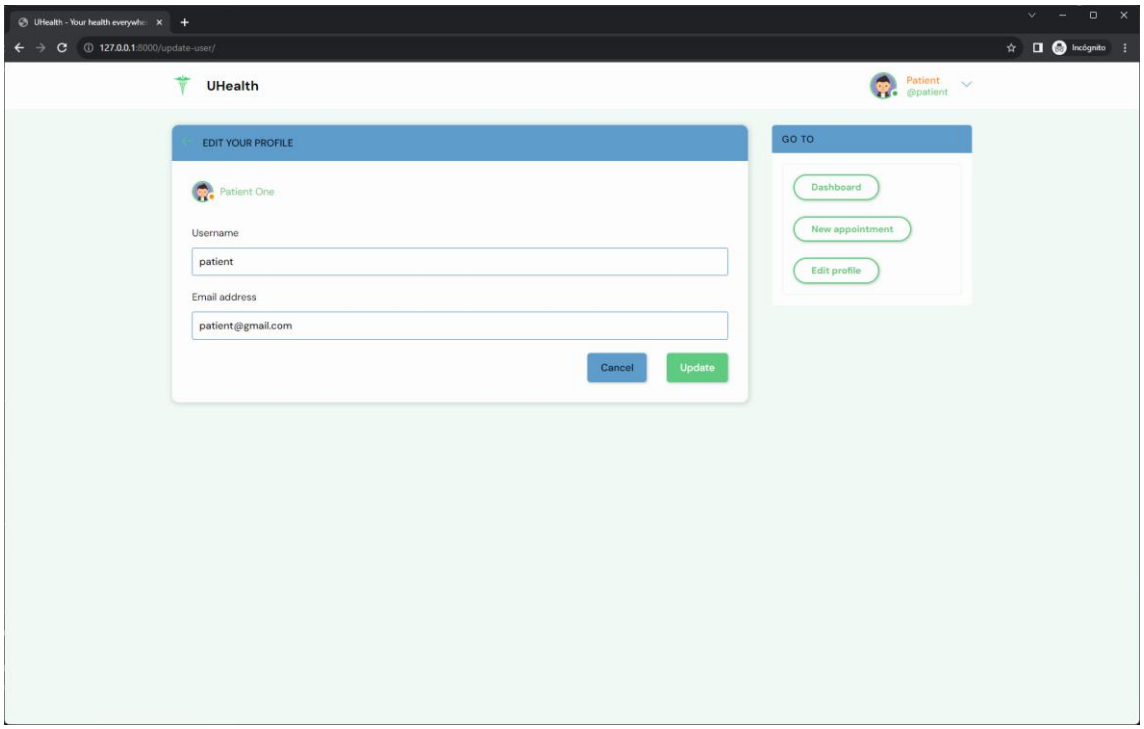

Figura 34. Captura de pantalla de la interfaz "editar usuario" de un cliente.

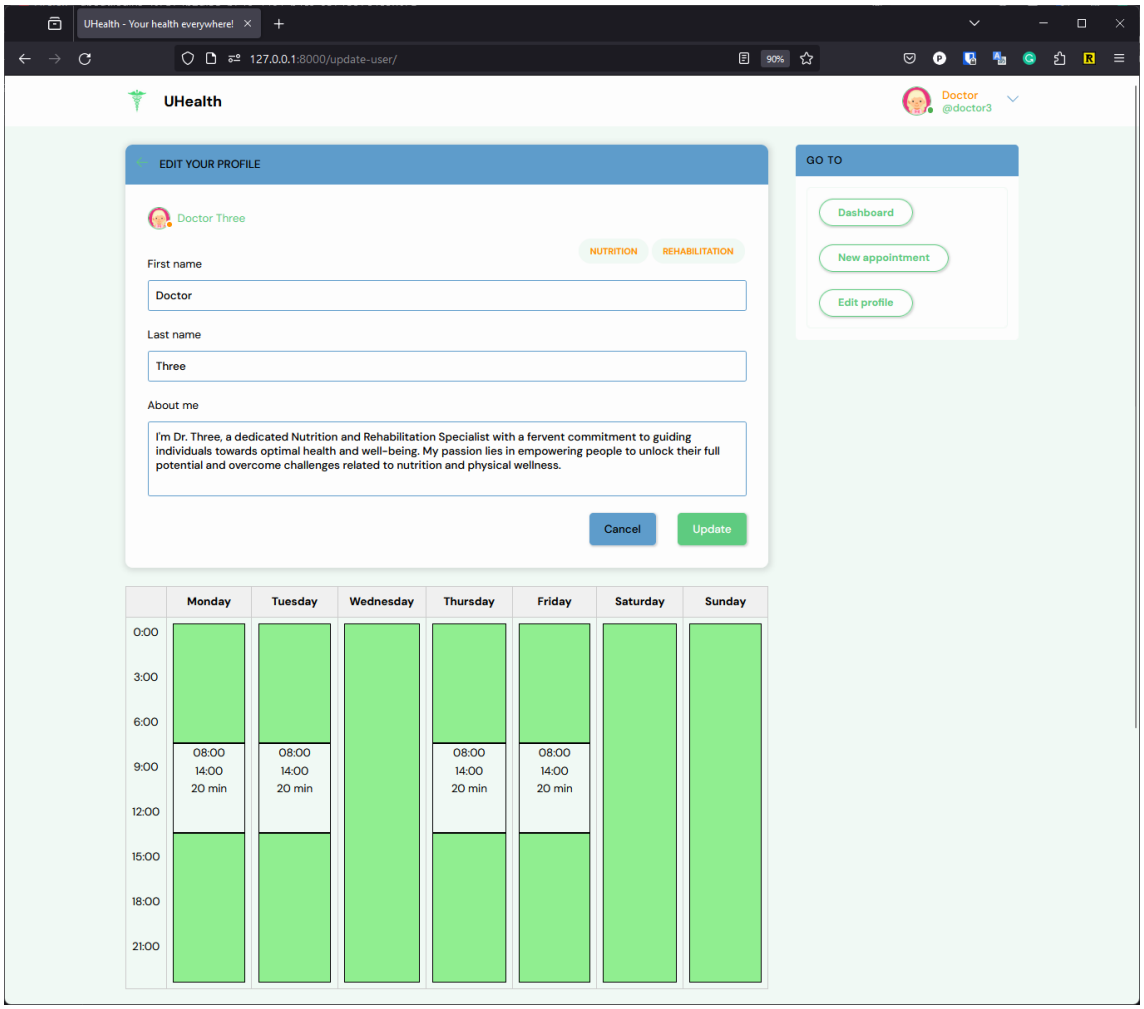

Figura 35. Captura de pantalla de la interfaz "editar usuario" de un doctor.

Capítulo 6. Despliegue

En este capítulo se comenta cual ha sido el procedimiento para el proceso de despliegue del proyecto. Tras esta fase el proyecto pasará a ser a accesible por los usuarios, o lo que es lo mismo,

Hasta ahora hemos estado trabajando en un entorno en desarrollo, empleando el *Django development web server*, que despliega la aplicación en la red local y ejecutando la aplicación con configuraciones inseguras que exponen información de depuración y otros detalles privados. Es por esto que para pasar a un entorno de producción deberemos de realizar una serie de pasos.

Un entorno de producción es el conjunto de recursos proporcionados por el servidor para que un sitio web esté disponible externamente. Incluye equipos, sistema operativo, lenguajes de programación, servidor web, servidor de aplicaciones y bases de datos. Puede ubicarse en instalaciones propias o en la nube, donde el código se ejecuta en un servidor remoto. Este entorno puede ser *Infrastructure as a Service* (IaaS), ofreciendo flexibilidad, pero que requiere de una mayor configuración, o *Platform as a Service* (PaaS), que simplifica el despliegue al encargarse de la mayoría de los componentes. La elección depende de las preferencias y necesidades del desarrollador. [22]

6.1 El proveedor de hosting

Para nuestro proyecto hemos elegido Heroku como proveedor de hosting, ya que destaca como uno de los servicios PaaS basados en la nube más establecidos y reconocidos, que admite una variedad de entornos de programación, incluyendo Django. Se ha elegido Heroku por las siguientes razones:

- Heroku ofrece un nivel gratuito que, a pesar de tener ciertas limitaciones, es suficiente para nuestro proyecto.
- Al ser un PaaS, Heroku se encarga de gran parte de la infraestructura web, eliminando la necesidad de preocuparse por servidores, balanceadores de carga, proxis inversos y otros aspectos de la infraestructura.
- Aunque presenta algunas limitaciones, estas no afectarán significativamente a nuestra aplicación específica. Por ejemplo, Heroku solo proporciona almacenamiento efímero y limita el tiempo de ejecución de la aplicación, especialmente en el nivel gratuito. Sin embargo, estas restricciones son aceptables para sitios de baja demanda o demostrativos.
- Heroku ha demostrado ser confiable y eficiente. Además, la escalabilidad de las aplicaciones en caso de que se requiriese es sencilla.

Es importante tener en cuenta que, si bien Heroku es ideal para albergar demostraciones y proyectos iniciales, puede no ser la opción más adecuada para un sitio web en producción. Aunque facilita la instalación y el escalado, su flexibilidad es limitada y los costos pueden aumentar considerablemente fuera del nivel gratuito. La elección dependerá de las necesidades y metas específicas de cada proyecto. [22]

6.2 La infraestructura

Heroku opera ejecutando sitios web Django en uno o más "Dynos", que son contenedores Unix virtualizados y aislados, proporcionando el entorno necesario para la ejecución de aplicaciones. Cada Dyno está completamente aislado y posee un sistema de archivos efímero, limpiado en cada reinicio. Las variables de configuración son compartidas entre los Dynos, y un balanceador de carga interno distribuye el tráfico web entre ellos. Heroku puede escalar horizontalmente agregando más Dynos, aunque se podría requerir escalar la base de datos para gestionar conexiones adicionales. [22]

Debido a la efimeridad del sistema de archivos, no es posible instalar directamente servicios necesarios (como bases de datos, colas, sistemas de caché, almacenamiento, servicios de correo electrónico, etc.). En su lugar, las aplicaciones web de Heroku utilizan servicios proporcionados por Heroku o terceros como “add-ons”. Los Dynos acceden a estos servicios mediante la información contenida en las variables de configuración de la aplicación.

Para ejecutar una aplicación en Heroku, es necesario especificar el entorno y las dependencias adecuadas. En el caso de aplicaciones Django, esta información se proporciona a través de archivos como runtime.txt (para el lenguaje de programación y versión), requirements.txt (para las dependencias, incluyendo Django), Procfile (para la lista de procesos que ejecutan la aplicación, como Gunicorn), y wsgi.py (configuración WSGI para invocar la aplicación Django en el entorno Heroku).

Los desarrolladores interactúan con Heroku a través de una aplicación/terminal cliente especial, similar a un script de bash en Unix. Esto permite subir código desde un repositorio git, inspeccionar procesos en ejecución, visualizar registros, configurar variables y más.

Para implementar una aplicación en Heroku, se necesita colocar la aplicación en un repositorio git, agregar los archivos mencionados, integrarse con una base de datos "add-on", y realizar ajustes para manejar adecuadamente los archivos estáticos. Luego, se crea una cuenta en Heroku, se obtiene el cliente Heroku y se utiliza para instalar la aplicación web.

Es importante tener en cuenta que estas instrucciones reflejan el proceso en el momento de la redacción, y se recomienda consultar la documentación oficial de Heroku para obtener información actualizada.

6.3 Preparación del proyecto para el despliegue

Para el despliegue del proyecto en Heroku, existen una serie de configuraciones que deberemos de realizar antes de desplegar nuestro proyecto y pasar a un entorno en producción, existen una serie de configuraciones que deberemos realizar en nuestro proyecto.

6.3.1 Configuraciones de seguridad

Estas configuraciones se realizan modificando el fichero *settings.py* para optimizar la aplicación en términos de seguridad y rendimiento.

El primer cambio que se debería de realizar es desactivar el modo de depuración. La variable que indica el modo de depuración debe establecerse como False en producción (DEBUG = False) para evitar la exposición de información sensible y trazas de depuración.

Ocultar el valor de la variable SECRET KEY. Este valor aleatorio, vital para la seguridad de la aplicación, debe ser manejado con precaución. Se sugiere cargarlo desde una variable de entorno o leerlo desde un archivo de solo servicio. Este valor es utilizado por ejemplo en la protección CRSF.

Para realizar las configuraciones mencionadas anteriormente se han modificado el fichero de configuración “*settings.py*” para leer las variables SECRET_KEY y DEBUG desde variables de entorno, o usar valores por defecto si no están definidas. Estos cambios permiten flexibilidad al trabajar con valores por defecto durante el desarrollo y variables de entorno en producción.

```
# SECURITY WARNING: keep the secret key used in production secret!
# SECRET_KEY = 'django-insecure-q9y2=jrlg9uwv!@s^vg*1rg2z!q2$vdzi&2&cx20^e)8(sg+(&'
import os
SECRET_KEY = os.environ.get('DJANGO_SECRET_KEY', 'django-insecure-q9y2=jrlg9uwv!@s^vg*1rg2z!q2$vdzi&2&cx20^e)8(sg+(&')

# SECURITY WARNING: don't run with debug turned on in production!
# DEBUG = True
DEBUG = bool( os.environ.get('DJANGO_DEBUG', True) )
```

Figura 36. Fragmento del archivo settings.py en el que se muestra la configuración añadida.

6.3.2 Configuración de la Base de Datos:

La base de datos SQLite por defecto no puede ser utilizada en Heroku, ya que está basada en archivos y sería eliminada del sistema de archivos efímero cada vez que la aplicación se reinicie. Para gestionar esta situación, se utiliza un *add-on* de base de datos en Heroku, y la aplicación web se configura utilizando información de una variable de configuración del entorno establecida por dicho *add-on*. Se utilizará el nivel gratuito de la base de datos Heroku PostgreSQL ya que, compatible con Django y está incluida en el nuevo plan *dyno* de nivel gratuito.

La información de conexión a la base de datos es proporcionada a la web *dyno* a través de una variable de configuración denominada DATABASE_URL. En lugar de codificar esta información en Django, Heroku recomienda que los desarrolladores utilicen el paquete *dj-database-url* para extraer la variable de entorno DATABASE_URL y convertirla automáticamente al formato de configuración requerido por Django. Además, será necesario instalar psycopg2 para que Django pueda interactuar con la base de datos PostgreSQL.

Para la realización de todo lo mencionado anteriormente se modificará el archivo *settings.py* y se añadirán las siguientes líneas de código.

```
# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

# Heroku: Actualice la configuración de la base de datos desde $DATABASE_URL.
import dj_database_url
db_from_env = dj_database_url.config(conn_max_age=500)
DATABASES['default'].update(db_from_env)
```

Figura 37. Fragmento del archivo settings.py en el que se muestra la configuración de la BD.

Como se puede ver en la imagen anterior se sobrescribe el valor de configuración de la base de datos “default” por la nueva base de datos cuya información de conexión será proporcionada en la variable de entorno DATABASE_URL.

6.3.3 Sirviendo ficheros estáticos

Durante el desarrollo, Django y el servidor web de desarrollo de Django se utilizaban para servir los archivos estáticos (CSS, JavaScript, etc.). En un entorno de producción, comúnmente, los archivos estáticos se sirven desde una red de entrega de contenidos (CDN) o desde el servidor web.

La entrega de archivos estáticos a través de Django es ineficiente, ya que las solicitudes deben pasar por código adicional innecesario, en lugar de ser gestionadas directamente por el servidor web o una CDN completamente independiente. Aunque esto no tiene relevancia en el uso local durante el desarrollo, el uso de este mecanismo en producción tiene un impacto significativo en el rendimiento.

Con el fin de facilitar el alojamiento de archivos estáticos de forma separada de la aplicación web Django, se proporciona la herramienta "*collectstatic*" para recoger estos archivos para el despliegue (hay una variable de configuración que define de dónde se deben recopilar los archivos cuando se ejecuta "*collectstatic*"). Las plantillas Django hacen referencia a la ubicación de almacenamiento de los archivos estáticos en relación con una variable de configuración STATIC_URL, por lo tanto, esto puede modificarse si los archivos estáticos son movidos a otro host/servidor.

Las variables de configuración más relevantes son:

STATIC_URL: Es la ubicación URL base desde la cual se servirán los archivos estáticos, por ejemplo, en una CDN. Se utiliza para variables de plantilla estáticas a las que se accede en nuestra plantilla base.

STATICFILES_DIRS: Relaciona directorios adicionales en los que la herramienta *collectstatic* de Django debería buscar archivos estáticos.

STATIC_ROOT: Es la ruta absoluta a un directorio en el que la herramienta *collectstatic* de Django reunirá todos los archivos estáticos referenciados en nuestras plantillas. Una vez recopilados, podrán ser cargados como un grupo a donde hayan de ser alojados.

Con todo esto modificaremos el fichero "*settings.py*" con la siguiente configuración:

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = 'static/'

# La URL que se utilizará cuando se haga referencia a archivos estáticos (desde donde se entregarán)
STATICFILES_DIRS = [
    BASE_DIR / 'static'
]

# La ruta absoluta al directorio donde Collectstatic recopilará archivos estáticos para su implementación.
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
```

Figura 38. Fragmento del archivo settings.py en el que se muestra la configuración de los ficheros estáticos.

Para servir los ficheros estáticos en producción se utilizará una biblioteca llamada *WhiteNoise*, recomendada por Heroku para servir objetos estáticos directamente desde *Gunicorn*. Heroku llama automáticamente a *collectstatic* y prepara los ficheros estáticos para ser usados por *WhiteNoise* después de que se cargue la aplicación.

Para instalar WhiteNoise en nuestra aplicación Django, deberemos de agregar el middleware "*WhiteNoiseMiddleware*" a la lista MIDDLEWARE de la configuración de Django.

```
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',
    'whitenoise.middleware.WhiteNoiseMiddleware',
]
```

Figura 39. Fragmento del archivo settings.py en el que se muestra la adición del middleware.

6.3.4 Procfile

El archivo Procfile en Django es una declaración de procesos utilizado especialmente en entornos de implementación como Heroku. Este archivo, proporciona información clave a Heroku sobre cómo gestionar y ejecutar la aplicación de manera efectiva. En su contenido, se especifica el tipo de proceso y el punto de entrada para la aplicación.

Se creará un nuevo archivo nombrado simplemente "Procfile" y sin extensión, que se coloca en la carpeta raíz del repositorio de la aplicación Django. El contenido del archivo será el siguiente:

```
web: gunicorn project.wsgi --log-file -
```

La etiqueta "web:" precediendo la instrucción, informa a Heroku sobre el tipo de *dyno* y cómo debe ser manejado en términos de tráfico. En este caso indica que se tratara de un *dyno* web, apto para manejar tráfico HTTP.

El comando "gunicorn project.wsgi --log-file -" inicia Gunicorn utilizando la configuración definida en el módulo "project.wsgi", el cual es creado durante la configuración inicial de la aplicación Django. Gunicorn es un servidor de aplicaciones web ampliamente utilizado y recomendado por Heroku.

6.3.5 Requerimientos

Durante la reconstrucción del entorno, los requisitos de la aplicación web serán instalados por Heroku. Para indicar cuales serán estos requisitos deberemos crear un archivo llamado "requirements.txt" situado en la carpeta principal del repositorio. El contenido de este fichero será el siguiente:

```
agora-token-builder==1.0.0
dj-database-url==2.1.0
Django==4.2
gunicorn==21.2.0
psycopg2==2.9.9
whitenoise==6.6.0
```

6.3.6 Realización de un commit con los cambios

Una vez finalizada la configuración, se realiza un commit con todos los cambios. Este commit será el que será subido al repositorio remoto de Heroku.

6.4 Despliegue en Heroku

El despliegue del proyecto en Heroku se realizará a través de un cliente con el que podremos ejecutar comandos a través de la terminal.

Una vez instalado y configurado el cliente, lo primero que realizaremos será crear la aplicación.

```
$ heroku create uhealth
```

Una vez creada realizaremos un push de nuestro repositorio a la rama "heroku" que habrá sido creada automáticamente en nuestro repositorio durante la instalación del cliente.

```
$ git push heroku master
```

El siguiente paso será añadir a el "add-on" con la base de datos PostgreSQL desde el *dashboard* de la interfaz de Heroku.

Figura 40. Captura de pantalla del add-on de la BD PostgreeSQL

También desde la interfaz de Heroku crearemos las variables de entorno DEBUG y SECRET_KEY y les asignaremos los valores.

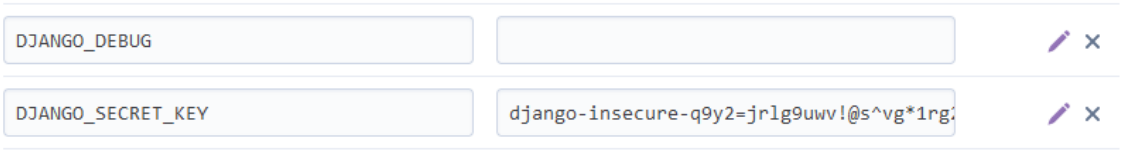

Figura 41. Captura de las variables de estado en la interfaz de Heroku.

Una vez añadida la base de datos, ya podremos crear las tablas necesarias para nuestra aplicación. Para ello ejecutamos el comando "migrate" de Django.

```
$ heroku run python manage.py migrate
```

Por último, crearemos un usuario administrador para añadir las especialidades y verificar a los doctores a medida que estos se vayan registrando.

```
$ heroku run python manage.py createsuperuser
```

Con todo esto ya seremos capaces de acceder a la aplicación. Para ello podremos servirnos del cliente y utilizar el siguiente comando

```
$ heroku open
```

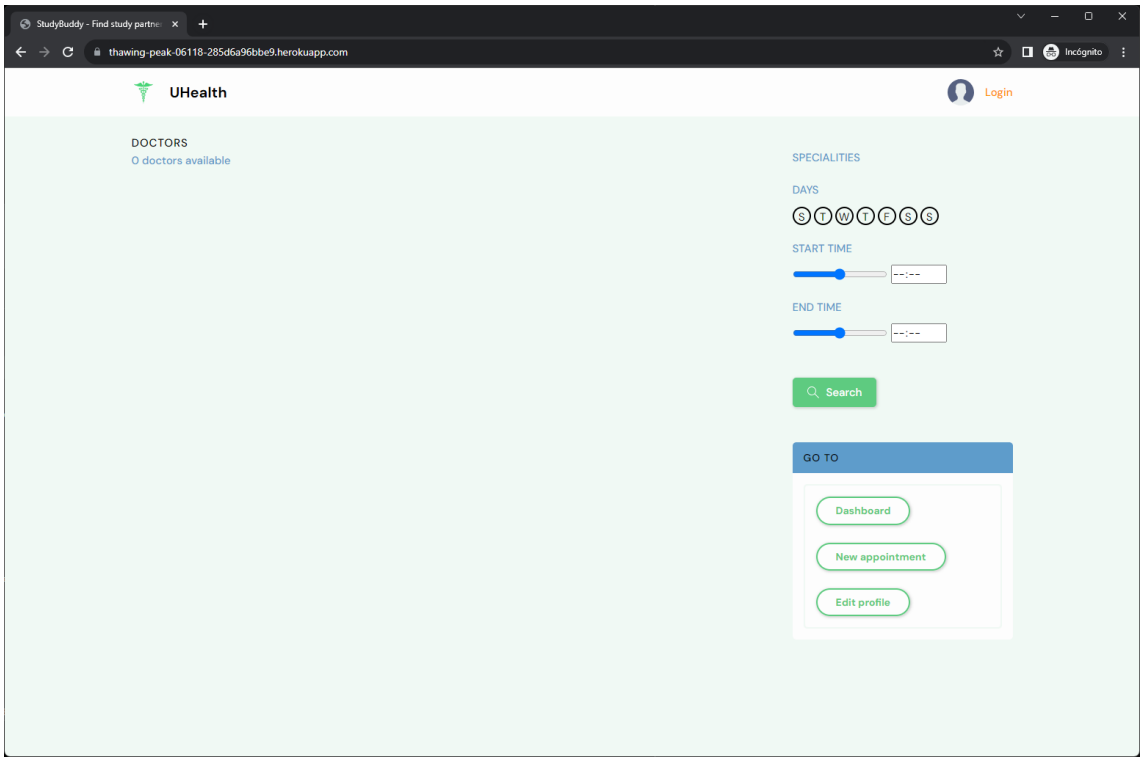

Figura 42. Captura de pantalla de la aplicación tras ser redirigidos a la página principal.

Como se puede apreciar en la imagen anterior, el despliegue se ha realizado con éxito. Somos capaces de acceder a la aplicación WEB a través del enlace proporcionado por Heroku. No aparece ningún doctor porque todavía no han sido añadidos.

Capítulo 7. Conclusiones y propuesta de trabajo futuro

El desarrollo de la aplicación web planteada durante todo este proyecto ha resultado en la creación de una herramienta efectiva y accesible. Que cumple con todos los requisitos planteados. Además de estos logros, se derivan conclusiones significativas.

La selección de tecnologías ampliamente consolidadas y respaldadas, con abundancia de tutoriales, ha demostrado ser clave. Esta elección ha facilitado significativamente la resolución de desafíos, permitiéndome abordar problemas de manera más eficiente al aprovechar las soluciones compartidas por la comunidad.

Llevar a cabo las diversas etapas del desarrollo de la aplicación, desde el diseño hasta el despliegue, ha representado un esfuerzo considerable. Sin embargo, me ha proporcionado una visión integral de los procesos involucrados. Ha sido una experiencia formativa que no solo ha contribuido al éxito de la aplicación, sino que también ha enriquecido mi comprensión global de la implementación de proyectos tecnológicos, brindándome valiosas lecciones y habilidades que serán de utilidad en futuros emprendimientos.

A pesar de que la aplicación cumple con los requisitos establecidos, se identifican funcionalidades y características que podrían enriquecerla de manera significativa.

En particular, no se ha contemplado la integración de métodos de pago en este proyecto. Una mejora sugerida sería la incorporación de una funcionalidad que permita a los pacientes realizar pagos por sus consultas directamente dentro de la aplicación. Al considerar esta adición, sería crucial abordar los requisitos y procesos de documentación necesarios para gestionar eficientemente esta transacción financiera, resolviendo cualquier implicación administrativa y de seguridad.

Además, otra característica que podría mejorar la utilidad de la aplicación sería la implementación de un sistema para la expedición de recetas médicas. Aunque la filosofía de la aplicación se centra en la simplicidad y el minimalismo, se podría explorar la posibilidad de establecer un sistema colaborativo con instituciones de salud y farmacias. Este sistema podría crear un proceso de recetas que requiera una interacción mínima o nula por parte del paciente, contribuyendo así a la eficiencia y conveniencia del servicio.

Además de la implementación de nuevas funcionalidades, es imperativo llevar a cabo un mantenimiento continuo de los servicios para abordar la resolución de bugs y mejorar aspectos fundamentales de la aplicación.

La seguridad de la aplicación debe ser una prioridad constante, ya que los entornos digitales están sujetos a amenazas en evolución. Implementar actualizaciones de seguridad de manera regular, así como realizar auditorías periódicas, contribuirá a garantizar la protección de los datos de los usuarios y la integridad del sistema.

La mejora continua de la interfaz de usuario es esencial para mantener la aplicación actualizada y alineada con las expectativas de los usuarios. Esto implica la optimización de la usabilidad, la claridad en el diseño y la incorporación de comentarios de los usuarios para adaptarse a sus necesidades cambiantes.

El soporte efectivo para dispositivos móviles también es crucial en un entorno donde la accesibilidad desde teléfonos y tabletas es común. Asegurarse de que la aplicación sea completamente funcional y ofrezca una experiencia óptima en diversos dispositivos móviles garantizará una mayor satisfacción del usuario y una mayor cobertura de la audiencia.

Capítulo 8. Bibliografía

[1] Metodología Waterfall. https://www2.deloitte.com/es/es/pages/technology/articles/waterfall-vs-agile.html

[2] Colaboradores de Wikipedia. (2023, 14 noviembre). Python. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Python

[3] Amazon Web Services, Inc. (s. f.). ¿Qué es Python? - Explicación del lenguaje Python. https://aws.amazon.com/es/what-is/python/

[4] MDN Web Docs. (2023, 2 agosto). Introducción a Django - Aprende Desarrollo Web. https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introduction

[5] Sanwo, S. (2022, 11 abril). How to set up a virtual environment in Python - and why it's useful. freeCodeCamp.org. https://www.freecodecamp.org/news/how-to-setup-virtual-environments-in-python/

[6] VENV - Creación de entornos virtuales. (s. f.). Python documentation. https://docs.python.org/es/3/library/venv.html

[7] Agora Docs. (s. f.). Video Calling SDK Quickstart. https://docs.agora.io/en/video-calling/get-started/get-started-sdk?platform=web

[8] Ihor. (s. f.). All you need to know about a FullCalendar View module. Five Jars. https://fivejars.com/blog/all-you-need-know-about-fullcalendar-view-module

[9] FullCalendar - JavaScript Event Calendar. (s. f.). https://fullcalendar.io/

[10] Heroku. (s. f.). What is Heroku. https://www.heroku.com/what

[11] Papertrail. (2022b, abril 14). What is Heroku? Everything you need to know. https://www.papertrail.com/solution/guides/heroku/

[12] Clark, M. (2023, 9 junio). What is Heroku? All the secrets unlocked. Back4App Blog. https://blog.back4app.com/what-is-heroku/

[13] Django. (s. f.). Django Project - Models. https://docs.djangoproject.com/en/4.2/topics/db/models/

[14] Django. (s. f.). Django Project - Forms. https://docs.djangoproject.com/en/4.2/topics/forms/

[15] Django. (s. f.). Django Project - Views. https://docs.djangoproject.com/en/4.2/topics/http/views/

[16] Django. (s. f.). Django Project - Templates. https://docs.djangoproject.com/en/4.2/topics/templates/

[17] Django. (s. f.). Django Project - Decorators. https://docs.djangoproject.com/en/4.2/topics/http/decorators/

[18] Django. (s. f.). Django Project - URLs. https://docs.djangoproject.com/en/4.2/topics/http/urls/

[19] GeeksforGeeks. (2019, 6 diciembre). Git features. https://www.geeksforgeeks.org/git-features/

[20] Mijacobs. (2022, 28 noviembre). What is Git? - Azure DevOps. Microsoft Learn. https://learn.microsoft.com/en-us/devops/develop/git/what-is-git

[21] PMBC. (2023, 16 agosto). Metodología Waterfall: ¿Qué es y cómo se aplica? PMBC. https://pmbc.es/metodologia-waterfall-que-es-y-como-se-aplica/#:~:text=An%C3%A1lisis%3A%20planificaci%C3%B3n%2C%20an%C3%A1lisis%20y%20especificaci%C3%B3n,de%20sistema%20y%20de%20integraci%C3%B3n.

[22] MDN Web Docs. (2023, 1 septiembre). Tutorial de Django Parte 11: Desplegando Django a producción. https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Deployment.