Universidad de La Habana
Facultad de Matemática y Computación

DermaUH, un sitio de ayuda al diagnóstico dermatoscópico.

Autor:
Mauricio Salim Mahmud Sánchez

Tutores:

Dra. Marta L. Baguer Díaz-Romañach

Lic. Rocío Ortiz Gancedo
Lic. Manuel Vilas Valiente

Trabajo de Diploma
presentado en opción al título de
Licenciado en (Matemática o Ciencia de la Computación)

5 de diciembre de 2023

Dedicación

Agradecimientos

Agradecimientos

Opinión del tutor

Opiniones de los tutores

Resumen

El presente trabajo propone diseñar e implementar el sitio web DermaUH para brindar ayuda a los médicos en el estudio y el análisis de lesiones de piel utilizando herramientas modernas y de código abierto. Brinda administración de pacientes, lesiones e imágenes de estas, acceso a herramientas de aprendizaje de máquinas orientado a la clasificación de afecciones y promueve el intercambio de información entre el personal de la salud.

Abstract

This paper proposes to design and implement the DermaUH website to assist doctors in the study and analysis of skin lesions using modern and open source tools. It provides management of patients, lesions and lesion images, access to machine learning tools oriented to the classification of diseases, and promotes the exchange of information among health care personnel.

Índice general

Introducción

La dermatología es un campo médico de vital importancia que aborda el diagnóstico y tratamiento de enfermedades y trastornos de la piel. El seguimiento y estudio de las lesiones cutáneas son un componente crítico de esta disciplina, ya que ayudan a los médicos a entender la evolución de las afecciones de la piel y a desarrollar tratamientos efectivos.

Con el avance de las técnicas de aprendizaje de máquinas surgen un conjunto de herramientas de gran utilidad para el personal de la salud, permitiendo la clasificación y segmentación de imágenes de lesiones, disminuyendo la fatiga visual y sirviendo de referencia y ayuda al personal de salud en adiestramiento. En este contexto, la tecnología digital y la web han abierto la puerta a nuevas oportunidades para mejorar la atención médica. Los sitios web pueden ser una herramienta valiosa para los médicos, proporcionando un espacio para el seguimiento y estudio de las lesiones cutáneas, dando acceso y democratizando estas herramientas a cualquier médico con acceso a internet.

Este trabajo de diploma se centrará en el desarrollo de un sitio web diseñado para ayudar al personal médico en el seguimiento y estudio de las lesiones de piel. El sitio web proporcionará una plataforma para documentar y analizar las lesiones cutáneas, permitiendo a los médicos seguir la evolución de estas afecciones a lo largo del tiempo y, en última instancia, mejorar los enfoques de diagnóstico y tratamiento. También se integrarán modelos previos de aprendizaje de maquinas con muy buenos resultados para la clasificación y segmentación de imágenes de afecciones, brindando una herramienta de gran apoyo al personal de la salud. Finalmente, el sitio web puede facilitar la colaboración entre los médicos, permitiéndoles compartir y discutir casos, lo que puede conducir a una mejor comprensión y tratamiento de las lesiones cutáneas.

Existen muchas plataformas a nivel global que permiten el estudio y análisis de lesiones de piel, a pesar de esto, la accesibilidad a estas puede ser limitada o inexistente para el personal de la salud en Cuba debido a restricciones geográficas, monetarias o políticas. En la actualidad en nuestro pais no existe ninguna aplicación que pueda suplir las necesidades de los dermatólogos, la creación de esta plataforma puede tener un impacto significativo en la mejora de la atención dermatológica, permitiendo a los

médicos realizar un seguimiento más efectivo de las lesiones de la piel y mejorar su capacidad para diagnosticar y tratar estas afecciones.

Problema

No existe una plataforma que sirva de apoyo a los médicos cubanos donde puedan seguir, estudiar, compartir, discutir sobre las lesiones de piel y acceder a herramientas de software de su campo.

Objetivo General

~~La creación de~~ una plataforma para los médicos cubanos que cumpla los requerimientos planteados.

Objetivos Específicos

1. Diseñar e implementar la plataforma que sea lo más extensible posible.
2. ~~Creación~~ del Frontend[1] de la plataforma como sitio web.
3. ~~Creación~~ del Backend[2] de la plataforma como una API Rest[3] .
4. ~~Integración~~ de los modelos de aprendizaje de máquinas ya existentes con la aplicación [21, 14].

Este documento se divide en 5 capítulos: el primero dedicado al estado del arte donde analizan las plataformas similares más conocidas del mercado. En el segundo capítulo se expone la propuesta de lo que se quiere conseguir como producto final. El tercer capítulo se explican los detalles teóricos de la plataforma como la arquitectura usada, casos de uso de los usuarios y modelo de datos. En el cuarto capítulo se expondrán los detalles de implementación, tecnologías utilizadas y funcionamiento. Y por último, en el quinto capítulo las pruebas de funcionalidad llevadas a cabo.

[1]Se refiere a la parte de una aplicación o sistema que se ocupa de la interacción y presentación de la interfaz de usuario. Es la capa visible y accesible para los usuarios finales.[8]

[2]Se refiere a la parte del software que se ejecuta en el servidor y los usuarios finales no ven. Es la capa que se encarga de procesar las solicitudes, manipular los datos y enviar una respuesta a la interfaz de usuario.[8]

[3]Es una interfaz de programación de aplicaciones que se adhiere a los principios del estilo de arquitectura REST (Transferencia de Estado Representacional)[20]

Capítulo 1

Estado del Arte

A continuación se expondrán algunas de las plataformas web que se utilizan en la actualidad en el campo de la dermatología. Se analizarán cada una de las funcionalidad que brindan así como sus desventajas y ventajas para el personal médico cubano.

VisualDx

Esta aplicación web es una herramienta diagnóstica visual que ayuda a los médicos a identificar lesiones cutáneas. Incluye información detallada sobre más de 1,200 diagnósticos dermatológicos y 30,000 imágenes. [Link].

Desventajas

- Posee un plan gratuito de un mes, después de eso es un servicio de suscripción que puede resultar costoso para algunos médicos, especialmente en países donde los ingresos pueden ser más bajos, como Cuba.
- No está disponible en español, esto podría ser una barrera para los médicos que no hablan inglés con fluidez.

SkinVision

Esta aplicación móvil utiliza inteligencia artificial para analizar imágenes de lunares y manchas en la piel y evaluar el riesgo de cáncer de piel. Los usuarios pueden tomar fotos de las lesiones y recibir una evaluación en línea. [Link].

Desventajas

- Está orientada al uso del paciente, que a pesar que en otros casos puede ser un beneficio, no es lo que se está buscando ya que se espera que sea orientado a

ayudar al personal médico profesional.

- Es un servicio que tiene un costo bastante alto para un médico cubano.
- Dado que esta orientada al uso directo del usuario y emplea herramientas de inteligencia artificial para dar diagnósticos, puede provocar pánico en los pacientes si ocurre un falso positivo.

First Derm

Esta aplicación móvil ofrece a los usuarios una evaluación en línea de cualquier problema de la piel que tengan. Los usuarios pueden tomar fotos de la lesión y enviarlas a un dermatólogo certificado para recibir un diagnóstico en línea. [Link].

Desventajas

- También es un servicio que tiene un costo bastante alto para un médico cubano.
- También está orientada al uso directo de los pacientes a pesar de que tienen una interacción más directa con un dermatólogo en línea según se plantea.

DermEngine

Esta aplicación web utiliza tecnología de inteligencia artificial para ayudar a los médicos a detectar lesiones de piel sospechosas. Incluye una herramienta de detección de cambios en lesiones, una biblioteca de imágenes de dermatología, es capaz de compartir pacientes y otros beneficios. [Link].

Desventajas

- También es un servicio que tiene un costo bastante alto para un médico cubano.

Conclusiones del Capítulo

Después de un análisis detallado de cada uno de los sitios con mas presencia en el mercado, se evidencia que no son de fácil acceso, están dirigidas directamente al paciente, no llevan un registro de estos o no tienen bien establecida una forma de compartir información entre doctores.

Finalmente el que pareció mas conveniente fue DermEngine por poseer casi todos los requerimientos, pero al final, como todos los productos analizados no puede ser utilizado por los médicos cubanos producto al costo de subscripción.

Capítulo 2

Propuesta

Se propone el desarrollo de un sitio web con tecnologías modernas y de código abierto, al cual los médicos cubanos tendrán acceso de manera gratuita, permitiéndole llevar un registro de sus pacientes e información relacionada a estos. Facilitando el intercambio de datos entre doctores, así como el acceso a herramientas de apoyo al diagnóstico dermatoscópico como los modelos de aprendizaje de máquinas previamente implementados [21, 14].

Las principales funcionalidades serían:

- **Solicitud de registro:** Los nuevos usuarios en la página podrán solicitar el registro a partir de una sección de la página enviando sus datos. Estos serán notificados por correo de ser aprobados.
- **Inicio de sesión:** Los usuarios tendrán la posibilidad de autenticarse en la página, ganando acceso a sus datos y pacientes.
- **Cambiar contraseña:** Los usuarios podrán cambiar su contraseña de autenticación desde su correo o una vez iniciada sesión.
- **Registrar, modificar y eliminar pacientes :** Los usuarios comunes tendrán la posibilidad de administrar sus pacientes en la plataforma.
- **Registrar, modificar y eliminar lesiones:** Será posible por cada paciente administrar las lesiones asociadas a estos con sus datos y características.
- **Registrar y eliminar Imágenes:** Por cada lesión será posible registrar periódicamente imágenes clínicas y dermatoscópicas, con el objetivo de dar seguimiento y comprobar la efectividad de tratamientos. También será posible la eliminación de estas.

- **Herramientas sobre imágenes:** Será posible el uso de herramientas sobre las imágenes tales como clasificadores o segmentadores de lesiones para el apoyo al especialista.
- **Compartir Lesiones:** Existirá la posibilidad de que un doctor comparta una lesión a otro, permitiéndole acceder a la información. Será posible dejar de compartir en todo momento.
- **Diagnosticar:** Será posible para un doctor que tenga acceso a una lesión dar un diagnostico de esta basándose en los datos a los que tiene acceso.
- **Listar:** El usuario podrá listar sus pacientes, lesiones que ha compartido y lesiones a las que tiene acceso.
- **Usuario administrador:** Podrán existir usuarios administradores que se crearán desde la administración del sitio los cuales podrán validar o denegar nuevos usuarios, además de tener acceso al panel de configuración y la libre modificación de la base de datos.

Capítulo 3

Solución Teórica-Computacional

En este capítulo se conocerá la estructura de la aplicación a nivel teórico así como las acciones que pueden realizar los usuarios en la aplicación mediante un diagrama de casos de uso.

Casos de usos de los usuarios

Existen dos tipos de usuarios:

- El **usuario común**, el cual esta destinado a que lo usen médicos que accedan a la plataforma, permitiéndole el uso y acceso a las funcionalidades que se ofrecen.

  Este usuario puede:

  - Iniciar sesión.
  - Registrar/Editar/Eliminar sus pacientes.
  - Registrar/Editar/Eliminar lesiones de sus pacientes.
  - Registrar/Eliminar imágenes en las lesiones de sus pacientes.
  - Compartir lesiones de sus pacientes a otros usuarios.
  - Acceder a los datos de sus pacientes.
  - Acceder a los datos de las lesiones que otros usuarios (doctores) compartieron a este.
  - Cambiar su contraseña.

- El **usuario administrador**, el cual puede ser médico. Está dirigido a la administración del sitio, tiene acceso a todas las operaciones CRUD[1] del sistema

[1]Operaciones de creación, obtención, actualización y eliminación en la base de datos (Create-Retrieve-Update-Delete por sus siglas en inglés)

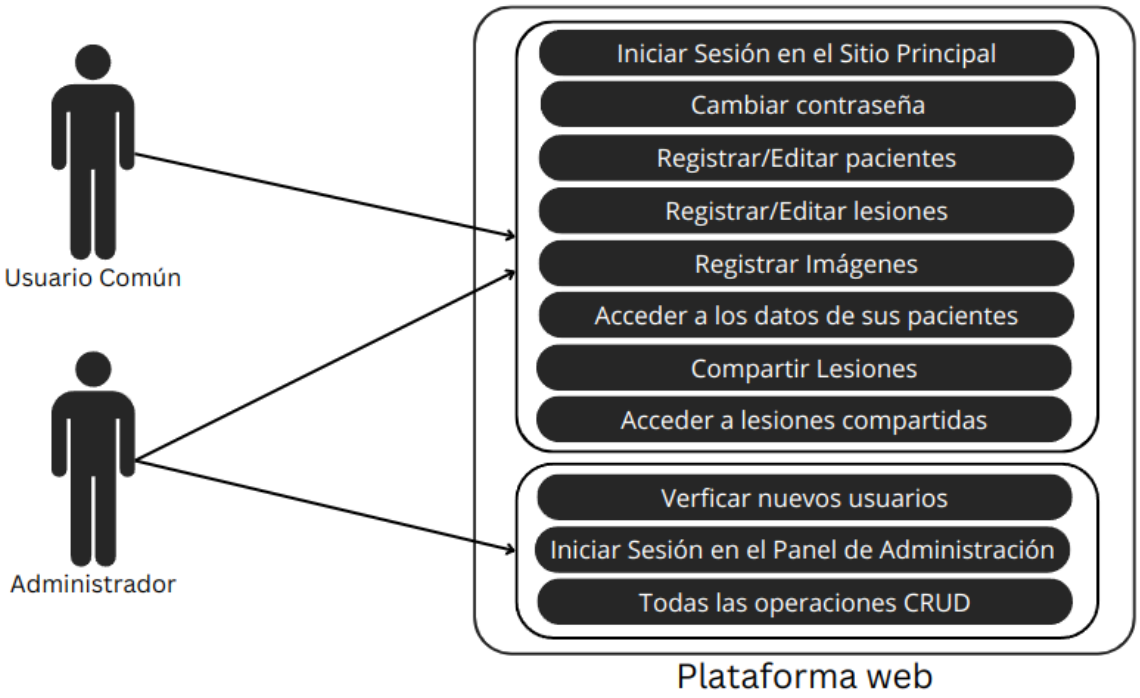

Figura 3.1: Diagrama de casos de uso de los usuarios

usando el panel de administración así como verificar el registro de nuevos usuarios que hayan solicitado la entrada a la página.

Arquitectura de software utilizada

Se trabaja con una arquitectura **cliente-servidor**[2] la cual se basa en la idea de que el cliente solicita servicios o recursos al servidor, y este último los proporciona en respuesta a las peticiones recibidas.

El cliente es la entidad que inicia la comunicación enviando solicitudes al servidor. El cliente puede ser cualquier dispositivo o aplicación que requiera servicios o recursos, como una computadora personal, un teléfono móvil o una aplicación web (como es nuestro caso). El cliente está a cargo de enviar las solicitudes al servidor y procesar las respuestas recibidas.

El esquema de funcionamiento de un sistema según esta arquitectura es:

1. El Cliente solicita una información al Servidor.
2. El Servidor recibe la petición.
3. El Servidor procesa dicha solicitud.

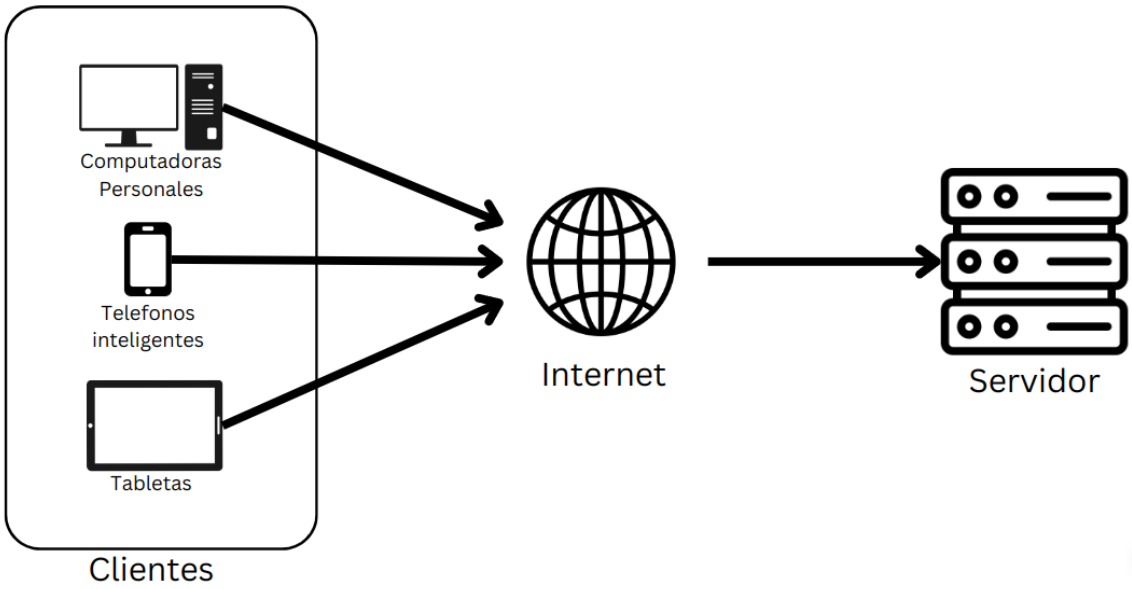

Figura 3.2: Arquitectura cliente-servidor

4. El Servidor envía el resultado obtenido al Cliente.
5. El Cliente recibe el resultado y lo procesa.

Arquitectura del servidor utilizada

La sección conocida como backend de la aplicación que se explicará más adelante implementa una variación de **Model-View-Controller(MVC)**[13] que es la utilizada por Django[4], una de las tecnologías principales que se utilizó en el desarrollo del sitio, la cual también se explicará mas adelante.

Este patrón divide una aplicación en tres componentes principales: el modelo (Model), la vista (View) y el controlador (Controller). Cada uno de estos componentes tiene un propósito y responsabilidades específicas dentro de la arquitectura.

1. **Modelo (Model):** El modelo representa la estructura de datos de la aplicación. Es responsable de almacenar, manipular y acceder a los datos subyacentes. Esto podría incluir la interacción con una base de datos, la gestión de archivos o cualquier otra fuente de datos. El modelo encapsula la lógica necesaria para crear, leer, actualizar y eliminar los datos. También puede contener validaciones y reglas de negocio relacionadas con la gestión de los datos.
2. **Vista (View):** La vista se encarga de la presentación de la interfaz de usuario. Es responsable de mostrar los datos al usuario y recibir las interacciones del

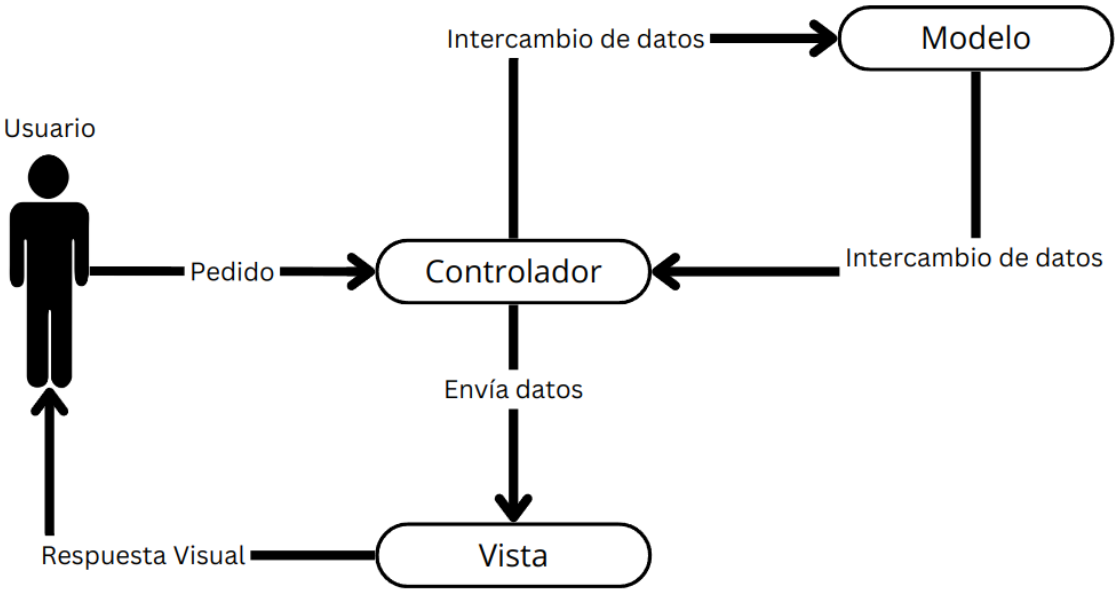

Figura 3.3: Arquitectura Modelo-Vista-Controlador

usuario, como clics de botones o entradas de formularios (en el caso de que se espere interacción). La vista no realiza lógica de negocio ni manipulación de datos; simplemente muestra la información proporcionada por el modelo y notifica al controlador sobre las acciones del usuario.

3. **Controlador (Controller):** El controlador actúa como intermediario entre el modelo y la vista. Recibe las interacciones del usuario desde la vista y las procesa, determinando qué acciones llevar a cabo en el modelo. Se encarga de actualizar el modelo según las solicitudes del usuario y de notificar a la vista sobre los cambios relevantes en los datos. También puede manejar la lógica de flujo de la aplicación, coordinando las interacciones entre el modelo y la vista.

La comunicación entre los componentes se realiza siguiendo un flujo unidireccional. La vista se comunica con el controlador para notificar eventos de usuario, y el controlador actualiza el modelo según sea necesario. A su vez, el modelo notifica al controlador sobre cambios en los datos, y el controlador actualiza la vista para reflejar esos cambios.

La arquitectura MVC promueve la separación de preocupaciones y la modularidad en el diseño de aplicaciones. Esto permite una mayor flexibilidad, mantenimiento y reutilización de código.

Modelo de Datos

Para el desarrollo de la base de datos se utilizó el ORM[15] de Django el cual nos permite mediante el uso de clases de Python la creación y modelado de esta. De estas clases denominadas modelos, se crean las siguientes tablas en la base de datos:

- **CustomUser:** El usuario del sistema, es el usuario creado por Django levemente modificado para poder identificar si este es Doctor o está verificado además de tener campos de información de usuario extra.

  Campos:

  - `email`, `first_name`, `last_name` y `is_staff` son campos creados por Django automáticamente en el usuario, estos representan el email, nombre, apellidos y si un usuario es puede entrar al panel de administración de Django respectivamente.
  - `is_doctor`: Si el usuario es doctor.
  - `provincia`: La provincia a la que pertenece el usuario.
  - `municipality`: El municipio al que pertenece el usuario.
  - `reg_professional`: Registro profesional del doctor.
  - `work_place`: Centro de trabajo del doctor.
  - `verified`: Si ha sido verificado en la plataforma el usuario.

- **Pacient:** Un paciente asociado a un doctor.

  Campos:

  - `doctor`: El doctor que registró al paciente en la plataforma.
  - `name`, `last_name`, `years_old`, `sex`: El nombre, apellidos, edad y sexo del paciente respectivamente.
  - `register_date`: La fecha de registro del paciente en la plataforma.
  - `app`: Antecedentes patológicos del paciente.

- **Injury:** Una lesión asociada a un paciente.

  Campos:

  - `pacient`: El paciente asociado al que pertenece la lesión.
  - `phototype`: Fototipo cutáneo.
  - `clinical_diagnosis`: Diagnóstico clínico de la lesión.
  - `histopathological_diagnosis`: Diagnóstico histopatológico de la lesión.

  - `localization`: Localización de la lesión.
  - `comentary`: Comentario dado por el doctor sobre datos no comprendidos en los campos de la tabla que puedan ser de interés.

- **Image:** Una imagen dermatoscópica y una clínica de la lesión en cuestión.

  Campos:

  - `image`: La url de la imagen dermatoscópica en el servidor.
  - `image_thumbnail`: La url de una versión de la imagen dermatoscópica con calidad reducida, esta imagen es generada automáticamente.
  - `image_simple`: La url de la imagen clínica en el servidor. Registrar esta imagen no es obligatorio para crear la fila en la tabla.
  - `image_simple_thumbnail`: La url de una versión de la imagen clínica con calidad reducida, esta imagen es generada automáticamente.
  - `date`: Fecha de registro de la imagen.
  - `commentary`: Un comentario dado por el doctor a la hora de registrar la imagen.

- **Visibility:** Una terna doctor-doctor-lesión, representa el acceso dado por un doctor a otro a una lesión

  Campos:

  - `owner_doctor`: El doctor que compartió la lesión de uno de sus pacientes a otro.
  - `doctor`: El doctor al que se le compartió la lesión.
  - `injury`: La lesión que se esta compartiendo.

- **Diagnosis:** El diagnóstico dado por un doctor a una lesión.

  Campos:

  - `date`: Fecha en la cual se creó el diagnóstico.
  - `doctor`: El doctor que dió el diagnóstico.
  - `injury`: La lesión a la cual se le está dando el diagnóstico.
  - `diagnosis_result`: El diagnóstico dado por el doctor en cuestión.

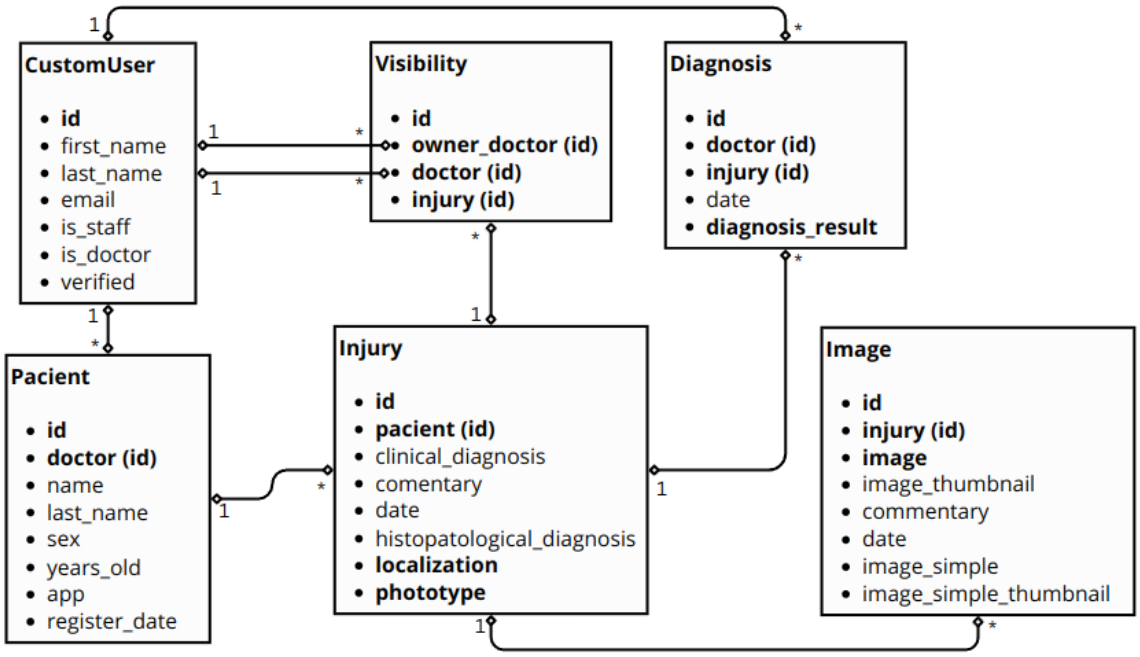

Figura 3.4: Diagrama del modelo de datos

Capítulo 4

Detalles de Implementación

En este capítulo se verán las principales tecnologías usadas en el desarrollo de la plataforma así como detalles de implementación.

La aplicación se compone de dos partes fundamentales, la aplicación servidor (o backend[8]) y la aplicación cliente (o frontend[8]).

Backend

Por **backend** se refiere a la parte del software que se ejecuta en el servidor y los usuarios finales no ven. Es la capa que se encarga de procesar las solicitudes, manipular los datos y enviar una respuesta al **frontend** o interfaz de usuario.

Sería en este caso la parte que implementa una variante de la arquitectura MVC como se explicó en el Capítulo-3.

Se implementó una **API REST** [20] encargada de brindar los datos al lado del cliente a través del protocolo HTTP[9] cada vez que este los solicite, usualmente en formato **JSON**[11]. Es independiente del tipo de plataforma o lenguaje de programación, lo que hace que la integración entre diferentes sistemas sea más fácil permitiendo a largo plazo llevar la lógica de la aplicación a distintos tipos de clientes, como aplicaciones móviles o de escritorio.

Python

Python[18] es un lenguaje de programación de alto nivel, interpretado y de propósito general, que se destaca por su legibilidad y simplicidad. Fue creado por Guido van Rossum y lanzado por primera vez en 1991. Python es muy popular en una variedad de aplicaciones, desde desarrollo web hasta ciencia de datos, aprendizaje automático y más.

La razón principal de haberlo seleccionado como lenguaje principal del servidor es que los modelos ya existentes que se usan para hacer predicciones de lesiones están

escritos en este lenguaje, lo que permite una integración sencilla. También la existencia de múltiples herramientas especializadas (como se verán más adelante) destinadas a la construcción de servidores para sitios web.

Django

Django[4] es un framework de desarrollo web de alto nivel y de código abierto basado en Python. Proporciona un conjunto de herramientas y funcionalidades predefinidas que simplifican la creación de aplicaciones web seguras y escalables. Incluye un ORM[1] para interactuar con la base de datos, un sistema de enrutamiento de URL[2], un sistema de plantillas y un sistema de autenticación y autorización.

Django sigue el principio de diseño "baterías incluidas", lo que significa que viene con muchas funcionalidades incorporadas, como administración de usuarios, manejo de formularios, internacionalización y manejo de archivos estáticos.Además, promueve buenas prácticas de desarrollo web, como la separación estricta entre la lógica del negocio y la presentación, siguiendo una variación del patrón de diseño Modelo-Vista-Controlador (MVC) el cual se explicó en el Capitulo-3.

Fue seleccionado para este proyecto principalmente por el manejo de la base datos con su ORM no necesitando de bibliotecas extras para este propósito, además de la comunidad y gran cantidad de documentación existente.

Django Rest Framework

Django Rest Framework[5] es una biblioteca de Python de código abierto que se utiliza para construir API web de manera rápida y sencilla utilizando el framework Django. Proporciona un conjunto de herramientas y utilidades que simplifican el desarrollo de una **API RESTful**[20]. Se integra perfectamente con Django y aprovecha su potencia y flexibilidad, proporcionando características como la serialización automática de modelos en formatos como JSON[11], validación de datos, autenticación, autorización y manejo de vistas basadas en clases.

Fue seleccionado para este proyecto debido a la decisión de crear una API para el intercambio de datos entre las múltiples aplicaciones clientes con el servidor y ser la mejor de las tecnologías al integrarse con Django para este propósito.

Djoser

Djoser[6] es una biblioteca de autenticación de código abierto que proporciona una solución lista para usar y altamente personalizable para implementar la autenticación

[1] Object-Relational Mapping por sus siglas en inglés[15]

[2] Uniform Resource Locators por sus siglas en inglés [26]

de usuarios, la gestión de tokens de acceso y otras funcionalidades relacionadas con la seguridad en una aplicación Django.

Se añadió pues al ser mantenido por la comunidad hay menos probabilidades de que exista una falla de seguridad que en un sistema de autenticación desarrollado desde cero para este proyecto.

PostgreSQL

PostgreSQL[17] es un sistema de gestión de bases de datos relacional de código abierto, altamente potente y confiable. Se destaca por su capacidad para manejar grandes volúmenes de datos, su flexibilidad y su enfoque en la integridad de los datos.

Fue seleccionado principalmente sobre las demas tecnologías del mercado por ser una alternativa de código abierto.

Frontend

Por **frontend**[8] se refiere a la parte de una aplicación o sistema que se ocupa de la interacción y presentación de la interfaz de usuario. Es la capa visible y accesible para los usuarios finales. Se encarga de todo lo relacionado con la presentación y la interacción con el usuario, incluyendo la estructura, el diseño, la disposición de los elementos visuales y la manipulación de eventos.

En este caso se creó una aplicación web del tipo **Single Page Application(SPA)**[22] que funciona cargando una sola página y proporciona una experiencia de usuario fluida y dinámica, similar a una aplicación de escritorio. A diferencia de las aplicaciones web tradicionales que cargan nuevas páginas completas al navegar entre secciones, una SPA carga solo los recursos necesarios y actualiza el contenido de forma dinámica sin necesidad de recargar la página.

Se optó por una aplicación web como cliente sobre una de móvil o de escritorio por las ventajas multiplataforma que ofrecen los navegadores, permitiendo que desde cualquier dispositivo con acceso a internet se pueda acceder a la aplicación.

TypeScript

TypeScript[25] es un lenguaje de programación de código abierto desarrollado por Microsoft. Es una extensión de JavaScript[10] que agrega características de tipado estático principalmente y otros elementos de programación orientada a objetos al lenguaje base. Permite asignar tipos a variables, parámetros de función y valores de retorno, lo que ayuda a detectar errores y proporciona un mayor nivel de seguridad y claridad en el código. El compilador de TypeScript realiza verificaciones estáticas de tipos y emite errores si se encuentra alguna discrepancia.

Vue.js

Vue.js[27] es un framework de Javascript[10] de código abierto utilizado para construir interfaces de usuario interactivas y reactivas en aplicaciones web. La característica principal, es su enfoque en la capa de vista de una aplicación web. Utiliza una sintaxis declarativa y basada en componentes para crear y reutilizar elementos de interfaz de usuario. Los componentes de Vue.js son bloques autónomos de código que encapsulan la estructura, el estilo y la lógica de una parte de la interfaz de usuario.

Una de las características clave de Vue.js es su capacidad para manejar el enrutamiento mediante **Vue Router**[28]. Es una biblioteca oficial de Vue.js que permite crear rutas y navegar entre diferentes vistas de forma fácil y elegante. Proporciona una manera sencilla de definir las rutas de una aplicación y gestionar la navegación, lo que facilita el desarrollo de aplicaciones de una sola página (Single Page Applications) con múltiples vistas.

Además, Vue.js también se beneficia de **Pinia**[16], una biblioteca de estado con una sintaxis más moderna y basada en funciones. Pinia proporciona una gestión de estado centralizada para las aplicaciones basadas en Vue.js, lo que facilita el almacenamiento, la actualización y el acceso a los datos de la aplicación. Con Pinia se puede organizar y compartir el estado de manera eficiente en toda la aplicación, lo que mejora la escalabilidad y la mantenibilidad del código.

A pesar de que existen otras librerías igual de potentes en el mercado, se seleccionó esta principalmente por tener bastante tiempo siendo exitosa pero no siendo lo suficientemente antigua como para que pueda quedarse obsoleta, esto sumado a la gran popularidad, comunidad y documentación, pareció buena selección para el proyecto.

Tailwind

TailwindCSS[23] es un framework de diseño de código abierto utilizado para construir interfaces de usuario en aplicaciones web. A diferencia de otros frameworks, en lugar de proporcionar componentes predefinidos, TailwindCSS se basa en un enfoque de utilidad de clases.

Funcionamiento

La SPA es la encargada de mostrar el contenido y estructura de la página, mientras que la API REST[20] será la encargada de proporcionar los datos a esta usando el protocolo HTTPS[7]. La comunicación seguirá la arquitectura hablada en el Capitulo-3, la SPA hace una petición a alguna ruta[7] de la API usando algún método HTTP[12], como **GET** o **POST** y esta dará una respuesta basada en la misma ruta, argumentos de esta o cuerpos de las solicitudes. Algunos ejemplos de solicitudes de la SPA a la

API pueden ser la edición de campos en la base de datos, eliminación, inserción o simplemente obtención de datos para mostrarlos al usuario final a través de la interfaz de usuario del lado del cliente.

Seguridad

Djoser[6] trae implementado el uso de **tokens**[24] de autenticación, los cuales son pedidos por el lado del cliente inicialmente y van a ser usados por este para autenticarse en la aplicación, permitiéndole al lado del servidor saber en todo momento con quien se comunica, dando contenido personalizado según las peticiones que se hagan y las características del usuario.

Para llevar a cabo la autenticación, cada pedido HTTP que haga el cliente a la API debe contener el **header**[19]:

```
Authorization: Token <token>
```

Siendo `<token>` el token que identifica a el usuario. Este token puede ser obtenido por el cliente accediendo a un **endpoint**[7], el cual, en nuestro caso es `/api/v1/auth/token/login`, es necesario enviar el email y la contraseña del usuario en el cuerpo de esta petición[3].

Aplicaciones de Django Utilizadas

Cada aplicación[1] de Django es un paquete de Python que brinda características específicas. Estas pueden ser reutilizadas en varios proyectos. Las aplicaciones suelen ser una combinación de modelos, vistas, archivos estáticos y templates (en caso de usar las templates de django).

Core

Es la aplicación principal que posee todos los modelos y vistas principales del proyecto, así como todas las herramientas para su correcto funcionamiento. A continuación se muestran los endpoints[7] o vistas que **core** brinda al cliente con su método HTTP[12].

Destinado a el usuario:

- **GET** `/api/v1/profile/` da información basica del doctor logueado en el sistema.

Destinados a pacientes:

- **GET** `/api/v1/pacients/` lista todos los pacientes del doctor logueado en el sistema.

- **POST** `/api/v1/pacients/create/` registra un paciente en el sistema a partir de los datos brindados.
- **GET** `/api/v1/pacients/<pk>` detalles de el paciente con llave primaria `<pk>`
- **PATCH** `/api/v1/pacients/<pk>/modify` modifica el paciente con llave primaria `<pk>` a partir de los datos brindados.

Destinados a lesiones:

- **POST** `/api/v1/injuries/create/` registra una lesión en el sistema a partir de los datos brindados.
- **GET** `/api/v1/injuries/<pk>` detalles de la lesión con llave primaria `<pk>`
- **PATCH** `/api/v1/injuries/<pk>/modify` modifica la lesión con llave primaria `<pk>` a partir de los datos brindados.
- **GET** `/api/v1/injuries/<pk>/images` lista las imágenes de la lesión con llave primaria `<pk>`
- **POST** `/api/v1/injuries/upload-image` registra una imagen dermatoscópica asociada a una lesión y otros datos brindados, puede contener también una imagen clínica.
- **GET** `/api/v1/injuries/visible/` lista las lesiones a las que el doctor logueado en el sistema tiene acceso que no son de pacientes propios.
- **GET** `/api/v1/visibilities/given/` lista las visibilidades que el doctor logueado en el sistema ha dado.
- **POST** `/api/v1/visibilities/create` crea una terna de visibilidad dado los datos brindados y el doctor logueado.
- **GET** `/api/v1/images/<pk>` los datos de la imagen con llave primaria `<pk>`

Destinados a imágenes:

- **GET** `/api/v1/images/<pk>/img` la imagen dermatoscópica con llave primaria `<pk>` importante señalar que no es un JSON, el servidor responde con la imagen.
- **GET** `/api/v1/images/<pk>/thumbnail` la imagen dermatoscópica con llave primaria `<pk>` con una calidad muy reducida, importante señalar que no es un JSON, el servidor responde con la imagen.

- **GET** `/api/v1/images/<pk>/img-simple` la imagen clínica con llave primaria `<pk>` importante señalar que no es un JSON, el servidor responde con la imagen.
- **GET** `/api/v1/images/<pk>/simple-thumbnail` la imagen clínica con llave primaria `<pk>` con una calidad muy reducida, importante señalar que no es un JSON, el servidor responde con la imagen.

Destinados al administrador:

- **GET** `/api/v1/users/<pk>/confirm` orientado a los usuarios Administradores, confirma la cuenta de un doctor que se registró en la plataforma para que la pueda empezar a usar.
- **GET** `/api/v1/users/<pk>/remove` orientado a los usuarios Administradores, desactiva la cuenta de un doctor que se registró en la plataforma, no la podrá usar.

BasicModels

Es una aplicación que brinda endpoints[7] para el uso de modelos básicos de aprendizaje de máquinas como el un modelo de clasificación de lesiones [14] y un modelo de segmentación de imágenes [21].

Endpoints disponibles en esta aplicación:

- **GET** `/api/v1/models/classify-image/<pk>` da una predicción del tipo de lesión que está en la imagen con llave primaria `<pk>`.
- **GET** `/api/v1/models/segment-image/<pk>` da una imagen con la segmentación de la lesión con llave primaria `<pk>`.

Agregar Nuevas Aplicaciones

Es útil la creación de nuevas aplicaciones de Django con el objetivo de ampliar las funcionalidades del sitio. Algunas posibles funcionalidades que se pueden agregar son otros modelos mas específicos de predicción y de preprocesamiento de imágenes o para extender la estructura de la base de datos y la posibilidad de hacer algo con estos nuevos datos.

Siguiendo la documentación[1] que provee Django es posible añadir nuevas aplicaciones agregandolas a la carpeta principal del proyecto del lado del servidor y seguidamente modificar el archivo `/dermabackend/settings.py` (el cual posee todas los ajustes generales del proyecto de Django) agregandole la aplicación a `INSTALLED_APPS` quedando de la siguiente manera

```
# archivo /dermabackend/settings.py

#...
INSTALLED_APPS = [
    'nueva_applicacion'
    # ...
]
#...
```

Siendo `nueva_aplicacion` el valor del campo `name` dentro de la clase que hereda de `AppConfig` en el archivo `apps.py` en la ruta de la nueva aplicación y también debe ser el nombre de la carpeta que contiene la aplicación. Un ejemplo de este archivo puede ser

```
# archivo  /nueva_aplicacion/apps.py

from django.apps import AppConfig

class NuevaAplicacionConfig(AppConfig):
    name = 'nueva_aplicacion'
```

En caso de que la nueva aplicación contenga endpoints[7] que se quieran incluir en el proyecto, estos deben estar definidos en el archivo `urls.py` de la aplicación, para luego ser llamados importados desde el archivo `urls.py` general del proyecto (`/dermabackend/urls.py`) de la siguiente manera:

```
# archivo  /dermabackend/urls.py

from django.urls import path, include

urlpatterns = [
    # ...
    path('api/v1/nueva_aplicacion/', include('nueva_aplicacion.urls'
    ))
    #...
]
```

Conclusiones

Se puede concluir que se logró el cumplimiento de los objetivos iniciales, teniendo un software funcional y extensible que cumple los requerimientos del cliente. Se hizo un estudio del estado del arte alrededor de las web de ayuda dermatoscópica similares al producto buscado, beneficios y defectos de estas. Se desarrollaron una Single Page Application y una API Rest siguiendo la arquitectura cliente-servidor entre ellas. Se hizo uso únicamente de herramientas de código abierto y quedó la posibilidad de integración futura de otras aplicaciones del lado del cliente con esta API Rest, como aplicaciones de escritorio o móviles.

Recomendaciones

Recomendaciones

Bibliografía

[1] *Aplicaciones de Django*. URL: https://docs.djangoproject.com/en/4.2/ref/applications/ (vid. págs. 18, 20).

[2] *Cliente-Servidor*. URL: https://en.wikipedia.org/wiki/Client%E2%80%93server_model (vid. pág. 8).

[3] *Cuerpo de la petición en POST*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST (vid. pág. 18).

[4] *Django*. URL: https://www.djangoproject.com (vid. págs. 9, 15).

[5] *Django Rest Framework*. URL: https://www.django-rest-framework.org/ (vid. pág. 15).

[6] *Djoser*. URL: https://djoser.readthedocs.io/en/latest/ (vid. págs. 15, 18).

[7] *Endpoint*. URL: https://www.cloudflare.com/learning/security/api/what-is-api-endpoint/ (vid. págs. 17, 18, 20, 21).

[8] *Frontend and Backend*. URL: https://en.wikipedia.org/wiki/Frontend_and_backend (vid. págs. 2, 14, 16).

[9] *HTTP*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP (vid. pág. 14).

[10] *Javascript*. URL: https://developer.mozilla.org/en-US/docs/Web/javascript (vid. págs. 16, 17).

[11] *JSON*. URL: https://www.json.org/json-en.html (vid. págs. 14, 15).

[12] *Metodos HTTP*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods (vid. págs. 17, 18).

[13] *Model-View-Controller*. URL: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller (vid. pág. 9).

[14] Claudia Olavarrieta. «Ensemble of convolutional neural networks to aid diagnosis of skin cancer». En: (2022) (vid. págs. 2, 5, 20).

[15] *ORM*. URL: https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping (vid. págs. 11, 15).

[16] *Pinia*. URL: https://pinia.vuejs.org (vid. pág. 17).

[17] *Postgresql*. URL: https://www.postgresql.org (vid. pág. 16).

[18] *Python*. URL: https://www.python.org (vid. pág. 14).

[19] *Request Header*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers (vid. pág. 18).

[20] *Rest API*. URL: https://www.ibm.com/topics/rest-apis (vid. págs. 2, 14, 15, 17).

[21] Marcos Adrián Valdivié Rodríguez. «Segmentación de imágenes dermatoscópicas utilizando la arquitectura U-Net». En: (2022) (vid. págs. 2, 5, 20).

[22] *SPA*. URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (vid. pág. 16).

[23] *TailwindCSS*. URL: https://tailwindcss.com/ (vid. pág. 17).

[24] *Tokens de autenticación*. URL: https://www.cloudflare.com/learning/access-management/token-based-authentication/ (vid. pág. 18).

[25] *Typescript*. URL: https://www.typescriptlang.org (vid. pág. 16).

[26] *URL*. URL: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_URL (vid. pág. 15).

[27] *Vue.js*. URL: https://vuejs.org (vid. pág. 17).

[28] *VueRouter*. URL: https://router.vuejs.org (vid. pág. 17).