Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Mejorando el rendimiento y características de una aplicación web para la evaluación automática del glaucoma.

Improving performance and features of a web app for automatic glaucoma assesment

Pablo Quintín Hernández Caracena

La Laguna, 14 de junio de 2022

D. **José Ignacio Estévez Damas**, con N.I.F. 43.786.097-P profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. **Francisco José Fumero Batista**, con N.I.F. 45.731.321-F doctorando adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

”*Mejorando el rendimiento y características de una aplicación web para la evaluación automática del glaucoma*”

ha sido realizada bajo su dirección por D. **Pablo Quintín Hernández Caracena**, con N.I.F. 79.071.650-L.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 14 de junio de 2022

Agradecimientos

En primer lugar, me gustaría agradecer a todas las personas que me han acompañado durante estos años en esta etapa de mi vida, compañeros, familiares y amigos.

También me gustaría agradecer a mi tutor y cotutor, José Ignacio Estévez Damas y Francisco José Fumero Batista, por darme la oportunidad de participar en este proyecto y permitirme aprender tanto.

Licencia

Resumen

El desarrollo de este proyecto parte de la necesidad de actualizar, debido a un cambio en los requisitos del sistema, una aplicación web que permite al médico el acceso a una red neuronal que proporciona el diagnóstico de glaucoma a través de una imagen del globo ocular. Esta aplicación web, además de facilitarle el juicio al profesional mediante la explotación de una red neuronal convolucional ya entrenada, nos permite seguir obteniendo nuevas imágenes con las que mejorar las predicciones del sistema. Las mejoras se pueden englobar en tres vertientes: adquisición y utilización de los datos para la producción de redes neuronales orientadas al diagnóstico del glaucoma; mejoras técnicas en el funcionamiento global de la aplicación web y en el servicio de diagnóstico proporcionado, adaptándola a las nuevas versiones de librerías, sistemas de bases de datos alternativos y nuevas características; y, por último, la experiencia de usuario.

Palabras clave: glaucoma, diagnóstico, aplicación web, red neuronal convolucional, librerías, base de datos, globo ocular, experiencia de usuario.

Abstract

This project is born from the necessity to update, due to a change in its requirements, a web application that gives a doctor access to a neural network that provides an automatic diagnosis of glaucoma through an eyeball's image. This web application, on top of facilitating the professional its judgement through the trained convolutional neural network exploitation, it grants us the ability of aquiring more images to improve our automatic diagnosis system. The changes can be grouped to three different categories: glaucoma diagnosis oriented neural network's data gathering and exploitation, tecnical improvements in the global web application functioning and the diagnosis service that has been provisioned, adapting them to new framework versions, alternative database systems and new features; and, finally, the user experience.

Keywords: glaucoma, diagnosis, web application, convolutional neural network, frameworks, databse, eyeball, user experience

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Objetivos

El diagnóstico automático de enfermedades es una práctica cada vez más utilizada por profesionales sanitarios. En la actualidad una de las técnicas con más éxito se basa en el aprendizaje profundo ("Deep learning"Goodfellow et al. (2016)), donde un modelo matemático conocido como red neuronal con millones de parámetros es “entrenado” para que pueda responder adecuadamente a un tipo de entrada, por ejemplo, una imagen médica. En el campo del diagnóstico automático, la respuesta de la red suele ser una propuesta de clasificación, directamente relacionada con el diagnóstico.

Por lo tanto, una red neuronal no es más que un modelo computacional que realiza operaciones sobre los datos que recibe, permitiéndole identificar patrones en nuevos conjuntos de datos. En otras palabras, su intención es resolver problemas de la misma manera que lo haría un cerebro humano.

Para que los médicos puedan confiar en este procedimiento es necesario que las redes neuronales tengan una probabilidad de acierto muy alta, lo que se consigue entrenándolas con una gran cantidad de imágenes bien etiquetadas.

Hay que decir, que en este trabajo no se ha tratado en su totalidad el problema del entrenamiento, sino más bien los problemas de: **explotación** de la red neuronal, validación del sistema de diagnóstico y obtención de nuevos casos con los que perfeccionar los modelos existentes o entrenar otros modelos nuevos.

Así pues, el objetivo principal de este trabajo es poner a disposición del experto médico una red neuronal previamente entrenada, a través de una interfaz de usuario sencilla de usar, tanto para introducir nuevos problemas a diagnosticar como para visualizar los resultados. Para ello, se ha partido de una aplicación web previamente desarrollada Arvelo García (2020), añadiéndose mejoras tanto en el desempeño como en la funcionalidad. Con ellas, el profesional sanitario podrá subir las imágenes oculares que quiera analizar, junto con metadatos asociados a los casos y que son de interés en el diagnóstico de la patología del glaucoma. Además de esto, el usuario tendrá la posibilidad de acceder a un historial para ver todas las imágenes que ha subido ya previamente, así como el diagnóstico de la red y una serie de datos relativos a la imagen que debe introducir el médico al subir el archivo. También se incluyen funcionalidades de cara a facilitar la gestión y explotación de los casos albergados en la base de datos

1.2. Antecedentes y estado actual del sistema

1.2.1. Estado actual del sistema

Este proyecto parte de una aplicación web previa desarrollada en 2020 Arvelo García (2020). En ella se le permite al usuario de una manera muy básica interactuar con una red neuronal. Esta implementación consiste en una aplicación web con una página en la que se sube una imagen de un globo ocular y se enviará al servidor encargado de analizarla, dejando el sistema en espera hasta que se reciba el diagnóstico de la red neuronal. Tras esto el usuario tiene la posibilidad de introducir una serie de datos relativos a la imagen y al diagnóstico, como la edad del paciente o si el diagnóstico automático es correcto.

El sistema utiliza un *frontend* con *React* y *Typescript*, junto con un *backend* que está implementado en *Python* usando el *framework* web *Flask* y *Tensorflow* con *Keras* para el uso de la red neuronal. Además de todo esto, la persistencia de los datos está implementada con una base de datos *NoSQL*, orientada a documentos, específicamente *MongoDB* MongoDB (2009).

1.2.2. Uso de inteligencia artificial en el campo de la medicina

El diagnóstico médico con redes neuronales es una práctica cada vez más extendida en la medicina actual, gracias a los avances tecnológicos se pueden detectar precozmente distintas enfermedades y ayudar así a su tratamiento. Algunos de los ejemplos más importantes para las que se usa este tipo de diagnóstico son: cáncer de mama, cáncer de piel o infarto agudo de miocardio. En cada caso, el uso de las redes neuronales es distinto, se usan desde imágenes hasta resultados de exámenes médicos como mamografías o electrocardiogramas.

A modo de ejemplo, en el caso del cáncer de mama, existe un estudio (“Redes neuronales en el diagnóstico del infarto agudo de miocardio” Sprockel et al. (2014)) en el que se descubre que hay miles de mujeres a las que se les detecta alguna lesión en el pecho y es diagnosticada para que les sea extirpada, cuando en el 90 % de los casos estás anomalías resultan no ser malignas,por ello, para reducir intervenciones innecesarias, se planteó la introducción de esta tecnología en el diagnóstico de la enfermedad. Un equipo del MIT desarrolló una serie de modelos para mejorar la detección y el diagnóstico usando las biopsias como método de decisión, con los que consiguieron reducir el número de intervenciones benignas en más de un 30 % en más de 330 casos.

Otro de los ejemplos del uso de esta técnica es la detección precoz del infarto de miocardio, su diagnóstico es complejo ya que los síntomas son muy variables y cada paciente los percibe de una forma diferente. En un estudio realizado por la Revista Colombiana de Cardiología (“*Using artificial intelligence to improve early breast cancer detection*” Conner-Simons (2017)) se compara el desempeño de diferentes redes neuronales con diferente número de entradas. Para el diseño de las redes, se decidió utilizar un número variable de neuronas desde 5 hasta 18 entre los que se definían distintos tipos de entradas. En las redes con cinco neuronas, como entrada se decidió usar los resultados de los electrocardiogramas realizados a los pacientes, según se aumenta el número de neuronas, se incrementa también las entradas de datos añadiendo diferentes pruebas. Por último, en el caso de 18 neuronas de entrada se utilizan todos los componentes de la escala Braunwald, en la que disponemos de diferentes variables utilizadas para determinar la probabilidad de la enfermedad. Los resultados de este estudio concluyeron que no siempre más es

mejor, la red que otorgó mayor precisión en el diagnóstico fue la de 10 neuronas de entrada, pero, realmente, no hubo una mejora global desde el uso de 6 neuronas de entrada. El único caso que obtuvo resultados inferiores fue la entrada de solamente 5 datos que obtuvo una efectividad del 60 % mientras que los otros se acercaban todos al 99 %.

Esta forma de diagnóstico de enfermedades está probada en muchos más experimentos y esto nos otorga una seguridad a la hora de usarlo, además, nos permite reducir la carga de trabajo del profesional así como conseguir que el diagnóstico pueda ser mucho más rápido y precoz. Por otro lado, tiene algunas desventajas, ya que no tiene una seguridad al 100 %, debe haber un profesional revisando los resultados y confirmando su veracidad ya que se trata de un tema tan delicado como la salud de las personas.

En el caso que nos compete, debemos estudiar el glaucoma una de las enfermedades oculares más comunes entre la población mayor, y de las que tiene un diagnóstico precoz más complejo. En España, afecta casi a un 3 % de la población (Sociedad Española de Glaucoma). Esta es una enfermedad que se ocasiona debido a un aumento en la presión intraocular en el paciente, por culpa de un mal drenaje del humor acuoso del ojo, un líquido producido constantemente en el globo ocular. Si no se evacua correctamente a través del ángulo de drenaje se produce una lesión en el nervio óptico, que se compone por muchas fibras nerviosas muy pequeñas, de manera progresiva. A medida que esas fibras se van deteriorando, van apareciendo puntos ciegos en la visión del paciente, aunque es probable que no se noten hasta que se hayan muerto la mayoría de las fibras, produciendo así la ceguera. Este proceso es irreversible, por ello la importancia de la detección precoz y correcta del mismo. En la figura 1.1 se puede ver un ojo que tiene un ángulo correcto de drenaje, en contra posición con la figura 1.2 en la que se ve cómo el ángulo de drenaje está bloqueado y el ojo no evacúa correctamente el líquido generado.

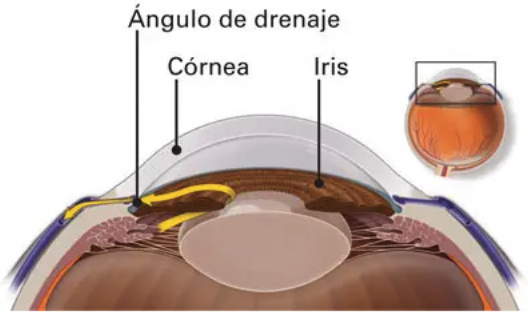

Figura 1.1: Ojo con un correcto ángulo de drenaje

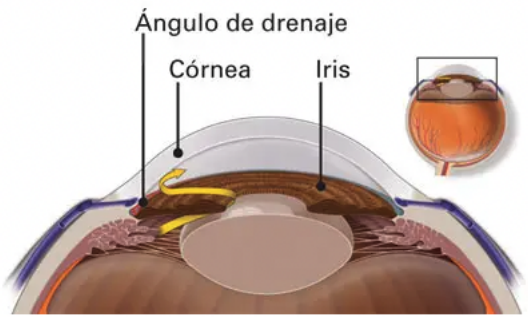

Figura 1.2: Ojo con un ángulo de drenaje bloqueado

En el estudio del glaucoma también ncontramos diferentes artículos sobre el uso de la inteligencia artificial. Por ejemplo, en el estudio titulado "Análisis de la capa de células ganglionares con *deep learning* en el diagnóstico de glaucoma" (Diaz Aleman et al. (2021)), se realizó una comparativa entre diferentes modelos de redes neuronales para discernir cuál es el más óptimo para el estudio de esta enfermedad. Se diagnosticaron los ojos de 298 pacientes con dos modelos de redes neuronales (VGG19 y ResNet50), tras comparar los resultados se obtuvo que, pese a que ambos son óptimos para la valoración sobre la existencia de esta enfermedad debido a su bajo porcentaje de fallos, el modelo VGG19 ofrece una ligera mejora en casi todos los campos estudiados. Aún así se concluyó que ambos ofrecen una alta precisión en el diagnóstico de glaucoma. Por otro lado, tenemos casos de estudios en los que no se prueba si una red neuronal es óptima o no, sino que se centran en disponer de una correcta y completa base de datos con la que entrenar al modelo ("*A unified retinal image database for assessing glaucoma using deep learning.*" Alayon et al. (2020)). En este proyecto se unificó y limpió de casos erróneos, con la ayuda de dos expertos en el tema, tres repositorios en los que se encontraban imágenes para utilizar en el entrenamiento de redes neuronales convolucionales. El resultado fue un *paper* en el que se publica una versión correcta y bien etiquetada, disponible a cualquier persona, de esta base de datos.

Como se puede ver, cada vez el uso de la inteligencia artificial en la medicina es más común, pero para poder tener un buen uso de la misma es crucial disponer de un buen conjunto de datos de garantías, en el que no haya errores en el etiquetado de las mismas. Si a una red neuronal la entrenas con un set de datos en el que haya algún error, la eficacia de esta se reduce enormemente. Además de esto, siempre deben estar correctamente documentadas para el uso para el cuál han sido entrenadas. Aún con todo esto, todos los diagnósticos automáticos deben tomarse como un apoyo al experto en el campo médico y no como un sustituto del mismo, ya que deben siempre ser supervisado por cuantas más personas mejor, pero por lo menos una, para así evitar equivocaciones y también poder seguir etiquetando imágenes de manera correcta y poder seguir entrenando el sistema.

1.3. Problemas a resolver

Para el desarrollo de este trabajo, se han de resolver una serie de problemas en la aplicación web que se han planteado tras el uso de la misma. Para empezar, se deben actualizar algunas de las librerías utilizadas ya que han quedado obsoletas y las mejoras implementadas justifican el esfuerzo en el. reajuste de las mismas. La principal a poner al día es *Tensorflow*, ya que ha tenido un gran salto de la versión 1 (actual) a la 2.

Por otro lado, se ha decidido cambiar el tipo de la base de datos de una orientaba a documentos a una relacional como es *PostgreSQL*. Pese a que las bases de datos *NoSQL* tienen una gran facilidad de uso, ya que aceptan casi cualquier tipo de entrada mientras siga unos formatos, las bases de datos relacionales nos aportan una serie de beneficios muy útiles en este caso de uso. Primero, aportan una mayor seguridad, sobre todo con respecto a los tipos de entrada de los datos y los formatos de los mismos, además de mantener la atomicidad y uniformidad necesarias para evitar corrupciones en los datos. También nos encontramos con que las bases de datos *SQL* nos dan más facilidad al relacionar las diferentes entidades guardadas, en nuestro caso, queremos asociar las imágenes con el usuario que las sube. Otra de las ventajas, es la mayor rapidez de actualización o búsqueda debido a las diferentes formas internas de ordenación que

tienen este tipo de base de datos. Aparte de lo ya mencionado, las claves primarias y las restricciones *UNIQUE* nos permiten evitar la duplicidad en las bases de datos, algo que junto con la gestión de concurrencia intrínseca de la mayoría de este tipo de sistemas nos evita muchos problemas.

Por último, se le añadirán nuevas funcionalidades al sistema, como serán la restricción de uso a quién esté registrado como usuario, la introducción de una mayor cantidad de datos relativos a la imagen, la creación de una cola de imágenes visual para el usuario o la preparación del sistema para que permita no sólo realizar el diagnóstico, sino generar imágenes explicativas relativas al mismo. Asimismo, de cara a la administración de los usuarios se necesita una forma de administrar toda la información almacenada en la base de datos de manera simple y rápida mediante un panel de administración, junto con una forma de poder exportar todas las imágenes de la base de datos a un formato legible por los administradores.

Capítulo 2

Técnicas y herramientas utilizadas

2.1. Arquitectura de la aplicación web

La aplicación sigue una estructura de 3 capas en las que tenemos:

- El *frontend*: es la capa de presentación, donde se implementan aspectos relacionados con el consumidor finalista / productor primario de la información. Es el caso del interfaz de usuario, a través del cual, los datos son mostrados en un cliente y, desde donde se realiza la introducción de los mismos al sistema.
- El *backend*: es la capa de aplicación, donde se desarrolla toda la lógica.
- Por último, tenemos la capa de persistencia, que en este caso es la base de datos encargada de guardar las imágenes y los datos.

2.2. Herramientas de la capa de presentación

Para la capa de presentación, se ha decidido desarrollar una aplicación en un framework basado en *Typescript* (Typescript (2012)) llamado *React* (React (2012)) Se trata de una librería que nos permite crear interfaces de usuario para facilitar la entrada de información y la lectura de los resultado.

La estructura de este *framework* permite encapsular las diferentes partes de las páginas en pequeños componentes para poder mantener un diseño en bloques especializados cada uno en una función particular. Además:

- Permite la reutilización de partes de forma sencilla.
- Para unir varios componentes, se han creado otros que los engloban. En particular, las páginas que utiliza el usuario final serían componentes creados así.

Se ha elegido esta herramienta por los siguientes motivos:

- Al ser un proyecto continuación de otro anterior debemos reaprovechar en lo posible el software ya construido. Utilizar otro tipo de tecnología para incorporar las nuevas páginas complicaría la integración con el proyecto actual y dificultaría su mantenimiento.
- En todo caso, el framework React facilita los cambios en los estados de los componentes y simplifica el proceso de desarrollo. Algunas bibliotecas con el mismo propósito que *React*, como Vue.js o Angular.js no tienen el mismo grado de sencillez y necesitan otro tipo de desarrollo. Además no tienen la facilidad que da la librería elegida para aplicar estilos y modificar la apariencia visual de la aplicación.

2.3. Herramientas de gestión de usuarios

Para la gestión de la identificación de usuarios, se ha seleccionado *Auth0* (Auth0 (2020)). Esta plataforma permite externalizar las funciones de registro y "autenticación" de usuarios. Gracias a ella, no tenemos que preocuparnos de toda la creación, seguridad, mantenimiento, recuperación o almacenamiento de los usuarios y sus credenciales. Esto nos permite centrarnos en el desarrollo de otras partes de la aplicación.

2.4. Herramientas de la capa de aplicación

En cuanto a la capa de aplicación, se ha montado un servidor en *Flask* (Flask (2010)), una librería de *Python*.

Por lo tanto la lógica de la aplicación está implementada fundamentalmente en Python. Se ha elegido este lenguaje por las siguientes razones principales:

- Como se mencionó anteriormente, se trata de ampliar las funcionalidades de un proyecto existente donde el backend está programado en Python / Flask, por lo que la integración y el mantenimiento del proyecto serán tareas más sencillas manteniendo estas tecnologías.
- Además Python presenta una integración más fácil con el uso de redes neuronales gracias a las librerías existentes, como veremos después.
- Finalmente, se simplifica bastante la tarea de programación, ya que al ser un lenguaje interpretado de alto nivel, utiliza una sintaxis muy similar a la del habla común y se obtiene de forma natural código legible.

Otros lenguajes como *Javascript* con *Node.js* o *Java* con *Springboot* permiten realizar un servidor en el *backend* también de una forma bastante rápida pero no tienen una funcionalidad que se puede orientar tan fácilmente hacia el uso de redes neuronales e inteligencia artificial. Además de esto, con *Flask* disponemos de bastantes extensiones que nos permiten integrar funcionalidades sin tener que desarrollarlas totalmente desde cero. Tenemos el caso de *Flask-sqlalchemy* para la gestión de la base de datos, ya que se encarga totalmente de la gestión de la conexión y las operaciones con la misma, y *Flask-admin* que, integrándose con *Flask-sqlalchemy*, nos genera un panel de administración para poder gestionar de una forma más visual la base de datos.

2.5. Herramientas para la explotación de la red neuronal

Por otro lado, adjunto al *backend* tenemos una gran red neuronal compleja, funcionando como un servicio, a la que le proporcionamos las imágenes para su análisis. La red neuronal es un modelo matemático que debe realizar una gran cantidad de operaciones, millones de multiplicaciones y sumas, para obtener un resultado, lo que actualmente es posible de realizar en un tiempo razonable para un servicio web del tipo que se desea, gracias a la paralelización de los cómputos en procesadores multi-núcleo y sobre todo en hardware especializados como las GPU (unidades de procesamiento gráfico) o las TPU (unidades de procesamiento tensorial). La implementación de las redes neuronales en estos sistemas aprovechan la capacidad de cómputo paralela gracias a la definición de la mayoría de las operaciones como cálculos tensoriales (generalizando el producto

de matrices) y la creación a partir de estos, de grafos de cómputo donde conjuntos de imágenes son procesados de forma paralela. Estas implementaciones son complejas, pero actualmente están al alcance de los desarrolladores gracias a librerías que nos proporcionan bloques de mayor nivel que nos abstraen de los detalles de la implementación de estos grafos de cómputo optimizados.

En este proyecto se está haciendo uso de la librería *Tensorflow* (Abadi et al. (2015)), una librería desarrollada por la empresa *Google* de código abierto que se dedica a entrenar y explotar modelos de aprendizaje automático. La cantidad de herramientas de las que disponemos en esta librería y la gran comunidad que tiene detrás han sido factores definitorios para decantarnos por esta tecnología. Una de las herramientas principales que usa nuestra aplicación es *Keras* (Chollet et al. (2015)), una API que nos facilita la creación de redes neuronales y la puesta a disposición, de estas, como servicio, a otras partes de la aplicación.

2.6. Herramientas para la persistencia de los datos

Para la persistencia de datos disponemos de una base de datos relacional. Este tipo de bases de datos destaca por su sencillez de uso, con el lenguaje *SQL*, y su fácil gestión de las relaciones entre diferentes entidades, lo que ha hecho que la elijamos por encima de otros tipos de bases de datos como las orientadas a documentos o *NoSQL*. Debido al nuevo diseño de la aplicación que se realizó, la base de datos relacional nos permite también tener mejoras en la eficiencia de uso, incluso usando tipos de datos más complejos de lo normal. Se ha decidido usar *PostgreSQL* PostgreSQL (1996) como base de datos relacional por la gran cantidad de tipos nativos que acepta.

2.7. Herramientas para el despliegue

Por último, para poder aislar el *backend* y la base de datos del sistema en el que se ejecuten, se ha utilizado la *contenerización* de estas partes del sistema. *Docker* Docker (2013) es la herramienta elegida para ello, tanto por su gran comunidad, como por su rendimiento en el sistema de hospedaje. Además de esto, para poder conectar diferentes contenedores, hacemos uso de *Docker-Compose*, lo que nos permite levantar varios contenedores al mismo tiempo (*backend* y base de datos) y conectarlos entre sí de una forma muy sencilla.

Capítulo 3

Desarrollo de nuevas funcionalidades

3.1. Estado previo del sistema

El trabajo a realizar consiste en la ampliación de las funcionalidades de una aplicación web para el diagnóstico del glaucoma. La aplicación original fue desarrollada en para otro Trabajo de Fin de Grado titulado “Aplicación Web para diagnóstico de glaucoma basado en Deep Learning a partir de la recopilación de imágenes de la retina” Arvelo García (2020) . Las funcionalidades del sistema original se centran en los procesos de subir imágenes que son almacenadas en la base de datos, producir un diagnóstico usando la red neural y hacer que el usuario reciba el resultado en el mismo momento. Tras recibir la respuesta del diagnóstico, se guardaban algunos datos relativos a la mismas. El usuario no podía ver esos datos posteriormente.

Para esto, la aplicación original permite subir la imagen al *dropzone* que se ve en la figura 3.1, después el usuario debe seleccionar la sección a la que quiere dar énfasis y se enviar la imagen al servidor de *backend*. Tras recibir la respuesta de que la imagen ha sido analizada se permite introducir los siguientes datos relativos a la imagen: edad del paciente, modelo de cámara con el que se ha tomado la imagen del globo ocular y si el diagnóstico fue correcto o erróneo.

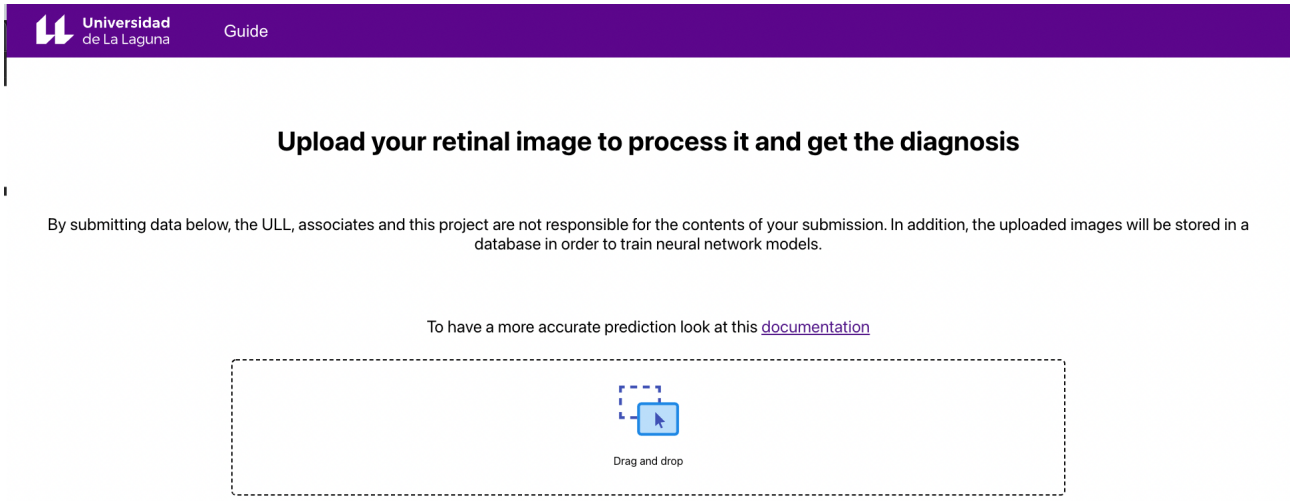

Figura 3.1: Página principal de la aplicación original

En la parte del *backend* solo disponíamos de los *endpoints* imprescindibles para que la parte del cliente funcione. La ruta para recibir la imagen desde el *frontend*, guardarla en la base de datos y empezar a procesarla; y las necesarias para enviar la imagen de vuelta además del resultado, para acabar recibiendo la información relativa a la imagen y guardarla en la base de datos. El sistema se encontraba ya funcionando en un contenedor de docker utilizando la imagen original de *Python*.

En cuanto a la red neuronal, funciona ya como un servicio al que se le envía una imagen y te devuelve el resultado. El flujo del sistema funciona gracias a una cola en la que se guardan los identificadores de las imágenes que se van recibiendo por el sistema y gracias a un único hilo sobre el que se ejecuta la red neuronal, el cual es bloqueado cada vez que se analiza una imagen. Este servicio coge un identificador de la cola para generar la imagen a analizar, a partir de las coordenadas almacenadas en la base de datos de la zona que eligió el usuario. Esta imagen se le proporciona a la red y ésta, gracias a su entrenamiento previo, ya dispone de una matriz de pesos calculados, devuelve una predicción sobre la posibilidad de glaucoma.

Para la base de datos, se utiliza una base de datos no relacional, *MongoDB* para ser exactos. Esta permite al sistema tener una gran facilidad de guardado de las imágenes y los datos ya que usa un formato similar al *JSON*, en el que la información se guarda gracias a parejas de clave y valor. Esta base de datos funciona gracias a la imagen oficial de *MongoDB* para *Docker*. Para poder desplegar de forma conjunta ambos servicios (*backend* y base de datos) se disponía de un fichero *docker-compose.yaml* que nos permite además conectarlos.

A contnuaci’on se explicarán las modificaciones realizadas en la aplicación, comenzando con el modelo de datos.

3.2. Incorporación de una base de datos SQL

Para finalizar, la base de datos ha sido el componente que más cambios ha sufrido. Se ha tenido que cambiar totalmente la estructura y el tipo de la misma por las nuevas necesidades del sistema de relacionar las imágenes con los usuarios. Debido a que se ha migrado de una base de datos documental, *MongoDB*, a una relacional, *PostgreSQL*, se ha tenido que definir totalmente un nuevo esquema de la base de datos. Este se compone de tres entidades: *Image*, *User* y *ExplanatoryImage*. Como se puede observar en la figura 3.2.

La primera se encarga de almacenar las imágenes subidas, codificadas en base 64, junto con su identificador único, generado por el sistema, y el nombre del archivo; además de esto, se almacenan datos médicos o personales del paciente:

- Edad del paciente, ya que la enfermedad suele aparecer a edades más mayores.
- Patologías previas que haya tenido el paciente, por si alguna puede tener relevancia en el diagnóstico del glaucoma.
- El sexo del paciente.
- El nombre del paciente.
- La etnia del paciente y si puede tener mayor riesgo de tener la enfermedad debido a esto.
- Si tiene algún familiar cercano *firsRelativeOAG* o lejano *otherRelativeOAG* con glaucoma ocular diagnosticado.
- La presión intraocular del ojo, ya que el aumento de esta es la principal causa de la

enfermedad.

- Si el paciente tiene dioptrías y si esto le produce tener un riesgo mayor de tener glaucoma, para poder almacenar los casos en los que afecta y en los que no la falta de vista.
- La presión sanguínea tanto sistólica como diastólica, porque la hipertensión también es causa del aumento de presión ocular.
- El grosor central de la cornea y si, debido a esto el paciente tiene una mayor probabilidad de tener glaucoma.

En esta tabla también se almacena la información referente al diagnóstico de la red neuronal:

- Las coordenadas donde comienza la zona de énfasis seleccionada por el usuario y el tamaño de la misma, para poder realizar el recorte de la imagen que se le proporcionará a la red neuronal
- El resultado del diagnóstico automático de la imagen producido por la red neuronal.
- El diagnóstico previo realizado por el experto, previo al procesado automático de la imagen.
- La opinión del experto referente al diagnóstico automático, dependiendo del acuerdo que tenga con ella.
- Las observaciones generales que quiera introducir el médico con respecto de la clasificación de la imagen

Por último, esta tabla almacena el estado de la imagen (*pending, processing, processed, finished*), el identificador del usuario que la ha subido, al que tiene que estar asociada, y la fecha de subida.

La tabla *User* guarda únicamente un identificador generado por el sistema para los usuarios y el campo "*sub*"del token generado por la plataforma de gestión de usuarios (suele ser el identificador que se le asigna al usuario en el entorno que genera, *auth0*. La última tabla, *Explanatory Image* es la encargada de guardar las imágenes explicativas del diagnóstico, de estas se va a guardar un identificador único, el identificador de la imagen original a la que pertenecen, la imagen explicativa codificada en base 64 y las opciones elegidas para el procesado de esa imagen explicativa.

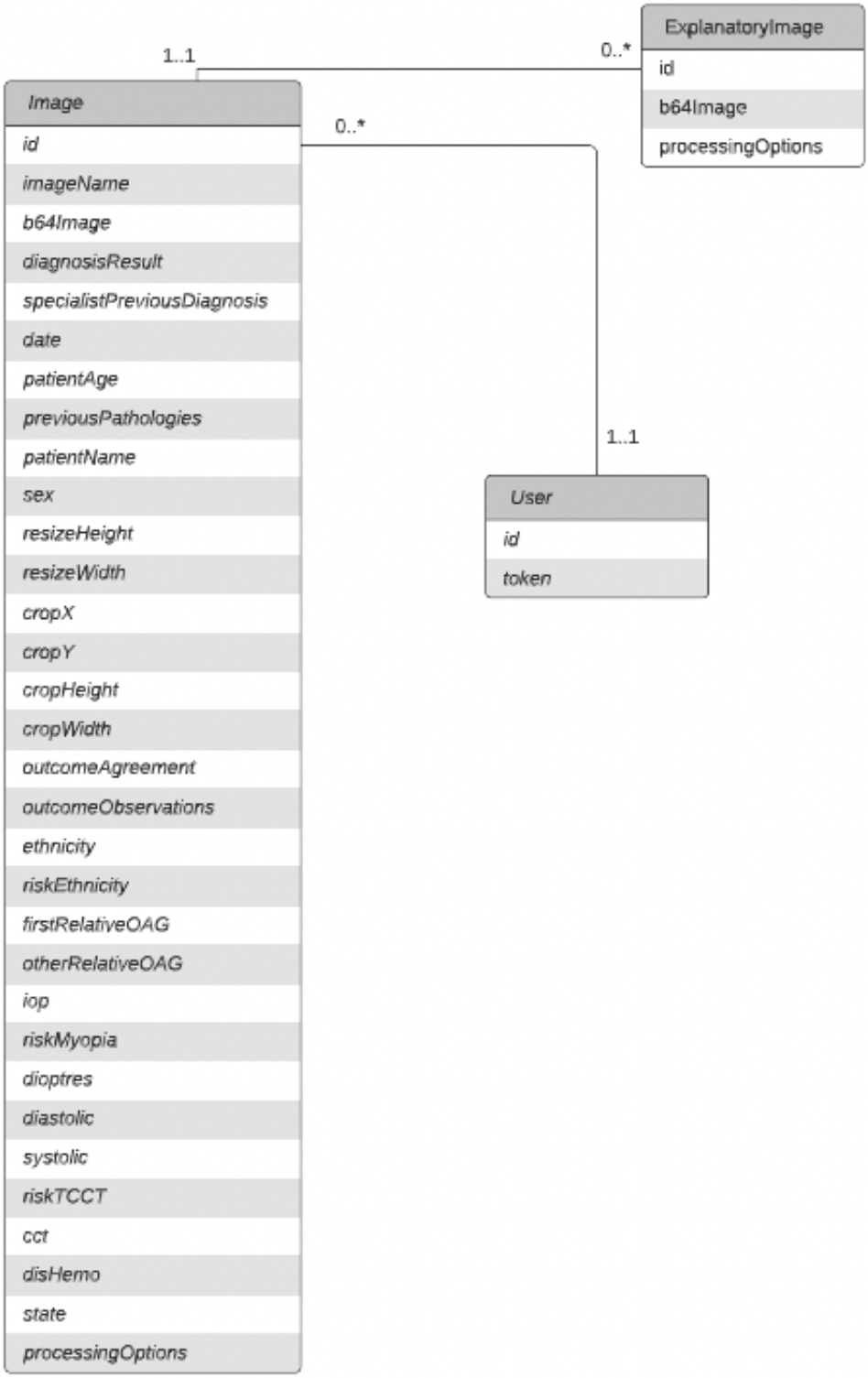

Figura 3.2: Modelo entidad relación de la Base de Datos

Todo esto funciona también gracias a *Docker,* ya que se ha montado sobre un contenedor utilizando la imagen oficial de *PostgreSQL*. Para que este contenedor y el del *backend* se puedan conectar y funcionen al mismo tiempo se hace uso de un *docker-compose* (figura 3.3)en el que se levantan ambos servicios al mismo tiempo, y se conecta uno con otro internamente. Esto nos permite tener un total aislamiento del sistema operativo sobre el que se levanta el proyecto y así evitar posibles errores de compatibilidad.

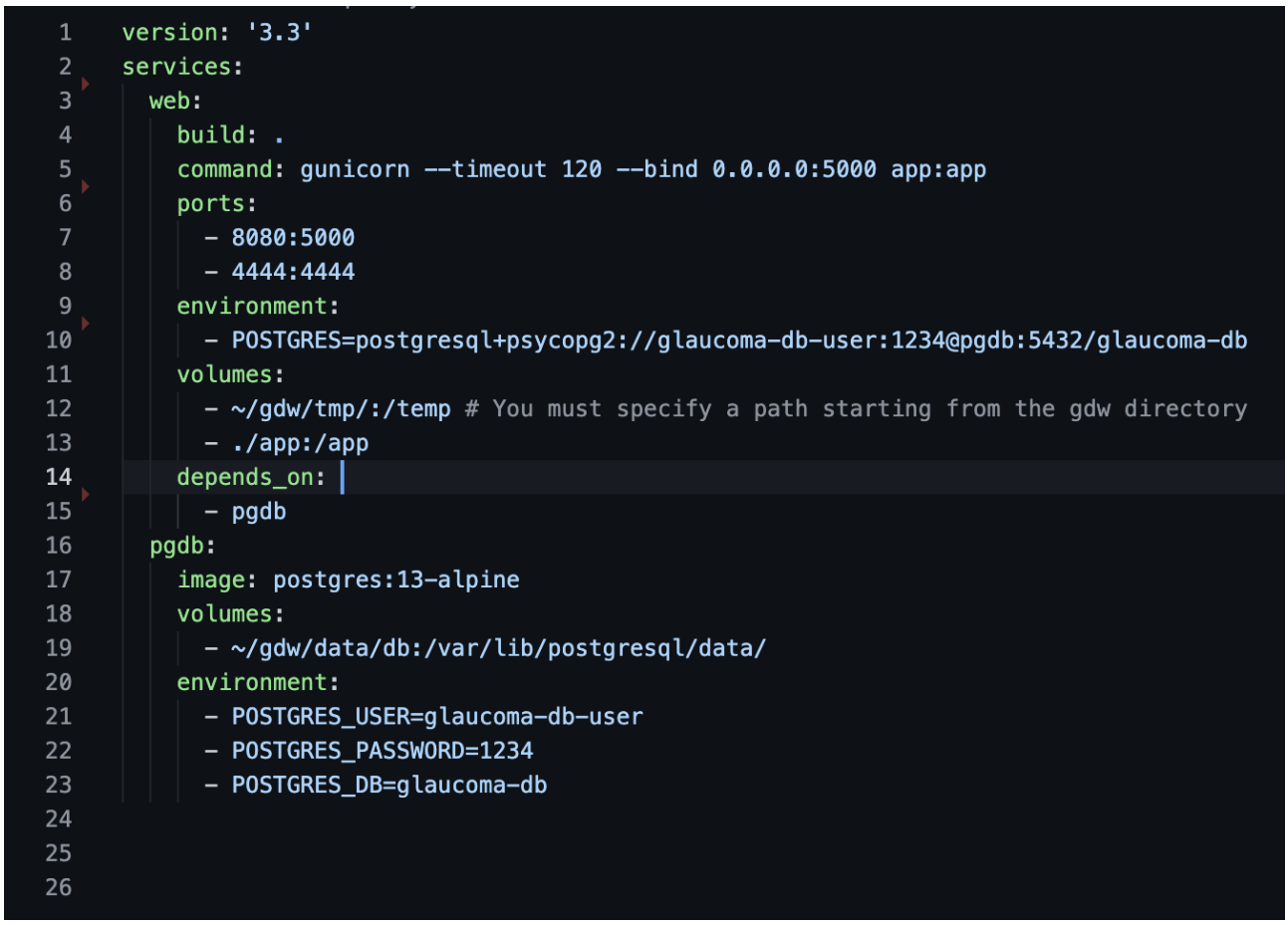

Figura 3.3: Fichero *docker-compose.yaml* en el que se definen los contenedores del *backend* y la base de datos

3.3. Mejoras implementadas en el Frontend

La aplicación, tras los cambios realizados, se compone de diferentes pantallas con las que interactúa el usuario. Para empezar, el usuario se encontrará con un historial de las imágenes subidas y analizadas, donde podrá ver una pequeña parte de la información relativa a la imagen, como se puede ver en la figura 3.4. Para poder ver toda la información de las mismas junto con el diagnóstico realizado, debe hacer *click* en la información de la imagen y se le abrirá un desplegable para poderlo ver todo de forma más clara (figura 3.5). Desde esta ventana podrá hacer *click* a un botón para acceder al componente que le permitirá subir una imagen de un globo ocular, figura 3.6.

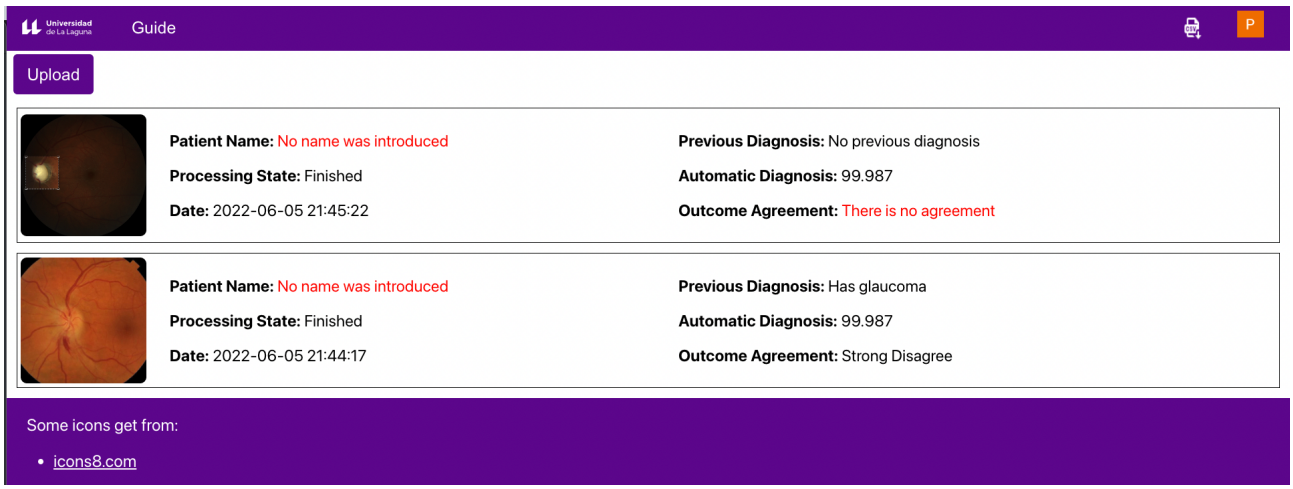

Figura 3.4: Historial de imágenes. Pantalla de entrada a la aplicación

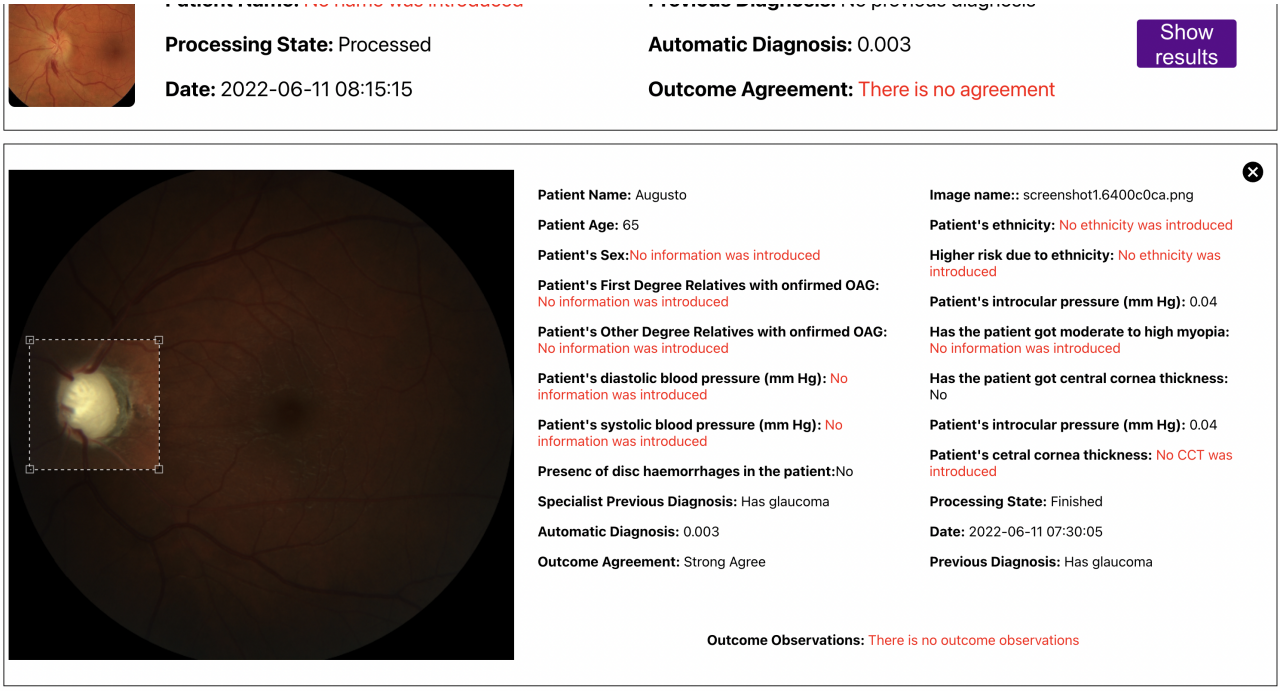

Figura 3.5: Historial de imágenes con una imagen mostrando su información detallada

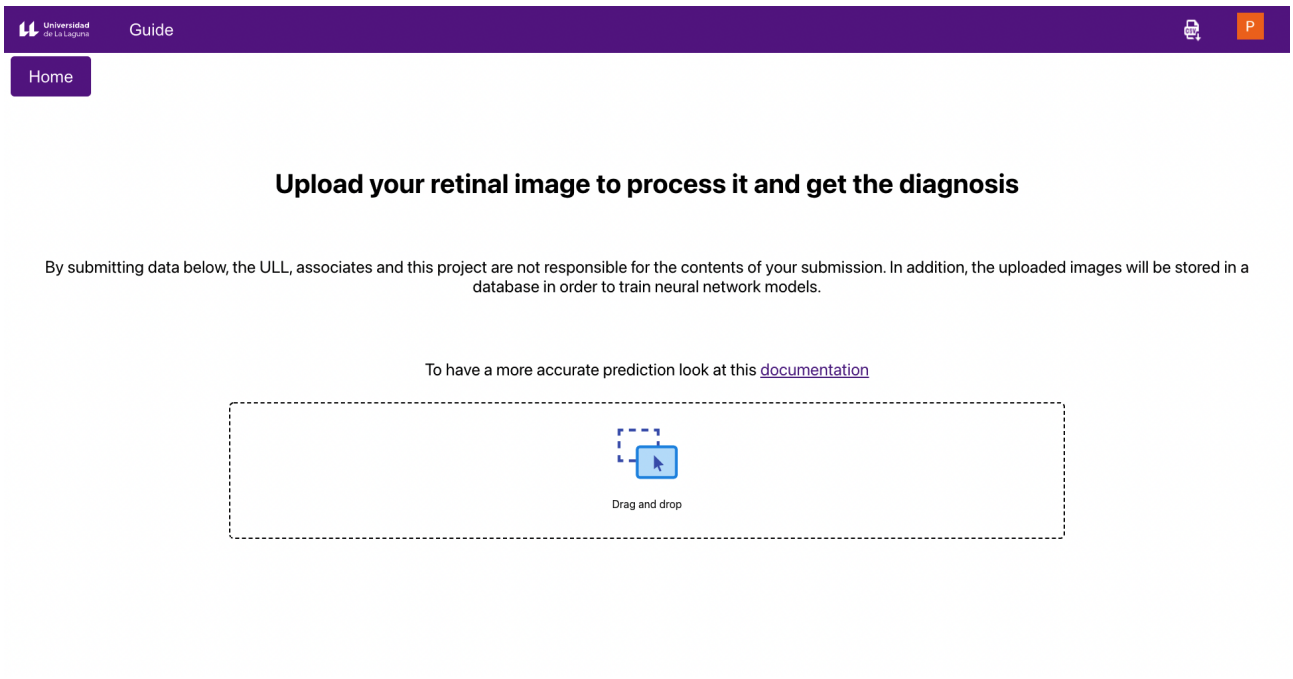

Figura 3.6: Componente para subir una imagen a analizar

Tras esto el usuario deberá hacer un recorte en la imagen para indicar a la red neuronal la zona que analizar con más énfasis y proporcionar al sistema un diagnóstico previo realizado por el experto, junto con la posibilidad de introducir un diagnóstico previo y unos parámetros relativos al procesado que hará la red neuronal.

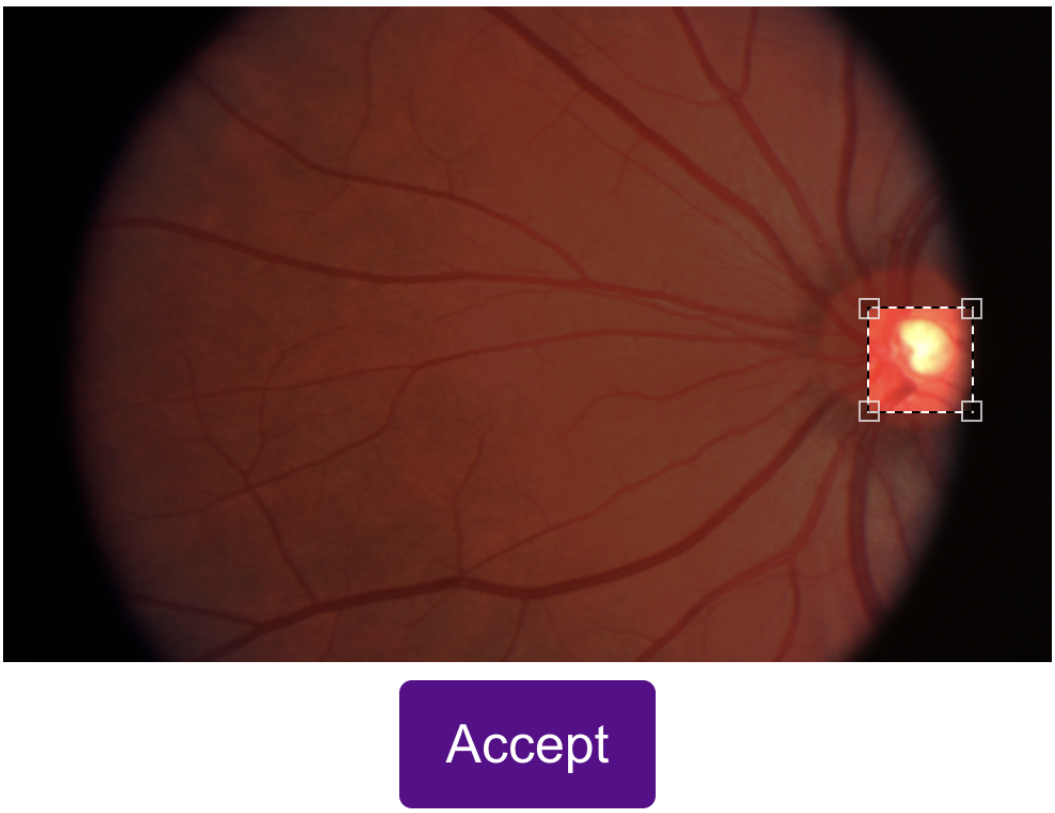

Figura 3.7: Selección de la zona de énfasis de la imagen

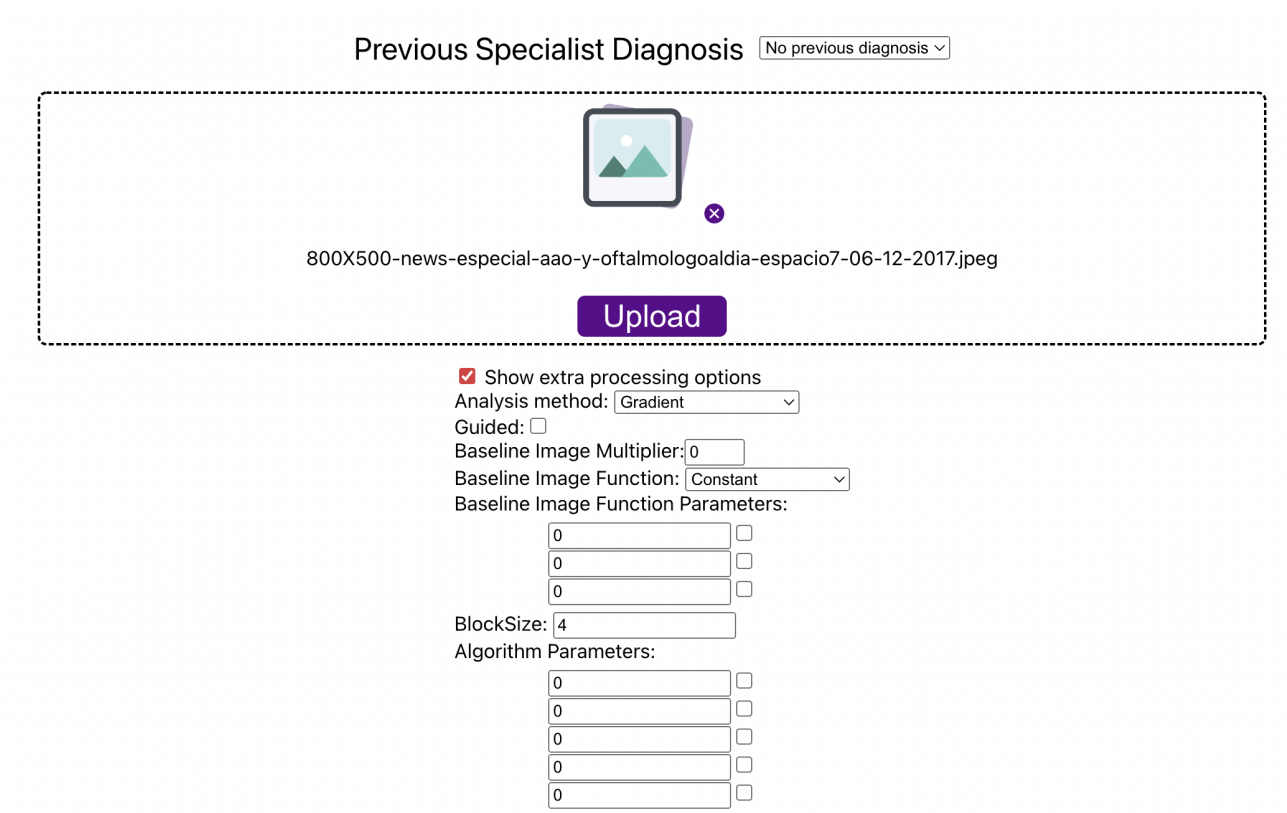

Figura 3.8: Opciones que puede introducir el usuario antes de subir la imagen

Una vez recibido el diagnóstico de la red neuronal se deben introducir una serie de datos pertenecientes al paciente de la imagen subida. En el proyecto se han definido algunos metadatos provisionales, que se muestran en las figuras 3.9 y 3.10, pero el conjunto exacto de metadatos aún debe ser refinado con la colaboración de los especialistas.

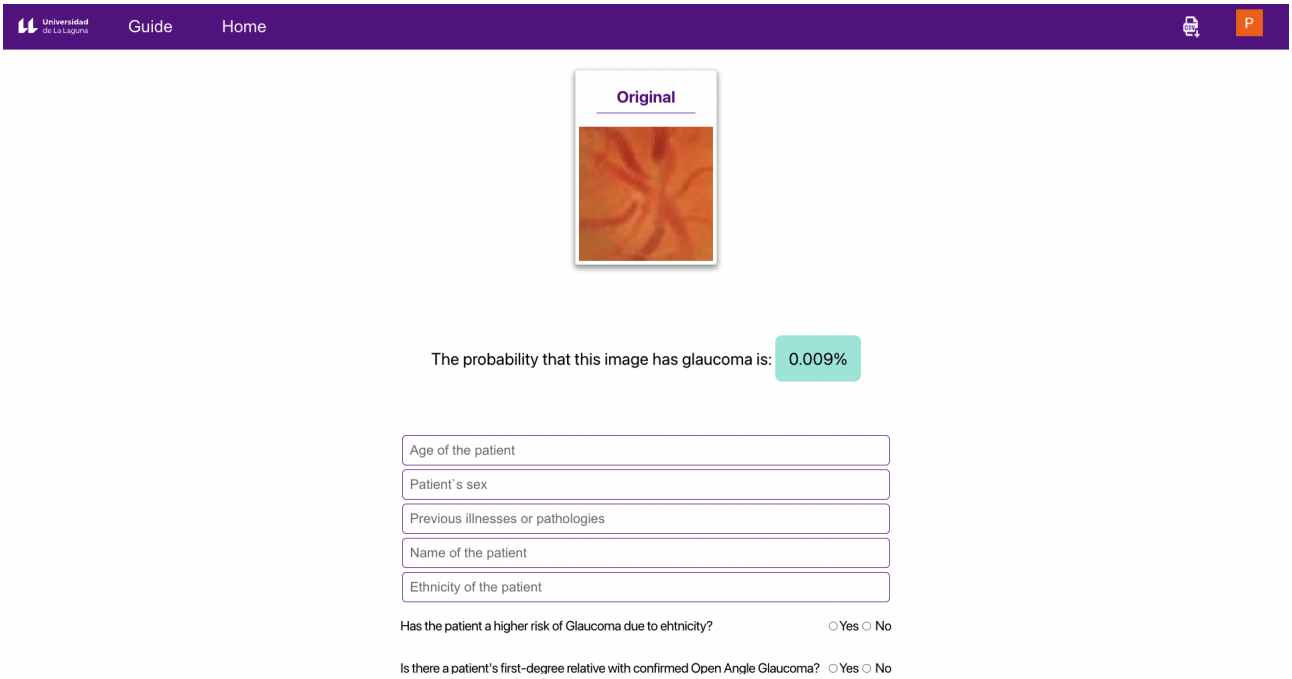

Figura 3.9: Introducción de los datos relativos al paciente (1/2)

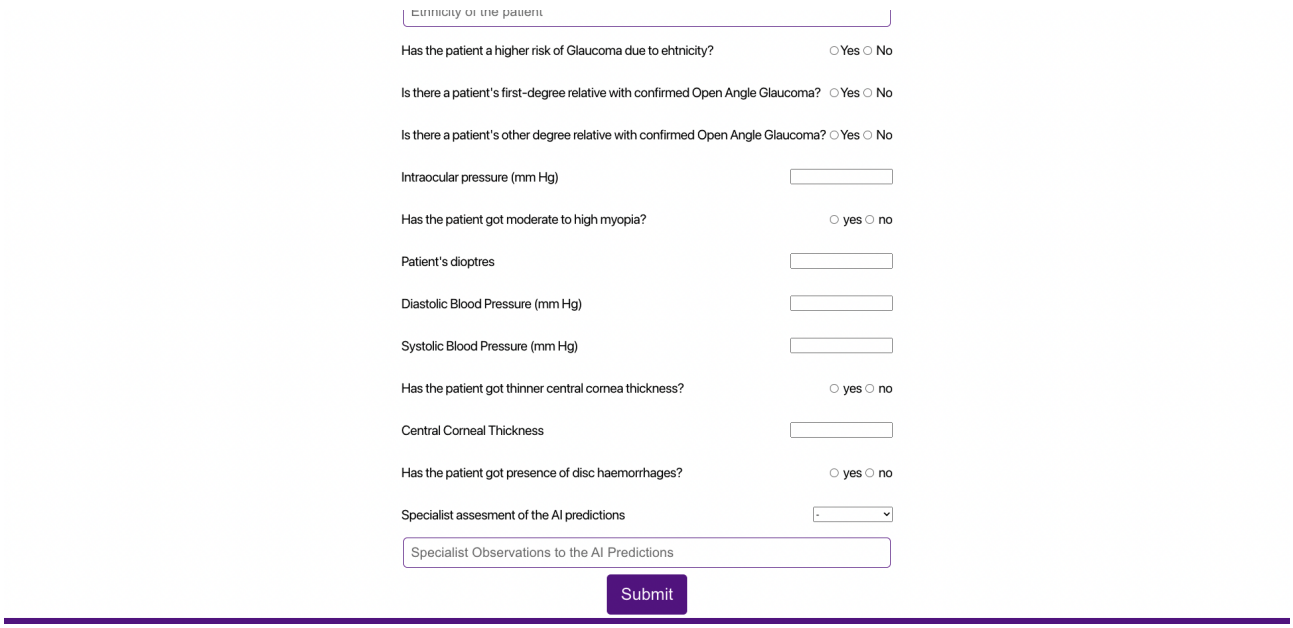

Figura 3.10: Introducción de los datos relativos al paciente (2/2)

Otro cambio realizado en el lado del cliente fue la implementación de la gestión y autenticación de usuarios con la plataforma *Auth0*. Recordemos que esta plataforma realiza las funciones de autenticación y gestión proporcionando toda la seguridad necesaria, evitando tener que manejar localmente la información de los clientes o aspectos como la generación y seguridad de sus token. Para su uso, se ha creado una cuenta en la plataforma. Tras esto, se han creado las claves necesarias para la conexión de la aplicación con la plataforma.

Además de incluir las URL desde las que se permiten llamadas a la API de la misma, hay que obtener el *Global Client ID* y el *Global Client Secret*. Junto con esto se pueden añadir otras muchas funciones que permite la plataforma de forma gratuita, además de las funciones de pago por un precio mensual. También debemos incluir el *SDK(Software Development Kit)* de *Auth0* para React y utilizar el componente *Auth0Provider*, proporcionado por el *developers kit*, en el fichero *index.tsx* englobando toda la aplicación. En la figura 3.11, se muestra como hacemos uso de las variables de entorno que declaramos en un fichero *.env* e incluimos los datos de autenticación proporcionados por *Auth0*.

```
const domain = (process.env.REACT_APP_AUTH0_DOMAIN as string)
const client_id = (process.env.REACT_APP_AUTH0_CLIENT_ID as string)



ReactDOM.render(
  <Auth0Provider
    domain={domain}
    clientId={client_id}
    redirect_uri={window.location.origin}
    onRedirectCallback={onRedirectCallback}
  >
    <React.StrictMode>
      <Modal />
      <App />
    </React.StrictMode>
  </Auth0Provider>,
  document.getElementById('root')
);
```

Figura 3.11: Inclusión de *auth0* en la aplicación de *React*

Para que el usuario pueda utilizar cualquier funcionalidad del sistema, debe tener la sesión iniciada. Esto lo puede realizar haciendo click en el botón *Log In* de la esquina superior derecha. Tras esto aparecerá una ventana que nos proporciona *auth0* para que el usuario inicie sesión o se registre.

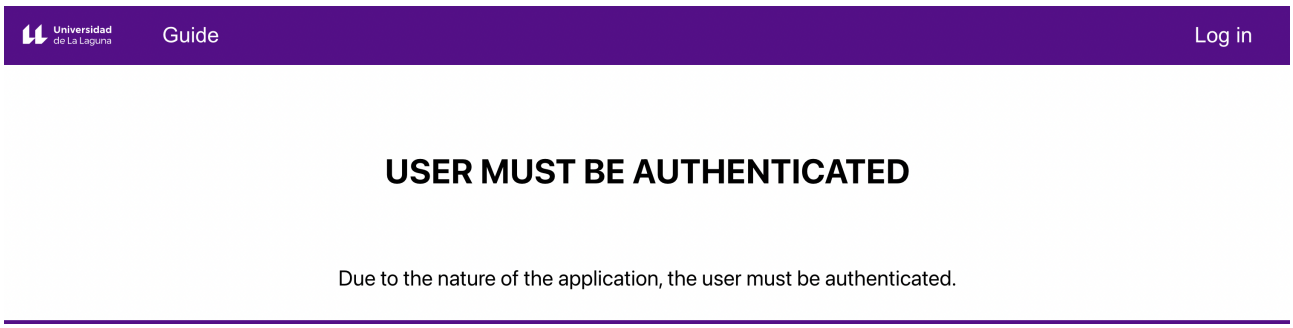

Figura 3.12: Los usuarios tienen el acceso a la aplicación restringido a no ser que tengan la sesión iniciada

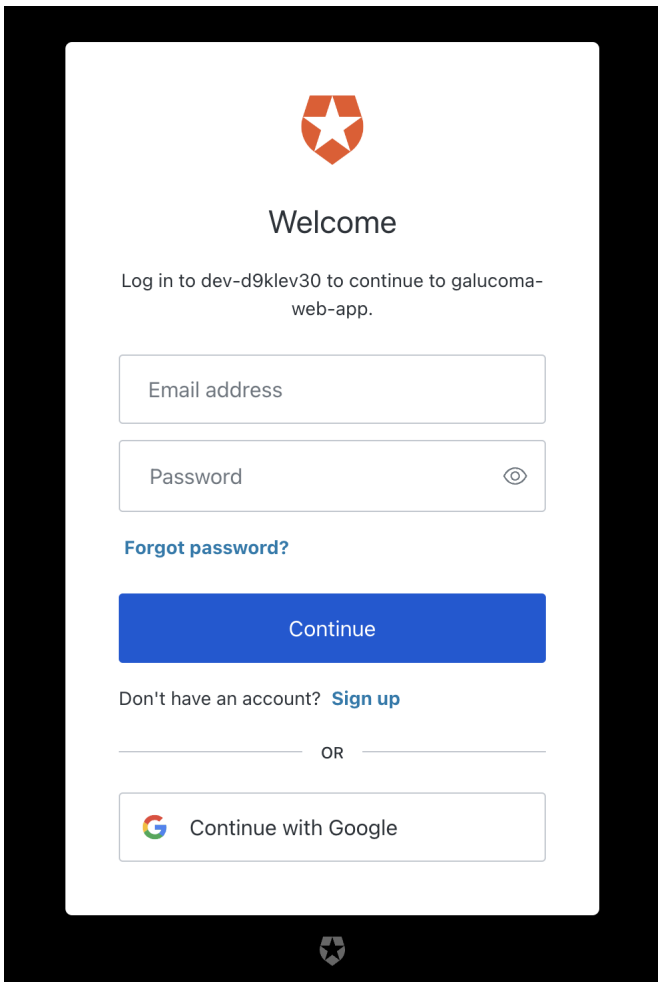

Figura 3.13: Ventana para iniciar sesión con *auth0*

Una vez el usuario ha iniciado sesión, tiene acceso a una página con la información de su usuario (email, nombre de usuario e imagen por defecto que proporciona *auth0*).

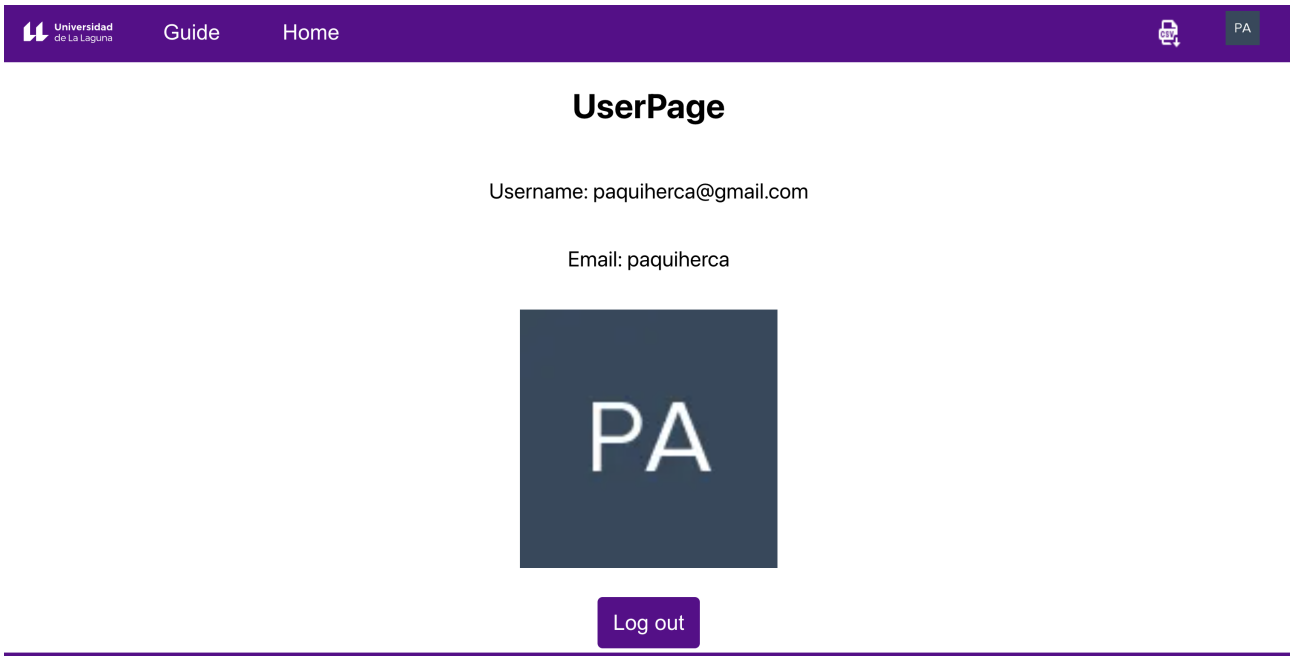

Figura 3.14: Página de información del usuario

Para finalizar, el usuario del sistema dispone de una página *Guide*, para la que no hace falta estar registrado, en la que se explica principalmente el funcionamiento del sistema (figura 3.15)

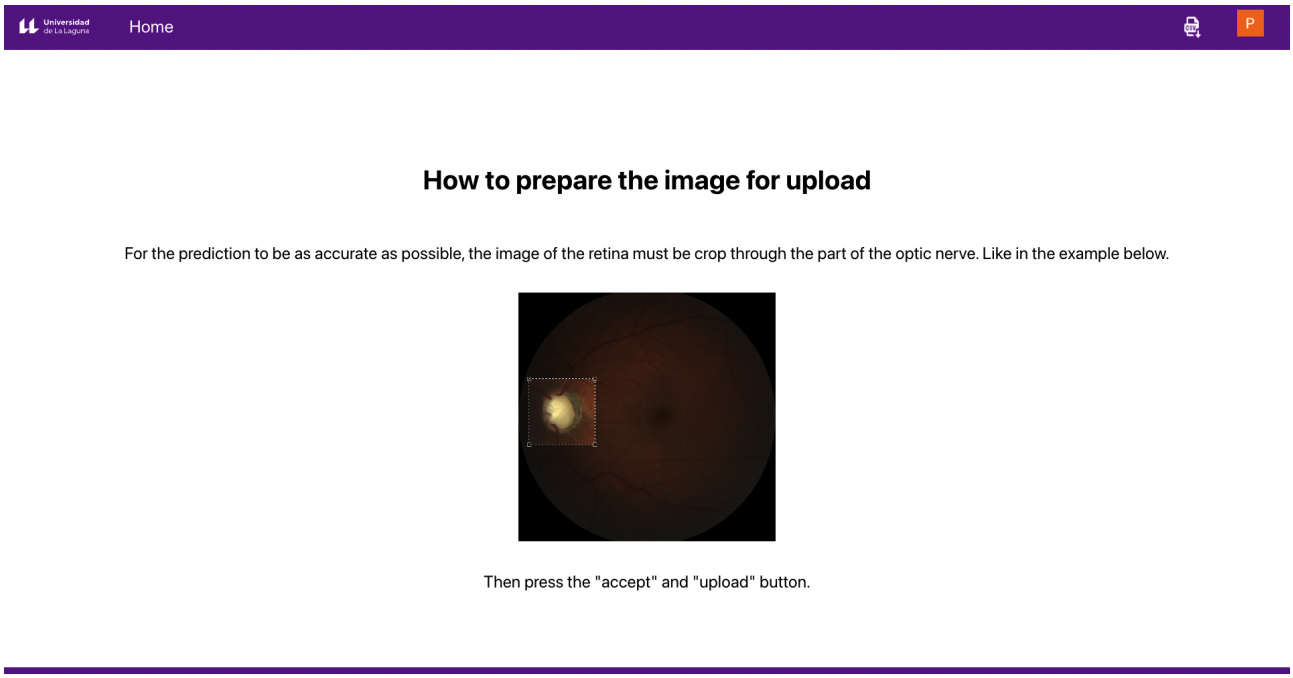

Figura 3.15: Pantalla mostrando la forma de uso del sistema

El *frontend* tras recibir la imagen que se ha subido debe mostrar los resultados del diagnóstico de la red neuronal, para conseguirlos debe comunicarse a través de peticiones HTTP con el backend, mediante *axios*. Se utilizan tanto peticiones POST para enviar datos como GET para solicitarlos, dependiendo de lo que queramos conseguir.

3.4. Modificaciones en el *Backend*

Por lo que al backend respecta veamos primero una lista de los cambios realizados, para luego pasar a comentar cada uno de ellos:

- Actualización de la versión de Tensorflow (de 1 a 2)
- Programación de la persistencia de datos utilizando la nueva base de datos SQL.
- Programación y modificación de endpoints en el servidor para atender las nuevas funcionalidades, como el historial o los nuevos metadatos.
- Incluir un estado para gestionar la situación del diagnóstico de cada imagen manteniendo informado al usuario en el historial.
- El usuario puede volcar todas sus imágenes con la información asociada a las mismas en un archivo CSV.
- Panel de administración para el administrador.
- Volcado general completo de la base de datos.

3.4.1. Actualización de la versión de Tensorflow

Lo primero que se ha realizado es la migración de la versión 1 de *Tensorflow* a la 2, más reciente y con mejores funcionalidades. Hay que tener en cuenta que Keras ha sido

integrado dentro del propio Tensorflow en esta versión, por lo que se modificó el código existente para utilizar este módulo de *keras* integrado en Tensorflow 2 (*tf.keras*), en lugar del provisto por la librería Keras de forma independiente.

Una de las grandes diferencias entre Tensorflow 2 y Tensorflow 1, es que en la nueva versión el modo de cómputo por defecto es “eager”, lo que significa que no hay una primera fase de definición del cómputo y otra a posteriori de realización del mismo, sino que la propia definición del cómputo puede estar dando pie a la librería tensorflow a considerarlo un cálculo inmediato y proceder al cálculo antes de continuar con el programa.

Esto simplifica los programas y su depuración porque evita la necesidad de crear explícitamente objetos para representar grafos de cómputo previamente y más adelante proceder a la ejecución. Así pues, estas modificaciones se realizaron en el código original.

Tras esto, hubo que reparar el sistema de despliegue de la aplicación. En concreto, actualizar las versiones de los paquetes en el fichero *Requirements.txt*, el cuál nos permite seleccionar las versiones de los módulos para ser instalados en el sistema, mediante el gestor de paquetes *pip* de *Python*, el cuál se encarga de buscar e instalar secuencialmente todas las librerías indicadas. En este hubo que actualizar casi todos los paquetes debido a las compatibilidades entre ellos. También se añadieron algunos paquetes necesarios (*psycopg*) y se eliminaron otros que ya no hacían falta (*flask-mongoengine*) derivados del cambio de base de datos.

Figura 3.16: Cambio en las versiones de los paquetes de *Python*

3.4.2. Programación de la persistencia basada en SQL

Por otro lado, debido al cambio del tipo de base de datos que se ha realizado se ha tenido que volver a implementar el módulo del sistema encargado de hacer persistir la información. Mediante los paquetes *psycopg* y *sqlalchemy* se han creado los modelos ne-

cesarios para la persistencia. El primero se encarga de realizar todas las comunicaciones con la base de datos *PostgreSQL* desde *Python*, y el segundo nos permite definir de una manera muy cómoda la estructura de la base de datos y de las entidades. Junto con esto se desarrolló un fichero que contiene todas las operaciones relativas a las operaciones de creación, lectura, actualización y borrado de la base de datos (*CRUD*) para que estén encapsuladas todas en el mismo lugar, además de mantener una pequeña abstracción de la base de datos (figura 3.17).

```
14
15  > def initDB():⋯
18
19
20  > def save_image_to_DB(new_image: Image):⋯
26
27
28  > def get_image_by_id(id: str):⋯
33
34
35  > def update_image_diagnosis(id: str, diagnosis: float):⋯
42
43
44  > def update_image_metadata(image_id: str, metadata: dict):⋯
56
57
58  > def get_all_images():⋯
63
64
65  > def get_all_images_from_user(token_to_search: str):⋯
74
75
76  > def check_if_exists_user_id(id: str):⋯
80
81
82  > def save_user_to_DB(new_user: User):⋯
89
90
91  > def get_user_by_sub_field(sub: str):⋯
96
97
98  > def set_image_as_processed(image_id: str):⋯
105
106
107 > def set_image_as_finished(image_id: str):⋯
115
```

Figura 3.17: Operaciones relativas con el CRUD implementadas

3.4.3. Modificación y creación de endpoints en el servidor para las nuevas funcionalidades

Añadido a esto, se han implementado nuevas rutas en el servidor para poder completar nuevas funcionalidades. La primera es la introducción de un *endpoint* para obtener todas las imágenes asociadas a un usuario, de forma que se pueda tener un histórico de las mismas. En él se obtienen todas las imágenes relativas al usuario del que se envía el

identificador. Para ello, se hace uso de la función *getAllImagesFromUser* en la que se solicita a la base de datos todas las imágenes que estén asociadas al usuario. La respuesta enviada al cliente serán las imágenes devueltas por el método que realiza la consulta, junto con todos sus datos asociados, en formato JSON por su facilidad de uso en la mayoría de lenguajes.

```
class getAllImagesFromUser(Resource):
    def get(self, token):
        methods = ['GET']

        if not request.method == 'GET':
            return {'message': 'Invalid method'}, 405


        imagesFromDB = get_all_images_from_user(token)
        if(imagesFromDB is None):
            return Response(response=json.dumps([]), status=200)
        images_as_JSON_Array = []
        for img in imagesFromDB:
            images_as_JSON_Array.append(json.loads(img.toJSON()))

        return Response(response=json.dumps(images_as_JSON_Array), status=200)
```

Figura 3.18: Método implementado para obtener todas las imágenes relativas a un usuario, de cara a su uso en como historial

A continuación se muestran todas las rutas de las que disponemos en el sistema para podernos conectar con el *backend* y recibir o enviar la información necesaria:

- */api/process*: Se envía una petición POST con la imagen y una serie de datos relativos a ella y a la zona a analizar indicada por el usuario. En este momento, en el *backend* se genera una entidad con un identificador único para la imagen que se guarda en la base de datos junto con los datos asociados a la misma. Además, se añadirá el identificador a una cola de imágenes a procesar por la red neuronal. Esta ruta devolverá al cliente la cadena de texto única que permite reconocer la imagen.
- */api/getdiagnosis/<string:uuid>*: Este *endpoint* recibirá una petición GET para ver si el diagnóstico automático ya ha sido generado para la imagen indicada en la ruta, mediante su identificador. Si se ha generado un resultado, se devuelve el diagnóstico, si no ha sido aún procesada devolverá una respuesta vacía.
- */api/download/<string:uuid>*: Haciendo una petición GET a esta ruta se devuelve una imagen, que será la zona que seleccionó el usuario al subir la imagen al sistema.
- */api/metadata/<string:uuid>*: A esta url debemos hacer una petición POST en la que mandemos los datos relativos a la imagen introducidos por el experto tras saber el diagnóstico. La imagen a la que se asocian es indicada por el identificador escrito en la ruta.
- */api/image/<string:uuid>*: Aquí debemos hacer una petición GET y el sistema nos devolverá toda la información que tiene relativa a la imagen que indiquemos en la url mediante el identificador único.
- */api/allImages/<string:token>*: Este *endpoint* nos devuelve a una petición HTTP, del tipo GET, todas las imágenes y su información relativas al usuario especificado en la

url.

- */api/allImagesToCSV/<string:token>*: Esta última url nos permite hacer una petición GET, a la que el sistema nos responde con un archivo en formato csv con los datos de todas las imágenes subidas por el usuario, además de la imagen codificada en base64.

```
37  api.add_resource(ok, '/api/ok')
38  api.add_resource(process, '/api/process')
39  api.add_resource(getDiagnosis, '/api/diagnosis/<string:uuid>')
40  api.add_resource(downloadImage, '/api/download/<string:uuid>')
41  api.add_resource(saveMetadata, '/api/metadata/<string:uuid>')
42  api.add_resource(getImageMetadata, '/api/image/<string:uuid>')
43  api.add_resource(getAllImages, '/api/allImages')
44  api.add_resource(getAllImagesFromUser, '/api/allImages/<string:token>')
45  api.add_resource(allImagesFromUserToCSVFile, '/api/allImagesToCSV/<string:token>')
46
```

Figura 3.19: Declaración de los *endpoints* de la *API*

Entre las nuevas funcionalidades implementadas, se han añadido nuevos metadatos relativos a la imagen que se guardan en la base de datos. Estos son referentes principalmente al historial clínico del paciente, algunos ejemplos son: si tiene familiares con antecedentes de glaucoma ocular, las dioptrías del paciente, la presión sanguínea, tanto sistólica como diastólica del paciente o el grosor de su cornea central. Esto tiene como consecuencia, la modificación del endpoint relativo a los metadatos.

3.4.4. Modificaciones relativas al flujo de operaciones en la determinación del diagnóstico

Se ha añadido un estado al registro que representa la imagen para poder identificar en qué punto del flujo de operaciones se encuentra el proceso de diagnóstico. El estado puede tomar diferentes valores: *pending* para cuando está pendiente de ser procesada, *processing* para cuando está siendo procesada, *processed* para cuando se ha obtenido el resultado del diagnóstico de la red neuronal y se marcará como *finished* cuando se introduzcan todos los metadatos. En relación a esta inclusión de estados, se ha modificado el funcionamiento de la ruta */api/process*, añadiéndole el estado *pending* a la imagen, además debido al cambio en la base de datos se ha modificado ligeramente la forma en la que se persiste la imagen: al crearla se asocia el usuario a la misma (creándolo si no existe) y se incluyen algunas comprobaciones por si ocurre un error al insertarla en la base de datos. En */api/metadata* se marca la imagen como *finished*, una vez todos los datos han sido guardados. El funcionamiento de la red neuronal como servicio no se ha modificado, aunque al flujo de la misma se han añadido las funcionalidades de marcar la imagen como *processing* y como *processed*. Así, con esta información, se permite que el cliente sepa en todo momento en que estado está su imagen, ya que, debido a la particularidad del sistema, únicamente se utiliza un hilo para la red neuronal, para evitar problemas de concurrencia con la cola, cuando se reciban varias peticiones al mismo tiempo se deberán procesar una después de la otra. A continuación se muestra el flujo del procesado de imágenes por la red neuronal:

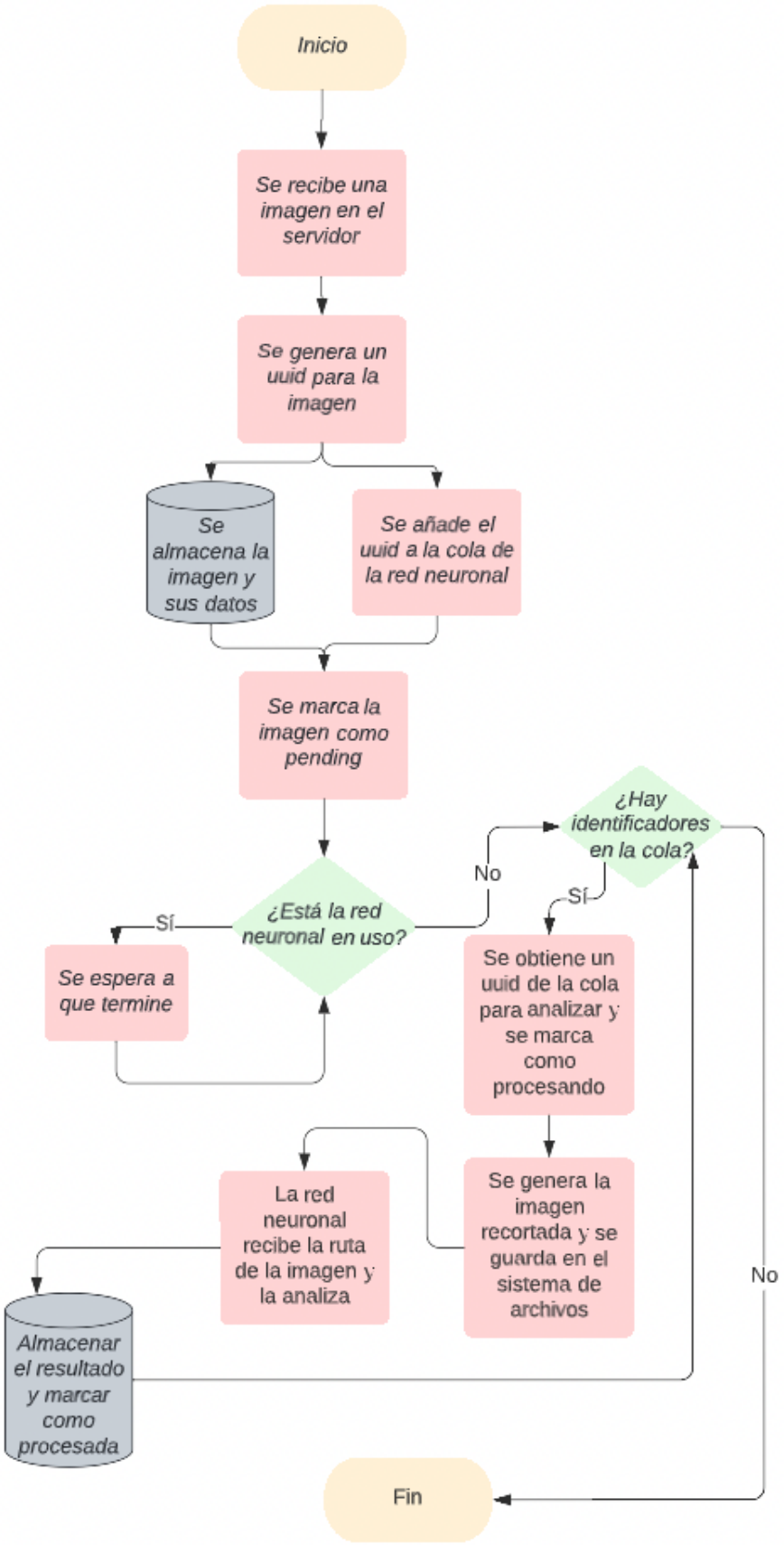

Figura 3.20: Flujo del proceso de diagnóstico de imagen por la red neuronal

3.4.5. Volcado por usuario de todos sus registros

Otra nueva funcionalidad para los usuarios es la posibilidad de descargar un fichero *csv* con los datos y las imágenes que han subido ellos mismos. En este archivo las imágenes estarán codificadas en base 64. Esto se hace mediante el módulo *csv* de *Python*, primero busca todas las imágenes del usuario en la base de datos, luego crea el fichero y le añade la primera línea con las cabeceras de las columnas para finalizar añadiendo todas las imágenes junto con sus datos, línea a línea.

3.4.6. Panel de administración

Para los administradores se han añadido igualmente nuevas funcionalidades. Se ha creado un panel de administración en el servidor, con el que fácilmente pueden gestionar todas las tablas de la base de datos, tanto usuarios como imágenes. Esto se ha realizado con la librería *flask-admin*, la cuál tiene una integración bastante sencilla con *sqlalchemy*, el gestor de bases de datos de *Flask* en *Python*.

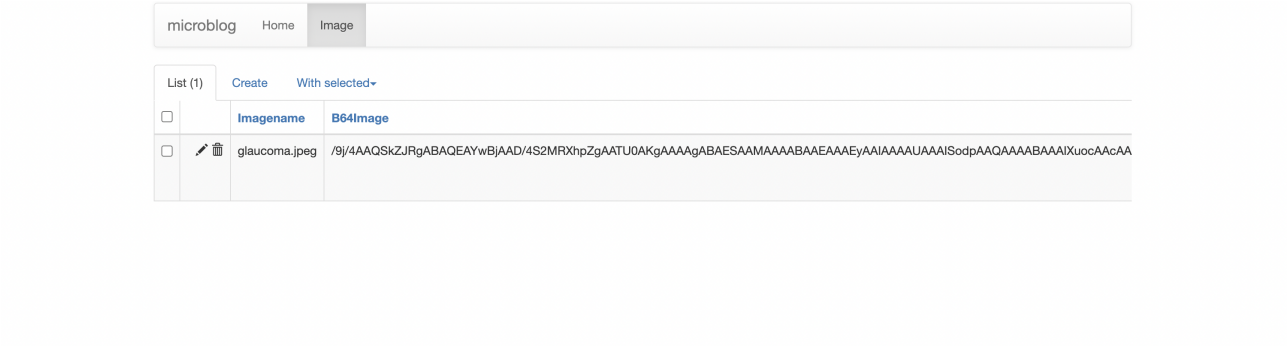

Figura 3.21: Interfaz de *flask-admin* para gestionar la tabla de imágenes

3.4.7. Volcado general de la base de datos

Adicionalmente, los administradores tienen la posibilidad de realizar un volcado de la base de datos al sistema de archivos de la máquina en la que se ejecute el sistema. Esto se realiza mediante un *script* realizado en *Python* en el que se descarga toda la base de datos a través del *endpoint /api/allImages*, el cuál solo acepta peticiones desde *localhost* para ponerle restricciones de acceso. Una vez se recibe los datos del servidor, el programa crea, en la ruta del usuario, un directorio llamado *databaseImages*, que contendrá una nueva carpeta nombrada con la fecha y hora en la que se realizó el volcado. Dentro de esa, se crea una carpeta por cada imagen en la base de datos, el nombre para esta carpeta será el identificador único que tiene la imagen. Dentro de esta carpeta se guardará la imagen en el formato que fue subida originalmente. Tras esto, se genera un fichero csv con todos los campos existentes en la base de datos de las imágenes, sustituyendo el contenido del campo en el que se encontraba la imagen codificada en base 64 por la ruta a la imagen en el sistema de archivos.

Figura 3.22: Estructura de ficheros generada por el volcado

Esta parte de la aplicación se ha contenerizado con un fichero *Dockerfile*, gracias a *Docker*. Para la imagen se utiliza la oficial de *Python* en la versión 3.9. Una vez se ha generado el contenedor se actualiza el paquete *pip*, encargado de instalar módulos de *Python*, para hacer una instalación de todos los paquetes que se encuentran en el fichero *Requirements.txt* (figura 3.16). Tras haber instalado los paquetes, se instalan algunos componentes del sistema necesarios para su utilización. Por último, se copia el directorio de trabajo al sistema de ficheros del contenedor.

```
1   FROM --platform=linux/x86_64 python:3.9
2
3   WORKDIR /usr/src/app
4
5   ENV PYTHONDONTWRITEBYTECODE 1
6   ENV PYTHONUNBUFFERED 1
7
8   RUN pip3 install --upgrade pip
9   COPY ./requirements.txt .
10  RUN pip install -r requirements.txt
11  RUN apt-get update
12  RUN apt-get install ffmpeg libsm6 libxext6  -y
13
14
15  COPY ./app /usr/src/app
16
```

Figura 3.23: *Dockerfile* para la generación de la imagen del *backend*

3.5. Dificultades encontradas

A lo largo de la implementación de las funcionalidades mencionadas previamente, me he encontrado con una serie de problemas no esperados que he tenido que resolver. El primero se me planteó debido al uso de estados en *React*, ya que para su actualización se utiliza una función propia del lenguaje *useState* que funciona de forma asíncrona, por

lo que en algunas ocasiones al implementar algunos campos de entrada no se reflejaba el cambio en el dato. Para solucionarlo, tras investigar bastante y entender que los estados en este *framework* se actualizan cuando se vuelven a renderizar los componentes, decidí encapsular en componentes más pequeños algunas partes y así permitir que la actualización fuera de elementos más pequeños y así fuera más rápida.

La mayor dificultad encontrada fue la migración de la base de datos de un sistema a otro. Esto fue por culpa de las pocas similitudes que tienen en la forma de almacenar la información. Para poder solucionarla, gracias a la ayuda de un experto en el sistema y en el tema, hice desde cero el nuevo esquema de la base de datos, simplificándolo al máximo para que las consultas fueran lo más fáciles posibles. Otro problema con el que me topé fue la decisión de cómo persistir las imágenes en el sistema. Almacenarlas en el sistema de archivos de la máquina en la que se ejecutara el programa o guardarlas en la base de datos codificadas en base 64 fueron las dos opciones que contemplé. Finalmente, la decisión fue persistirlas en la base de datos porque, aunque este sistema de cara a tener una cantidad muy grande de imágenes en la base de datos puede volverse un poco lento si se hacen búsquedas de imágenes, nos permite tener toda la información localizada en un mismo repositorio y no tener que referenciar a un fichero que esté en el sistema, y acceder a él cada vez que se necesite usar una imagen. Si se quieren realizar copias de seguridad de la base de datos, estarán las imágenes en ellas, si se quiere migrar el sistema a otra máquina, con mover la base de datos tendremos todas las imágenes en ella, o en el caso de que se quiera usar la base de datos para entrenar nuevas redes neuronales o nuevos sistemas podremos hacer uso de las imágenes de forma sencilla; estas han sido algunas de las razones que nos han hecho decantarnos por esta opción por encima de la otra.

Capítulo 4

Conclusiones y líneas futuras

4.1. Conclusión

El diagnóstico precoz de enfermedades es una tarea pendiente de la medicina actual y, gracias a herramientas de aprendizaje automático, se permite a los profesionales del sector contar con la ayuda de una red neuronal entrenada para conseguir su propósito y evitar enfermedades. La inteligencia artificial añade así un campo más en el que puede tener una influencia mayúscula en los próximos años y con este tipo de aplicaciones se consigue que la población deje de verla como una amenaza a sus puestos de trabajo, y la vea más como una herramienta. En lo que se refiere al glaucoma específicamente, la detección precoz es muy compleja debido a que no hay síntomas especialmente definitorios, por ello es necesario que los especialistas que los diagnostiquen tengan una dilatada experiencia. Con modelos de *Deep Learning* se facilita esta tarea en gran medida, siempre que se disponga de una cantidad necesaria y correctamente etiquetada de imágenes.

Esta aplicación web le proporciona a los médicos la posibilidad de, mediante una retinografía, obtener un diagnóstico automático de si ese globo ocular tiene glaucoma. Todo funciona gracias a una red neuronal convolucional que actúa como un servicio y a la que se hacen llamadas según se van subiendo nuevas imágenes. Estas imágenes subidas, también permiten a la red seguir mejorando, siempre y cuando se reciba una retroalimentación correcta del profesional sobre el juicio automático realizado.

Las mejoras realizadas a la aplicación web original permiten al cliente tener una mejor experiencia en el uso del sistema. Con la inclusión de usuarios, el sistema se vuelve más personalizado y, junto con la inclusión de un historial de las imágenes analizadas, el médico ahora tiene la posibilidad de comprobar diagnósticos ya realizados, junto con sus datos, además de poder comparar unos con otros para tener mejores resultados. Por otro lado, con la inclusión de nuevos metadatos, relativos a la imagen y al paciente podemos permitir etiquetar las imágenes en la base de datos de una forma más óptima. También el usuario puede identificar por los datos del paciente los diagnósticos realizados y así incluir comentarios en los informes pertinentes. Todo esto ha sido gracias a la inclusión de la base de datos relacional en el sistema, ya que nos permite tener una mayor conexión entre las entidades. La actualización realizada a las diferentes librerías nos permite mantener el sistema seguro y con las últimas mejoras en cuanto a rendimiento de algunos componentes. Finalmente, los administradores ahora tienen una nueva forma de gestionar los principales datos que hay almacenados, además de poder exportar de una forma rápida y sencilla las imágenes junto con sus datos en un directorio.

El mayor problema que se me presentó con este Trabajo de Fin de Grado y este

proyecto fue el utilizar y entender un sistema con muy poca documentación y desarrollado por una persona con la que no he tenido ningún tipo de contacto. Por ello, entender el código y poner el sistema a funcionar me llevaron más tiempo del previsto inicialmente. Además, el código no contaba con tests, por lo que cualquier cambio que realizaba en el mismo debía probarlo manualmente. Adicionalmente, adaptarme al estilo de programar del código que ya había en el proyecto también hizo que al principio me costara más.

Pese a todas las dificultades comentadas, y que era la primera vez participando en un proyecto *fullstack* como este, me ha servido para entender de una mejor manera el funcionamiento y las utilidades de *Python* como lenguaje con el uso de redes neuronales y modelos de *Deep Learning*, y con él también he aprendido muchas cosas sobre la inteligencia artificial y sus posibilidades. Además, me ha permitido aprender el uso de componentes tipados en *React* con *Typescript* y el funcionamiento de las librerías del lenguaje. Por último, el uso de *docker* en el *backend* me ha ayudaddo a entender del todo el funcionamiento de los contenedores y sus interconexiones, ya que lo había aprendido de manera teórica previamente, pero nunca había tenido la posibilidad de usarlos en un proyecto real.

4.2. Líneas futuras

Con la inclusión de la posibilidad de que el usuario pueda incluir opciones sobre el procesado de la imagen y la creación de una tabla en la base de datos relativa a las imágenes explicativas del diagnóstico, se abre una línea de futuro desarrollo. Esto permitirá que se puedan estudiar nuevos modelos de aprendizaje automático y proporcionar al usuario una explicación del diagnóstico realizado mediante los *saliency maps*. Además de esto, me han surgido algunas ideas para mejoras futuras de la aplicación. La posibilidad de que el usuario pueda gestionar sus imágenes, eliminar las que no le hagan falta ya o poder editar los datos introducidos es una mejora que aumenta la funcionalidad en gran medida y no es demasiado costosa. Añadido a esto, se podría añadir la posibilidad de que un usuario, en caso de tener dudas con un diagnóstico, pueda enviar el resultado proporcionado por el sistema a otro médico y que este lo evalúe.

Por último, más a largo plazo, se podrían implementar otras redes neuronales para el diagnóstico de otras enfermedades oculares y así aumentar las funcionalidades de la aplicación enormemente: aprovechando que el médico le ha realizado una retinografía al paciente y la sube al sistema, se le podría proporcionar un informe de la evaluación de esa imagen referentes a las enfermedades oculares pertinentes.

Capítulo 5

Summary and Conclusions

5.1. Conclusion

Early disease diagnosis is a pending task in today's medicine and, due to deep learning, specialists can get help from a trained neural network to fulfill their objective and prevent illnesses. This way artificial intelligence adds a new field in which it can influence enormously in the coming years and, with this kind of applications people can start seeing it as a tool and not as threat to their jobs. Referring specifically to glaucoma, early detection is really tough considering there are no precise or specific synthoms, that is why extremely experienced doctors are needed for this job. With deep learning models this job gets highly simplified, as long as there is a correctly labeled and of a significant amount set of images.

This web application supplies specialists the possibility of, uploading a retinography, obtain an automatic diagnosis about if that eyeball has glaucoma. Everything works thanks to a convolutional neural network, working as a service and to which calls are made every time an image is uploaded. These also allow the neural network to continue improving it's diagnoses, taking into account it receives a correct feedback about the judgement from the expert.

The main improvements done to this web application let the customer have a better user experience. With the inclusion of users, the system becomes more custom to each one, and adding that to the analized images history implemented, the doctor is now able to check already finished diagnoses, next to the relative data, in addition to being able to compare them all one to another to obtain overall better results. Furthermore, the new metadata introduced referring to the image and to the patient, allows the expert to identify all the finished diagnoses, thus include new comments on the relevant reports. All of this is possible due to the relational database, that grants us with the possibility to connect in a better way different entities. The update done to some frameworks boosts our performance and reduces the risks of a leak in some components. Lastly, administrators are now able to manage the main data saved in the database, on top of having the ability to export the images and its data from the database in a fast and easy process to a directory.

The greatest problem I encountered during this project was having to use and understand a system with very little documentation and developed by a person I did not have any kind of touch. Therefore, having a complete idea of all the code that was written and starting up the system were really time consuming, more than I firstly expected. Moreover, the code had no tests and that made me test manually any change I did. Moreover, fitting myselve to the coding style that was already in the project made it more difficult for me

in the beginning.

Despite all os the complications already stated, and that it was my first time taking part of a fullstack project like this one, it made me understand in a better way about Python's functionning and utilties as a language used in most of the neural network activities and deep learning models, along with learning about artificial intelligence and its wide range of possibilities. On top of that, it gave me the possibility to discover about tiped components in React as a result of using it on top of React, and the operations that can be done with add-ons of this language. Finally, using docker in the backend helped me understand containerization and the way to connect containers between them, I had read and learned about this technique previously, nevertheless only theoretically, and in this project I could put this knowledge to practice.

5.2. Future activities.

Including the possibility for the user to include options about the image processing and creating a table in the database referring to explanatory images of the diagnosis opens a future development path. This will grant that new automatic learning models can be studied and provide users the chance to get an explication about the diagnosis thanks to saliency maps. Moreover, I have come up with some ideas to improve the application in the future. Giving the user the capability of managing their images, deleting those they will not need anymore or modifying the relative data would be a meaningful improvement that is not very time consuming. On top of that, it would be excellent for users to send images and diagnoses to other experts within the application and get their opinion, whenever they have any doubts about the result.

Finally, on a longer run, the system could have other neural networks implemented for diagnosing other eye diseases, and, thanks to this, enlarge remarkably the services of the application: taking advantage of the images that are already taken and uploaded to the system, it could produce an evaluation report of that image regarding all the relevant eye diseases.

Capítulo 6

Presupuesto

6.1. Presupuesto

Concepto	Coste por horas	Horas	Coste total
Documentación	20	60	1200€
Diseño de la base de datos	20	50	1000€
Implementación de mejoras en la interfaz de usuario	20	90	1800€
Implementación de mejoras en la API	20	70	1400€
Implementación de la base de datos y adaptación del servicio de docker	20	40	800€
Desarrollo de funcionalidades para administradores (Script y panel)	20	35	700€
	Total	345	6900€

Tabla 6.1: Presupuesto del proyecto

Capítulo 7

Bibliografía

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Alayon, S., Diaz-Aleman, T., Angel-Pereira, D., Sigut, J., Arnay, R., and Fumero Batista, F.-J. (2020). Rim-one dl: A unified retinal image database for assessing glaucoma using deep learning. *Image Analysis & Stereology*, 39(3):161–167.

Arvelo García, C. (2020). Aplicación web para diagnóstico de glaucoma basado en deep learning a partir de la recopilación de imágenes de retina. Memoria del trabajo de fin de grado correspondiente a la titulación: Grado en Ingeniería Informática de la Universidad de La Laguna:`https://riull.ull.es/xmlui/handle/915/21337`.

Auth0 (2020). Auth0 website. `https://auth0.com/`. Accessed: 2022-06-10.

Chollet, F. et al. (2015). Keras. `https://github.com/fchollet/keras`.

Conner-Simons, A. (2017). Using artificial intelligence to improve early breast cancer detection. *https://news.mit.edu/*.

Diaz Aleman, V.-T., Fumero, F., Alayon Miranda, S., Angel Pereira, D., Arteaga Hernandez, V., and Sigut Saavedra, J.-F. (2021). Analisis de la capa de celulas ganglionares con deep learning en el diagnostico de glaucoma. *Archivos de la Sociedad Española de Oftalmologia*, 96(4):181–188.

Docker (2013). Web de docker. `docker.com`. Accessed: 2022-06-10.

Flask (2010). Documentación de flask online. `https://flask.palletsprojects.com/en/2.1.x/`. Accessed: 2022-06-10.

Goodfellow, I. J., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA, USA. `http://www.deeplearningbook.org`.

MongoDB (2009). Web de mongodb. `https://www.mongodb.com/es`. Accessed: 2022-06-10.

PostgreSQL (1996). Postgresql website. https://www.postgresql.org/. Accessed: 2022-06-10.

React (2012). React website. https://es.reactjs.org/. Accessed: 2022-06-10.

Sprockel, J. J., Diaztagle, J. J., Alzate, W., and González, E. (2014). Redes neuronales en el diagnóstico del infarto agudo de miocardio. *Revista Colombiana de Cardiología*, 21(4):215–223.

Typescript (2012). Typescript website. https://www.typescriptlang.org/. Accessed: 2022-06-10.