GRADO EN INGENIERÍA INFORMÁTICA

HERRAMIENTA PARA AYUDAR A LA DETECCIÓN Y EL CONTEO DE PARTÍCULAS DE POLEN EN MUESTRAS BIOLÓGICAS

TOOL TO HELP TO DETECT AND COUNT POLLEN PARTICLES IN BIOLOGICAL SAMPLES

Realizado por

Antonio Jesús Chaves García

Tutorizado por

Enrique Soler Castillo

Cristian Martín Fernández

Departamento

Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, junio de 2021

Resumen

El objetivo general de este Trabajo de Fin de Grado ha sido desarrollar una aplicación junto al Grupo de Investigación ERTIS. Esta aplicación tiene como principal función que científicos del departamento de aerobiología puedan subir imágenes de muestras de polen de microscopio, para que sean analizadas en la aplicación y puedan obtener un resultado estimado para cuando no existan recursos humanos para analizar dichas muestras.

El ciclo de uso de la aplicación consiste en: 1) la subida de imágenes de microscopio, asignando algunos detalles y el análisis de estas; 2) registro de porcentajes aproximados de los distintos tipos de polen por rangos de fecha; 3) generación de estadísticas a partir de las imágenes proporcionadas y los datos en la plataforma.
Todo esto bajo una capa de usuarios, donde cada usuario tendrá únicamente acceso los resultados de sus imágenes, sus porcentajes y sus estadísticas.

El hecho de analizar muestras de polen de manera autónoma surge de la necesidad de obtener una respuesta rápida y fiable sobre la cantidad de partículas de polen que existen en una muestra diaria. Esta necesidad se puede deber a dos motivos, a que se quiera corroborar la cantidad de partículas contadas por un experto, o por la ausencia de expertos que puedan contar las partículas. Independientemente del motivo, se estaría automatizando un proceso, ahorrando tiempo y dinero.

Palabras clave: **Visión por Computador, OpenCV, Aplicación Web**

Abstract

The main objective of this project has been to develop an application together with the ERTIS Research Group. The main function of this application is that scientists from the aerobiology department can upload images of microscope pollen samples, so that they can be analysed in the application and can obtain an estimated result for when there are no human resources to analyse said samples.

The application's lifecycle consists in 1) uploading microscope images, assigning some details, and analysing them; 2) record of approximate percentages of the different types of pollen by date ranges; 3) generation of statistics from the images provided and the data on the platform.
All this under a layer of users, where each user will only have access to the results of their images, their percentages, and their statistics.

The reason of analysing pollen samples autonomously arises from the need to obtain a fast and reliable answer on the amount of pollen particles that exist in a daily sample. This need may be due to two reasons, because it is necessary to corroborate the quantity of particles counted by an expert, or because of the absence of experts who can count the particles. Regardless of the reason, a process is being automated, saving time and money.

Keywords: **Computer Vision, OpenCV, Web Application**

Índice de contenidos

Índice de figuras

Índice de tablas

Acrónimos

TFG	Trabajo de Fin de Grado
API	Application Programming Interface
REST	Representational State Transfer
JSON	JavaScript Object Notation
HTML	HyperText Markup Language
CSS	Cascading Style Sheet
URL	Uniform Resource Locator
CPU	Central Processing Unit
GPU	Graphics Processing Unit

1

Introducción

1.1. Motivación

LifeWatch ERIC es un consorcio europeo de infraestructura que proporciona facilidades a la investigación en e-Science a científicos que buscan aumentar nuestro conocimiento y profundizar en nuestra comprensión de la organización de la biodiversidad y las funciones y servicios de los ecosistemas para ayudar a la sociedad civil a abordar los desafíos planetarios clave.

Con LifeWatch ERIC, la Comisión Europea busca abordar, mediante inversiones a largo plazo, los factores globales responsables de la pérdida continua de diversidad biológica y el funcionamiento de los ecosistemas, con impactos en el bienestar y el desarrollo de la sociedad actual. Comprender la evolución y las funciones de la biodiversidad y los servicios de los ecosistemas es de crucial importancia hoy en día. Se busca comprender las complejas interacciones entre las especies y el medio ambiente, aprovechando los sistemas informáticos de alto rendimiento, Big Data y el desarrollo de herramientas para implementar medidas de gestión destinadas a preservar la vida en la Tierra [1].

La visión por computador es una disciplina científica que incluye métodos para adquirir, procesar y analizar las imágenes del mundo real con la finalidad de que estas puedan ser tratadas por un ordenador. Gracias a esto, se permite la automatización de

una gran variedad de tareas. Además, en la visión por computador se encuentran basadas distintas tecnologías usadas actualmente, como el reconocimiento de objetos o la detección de sucesos, entre muchas otras [2].

Se llama aplicación Web a las aplicaciones software que los usuarios pueden usar accediendo a un servidor Web usando un navegador. Este tipo de aplicaciones son cada vez más populares, debido a lo práctico que es un navegador web como cliente ligero, así como otras ventajas, tanto para el usuario, como para los desarrolladores. Cabe destacar que una página web puede contener elementos que permitan una comunicación activa entre el usuario y la información, permitiendo al usuario acceder a los datos de manera interactiva.

Comentando brevemente las ventajas de las aplicaciones web con respecto a las aplicaciones de escritorio convencionales, se encuentran las siguientes:

- Ahorran costes de hardware y software.
- Por lo general, fáciles de usar.
- Escalables y de rápida actualización
- Por lo general, provocan menos errores y los datos están más seguros.

1.2. Objetivos

El objetivo general del proyecto es el desarrollo de una aplicación web que permita la subida de imágenes directamente extraídas del microscopio, y su análisis de manera sencilla y transparente de cara al usuario. Para ello, se han definido algunos objetivos:

1. El sistema deberá proporcionar un medio para la subida de imágenes de microscopio en el propio formato de este.

2. El sistema deberá ser capaz de leer estas imágenes y dar opciones al usuario sobre cómo quiere analizar dicho fichero.

3. El sistema deberá de poder analizar las imágenes según los datos introducidos por el usuario.

4. El sistema deberá permitir al usuario definir los tipos de polen que desee.

5. El sistema deberá permitir al usuario definir porcentajes aproximados de tipos de polen en los rangos de fecha que desee.

6. El sistema deberá, a partir de los rangos mencionados anteriormente y de los datos de los análisis, ser capaz de generar algunas estadísticas y gráficos.

7. El sistema deberá ser capaz de, por cada usuario, mostrarle únicamente los datos de sus análisis.

1.3. Estructura de la memoria

Este Trabajo de Fin de Grado está organizado en 9 capítulos, entre los cuales se encuentra el capítulo de Introducción, que acaba de ser presentado. A continuación, se van a describir los principales aspectos del resto de ellos:

- *Capítulo 2. Metodología*
  En este capítulo se indica la metodología de trabajo utilizada en el trascurso del proyecto, así como una breve explicación.

- *Capítulo 3. Tecnologías utilizadas*
  En este capítulo se exponen brevemente las tecnologías que componen el proyecto.

- *Capítulo 4. Análisis del sistema*
  En este capítulo se desarrolla un análisis del proyecto, diferenciando los distintos módulos del sistema, y mostrando las características y requisitos de cada uno.

- *Capítulo 5. Especificación*

  En este capítulo se expone todo lo necesario para una correcta implementación del sistema.

- *Capítulo 6. Diseño*

  En este capítulo se define el diseño que se ha llevado a cabo para la posterior implementación del sistema.

- *Capítulo 7. Implementación*

  En este capítulo se muestra la implementación realizada del sistema.

- *Capítulo 8. Evaluación*

  En este capítulo se indican las diversas pruebas de rendimiento y precisión realizadas al sistema.

- *Capítulo 9. Conclusiones y líneas futuras*

  En este capítulo se exponen las conclusiones obtenidas tras finalizar el proyecto, así como las líneas futuras para una posible ampliación de este.

2

Metodología

En este capítulo, se detalla la metodología de trabajo empleada en el desarrollo del sistema.

Un proyecto de desarrollo de software lleva consigo varias fases, que son la recogida de requisitos, el desarrollo, las pruebas y el mantenimiento entre otros. Para lograr que el producto desarrollado sea de calidad y realmente el deseado, lo ideal es adaptarse a una metodología de desarrollo de software.

Las metodologías de desarrollo de software son un conjunto de filosofías, técnicas y métodos organizativos aplicables al desarrollo de proyectos software. El objetivo de estas metodologías es tratar de gestionar las distintas fases de los proyectos software de la mejor manera posible, para realizar el trabajo de manera eficiente y con la mayor calidad posible.

Durante este proyecto, se ha seguido una metodología de tipo ágil. Este tipo de metodologías son las más utilizadas en la actualidad, dada su alta capacidad de flexibilidad e inmediatez en la respuesta a las diferentes necesidades que surjan en el proyecto. Las metodologías ágiles están basadas en el desarrollo incremental, donde en cada ciclo de desarrollo se implementan nuevas funcionalidades de la aplicación final [3].

2.1. Metodología ágil

El concepto de metodología ágil surge de la necesidad de agilizar los métodos tradicionales del desarrollo del software, ya que las metodologías anteriores eran muy estáticas. La metodología ágil pretende acabar con esto, siendo esta una fórmula que destaca por su rapidez y flexibilidad, adecuándose en la medida de lo posible siempre a las necesidades del cliente.

En los desarrollos que usan esta metodología, no se suele planificar ni diseñar por adelantado, sino que se evoluciona según unos bucles de retroalimentación. Durante estos periodos, se diseñan y se desarrollan las determinadas tareas y al final de este, se entregan los avances al cliente y se comienza de nuevo el proceso. De esta forma, el cliente va recibiendo poco a poco las últimas novedades de su producto, y puede agregar cambios o establecer prioridades [4].

Figura 1: Ciclo de vida de un proyecto ágil.

3

Tecnologías utilizadas

En este capítulo se describen las diversas tecnologías utilizadas en el desarrollo del proyecto.

3.1. Python

Python [5] es un lenguaje de programación interpretado, cuya filosofía se centra en una sintaxis que favorezca la legibilidad del código. Se trata de un lenguaje de tipado dinámico y multiparadigma, ya que soporta programación imperativa, programación orientada a objetos y en menor medida, programación funcional. Tiene una licencia de código abierto y está disponible para Windows, Mac y Linux.

Figura 2: Logotipo de Python.

Es un lenguaje simple y de sintaxis clara, haciendo posible que cualquier principiante pueda empezar a programar. Es de propósito general, estando preparado para cualquier tipo de aplicación. Es un lenguaje interpretado, es decir, no hay que compilarlo. Gracias a esto, se tiene un desarrollo más rápido, aunque, por el contrario, se tendrá una ejecución más lenta con respecto a los lenguajes compilados [6].

3.2. OpenCV

OpenCV (Open Source Computer Vision Library) [7] es una librería open-source de visión por computador. OpenCV se creó para proveer una infraestructura común para aplicaciones de visión por computador y acelerar su uso en productos comerciales.

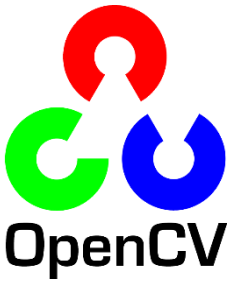

Figura 3: Logotipo de OpenCV.

La librería tiene más de 2500 algoritmos optimizados, que incluyen tanto algoritmos clásicos de visión por computador como algoritmos de "Machine Learning". Es usada por una multitud de compañías, grupos de investigación y cuerpos gubernamentales. Tiene soporte para C++, Python, Java y MATLAB, y puede usarse en cualquier sistema operativo.

3.3. Django

Django [8] es un framework web de alto nivel de código abierto, escrito en Python y que permite el desarrollo rápido de sitios webs seguros y mantenibles. Este framework respeta el patrón de diseño MVC (modelo-vista-controlador). El principal objetivo de Django es facilitar la creación de webs complejas, sin necesidad de reinventar lo ya existente.

Figura 4: Logotipo de Django.

La librería Django Rest Framework permite que se use el servidor Django como una API REST de una manera fácil y sencilla, permitiendo, como se verá posteriormente, usar otros servicios como un servidor front-end como Angular, de una manera sencilla y eficaz.

3.4. Angular

Angular [9] es un framework open-source desarrollado por Google para facilitar la creación y programación de aplicaciones web de una sola página, las webs SPA (Single Page Application). Su principal objetivo es aumentar las aplicaciones basadas en navegador con la capacidad del patrón de diseño Modelo Vista Controlador (MVC), en un esfuerzo para hacer que el desarrollo sea más sencillo y menos repetitivo. El lenguaje principal de programación de Angular es TypeScript.

Figura 5: Logotipo de Angular.

Angular se basa en clases llamadas "Componentes", cuyas propiedades son las usadas para hacer la unión de los datos. En dichas clases, existen propiedades (variables) y métodos (funciones a llamar).

3.5. TypeScript

TypeScript [10] es un lenguaje de programación de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript, una de las herramientas más usadas del mundo, y le añade programación orientada a objetos y tipos estáticos. Puede ser usado para aplicaciones tanto del lado del cliente como del servidor. El código TypeScript es transformado a código JavaScript a través del compilador Babel, dando como resultado un código ejecutable en cualquier lugar donde se pueda ejecutar JavaScript.

Figura 6: Logotipo de TypeScript.

4

Análisis del sistema

Antes de empezar cualquier proyecto de desarrollo software, es necesario realizar ciertas tareas que determinaran decisivamente el éxito de este. Dichas tareas son la planificación y el análisis del sistema.

Primeramente, hay que adentrarse en una fase de planificación. Algunas de las tareas de esta fase incluyen actividades como la planificación temporal, la estimación del coste del proyecto o la asignación de recursos y personal entre otros.

Posteriormente, y una vez planificada nuestra manera de trabajar, se pasa al análisis del sistema a desarrollar. En esta parte se trata de averiguar qué es exactamente lo que tiene que hacer el software, adaptándose a las necesidades del cliente. Por ello, esta etapa de análisis corresponde una parte vital a la hora de desarrollar cualquier proyecto software, ya que se necesita poseer una alta comprensión del trabajo que se quiere realizar.

Para ello, habrá que establecer una serie de reuniones con el cliente donde, a partir de estas, se podrán extraer requisitos, tanto funcionales como no funcionales, del producto, para así poder desarrollar la aplicación requerida con el comportamiento y el rendimiento deseado.

En este capítulo, se expondrá de manera general, una descripción del sistema junto a los requisitos funcionales y no funcionales.

4.1. Descripción general del sistema

A grandes rasgos, el sistema está formado por una serie de herramientas y módulos cuyo objetivo principal es ofrecer al usuario una herramienta de análisis de muestras de partículas de polen, apoyada en una aplicación web donde podrá subir las imágenes de manera sencilla, así como poder obtener estadísticas acerca de dichas muestras.

Para proporcionar toda la funcionalidad requerida, el sistema cuenta con una serie de módulos:

- **Servidor web.** Procesa la información enviada por el usuario y genera una respuesta a estas, accediendo, si es necesario, a la base de datos.

- **Módulo de extracción y conversión.** Dado a que las imágenes que da como salida el microscopio no son de un formato común, dado a que cada fabricante de estos tiene su propio formato, primeramente, se extraen las imágenes y se convierten a un formato conocido.

- **Módulo de análisis.** Una vez ya se tienen las imágenes convertidas a un formato común, se usan estas imágenes para, mediante técnicas de visión por computador, contar cuantas partículas de polen existen en la muestra. Posteriormente, este resultado se almacena en una base de datos.

- **Base de datos.** Todos los análisis realizados, los porcentajes estadísticos por fecha, así como los usuarios, se encuentran almacenados en una pequeña base de datos.

4.2. Requisitos funcionales

Un requisito funcional define una función de un sistema software o de alguno de sus componentes. Una función es descrita por un conjunto de entradas y salidas. Los requisitos funcionales pueden ser cálculos, detalles técnicos y otras funcionalidades específicas que el sistema debe de cumplir.

Los requisitos de comportamiento para cada requisito fundamental se muestran en los casos de uso [11].

A continuación, se detallarán los requisitos funcionales del sistema:

- **RF-1.** Los usuarios convencionales podrán subir y eliminar muestras y análisis.

- **RF-2.** Los usuarios convencionales podrán definir los tipos de polen que ellos deseen.

- **RF-3.** Los usuarios convencionales podrán definir y eliminar porcentajes aproximados de tipos de polen en los rangos de fecha que ellos deseen.

- **RF-4.** Los usuarios podrán visualizar todos sus resultados de los análisis.

- **RF-5.** Los usuarios convencionales podrán visualizar diferentes estadísticas a partir de los resultados de los análisis de sus muestras.

- **RF-6.** Los usuarios con un rol de administrador podrán crear, eliminar o modificar los perfiles de los usuarios convencionales.

4.3. Requisitos no funcionales

Los requisitos no funcionales son aquellos requisitos que imponen restricciones en el diseño o la implementación del sistema, como restricciones en el diseño o estándares de calidad. Debe de pensarse en estas propiedades como características que hacen al producto más atractivo y usable.

Estos requisitos no deben de modificar la funcionalidad principal del producto, pero si pueden influir en el desarrollo de dicha funcionalidad [12].

A continuación, se detallarán los requisitos no funcionales del sistema:

- **RNF-1.** La aplicación deberá permanecer siempre en activo a la escucha de nuevas peticiones.

- **RNF-2.** El sistema deberá estar organizado y limitado para minimizar los errores de los usuarios.

- **RNF-3.** La aplicación deberá visualizarse y funcionar correctamente en cualquier navegador de escritorio moderno.

- **RNF-4.** La aplicación deberá ser amigable y fácil de usar para los usuarios finales.

- **RNF-5.** La aplicación resultante deberá ser fácilmente ampliable o modificable en la incorporación de nuevos módulos.

5

Especificación del sistema

Esta fase consiste en la descripción con detalle del software de forma rigurosa. Para ello, se describe el comportamiento esperado del sistema, así como su interacción con los usuarios finales.

Primeramente, se hará uso de un modelo entidad-relación, donde se presenta, de manera conceptual, como se almacenan los datos dentro de la aplicación. Dentro de este diagrama, se podrá observar también la relación que tienen los datos entre ellos, así como sus atributos.

Además, y para mostrar de manera conceptual la funcionalidad del software, se utilizarán casos de uso. Un diagrama de casos de uso es una descripción de las acciones que un usuario puede realizar para llevar a cabo una tarea. Estos sirven sobre todo para poder determinar la comunicación o el comportamiento del software tras la interacción de los usuarios u otros sistemas.

En este capítulo, se incluye tanto un modelo entidad-relación, como un diagrama de casos de uso, así como una breve descripción de cada uno de ellos.

5.1. Modelo Entidad-Relación

A continuación, se detalla brevemente la funcionalidad de cada una de las entidades.

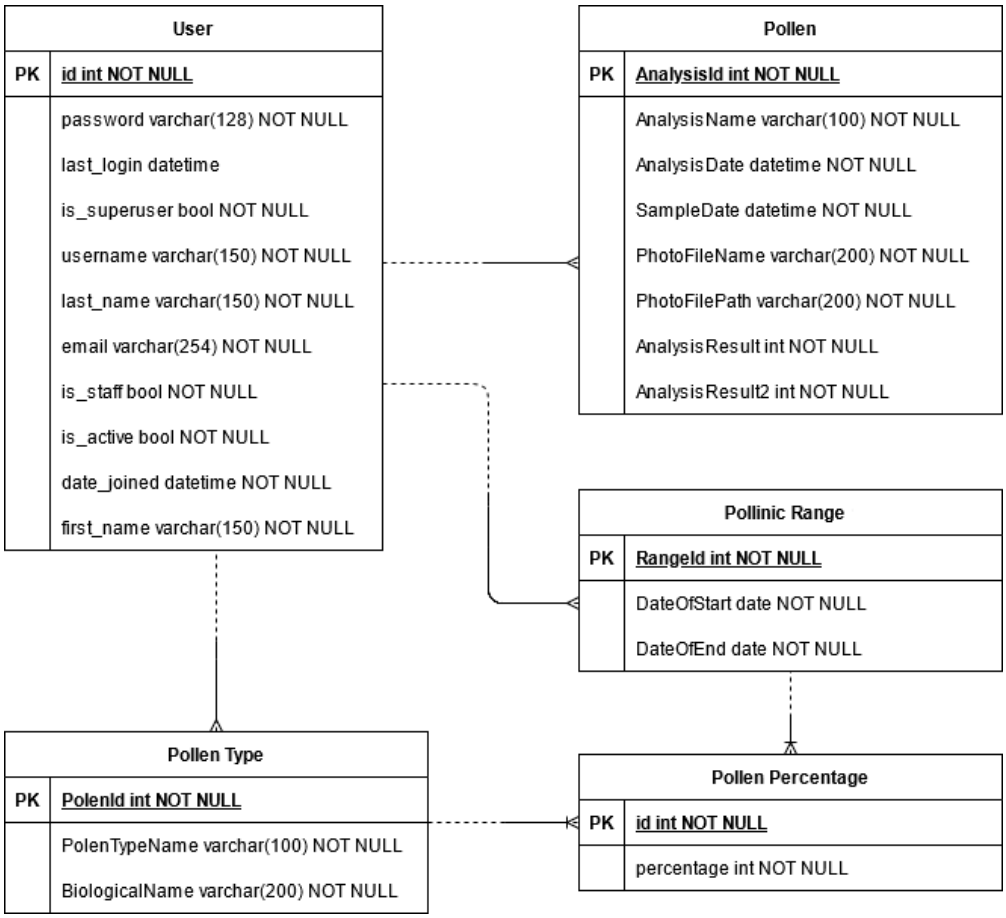

Figura 7: Diagrama Entidad-Relación del sistema.

- **User.** Es la entidad que almacena los datos de los usuarios de la plataforma. En la figura 7 se pueden apreciar los distintos parámetros que se almacenan. Hay que destacar que las contraseñas no se almacenan, sino que se almacena el hash de esta, siguiendo el algoritmo pbkdf2_sha256.

- **Pollen.** Es la entidad que almacena los datos referentes a los análisis de muestras de polen. Como comentario, se almacena el resultado en dos partes, AnalysisResult y AnalysisResult2, dado a que las muestras no representan a un solo día, aunque eso se comentará más adelante.

- **Pollinic Range.** Es la entidad que almacena los distintos porcentajes aproximados de los distintos tipos de polen por fecha**.**

- **Pollen Type.** Es la entidad que almacena los distintos tipos de polen que el usuario puede crear.

- **Pollen Percentage.** En esta entidad, se almacena el porcentaje de aparición de un tipo de polen declarado en la tabla Pollen Type, en un rango polínico declarado en Pollinic Range.

5.2. Diagrama de casos de uso

En este subapartado, se muestra el diagrama de los principales casos de uso de un usuario, tanto básico como administrador del sistema.

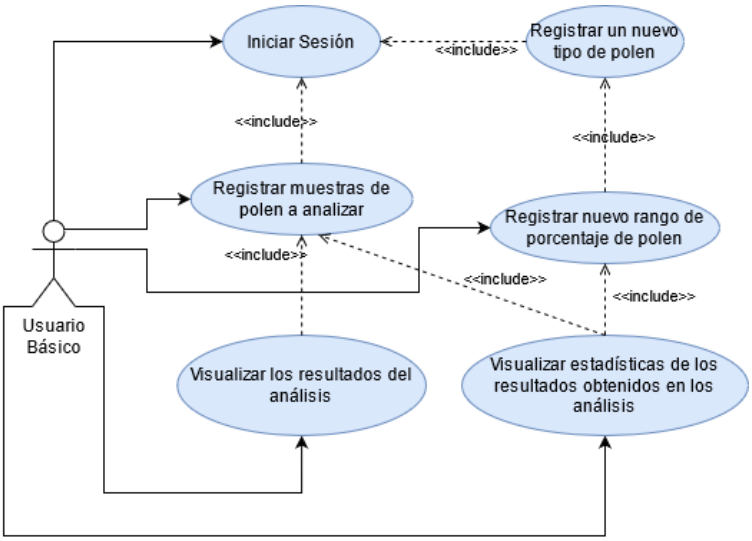

Figura 8: Diagrama de casos de uso del sistema por un usuario básico.

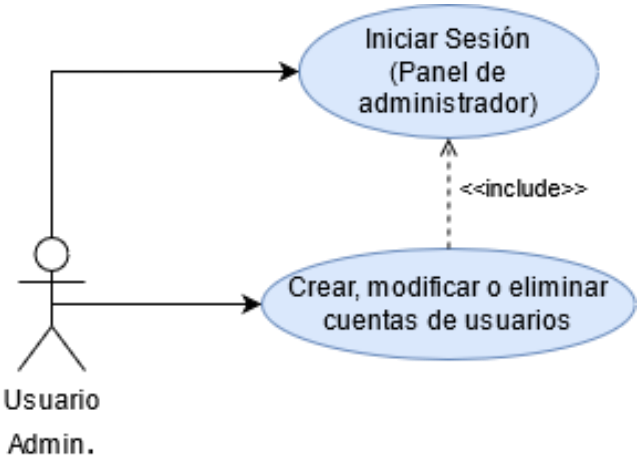

Figura 9: Diagrama de casos de uso del sistema por un usuario administrador.

5.3. Descripción de los casos de uso

En este apartado, se detallan los distintos casos de uso más relevantes del sistema. Se especifican las interacciones de los usuarios con el sistema, los prerrequisitos y el comportamiento esperado en cada caso.

Tabla 1: Descripción de Caso de Uso de Usuario Básico 1.

CU.UB.1.	**Los usuarios podrán iniciar sesión**
Contexto de uso	Inicio de sesión en la aplicación para comenzar a usarla
Ámbito	Aplicación Web de Análisis
Nivel	Usuario
Actor principal	Cualquier usuario
Participantes y objetivos: • Usuario: Iniciar sesión para poder iniciar la aplicación	
Precondiciones	Que el servidor web este iniciado
Garantías de éxito	El usuario accede a la aplicación para realizar sus trámites
Escenario de éxito principal: 1. El usuario accede a la página de inicio de sesión 2. En caso de que introduzca un usuario registrado, accede a la aplicación En caso contrario, se notifica una advertencia de usuario erróneo	

Tabla 2: Descripción de Caso de Uso de Usuario Básico 2.

CU.UB.2.	**Los usuarios podrán registrar muestras de polen para su análisis**
Contexto de uso	Cuando un usuario lo desee, puede subir muestras de polen para que sean analizadas por la aplicación
Ámbito	Aplicación Web de Análisis
Nivel	Usuario
Actor principal	Cualquier usuario
Participantes y objetivos: • Usuario: Acceder a la aplicación para añadir muestras de polen	
Precondiciones	1) Que el servidor web este iniciado 2) Que el usuario tenga la sesión iniciada
Garantías de éxito	El usuario envía correctamente su muestra junto a los parámetros necesarios
Escenario de éxito principal: 1. El usuario accede a la aplicación 2. El usuario accede al formulario pulsando el botón "Add a sample" 3. El usuario rellena el formulario y pulsa el botón "Analyse" 4. El usuario sube una muestra de polen con sus parámetros	

Tabla 3: Descripción de Caso de Uso de Usuario Básico 3.

CU.UB.3.	**Los usuarios podrán registrar nuevos tipos del polen**
Contexto de uso	Cuando un usuario lo desee, puede registrar nuevos tipos de polen para el uso estadístico de la aplicación
Ámbito	Aplicación Web de Análisis
Nivel	Usuario
Actor principal	Cualquier usuario
Participantes y objetivos: • Usuario: Acceder a la aplicación para añadir tipos de polen	
Precondiciones	1) Que el servidor web este iniciado 2) Que el usuario tenga la sesión iniciada
Garantías de éxito	El usuario envía correctamente el tipo de polen junto a los parámetros necesarios
Escenario de éxito principal: 1. El usuario accede a la aplicación 2. El usuario accede a la pestaña "Pollen types" 3. El usuario accede al formulario pulsando el botón "Add a type of pollen" 4. El usuario rellena el formulario y pulsa el botón "Add" 5. El usuario añade un nuevo tipo de polen.	

Tabla 4: Descripción de Caso de Uso de Usuario Básico 4.

CU.UB.4.	**Los usuarios podrán registrar nuevos rangos del polen**
Contexto de uso	Cuando un usuario lo desee, puede registrar nuevos rangos de polen para el uso estadístico de la aplicación
Ámbito	Aplicación Web de Análisis
Nivel	Usuario
Actor principal	Cualquier usuario
Participantes y objetivos: • Usuario: Acceder a la aplicación para añadir porcentajes de rangos de polen	
Precondiciones	1) Que el servidor web este iniciado 2) Que el usuario tenga la sesión iniciada
Garantías de éxito	El usuario envía correctamente el rango polínico junto a los parámetros necesarios
Escenario de éxito principal: 1. El usuario accede a la aplicación 2. El usuario accede a la pestaña "Pollinic Ranges" 3. El usuario accede al formulario pulsando el botón "Add a pollinic range" 4. El usuario rellena el formulario y pulsa el botón "Add" 5. El usuario añade un nuevo rango polínico con sus porcentajes	

Tabla 5: Descripción de Caso de Uso de Usuario Básico 5.

CU.UB.5.	**Los usuarios podrán visualizar los datos de los análisis**
Contexto de uso	Cuando un usuario lo desee, puede visualizar los resultados de sus análisis
Ámbito	Aplicación Web de Análisis
Nivel	Usuario
Actor principal	Cualquier usuario
Participantes y objetivos: • Usuario: Acceder a la aplicación para visualizar los resultados de los análisis de sus muestras	
Precondiciones	1) Que el servidor web este iniciado 2) Que el usuario tenga la sesión iniciada 3) Que se hayan realizado una subida prueba de las muestras
Garantías de éxito	El usuario accede a la aplicación para visualizar los resultados del análisis
Escenario de éxito principal: 1. El usuario accede a la aplicación 2. El usuario accede a la pestaña "Analysis Results" 3. El usuario puede interaccionar con los distintos análisis	

Tabla 6: Descripción de Caso de Uso de Usuario Básico 6.

CU.UB.6.	**Los usuarios podrán visualizar estadísticas acerca de los resultados de sus análisis**
Contexto de uso	Cuando un usuario lo desee, puede visualizar estadísticas de los resultados de sus análisis
Ámbito	Aplicación Web de Análisis
Nivel	Usuario
Actor principal	Cualquier usuario
Participantes y objetivos: • Usuario: Acceder a la aplicación para visualizar estadísticas de los resultados de los análisis de sus muestras	
Precondiciones	1) Que el servidor web este iniciado 2) Que el usuario tenga la sesión iniciada 3) Que se hayan realizado una subida prueba de las muestras 4) Que se hayan registrado rangos polínicos en la aplicación
Garantías de éxito	El usuario accede a la aplicación para visualizar las estadísticas asociadas a los resultados del análisis
Escenario de éxito principal: 1. El usuario accede a la aplicación 2. El usuario accede a la pestaña "Statistics"	

Tabla 7: Descripción de Caso de Uso de Usuario Administrador 1.

CU.UA.1.	**Los usuarios administradores podrán iniciar sesión en el panel de administración**
Contexto de uso	Inicio de sesión en el panel de administración para gestionar los usuarios
Ámbito	Panel de Administración de la aplicación
Nivel	Administrador
Actor principal	Usuarios con el rol de Administrador
Participantes y objetivos: • Usuario Administrador: Iniciar sesión para poder acceder al panel de administración	
Precondiciones	1) Que el servidor web este iniciado 2) Que el usuario tenga el rol de administrador
Garantías de éxito	El usuario accede al panel de administración para realizar sus trámites
Escenario de éxito principal: 1. El usuario accede a la página de inicio de sesión del panel de administración 2. En caso de que introduzca un usuario registrado con el rol de administrador, accede al panel En caso contrario, se notifica una advertencia de usuario incorrecto o sin permisos	

Tabla 8: Descripción de Caso de Uso de Usuario Administrador 2.

CU.UA.2.	**Los usuarios administradores podrán crear, modificar o eliminar cuentas de usuarios**
Contexto de uso	Cuando un usuario administrador lo desee, podrá crear, modificar o eliminar cuentas de usuarios
Ámbito	Panel de Administración de la aplicación
Nivel	Administrador
Actor principal	Usuarios con el rol de Administrador
Participantes y objetivos: • Usuario Administrador: Acceder al panel de control para crear, modificar o eliminar cuentas de usuarios	
Precondiciones	1) Que el servidor web este iniciado 2) Que el usuario tenga el rol de administrador 3) Que haya iniciado sesión correctamente en el panel de administración
Garantías de éxito	El usuario accede al panel de administración para realizar sus trámites
Escenario de éxito principal: 1. El usuario accede al panel de administración 2. El usuario presiona el botón "Users" 3. El usuario podrá gestionar los usuarios desde el panel al que ha accedido	

6

Diseño

El diseño es de los apartados más importantes a la hora del desarrollo de un proyecto software. Este proceso consiste en la definición de los componentes, de la estructura y de las características, y se define la manera de comunicarse entre ellos.

En este capítulo se trata el diseño de la aplicación web, mostrando los módulos que la componen.

6.1. Arquitectura general

La arquitectura de la aplicación consiste principalmente en dos módulos. Un módulo *front-end*, desde el cual el usuario podrá interactuar con las diversas funcionalidades de la aplicación, y un módulo *back-end*, en el cual se tratan los datos y las interacciones realizadas por el usuario usando el *front-end*. Además, dentro del módulo *back-end* se encuentran dos submódulos, uno asociado a la conversión de las imágenes de microscopio a un formato más común, así como otro asociado al análisis de las imágenes. Tener esta separación en submódulos es útil ya que, si en un futuro se desea reemplazar alguno de estos módulos por otro, se podrá hacer sin mucho problema.

Al igual que las ventajas de la separación en módulos del *back-end*, se puede extrapolar dicha ventaja a la separación en módulos *front-end* y *back-end*, ya que gracias

a esto se puede, por ejemplo, modificar la interfaz que ve el usuario sin que esto afecte a la forma en la que se tratan los datos en el otro módulo.

En la Figura 10 se muestra un diagrama con la arquitectura general del sistema.

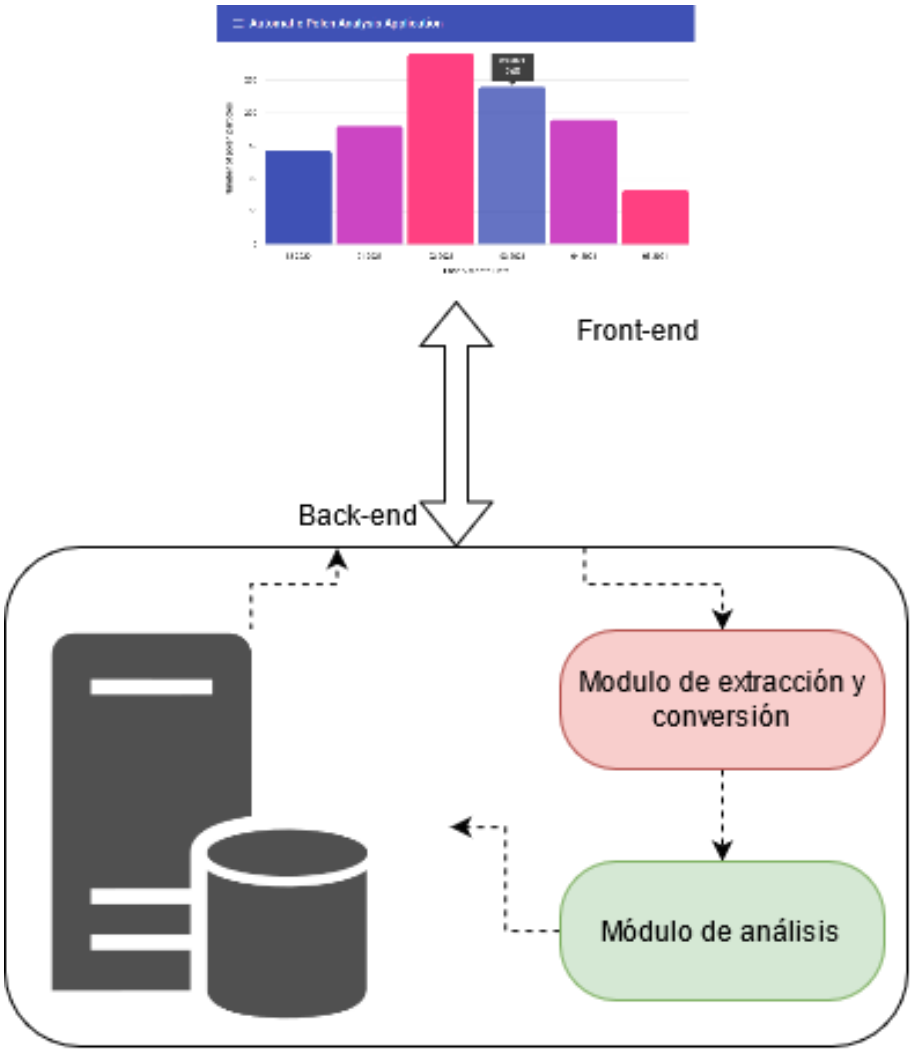

Figura 10: Arquitectura general de la aplicación.

6.2. Módulo front-end

Este componente es el que se encarga de mostrar de manera elegante los datos transmitidos por el módulo back-end, para que el usuario pueda ver e interactuar con la aplicación. En este módulo se definen cada una de las vistas que forman parte de la aplicación. Todo este módulo ha sido desarrollado haciendo uso del framework de desarrollo web Angular.

Cada una de estas vistas cuenta con distintos ficheros. Primeramente, se encuentran los ficheros con formato *.html* y *.css*, que definen el diseño que tendrán las distintas pantallas. Además, también se cuenta con ficheros de formato *.ts* y *.spec.ts*, los cuales implementan la lógica necesaria para que la vista reaccione a los impulsos del usuario, así como a los datos que llegan desde el módulo back-end.

Aparte de las distintas vistas que hay en la aplicación, existe un servicio general que es el que se usará en las distintas vistas para hacer llamadas al módulo back-end enviando y recibiendo información.

6.3. Módulo back-end

Este componente es la piedra angular de la aplicación web, y tiene un papel fundamental puesto a que posee prácticamente toda la lógica necesaria para que la aplicación funcione correctamente. En él se han definido, mediante el uso del framework de desarrollo para aplicaciones web Django, cada una de las diversas entidades que forman la base de datos de nuestra aplicación. En este módulo además se implementan todos los servicios que ofrece la aplicación a través de una API REST. Para hacer más sencillo el desarrollo de esta API REST, se ha utilizado la librería Django REST Framework, que facilita el desarrollo de APIs REST [13].

Por último, y como ya se comentó brevemente en anteriores capítulos, en este módulo se encuentran dos submódulos dedicados a la extracción y conversión de imágenes de microscopio a un formato común, así como un módulo de análisis de las muestras. A continuación, se procederá a describir detalladamente dichos módulos.

6.3.1. Módulo de extracción y conversión

En este pequeño módulo se llevan a cabo las tareas, como su propio nombre indica, de extracción y conversión de las imágenes. Lo primero que hace es recibir una imagen de microscopio, en este caso, como el equipo de aerobiología de la Universidad de Málaga trabaja con equipos de la marca Olympus, estos tienen un formato propietario llamado *Olympus CellSens VSI*. Este conjunto de archivos tiene dentro de si las distintas imágenes que ha tomado el equipo. Para ello, usando diversas llamadas la librería Bio-Formats, se extraen y se convierten las imágenes pertinentes a un formato

más usual y tratable posteriormente por las distintas librerías disponibles. El formato elegido ha sido OME-TIFF, ya que es un formato sin apenas compresión, y es el que mejor resultados otorgaba.

6.3.2. Módulo de análisis de muestras

Este módulo se encarga del análisis de las distintas imágenes obtenidas previamente del módulo descrito anteriormente. Para ello, se hace principalmente uso de la librería OpenCV.

A continuación, se va a describir brevemente el funcionamiento de este módulo, usando como apoyo algunas imágenes.

En primer lugar, como ya se ha comentado, se toma la imagen extraída del módulo de extracción y conversión y se transforma del formato RGB al formato HSV. Se realiza esta transformación debido a que, filtrar por colores en RGB, es complicado a la par que ineficiente, mientras que, filtrar por rangos de colores en HSV es mucho más sencillo. Por ello, se transforma la imagen a este formato y se filtra para que únicamente queden en la imagen colores rosados.

Hay que destacar que, se filtra por colores rosados debido a que el grupo de aerobiología de la Universidad de Málaga aplica una serie de soluciones reactivas sobre las muestras que hacen que los granos de polen aparezcan de un color rosado.

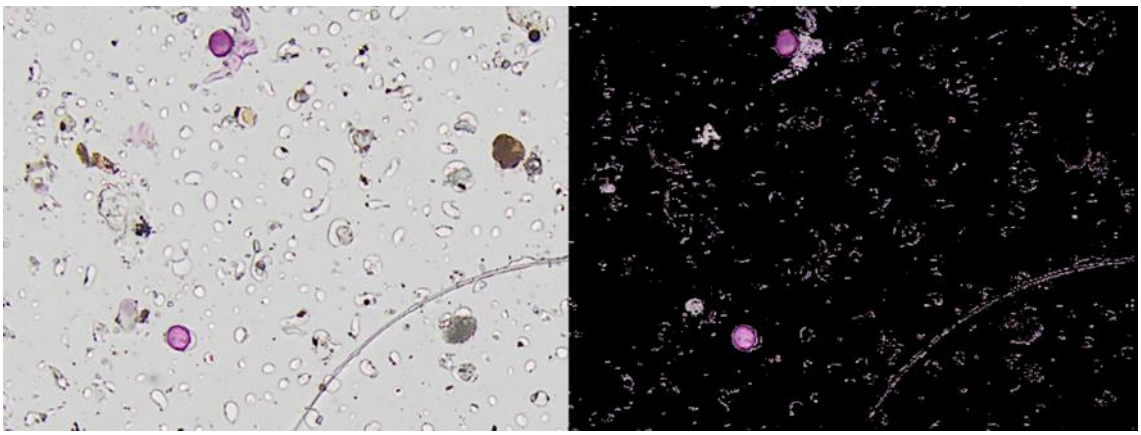

Figura 11: Ejemplo de imagen inicial y tras el filtrado.

Como se puede observar, pese a que se ha limpiado gran cantidad de los elementos innecesarios de la imagen, aún queda gran cantidad de ruido. Por ello, lo que se realizará a continuación será una reducción del ruido. Para ello, se aplica lo que en tratamiento de imágenes se conoce como "Opening", que consiste en aplicar primeramente una erosión y posteriormente una dilatación.

Una erosión sigue la idea básica de la erosión de la tierra. Se aplica a la imagen un kernel donde, si todos los pixeles que la rodean son 1, ese pixel será un 1, pero si no se cumple esto, se transforma a un cero. Una dilatación es prácticamente lo contrario a esto. Con que algún pixel de los que rodea sea 1, ya será 1.

A continuación, se muestra una imagen que ayudará a entender el concepto.

Figura 12: En orden: Imagen original, imagen erosionada e imagen dilatada.

Una vez ya conocido el funcionamiento básico de estas transformaciones, el Opening es básicamente aplicar una erosión y posteriormente una dilatación, eliminando así cualquier ruido menor de nuestra imagen, y posteriormente reconstruyendo los pixeles erosionados que forman parte de las partículas de polen.

En la figura 13 se muestra un ejemplo del proceso mencionado aplicado.

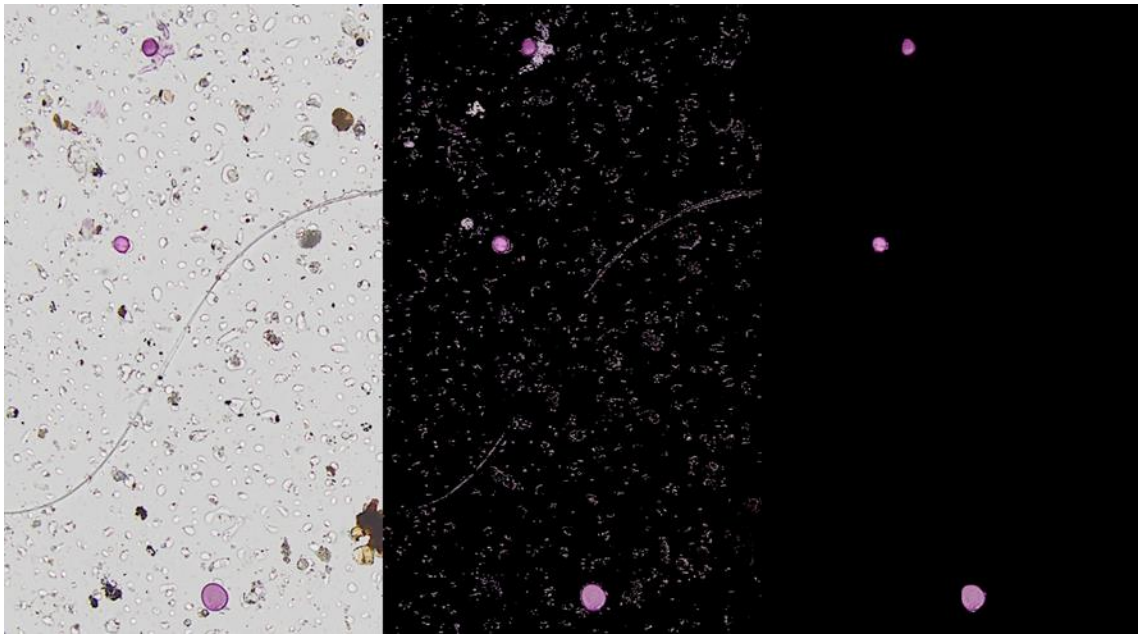

Figura 13: Aplicación de los distintos pasos para el filtrado.

Por último, solo quedaría contar las partículas de polen. Para ello, y como en todo este tiempo se han estado generando imágenes binarias (mascaras) para filtrar en la imagen, se va a usar la última máscara generada tras la limpieza de ruido para contar cuantas partículas hay. Para ello, se usa la función *findContours* de OpenCV, que detecta puntos continuos que tienen el mismo color e intensidad y los agrupa. Luego simplemente se ha de contar cuantos conjuntos de puntos se tienen, y este número final sería la cantidad de partículas de polen tiene la muestra.

Como dato, si se dibujan los puntos encontrados sobre la imagen original, se obtienen los siguientes resultados.

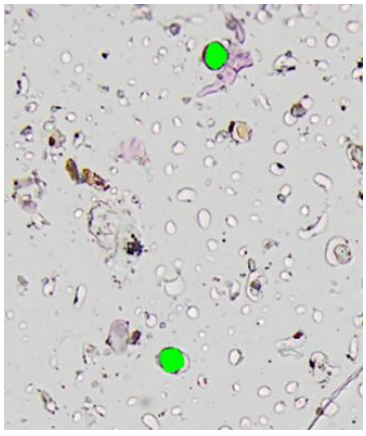

Figura 14: Puntos obtenidos por findContours dibujados en la imagen.

7

Implementación

Tras haberse especificado y diseñado lo que el sistema debe de hacer y cómo van a trabajar los diferentes módulos de la aplicación, ahora se va a comentar como se han implementado dichos módulos para que la aplicación funcione como se ha contemplado. Todo lo diseñado en los capítulos anteriores es la guía que se debe seguir para construir los módulos y funciones del sistema.

Por ello, en este capítulo se explica la implementación de las diferentes clases y funciones que forman la aplicación. Se irán mostrado imágenes del código de la aplicación, así como comentarios para que se pueda entender su funcionamiento.

7.1. Módulo back-end

A continuación, se exponen las clases principales de este módulo.

7.1.1. Models

Los objetos de esta clase representan el esquema de la base de datos de la aplicación web. En ella, con la ayuda de Django, se especifican cada una de las entidades del sistema y las relaciones que existen entre ellas, como ya se vio en el diagrama entidad-relación.

A continuación, se detallan las distintas entidades que la forman y sus atributos.

- **User.** Esta entidad es una entidad por defecto de Django, ideal para el tratado de usuarios en la plataforma. La entidad está formada por:
  - *id:* Identificador general del usuario.
  - *username:* Nombre de usuario.
  - *first_name:* Nombre real del usuario.
  - *last_name:* Apellido real del usuario.
  - *email:* Dirección de correo electrónico del usuario.
  - *password:* Hash + salt de la contraseña del usuario
  - *is_staff:* Flag que indica si el usuario pertenece al staff.
  - *is_active:* Flag que indica si el usuario está en activo
  - *is_superuser:* Flag que indica si el usuario es un superusuario
  - *last_login:* Fecha de la última conexión del usuario (solo administradores)
  - *date_joined:* Fecha de la creación del usuario.

El siguiente fragmento de código muestra una representación de la implementación de dicha clase, ya que esta ya se encuentra implementada [14].

```
class User(model.Model):
    id = models.Autofield(primary_key = True)
    username = models.CharField(max_length=150)
    first_name = models.CharField(max_length=150, blank = True)
    last_name = models.CharField(max_length=150, blank = True)
    email = models.CharField(max_length=254, blank = True)
    password = models.CharField(max_length=128) # Hash + Salt
    is_staff = models.BooleanField()
    is_active = models.BooleanField()
    is_superuser = models.BooleanField()
    last_login = models.DateField()
    date_joined = models.DateField()
```

Fragmento de código 1: Representación de la implementación del modelo User.

- **Pollen.** Esta entidad está formada por:
  - *AnalysisId:* Identificador general del análisis.
  - *UserId:* Identificador del usuario al que está asociado este análisis.
  - *AnalysisName:* Nombre descriptivo del análisis.
  - *AnalysisDate:* Fecha del análisis.
  - *SampleDate:* Fecha de la muestra.
  - *PhotoFileName:* Nombre del fichero subido.
  - *PhotoFilePath:* Ruta del fichero dentro del sistema.
  - *AnalysisResult:* Resultado del 58.3% de la muestra.
  - *AnalysisResult2:* Resultado del resto de la muestra.

Hay que destacar que el resultado de los análisis está separado en dos, dado a que las muestras se toman a partir de las 10h. Esto implica que de las 10h a las 23:59h (58,3% de la muestra), forma parte del día de la muestra, y de las 00h a las 9:59h (el resto de la muestra), forma parte del día siguiente.

El siguiente fragmento de código muestra la implementación de Pollen:

```
class Pollen(models.Model):
    AnalysisId = models.AutoField(primary_key=True)
    UserId = models.ForeignKey(User, on_delete=models.CASCADE)
    AnalysisName = models.CharField(max_length=100)
    AnalysisDate = models.DateField()
    SampleDate = models.DateField()
    PhotoFileName = models.CharField(max_length=200)
    PhotoFilePath = models.CharField(max_length=200)
    AnalysisResult = models.IntegerField()
    AnalysisResult2 = models.IntegerField(default=0)
```

Fragmento de código 2: Implementación de Pollen.

- **Pollinic Range:** Esta entidad está formada por:
  - *RangeId*: Identificador general del rango polínico.
  - *UserId:* Identificador del usuario que ha creado el rango polínico.
  - *DateOfStart*: Fecha de inicio.
  - *DateOfEnd:* Fecha de finalización.

```
class PollinicRange(models.Model):
    RangeId       = models.AutoField(primary_key=True)
    UserId        = models.ForeignKey(User, on_delete=models.CASCADE)
    DateOfStart   = models.DateField()
    DateOfEnd     = models.DateField()
```

Fragmento de código 3: Implementación de PollinicRange.

- **Pollen Type:** Esta entidad está formada por:
  - *PollenId:* Identificador general del tipo de polen.
  - *Pollen_Type_Name:* Nombre común del tipo de polen.
  - *Biological_Name:* Nombre biológico del tipo de polen.
  - *UserId:* Identificador del usuario al que pertenece el tipo de polen.

```
class PollenType(models.Model):
    PollenId = models.AutoField(primary_key=True)
    Pollen_Type_Name = models.CharField(max_length=100)
    Biological_Name = models.CharField(max_length=200)
    UserId = models.ForeignKey(User, on_delete=models.CASCADE)
```

Fragmento de código 4: Implementación de PollenType.

- **Pollen Percentage:** Esta entidad está formada por:
    - *PollinicRangeId:* Identificador general del rango polínico.
    - *PollenTypeId:* Identificador general del tipo de polen.
    - *Percentage:* Porcentaje de aparición del tipo de polen en el rango polínico.

```
class PollenPercentage(models.Model):
    PollinicRangeId = models.ForeignKey(PollinicRange, on_delete=models.CASCADE)
    PollenTypeId     = models.ForeignKey(PollenType, on_delete=models.CASCADE)
    Percentage      = models.IntegerField(default=0)
```

Fragmento de código 5: Implementación de PollenPercentage.

7.1.2. Serializers

Los *serializers* permiten definir, al gusto del desarrollador, las respuestas que devolverá la API ante las peticiones del módulo front-end. Esto permite que complejos modelos sean convertidos a estructuras de Python, y a su vez, puedan ser transformadas a JSON de una manera sencilla.

7.1.3. Views

En esta clase se encuentran funciones de Python, encargadas de gestionar las peticiones que envía el módulo front-end, así como de devolver respuestas a dichas peticiones, definiendo así el funcionamiento de la API REST de la aplicación.

A continuación, se muestran las distintas funciones de esta clase.

- ***polenApi.*** Esta vista tiene varias funcionalidades dependiendo del tipo de petición que se le realice.

    - Si se le realiza una petición GET, esta vista se encarga de devolver el conjunto de resultados de los análisis del usuario que lo haya solicitado.

    - Si se le realiza una petición DELETE y se le pasa como parámetro un identificador de la tabla, eliminará dicho análisis si pertenece a ese usuario

- ***uploadVSI.*** Esta vista se encarga de recibir la imagen de microscopio, descomprimirla del ZIP en el que se encuentra y posteriormente analizar los metadatos de la imagen, obteniendo así los distintos identificadores de las series que se encuentran dentro de la imagen de microscopio, para que el usuario posteriormente pueda elegir cuales de estas extraer y analizar.

```python
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def uploadVSI(request):
    """
    This view is used by the frontend module to send to the backend the file with the sample to analyse.

    """
    _, code = getUserDetails_fromJWT(request)
    if code == 401:
        return JsonResponse([], safe=False)

    file=request.FILES['uploadedFile']
    file_name = default_storage.save(file.name, file)
    file_path = default_storage.path(file_name)

    unzip_path = 'images\\'+Path(file_name).stem
    af.unzipFile(file_path, unzip_path)

    vsiFiles = af.listFiles(unzip_path, 'vsi')
    if(len(vsiFiles) == 0 or len(vsiFiles) > 1):
        return JsonResponse('Error uploading file. The VSI file cannot be found or there is more than one.', safe=False)

    responseList = []
    paths = OrderedDict()
    paths['unzip']    = unzip_path
    paths['filename'] = vsiFiles[0]
    responseList.append(paths)
    for image in af.getImageInfo(unzip_path+'\\'+vsiFiles[0]):
        images = OrderedDict()
        images['identifier'] = image[0]
        images['name'] = image[1]
        responseList.append(images)
    return JsonResponse(responseList, safe=False)
```

Fragmento de código 6: Implementación de uploadVSI.

- ***analyseSelectedImages***. Esta vista se encarga de dada la muestra almacenada previamente, y tras el envío de determinados parámetros, de analizar dicha muestra. Para ello, primeramente, se usa el módulo de extracción y conversión para extraer las imágenes de la imagen de microscopio y posteriormente se usa el módulo de análisis para analizar las distintas imágenes extraídas. El funcionamiento de estos módulos se comentará posteriormente.

```
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def analyseSelectedImages(request):
    """
    This view is used by the frontend module to send to the backend how it has to analyse the image previously sent.

    """
    userId, code = getUserDetails_fromJWT(request)
    if code == 401:
        return JsonResponse("Analysis Failed", safe=False)

    reqStr  = JSONParser().parse(request)
    if reqStr['UserId'] != userId:
        return JsonResponse("Analysis Failed", safe=False)

    parsed_route = reqStr['PhotoFilePath']+'\\parsed'
    af.parseImages(reqStr['SelectedRowsIds'], reqStr['PhotoFilePath']+'\\'+reqStr['PhotoFileName'], parsed_route)

    totalBlobsDetected = [0,0]
    if (len(reqStr['SelectedRowsIds']) != 0):
        for image in af.listFiles(parsed_route, 'ome.tiff'):
            res = af.analyseImage(image, parsed_route, 0.5833)
            totalBlobsDetected = np.add(totalBlobsDetected, res )
```

Fragmento de código 7: Implementación de analyseSelectedImages (1/2).

```
    new_pollen = OrderedDict()
    new_pollen['AnalysisId'] = reqStr['AnalysisId']
    new_pollen['AnalysisName'] = reqStr['AnalysisName']
    new_pollen['SampleDate'] = reqStr['SampleDate']
    new_pollen['AnalysisDate'] = reqStr['AnalysisDate']
    new_pollen['PhotoFileName'] = reqStr['PhotoFileName']
    new_pollen['PhotoFilePath'] = reqStr['PhotoFilePath']
    new_pollen['UserId'] = reqStr['UserId']
    new_pollen['AnalysisResult']  = totalBlobsDetected[0]
    new_pollen['AnalysisResult2'] = totalBlobsDetected[1]

    polen_serializer = PolenSerializer(data= new_pollen)
    if polen_serializer.is_valid():
        polen_serializer.save()
        return JsonResponse("Analysis Okay", safe=False)
    return JsonResponse("Analysis Failed", safe=False)
```

Fragmento de código 8: Implementación de analyseSelectedImages (2/2).

- ***rangeAPI***. Esta vista tiene varias funcionalidades dependiendo del tipo de petición que se le realice.
    - Si se le realiza una petición GET, esta vista se encarga de devolver el conjunto de los rangos polínicos del usuario que lo haya solicitado.

    - Si se le realiza una petición POST y se le pasa como parámetro un rango polínico en formato JSON, se añadirá dicho rango polínico al conjunto de estos del usuario.

    - Si se le realiza una petición DELETE y se le pasa como parámetro un identificador de la tabla, eliminará dicho rango polínico si pertenece a ese usuario

```
@api_view(['GET', 'POST', 'DELETE'])
@permission_classes([IsAuthenticated])
def rangeApi(request, id=0):
    """
    This view is used by the frontend module to get the list of pollinic ranges,
    create a new one or delete an existing one

    """
    UserId, code = getUserDetails_fromJWT(request)
    if code == 401:
        return JsonResponse([], safe=False)

    if request.method=='GET':
        polRanges = PolinicRanges.objects.filter(UserId=UserId)
        polRanges_serializer = PolinicRangesSerializer(polRanges, many=True)
        return JsonResponse(polRanges_serializer.data, safe=False)

    elif request.method=='POST':
        polRanges_data = JSONParser().parse(request)
        if polRanges_data['UserId'] == UserId:
            try:
                user = User.objects.get(id=polRanges_data['UserId'])
                pr = PolinicRanges(UserId= user, DateOfStart=polRanges_data['DateOfStart'],
                DateOfEnd=polRanges_data['DateOfEnd'])
                pr.save()
                for pt in polRanges_data['PolenTypeValues']:
                    polenType = PolenType.objects.get(PolenId=pt['PolenTypeId'])
                    polenPercentage = PolenPercentage(PolinicRangesId=pr,
                    PolenTypeId=polenType, Percentage=pt['value'])
                    polenPercentage.save()

                return JsonResponse("Added Successfully", safe=False)
            except:
                return JsonResponse("Failed to Add.", safe=False)
        return JsonResponse("Failed to Add.", safe=False)

    elif request.method=='DELETE':
        try:
            polRanges = PolinicRanges.objects.get(RangeId=id, UserId_id = UserId)
            polRanges.delete()
            return JsonResponse("Deleted Successfully!", safe= False)
        except:
            return JsonResponse("", safe= False)
```

Fragmento de código 9: Implementación de rangeApi.

- ***getLast6MonthData***. Esta vista se encarga de proporcionar al módulo front-end los datos necesarios para dibujar la estadística de la cantidad media de polen en los últimos seis meses.

- ***getDistrValues***. Esta vista se encarga de proporcionar al módulo front-end los datos necesarios para las predicciones por día, según los rangos polínicos, de cada tipo de polen.

```
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def getDistrValues(request):
    """
    This view is used by the frontend module to get the necessary information
    to make "Approximate Daily Types of Pollen" pie plot.

    """
    UserId, code = getUserDetails_fromJWT(request)
    if code == 401:
        return JsonResponse([], safe=False)

    dateToCheck  = request.GET['dateToCheck']
    getDayDataQuery =  ("select 0 as AnalysisId, sum(mean) as sum from( "
                        "select avg(AnalysisResult) as mean  "
                        "from AnalysisApp_polen "
                        "where sampleDate=='"+dateToCheck+"' "
                        "and UserId_id =='"+str(UserId)+"' "
                        "group by sampleDate "
                        "UNION ALL "
                        "select avg(AnalysisResult2)  "
                        "from AnalysisApp_polen "
                        "where sampleDate==date('"+dateToCheck+"','-1 day') "
                        "and UserId_id =='"+str(UserId)+"' "
                        "group by sampleDate)"
    )
    dateData = Polen.objects.raw(getDayDataQuery)[0].sum
    if dateData is None:
        return JsonResponse("Failed: No data for that day", safe=False)

    getRangeQuery = ("SELECT RangeId from AnalysisApp_polinicranges "
                     "WHERE '"+dateToCheck+"' between DateOfStart and DateOfEnd "
                     "and UserId_id =='"+str(UserId)+"' "
                     "order by RangeId")

    percentageId = PolinicRanges.objects.raw(getRangeQuery)
    if len(percentageId) < 1:
        return JsonResponse("Failed: No percentages found for this data and date", safe=False)
    retList = []

    getPolenValuesQuery =  ("SELECT 0 as RangeId, pt.Polen_Type_Name name, pp.Percentage value "
                            "from AnalysisApp_polinicranges pr, AnalysisApp_polentype pt "
                            ", AnalysisApp_polenpercentage pp  where pr.RangeId == pp.PolinicRangesId_id "
                            "and pp.PolenTypeId_id == pt.PolenId and pr.RangeId == "+str(percentageId[0].RangeId))

    percentageData = PolinicRanges.objects.raw(getPolenValuesQuery)

    for pr in percentageData:
        dataIter = OrderedDict()
        dataIter['name'] = pr.name
        dataIter['value'] = dateData * pr.value / 100
        retList.append(dataIter)

    return JsonResponse(retList, safe=False)
```

Fragmento de código 10: Implementación de getDistrValues.

- ***getRangeInformation***. Esta vista se encarga de proporcionar al módulo front-end la información necesaria acerca de un rango polínico.

- ***polenTypeApi***. Esta vista tiene varias funcionalidades dependiendo del tipo de petición que se le realice.
    - Si se le realiza una petición GET, esta vista se encarga de devolver el conjunto de los tipos de polen del usuario que lo haya solicitado.

    - Si se le realiza una petición POST y se le pasa como parámetro un tipo de polen en formato JSON, se añadirá dicho tipo de polen al conjunto de estos del usuario.

    - Si se le realiza una petición DELETE y se le pasa como parámetro un identificador de la tabla, eliminará dicho tipo de polen si pertenece a ese usuario.

Como se ha podido observar, en todas las vistas se está usando una función auxiliar llamada getUserDetails_fromJWT. Dado a que, en la aplicación, los usuarios deben iniciar sesión para poder usarla, se debe de comprobar que las peticiones del usuario correspondan a ese usuario, además de que sea ese usuario y no otro suplantándole.

Para ello, se ha decidido utilizar la autenticación mediante tokens JWT (JSON Web Token). Este método consiste en que cuando el usuario inicia su sesión, se le otorga un Token seguro y firmado que se usará en las diversas peticiones que realice, permitiendo al usuario acceder a los recursos que le son permitidos mediante dicho token [15].

```python
def getUserDetails_fromJWT(request):
    jwt_token = request.META['HTTP_AUTHORIZATION'][4:]
    jwt_token_decoded = jwt.decode(str(jwt_token), SECRET_KEY, algorithms="HS256")
    UserId = jwt_token_decoded['user_id']
    username = jwt_token_decoded['username']
    email = jwt_token_decoded['email']

    try:
        User.objects.get(id=UserId, username = username, email = email)
    except:
        return -1, 401 # Unauthorized

    return UserId, 200 # OK
```

Fragmento de código 11: Implementación de la función auxiliar getUserDetails_fromJWT.

7.1.4. Módulo de extracción y conversión

En este módulo se encuentran todas las funciones auxiliares necesarias previas al análisis de la muestra. A continuación, se describirán las distintas funciones:

- ***unzipFile***. Esta simple función se encarga de extraer la imagen de microscopio del ZIP en el que ha sido subida, dado a que estas imágenes poseen varios ficheros.

```python
def unzipFile(file, directory):
    with zipfile.ZipFile(file, 'r') as zip_ref:
        zip_ref.extractall(directory)
```

Fragmento de código 12: Implementación de unzipFile.

- ***getImageInfo***. Esta función se encarga, dada la ruta de la imagen extraída, de obtener las distintas series útiles que se encuentran dentro de la imagen. Esto se debe a que, dentro de las imágenes de microscopio, se encuentran diversas series (imágenes), donde algunas de ellas son iguales, pero con menor resolución. Por tanto, lo que se realiza aquí es obtener los identificadores de las diversas series con mayor resolución, así como su nombre, para que posteriormente el usuario pueda elegir cuales de ellas desea analizar.

  En el fragmento de código 13 se muestra la implementación de esta función.

```
def getImageInfo(input_image):
    javabridge.attach()

    file_full_path = input_image # Ruta del fichero
    md = bf.get_omexml_metadata(file_full_path)
    # Lectura de sus metadatos para obtener el ID de las series sin comprimir

    ome = bf.OMEXML(md)
    niceImages = []
    for serie in range(ome.get_image_count()):
        # Por cada serie que contenga nuestra imagen
        currentImage = ome.image(serie)

        acquisitionDate = None
        acquisitionDate = currentImage.get_AcquisitionDate()

        if(acquisitionDate != None):
            # Nos quedamos unicamente con las que tengan fecha de captura, es decir, las que son sin comprimir

            niceImages.append( (serie, currentImage.get_Name()) )
            # Y almacenamos ese numero de serie en una lista que despues utilizaremos para la conversión

    javabridge.detach()
    return niceImages
```

Fragmento de código 13: Implementación de getImageInfo.

- ***parseImages***. Esta función transforma las series que el usuario quiere analizar y previamente ha elegido en la aplicación. Para ello, y usando un script que llama a una librería de Java llamada BioFormats, se extraen y convierten las series a imágenes de formato .ome.tiff para que posteriormente puedan usarse como entrada para el análisis.

7.1.5. Módulo de análisis

Este módulo de una única función se encarga, una vez ya se tienen las imágenes en un formato común, de analizar dichas imágenes para contabilizar cuantas partículas de polen se encuentran en la imagen. Para ello, y como las partículas de polen tienen un color rosado dada la aplicación de una solución química, se filtran estos colores usando el espacio de color HSV. Posteriormente, se reduce el ruido de la imagen, y se cuentan cuantas partículas han quedado como resultado del filtrado.

```
def analyseImage(image_name, image_directory, percentage=0.7):

    print('Image analyzed: '+image_name)
    im = cv2.imread(image_directory+'\\'+image_name)

    # Splitting image in two different images for each day
    dims = im.shape
    cropped_images = []
    cropped_images.append(im[0:int(dims[0]*percentage), 0:dims[1]])
    cropped_images.append(im[int(dims[0]*percentage):dims[0], 0:dims[1]])

    # Colors to Filter
    lower_pink  = np.array([135,19,89])
    higher_pink = np.array([179,255,255])

    res = []
    for cp_image in cropped_images:

        # Color filtering
        hsv = cv2.cvtColor(cp_image, cv2.COLOR_BGR2HSV)
        mask = cv2.inRange(hsv, lower_pink, higher_pink)
        cv2.bitwise_and(cp_image, cp_image, mask=mask)

        # Noise reduction
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (6,6))
        opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel=kernel, iterations=2)

        cnts = cv2.findContours(opening, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = cnts[0] if len(cnts) == 2 else cnts[1]

        res.append(len(cnts))
    return res
```

Fragmento de código 14: Implementación de analyseImage.

7.2. Módulo front-end

En esta sección se muestran los distintos componentes que conforman el módulo front-end, así como su implementación. Para el desarrollo de estos módulos, son necesarios tres tipos de fichero:

- ***Fichero HTML*.** Se encarga de estructurar el contenido del componente, como los textos e imágenes.
- ***Fichero CSS*.** Se encarga de dar estilo al componente web, definiendo diversos atributos a los distintos apartados del fichero HTML correspondiente.
- ***Fichero TypeScript.*** Se encarga de almacenar las variables y funciones con las que el usuario interaccionará en la aplicación web.

A continuación, se muestran los distintos componentes que forman este módulo, así como algunos fragmentos de código relevantes.

7.2.1. Inicio de sesión

HTML + CSS

Este componente se encarga del inicio de sesión para acceder a la aplicación, para ello, se encuentran dos campos para que el usuario introduzca sus datos, y dependiendo de si los datos son válidos o no accederá a la aplicación, se le responderá con una notificación de datos no válidos.

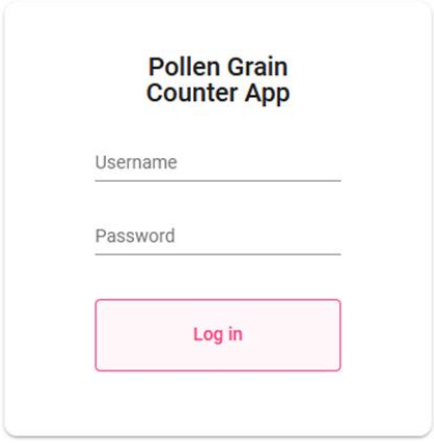

Figura 15: Vista de inicio de sesión.

TypeScript

El fichero TypeScript de este componente está compuesto por una serie de funciones las cuales se comunica con el módulo back-end para verificar si el usuario es correcto. Esta comunicación se hace a través del servicio user.service, del cual se comenta su funcionamiento posteriormente.

7.2.2. Pollen Samples Analysis List

HTML + CSS

Este componente se encarga de mostrar, en forma de tabla, los distintos resultados de los análisis que existen en el sistema asociados a un usuario. En cada elemento de la tabla, aparecen diversos atributos que poseen los análisis, como el nombre descriptivo, la fecha de la muestra o el resultado entre otros, además de un botón para poder eliminar dicho análisis.

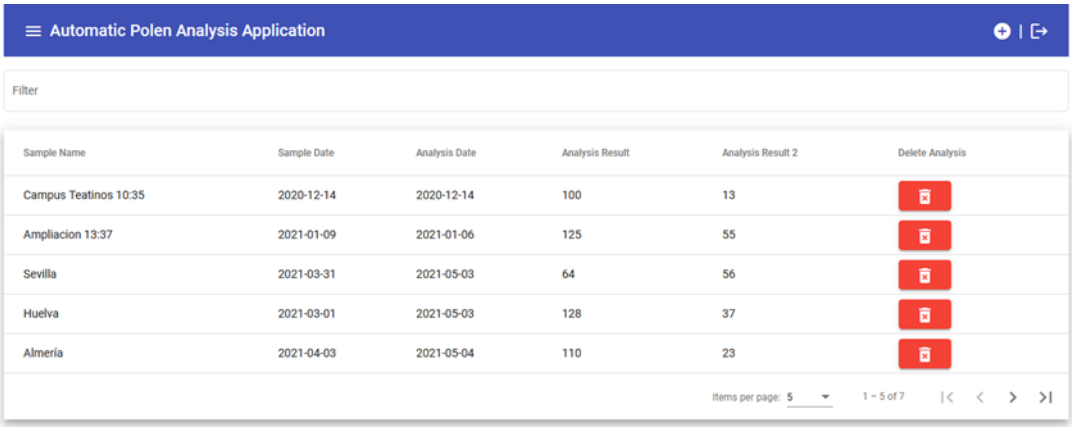

Figura 16: Vista de Pollen Samples Analysis List.

TypeScript

La lógica de esta vista está formada por funciones que permiten obtener la información del módulo back-end acerca de los distintos análisis que el usuario posee. Además, posee también la lógica necesaria para mandar al back-end la petición de que se borre un análisis.

```
ngOnInit() {
  this.refreshPolenList();
}

refreshPolenList() {
  this.service.getPolenList().subscribe(data => {
    this.PolenList = data;
    this.dataSource = new MatTableDataSource(this.PolenList);
    this.dataSource.sort = this.sort;
    this.dataSource.paginator = this.paginator;
  });
}

applyFilter(event: Event) {
  const filterValue = (event.target as HTMLInputElement).value;
  this.dataSource.filter = filterValue.trim().toLowerCase();

  if (this.dataSource.paginator) {
    this.dataSource.paginator.firstPage();
  }
}

deleteDialog(item: any): void {
  if (this._userService.checkIfStillValid()) {
    const deleteDialogRef = this.dialog.open(DeleteDialog, {
      data: { AnalysisId: item.AnalysisId }
    });
    deleteDialogRef.afterClosed().subscribe(_ => {
      this.refreshPolenList();
    });
  }
}
```

Fragmento de código 15: show-polen.component.ts.

7.2.3. Add Pollen Sample

HTML + CSS

Este componente se encarga de gestionar la subida de la imagen de microscopio, así como de la comunicación con el módulo back-end para el análisis de la muestra. En esta vista aparece un formulario por pasos, en el que el usuario depositará toda la información necesaria para comenzar el proceso de análisis.

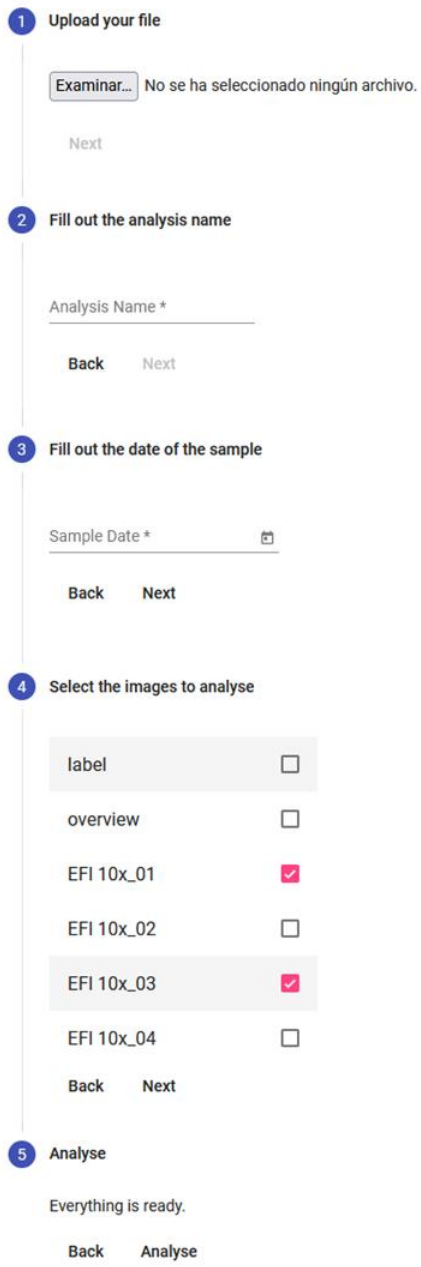

Figura 17: Vista de Add Pollen Sample (Totalmente desplegada).

TypeScript

El fichero TypeScript de este componente está formado por una serie de funciones que permiten obtener los distintos parámetros que está introduciendo el usuario, para después recopilarlos y enviarlos al módulo back-end para que este realice el análisis.

```
addPolen() {
  let today = new Date().toISOString().slice(0, 10)
  let dateAux = new Date(this.dateFormGroup.value.dateCtrl);
  var val = {
    AnalysisId: "",
    UserId: this._userService.getCacheUser_Id(),
    AnalysisName: this.nameFormGroup.value.nameCtrl,
    AnalysisDate: today,
    SampleDate: dateAux.toISOString().slice(0, 10),
    PhotoFileName: this.imagePaths.filename,
    PhotoFilePath: this.imagePaths.unzip,
    SelectedRowsIds: Array.from(this.selectedRowIds)
  };

  this._snackBar.openFromComponent(LoadingAnalysisComponent, {
    horizontalPosition: 'center',
    verticalPosition: 'bottom',
  });


  this.dialogRef.close();

  this.service.analyseSelectedImages(val).subscribe(res => {
    if (res.toString() == "Analysis Okay") {
      window.location.reload();
    } else {
      this._snackBar.dismiss();
      this._snackBar.open('Analysis Failed. Try to analyse later.', 'Close', {
        horizontalPosition: 'end',
        verticalPosition: 'bottom',
        duration: 5 * 1000,
      });
    };
  });

}
```

Fragmento de código 16: add-polen.component.ts.

7.2.4. Pollinic Ranges List

HTML + CSS

Este componente es el encargado de mostrar la lista de rangos polínicos que el usuario tiene registrados en la aplicación. En cada entrada de la tabla, se visualizan las fechas que comprenden dicho rango, así como una pestaña de información en la que, si el usuario accede a ella, podrá observar los distintos tipos de polen que se aplican en el rango polínico, así como su porcentaje de aparición.

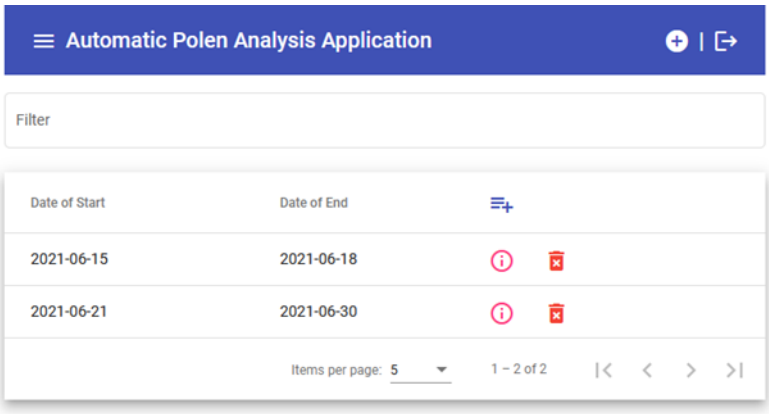

Figura 18: Vista de Pollinic Ranges List.

TypeScript

La lógica de esta vista está formada por funciones que permiten obtener la información del módulo back-end de los rangos polínicos que el usuario posee. Además, posee también la lógica necesaria para mandar al back-end la petición de que se borre un rango polínico.

7.2.5. Add Pollinic Range

HTML + CSS

Este componente se encarga de la creación de rangos polínicos. En esta vista aparece un pequeño formulario, en el que el usuario depositará las fechas que comprenden el rango polínico, así como los porcentajes que lo componen.

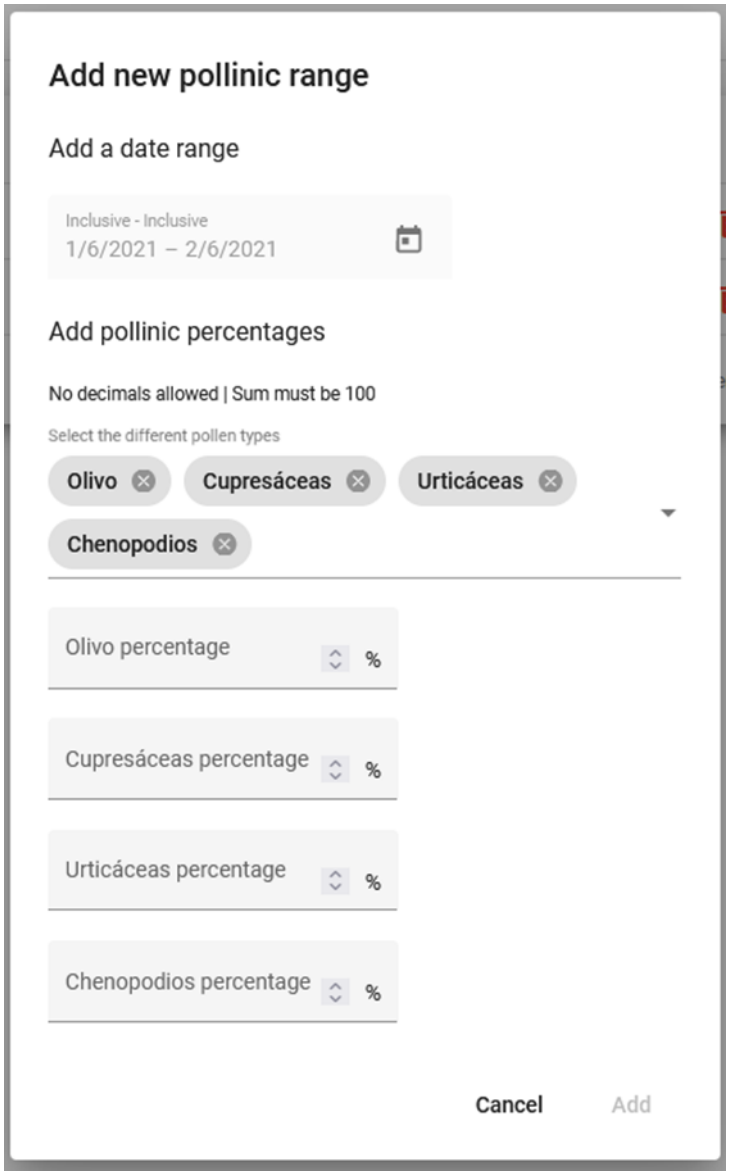

Figura 19: Vista de Add Pollinic Range.

TypeScript

El fichero TypeScript de este componente está formado por una serie de funciones que permiten obtener los distintos parámetros que está introduciendo el usuario en el formulario para posteriormente enviarlos al módulo back-end y se almacenen para su posterior consulta.

```
onPolenRemoved(pollen: string) {
  const pollens = this.polenListControl.value as string[];
  this.removeFirst(pollens, pollen);
  this.polenListControl.setValue(pollens); // To trigger change detection
}

private removeFirst<T>(array: T[], toRemove: T): void {
  const index = array.indexOf(toRemove);
  if (index !== -1) {
    array.splice(index, 1);
  }
}

refreshRangeList() {
  this.service.getPolinicRanges().subscribe(data => {
    this.RangeList = data;
  });
}

myFilter = (d: Date): boolean => {
  let ret: boolean = true;
  this.RangeList.forEach(function (range:any) {
    let startDat: Date = new Date(range.DateOfStart);
    let endDat: Date = new Date(range.DateOfEnd);
    if (d >= startDat && d <= endDat) {
      ret = false;
    }
  });

  return ret;
}
```

Fragmento de código 17: add-range.component.ts.

7.2.6. Pollen Type

HTML + CSS

Este componente es el encargado de mostrar la lista de los tipos de polen que el usuario tiene registrados en la aplicación. En cada entrada de la tabla, se visualizan los datos del tipo de polen. Posee también un botón a través del cual podrá añadirse, accediendo a un formulario, un nuevo tipo de polen.

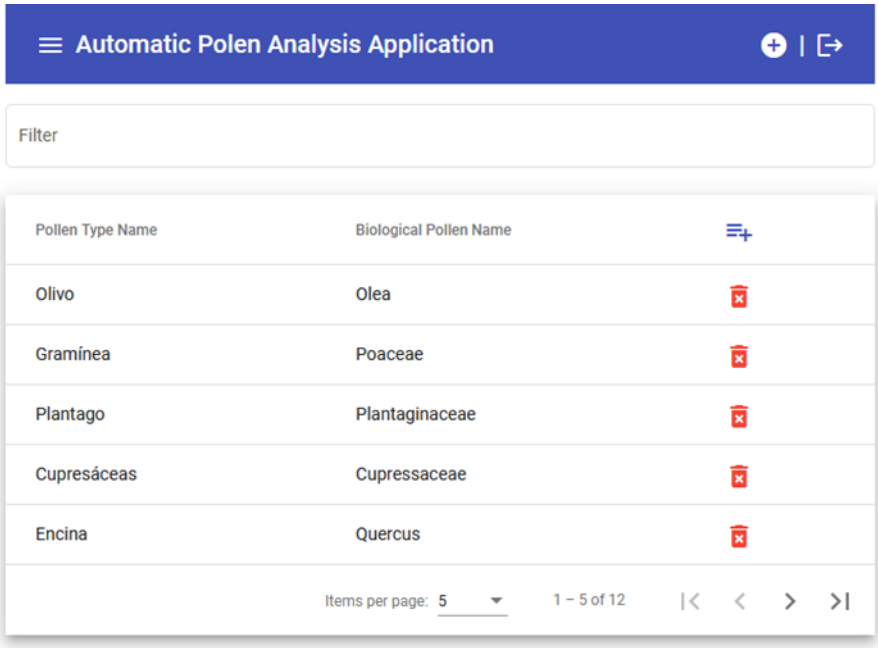

Figura 20: Vista de Pollen Type.

TypeScript

La lógica de esta vista está formada por funciones que permiten obtener la información del módulo back-end de los tipos de polen que el usuario posee. Además, posee también la lógica necesaria para mandar al back-end la petición de que se borre un tipo de polen especifico, así como la lógica para generar un formulario en el que introducir datos para generar un nuevo tipo de polen.

7.2.7. Statistics

HTML + CSS

Este componente se encarga, dados los resultados de los análisis y de los rangos polínicos definidos, de generar estadísticas acerca de los resultados. Actualmente se encuentran dos estadísticas.

- La cantidad de polen media por mes.
- La cantidad aproximada de polen por día aproximada.

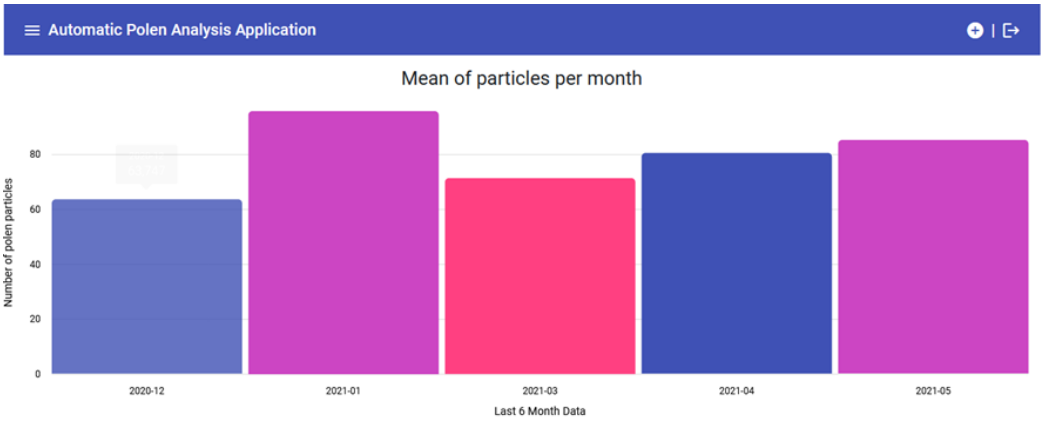

Figura 21: Vista de Statistics (1/2).

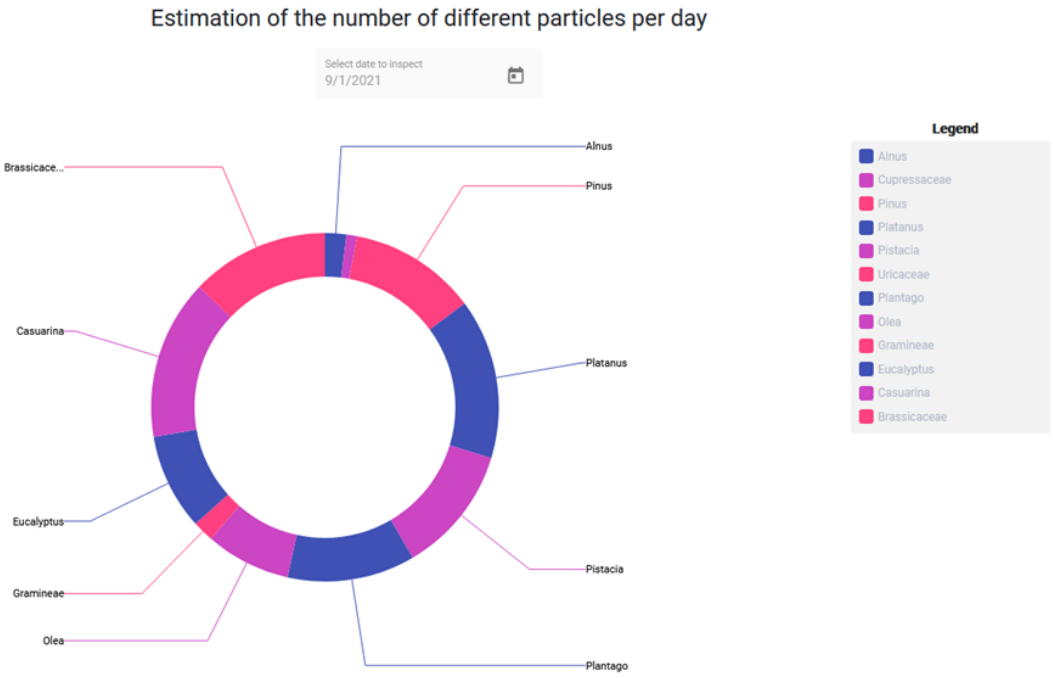

Figura 22: Vista de Statistics (2/2).

TypeScript

El fichero TypeScript de Statistics está compuesto por diversas funciones que piden al módulo back-end la información que necesita para la generación de las estadísticas. El módulo back-end proporciona a estas funciones los datos ya formateados.

7.2.8. Servicios

El módulo front-end también posee varias clases encargadas de llamar a la API REST desarrollada en el módulo back-end. Por tanto, estas clases son las que hacen de conexión entre estos módulos. A estas clases se llaman por nomenclatura servicios. Para realizar las conexiones, se definen unas URLs para exponer las vistas del back-end, y a través de las cuales puede pedir información el módulo front-end.

A continuación, se muestran algunos fragmentos que muestran parte de la implementación de estos servicios.

```
readonly APIUrl = "http://127.0.0.1:8000/";
readonly PhotoUrl = "http://127.0.0.1:8000/media/"

  constructor(private http:HttpClient, private _userService:UserService) { }

    getHttpHeaders() {
      return new HttpHeaders({
          'Content-Type': 'application/json',
          'Authorization': 'JWT ' + this._userService.token
        });
    }

    // ---------------------------------------------

    getPolenList(){
      return this.http.get<any[]>(this.APIUrl + 'polen/', {headers: this.getHttpHeaders()})
    }

    deletePolenAnalysis(val:any){
      return this.http.delete(this.APIUrl + 'polen/'+val, {headers: this.getHttpHeaders()})
    }

    UploadVSIPhoto(val:any){
      return this.http.post(this.APIUrl+'uploadVSI/', val,
      {headers: {'Authorization': 'JWT ' + this._userService.token}})
    }

    analyseSelectedImages(val:any){
      return this.http.post(this.APIUrl+'analyse/',val, {headers: this.getHttpHeaders()})
    }
```

Fragmento de código 18: shared.service.ts.

```
public login(user:any) {
  this.http.post(this.APIUrl+'api-token-auth/', JSON.stringify(user), this.httpOptions).subscribe(
    data => {
      this.updateData(data['token']);
    },
    err => {
      this.errors = err['error'];
      this._snackBar.open('Unable to log in with provided credentials', 'Close', {
        duration: 10000,
        horizontalPosition: "center",
        verticalPosition: "bottom"
      });
    }
  );
}

public logout() {
  this.token = null;
  this.token_expires = null;
  this.username = null;
  localStorage.removeItem("userToken");
}

public updateData(token:any) {
  this.token = token;
  this.errors = [];

  // decode the token to read the username and expiration timestamp
  const token_parts = this.token.split(/\./);
  const token_decoded = JSON.parse(window.atob(token_parts[1]));
  this.token_expires = new Date(token_decoded.exp * 1000);
  this.username = token_decoded.username;

  localStorage.setItem("userToken", token);
}
```

Fragmento de código 19: user.service.ts.

8

Evaluación

Las pruebas al software son una parte esencial en el proceso del desarrollo de software. En esta etapa del desarrollo de software, se tiene como objetivo detectar los errores que se hayan cometido durante la implementación lo antes posible. Existen multitud de pruebas a la que se puede someter a las aplicaciones software, que por ejemplo pueden ir desde la evaluación del rendimiento hasta el recorrido de todos los posibles cauces de la aplicación en busca de algo que esté indeterminado. Tras estas pruebas, se analizan los resultados en busca de posibles errores en la aplicación para su posterior corrección.

Una vez se localizan y se corrigen estos fallos, se vuelven a pasar dichas pruebas o incluso nuevas, en busca de posibles errores. Tras todo esto, como resultado final se obtiene un cierto nivel de confianza en el software que se ha probado.

8.1. Pruebas realizadas

Una de las pruebas que se han realizado a la aplicación miden el tiempo que toma está a la hora de realizar las diferentes funciones que un usuario básico realizaría en la aplicación. Para ello, se medirá el tiempo transcurrido entre el inicio del proceso y el final de este.

El hardware en el que se ha estado ejecutando la aplicación es el siguiente:

- CPU: Intel(R) Core(TM) i5-8250U
- Memoria RAM: 8.0 GB a 2400 MHz
- GPU: NVIDIA GeForce GTX 1050M

Para cada una de las pruebas realizadas, se han realizado cuatro intentos, para obtener un tiempo medio y evitar así posibles anomalías en el resultado.

Prueba	**Tiempo**
Extracción de datos de las muestras	55,26 s
Análisis de las muestras	3,35 s
Generación de estadísticas	2 ms
Envío de formulario básico al back-end	122,75 ms

Como se puede observar en los resultados de las pruebas, lo que más tiempo necesita es la extracción de los datos previa al análisis. Esto puede deberse a varios motivos. Uno de ellos es que al usar la librería BioFormats y esta estar desarrollada en Java, las llamadas a esta librería podrían ralentizar el proceso. Otro de los motivos puede ser el tamaño de las imágenes. La muestra que se ha usado para las pruebas tiene un tamaño de 409 MB y las imágenes una vez extraídas tienen un tamaño de 192 MB.

Con respecto al resto de resultados, se puede observar que, pese a que el equipo en el que se han realizado las pruebas es un equipo modesto, el tiempo medio evaluado en las distintas pruebas es pequeño. Por tanto, cuando esta aplicación pase a producción en un servidor con mejores características, estos resultados mejorarán.

Otra prueba que se ha realizado ha sido una prueba de precisión del algoritmo de análisis. En ella, y dado a las pocas imágenes que se han proporcionado, se ha decidido hacer la prueba a las distintas series de una imagen. Para ello, se mostrarán las salidas del algoritmo con respecto al resultado real. Finalmente, también se mostrará una comparación de la suma de las series.

	Salida del algoritmo	**Resultado real**
Serie 1	27	24
Serie 2	55	51
Serie 3	46	41
Serie 4	37	35
Total	165	151

Como se puede observar tras las pruebas, el resultado obtenido con el algoritmo es bastante cercano al resultado real obtenido por el grupo de aerobiología de la Universidad de Málaga. Hay que destacar que desde el grupo de aerobiología se informó de que a la imagen proporcionada no se le habían aplicado exactamente los mismos barridos, por lo que la salida podría variar.

9

Conclusiones y líneas futuras

El resultado de este Trabajo de Fin de Grado ha sido el desarrollo de una aplicación web que permite a equipos de aerobiología analizar de manera autónoma muestras de polen que suban tras el barrido del microscopio. La aplicación está formada por diversos módulos los cuales permiten la extracción y el análisis de las imágenes, para luego, si fuese necesario, obtener también resultados estadísticos.

Como líneas futuras se plantean varias posibles. La primera de ellas es sustituir el actual módulo de análisis de muestras. Esto se debe a que actualmente es un módulo basado en el análisis de imágenes sin inteligencia artificial, y esto podría llevar a resultados de baja precisión. Por ello, y a la espera de que el equipo de aerobiología de la Universidad de Málaga proporcione más imágenes, se propone la sustitución del actual módulo por un módulo donde haya un modelo basado en inteligencia artificial entrenado, capaz de analizar, con mayor precisión, la cantidad de polen que hay en una muestra. Siguiendo por esa línea, también se podría actualizar dicho módulo, para que, en vez de tan solo contar las partículas, sea capaz de clasificar algunos tipos de polen, haciendo así las estadísticas más reales y no orientativas.

Apéndice A

Manual de usuario

A continuación, se mostrará el funcionamiento de la aplicación web para el análisis de muestras de polen, que como su propio nombre indica, permite la subida de muestras de polen escaneadas con un microscopio de alta precisión y su análisis automatizado en la aplicación.

A.1. Instalación de la aplicación web

Para la instalación de la aplicación web, será necesario tener instalado en el equipo Python 3.8, Node.js y Git.

El primer paso que dar es clonar el repositorio donde está alojada la aplicación. Este repositorio está en GitHub y forma parte del workspace común del grupo de investigación ERTIS. *https://github.com/ertis-research/pollen-analysis-app* .

Tras clonar la aplicación, se instalarán en la máquina tanto las dependencias de Python como los paquetes de Node.js necesarios. Para ello se proporciona un fichero requirements.txt para instalar las dependencias de Python, así como un fichero package.json para instalar los paquetes necesarios para el módulo front-end. En caso de duda, los pasos para la instalación se indican en el repositorio.

Una vez se tiene todo instalado, se procederá a migrar los modelos de Django a una nueva base de datos. Para ello, se usan los siguientes comandos.

- *python manage.py makemigrations*
- *python manage.py migrate*

Con esto, se genera una base de datos limpia para la aplicación. Como está totalmente limpia, no existen usuarios. Para crear un usuario de administrador que posteriormente permita crear más usuarios, se usa el siguiente comando.

- *python manage.py createsuperuser*

En ese momento la consola pedirá algunos datos para crear el primer usuario, que tendrá todos los permisos necesarios para crear nuevos usuarios.

Una vez se tiene todo esto listo, la aplicación web podrá arrancarse. Para ello, se accederá a las carpetas DjangoAPI y frontend (si no se estaba ya anteriormente) y se ejecutarán respectivamente los siguientes comandos en distintas consolas:

- *python manage.py runserver*
- *ng serve*

Cuando ambas consolas indiquen que ambos servidores están listos, se podrá acceder a la dirección *http://localhost:4200/* para comenzar a usar la aplicación.

Cuando se inicia la aplicación, aparecerá la ventana de inicio de sesión. En la instalación se ha creado un usuario administrador. Este podrá acceder a la aplicación, pero, además, también tendrá acceso al panel de administrador, donde podrá crear los usuarios pertinentes. Para ello, deberá de acceder a la dirección *http://localhost:8000/admin*.
En esta dirección podrá crear, modificar y eliminar los usuarios de la plataforma.

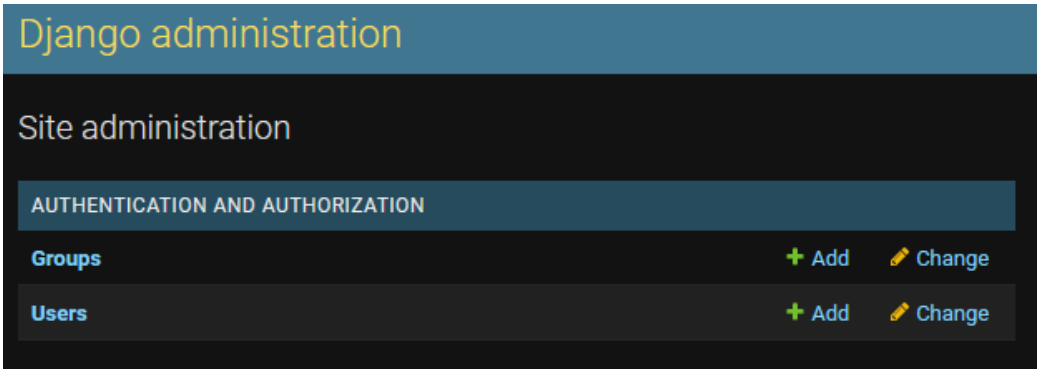

Figura 23: Web de administración de Django.

Una vez se inicia sesión dentro de la aplicación, ya se tiene acceso a la interfaz de la aplicación. En el menú de la barra principal se encuentran las diversas opciones dentro de la aplicación. Además, se encuentra el botón para subir nuevas muestras, así como el botón para desconectarse de la aplicación.

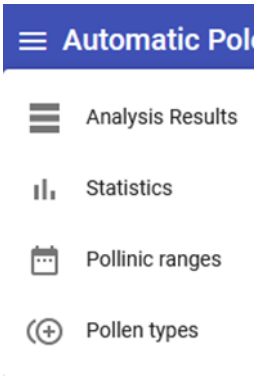

Figura 24: Menú con diversas opciones.

A.2. Subida de imágenes a la aplicación

Una vez dentro de la aplicación, si se pulsa el botón de subida, se abrirá un diálogo con un formulario por pasos para la subida de imágenes a la aplicación. En él, primeramente, se pide que se suba la imagen. Actualmente, la aplicación solo permite la subida de imágenes de formato VSI (Virtual Slide Image) comprimida en un ZIP.

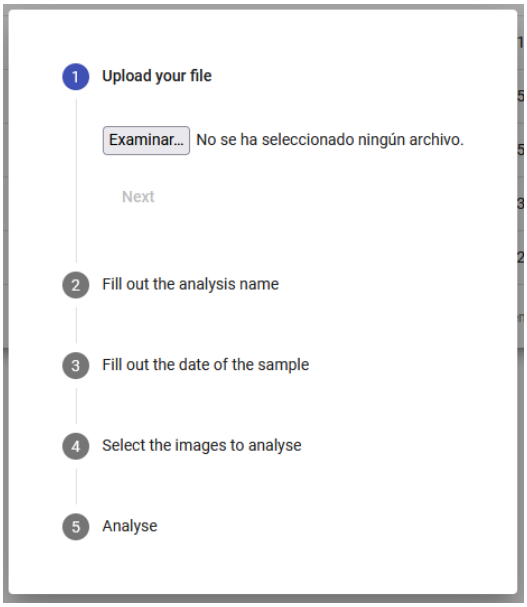

Figura 25: Formulario por pasos para la subida de muestras.

En los siguientes pasos, se piden algunos atributos para el análisis, que son meramente descriptivos y que servirán para la creación de estadísticas. Tras estos atributos, la aplicación preguntará que series de la imagen se desean analizar, ya que estas imágenes poseen distintas imágenes (series) dentro de ellas. Una vez seleccionadas las imágenes, se podrá mandar la petición a la aplicación para el análisis.

Tras el análisis de la imagen, los datos serán almacenados en la plataforma. Estos datos serán accesibles desde la pestaña *Analysis Results*.

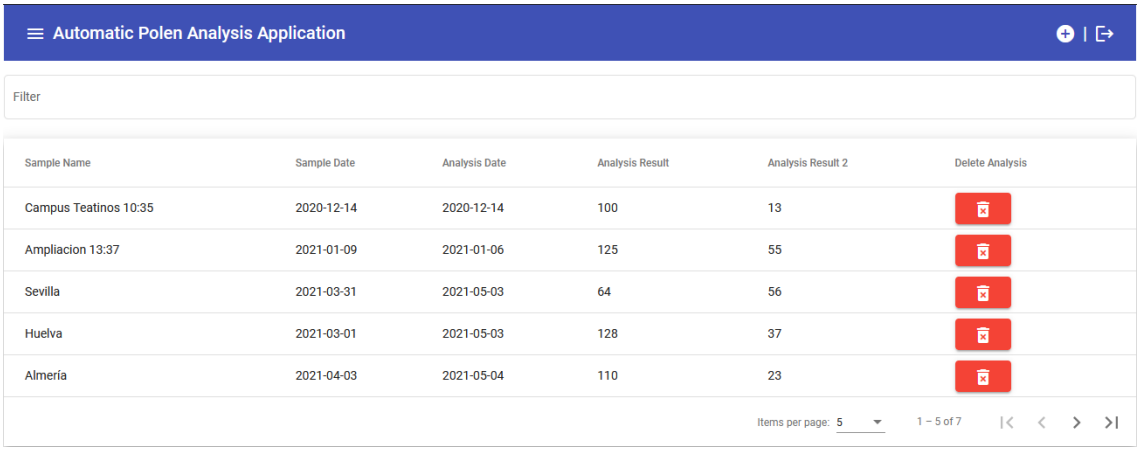

Figura 26: Vista de *Analysis Results*

A.3. Creación de nuevos tipos de polen

Para crear rangos polínicos, es necesario añadir a la aplicación tipos de polen. Estos tipos de polen representan tipos de polen de la realidad. Es necesario introducir los datos del polen.

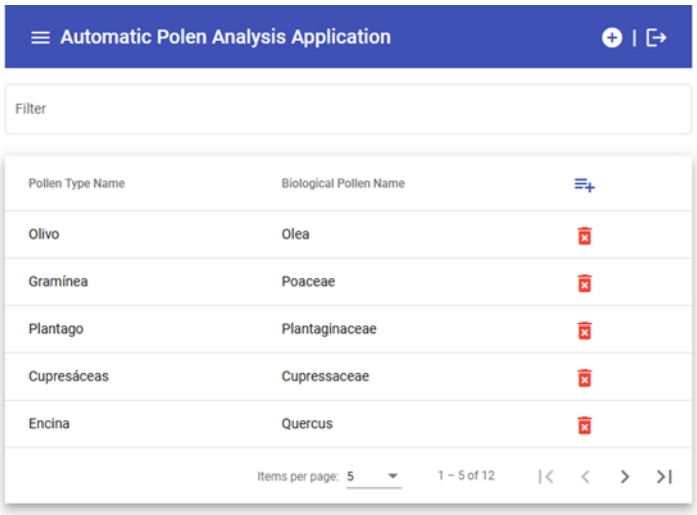

Figura 27: Vista de *Pollen Types.*

Para crear un nuevo tipo de polen, se debe ir a la pestaña *Pollen Types*. En esta pestaña, se encuentra un botón que al ser pulsado abrirá un modal con un formulario para introducir los distintos datos. Una vez enviados los parámetros, la aplicación almacenará el nuevo tipo de polen.

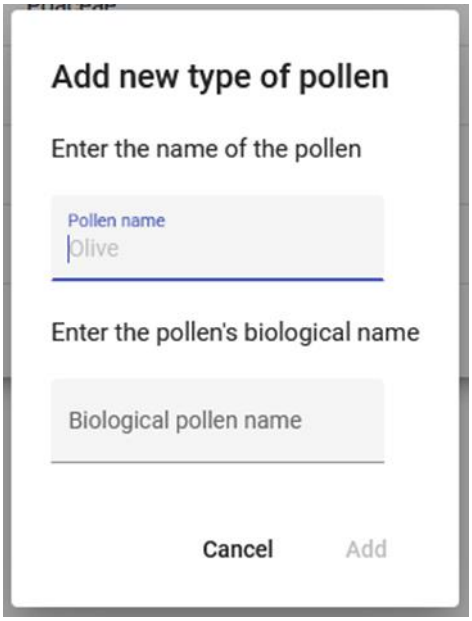

Figura 28: Formulario para añadir un nuevo tipo de polen.

A.4. Creación de rangos polínicos

Para algunas estadísticas, es necesario lo que se ha llamado como rangos polínicos. Estos rangos polínicos se conforman de dos fechas, que representan la fecha de inicio y final del rango polínico, así como el porcentaje aproximado de aparición de los distintos tipos de polen más comunes.

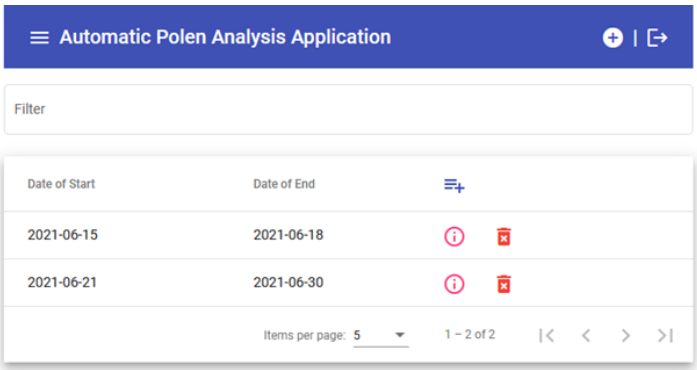

Figura 29: Vista de *Pollinic Ranges.*

Para crear un nuevo rango polínico, se debe ir a la pestaña *Pollinic Ranges*. En esta pestaña, se encuentra un botón que al ser pulsado abrirá un modal con un formulario para introducir los distintos datos. Una vez enviados los parámetros, la aplicación almacenará el nuevo rango polínico. Todos los rangos polínicos pueden ser consultados en la pestaña *Pollinic Ranges.*

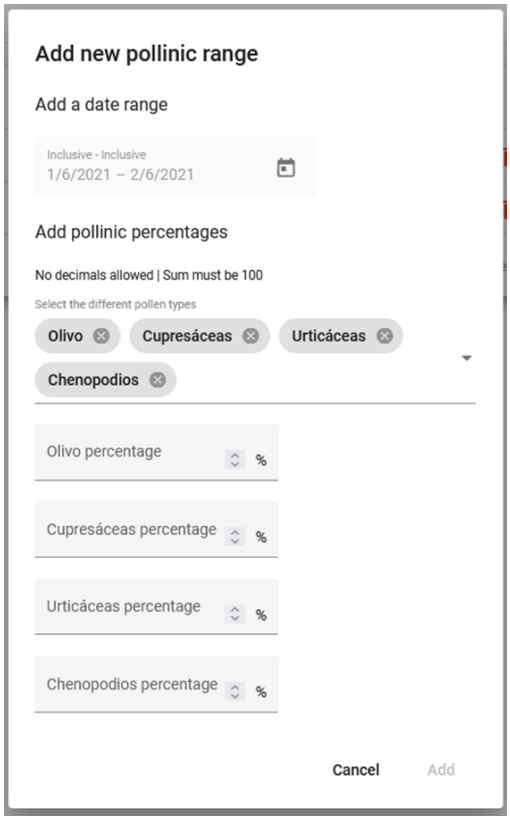

Figura 30: Formulario para añadir un nuevo rango polínico.

A.5. Visualización de estadísticas

Para visualizar las estadísticas, se debe ir a la pestaña Statistics. En esta pestaña, se encuentran los dos tipos de gráficas que actualmente existen en la aplicación.

En primer lugar, se encuentra la cantidad de polen media de los últimos 6 meses, ideal para observar el crecimiento o el descenso de la cantidad de polen en los distintos durante los distintos meses.

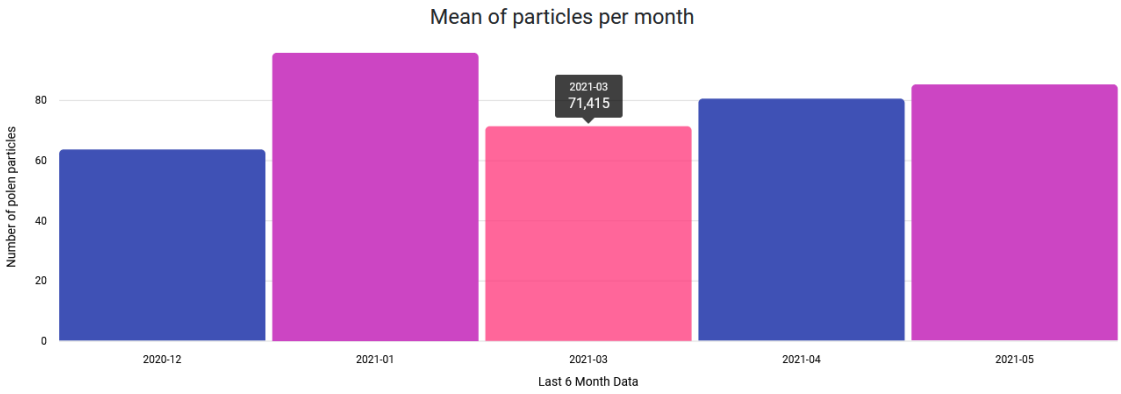

Figura 31: Estadística de cantidad media de polen en los últimos 6 meses.

En segundo lugar, se encuentra la cantidad aproximada de los distintos tipos de granos de polen por cada día. Para esto, se hace uso de los rangos polínicos explicados en el apartado anterior. Dada una fecha en la que haya habido una muestra y dado un rango polínico en el que esta fecha este incluida, se podrá obtener una aproximación de la cantidad de polen que hay de cada especie.

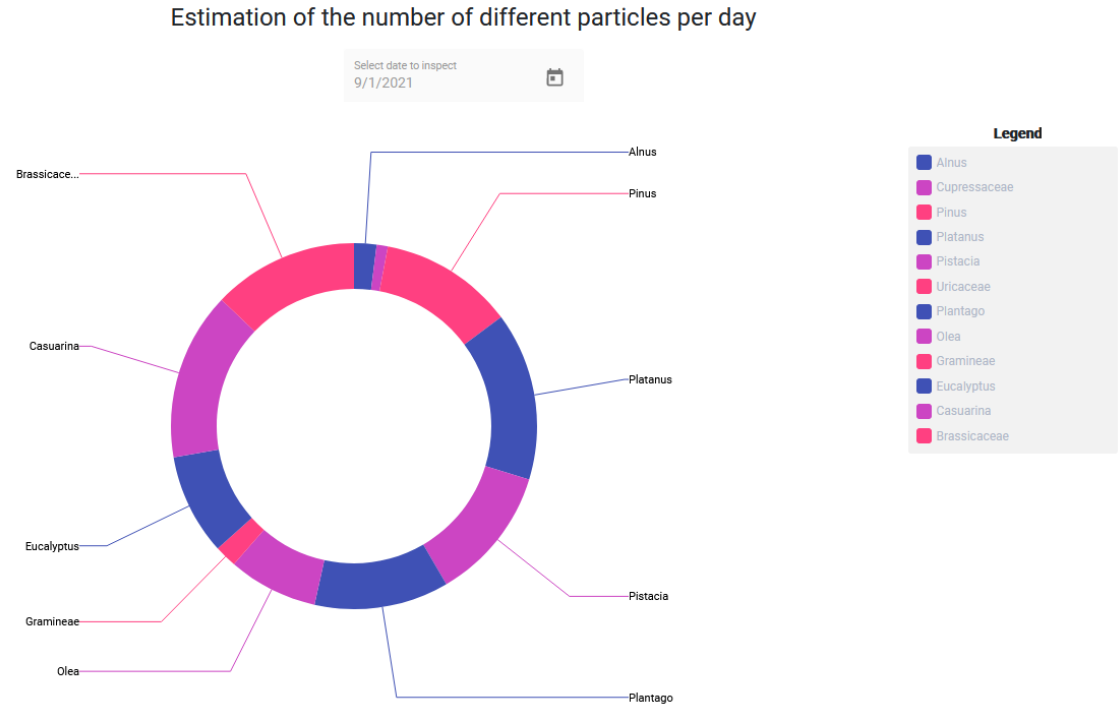

Figura 32: Estimación de los distintos tipos de polen por día.

Referencias

[1] <<Who we are - LifeWatch-ERIC>>, LifeWatch ERIC.
Accedido: may. 14, 2021. [En línea]. Disponible en:
https://www.lifewatch.eu/web/guest/who-we-are.

[2] <<Visión por computador: qué es y cuáles son sus usos más comunes>>,
INFAIMON. ene. 18, 2018
Accedido: may. 14, 2021. [En línea]. Disponible en:
https://blog.infaimon.com/vision-computador-soluciones-permite/.

[3] <<Metodologías de desarrollo software>>, Blog Becas Santander. dic. 21, 2020,
Accedido: may. 14, 2021. [En línea]. Disponible en:
https://blog.becas-santander.com/es/metodologias-desarrollo-software.html.

[4] <<Metodología Agile y sus beneficios>>, We Are Marketing. feb. 06, 2020,
Accedido: may. 14, 2021. [En línea]. Disponible en:
https://www.wearemarketing.com/es/blog/que-es-la-metodologia-agile-y-que-beneficios-tiene-para-tu-empresa.html.

[5] <<Python>>, Wikipedia, la enciclopedia libre. may. 11, 2021
Accedido: may. 14, 2021. [En línea]. Disponible en:
https://es.wikipedia.org/w/index.php?title=Python&oldid=135459242.

[6] <<Python: ventajas y desventajas>>, Discoder. ene. 23, 2021
Accedido: may. 14, 2021. [En línea]. Disponible en:
https://www.discoder.tech/python-ventajas-desventajas/.

[7] <<About OpenCV>> OpenCV,
Accedido: may. 14, 2021. [En línea]. Disponible en:
https://opencv.org/about/.

[8] <<Introducción a Django>> MDN Web Docs (Mozilla). may. 27, 2021
Accedido: may. 28, 2021. [En línea]. Disponible en:
https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introduction.

[9] <<¿Qué es Angular y para qué sirve?>> QualityDevs. sep. 16, 2019
Accedido: may. 14, 2021. [En línea]. Disponible en:
https://www.qualitydevs.com/2019/09/16/que-es-angular-y-para-que-sirve/.

[10] <<TypeScript: Typed JavaScript at Any Scale>> TypeScript mar. 9, 2021
Accedido: may. 14, 2021. [En línea]. Disponible en:
https://www.typescriptlang.org/.

[11] <<Requisito funcional>> Wikipedia, la enciclopedia libre. may. 12, 2021
Accedido: may. 17, 2021. [En línea]. Disponible en:
https://es.wikipedia.org/w/index.php?title=Requisito_funcional&oldid=135475696.

[12] <<Requisitos no funcionales>> EcuRed. jul. 17, 2013
Accedido: may. 17, 2021. [En línea]. Disponible en:
https://www.ecured.cu/index.php?title=Requisitos_no_funcionales&oldid=1995934.

[13] <<Django or Django Rest Framework>> Stack Overflow. mar, 18, 2018
Accedido: may. 19, 2021. [En línea]. Disponible en:
https://stackoverflow.com/questions/49109791/django-or-django-rest-framework.

[14] <<User model - django.contrib.auth>> Django Docs
Accedido: may. 19, 2021. [En línea]. Disponible en:
https://docs.djangoproject.com/en/3.2/ref/contrib/auth/.

[15] <<JSON Web Token Introduction>> jwt.io.
Accedido: may. 24, 2021. [En línea]. Disponible en:
https://jwt.io/introduction.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA