Aplicación nativa para el diagnóstico en la nube

Sergio Espina Vico

Resumen—Este proyecto consiste en el desarrollo de una aplicación en la nube que pretende funcionar como soporte al diagnóstico de tumores pulmonares. Para hacer esto se ha desarrollado dos módulos distintos, el primero que permite a un doctor explorar las imágenes resultantes de una tomografía computarizada e identificar dónde se encuentran los distintos tumores. El segundo indica el camino que a través de los bronquios que debe seguir el médico durante una broncoscopia para llegar a la localización de estos tumores.

Palabras clave—Cáncer de pulmón, broncoscopia, diagnóstico como servicio, aplicación basada en la nube, visión por computador.

Abstract—This project consists in the development of a cloud-based application that intents to be used as a diagnosis support of lung tumours. To get these objectives, we have developed two distinct modules, a first module that allows a doctor to explore the result images of a computerized tomography and identify where those tumours are located. The second module that indicates the path that must be followed through the bronchi by a doctor when doing a bronchoscopy to get to the location of the tumours.

Index Terms— Lung cancer, bronchoscopy, diagnosis as a service, cloud-based application, computer vision.

1 Introducción

El cáncer de pulmón es uno de los cánceres más diagnosticados actualmente, representando aproximadamente el 10,68% de los casos totales diagnosticados de cáncer en 2020 en España [1]. Para el diagnóstico de esta enfermedad suele ser necesaria la toma de una muestra del tejido pulmonar potencialmente canceroso para la realización de una biopsia. Uno de los métodos utilizados para la realización de esta toma de muestras es la broncoscopia [2]. Para realizar este procedimiento se utiliza un broncoscopio, un instrumento en forma de tubo que es introducido en los pulmones del paciente a través de las fosas nasales o la boca. Este instrumento permite al doctor observar y navegar el árbol bronquial a través de un dispositivo óptico y realizar una extracción de tejido de la zona que se requiera estudiar. Este en un procedimiento muy complejo debido en parte a la dificultad de navegar por el árbol bronquial cosa que hace que en ocasiones no se pueda realizar completamente con el éxito deseado.

Por ello, el equipo de investigación IAM del Centre de Visió per Computador, equipo con el que he trabajado para la realización de este proyecto, está desarrollando el sistema BronchoX (Bronchoscopy Exploration) [3]. Este sistema consta de dos módulos principales, uno de planificación y otro de guiado durante la realización de broncoscopias. El módulo de planificación que permite, a través de información proporcionada por Tomografías Computarizadas, la construcción de un modelo 3D del árbol bronquial de un paciente y calcular la mejor ruta para llegar a la zona que se quiera estudiar. El segundo módulo de intervención utiliza la información generada en la planificación, da instrucciones al médico de cómo debe desplazar el broncoscopio por los bronquios del paciente a partir de un seguido de imágenes de cada una de las bifurcaciones bronquiales que se encontrará durante la intervención.
Para facilitar el uso generalizado de este sistema en centros clínicos susceptibles de tener recursos computacionales limitados, BronchoX se ha diseñado como un sistema en la nube tipo SaaS.
Para conseguir realizar esta aplicación se ha propuesto un diseño basado en pequeños servicios, en el que cada uno de estos servicios se encarga de una funcionalidad específica del sistema. Estos servicios se explicarán a continuación, en la figura 1 se puede ver un esquema general del sistema.

- **Servicio Dispatcher**: Servicio controlador de la aplicación, gestionará el flujo de datos entre los diferentes servicios y se encargará de iniciar los diferentes trabajos que realizarán el resto de los servicios.

- **Servicio de control de usuarios:** La meta de este

- *E-mail de contacto: Sergio.Espina@e-campus.uab.cat*
- *Mención realizada: Ingeniería del Software*
- *Trabajo tutorizado por: Carles Sanchez Ramos y Debora Gil Resina*
- *Curso 2020/21*

servicio es el control de acceso al sistema a partir de una combinación de usuario y contraseña.

• **Servicio de procesado de TCs:** Su objetivo es el de recibir un archivo en formato DICOM y procesar la información correspondiente a Tomografías Computarizadas de pulmones para poder organizarla en un formato en el que pueda ser tanto almacenada en el sistema como ser procesada por los servicios de cálculo.

• **Servicio de procesado de datos:** Procesa los datos de tomografías proporcionadas y seleccionadas por el usuario de forma asíncrona.

• **Servicio de datos/almacenamiento:** Este servicio gestiona el almacenaje de datos, tanto antes de ser procesados como después de aplicar los diferentes procesos.

• **Servicio/módulo de renderizado:** El propósito de este módulo es el renderizado en 3D de la información generada por el servicio de procesado para que el usuario pueda tanto planificar como realizar las intervenciones.

• **Servicio WEB:** El objetivo de este servicio será el de servir como interfaz con una aplicación cliente utilizando un servidor web.

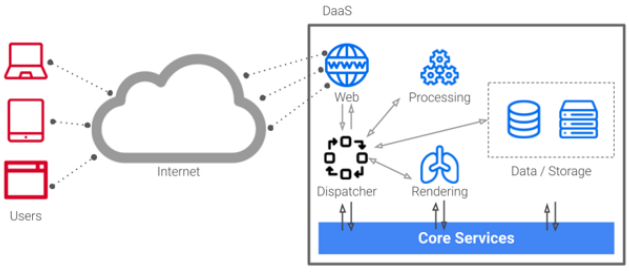

Figura 1. Diagrama de los diferentes servicios de la aplicación

2 Objetivos

El objetivo principal de este proyecto es el diseño, implementación, testeo y despliegue de una aplicación basada en la nube tal y como se ha explicado en la introducción. A partir de los servicios que se han definido se explicará el objetivo de desarrollo dentro de cada uno de ellos durante el desarrollo de este proyecto.

• **Obj-1, Diseño e implementación del servicio dispatcher:** El objetivo es el de diseñar e implementar este servicio dispatcher. Se deberá implementar toda la funcionalidad que permita el control y la comunicación del resto de servicios del sistema.

• **Obj-2, Diseño e implementación del servicio de control de usuarios:** El objetivo es el de diseñar e implementar este servicio de autenticación de usuarios. Este control de acceso se realizará a través de tokens de autenticación.

• **Obj-3, Diseño e implementación del servicio de procesado de TCs:** El objetivo es el de diseñar e implementar este servicio de tal forma que reciba un archivo comprimido por parte del usuario y sea capaz de descomprimirlo y almacenarlo tanto en el sistema de archivos del sistema como en una base de datos.

• **Obj-4, Diseño e implementación del servicio de procesado de datos:** El objetivo es el de diseñar e implementar la base a partir de la cual el equipo de investigación podrá aplicar los diferentes tipos de tareas de procesado.

• **Obj-5, Diseño e implementación del servicio de datos/almacenamiento:** El objetivo es el de diseñar e implementar este servicio para que pueda soportar diferentes sistemas de bases de datos e incluso pueda poder adaptarse en fácilmente en un futuro para que pueda utilizar sistemas de bases de datos ya existentes.

• **Obj-6, Diseño e implementación del Servicio WEB:** El objetivo es el de diseñar e implementar un servidor web con el que una aplicación cliente pueda interaccionar. Este servidor debe responder a peticiones basadas en una API Rest.

• **Obj-7, Implementación de un prototipo de cliente:** El objetivo es la implementación de un cliente que interactúa con nuestra aplicación. Para este proyecto se propone la creación de un prototipo de interfaz que podría servir como demostración del funcionamiento completo del sistema.

3 Estado del arte

En este apartado exploraremos sistemas actuales de soporte al diagnóstico médico o de exploración de imágenes en la nube. A continuación, se realizará un análisis de los sistemas explorados comparándolos con el sistema que se ha desarrollado en la sección 3.2.

3.1 Exploración de sistemas actuales

3.1.1 PostDICOM

PostDICOM [7] es un servicio que ofrece la posibilidad de almacenar, gestionar y consultar resultados de exploraciones médicas basadas en imágenes. Es un servicio basado en el cloud que permite la integración con el sistema informático principal de un hospital y ofrece la posibilidad del almacenaje remoto o redundante del resultado de este tipo de exploraciones. Para el visualizado de imágenes utiliza una aplicación web y ofrece una API a partir de la cual se puede desarrollar aplicaciones propias que interactúen con la información almacenada en la red.

3.1.2 RapidAI

RapidAI [8] ofrece sistemas de soporte al diagnóstico de enfermedades cerebrovasculares utilizando inteligencia artificial. Esta empresa ofrece diferentes servicios dependiendo del tipo de exploración de imágenes que se le ha realizado al paciente, por ejemplo, Rapid MRI que analiza el resultado de resonancias magnéticas del cerebro y ofrece al personal médico de forma rápida el análisis de las imágenes que puede ser utilizado para tratar situaciones críticas cómo puede un infarto cerebral.

3.1.3 Arterys

Arterys [9] es una empresa que ofrece diferentes servicios de análisis de imágenes médicas basadas en inteligencia artificial. Ofrece cuatro servicios de diagnóstico distintos: CardioAI que analiza resonancias magnéticas del corazón del paciente, LungAI que analiza tomografías computarizadas de pulmones y, entre otros análisis, puede detectar automáticamente la presencia de nódulos en los pulmones, Chest|MSK AI que analiza resultados de radiografías y NeuroAI que analiza resonancias magnéticas del cerebro y permite entre otras cosas detectar la presencia de tumores cerebrales.

3.1.4 Ambra

Ambra [10] es una empresa dedicada a ofrecer servicios de cloud para el almacenaje, gestión, compartición y visualización de exploraciones por imágenes médicas. Soporta todas las modalidades de imágenes soportadas por el estándar DICOM y permite a los hospitales almacenar estos archivos y consultarlos de forma remota. Este servicio permite también adjuntar cualquier otro tipo de archivos necesarios que junto a DICOM permiten al personal médico realizar sus tareas de exploración y diagnóstico.

3.2 Análisis del estado del arte

Después de analizar varias opciones de aplicaciones dentro del mismo ámbito a la que estamos desarrollando, hemos podido observar que se suele ofrecer dos servicios distintos. El primero es un servicio de alojamiento de imágenes médicas que permite a los usuarios de estos servicios gestionarlas y explorarlas y el segundo tipo de servicios es el análisis de estas imágenes a través de inteligencias artificiales para dar soporte de diagnóstico al equipo médico.
Creo que, todo y que nuestra aplicación presenta ambos servicios, aunque de una forma mucho más especializada, nuestra aplicación aporta nuevas funcionalidades que ninguna de estas aplicaciones ofrecen en este momento. Estas funcionalidades son principalmente el soporte a la intervención médica una vez se ha realizado el diagnóstico y el renderizado de imágenes tridimensionales. El hecho de que nuestra aplicación permitirá preparar e incluso servirá de guía a la hora de realizar intervenciones de broncoscopias hace que esta aplicación tenga un nicho de usos en este tipo de servicios.

4 Metodología

Para el desarrollo de este proyecto se ha decidido seguir una metodología ágil [4]. Este tipo de metodologías permiten que durante el desarrollo del proyecto se evalúe el estado actual del mismo y se realicen cambios en la planificación según las necesidades de ese momento. Esto es conveniente en este caso ya que el desarrollo del sistema se deberá adaptar a las necesidades de los módulos y servicios en desarrollo por parte de los miembros del equipo de investigación. Se propone la utilización del método Kanban [5], este método se basa en la distribución de tareas en un tablón dividido en columnas dependiendo del estado de desarrollo de la tarea en cada momento.
Se utilizará la aplicación en línea Trello [6] para llevar el control del Kanban board.
Cada una de estas columnas representará una de las fases de la vida del ciclo del software, estas son las siguientes:

- **Backlog**: En esta columna se colocarán las tareas una vez han sido definidas y se ha hecho un análisis de sus requerimientos.

- **Todo**: En esta columna se colocarán las tareas que se decida son necesarias a realizar en ese momento.

- **Development**: En esta columna se colocarán las tareas que están activamente en desarrollo.

- **Test**: En esta columna se colocarán las tareas que están pendientes de ser testeadas. Estas pruebas serán tanto test unitarios como test de integración con el resto del sistema.

- **Completado**: En esta columna se colocarán las tareas que han pasado las pruebas y pueden ser puestas en producción.

5 Planificación

Tal y como se ha explicado en la descripción de la metodología durante el desarrollo del proyecto no se va a seguir un ritmo de trabajo iterativo en el que se definen nuevos objetivos que cumplir en un plazo de tiempo determinado, sino que se va a realizar una evaluación continua de las tareas a realizar a partir de las necesidades y el feedback del equipo de investigación.

5.1 Planificación inicial

Se ha decidido organizar el proyecto en diferentes fases y a cada una de ellas se les ha asignado unos objetivos principales y unos tiempos aproximados. Estas fases son las siguientes con su planificación temporal inicial:

1. Fase inicial: Los objetivos principales de esta fase son la definición de objetivos, elección de metodología de

desarrollo y redacción del informe inicial. Realización entre la semana 1 y la semana 4.

2. Fase de recogida de requisitos: Recogida de requisitos funcionales y no funcionales de cada uno de los diferentes servicios. Se espera un trabajo de dos semanas para esta fase por lo que el tiempo aproximado es entre la semana 5 y la semana 6.

3. Fase de diseño: Diseño de la arquitectura general, diseño específico de cada uno de los servicios, diseño de las APIs de comunicación entre servicios. Se espera también un trabajo de dos semanas para definir los diferentes diseños a partir de los requisitos, por lo tanto, el tiempo aproximado es entre la semana 7 y la 8.

4. Fase de desarrollo y testeo: En esta fase se espera realizar la implementación de todo lo diseñado en la fase anterior. A la vez se realizarán pruebas unitarias de todo el código generado y pruebas de integración con el sistema. Esta es la fase del proyecto con más carga de trabajo, se espera iniciar poco después que la fase de diseño y se espera un tiempo aproximado entre el inicio de la semana 8 y el final de la semana 15.

5. Fase de despliegue: En esta fase se espera realizar la configuración de la aplicación como servicio en la nube y la realización del correcto despliegue en un entorno de producción simulado. El tiempo aproximado será entre la semana 16 y la 17.

6. Fase de documentación: Esta fase se realizará durante toda la duración del proyecto después de la fase inicial. Consistirá en la generación de toda la documentación asociada a las tareas que se vayan realizando en cada fase como a la generación de los documentos de seguimiento, informe final, poster y presentación. Por lo tanto, se espera empezar a partir de la semana 5 y acabar la semana 20 con la entrega del dosier y el poster.

5.2 Modificaciones a la planificación inicial

La distribución de trabajo final ha sufrido varias modificaciones a lo largo del desarrollo del proyecto. Estas modificaciones han sido principalmente dos, la primera es que tanto la fase de recogida de requisitos como la fase de diseño llevaron ambas más tiempo del previsto inicialmente, por lo que se le asignó a cada una de estas fases una semana más. La segunda es que debido a los cambios en las dos fases anteriores se retrasó el inicio de la fase de desarrollo. Esto hizo que se decidiera atrasar el final de esta fase y se hizo coincidir con la fase de despliegue. Estos cambios no han supuesto problema ya que se ha adaptado el ritmo de trabajo a esta nueva planificación de forma satisfactoria y por cómo se ha desarrollado la aplicación en la nube primero y luego el prototipo no ha habido problema para realizar las pruebas de despliegue de la aplicación de backend. En el apéndice 1 se puede observar un gráfico tanto de la planificación inicial como de la planificación final una vez se han aplicado las modificaciones.

6 Desarrollo

6.1 Análisis de requisitos

Tal y como se ha explicado en la planificación una de las primeras tareas realizadas en el proyecto ha sido la recogida y definición de requisitos. Para obtener estos requisitos se realizó un seguido de entrevistas a diferentes personas involucradas en el proyecto. Estas personas fueron tanto integrantes del equipo de investigación que habían propuesto la base del proyecto, como una doctora del hospital Germans Tries i Pujol. De este análisis se obtuvo tanto una lista de requisitos funcionales de la aplicación como de requisitos no funcionales. Estos requisitos son los que se indican en los siguientes apartados.

6.1.1 Requisitos funcionales

Requisitos funcionales son aquellos que definen la funcionalidad del sistema. Después del análisis de requisitos de han definido los siguientes:

- RF-01: El sistema debe ser capaz de realizar una autenticación de usuarios basada en usuario y contraseña.
- RF-02: El sistema debe ser capaz de gestionar la subida y tratamiento de imágenes médicas en formato DICOM.
- RF-03: El usuario debe ser capaz de visualizar una tomografía computarizada subida anteriormente por él en formato DICOM.
- RF-04: El usuario debe ser capaz de seleccionar nódulos una vez está visualizando una tomografía.
- RF-05: El sistema debe ser capaz de procesar la tomografía de forma asíncrona a partir de la información generada por el usuario.
- RF-06: El sistema debe ser capaz de almacenar y consultar la información generada en una base de datos.
- RF-07: El usuario debe ser capaz de gestionar los trabajos que ha generado y están en cola de procesado.
- RF-08: El usuario deberá poder visualizar y explorar un modelo tridimensional a partir de la información resultante del procesado.
- RF-09: El usuario deberá poder visualizar paso a paso la información del gestor de intervenciones.

6.1.2 Requisitos no funcionales

Requisitos no funcionales son aquellos que no describen funcionalidades del sistema, describen principalmente aspectos para juzgar la operación del sistema más que comportamientos específicos.
Los requisitos no funcionales definidos de nuestro sistema han sido los siguientes:

- RNF-01: El sistema debe ser capaz de dar soporte como aplicación web alojada en un servicio de ejecución en la nube.
- RNF-02: El sistema debe estar diseñado de tal

forma que pueda funcionar utilizando diferentes sistemas de bases de datos sin la necesidad de modificar gran parte del sistema.
- RNF-03: Al tratar con información médica de pacientes el sistema debe cumplir con las diferentes normativas y legislaciones de tratamiento y protección de datos.

6.1.3 Casos de uso

A partir de la información recogida de los diferentes requisitos se han especificado los casos de uso que se pueden observar en la tabla 1 y su representación gráfica en la figura 2.

Identificador	Nombre	Requisito relacionado
UC_001	Login de usuario	RF-01
UC_002	Logout de usuario	RF-01
UC_003	Subir archivo DICOM	RF-02
UC_004	Consultar tomografías	RF-03
UC_005	Seleccionar tomografías	RF-03
UC_006	Eliminar tomografía	RF-06
UC_007	Localizar nódulo	RF-04
UC_008	Creación de trabajo	RF-05
UC_009	Consultar trabajo	RF-06
UC_0010	Cancelar trabajo	RF-05
UC_0011	Consultar resultados	RF-06
UC_0012	Seleccionar resultados	RF-08
UC_0013	Eliminar resultados	RF-06
UC_0014	Iniciar asistente	RF-09

Tabla 1. Casos de uso y objetivos a los que pertenecen

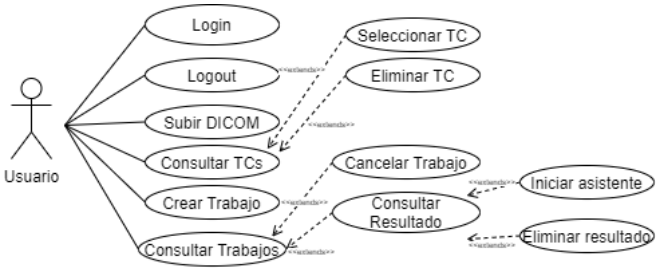

Figura 2. Diagrama de casos de uso

6.2 Herramientas de trabajo y tecnologías

En esta sección se describirán brevemente las herramientas y tecnologías que se ha decidido utilizado durante el desarrollo de este proyecto.

6.2.1 Visual Studio Code

Visual Studio Code es un editor de código multiplataforma desarrollado por Microsoft. Se ha decidido utilizar este editor en lugar de un IDE específico del lenguaje de programación utilizado ya que se va a utilizar distintas tecnologías para desarrollar tanto el servidor como el prototipo del cliente. Este editor soporta multitud de lenguajes distintos debido una gran cantidad de extensiones disponibles. Estas extensiones suelen aportar capacidad de debugado, completado inteligente de código y análisis de sintaxis.

6.2.2 FastAPI

Para el desarrollo del servidor web se ha utilizado el framework FastAPI [11]. Este framework para versiones de Python 3.6 o superiores permite la creación de APIs de alto rendimiento para aplicaciones web. Es un framework sencillo de utilizar que permite generar código de forma ágil y que ofrece la posibilidad de documentar automáticamente la infraestructura del servidor.

6.2.3 Vue.js / Quasar

Para el desarrollo del prototipo del cliente se ha utilizado el framework Quasar [12]. Quasar es un framework basado en Vue.js. Vue es un framework de JavaScript que permite la creación de Single-Page applications, es decir aplicaciones web que interactúa con el usuario de forma dinámica reescribiendo la página en lugar de cargar diferentes páginas independientes como en aplicaciones tradicionales.
Quasar permite la creación de estas aplicaciones de forma que con un único código puede generar aplicaciones que funcionen en multitud de dispositivos como puede ser teléfonos Android, IOs o aplicaciones de escritorio para Windows, Mac y Linux.

6.2.4 Git

Para llevar un control de las diferentes versiones de código y para compartirlo con el resto del equipo se ha utilizado un repositorio Git. Este repositorio se ha alojado en la plataforma GitHub.

6.2.5 MongoDB

Como sistema de base de datos hemos utilizado MongoDB [13]. Mongo es un sistema de base de datos NoSQL orientado a documentos. Se ha decidido utilizar este tipo de base de datos ya que proporciona mucha flexibilidad a la hora de almacenar información ya que los diferentes documentos no tienen por qué tener una estructura determinada.

6.2.6 Celery

Celery [14] es un framework de Python que permite la creación y gestión de un servidor encargado del procesado de tareas de forma asíncrona a la aplicación principal. Esta herramienta se utilizará en el servicio de procesado de nuestra aplicación.

6.3 Diseño

En esta sección se describirán los resultados obtenidos de la fase de diseño. Empezaremos explicando la arquitectura general de la aplicación y explicaremos cual ha sido la distribución en colecciones de la base de datos.

6.3.1 Arquitectura

Tal y como se ha expuesto en los objetivos la aplicación de ha desarrollado como una aplicación basada en la nube. Para hacer esto se ha desarrollado una aplicación basada en el modelo de cliente-servidor, siendo en este caso el servidor la aplicación que hemos desarrollado en Python basada en FastAPI.

La estructura de trabajo de esta parte del proyecto se realizará en lo que hemos llamado engines. Cada engine se encargará de realizar el procesado correspondiente al servicio que representa y estarán comunicados entre ellos a través de un engine principal llamado dispatcher engine.

El cliente prototipo desarrollado en Vue.js/Quasar y actúa como interfaz a partir de la cual el usuario interactúa con el sistema. Este cliente ofrece la posibilidad de realizar cada una de las acciones descritas por los casos de uso.
La estructura de este proyecto viene muy marcada por el framework utilizado. Cada una de las vistas de organizan en páginas y estas páginas se enlazan entre ellas a través del componente llamado router. El almacenamiento interno de la aplicación se realiza utilizando vuex, un gestor de estados, que permitirá guardar los datos y acceder a ellos desde cualquier página de la aplicación.

6.3.2 Colecciones de la base de datos

Como se ha indicado anteriormente se ha utilizado MongoDB como sistema de base de datos. Este sistema está basado en documentos que a su vez están agrupados en colecciones. A continuación, se explicará qué colecciones se ha decidido utilizar y se describirá qué tipo de información almacena cada una de ellas:

- **Usuarios:** En esta colección se almacena toda la información relativa a los usuarios del sistema, es decir, la información utilizada por el módulo de autenticación. Algunos de los campos utilizados por los documentos de esta colección son un identificador único del usuario, el nombre de usuario, la contraseña del usuario, el nombre de usuario que se muestra por pantalla e información relativa al histórico de uso del sistema,
- **Tomografías:** En esta colección se almacena toda la información referente a la información extraída desde los archivos DICOM subidos por el usuario. Estos documentos contienen las imágenes de las tomografías que son utilizadas por el módulo de procesamiento para generar el modelo 3D y calcular los caminos por el árbol bronquial. La estructura de la información de esta colección es la proporcionada por el módulo DICOM parser. Para identificar cada una de las tomografías se utilizará como identificador un identificador único que podrá ser la id del paciente o el número de serie dependiendo del caso.
- **Trabajos:** En esta colección se almacena la información relativa a la cola de trabajos del módulo de procesado. Cada uno de estos trabajos se identifican por un identificador único de trabajo, el identificador del usuario que lo ha creado, el estado actual del trabajo, fecha y hora en que se inició y en caso de error de procesado el motivo por el que no se pudo completar. Una vez estos trabajos han finalizado se añadirá el resultado de estos trabajos como un documento anidado.

6.4 Implementación

En este apartado pasaremos a explicar cómo se ha implementado la aplicación tanto en el backend como en el prototipo del cliente, y cuales han sido los pasos que se han seguido para que el resultado final cumpla con los objetivos propuestos.

6.4.1 Implementación del servicio dispatcher, Obj-1

En la definición del objetivo del servicio dispatcher se ha explicado que su principal utilidad es la gestionar el flujo de lógica y datos entre todos los distintos servicios, actuando, así como una especie de controlador de la aplicación. Para realizar este servicio se ha implementado la clase Dispatcher engine. En esta clase se ha implementado principalmente la lógica de los distintos procesos que involucran la comunicación de distintos servicios de la aplicación, principalmente el servicio web con el resto de los servicios.

6.4.2 Implementación del servicio del control de usuarios, Obj-2

El objetivo de este servicio es el de realizar el control de usuarios de la aplicación, para hacerlo se ha implementado principalmente la clase Authentication. El objetivo de esta clase es el de implementar un sistema de autenticación basado en token JWT o JSON Web Tokens. Estos jwt permiten comprobar que un usuario es quien dice ser, ya que estos tokens incluyen información de la identidad del usuario firmada por el propio servidor, por lo que no sería posible generar este tipo te tokens por otra entidad.
Para implementar estos tokens en nuestra aplicación hemos tenido que realizar funciones que codifican y firman la identidad del usuario, funciones que decodifiquen los tokens proporcionados por el cliente y funciones para verificar que la contraseña proporcionada por el cliente a la hora de realizar el inicio de sesión coincide con la contraseña almacenada de forma encriptada en la base de datos.

6.4.3 Implementación del servicio de procesado de tomografías, Obj-3

El objetivo de este servicio es el de organizar la información de las imágenes médicas de tal forma que puedan ser utilizadas posteriormente por el resto de los servicios de la aplicación. Para hacerlo se ha implementado principalmente la clase Dicom Engine y distintas funciones dentro del propio servicio de dispatcher. Los archivos que recibe la aplicación son archivos comprimidos en formato zip que contienen distintas series de imágenes médicas resultantes de exploraciones de un paciente. Este servicio recibe este archivo, lo descomprime y organiza todos sus contenidos de tal forma que sea accesible por el resto de los servicios. A su vez, se comunica con el servicio de datos para almacenar la información relativa al archivo en la base de datos.

6.4.4 Implementación del servicio de procesado de datos, Obj-4

Este servicio se encarga de crear, organizar y lanzar distintas tareas de procesado de datos de forma asíncrona al

servidor principal. Para hacerlo se ha utilizado el framework Celery. Este framework nos ha permitido crear un servidor "worker" que realizará estos procesados de forma independiente a la aplicación principal. Para utilizar Celery se deben definir los tipos de tareas que debe poder realizar el servidor, en este caso se han definido dos en el archivo tasks, la primera es la segmentación de las imágenes de los pulmones y la segunda es la planificación de caminos a partir del resultado de la segmentación y coordenadas en la imagen proporcionadas por el usuario. Este servicio de ejecuta de manera independiente del resto de la aplicación, por lo que es necesario utilizar un servicio de colas que comunique los dos servidores. Para hacerlo en este caso se está utilizando un servidor RabbitMQ [15]. RabbitMQ es un software de negociación de mensajes entre aplicaciones, en este caso se está utilizando como método de comunicación entre los dos servidores. Nuestra aplicación envía instrucciones y parámetros a través de RabbitMQ al worker de celery, este realiza el trabajo y una vez finalizado envía la información resultante a través de otro mensaje.

Tanto las ordenes de lanzar tareas como el procesado de los resultados de dichas tareas son gestionados por la propia clase Dispatcher engine.

6.4.5 Implementación del servicio de datos, Obj-5

Tal y como se ha expuesto anteriormente, este servicio se encarga de la administración del almacenamiento de datos. Para hacerlo, se ha implementado lo que hemos llamado DatabaseEngine. Esta clase se encarga de realizar las peticiones a la base de datos para obtener o almacenar información. Ya que uno de los requisitos del sistema es que la aplicación debe poder funcionar con distintos sistemas gestores de bases de datos se ha implementado una clase abstracta en la que se han definido distintos métodos que realizan las consultas a la base de datos. Al hacerlo de esta forma se garantiza que en el caso de querer hacer compatible la aplicación con un sistema de base de datos distinto en el futuro, simplemente se deberá implementar una clase basada en esa clase abstracta que realice las consultas en el sistema de base de datos que se quiera implementar. Para la realización de este proyecto se ha implementado a partir de esta clase abstracta una clase que realiza todas las consultas necesarias para el funcionamiento de la aplicación a una base de datos mongoDB.

6.4.6 Implementación del servicio Web, Obj-6

Este servicio es el que responde a las peticiones de una aplicación cliente a través de una API REST, se puede observar un ejemplo de este tipo de peticiones en la figura 3. Tal y como hemos explicado está implementado a partir del framework FastAPI de Python. Las clases y archivos principales de este servicio son los siguientes:

- Main.py: Es el archivo principal de entrada de la aplicación. Inicia una instancia del framework, aplica las configuraciones necesarias y define en qué archivos se van a implementar los endpoints de la API.
- web/routers: En este directorio se encuentran diferentes archivos en los que se definen e implementan los distintos endpoints de la API y todos sus métodos. Es en estos archivos en los que se define qué entradas tendrán las peticiones al servidor y como el servidor responde a estas peticiones.

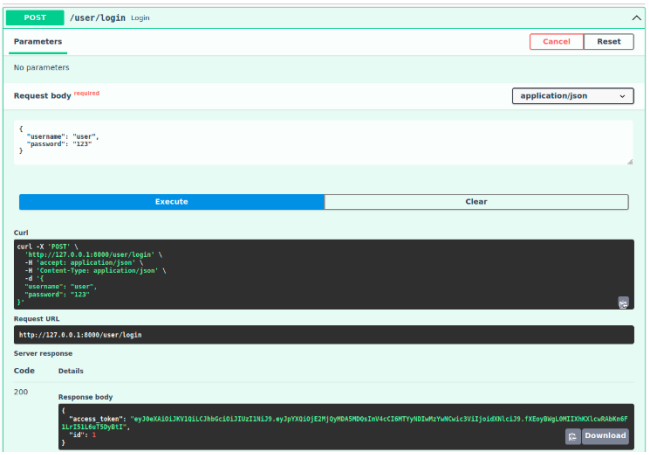

Figura 3. Ejemplo de endpoint del servicio web

6.4.7 Implementación del prototipo de cliente, Obj-7

Por último, se implementó un prototipo de aplicación cliente para que interactuara con nuestra aplicación y poder hacer una validación de que se podían realizar los casos de uso definidos. Para hacerlo se ha utilizado el framework Quasar que a su vez está basado en Vue.js. A partir de este framework se ha creado unas cuantas vistas que proporcionan de manera gráfica acceso a la información expuesta por el módulo web de la aplicación. Estas vistas de explorarán en el siguiente punto del informe y se explicará como interactúan con la aplicación.

6.5 Despliegue

Una vez desarrollada la aplicación se decidió hacer pruebas de despliegue en un entorno de producción simulado. Para ello se ha utilizado una máquina virtual que utiliza Ubuntu Server como sistema operativo y se ha desplegado la aplicación utilizando contenedores Docker.

Se han generado tres contenedores, uno que contiene la aplicación, otro que contiene un worker de celery y un contenedor de RabbitMQ que se utiliza para que los dos anteriores se comuniquen entre ellos. A parte, se ha instalado mongoDB en el servidor ya que es el sistema gestor de base de datos que nuestra aplicación utiliza actualmente.

Por último, se ha creado un documento Docker compose que es utilizado para crear los tres contenedores a la vez.

Todos estos documentos están accesibles en el dosier del proyecto y en el apéndice 2 se puede observar un ejemplo de cómo se crean los tres contenedores Docker.

7 Resultados

En este apartado se explorarán los resultados obtenidos a partir del desarrollo de este proyecto. Tal y como se ha explicado durante el desarrollo el resultado del proyecto ha sido tanto una aplicación web que actúa como servidor y un prototipo de cliente que es el que vamos a utilizar para demostrar el funcionamiento de la aplicación.
Por lo tanto, a continuación, se explicarán las distintas funcionalidades del prototipo y como este interactúa con nuestra aplicación.

7.1 Login

En esta pequeña pantalla el usuario debe introducir una combinación de usuario y contraseña registrada anteriormente en la base de datos por un administrador del sistema. Una vez introducido se envía esta información al servidor en el que se comprueba que sea una combinación válida. Si lo es, el servidor devuelve al cliente la información respectiva al usuario introducido junto a un token JWT que el cliente deberá enviar junto al resto de peticiones para que el servidor pueda comprobar que realmente el cliente está autentificado. En la figura 4 se muestra un ejemplo de inicio de sesión.

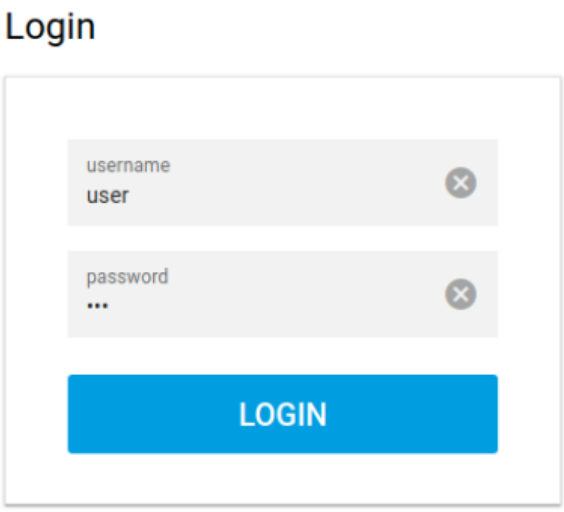

Figura 4. Login de la aplicación prototipo

7.2 Menú principal

Esta pantalla simplemente muestra las diferentes funcionalidades a las que el usuario de la aplicación puede acceder. Estas funcionalidades se explicarán en detalle en los siguientes puntos. Se puede observar este menú en la figura 5.

7.3 Subir DICOM

En esta pantalla se muestra al usuario un pequeño formulario con el que puede enviar al servidor un archivo en formato zip que contenga series de imágenes médicas en formato DICOM. También deberá enviar el identificador de este archivo que, dependiendo del centro médico en el que se utilice el programa, podrá ser tanto un identificador de paciente, de caso, de exploración, etc.

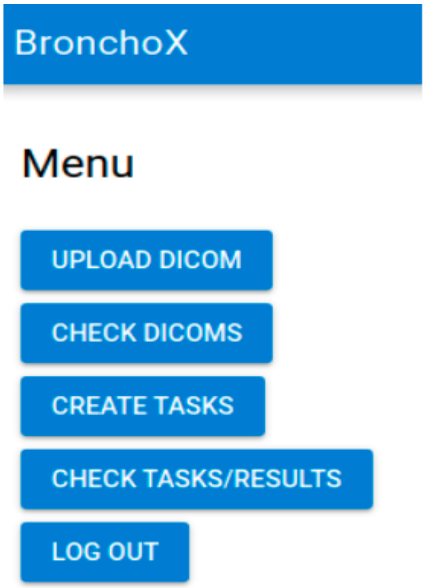

Figura 5. Menú principal de la aplicación prototipo

Una vez el servidor recibe el archivo procederá a descomprimirlo y guardarlo en su memoria interna de manera adecuada y a guardar en la base de datos información relativa a las imágenes y a la localización de dichas imágenes en disco. En la figura 6 se puede ver un ejemplo.

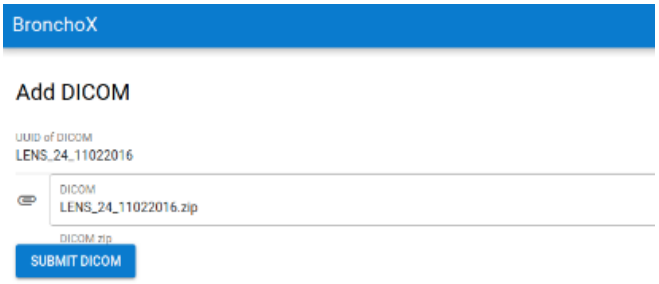

Figura 6. Ejemplo de subida de archivo DICOM

7.4 Consultar tomografías

En esta pantalla se muestra al usuario una lista con todos los archivos de imágenes médicas que han subido anteriormente. Haciendo clic en uno de ellos también puede consultar la información relativa a ese conjunto de imágenes que ha sido almacenada en la base de datos.

También se muestra un botón con el que se le da la opción al usuario de explorar estas imágenes de la forma que se explicará en el punto siguiente, y un botón con el que podrá eliminar esta tomografía del servidor. En la figura 7 se puede ver un ejemplo.

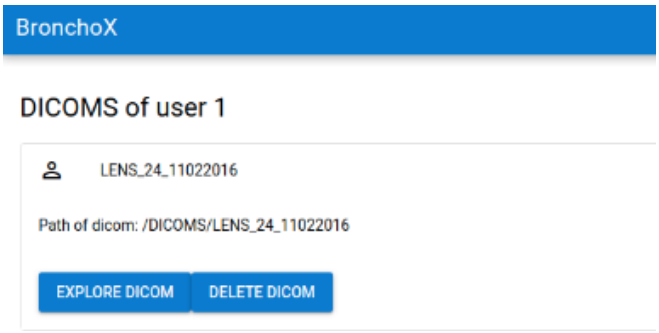

Figura 7. Lista de tomografías subidas por el usuario

7.5 Explorar tomografía

En esa opción el usuario puede explorar uno de los archivos de imágenes que ha subido con anterioridad. Tal y como se puede apreciar en la figura 8 se muestran tres figuras distintas que muestran las diferentes direcciones en las que se han tomado las imágenes que se pueden explorar con el uso de una slider, avanzando o retrocediendo imágenes según el usuario lo requiera. También es posible utilizar uno de los botones laterales para mostrar únicamente una de las vistas si el cliente así lo prefiere. Es visible también el botón con el que se accedería al modelado en 3D, tal como se ha indicado en los objetivos esta funcionalidad será implementada por el equipo de investigación.

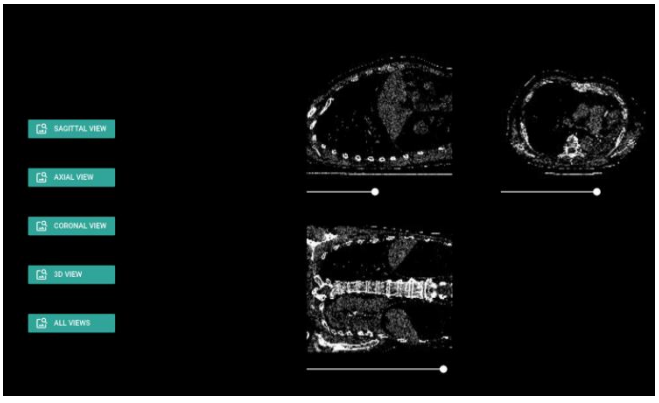

Figura 8. Vista del modelo de exploración de una tomografía

7.6 Crear tarea

En esta pantalla se muestra al usuario un formulario en el que puede crear diferentes tareas asíncronas de procesado. Dependiendo del tipo de tarea que quiera iniciar deberá introducir distintos parámetros. Por ejemplo, en el caso de la tarea de buscar caminos, tal y como se puede observar en la figura 9, se requiere al usuario que introduzca la ruta en la que está guardado el archivo resultante de una tarea de segmentación y una lista de coordenadas de los puntos hasta los que se quiere calcular el camino por el árbol bronquial.

Una vez el usuario envía la información al servidor este mandará una orden al worker de Celery para que inicie una tarea del tipo indicado con los parámetros proporcionados por el usuario. Una vez termine Celery de realizar el procesado enviará de vuelta a la aplicación principal el resultado y esta procederá a guardarlo en la base de datos para que pueda ser consultado posteriormente.

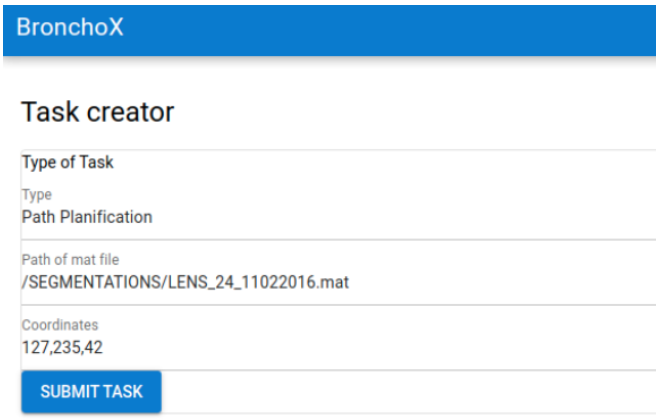

Figura 9. Vista de creación de una tarea de planificación de caminos

7.7 Consultar tareas/ resultados

Una vez el usuario selecciona esta opción se le muestra una lista con las distintas tareas que ha creado anteriormente. Seleccionando una de ellas puede acceder a la información relativa a esta tarea viendo, por ejemplo, el estado de la tarea en ese momento, los parámetros con los que se creó y, si el procesado de esa tarea está finalizado, el resultado de la tarea. Tal y como se ha explicado en los objetivos, se ha implementado la base a la que en un futuro el equipo de investigación podrá añadir las diferentes funcionalidades a cada una de las tareas, por lo tanto, como se puede

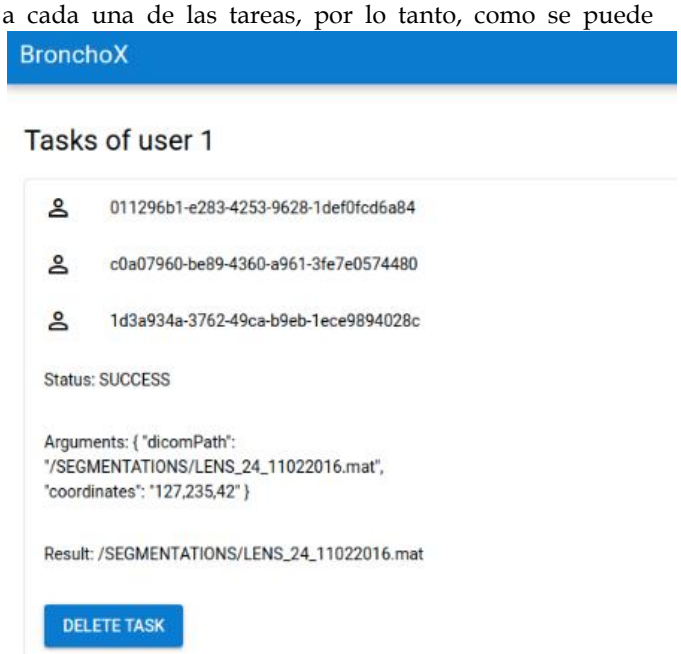

Figura 10. Vista de consulta de las tareas creadas por un usuario

observar en la figura 10 el resultado de estas tareas actualmente es un valor de prueba. Pero con esta prueba podemos comprobar que el sistema de creación de tareas funciona correctamente y que una vez ha finalizado se actualiza esta tarea con el resultado correspondiente. Por último, también se muestra un botón con el que el usuario puede eliminar una tarea junto a su resultado del sistema si así lo desea.

7.6 Cerrar sesión

Esta última opción del menú principal simplemente hace que el cliente elimine de su memoria toda la información relativa a la última sesión iniciada, haciendo que en el caso de que quiera volver a realizar cualquiera del resto de tareas deba volver a iniciar sesión.

8 Conclusiones

Para finalizar comentaremos las conclusiones que han podido obtener de la realización de este proyecto. Para empezar, creo que se han podido cumplir todos los objetivos propuestos al inicio del desarrollo. Se ha conseguido implementar una aplicación que servirá como base para el futuro desarrollo del sistema por parte del equipo de investigación.

Creo que el proyecto tiene potencial de llegar a completarse correctamente y creo que será beneficioso tanto para los equipos médicos como para los pacientes de este tipo de enfermedades que por desgracia son más comunes de lo que se pueda llegar a pensar.

De cara al futuro creo que lo más importante sería el implementar la funcionalidad del gestor de tareas y sobre todo el asistente de intervenciones.

En cuanto a conocimientos técnicos adquiridos, creo que este proyecto me ha servido personalmente para aprender distintas tecnologías con las que no había trabajado hasta ahora, como puede ser FastAPI o Celery y creo que, al haber tenido que adaptarme al estilo de trabajo que ya se estaba utilizando, me ha servido para familiarizarme con lenguajes de programación como Python o JavaScript con los que había trabajado muy poco hasta ese entonces.

9 Agradecimientos

Me gustaría empezar agradeciendo a familia por todo el apoyo que me han dado durante todos estos años de carrera, han sido años duros y me han ayudado mucho a seguir adelante.

Me gustaría seguir agradeciendo el soporte de compañeros y amigos que he conocido durante estos años en la universidad, han conseguido hacer mucho más llevadero el día a día y siempre nos hemos apoyado cuando ha hecho falta. Me gustaría mencionar especialmente a Arnau Real, compañero y amigo que desgraciadamente nos dejó hace un tiempo, siempre estuvo ahí cuando le necesitábamos y gracias a él conocí a gente maravillosa.

Por último, querría agradecer tanto a mis dos tutores del proyecto, Carles Sanchez y Debora Gil, como a Esmitt Ramírez por todo el soporte y consejos que me han aportado durante toda la duración del proyecto.

Bibliografía

[1] "Las cifras del cáncer en España 2020", Sociedad Española de Oncología Médica, 12-03-2021 https://seom.org/seomcms/images/stories/recursos/Cifras_del_cancer_2020.pdf

[2] "Bronchoscopy, How the Test is Performed", MedlinePlus, 12-03-2021 https://medlineplus.gov/ency/article/003857.htm

[3] BronchoX, Centre de Visió per Computador, 12-03-2021. http://iam.cvc.uab.es/portfolio/bronchox

[4] "What is agile methodology? Modern software development explained", InfoWorld, 12-03-2021. https://www.infoworld.com/article/3237508/what-is-agile-methodology-modern-software-development-explained.html

[5] "Kanban Explained for Beginners | The Complete Guide", Kanbanize, 12-03-2021. https://kanbanize.com/kanban-resources/getting-started/what-is-kanban

[6] Web de Trello, 12-03-2021. https://trello.com

[7] Web de PostDICOM, 15-06-2021. https://www.postdicom.com/

[8] Web de RapidAI, 15-06-2021. https://www.rapidai.com/

[9] Web de Arterys, 15-06-2021. https://arterys.com/

[10] Web de Ambra, 15-06-2021. https://ambrahealth.com/

[11] Documentación de FastAPI, 17-06-2021. https://fastapi.tiangolo.com/

[12] Documentación de Quasar, 17-06-2021. https://quasar.dev/

[13] Documentación de MongoDB, 17-06-2021. https://docs.mongodb.com/

[14] Documentación de Celery, 17-06-2021. https://docs.celeryproject.org/

[15] Documentación de RabbitMQ, 15-06-2021. https://www.rabbitmq.com/documentation.html

Apéndice

A1. PLANIFICACIÓN TEMPORAL

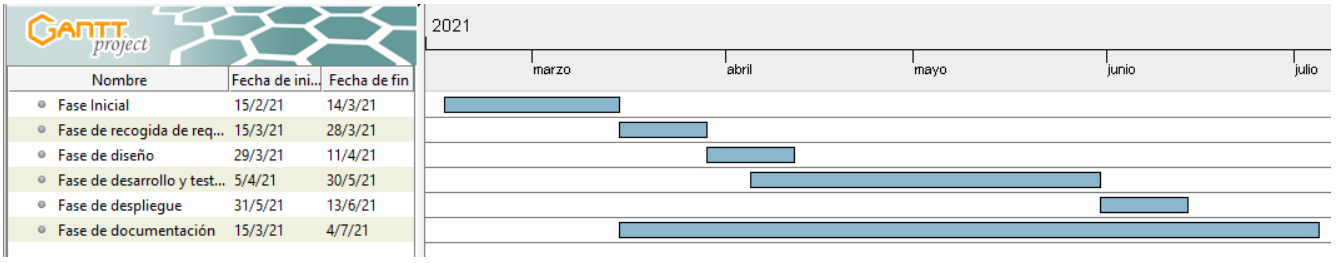

Figura 11. Planificación inicial

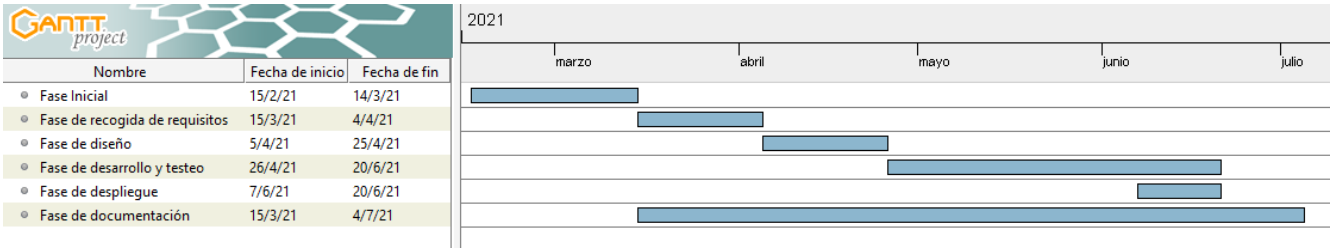

Figura 12. Planificación Final

A2. DESPLIEGUE

```
$ sudo docker-compose up -d --build
Building web
Step 1/6 : FROM python:3.9.5-slim-buster
 ---> 609da079b03a
Step 2/6 : WORKDIR /usr/src/app
 ---> Using cache
 ---> c3e0192da836
Step 3/6 : RUN pip3 install --upgrade pip
 ---> Using cache
 ---> d997e25bf21c
Step 4/6 : COPY ./r3.txt .
 ---> Using cache
 ---> 883f7c52943e
Step 5/6 : RUN pip3 install -r r3.txt
 ---> Using cache
 ---> f459705cfffc
Step 6/6 : COPY . .
 ---> d66efafcf205

Successfully built d66efafcf205
Successfully tagged pruebasdocker_web:latest
Building worker
Step 1/6 : FROM python:3.9.5-slim-buster
 ---> 609da079b03a
Step 2/6 : WORKDIR /usr/src/app
 ---> Using cache
 ---> c3e0192da836
Step 3/6 : RUN pip3 install --upgrade pip
 ---> Using cache
 ---> d997e25bf21c
Step 4/6 : COPY ./r3.txt .
 ---> Using cache
 ---> 883f7c52943e
Step 5/6 : RUN pip3 install -r r3.txt
 ---> Using cache
 ---> f459705cfffc
Step 6/6 : COPY . .
 ---> Using cache
 ---> d66efafcf205

Successfully built d66efafcf205
Successfully tagged pruebasdocker_worker:latest
Recreating pruebasdocker_web_1    ... done
Starting pruebasdocker_rabbitmq_1 ... done
Recreating pruebasdocker_worker_1 ... done
```

Figura 13. Ejemplo de despliegue usando docker-compose