UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería de la Salud (Bioinformática)

Desarrollo de aplicación web FIMED 2.0

Realizado por
Daniel Doblas Jiménez

Tutorizado por
Ismael Navas Delgado
José Manuel García Nieto

Departamento
Lenguajes y Ciencias de la Computación

MÁLAGA, Febrero de 2021

Resumen

FIMED nació ante la premisa de resolver los problemas que presentan los sistemas de gestión de ensayos clínicos. Estos problemas están relacionados con las limitaciones de diseño debido a la naturaleza de los datos clínicos.Esta herramienta estaba basada en páginas JSP *(Java Server Page)* y usa como gestor de base de datos MongoDB, siendo este un gestor NoSQL.

En este trabajo fin de grado se ha desarrollado la actualización de FIMED a FIMED 2.0 realizando cambios tales como, creación de un *Backend* basado en tecnología Python, en concreto usando *FastAPI*, y un *Frontend* usando *NextJS* como *Framework*.

Posteriormente, se ha creado un algoritmo de análisis de datos basado en *Machine Learning* para ejemplificar el uso de la aplicación.

Palabras clave: Desarrollo Web, Machine Learning, NextJS, MongoDB, FastAPI

Abstract

FIMED was developed in order to resolve the current problems in the in the Clinical Trial Management Systems (CTMS). These problems are related with the design limitations due to the nature of the clinical data. This tool was based on JSP (Java Server Page) as Web technology and MongoDB as NoSQL database management system.

In this Degree Thesis, it has been developed the update from FIMED to FIMED2.0 making improvements such as the creation of a Backend based on Python technology, using FastAPI; and a Frontend using NextJS as Framework.

Moreover, we have created an algorithm of data analysis based on Machine Learning to exemplify the use of the application.

Keywords: Web development, Machine Learning, NextJS, MongoDB, Fast API.

Índice

1 Introducción

1.1. Contexto

La bioinformática [13], la biología computacional y la informática biomédica tienen como objetivo común el análisis de datos, ya sea de origen biológico, o de origen clínico. Grandes proyectos de impacto como fue la secuenciación del genoma humano [5], la caracterización de los diferentes tipos de cáncer [6], etc, han ocurrido en apenas 30 años y a una velocidad que los especialistas en sus respectivos campos no son capaces de seguir, esto se puede observar en el gran aumento de datos y publicaciones que ha habido durante estos años (Figura 1).

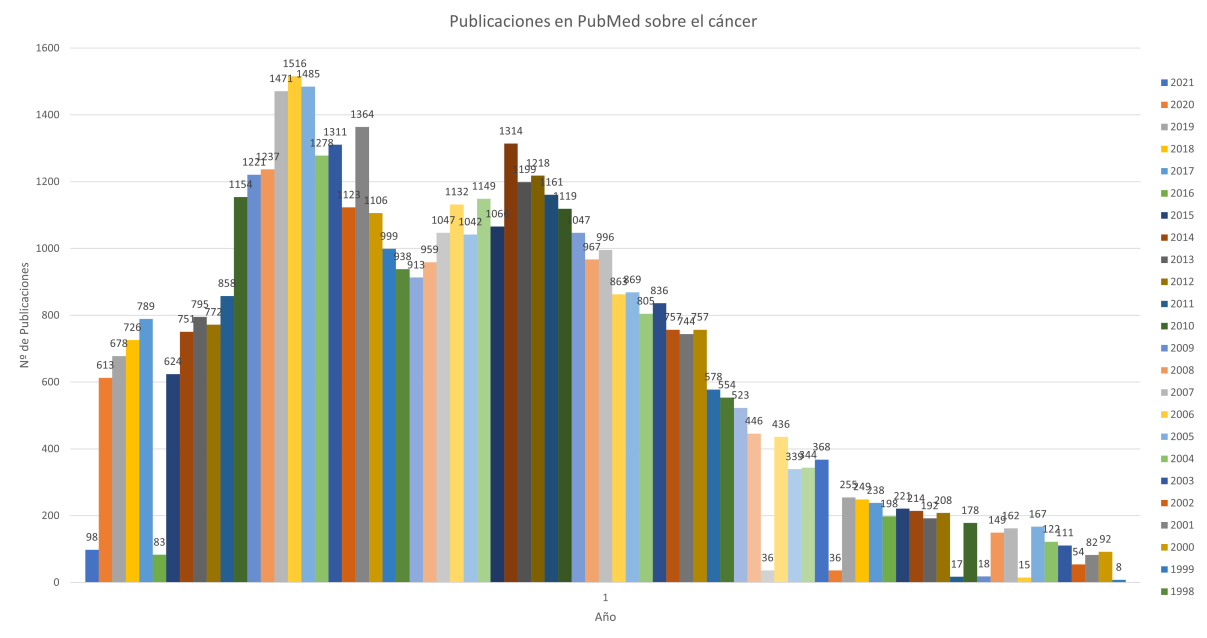

Figura 1: Gráfica en orden descendente de las publicaciones sobre el cáncer en PubMed (1945-2021)

Como consecuencia de este fenómeno, nació la bioinformática y la figura de los especialis-

tas en esta rama de la informática, los bioinformáticos.El área de la bioinformática se pueden dividir en diversas ramas en las que se especializan, algunos puede especializarse en la programación y tratamiento de datos, otros en los conocimientos de genética y biología molecular, etc. Los campos a los que puede acceder un bioinformático van desde la informática pura, hasta la medicina pasando por todas las ramas de la biología.

Estos especialistas generan herramientas para facilitar, agilizar y mejorar los procesos de análisis de datos. Algunas de estas herramientas son:

- REDCap (Research Electronic Data Capture) [4]: Es un software que proporciona un conjunto de herramientas basadas en aplicaciones web para ayudar a los investigadores biomédicos a la captura, gestión y análisis de datos.

- OpenClinica [10]: Es una herramienta electrónica eCRF (del inglés: Electronic Case Report) disñeada para capturar datos de ensayos clínicos.

- Phoenix CTMS [11]: Es una herramienta, con soporte en aplicación web, que combina las capacidades software de las bases de datos de investigación juntándolas en un sistema modular.

Estas aplicaciones, como he mencionado antes, sirven para agilizar los procesos de análisis de datos. FIMED nació como una propuesta más a este tipo de herramientas, facilitando el tratamiento de datos clínicos para realizar diferentes tipos de análisis tales como mapas de calor, clústeres y redes de regulación génica. FIMED intenta aportar flexibilidad a la hora de tratar los datos permitiendo que el usuario (clínico, biólogo e investigador) pueda crear sus propios análisis en un futuro. Es desde esta aplicación desde la que parte este proyecto, cuyo objetivo es actualizar FIMED para aportar un mayor rendimiento a sus funcionalidades y hacerla más escalable.

1.2. Motivación

Como se ha mencionado anteriormente, de la misma manera que la ciencia avanza, la tecnología avanza a una velocidad aún mayor, permitiendo crear herramientas que agilicen y mejoren las técnicas usadas por los especialistas en sus respectivos campos. Es por esta razón por la que las aplicaciones cada cierto período de tiempo, necesitan ser actualizadas ya

que las tecnologías, que hoy día consideramos de vanguardia, en pocos años se convierten en obsoletas. Esta es una de las motivaciones para realizar este TFG, la de ver cómo al actualizar una aplicación, sus funcionalidades se ven afectadas positivamente tanto en rendimiento como en funcionalidades, ya que al crearse nuevas herramientas y, en el caso de la programación, nuevos algoritmos, librerías y módulos.

Además del afán de contribuir en la actualización y progreso de esta herramienta y sus diferentes análisis de datos, este TFG supone una motivación académica de aprender a diseñar aplicaciones web, crear un Frontend y un Backend, enlazar ambas partes y darle soporte con una base de datos (En nuestro caso MongoDB). Además poder aprender a enlazar algoritmos de Machine Learning a la aplicación y ver como estos algoritmos interactúan con esta.

1.3. Objetivos

Los objetivos específicos de este trabajo son:

1. Cambio en la estructura de la base de datos para optimizar las consultas realizadas en el Backend.

2. Actualización del Frontend usando tecnologías más recientes, como NextJS.

3. Creación de una API para el Backend usando FastAPI y Python.

4. Uso de un algoritmo de *Machine Learning* para realizar un ejemplo de la aplicación para el análisis de datos de ensayos clínicos.

Todos estos puntos serán tratados en mayor profundidad en sus respectivas secciones dentro de este trabajo.

1.4. Estructura del documento

En esta sección se resume la estructura de este documento:

1. FIMED 1.0: Breve descripción de las tecnologías usadas en la herramienta original, estructura de la base de datos y problemas actuales.

2. Reestructuración de la base de datos: Descripción de la nueva estructura creada para este proyecto usando tecnologías como MongoDB y HDFS *(Hadoop Distributed File System)*.

3. Creación de los métodos de la API/Backend: Explicación de los métodos usados en la nueva API mediante el uso de FastAPI.

4. Análisis de datos: Discusión sobre la creación del algoritmo *Machine Learning* así como una breve explicación de los pasos a realizar para la ejecución de dicho algoritmo.

5. Conclusiones y trabajos futuros

6. Apéndice A: Manual de instalación, incluyendo referencia al repositorio *Github* para que cualquier usuario pueda probar la herramienta.

1.5. Tecnologías usadas

Como el objetivo principal de este proyecto es la actualización de la herramienta FIMED a una versión con tecnologías más modernas, se va a realizar una descripción de las tecnologías que se han usado para la realización de este trabajo.

1.5.1. MongoDB

MongoDB [15] es una base de datos NoSQL, dentro de ella se encuentra un conjunto de colecciones. Estas colecciones son conjuntos de documentos en formato JSON con estructuras similares aunque no necesariamente idénticas. MongoDB se caracteriza por ser flexible. Es decir, aunque en una colección se mantenga una estructura generalizada en los documentos, si hay otros documentos con otra estructura esto no afecta a la base de datos en si, aunque si a las futuras consultas que se pueda realizar. MongoDB además posee un mecanismo de indexación que nos facilita la búsqueda de los documentos que queramos encontrar en futuras consultas.

Para acceder a las funciones que posee MongoDB, usamos las funciones CRUD (Create, Read, Update y Delete), aunque en nuestro caso, usamos pymongo para realizar las consultas a través de la API.

A la hora de insertar nuevos documentos, MongDB no posee un mecanismo para declarar el esquema de forma explícita. Es decir, el esquema de una colección se construye de forma

implícita según los documentos insertados en la misma. La interacción en la inserción, actualización o borrado de datos se realiza usando documentos JSON. Sin embargo, MongoDB realmente almacena documentos BSON. Este estándar ofrece un conjunto ampliado de tipos de datos, como por ejemplo datos binarios y fechas no existentes en el estándar de JSON.

MongoDB ofrece una gran flexibilidad a la hora de modificar una colección, permitiendo que sea una modificación directa al documento o insertando nuevos datos a la colección.

Por último, MongoDB es escalable por su propio diseño incluyendo características de alto rendimiento y alta disponibilidad. Estas características son fundamentales para nuestra aplicación ya que queremos que sea escalable con el tiempo.

En resumen, las características claves de MongoDB son:

- Almacena datos en documentos flexibles similares a JSON: Esto provoca que no sea necesaria una estructura fija ya que puede variar con el tiempo.
- El modelado de los objetos se realizan a través del código de nuestra aplicación.
- Las consultas ad hoc, la indexación y la agregación en tiempo real nos permite acceder de forma rápida a los datos a la vez que podemos analizarlos.
- MongoDB está distribuida en su núcleo por lo que posee herramientas potentes y fáciles de usar para cualquier usuario.
- Es gratuito.

1.5.2. MinIO

MinIO [7] es un sistema de almacenamiento de objetos distribuido de alto rendimiento. Está definido por software, se ejecuta en hardware estándar de la industria y es 100 % de código abierto bajo la licencia Apache V2[14].

MinIO es diferente porque fue diseñado desde sus inicios para ser el estándar en el almacenamiento de objetos en la nube privada. Debido a que MinIO está diseñado específicamente para servir solo objetos, una arquitectura de una sola capa logra toda la funcionalidad necesaria sin compromiso. El resultado es un servidor de objetos nativo de la nube que es a la vez eficaz, escalable y ligero.

Si bien MinIO sobresale en los casos de uso de almacenamiento de objetos tradicionales como el almacenamiento secundario, la recuperación de desastres y el archivo, es único para superar los desafíos de la nube privada asociados con el aprendizaje automático, el análisis y las cargas de trabajo de aplicaciones nativas de la nube. Las características principales de MinIO son:

- Codificación de borrado: MinIO protege los datos con codificación de borrado en línea por objeto escrito en código ensamblador para ofrecer el mayor rendimiento posible.
- Protección Bitrot: La implementación optimizada de MinIO del algoritmo HighwayHash garantiza que nunca leerá datos corruptos.
- Cifrado: MinIO admite múltiples y sofisticados esquemas de cifrado del lado del servidor para proteger los datos, donde sea que estén.
- WORM: Cuando WORM está habilitado, MinIO deshabilita todas las API que potencialmente pueden mutar los datos y metadatos del objeto.
- Gestión de identidad:MinIO es compatible con los estándares más avanzados en la gestión de identidades.
- Replicación continua: La replicación continua de MinIO está diseñada para implementaciones de centros de datos cruzados a gran escala.
- Federación Global: MinIO permite que que diversas instancias se combinen para formar un espacio de nombres global unificado.
- Puerta de enlace multinube.

En nuestro caso, usamos MinIO debido a que MongoDB posee limitaciones en cuanto al tamaño de los archivos que almacena (máximo 16MB por documento BSON almacenado).

1.5.3. FastAPI

FastAPI [3] es un marco web moderno y rápido que nos permite crear APIs. Usa de base Python 3.6+ y Python estándar como sugerencias. Sus características clave son:

- Rápido, de alto rendimiento. Está situado a la par que NodeJS y Go.

- Rápido para codificar.
- Reduce aproximadamente el 40 % de los errores producidos por los desarrolladores.
- Intuitivo y Fácil de aprender y usar.
- Corto y Robusto. Minimiza la duplicación de código.
- Está basado en estándares abiertos (OpenAPI y JSON Schema).

1.6. NextJS

Es un pequeño framework que realiza renderizado en servidores de aplicaciones basadas en JavaScript. Puede realizar renderizado híbrido estático y de servidor, posee compatibilidad con TypeScript. Además posee grandes características como, por ejemplo, la optimización de imagen, enrutamiento y subdominio integrado con detección automática de idioma y muchas más. En resumen las funcionalidades más destacadas de NextJS [8] son:

- Optimización de imagen: Optimiza automáticamente las imágenes con compilaciones instantáneas.
- Internacionalización: Enrutamiento de dominio y subdominio y detección automática de lenguaje.
- Configuración cero: Recopilación y agrupación automática.
- Híbrido: SSG y SSR.
- Generación estática incremental: Agregar y actualizar páginas estáticas de forma incremetnal después del tiempo de construcción.
- Soporte TypeScript
- Actualización rápida
- Enrutamiento del sistema de archivos: Cada componente de la carpeta *page* se convierte automáticamente en una ruta.
- Rutas API: Capacidad de crear *endpoints* para proporcionar apoyo al Backend.

- Soporte CSS incorporado.
- División y agrupación de códigos.

2
FIMED 1.0

2.1. Descripción

FIMED [12] es una aplicación web (ver Figura 2) cuya premisa es la recopilación de datos clínicos para la ayuda a la gestión de la información en la investigación clínica, usando MongoDBb para proporcionar flexibilidad en el esquema de los datos. De esta forma es posible adaptar los formularios de recogida de datos a un esquema que puede variar durante la vida de un ensayo clínico. Esta herramienta fue desarrollada en lenguaje JAVA, JSP y JavaScript siguiendo el modelo Vista-Controlador desde donde se gestiona la base de datos MongoDB. A través de Tomcat9[1] da servicio a la interfaz de usuario.

2.2. Funcionalidades

Una de las grandes ventajas de FIMED en las funcionalidades que ofrece a usuarios finales (investigadores clínicos):

- Capacidad de crear diseños de formularios personalizados para cada paciente y/o análisis.
- Exploración a la hora de almacenar ensayos de expresión génica asociados a cada paciente usando, además, metadatos para obtener información adicional de las muestras.
- Capacidad de modificar y actualizar todos los datos asociados a los pacientes.
- Herramienta de búsqueda para dar acceso directo mediante filtrado a los datos de los pacientes.

Además de estas funcionalidades, FIMED fue diseñado para ser escalable y así permitir ir aumentando su biblioteca de herramientas de análisis con el tiempo. Ofrece por tanto me-

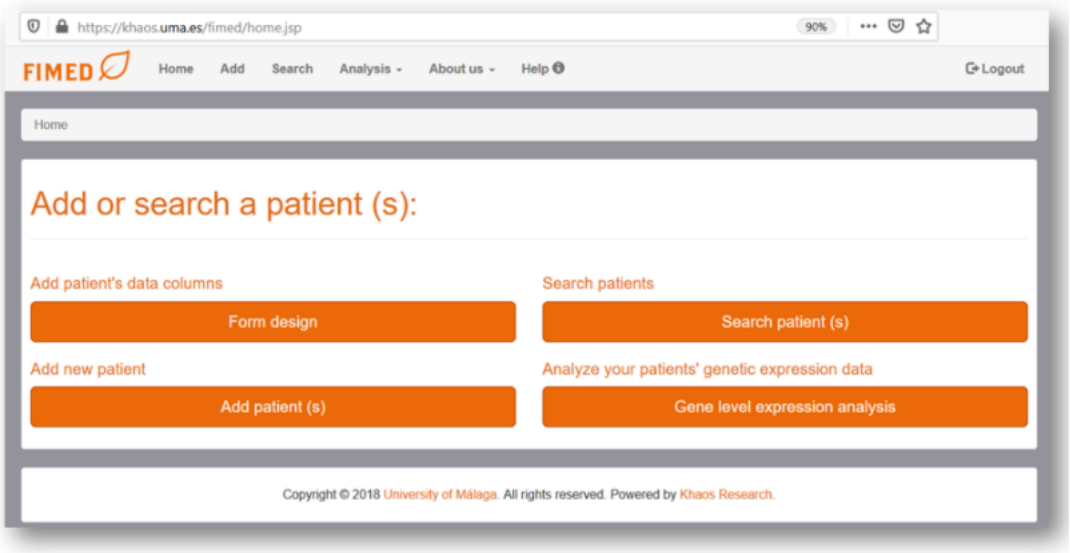

Figura 2: Home de la aplicación web FIMED 1.0

canismos para crear nuevos análisis y enlazarlos a la aplicación sin necesidad de tener que reconfigurar toda la estructura previamente diseñada.

2.3. Análisis

Aquí es donde FIMED explota todo su potencial, con la capacidad que tiene de crear formularios personalizados, FIMED nos permite adaptar esos formularios para poder realizar análisis clínicos.

Esta versión sólo puede realizar análisis mediante la recopilación de ficheros, que se guardan con GRIDFS en la base de datos MongoDB. Los análisis actualmente disponibles son (Figura 3):

- Mapas de Calor (*Heatmap*). Un mapa de calor es una representación gráfica de la magnitud de un fenómeno usando colores en dos dimensiones.
- Clúster de Mapas de Calor (*Cluster Heatmaps*). Un clúster es una agrupación de un conjunto de datos, objetos, personas, etc. Que tienen en común una serie de características.
- Reconstrucción de las redes de regulación de genes (*Gene Regulatory Networks*). Una red de regulación génica es una colección de segnmentos de ADN de una célula que interactúan entre si y con otras sustancias de la célula.

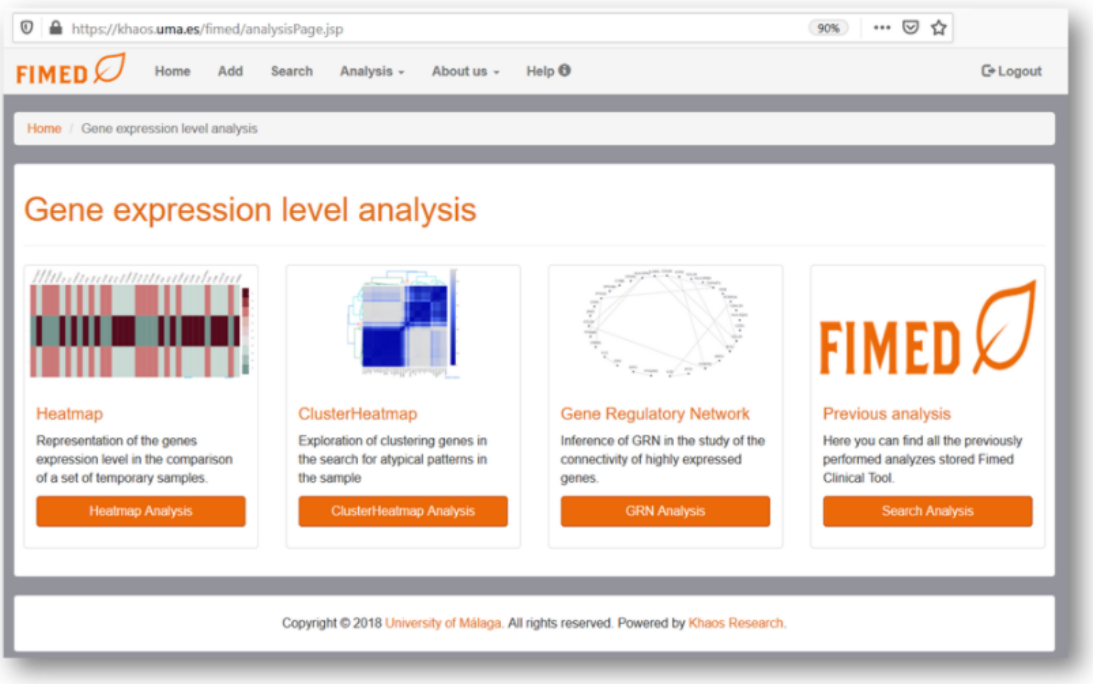

Figura 3: Interfaz de análisis de la aplicación web FIMED

2.4. Problemas

Como se ha mencionado anteriormente, FIMED esta construido en lenguaje JAVA, JSP y JavaScript. Estos lenguajes, a primera vista, no presentan ningún problema pero, a la hora de actualizar el código para hacerlo más accesible e intuitivo de cara a su escalabilidad, aparecen una serie de problemas que son los que hemos tratado y solucionado en este trabajo fin de grado:

- Código complejo y poco entendible.
- Aplicación casi monolítica.
- Uso de scripts en shell para conectar con los algoritmos de análisis.
- Estructura tipo SQL en una base de datos noSQL que no saca provecho de las capacidades de las mismas.
- No permite la creación de formularios a través de CSV.
- No permite la creación de pacientes a través de CSV.

3

Reestructuración de la base de datos

3.1. Introducción

A la hora de plantear la realización de este trabajo, uno de los objetivos era realizar un cambio en la base de datos. Sin embargo, no se había tomado la decisión sobre si debíamos hacer un cambio en la estructura o cambiar de gestor de base de datos. Como primeras iteraciones de los *Sprints*, realizamos los cambios que detallamos en este capítulo.

Una de las primeras mejoras en la que se centra este trabajo fin de grado es en el cambio de la estructura en la base de datos. Aunque la actual aplicación usa como gestor de base de datos MongoDB, su estructura era muy similar a una estructura construida en un sistema gestor de bases de datos relacional. En sí esto no es un problema ya que funciona correctamente pero, usando un gestor noSQL como lo es MongoDB, se perdía la capacidad de agilizar los procesos CRUD (*Del inglés Create, Read, Update y Delete*).

3.2. Estructura de la base de datos de FIMED 1.0

Como se puede observar en la imagen (Figura 4) aunque FIMED 1.0 está montado sobre un gestor noSQL como es MongoDB, mantiene una estructura relacional. Esto con pocos datos no supone un problema, pero nuestro objetivo es que FIMED 2.0 sea escalable y pueda albergar gran cantidad de datos. En este caso puede ser un problema ya que para acceder por ejemplo al nombre de un paciente tenemos que acceder al usuario, luego buscar el paciente, encontrar su información y devolver el campo que contenga el nombre. Son muchos pasos a realizar, por ello pensamos en un cambio de estructura a una que, aunque tenga datos repetidos, son estos datos los que nos permiten obtener el resultado aún más rápido. Esto se detallará más en la

siguiente sección. Esto en lenguaje Python sería:

```
database = get_connection()
col = database.users
user_data = col.find_one({"_id": _id)
patient = user_data["Patients"].find_one({"patient_id":
   patient_id})
patient_info = patient["_patientInformation]
patient_name = patient_info["name"]
```

3.3. Estructura de la base de datos de FIMED 2.0

A continuación se muestra un ejemplo de la nueva estructura. Como podemos observar, es una estructura tipo JSON propia de la base de datos MongoDB, pero no sigue ninguna estructura tipo relacional sino que cada sección está dividida en distintos documentos y cada uno de ellos contiene datos repetidos"de otros documentos. Estos datos son indicadores para poder relacionar los distintos documentos y agilizar las búsquedas en la base de datos. Es decir, al no tener que ir buscando mediante las relaciones como es propio de las consultas SQL, podemos encontrar nuestra información redirigiendo las búsquedas directamente al documento que queremos encontrar, provocando como consecuencia que la velocidad para realizar las consultas sea mucho más rápida.

3.3.1. Usuarios

```
{"User":{
  "_id": "j1",
  "name": "juan",
  "surname": "lblalb",
  "username": "juanusername",
  "password": "eheh",
  "encryption_key": "KDWKDAWDQ3JRQ",
  "patients": ["p1"],
```

```
  "signup_form_design": {
  },
  "analysis_form_design": [{
    "code": "GENE_EXPRESSION",
    "kwargs": {("age", Integer)}
  }]
}
}
```

3.3.2. Pacientes

```
{"Patients":{
  "_id": "p1",
  "name": "pedro",
  "age": 22,
  "surname": "hawdh",
  "samples": ["s1"],
  "data": {
    "Sexo": 2,
    "FNac": 20-10-1996
  },
}
}
```

3.3.3. Archivos

```
{"Files":{
  "_id": "s1",
  "_submitted": 22-10-2020,
  "name": "sample 1",
  "description": "",
```

```
  "url": "",
}
}
```


3.3.4. Análisis


```
{"Analysis":{
  "_id": "a1",
  "_submitted": 22-10-2020,
  "observation": "",
  "analysis_code": "GENE_EXPRESSION",
  "author": "j1",
  "patient": "p1",
  "samples": ["s1"],
  "data": {
    "": "",
    "age": "2"
  },
  "result": ""
}
}
{"Enum":{
  "GENE_EXPRESSION",
  "HUMAN_ACTIVITY_RECOGNITION"
}
}
```


Siguiendo con el ejemplo de la sección anterior, en este caso para acceder al nombre del paciente, tan sólo se necesita el "_id" del paciente que genera MongoDB, sin necesidad de acceder al clínico que tiene a ese paciente. De esta forma nos hemos saltado los pasos de buscar el "_id" del clínico y encontrar, entre todos sus pacientes, el "_id" del nuestro. Al saltarnos estos pasos podemos corroborar que se agilizan las consultas realizadas ya que, como se menciona

anteriormente, al tener datos repetidos en diversas colecciones de MongoDB, podemos acceder a la misma información que teníamos con la estructura relacional pero con mayor velocidad y facilidad. Esto en lenguaje Python sería:

```
database = get_connection()
col = database.patients
patient_data = col.find_one({"_id": _id)
patient_name = patient_data["name"]
```

Como añadido a la mejora de la velocidad de consultas, poseer una estructura tipo noSQL nos permite realizar futuros cambios sin comprometer la integridad estructural de las consultas ni del resto de documentos ayudando así a la escalabilidad que se espera en FIMED 2.0.

Figura 4: Estructura de la Base de Datos de FIMED 1.0

4

API/Backend

Como se mencionó en la sección *Introducción* subsección *Estructura del documento*, en este capítulo se describe de la construcción de la API de FIMED 2.0. Se explicará cada parte de la API desde la conexión a la base de datos hasta la inserción de los pacientes a través de CSV junto con el procesamiento automático del formulario.

Esta API ha sido desarrollada en este trabajo fin de grado para permitir desacoplar el front-end del back-end, y por tanto facilitar la evolución en el futuro de esta aplicación. De esta forma cambios posteriores en la estructura de la base de datos no afectarían a la funcionalidad ofrecida en el front-end. Dado el cambio de tecnología, además, no ha sido posible reutilizar el código existente en la versión anterior.

Para la construcción de la API se ha usado FastAPI que nos permite, además de crear las rutas de conexión entre el Backend y el Frontend, crear la documentación de forma automática. Una vez desplegado el proyecto de este trabajo fin de grado la ruta para ver la documentación es `localhost:8080/api/docs` (Figura 5).

4.1. Creación de Usuario e Inicio de sesión

A la hora de afrontar el inicio de sesión se ha tenido en cuenta la seguridad ya que tratamos con datos clínicos. Para comenzar (ver Figura 6) se ha usado como identificador de usuario el nombre de usuario y el id creado aleatoriamente, ya que de esta forma nos aseguramos que sea único y tengamos una forma de contacto validada con cada usuario. Por otra parte, al principio usamos una contraseña común, sin cifrar, pero se comprobó que, en el caso de que atacaran la base de datos podría obtener las credenciales para iniciar sesión. De esta forma en desarrollos posteriores del trabajo fin de grado se decidió usar la librería de python *CryptContext*. Con esta librería se puede encriptar la contraseña para que, en caso de ataque, no puedan obtener el acceso a las funcionalidades de la aplicación y a los datos clínicos.

Otro problema que se controló en este trabajo fue la presencia de bots. Para solucionarlo usamos la librería de Google para REACT *react-recaptcha*, que nos introduce un reCaptcha para evitar la creación de bots (Figura 7).

A la hora de iniciar sesión (previo registro como se muestra en la Figura 8), creamos y usamos cookies únicas para cada sesión con el fin de controlarlas. Estas cookies controlan el acceso a las funcionalidades, de forma que si el inicio de sesión no ha sido correcto (o se han saltado, debido a un ataque, la fase de inicio de sesión) la aplicación detectará que no se posee la cookie única y redireccionará a la página de inicio (Figura 9).

La cookie es creada al momento de iniciar sesión. Para ello, una vez que se ha iniciado sesión, el algoritmo *OAuth2PasswordBearer* de la librería *.oauth2* crea una clave única generada a partir de una clave secreta, que se ha construido específicamente para este propósito, y un codificador. Esta clave única tiene un tiempo de uso de treinta minutos, tras lo cuál caduca y queda inservible. De esta manera nos aseguramos que si al usuario se le ha olvidado cerrar sesión nadie pueda acceder a los datos de los pacientes pasado ese tiempo. Este control de sesión se realiza a través de la GUI que cada minuto pregunta a la API si la cookie recibida es correcta. En el caso de que no lo sea, se redireccionará, de nuevo, a la página de Login tal y como se mencionó anteriormente.

Esto nos ayuda a tener un nivel extra de seguridad a la hora de acceder a las funcionalidades de la aplicación. Además, la cookie tiene un tiempo límite de vida (30 minutos), una vez finalizado el tiempo, la página se redirige al *Login*.

Otro nivel de seguridad que se añadió al trabajo fue el encriptado de datos de los pacientes usando la biblioteca python *cryptography* y el cifrado tipo *Fernet* del que hablaremos posteriormente. En este apartado y, en relación a este cifrado, cuando se crea un usuario, se crea automáticamente, además, una clave de cifrado única asociada al usuario (Figura 10). Esto nos permite controlar que, en el caso de que algún usuario pueda entrar y acceder a los pacientes de otro usuario, le sea imposible leerlos ya que no posee la clave de desencriptado.

4.2. Métodos CRUD

Los métodos CRUD son los métodos de Crear, Leer, Actualizar y Borrar (del original en inglés Create, Read, Update, Delete). Son funciones básicas en todas las bases de datos. En nuestro caso las usamos para insertar pacientes, leer sus datos, actualizarlos en caso de error

e incluso eliminar pacientes. A continuación describimos con detalle cada uno de los métodos CRUD que hemos usado para desarrollar todas las funcionalidades de FIMED 2.0.

4.2.1. Creación del formulario de pacientes

Como mencionamos anteriormente, la principal funcionalidad de FIMED es la creación de formularios personalizados. Estos formularios se crean desde la Interfaz Gráfica de Usuario (GUI), y se procesan en la API obteniéndose tanto el nombre del campo que se va a rellenar como el tipo de datos *(String, integer, float, boolean...)* (Figura 11). Todo esto con el fin de procesar esos campos para que, a la hora de añadir un nuevo paciente, ya esté disponible el formulario y no tenga que crear uno nuevo y, a la hora de analizar los datos del conjunto de pacientes, tener ya los tipos de datos y evitar posibles errores a la hora de ejecutar los algoritmos de análisis.

Como se ha mencionado anteriormente, los datos clínicos deben de estar cifrados por seguridad, usando una codificación tipo *Fernet* y la clave única del usuario que se generó al crear su cuenta, ciframos todos los datos antes de que se guarden en la base de datos de MongoDB (Figura 12), de esta manera, protegemos los datos clínicos del paciente. Esta clave se genera aleatoriamente usando el método *Fernet.generate_key* de la biblioteca *cryptography*. Esta clave se guardada automáticamente en la base de datos y está asociada a un único usuario. Un ejemplo de clave Fernet sería:

'OU9VRlIyb1N5MFpYakpES1YycG1zS05zSk5vX1dfU1Y3MjBFUVJYcGhiZz0='

Posteriormente se añadió al proyecto un método por el cual mediante un fichero con varios pacientes(Figura 13), formato csv, podemos procesarlo para crear el formulario, procesar cada paciente de manera individual e insertar sus datos cifrándolos durante el proceso y todo esto de manera automática y simultánea.

A la hora de procesar el fichero csv, nuestro algoritmo va línea a línea del fichero, es decir, paciente a paciente, generando diccionarios de cada uno (usando las columnas como claves del diccionario) ya que es lo más parecido a un fichero JSON que es el formato que usa MongoDB. Posteriormente, cada campo del diccionario es encriptado, usando como clave de encriptado la clave única que posee el usuario. De esta forma se asegura que sólo el usuario sea capaz de desencriptar los campos de los pacientes. Tras el encriptado, usamos la consulta en lenguaje Python para MongoDB:

```
database.patients.insert(
{
    "user_id": user_data["_id"],
    "id_secondary": secondary_id,
    "clinical_information": d,
    "created_at": datetime.now()
}
)
database.users.update(
{"username": self.username}, {"$push": {"patients":
   secondary_id}}, upsert=True,)
```

Esta consulta se ejecutará para cada paciente en el fichero CSV.

4.2.2. Lectura de datos

Debido a que los datos están cifrados,se añadió al proyecto un método que descifraba todos los pacientes mediante la clave única que se generaba para cada usuario y se devolvía la información estructurada, de manera genérica para cada paciente, y los envía al *Frontend* en forma de diccionarios de diccionarios python (Figura 14). Este proceso de forma más detallada sería:

- Buscamos todos los pacientes que tiene asociado nuestro usuario.
- Por cada paciente que tenga el usuario, se le aplica el algoritmo de desencriptación, usando como clave la clave única asociada al usuario.
- A medida que se va desencriptando, se devuelve toda la información clínica del paciente directamente a la GUI, donde es procesada para su visualización (Figura 14).

Desde esta interfaz podemos acceder a los dos métodos restantes: Actualización de datos del paciente y eliminación de un paciente.

4.2.3. Actualización de los datos

A la hora de la actualización de los datos, durante el desarrollo del trabajo se pensó que un clínico querría cambiar los datos de cualquier paciente para futuros análisis, o por si por error había introducido datos erróneos. Para ello pensamos que desde el *Frontend* (Figura 15) cambiando cualquier campo de los datos del paciente, sólo esos campos se actualizaran en la base de datos, de esta manera las consultas serían más ágiles ya que no tendríamos que actualizar todos los campos del formulario sino que se actualizarían sólo los campos que han cambiado.

Por ejemplo, si queremos cambiar el campo *Name* que aparece la Figura 15, la consulta en lenguaje Python sería:

```
database.patients.update(
    {
        "id_secondary": id_patient
    },
    {
        "$set": {
            "clinical_information": patients.
   clinical_information
        }
    }
)
```

Es decir, la GUI envía, junto con la petición de actualizar el paciente, todos los datos que hemos cambiado en formato JSON. Una vez esos datos llegan a la API, se encriptan de la misma forma que cuando insertamos un paciente nuevo y, por último, se realiza la consulta mongo en lenguaje Python mencionada anteriormente, que cambia sólo aquellos campos que aparecen en los datos. Esto nos permite actualizar el campo que queramos sin necesidad de cambiar el resto.

La forma que usamos para detectar a los pacientes es el id único que genera MongoDB a la hora de crear los documentos de cada paciente, de esta forma nos aseguramos que modificamos al paciente que queremos evitando problemas de mezclado de datos.

4.2.4. Eliminación de los datos

Para la eliminación de los datos de un paciente o, de un paciente en sí, es muy parecido a la actualización del paciente, usamos su id único generado por MongoDB para buscar el paciente y mediante consultas noSQL eliminamos el paciente de la base de datos.

Para una mayor visualización, la consulta mongo en lenguaje Python que se ejecuta es:

```
database = get_connection()
database.patients.delete_one({"id_secondary": id_patient})
database.users.update(
    {
        "username": self.username
    },
    {
        "$pull": {
            "patients": id_patient
        }
    }
)
```

La consulta busca a nuestro paciente en todas las colecciones en las que está usando su *id_secondary* que se crea al crear al paciente de forma aleatoria, tras encontrarlo lo elimina de la colección de pacientes. En el caso de la colección de usuarios, el id del paciente no se borra sino que se saca de la colección y el sistema de limpieza de mongo lo elimina automáticamente.

A la hora de eliminar el paciente, sólo se debe pulsar el icono ”Xz se elimina de forma automática. El paciente se elimina sin confirmación y no puede ser recuperado de ninguna forma. Posteriormente se pensó en añadir una confirmación para que el usuario no elimine el paciente por error, pero, debido a la extensión del TFG se pospuso el desarrollo para contemplarlo como futuro trabajo.

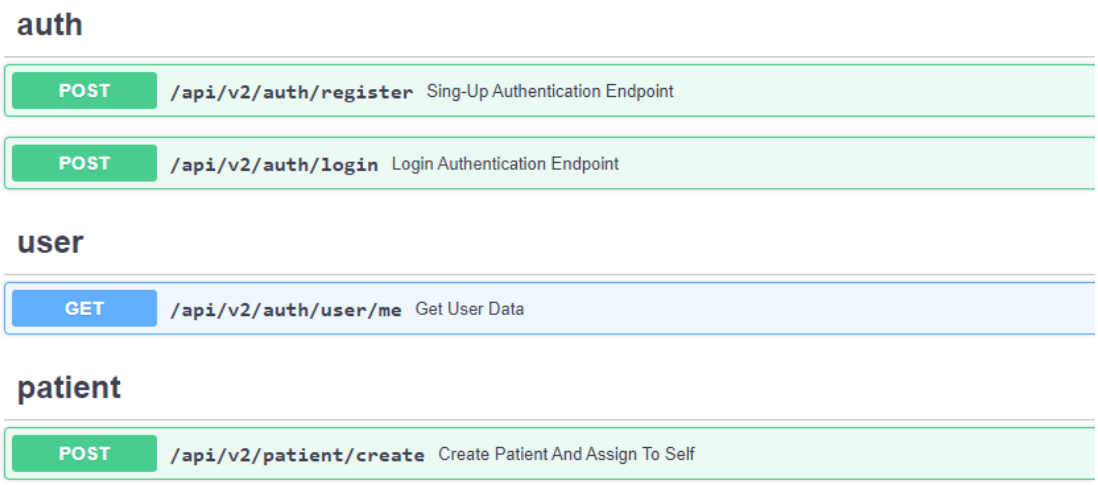

Figura 5: Muestra de la documentación creada por FastAPI de FIMED 2.0

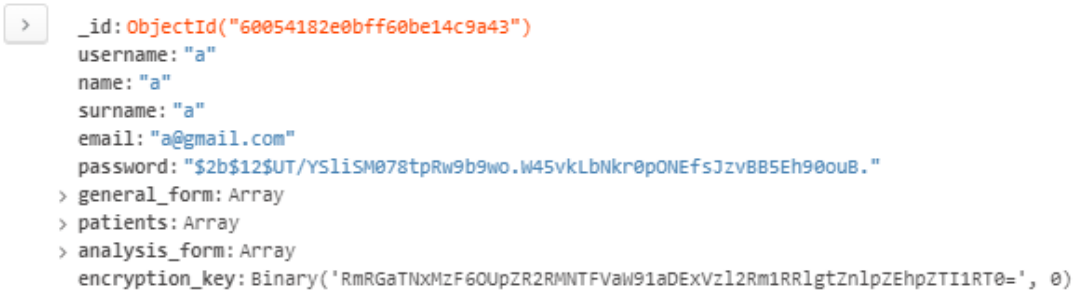

Figura 6: Ejemplo de usuario creado en MongoDB

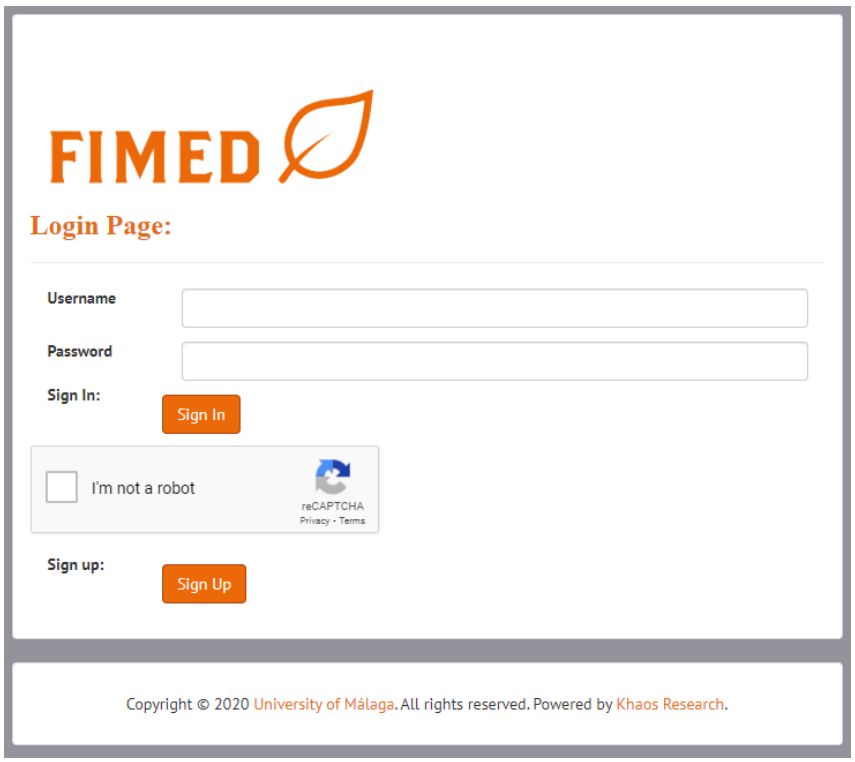

Figura 7: Interfaz de Inicio de Sesión de FIMED 2.0

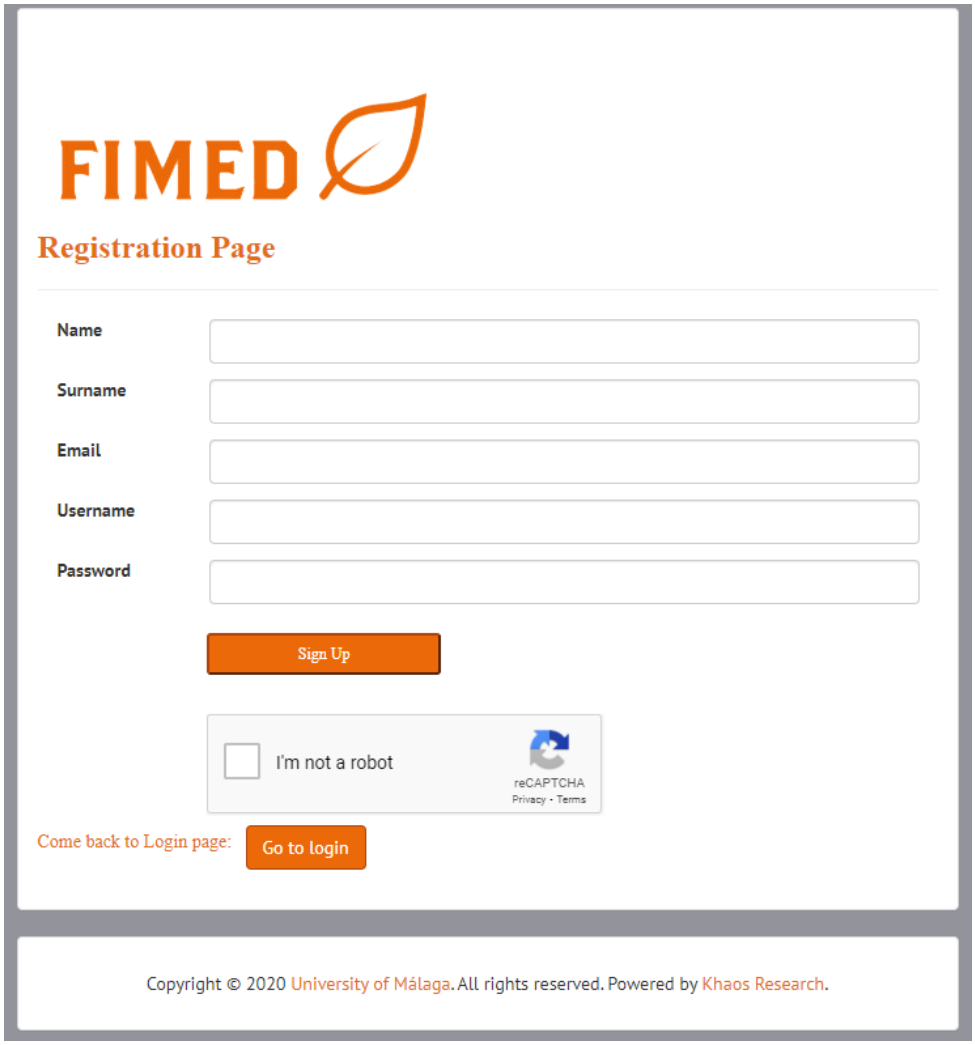

Figura 8: Interfaz de Creación de Usuarios de FIMED 2.0

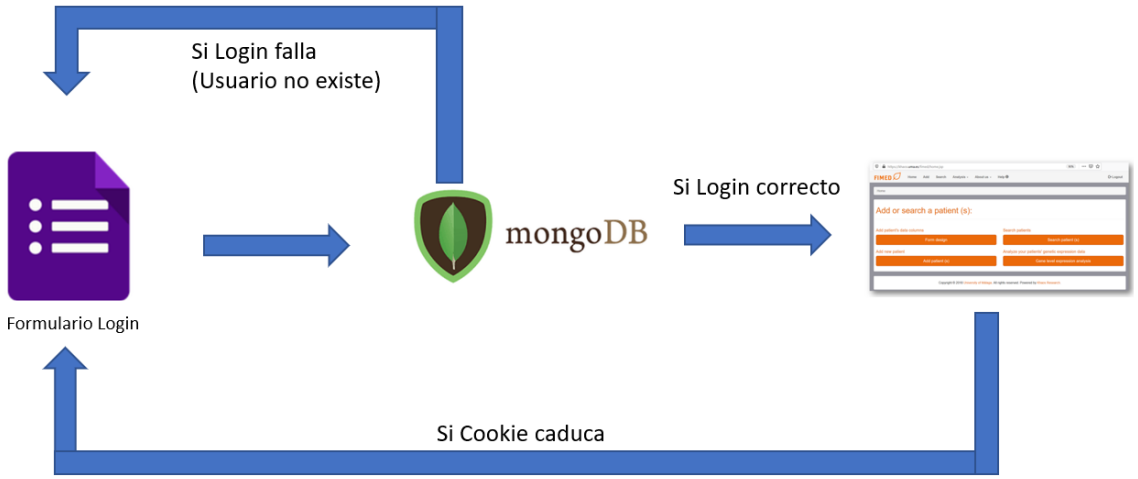

Figura 9: Diagrama del funcionamiento de control de Sesión

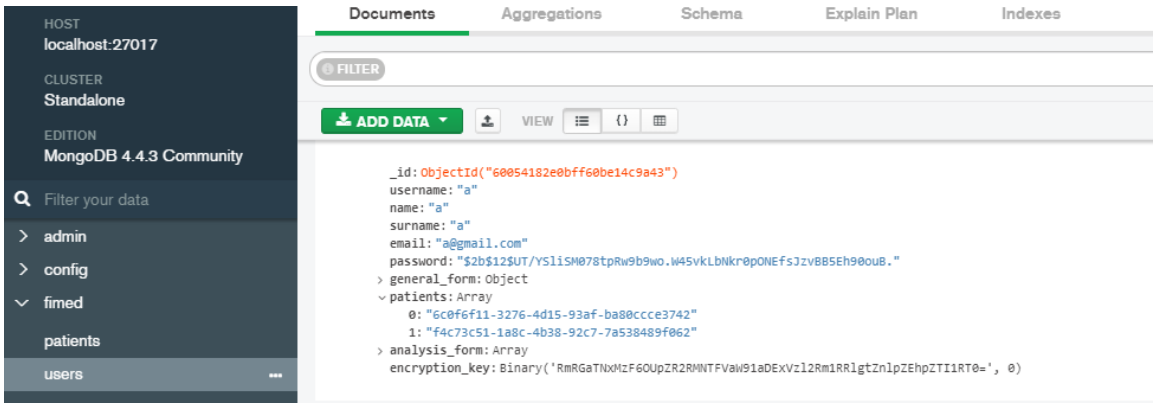

Figura 10: Vista de un usuario creado con pacientes

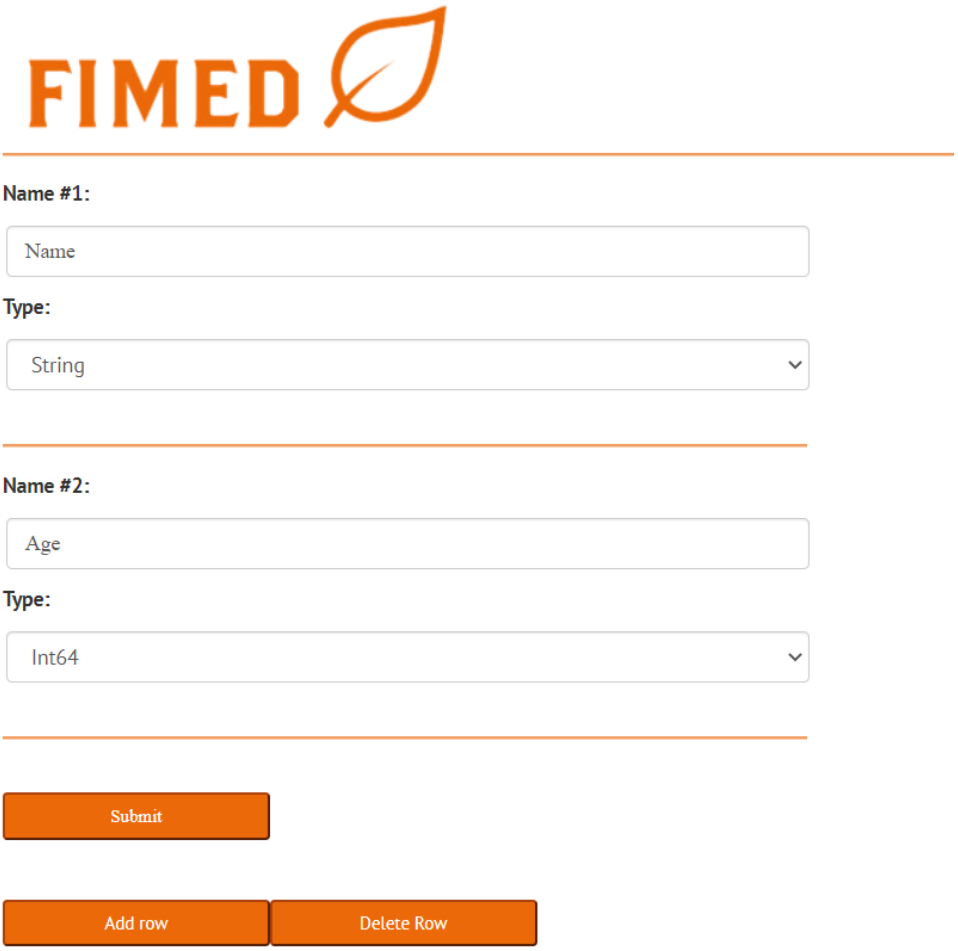

Figura 11: Vista del Diseño de Formularios

Figura 12: Vista de datos de pacientes cifrados en MongoDB

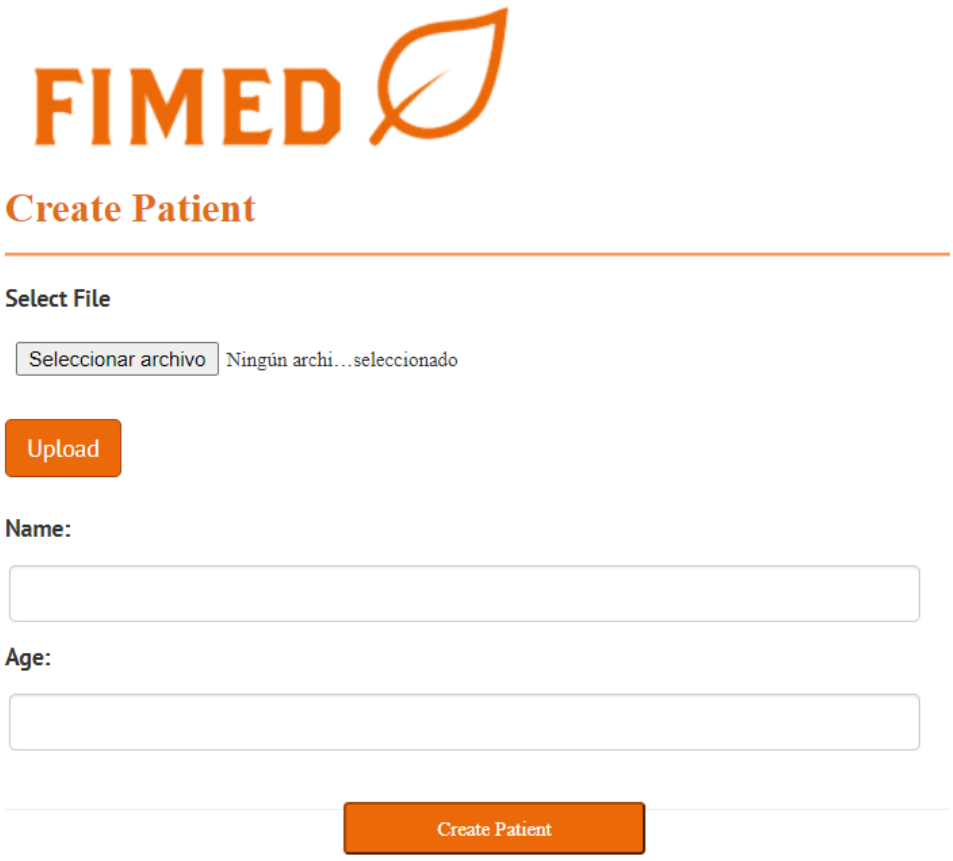

Figura 13: Vista del Diseño del Formulario de creación de pacientes

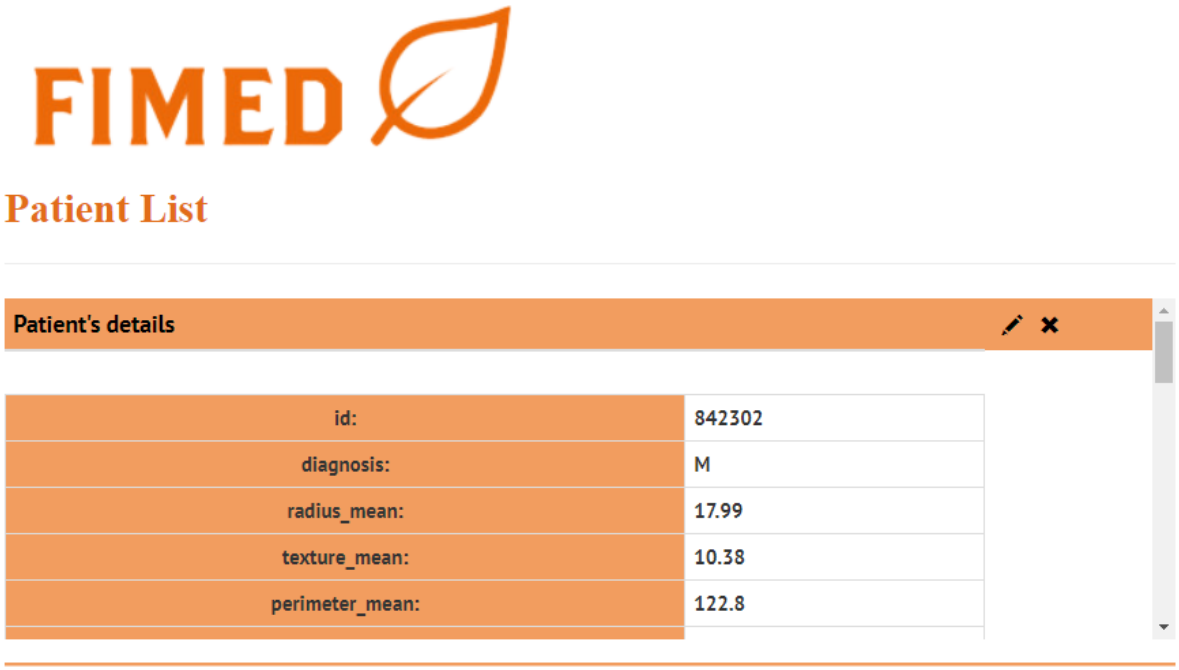

Figura 14: Vista de la Lista de Pacientes

Figura 15: Vista del formulario para actualizar los datos del paciente

5

Análisis de datos

5.1. Introducción

Como hemos estado mencionando, FIMED fué creada para la gestión y análisis de datos clínicos (Ver introducción). En esta sección vamos a describir un ejemplo que hemos creado para poder observar el funcionamiento de FIMED y como se ha habilitado el análisis de datos.

5.2. Descripción de los datos

Para poner en práctica este ejemplo, hemos usado un conjunto de datos para la predicción del cáncer de mama [2]. El conjunto de datos usado se compone de:

- ID.
- Diagnóstico (M = maligno, B = benigno).
- Radio (media de distancias desde el centro a los puntos en el perímetro).
- Textura (desviación típica de los valores en la escala de grises).
- Perímetro.
- Área.
- Suavidad (Variación local de las longitudes de los radios).
- Compacidad $\frac{perimetro^2}{area - 1{,}0}$.
- Concavidad (severidad de las porciones cóncavas del contorno).
- Puntos cóncavos (número de porciones cóncavas en el contorno).

- Simetría.

- Dimensión fractal (” Aproximación de la costa ” - 1).

5.3. Procedimiento

En esta sección hablaremos de los procesos, tanto internos como externos, que hemos realizado a la hora de ejecutar el análisis.

Figura 16: Interfaz del apartado de análisis de FIMED 2.0

5.3.1. Creación de modelo

Aunque FIMED 2.0 ya incorpora un modelo de datos para algoritmos de predicción, hemos querido implementar una función que permita al usuario crear sus propios modelos con sus datos (Figura 16).

Hemos querido que sea lo más simple posible de cara a que el usuario pueda usarlo cómodamente, tras varios intentos de interfaces, nos decantamos por la solución mas simplificada y, a la vez, eficaz para que el usuario pueda usar la aplicación.

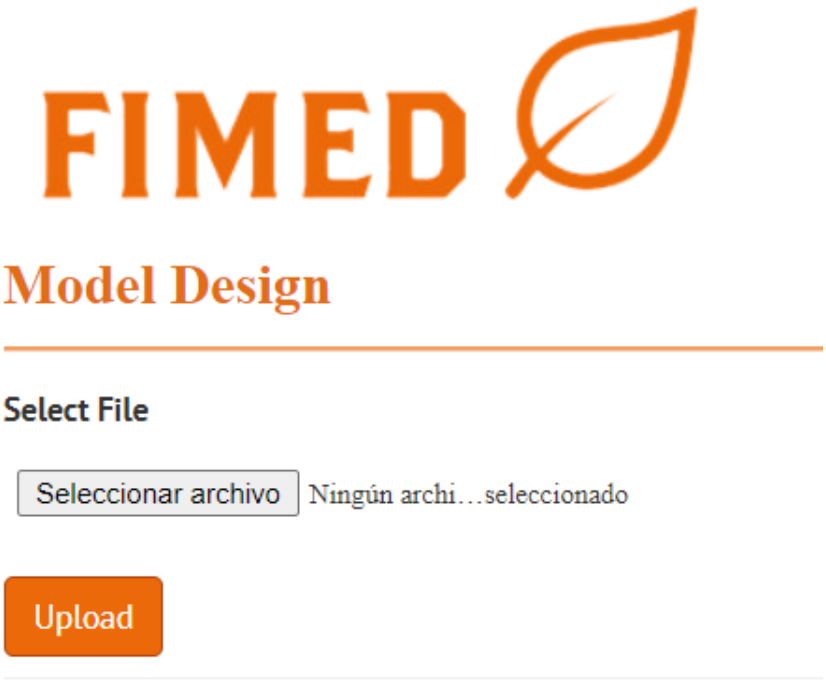

Figura 17: Interfaz creación de modelos de FIMED 2.0

El primer paso es entrar en el apartado de *Model design* (Figura 17), una vez allí simplemente tiene que buscar su fichero y subirlo a MinIO desde la interfaz.

Una vez subido, pasamos a leer los datos, procesarlos, crear el modelo y guardarlo:

Paso 1: Lectura de los datos: Para leer nuestro fichero, hemos usado la libreria *pandas* y el método *read_csv*, esto nos hace obtener un *dataframe* con el cuál podemos trabajar (Figura 18).

```
data = pd.read_csv(file.file, sep=";")
ncolumns = data.columns.size
```

	id	fractal_dimension_worst	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	842302	0.11890	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001
1	842517	0.08902	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
2	84300903	0.08758	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
3	84348301	0.17300	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414
4	84358402	0.07678	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980

5 rows × 32 columns

Figura 18: Resumen de datos del dataframe de cáncer de mama usado como ejemplo

Paso 2: Selección de variable dependiente. La selección de la variable dependiente es clave, ya que es la que vamos a estudiar y en la que nos vamos a basar para comprender los resultados obtenidos tras la predicción. En nuestro ejemplo usamos las variables ”B”(Benigno) y ”M”(Maligno) que nos dicen el tipo de tumor que estamos estudiando.

Paso 3: Transformación de variables categóricas. Debido a que nuestro algoritmo no es capaz de entrenarse usando variables categóricas, cambiaremos nuestras variables dependientes por ‘0’ y ‘1’ respectivamente. Tras la conversión, necesitamos que todos los datos estén estandarizados, así que, mediante el algoritmo *StandardScaler*.

```
X = data.iloc[:, 0:ncolumns - 1].values
Y = data.iloc[:, ncolumns - 1].values
labelencoder_Y = LabelEncoder()
Y = labelencoder_Y.fit_transform(Y)
sc = StandardScaler()
X = sc.fit_transform(X)
```

Paso 4: Creación del modelo. En este punto ya tenemos procesado los datos y procedemos a crear nuestro modelo, hemos elegido el clasificador *RandomForest* ya que en las pruebas previas es el que mejor resultado nos ha dado aunque podría servir cualquier otro.

```
classifier = RandomForestClassifier(n_estimators=10,
    criterion='entropy', random_state=0)
classifier.fit(X, Y)
```

Paso 4: Exportación del modelo. Por último, subimos el modelo creado a MinIO usando el contenedor creado mediante el ”_id”único que genera MongoDB al usuario. Antes de subirlo

pensamos en exportar el modelo a un fichero temporal llamado 'randomForest.joblib' para una mayor facilidad a la hora de importarlo a MinIO. Este fichero se eliminará automáticamente al terminar el algoritmo, de esta forma trabajamos en disco y no en memoria agilizando los procesos de subida de ficheros.

```
dump(classifier, 'randomForest.joblib')
#database
database = get_connection()
col = database.users
user_data = col.find_one({"username": self.username})
#minio
minio = minio_connection()
found = minio.bucket_exists(str(user_data["_id"]))
if not found:
    minio.make_bucket(str(user_data["_id"]))
else:
    print("Bucket 'asiatrip' already exists")

with open('randomForest.joblib', 'rb') as file_data:
    result = minio.put_object(str(user_data["_id"]),
              'model', file_data,
              length=-1, part_size=10 * 1024 * 1024)
print(
    "created {0} object; etag: {1}, version-id: {2}".format
(
        result.object_name, result.etag, result.version_id,
    ),
)
os.remove('randomForest.joblib')
```

5.3.2. Predicción

Una vez creado nuestro modelo de datos, procedemos a ejecutar nuestro algoritmo de predicción, el resultado del algoritmo nos aparecerá en la interfaz (Figura 20).

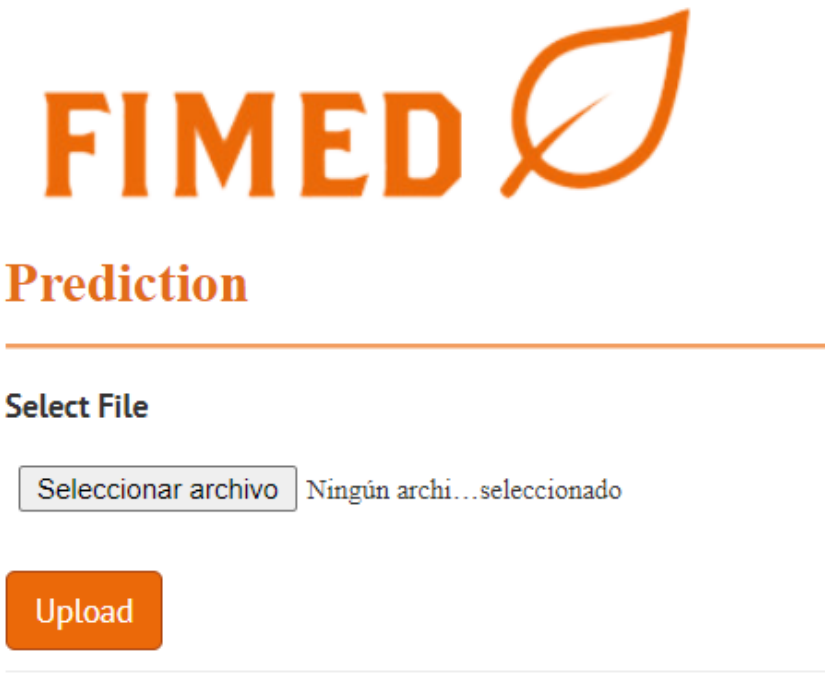

Figura 19: Interfaz uso de modelos de FIMED 2.0

Al igual que en la sección anterior vamos a ir detallando los pasos externos e internos a la hora de usar FIMED 2.0 para realizar la predicción.

Paso 1: Uso del fichero a predecir y descarga del modelo. Para poder ejecutar el algoritmo desde la interfaz, simplemente nos dirigimos a la sección *Prediction* de nuestra interfaz (Figura 16), subimos el fichero que queramos analizar y le damos al botón *Upload* (Figura 19).

Una vez pulsamos el botón, nuestro método de la API descargará el fichero del modelo que previamente hemos subido y lo cargará para que podamos trabajar con él. Esta descarga se realiza de manera similar a la subida, mediante el "_id"único del usuario creado por MongoDB.

```
 #Minio
database = get_connection()
col = database.users
```

```
    user_data = col.find_one({"username": self.username})
    minio = minio_connection()
    minio.fget_object(str(user_data["_id"]), "model", "model.
joblib")
    filename = 'model.joblib'
    loaded_model = joblib.load(filename)
```

Paso 2: Predicción de los datos y mostrar en pantalla. Tras la descarga y carga del modelo, procedemos a la predicción de los datos subidos. Tras la predicción, el resultado aparece en nuestra interfaz (Figura 20).

```
    data = pd.read_csv(file.file, sep=";")
    sc = StandardScaler()
    X = sc.fit_transform(data)
    pred = loaded_model.predict(X)
    os.remove('model.joblib')
    return pred
```

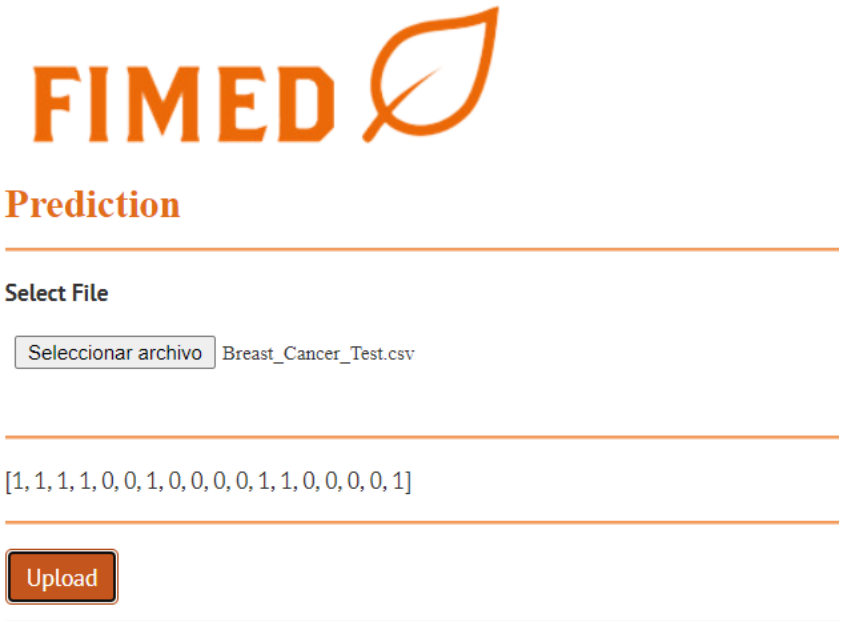

Figura 20: Interfaz uso de modelos de FIMED 2.0 con resultados del análisis

6

Conclusiones y Líneas Futuras

6.1. Conclusiones

Como se ha podido observar a lo largo del proyecto, aplicaciones como FIMED son necesarias ya que aunque los avances en el campo de la medicina son grandes, los avances en el campo de la informática son aún mayores, y ambos campos puede convivir en simbiosis perfecta. FIMED nos abre un mundo lleno de posibilidades, al estar diseñado para ser escalable y modular, permite implementar varias funciones extra al igual que implementar mayor número de análisis que, debido al tiempo escaso de este proyecto, no se han podido implementar.

Además, este trabajo de fin de grado me ha ayudado a afianzar los conceptos adquiridos en el grado tales como el desarrollo y gestión de bases de datos noSQL (Bases de Datos Biológicas), la creación de aplicaciones web usando lenguajes tipo javaScript (Ingeniería Web). He experimentado una mejora en los conocimientos de programación en lenguaje Python y como implementar las diferentes partes de una aplicación web completa para que se conecten entre ellas. Todo esto con un control de versiones usando Git y github. En definitiva ha sido una experiencia que ha puesto a prueba varios conceptos aprendidos en el grado y cuyo resultado ha sido una mejoría considerable en mis habilidades como programador.

6.2. Líneas Futuras

En un futuro, FIMED2.0 será lanzado como un *software* de código abierto para que otros desarrolladores puedan implementar sus propios análisis o módulos. Además, personalmente me gustaría realizar un cambio en la GUI, hacerla más atractiva para el publico y crear una modularización aún más visible a simple vista.

Otros campos a explorar es la capacidad de usar varios modelos de análisis y elegir uno de entre todos según su precisión durante el entrenamiento, de esta forma mejoraríamos las predicciones considerablemente.

A nivel de uso, FIMED puede ser usada en dispositivos móviles ya que es una aplicación web. Pero se podría mejorar su interfaz para que sea más intuitiva para dispositivos móviles e incluso implementar una máquina en un servidor en la nube que pueda crear modelos sin necesidad de tener una máquina con mucha potencia, siendo de esta forma más accesible a los usuarios que estén interesados.

Referencias

[1] *Apache TomcatÂ® - Apache Tomcat 9 Software Downloads*. URL: https://tomcat.apache.org/download-90.cgi (visitado 03-02-2021).

[2] *Breast Cancer Wisconsin (Diagnostic) Data Set*. URL: https://kaggle.com/uciml/breast-cancer-wisconsin-data (visitado 01-02-2021).

[3] *FastAPI*. URL: https://fastapi.tiangolo.com/ (visitado 15-01-2021).

[4] Paul A. Harris y col. "Research electronic data capture (REDCap)â€"A metadata-driven methodology and workflow process for providing translational research informatics support". En: *Journal of Biomedical Informatics* 42.2 (1 de abr. de 2009), págs. 377-381. ISSN: 1532-0464. DOI: 10.1016/j.jbi.2008.08.010. URL: https://www.sciencedirect.com/science/article/pii/S1532046408001226 (visitado 04-02-2021).

[5] Leroy Hood y David Galas. "The digital code of DNA". En: *Nature* 421.6921 (ene. de 2003), págs. 444-448. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature01410. URL: http://www.nature.com/articles/nature01410 (visitado 04-02-2021).

[6] Bong-Hyun Kim, Kijin Yu y Peter C W Lee. "Cancer classification of single-cell gene expression data by neural network". En: *Bioinformatics* 36.5 (2019). _eprint: https://academic.oup.com/bioi pdf/36/5/1360/32793901/btz772.pdf, págs. 1360-1366. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btz772. URL: https://doi.org/10.1093/bioinformatics/btz772.

[7] *MinIO | High Performance, Kubernetes Native Object Storage*. URL: https://min.io/ (visitado 01-02-2021).

[8] *Next.js by Vercel - The React Framework*. URL: https://nextjs.org (visitado 15-01-2021).

[9] Node.js. *Node.js*. Node.js. URL: https://nodejs.org/es/ (visitado 01-02-2021).

[10] *OpenClinica | Clinical Data Management, EPRO and eCRF*. OpenClinica. 13 de mayo de 2020. URL: https://www.openclinica.com/ (visitado 04-02-2021).

[11] *Phoenix CTMS â€“ The ultimative PRS/CTMS/CDMS. Integrated, web-based and free*. URL: https://www.phoenixctms.org/ (visitado 04-02-2021).

[12] sandrohr95. *sandrohr95/FIMED*. original-date: 2020-03-09T10:05:39Z. 2 de abr. de 2020. URL: https://github.com/sandrohr95/FIMED (visitado 15-01-2021).

[13] Álvaro Sebastián y Alberto Pascual-García. "0 - Prólogos e índice". En: Zenodo, 1 de sep. de 2014. DOI: 10.5281/zenodo.1065032. URL: https://zenodo.org/record/1065032#.YBwPgehKiUk (visitado 04-02-2021).

[14] *Welcome to The Apache Software Foundation!* URL: https://www.apache.org/ (visitado 03-02-2021).

[15] *What Is MongoDB?* MongoDB. URL: https://www.mongodb.com/what-is-mongodb (visitado 15-01-2021).

Apéndice A
Manual de Instalación

A.1. Frontend

Para poder usar la interfaz de FIMED 2.0 es necesario tener instalado *NodeJS* [9].

Tras su instalación siga los siguientes pasos:

- Abra una consola de comandos (cmd: Windows, terminal: Mac o Linux).
- Ejecute: git clone https://github.com/dandobjim/FIMED2.0-FRONTEND.git
- Ejecute: cd FIMED2.0-FRONTEND
- Ejecute: npm install (De esta forma todos los módulos se instalará y se podrán usar)
- Por último ejecute npm run dev (Para poder entrar en el modo desarrolador)

A.2. Backend

Para que el Backend de FIMED 2.0 funcione es necesario tener instalado Python y tener un entorno virtual para los paquetes que vamos a usar.

Tras la instalación de Python y la creación del entorno virtual siga estos pasos:

- Abra una consola de comandos (cmd: Windows, terminal: Mac o Linux).
- Ejecute: git clone https://github.com/dandobjim/FIMED2.0-BACKEND.git
- Ejecute: cd FIMED2.0-BACKEND
- Ejecute: pip3 install requirements.txt
- Cambiar el fichero .env.template a .env
- Ejecute: python setup.py

- Ejecute: fimed -h (Para confirmar que se ha instalado correctamente)
- Ejecute: fimed deploy (Para inciar el Backend)

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