



Universidad de Murcia

Grado en Ingeniería Informática

Curso 2020/2021

Sistemas distribuidos

Proyecto : Entrega final

Profesor: Diego Sevilla Ruiz

Jose Antonio Pastor Valera - 48755819M - joseantonio.pastorv@um.es

Rubén Ortega Romero - 48846367W - ruben.ortegar@um.es

Jaime Ortiz Aragón - 49196689B - jaime.ortiza@um.es

Índice

Resumen inicial	2
Front-end	4
Base de datos	8
Servicios	10
Rest : Api Interna	10
GRPC	12
Rest : Api Externa	13
Docker	14
Manual de usuario	16

ficheros y ofrece en todo momento el estado del mismo. Hay que tener en cuenta que se trata de un servicio asíncrono, por lo que se podrá seguir haciendo uso de la aplicación, mientras este trabaja sobre un modelo.

Finalmente, encontramos el segundo servicio REST que está integrado en una api externa que implementa el predictor Word2VecUse. En este caso, al tratarse de un servicio externo, podemos acceder a él desde fuera del frontend realizando una autenticación al usuario que está accediendo al servicio o también, implementamos un acceso desde el frontend en el que no se realiza ningún tipo de autenticación. La prueba de acceso a este servicio la hemos implementado en el cliente de prueba en el que se realiza una petición GET y devolverá las 10 palabras más cercanas a la seleccionada.

Posteriormente entraremos más en detalle en cómo se realizan los intercambios de mensajes entre los servicios.

Front-end

Para desarrollar el front-end hemos optado por utilizar *Flask* en Python, como se ha comentado anteriormente. Hemos optado por esta opción porque trabaja bastante bien con Bootstrap, que ha sido la herramienta que hemos usado para nuestras plantillas.

Cabe destacar que ninguno de los integrantes del grupo había hecho uso anteriormente de herramientas de front-end, por lo que intentamos optar por lo que nos pareció más sencillo. En un principio intentamos comenzar con *Angular*, pero dado que utilizaba múltiples conceptos y forma de programación que no habíamos usado nunca, optamos por Flask.

En nuestra aplicación, la página de presentación, es decir, el índice está formado por un *template* básico que contiene una imagen y dos botones que nos redirigirá a las páginas de inicio de sesión y registro. Ni siquiera hemos necesitado un fichero *css* para dar forma a esta primera página debido a su simplicidad. La redirección a inicio de sesión o registro está definida por las acciones de las estructuras *form* que incluyen a los botones. Esta es la lógica que hemos seguido para las redirecciones y cambios entre páginas en nuestra aplicación proporcionados por flask. De esta forma, se define una función en python para cada una de las acciones. Esta puede ser cargar la página de registro, de inicio de sesión y otras funcionalidades más complejas que se verán posteriormente.

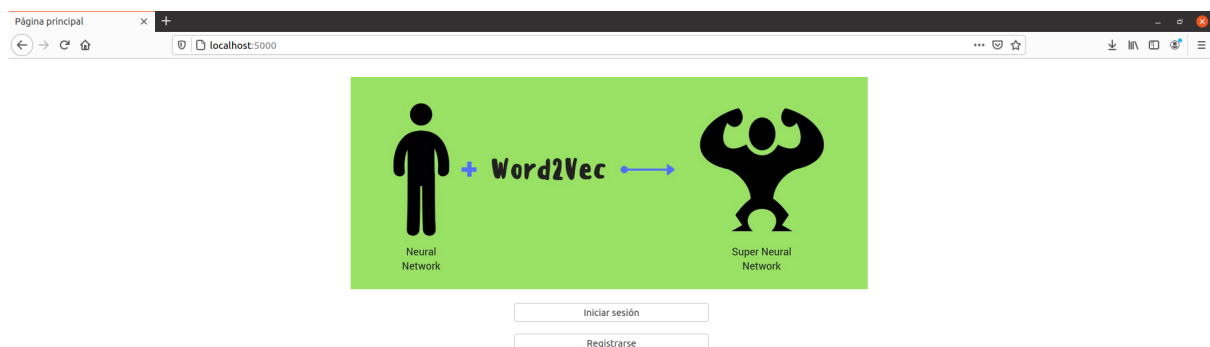


Imagen 2. Pantalla principal de la aplicación.

Según el botón seleccionado accederemos al inicio de sesión o al registro, que son dos templates muy similares. Ambos son un *form* con distintos campos, cuyos valores serán recibidos en el código de la función correspondiente. De esta forma se podrá conectar el front con el back o los datos. Para comprobar las credenciales, se recuperan en la función determinada por el *action* del form, incluyéndose en variables. De esta forma se podrá

acceder a la capa de datos y comprobar si son válidas. En este caso, incluimos para cada uno una hoja de estilo .css que dará forma a los cuadros de texto y la distribución de los elementos.

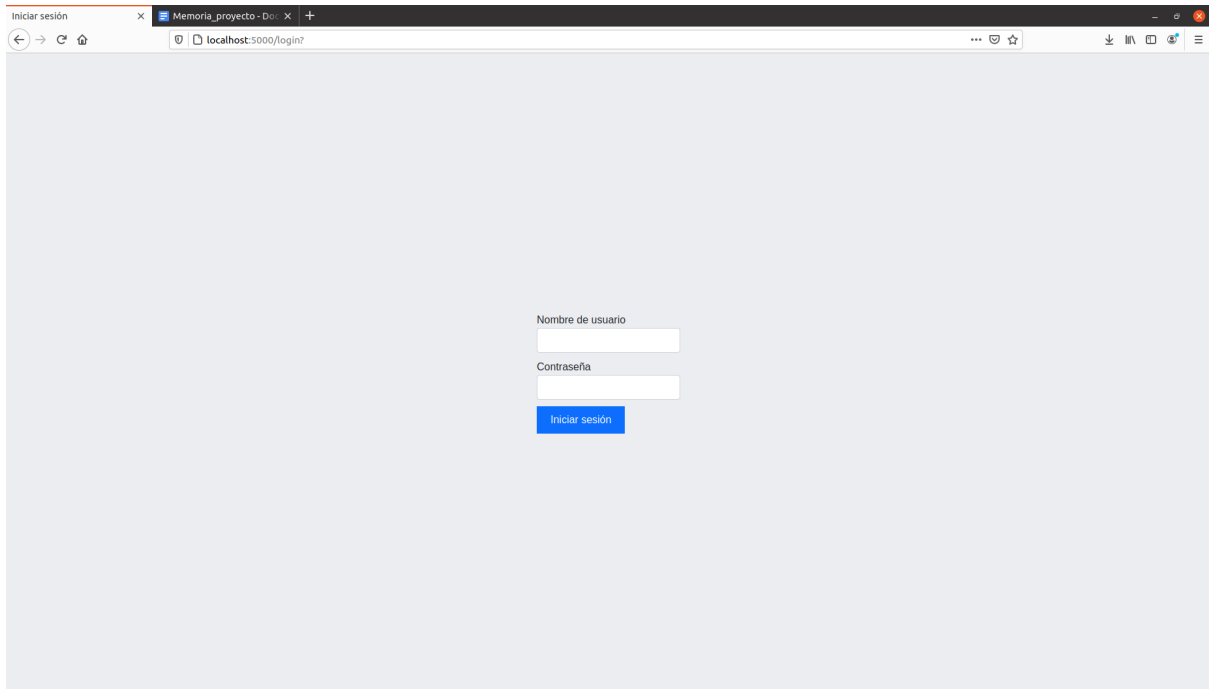


Imagen 3. *Pantalla de inicio de sesión en la aplicación.*

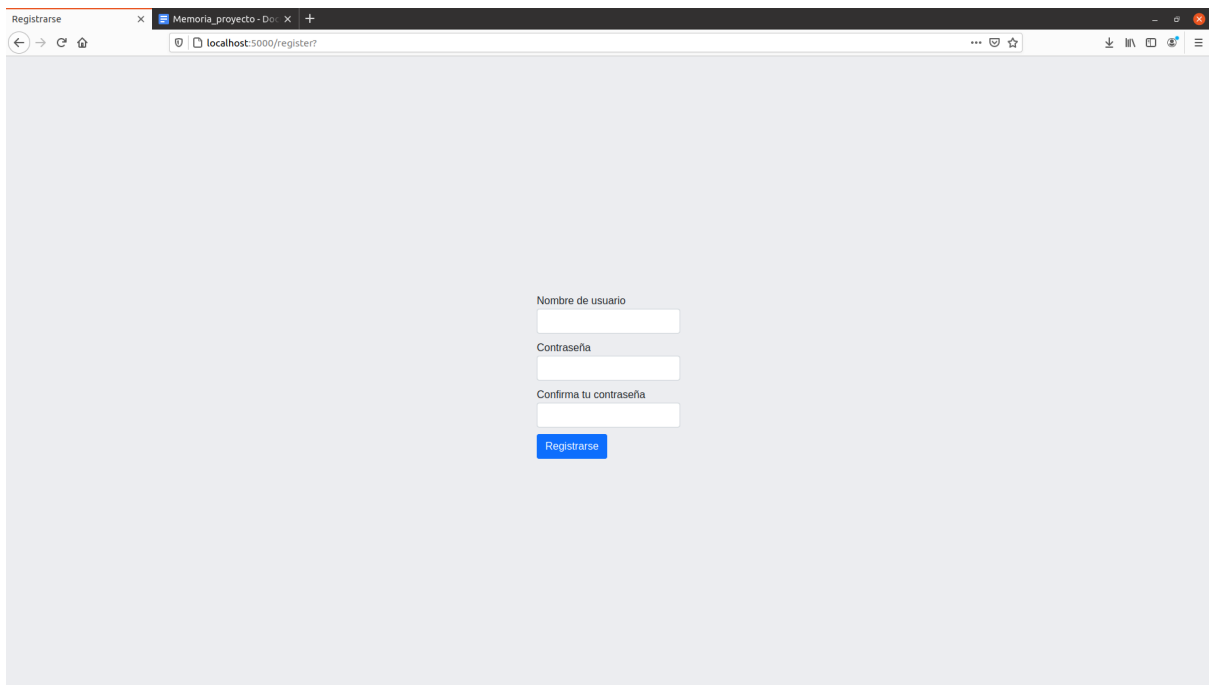


Imagen 4. *Pantalla de registro en la aplicación.*

Cuando el login o el registro concluyen satisfactoriamente se abre la página principal del usuario. En esta reside la principal funcionalidad de la aplicación: En la esquina superior

izquierda se mostrará un saludo, recuperando el valor del nombre de usuario desde el código python que cargó dicho template. Toda la información del usuario, incluido el nombre de usuario se recupera tras una llamada a <http://IP:PUERTO/es.um.ssdd.interna/aplicacion/user/ID>, que será atendida por el servicio REST de la api-interna y devolverá el nombre de usuario, los ficheros subidos, los ficheros entrenados y los datos actualizados de las estadísticas. Estas estadísticas están siempre actualizadas en la esquina superior izquierda, debajo del saludo, ya que cuando se realiza una acción concreta, se accede a la capa de datos para actualizarlas. Los datos de los ficheros están distribuidos en tablas en el template *user.html* con un recorrido de los valores de *files* y *trainings*, que hacen referencia al contenido de las variables que se recuperan en la función que carga esta página.

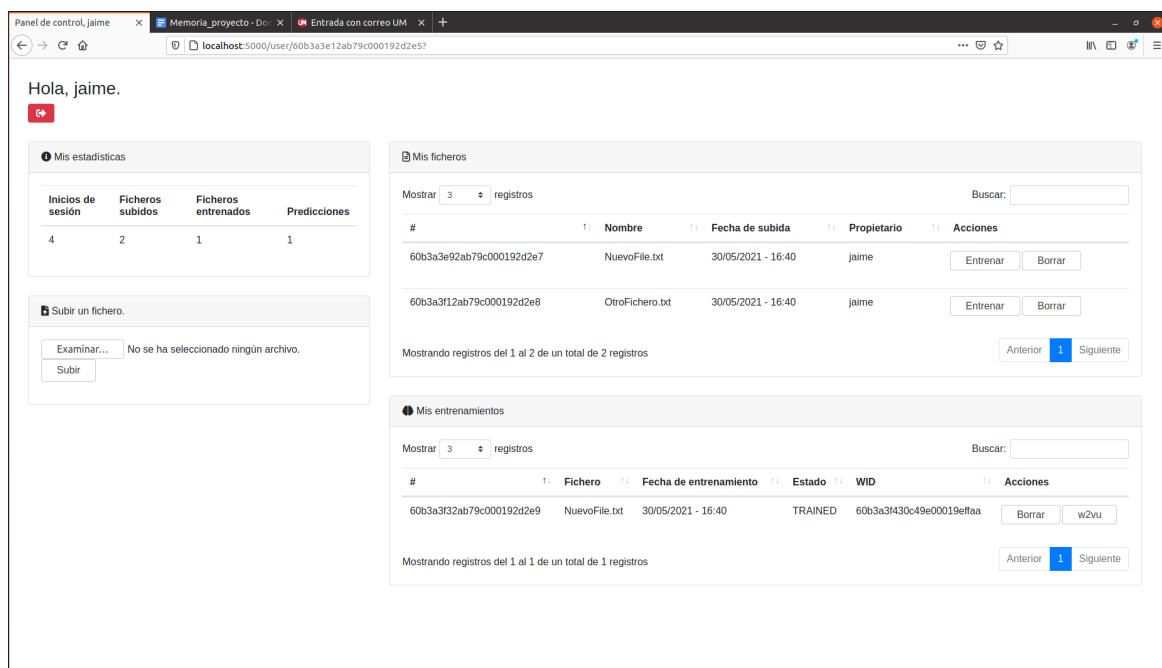


Imagen 5. Pantalla del usuario donde se muestran ficheros, entrenamientos, etc.

Como podemos observar, para las estadísticas se muestra el número de logins que ha realizado el usuario, el número de ficheros subidos y entrenados y el número de predicciones realizadas. En las tablas de los ficheros subidos y los entrenamientos, se muestra la información relativa a los mismos : nombre, identificadores, fechas y propietario. En el caso de los ficheros entrenados, también se muestra el estado del entrenamiento. A la derecha de cada uno de los ficheros subidos se ofrece la posibilidad de borrarlo o mandarlo a entrenar. Si se borra, se eliminará de la base de datos con una llamada al servicio REST y si hace clic en entrenar, este mismo servicio lo enviará al gRPC.

Una vez se ha entrenado el modelo pueden realizarse predicciones. Internamente, el servicio de entrenamiento devuelve un identificador con el que se llevarán a cabo las predicciones, pero esto lo explicaremos en la sección de backend. Si hacemos click en el botón de w2vu que aparece en las acciones del fichero entrenado, se nos abrirá un nuevo template que nos mostrará lo relativo a las predicciones.

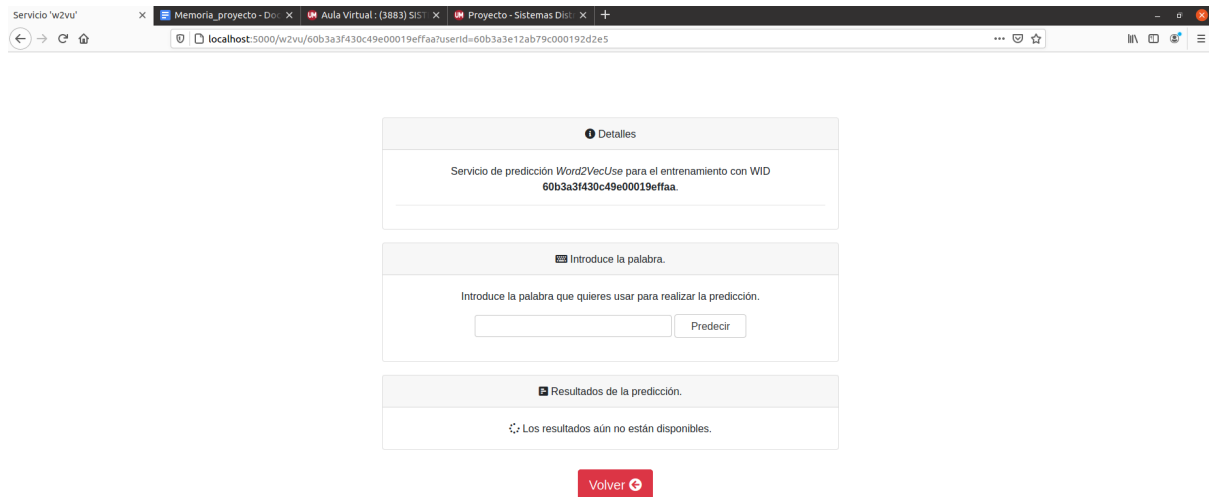


Imagen 6. Pantalla del servicio de predicción sin autenticación.

Cuando introducimos la palabra y hacemos click en predecir, se llevará a cabo la predicción con el servicio externo de REST y cuando termine se nos mostrarán las 10 palabras más cercanas a la introducida:

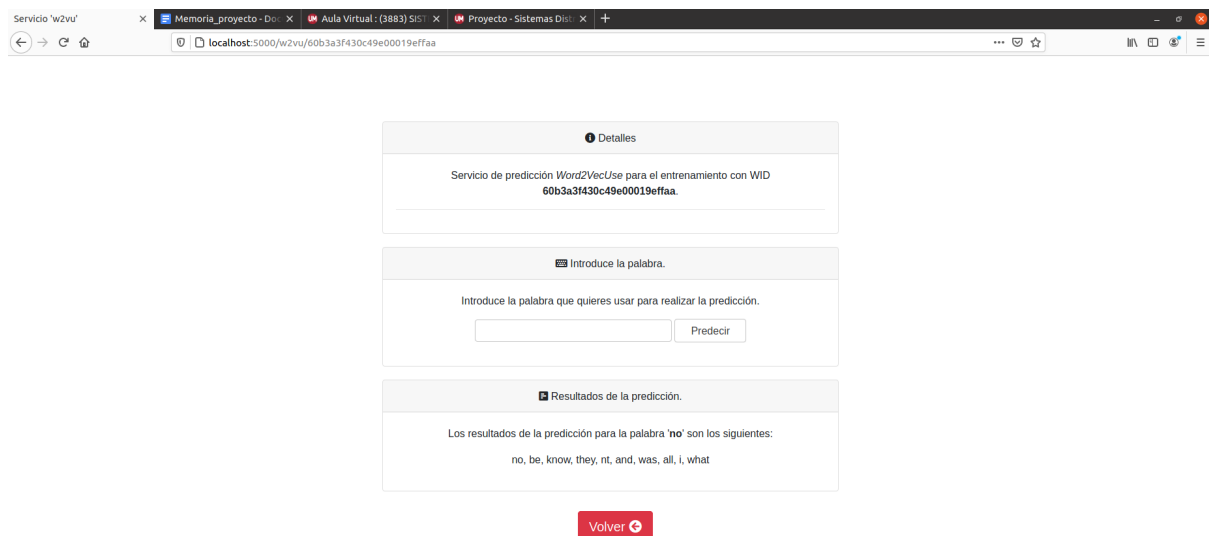


Imagen 7. Resultado de la predicción haciendo uso del servicio Word2VecUse sin autenticación.

Base de datos

En cuanto a la base de datos, hemos optado por *MongoDB*. En un principio intentamos trabajar con *MySQL* porque es la única con la que habíamos tratado anteriormente, pero por simplicidad elegimos Mongo.

Con esta herramienta, podemos almacenar objetos, que contendrán atributos, de la misma forma con la que trabajamos con ellos en el código Python y Java. El nombre que hemos dado a esta base de datos es *mydb* y estará corriendo en el puerto 27017, como indicamos en el fichero *docker-compose*.

```
services:
  # Base de datos MongoDB
  mongodb-ssdd:
    image: mongo
    expose:
      - "27017"
    env_file:
      - ./ssdd.env
    volumes:
      - mongodb_data_container1:/data/db
```

Imagen 8. Entrada para la base de datos MongoDB en el fichero *docker-compose.yml*.

En cuanto a qué vamos a almacenar en nuestra base de datos, tenemos distintos tipos de objetos. Por un lado tendremos los usuarios, que tendrán un identificador único asignado por la propia base de datos, un nombre, contraseña y datos estadísticos sobre sus acciones sobre los ficheros. Además, para realizar las acciones de predicción se genera un token que será almacenado como un atributo más. Por otro lado, también tenemos objetos referentes a los ficheros que se han subido, con el identificador, fecha, propietario... Ficheros entrenados, con atributos parecidos, pero añadiendo un identificador de ruta, que será importante para realizar las predicciones.

Estos objetos están separados por colecciones. Estas colecciones aglutinan a todos los objetos del mismo tipo.

Las referencias a la colección de usuarios se hacen desde el servicio REST interno y desde el front-end. Desde el servicio REST, se crean los usuarios, con el nombre y la contraseña que se pasa desde el front en una llamada a */register*. Estos datos son codificados en JSON, aunque acerca de la codificación de los datos y las llamadas hablaremos posteriormente. Desde el front-end, accedemos a la colección de usuario para comprobar las credenciales de acceso o modificar las estadísticas del usuario cuando se realiza un nuevo inicio de sesión, se sube un nuevo fichero, se manda a entrenar o predecir.

En lo referente a la colección de ficheros subidos, esta almacenará el identificador del mismo, el nombre del fichero y su contenido, el usuario propietario y la fecha de subida. Esta colección es accedida para añadir un nuevo fichero a la base datos o borrarlo.

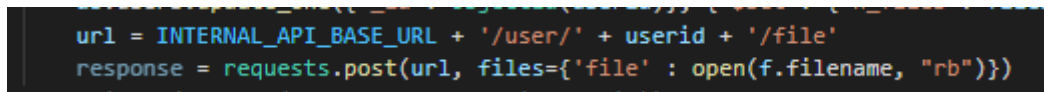
Las operaciones sobre la colección de ficheros entrenados es muy similar. En este caso, en vez de producirse la adición cuando subimos el fichero, se produce cuando el usuario manda a entrenar un fichero. En este caso, los atributos que contiene son el identificador, el nombre del fichero, su contenido, el propietario, la fecha y el WID. Este WID es el identificador del entrenamiento que usaremos para realizar las acciones de predicción. De la misma forma, también se puede eliminar un entrenamiento realizado.

Servicios

Rest : Api Interna

El primer servicio que hemos desplegado en el backend se corresponde con el soporte del frontend. En este caso se trata de un servicio REST desarrollado en Java. Este REST interno ofrece toda la información necesaria por el frontend en forma de URLs de recursos REST. Estos a su vez hacen uso de la capa de datos descrita en el apartado anterior. Los recursos proporcionados son:

- **/user/ID**: Donde ID se corresponde con el identificador del usuario almacenado en la base de datos y devuelve la información correspondiente a ese usuario, ficheros subidos, ficheros entrenados y estadísticas.
- **/user/ID/file**: Con este recurso podemos subir un nuevo fichero a la lista de ficheros del usuario. En este caso se enviará en un POST desde el frontend, como se muestra a continuación:

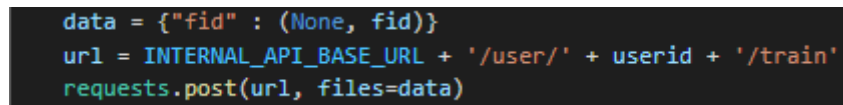


```
url = INTERNAL_API_BASE_URL + '/user/' + userid + '/file'
response = requests.post(url, files={'file' : open(f.filename, "rb")})
```

Imagen 9. Fragmento donde se hace la petición a la API interna para subir un fichero.

Esta petición es procesada por el servidor REST, que obtendrá el usuario y añadirá el fichero a la colección *files*. Este servidor responde a la petición del frontend con un 201, informando del identificador del fichero, al que llamaremos a partir de ahora con FID.

- **/user/ID/train**: Con el FID que se ha generado al subir el fichero, podemos realizar entrenamientos. Esto se lleva a cabo conectándonos con un hilo que estará corriendo en el puerto 50051, llamado ServiceClient, que se conectará con el servidor GRPC.



```
data = {"fid" : (None, fid)}
url = INTERNAL_API_BASE_URL + '/user/' + userid + '/train'
requests.post(url, files=data)
```

Imagen 10. Fragmento donde se hace la petición a la API interna para entrenar un fichero.

Estos son los recursos que se indican en el documento, pero existe mucha más funcionalidad adicional:

- **Registrar un usuario**: Desde el frontend, se recuperan los atributos desde el formulario y se envían al recurso **/register** en un POST.

```
data = {"username" : (None, username),
        "password" : (None, password)}

url = INTERNAL_API_BASE_URL + '/register'
res = requests.post(url, files=data)
```

Imagen 11. Fragmento donde se hace la petición a la API interna para registrar un usuario.

Este comprobará que no existe ningún usuario con dicho nombre y lo añadirá a la base de datos.

```
if ((id = c.getMongoController().registerUser(nombre, hashed)).equals("")) {
    return Response.status(Response.Status.CONFLICT).build();
}
```

Imagen 12. Fragmento de código de la API interna donde se intenta registrar al usuario.

- **Información de un fichero:** Cuando se accede a la página principal del usuario, para cada uno de los ficheros, se recuperan sus atributos para rellenar la tabla con los datos de los ficheros del usuario.

```
files = []
for file in values[1]:
    url = INTERNAL_API_BASE_URL + '/user/' + userid + '/file/' + file
    response = requests.get(url)
    files.append(json.loads(response.text))
```

Imagen 13. Fragmento de código donde se realiza la petición a la API interna para solicitar la información de cada uno de los ficheros del usuario.

Estos, se recuperan accediendo al recurso `/user/ID/file/FID`, que devolverá un objeto JSON con los campos. Esta información se recupera accediendo a la colección de ficheros de la base de datos.

```
JSONObject object = new JSONObject();
ConfigAPI c = ConfigAPI.getInstance();
try {
    object.put("fid", fid);
    object.put("nombre", c.getMongoController().getFileName(fid));
    object.put("fecha", c.getMongoController().getFileDate(fid));
    object.put("propietario", c.getMongoController().getUserName(userid));
} catch (JSONException e) { e.printStackTrace(); }

return Response.ok().entity(object.toString()).build();
```

Imagen 14. Fragmento de código en la API interna donde se devuelve la información del fichero.

- **Información de un entrenamiento:** En este caso, la funcionalidad es equivalente a la de mostrar información de un fichero, pero en este caso el recurso accedido será `/user/ID/train/TID` y la información de la base de datos, será de la colección `trained_files`.

- **Eliminación de un fichero:** Elimina el fichero de la colección files, de la base de datos.

```
url = INTERNAL_API_BASE_URL + '/user/' + userid + '/train/' + tid + '/remove'
requests.post(url)
```

Imagen 15. Fragmento donde se hace la petición a la API interna para eliminar un fichero.

- **Eliminación de un entrenamiento:** Manda la acción de eliminar el entrenamiento al servicio GRPC por medio del cliente.

```
url = INTERNAL_API_BASE_URL + '/user/' + userid + '/train/' + tid + '/remove'
requests.post(url)
```

Imagen 16. Fragmento donde se hace la petición a la API interna para eliminar un entrenamiento.

Todas las acciones que hemos comentado anteriormente, menos las de GRPC, que las trataremos en el siguiente apartado, son llevadas a cabo por un controlador que hace uso de los adaptadores DAO de usuarios y ficheros. Estos son realmente quienes tienen la funcionalidad en la que recae la aplicación sobre los objetos User y File. Estos dos objetos han sido creados como objetos POJO.

GRPC

El objetivo del servicio GRPC consiste en entrenar modelos. En este caso los modelos serán los ficheros que se han subido desde el frontend al servicio REST interno.

Antes se ha mencionado, que los métodos que llamarán a la API de GRPC serán las que posean las funcionalidades relacionadas con el entrenamiento y borrado de ficheros, es decir, los métodos *TrainFile* y *RemoveTraining*.

Estos se comunicarán con el cliente (*ServiceClient*), que realizará las peticiones sobre el servidor el cual pertenece a la clase *ServiceServer*, cuyo hilo se lanza independientemente. Este servidor se apoya en la clase *ServiceImpl* para entrenar el modelo:

El método *uploadFile* se encarga de ir almacenando el contenido del fichero en un fichero temporal por partes, este fichero temporal completo se corresponderá con el fichero que ha mandado a entrenar el usuario desde sus ficheros subidos. Además se crea el objeto y se añade a la colección *trained_files* de la base de datos por medio del método *storeFile*. En caso de no producirse ningún error comienza el entrenamiento y se asigna al objeto el estado *TRAINING* y se lanza un hilo que llama al servicio *Word2VecTrain*. El método *trainFile* nos devolverá el contenido del fichero entrenado y se actualizará la base de datos con el contenido de este fichero entrenado y el identificador WID con el que se llevarán a cabo las predicciones en el servicio REST de la api externa.

Como se pide en el proyecto, este es un proceso asíncrono, se hace en un hilo independiente para que podamos seguir trabajando sobre el hilo principal, dado que es un proceso que tarda

algo de tiempo. En todo momento se muestra al usuario el estado del entrenamiento con los estados *TRAINING* / *TRAINED*.

Por otra parte, cuando se quiere eliminar un entrenamiento, se hará por el método *deleteFile* que proporciona la misma clase. Esta, como en la operación anterior, se apoyará en el controlador y este a su vez en el adaptador para eliminar el fichero de la base de datos.

Rest : Api Externa

La funcionalidad del servicio REST de la api externa tiene una estructura similar al interno, aunque en este caso, se centra en las predicciones. Además, en este punto cobran especial importancia 2 atributos que hemos mencionado en los apartados anteriores: el token del usuario y el WID generado tras entrenar un fichero.

Este servicio puede ser usado de dos formas: Desde el frontend sin autenticación de cliente y desde el tester con autenticación. En el caso de usarlo desde el frontend, se hará por medio del botón w2vu que aparece en las acciones de los ficheros entrenados. En este caso, se manda una petición POST al recurso /w2buse-sin-auth con la palabra introducida en el formulario. Esta es procesada por la clase *PrectionSinAuth*, que lanzará un hilo para realizar la predicción con el predictor *Word2VecUse*, añadirá una nueva predicción a la base de datos del usuario que la lanzó. Este usuario se recupera por medio del controlador y el DAO de usuario implementados. Finalmente, se responderá a la petición con un objeto JSON que incluirá las 10 palabras más cercanas a la dada.

En el caso del acceso por medio del tester se recuperan de la petición el WID y la palabra, que son los parámetros necesarios para la predicción, al igual que en el caso de predicción sin autenticación por medio del frontend. Pero en este caso, debemos formar un token de autenticación para demostrar a la API que un usuario de la organización ha prestado su token y WID a otro usuario. El proceso de predicción es totalmente el mismo, lo único, que antes de comenzar a predecir se comprueba el token que se ha pasado. Para ello se compara el token de la petición con el que formamos nosotros, sabiendo la estructura que este tiene : url de la petición + fecha + token. El proceso a continuación es exactamente como antes, lanzando la predicción en un hilo y mostrando finalmente las 10 palabras más cercanas a la solicitada.

Docker

Con el objetivo de virtualizar la aplicación, se han creado contenedores para los distintos servicios que se ofrecen, los cuales hemos comentado anteriormente y vamos a repasar. Estas configuraciones se corresponden con el fichero *docker-compose*:

- Base de datos MongoDB:

```
mongodb-ssdd:
  image: mongo
  expose:
    - "27017"
  volumes:
    - mongodb_data_container1:/data/db
```

Imagen 17. Configuración de MongoDB en docker-compose.

- Aplicación de flask en python (Frontend):

```
web:
  build: ./app
  ports:
    - 5000:5000
  env_file:
    - ./ssdd.env
  depends_on:
    - mongodb-ssdd
```

Imagen 18. Configuración del frontend en docker-compose.

- Servicio REST interno:

```
internal-api:
  build: ./api-interna
  env_file:
    - ./ssdd.env
  depends_on:
    - mongodb-ssdd
```

Imagen 19. Configuración de la API interna en docker-compose.

- Servicio REST externo (predicción):

```
external-api:
  build: ./api-externa
  env_file:
    - ./ssdd.env
  depends_on:
    - mongodb-ssdd
```

Imagen 20. Configuración de la API externa en docker-compose.

- Servicio GRPC (entrenamientos):

```
grpc-server:
  build: ./grpc
  env_file:
    - ./ssdd.env
  depends_on:
    - mongodb-ssdd
```

Imagen 21. Configuración de GRPC en docker-compose

Como se puede observar, todos los servicios dependen de que se ejecute bien el primero, que es la base de datos. La base de datos y el frontend se lanzan en los puertos 27017 y 5000 respectivamente y todos los servicios tienen en cuenta la variable de entorno que se ha definido en *ssdd.env* para establecer dónde van a correr.

A continuación, vamos a comentar los Dockerfiles de cada uno de los servicios:

- Aplicación de flask en python (Frontend): Establecemos /frontend como directorio de trabajo e instalamos todos los paquetes de python necesarios para el funcionamiento de la aplicación. Estos paquetes están listados en el fichero *requirements.txt*. Con la directiva ENTRYPOINT tratamos el contenedor como un ejecutable.
- Servicio REST interno: Para la aplicación interna, se indica el servidor tomcat en el que va a correr junto a la versión de Java, el puerto 8080 y se indica donde va a ser copiado el war generado por Maven dentro del contenedor.
- Servicio REST externo (predicción): En este caso, el fichero Dockerfile es equivalente al del servicio interno.
- Servicio GRPC (entrenamientos) El fichero Dockerfile contiene la versión de java, se especifica el directorio de trabajo (*/usr/src/app*), el puerto en el que estará el servicio

de entrenamiento (50051), el comando que lo ejecutará (*java -jar*) y se copia al igual que en los servicios anteriores, el target de maven dentro del contenedor.

Manual de usuario

1. Lo primero que tenemos que hacer es clonar el repositorio en nuestra máquina; para ello, hacemos uso de la siguiente orden:

```
$ git clone https://gitlab.com/666ares/ssdd.git
```

2. Nos movemos a la carpeta del repositorio que acabamos de clonar con la siguiente orden:

```
$ cd ssdd
```

3. Ejecutamos el guión '*run.sh*' que se va a encargar de generar los paquetes correspondientes a cada uno de los servicios y poner en marcha la aplicación.

```
# Damos permisos de ejecución si fuese necesario.  
$ chmod +x run.sh  
$ ./run.sh
```

4. Cuando termine la ejecución del guión comentado anteriormente, podremos ver en nuestra terminal los *logs* generados por cada uno de los contenedores *Docker* durante su puesta en marcha. En concreto, debemos fijarnos en los *logs* del contenedor llamado *web_1*, de donde podremos obtener la dirección IP en la que se aloja el servicio Web que nos da acceso a la aplicación. La línea que buscamos es la siguiente (donde '*x.y.z.w*' puede tomar una dirección IP cualquiera):

```
web_1 | * Running on http://x.y.z.w:5000/ (Press CTRL+C to quit)
```

5. Finalmente, accedemos a la dirección IP obtenida en el paso anterior haciendo uso de nuestro navegador.

Una vez hemos accedido, estaremos en la página de inicio con las opciones Iniciar sesión o Registrarse. Si es la primera vez que accedemos deberemos registrarnos con un nombre de usuario y contraseña.

A continuación, se nos llevará a la página principal, donde podremos subir los ficheros, que se irán almacenando en la tabla superior de la derecha. Observamos que para cada fichero que subamos, tenemos las opciones de mandarlo a entrenar o borrarlo. En el caso en el que seleccionemos Entrenar, nos aparecerá en la tabla inferior mostrando su estado. Una vez finalizado el entrenamiento, podremos eliminar este entrenamiento o llamar al sistema de predicción, que nos pedirá la palabra a la cual queremos que sobre la cual hacer la predicción de las 10 palabras más cercanas.

Una vez terminado podremos volver a la pantalla principal. Esta pantalla principal también tiene un botón para cerrar sesión en la parte superior izquierda.