



Universidad
Francisco de Vitoria
UFV Madrid

Universidad Francisco de Vitoria
Escuela Politécnica Superior

Grado en Ingeniería Informática

Desarrollo e Integración de Software

Práctica 2 Convocatoria Extraordinaria

Enrique Aranguren Moliner

Índice

1. Introducción	1
2. Desarrollo	1
2.1. Gestión del desarrollo:	2
2.2. DataModel.java	2
2.3. Backend	4
2.3.1. DataController	5
2.3.2. DataService	5
2.4. Frontend	6
2.4.1. MainView	6
2.4.2. EntityService	8
2.5. Docker	9
2.5.1. Dockerfile Frontend	10
2.5.2. Dockerfile Backend	10
2.5.3. Despliegue de la aplicación	10
3. Gestión de git	12
4. Problemas encontrados	13
4.1. Problemas no solucionados	13
5. Próximos Pasos	14
6. Links:	15

Índice de ilustraciones

Ilustración 1 - Creación del backend	4
Ilustración 2 - Plantilla del frontend	6
Ilustración 3 - MainView - Tab 1.....	7
Ilustración 4 - MainView - Tab 2.....	7
Ilustración 5 - Dockerfile Frontend.....	10
Ilustración 6 - Dockerfile Backend	10
Ilustración 7 - Esquema git	12

1. INTRODUCCIÓN

El objetivo de esta práctica consiste en demostrar la capacidad del estudiante de realizar una aplicación web preparada para producción usando los conocimientos impartidos en la asignatura Desarrollo e Integración del Software.

El desarrollo de esta práctica se realizará utilizando el lenguaje de programación Java (Correto 17) y se puede dividir en tres secciones:

- Desarrollo del backend: se desarrollará una APIREST utilizando el framework Spring. En esta sección demostraremos nuestros conocimientos de gestión de dependencias (Maven), gestión de versiones (git) y pruebas unitarias.
- Desarrollo del frontend: se desarrollará una web utilizando el framework Vaadin. En esta sección demostraremos nuestros conocimientos de gestión de dependencias (Maven) y gestión de versiones (git)
- Contenerización y comunicación: se realizará utilizando el servicio Docker. En esta sección demostraremos nuestros conocimientos de Docker.

2. DESARROLLO

Se ha dividido el desarrollo en cuatro paquetes de trabajo, siendo estos:

- 1 Desarrollo del backend
- 2 Desarrollo del frontend
- 3 Integración front-back
- 4 Docker

Al ser un trabajo en solitario, se ha decidido que estos paquetes de trabajo se realizarán en el orden establecido y no se comenzará el trabajo en el siguiente paquete hasta finalizar el paquete anterior.

El paquete de trabajo Desarrollo del backend consistirá en la creación de un proyecto de backend funcional que lea un archivo json y cuente con métodos para resolver peticiones. Este proyecto, una vez finalizado, será un prototipo sobre el que se continuará el desarrollo en el paquete de trabajo Integración front-back.

El paquete de trabajo desarrollo del frontend pretende crear una estructura frontend que acepte datos del tipo que proporcionará nuestro backend y que construya en base a ellos una web con dos pestañas y un grid en cada una de ellas.

El paquete Integración front-back contempla las modificaciones a los proyectos de backend y frontend para conseguir una comunicación entre ellos, añadiendo y/o modificando funciones como sea necesario. El producto final de este paquete debería ser una webApp funcional.

El último paquete, Docker, consistirá en todas aquellas modificaciones al proyecto funcional necesarias para realizar un despliegue de nuestra aplicación usando el servicio Docker

2.1.GESTIÓN DEL DESARROLLO:

Para esta práctica contamos con dos archivos json, *cp-national-datafile.json* y *mcode_json.json*

Estos ficheros comparten la misma estructura de datos, con la única diferencia siendo que el fichero *mcode_json.json* tiene sus objetos agrupados por el campo *mcode*.

Para manejar estos datos, hemos creado la clase *DataModel.java*, que estará presente tanto en el proyecto Frontend como en el proyecto Backend.

2.2.DATAMODEL.JAVA

Esta clase utiliza las librerías gson y UUID para gestionar la serialización de los campos que forman los objetos DataModel con aquellos que están almacenados en los archivos json.

Este archivo contiene una definición de los campos contenidos, un constructor genérico y otro específico, y getters y setters para todos los campos.

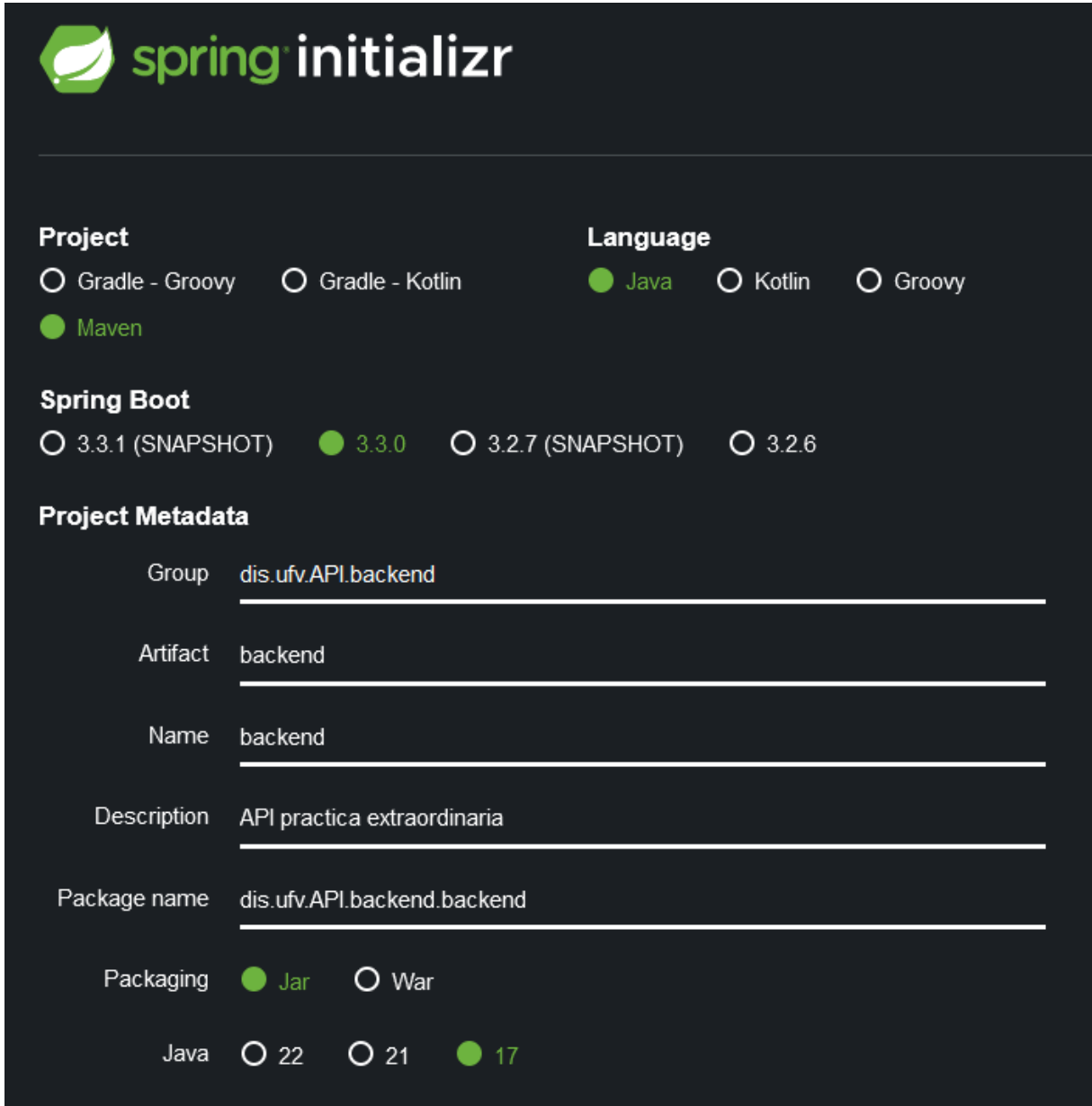
Los campos contenidos en esta clase son:

- String MsCode
- String Year
- String EstCode
- Float Estimate
- Float SE
- Float LowerCIB
- Float UpperCIB
- String Flag
- UUID _id

2.3.BACKEND

La estructura del backend ha sido generada utilizando la página web <https://start.spring.io>

Se ha seleccionado el gestor de dependencias Maven y la versión de Java 17



The screenshot shows the Spring Initializr web form with the following configuration:

- Project:** ☒ Maven, ☐ Gradle - Groovy, ☐ Gradle - Kotlin
- Language:** ☒ Java, ☐ Kotlin, ☐ Groovy
- Spring Boot:** ☐ 3.3.1 (SNAPSHOT), ☒ 3.3.0, ☐ 3.2.7 (SNAPSHOT), ☐ 3.2.6
- Project Metadata:**
 - Group:
 - Artifact:
 - Name:
 - Description:
 - Package name:
 - Packaging: ☒ Jar, ☐ War
 - Java: ☐ 22, ☐ 21, ☒ 17

Ilustración 1 - Creación del backend

El backend cuenta con el archivo de aplicación BackendApplication.java, que contiene el código para la inicialización de la aplicación de Spring, el controlador DataController.java que gestiona las requests recibidas por la API y el gestor de servicios DataServices.java que contiene las funciones requeridas por el controlador para responder a las requests.

Adicionalmente, contamos con un archivo de tests realizados con junit llamado `DataControllerTest`.

2.3.1. DataController

Como se ha mencionado previamente, este archivo gestiona las peticiones recibidas por la API conectándolas con las funciones de `DataService`.

Cuenta con los siguientes métodos:

- `getAllData()` enrutado a `/api/data` | Peticiones get
- `addData(DataModel)` enrutado a `/api/data` | Peticiones post
- `updateData(id)` enrutado a `/api/data/{id}` | Peticiones put
- `deleteData(id)` enrutado a `/api/data/{id}` | Peticiones delete
- `getAllReadOnlyData()` enrutado a `/api/data/readonly` | Peticiones get

2.3.2. DataService

Este archivo contiene las funciones que implementan la lógica de `DataController`. Estas funciones son:

- `getAllData()`

Extrae la información del archivo *`cp-national-datafile.json`*

- `addData(DataModel)` -> `/api/data` | Peticiones post

Añade el nuevo objeto `DataModel` al archivo *`cp-national-datafile.json`*

- `updateData(id)` -> `/api/data/{id}` | Peticiones put

Modifica el objeto cuyo id coincida con el recibido en el archivo *`cp-national-datafile.json`*

- `deleteData(id)` -> `/api/data/{id}` | Peticiones delete

Elimina el objeto cuyo id coincida con el recibido en el archivo *`cp-national-datafile.json`*

- `getAllReadOnlyData()` -> `/api/data/readonly` | Peticiones get

Extrae la información del archivo *`mrcode_json.json`*

2.4.FRONTEND

La estructura básica de nuestro front ha sido creada utilizando las plantillas Hello World de Vaadin 23 <https://vaadin.com/hello-world-starters>



Ilustración 2 - Plantilla del frontend

El frontend contiene la vista MainView y el servicio EntityService

2.4.1. MainView

En este archivo se gestiona la creación de la página web, se definen las pestañas y los grids que habrá en cada una de ellas, así como los botones y los popups para creación, edición y confirmación de eliminación.

Se ha tomado la decisión de diseño de no incluir la columna MsCode en el segundo grid “Filtered Grid” ya que, al realizar un filtrado a partir de este campo, no incluye información nueva.

CRUD Grid Filtered Grid

Create New

MsCode ↕	Year ↕	EstCode ↕	Estima... ↕	SE ↕	Lower ... ↕	Upper... ↕	Flag ↕	Actions
MEASA	2007	_Chang...	0.0	0.0	0.0	0.0	A	Delete
MEASA	2007	_Number	150.8	14.1	136.7	164.9		Delete
MEASA	2007	_Chang...	0.0	0.0	0.0	0.0	A	Delete
MEASA	2008	_Propor...	14.5	1.1	13.4	15.6		Delete
MEASA	2008	_Chang...	0.4	1.8	-1.3	2.2		Delete
MEASA	2008	_Number	155.9	11.7	144.2	167.6		Delete
MEASA	2008	_Chang...	0.0	0.0	0.0	0.0	A	Delete

Ilustración 3 - MainView - Tab 1

CRUD Grid Filtered Grid

Select MsCode

Year ↕	EstCode ↕	Estimate ↕	SE ↕	Lower CIB ↕	Upper CIB ↕	Flag ↕
2007	_Proportion	22.4	2.9	19.5	25.3	
2007	_ChangeInP...	0.0	0.0	0.0	0.0	A
2007	_Number	240.5	31.2	209.3	271.6	
2007	_ChangeIn...	0.0	0.0	0.0	0.0	A
2007	_Population	1073.4	0.0	0.0	0.0	
2008	_Proportion	23.9	2.8	21.2	26.7	
2008	_ChangeInP...	1.5	4.2	-2.7	5.8	
2008	_Number	257.8	29.9	227.9	287.7	
2008	_ChangeIn...	17.3	45.5	-28.2	62.8	
2008	_Population	1076.5	0.0	0.0	0.0	

Ilustración 4 - MainView - Tab 2

2.4.2. EntityService

EntityService maneja las peticiones a la API así como la lógica necesaria para darles forma de manera adecuada. Utiliza RestTemplate para manejar las conexiones http y cuenta con las siguientes funciones:

- findAll()

Realiza una petición a /api/data y devuelve un array de objetos DataModel

- findMs()

Realiza una petición a /api/data/readonly y devuelve un array de objetos DataModel

- save(DataModel)

Genera un _id único y lo añade a DataModel. Manda el objeto DataModel a /api/data

- delete(DataModel)

Extrae el _id de DataModel y lo manda a /api/data/{_id}

Adicionalmente hemos añadido una última petición que ha acabado siendo innecesaria pero que, a vista futura, podría ser útil:

- findById(id)

Realiza una petición get a api/data/{id}, espera recibir un objeto DataModel cuyo id se corresponda con el enviado

2.5.DOCKER

Docker es una plataforma de software que permite crear, probar e implementar de forma rápida y segura aplicaciones en contenedores estandarizados.

Esta tecnología utiliza el kernel de Linux y sus funciones para dividir los procesos y ejecutarlos de manera independiente. Esto permite que varias aplicaciones funcionen en diferentes entornos complejos dentro de una misma máquina host.

Los contenedores de Docker podrían considerarse máquinas virtuales muy livianas y modulares lo que te permite un cierto grado de flexibilidad a la hora de crear, implementar, copiar y mover de un entorno a otro tus aplicaciones.

El modelo de implementación de Docker está basado en imágenes, lo que permite compartir fácilmente una aplicación o un conjunto de servicios, con todas sus dependencias, en varios entornos.

A la hora de implementar Docker en nuestro proyecto, hemos tenido que definir primero varios parámetros debido a la necesidad de conectar los contenedores del Backend y del Frontend.

Para conectar dos contenedores, dependemos de redes de Docker (Docker network), que permiten a los contenedores visualizarse entre si. Nuestro frontend, que hasta este momento realizaba peticiones a `localhost:2223/api/data`, debe ser modificado de manera que realice las peticiones al contenedor de nuestro backend.

Por tanto, y para asegurarnos de que siempre se mantenga el mismo nombre, fijaremos los nombres tanto de la red como de los contenedores de backend y frontend.

- Red Docker: `dis`
- Contenedor backend: `backend`
- Contenedor frontend: `frontend`

Conociendo ya la arquitectura de nuestro proyecto de Docker, modificaremos la url base de las peticiones realizadas por nuestro frontend de `http://localhost:2223/api/data` a `http://backend:2223/api/data`

Teniendo ya bien configuradas las peticiones, utilizaremos la herramienta `install` de Maven para generar un `fat-jar` de cada proyecto y procederemos a construir el `Dockerfile`.

2.5.1. Dockerfile Frontend

Este archivo, que muestra la construcción de nuestra imagen para el frontend, es muy sencillo. Construye un entorno de java 17 (amazoncorretto 17) y en él, copia el archivo fat-jar generado. Una vez creado un contenedor a partir de la imagen, se ejecutará automáticamente las instrucciones definidas en su Entrypoint, comenzando el servicio de Frontend.

```
FROM amazoncorretto:17

COPY target/spring-skeleton-1.0.jar spring-skeleton-1.0.jar

EXPOSE 2222

ENTRYPOINT ["java", "-jar", "/spring-skeleton-1.0.jar"]
```

Ilustración 5 - Dockerfile Frontend

2.5.2. Dockerfile Backend

Como el Dockerfile anterior, este archivo es muy sencillo, aunque se han creado ciertas modificaciones para agilizar y simplificar su despliegue. En vez de crear un contenedor, lo cual resultaría en la necesidad de tener una arquitectura de carpetas en el proyecto fija, se ha decidido copiar los archivos json a la imagen, además del fat-jar generado con Maven, simulando la estructura de carpetas con la que ya cuenta el proyecto.

Una vez creado un contenedor en base a esta imagen, se ejecutará el comando especificado en su Entrypoint, dando comienzo el servicio de Backend

```
FROM amazoncorretto:17

COPY target/backend-1.0.jar target/backend-1.0.jar
COPY src/main/resources/cp-national-datafile.json src/main/resources/cp-national-datafile.json
COPY src/main/resources/MsCode_json.json src/main/resources/MsCode_json.json

EXPOSE 2223

ENTRYPOINT ["java", "-jar", "/target/backend-1.0.jar"]
```

Ilustración 6 - Dockerfile Backend

2.5.3. Despliegue de la aplicación

Para facilitar el despliegue, primero se intentó guardar las imágenes del proyecto en archivos .jar para facilitar luego su carga sin necesidad de reconstruir las imágenes, pero al ser estos

archivos demasiado grandes para github (~500Mb) se tomo la decisión de subirse las imágenes del frontend y del backend al repositorio de imágenes de Docker.

Teniendo esto en cuenta, a continuación mostramos las instrucciones necesarias para comprobar el funcionamiento del proyecto así como una breve explicación de su funcionamiento:

Primero se creará la red de Docker con el comando:

```
docker network create dis
```

Segundo, se descargará la imagen del backend del repositorio con el comando

```
docker pull enry24/dis-backend
```

Se levantará el contenedor con el nombre especificado y lo añadimos a la red creada anteriormente. También creamos una conexión entre el puerto 2223 interno del Docker con el puerto 2223 de nuestro equipo. Esto permitirá comprobar el funcionamiento de nuestro backend navegando a *localhost:2223/api/data*

```
docker run -d -p 2223:2223 --network dis --name backend enry24/dis-backend
```

Se descargará la imagen del frontend del repositorio

```
docker pull enry24/dis-frontend
```

Se levantará el contenedor con el nombre especificado, creando una conexión entre el puerto 2222 interno del Docker con el puerto 2222 de nuestro equipo, lo que permitirá visualizar la página web navegando a *localhost:2222*

```
docker run -d -p 2222:2222 --network dis --name frontend enry24/dis-frontend
```

Habiendo introducido estos cinco comandos en el orden especificado, la página debería estar visible en *localhost:2222*

3. GESTIÓN DE GIT

Se ha empleado la herramienta git para el control de versiones del proyecto. Al tratarse de un desarrollo individual no se ha podido explotar al máximo las capacidades colaborativas que ofrece git, si bien aun así ha sido una herramienta útil en el desarrollo de este proyecto.

Se han definido las siguientes pautas:

- El nombre de cada uno de los commits debe ser en inglés
- Los comentarios serán opcionales, pero deberán estar en inglés y serán usados como recordatorio del estado del proyecto en ese momento

Se decidió desde un primer momento que solo se usaría la rama master para releases o cambios menores (como añadir un archivo readme específico para la reléase). Trabajaremos por tanto principalmente sobre la rama develop y cada uno de los paquetes de trabajo definidos en Desarrollo tendrán una rama propia para su implementación.

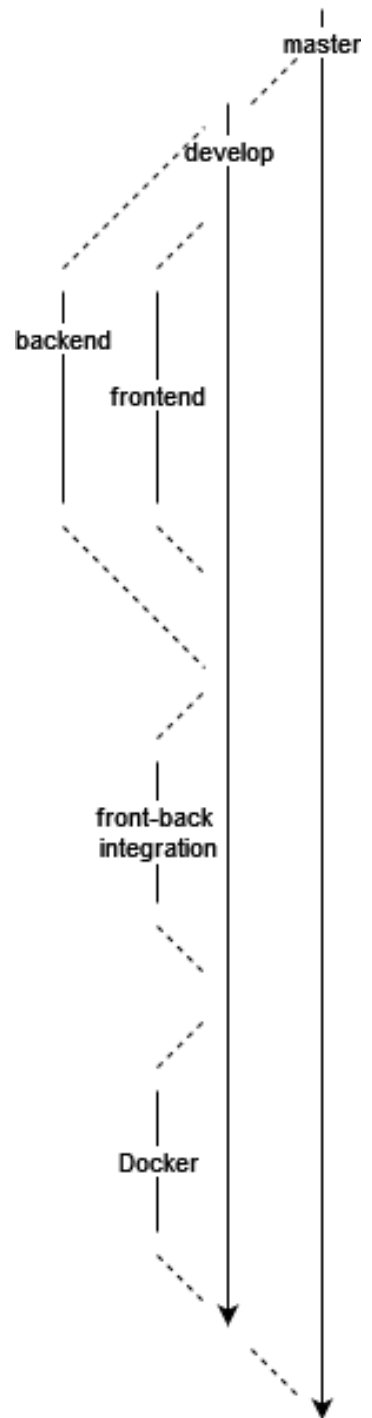


Ilustración 7 - Esquema git

4. PROBLEMAS ENCONTRADOS

Durante el desarrollo hemos encontrado diversos problemas, especialmente en la conexión entre backend y frontend. La inmensa mayoría de ellos se han debido a fallos humanos al especificar el tipo de dato a mandar/recibir.

En el diseño del frontend ha habido algún problema que otro, principalmente con los popups de edición y creación, y la ubicación de los botones.

En el diseño del backend hemos tenido problemas con la definición de la clase DataModel y sus constructores, estos problemas desaparecieron una vez implementamos un constructor con todos los campos y un constructor vacío.

Finalmente, en la parte de Docker, nos hemos encontrado con algún problema conectando los contenedores entre sí, asumiendo en un primer momento que, si la API es visible desde localhost, debería ser visible también al contenedor Frontend. Esto es incorrecto y la solución implementada utilizando redes de docker puede verse en el apartado de Docker. Otra solución posible podría haber sido usando un archivo docker-compose para el contenedor frontend, en el que especificásemos tanto la conexión 2222:2222 para visualizar la web, como la conexión 2223:2223 para realizar las peticiones.

4.1. PROBLEMAS NO SOLUCIONADOS

Hemos identificado dos problemas con nuestra WebApp que no hemos sido capaces de solucionar:

- En una misma sesión, definiendo sesión como actividad en la página web sin tener que refrescar la página, crear una nueva entrada en el grid CRUD y eliminar esa misma entrada. Es posible eliminar entradas creadas en sesiones anteriores o entradas que ya pertenecían al set de datos iniciales.
- Al cambiar de pestaña a Filtered Grid y volver luego a CRUD Grid, los campos de este son ocasionalmente duplicados. Este error no sucede cada vez que se cambia y debido a su dificultad en replicarlo de manera consistente no hemos sido capaces de determinar su origen

5. PRÓXIMOS PASOS

Este proyecto podría avanzar en múltiples direcciones. Primero, por supuesto, se debería encontrar una solución a los problemas no solucionados presentados con anterioridad.

Además de la solución de estos problemas, una posible vía para mejorar el proyecto sería integrar la primera práctica (creación del archivo `mcode_json.json`) y operaciones CRUD en el segundo grid, de manera que se reflejasen las operaciones CRUD del primer grid en el archivo `json mcode_json.json` y viceversa.

También se debería implementar un log en el que se almacenase un historial de operaciones, así como un backup y una opción para restaurar los archivos json a un estado anterior, en caso de fallo catastrófico o eliminación accidental de datos.

Otra posible dirección futura sería añadirle un estilo a la página mediante el uso de un archivo css personalizado.

6. LINKS:

Repositorio de github propio con la práctica: https://github.com/Enry2410/recuperacion_dis

Repositorio de Docker con el front: <https://hub.docker.com/r/enry24/dis-frontend>

Repositorio de Docker con el back: <https://hub.docker.com/r/enry24/dis-backend>