Ikaskuntza Birtual eta Digitalizatuen LHII
CIFP de Aprendizajes Virtuales y Digitalizados

DESPLIEGUE DE APLIACIONES WEB

UD05

Tarea Evaluativa 03: OTROS ASPECTOS

Curso: 2023/24

Nombre: Jesus Diego Rivero

EUSKO JAURLARITZA GOBIERNO VASCO
HEZKUNTZA SAILA
Lanbide Heziketako Sailburuordetza
DEPARTAMENTO DE EDUCACIÓN
Viceconsejería de Formación Profesional

ÍNDICE

1. EJERCICIO

En esta tarea se plantean una serie de aspectos relacionados con el despliegue de aplicaciones web desarrolladas en Java que no se han tratado todavía. Se pide **elegir al menos uno de ellos y desarrollarlo**.

Las tareas que se proponen son las siguientes pero si alguien está interesado/a en otro tema, lo puede proponer por correo.

1. Desplegar las aplicaciones de ejemplo (alguna de las que usan BBDD) en un **servidor JBoss/Wildfly** mediante su interfaz web. Podéis consultar **aquí** cómo hacerlo.
2. Análisis y despliegue de una servicio web Java y su cliente JavaScript sobre una pila multicontenedor Docker. Podemos utilizar el ejemplo descrito en el siguiente **enlace**.
3. Despliegue de contenedores docker en la nube usando herramientas como Microsoft Azure o Amazon Web Services (AWS).

Se documentará el trabajo de I+D desarrollado:

- **Análisis teórico** (estructura del proyecto/aplicación, servicios...). En este apartado se pretende que realicéis un análisis y descripción del proyecto. En el caso del primer y tercer punto, describiréis la estructura del sistema que vais a montar y cómo se comunican los servicios entre ellos, así como particularidades de cada tecnología en el contexto del ejercicio (JBoss/Wildfly o Azure/AWS). En la segunda opción, deberéis describir la estructura del proyecto escogido, cómo funciona el flujo de datos... Conciso y con información relevante.
- **Desarrollo**. En cada apartado se describirá con información gráfica cómo se ha realizado el proceso de desarrollo (no demasiado extenso, conciso y con información relevante).
- **Despliegue**. En cada apartado se describirá el despliegue realizado y la aplicación en marcha (podría no estar funcional al 100%). Se aportarán pruebas gráficas.
- **Dificultades** encontradas en el desarrollo y despliegue.

y se entregará junto con el producto obtenido aunque no haya funcionado del todo.

a introducir un email correcto.

2. Análisis y despliegue de una servicio web Java y su cliente JavaScript sobre una pila multicontenedor Docker

2.1. ANALISIS TEORICO

2.1.1. Estructura del Proyecto

Aplicación Angular: Implementada en Angular 17 para el frontend.

API REST Spring Boot: Implementada en Spring Boot para el backend.

Base de Datos: la BBDD es mysql

Contenedores Docker: Se han hecho contenedores de la BBDD, de la api desarrollada en Spring-boot y de la aplicación de angular

2.1.2. Flujo de Datos

El cliente Angular realiza solicitudes HTTP al backend Spring Boot.

El backend Spring Boot procesa estas solicitudes, interactúa con la base de datos si es necesario y devuelve respuestas.

El cliente Angular recibe las respuestas del backend y actualiza la interfaz de usuario según corresponda.

Tecnologías Utilizadas

Angular: Para el desarrollo del cliente.

Spring Boot: Para el desarrollo del backend.

MySQL: Como base de datos.

Docker: Para contenerizar las aplicaciones.

2.2. DESARROLLO

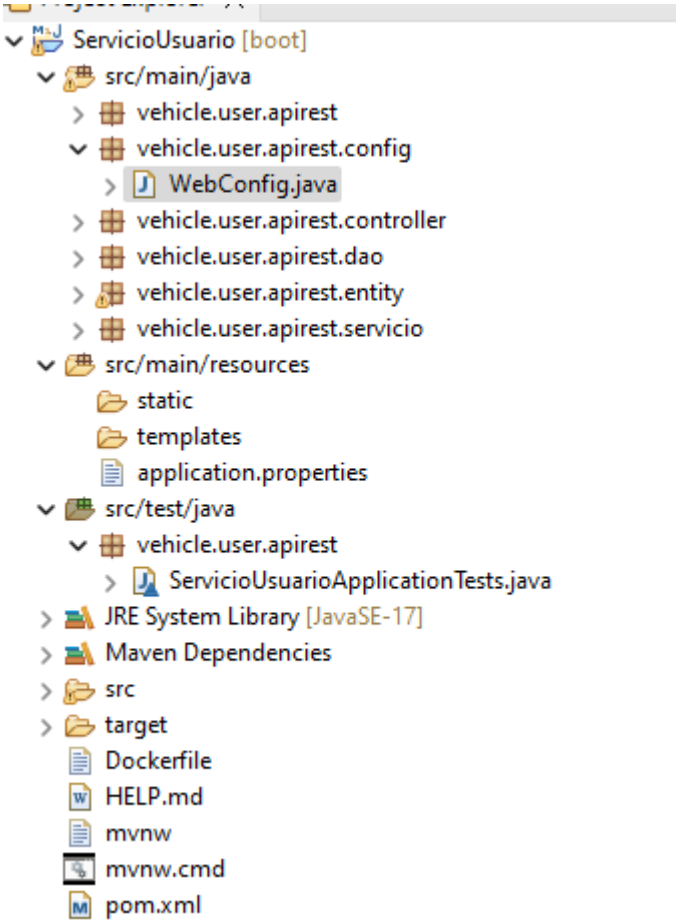
BBDD

He creado un script con la creación de una BBDD de usuarios muy simple. En este script se crea una tabla que contiene una serie de usuarios con 3 campos cada uno: id, nombre y departamento.

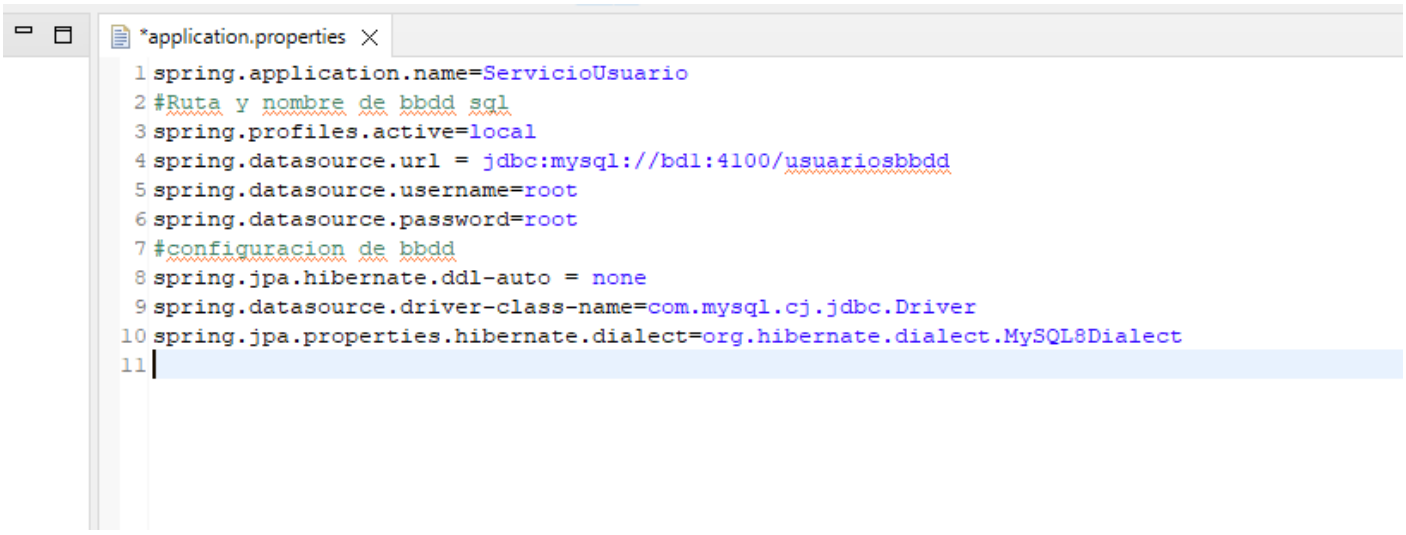
API REST

En el IDE eclipse he creado una api rest con el framework de spring-boot. Esta api hace un CRUD a la bbdd mysql

Se ha realizado la api en la arquitectura MVC:



Se ha configurado el acceso a la bbdd en el archivo application.porperties como se ve en la imagen siguiente:



Lo más destacado de esta configuración es que la ruta de acceso a la BBDD se realiza sobre el nombre y puerto del contenedor donde esta la bbdd. Además, ha sido necesario configurar las propiedades siguientes:

spring.jpa.hibernate.ddl-auto:none,

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

para que me permita generar el archivo compilado sin estar conectado a la BBDD. Este archivo compilado es el que luego se utiliza para generar el contenedor de spring-boot.

También se ha tenido que generar un archivo configuración denominado WebConfig para resolver los problemas por el CORS policy que se da cuando se realizan solicitudes desde otro dominio.

```java
package vehicle.user.apirest.config;
// calse de configuración para que permita solicitudes desde otro dominio (policy CORS)
import org.springframework.context.annotation.Configuration;

@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/api/**") // Establece el patrón de URL se permite
                .allowedOrigins("*") // Permite solicitudes desde cualquier origen
                .allowedMethods("GET", "POST", "PUT", "DELETE") // Permite los métodos HTTP especificados
                .allowCredentials(true); // Permite enviar credenciales (cookies) en la solicitud
    }
}
```

El dockerfile se ubica en la raíz de la aplicación.

Estructura del Proyecto:

- src/main/java para el código fuente.
- src/main/resources para archivos de configuración (application.properties).

Desarrollo de Funcionalizades:

- Implementa controladores REST para manejar las solicitudes HTTP entrantes.
- Define modelos de datos y realiza operaciones CRUD si es necesario.

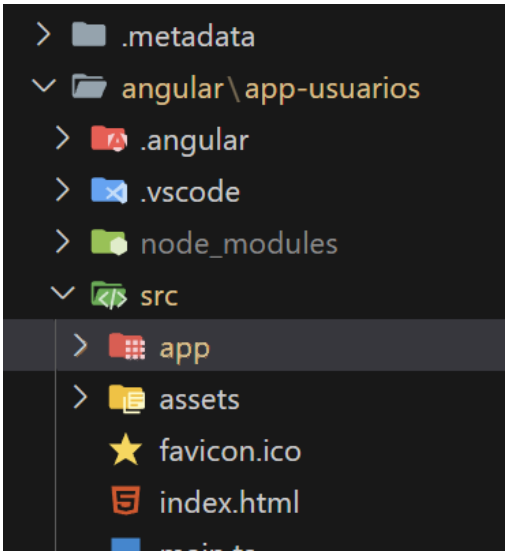
Configuración de Docker

Dockerfile para API Java:

```
# Use the official OpenJDK image from the Docker Hub
FROM openjdk:17-jdk-slim

# Set the working directory inside the container
WORKDIR /app



# Copy the Spring Boot application's JAR file into the container
COPY ./target/ServicioUsuario-0.0.1-SNAPSHOT.jar /app/ServicioUsuario.jar
EXPOSE 8080
# Specify the command to run the application
ENTRYPOINT ["java", "-jar", "/app/ServicioUsuario.jar"]
```

APP CLIENTE

Se ha desarrollado la aplicación en Angular 17.



La aplicación realizar peticiones http sobre la api. En este caso únicamente se han desarrollado las peticiones GET.

Para generar el contenedor se ha utilizado el servidor web nginx que gestiona las solicitudes.

La generación del contenedor se realiza realizando un build sobre la aplicación angular, copiando todos los archivos de configuración y llevándolo todo al servidor web donde se despliega.

El dockerfile se ubica en la raíz de la aplicación.

Estructura del Proyecto:

- src/app para los componentes, servicios y otros recursos de Angular.
- angular.json para la configuración del proyecto.

Desarrollo de Funcionalidades

- Implementa componentes para las distintas partes de tu aplicación.
- Utiliza servicios para manejar las solicitudes HTTP al backend.

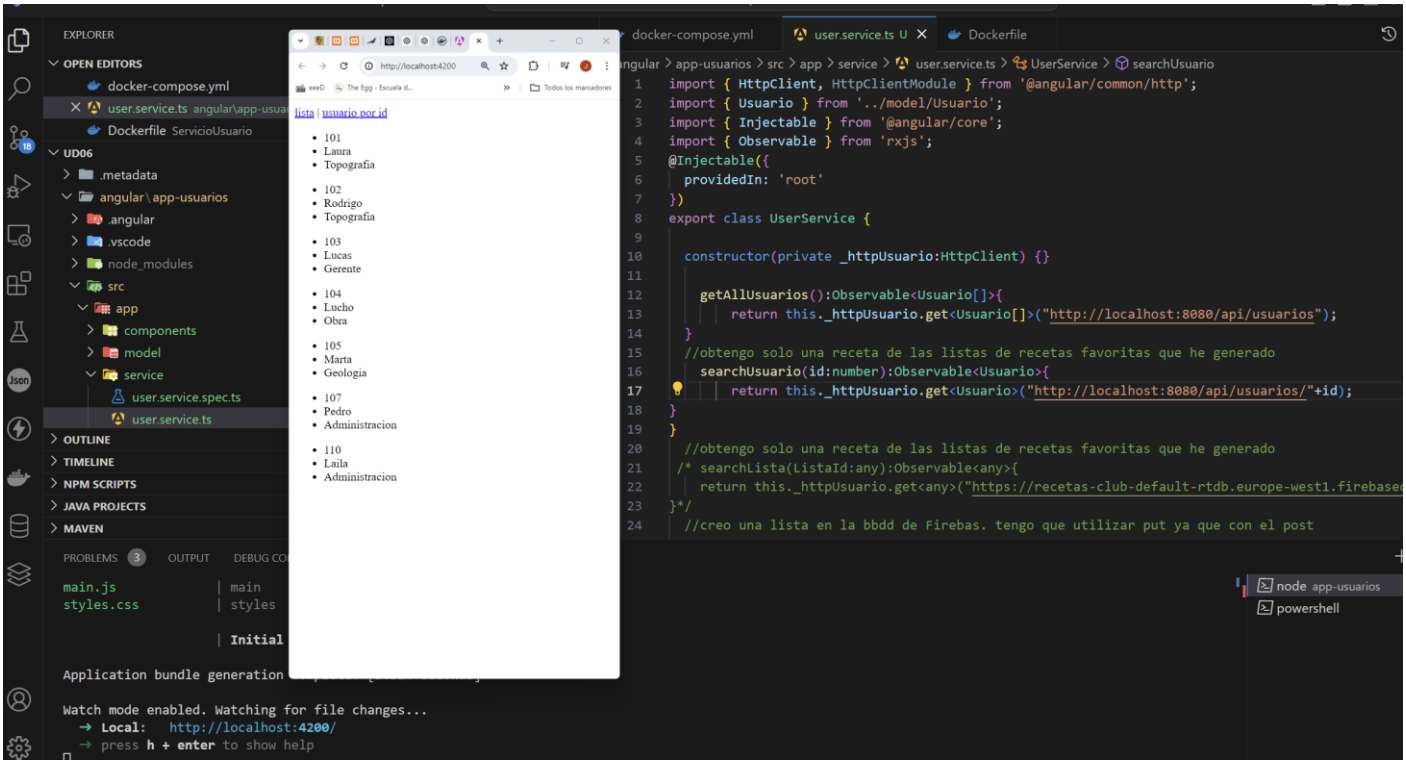
Configuración de Docker

Dockerfile para Angular:

```
angular > app-usuarios > Dockerfile > ...
1   # Stage 1: Build  Angular application usando the production config
2   FROM node:latest as build-stage
3   # establecer el directorio de trabajo
4   WORKDIR /app
5   #copiar los paquetes json de la app angular
6   COPY package*.json ./
7   # ejecutar una instalación limpia de las dependencias
8   RUN npm ci
9   # instalar Angular CLI global
10  RUN npm install -g @angular/cli
11  # copiar todos los ficheros
12  COPY . .
13  # hacer un build de la aplicacion
14  RUN npm run build --configuration=production
15
16  # Stage 2: uso aplicacion nginx
17  FROM nginx:latest
18
19  # copiar el archivo de configuracion de nginx
20  COPY ./nginx.conf /etc/nginx/conf.d/default.conf
21
22  # Copio el build de angular en nginx
23  COPY --from=build-stage /app/dist/app-usuarios/browser /usr/share/nginx/html
24
25  # Expongo puerto 80 (Nginx lo usa por defecto)
26  EXPOSE 80
27
28
```

2.3. Despliegue

Se han construido los contenedores a partir de los dockerfile generados. Todos los contenedores comparten una misma red. Antes del despliegue se ha probado que el flujo funcionaba correctamente con las aplicaciones sin estar en contenedores:



Posteriormente, se ha ejcutado el Docker-compose:

Parte de Docker-compose relativa a la bbdd:

```
version: '3'
networks:
    # red que comparte servicio
    mysql-java:
      driver: bridge
    java-angular:
      driver: bridge
services:
  db:
    build: ./bbdd
    container_name: bd
    environment:
      - MYSQL_ROOT_PASSWORD= root
    ports:
      - "4100:3306"
    networks:
      - mysql-java
  backend:
```

Parte de Docker-compose relativa a la api java:

```
      - mysql-java
  backend:
    build: ./ServicioUsuario
    container_name: apirest
    ports:
      - "4101:8080"
    depends_on:
      - db
    networks:
      - mysql-java
      - java-angular
  frontend:
```

Parte de Docker-compose relativa a angular:

```
  db:
    networks:
  backend:
    build: ./ServicioUsuario
    container_name: apirest
    ports:
      - "4101:8080"
    depends_on:
      - db
    networks:
      - mysql-java
      - java-angular
  frontend:
    build: ./angular/app-usuarios
    container_name: angular
    ports:
      - "8082:80"
    depends_on:
      - db
      - backend
    networks:
      - java-angular

```

2.4. Dificultades

La mayor dificultad y que ha impedido que la app funcione es la conexión entre los diferentes componentes. Sin contenerizar sí que funcionaba bien pero al hacer los contenedores la conexión entre los diferentes componentes se rompía. He llegado a lograr la conexión entre la BBDD y la api ambos como contenedor pero al hacer cambios para que también conectara la parte del frontend he perdido también esa conexión y no la he logrado recuperar. He llegado a ver que dentro del contexto del contendor cada componente había que identificarlo con el nombre del contenedor, al menos con la BBDD y la apirest así sucedía aunque la app de angular no he logrado que establezca conexión con la api en ningún momento mientras estaba en contenedor.