

Ejecución mediante aplicación Docker

Prerequisitos:

- Clonar el repositorio del proyecto
- Tener instalado la aplicación Docker Desktop

1. Crear archivo de variables de entorno. env para el frontend y el backend

a. Variables frontend

```
1
2  VITE_HOST=0.0.0.0
3  VITE_PORT=3000
4  VITE_BACKEND_URL="http://localhost:4000"
5
```

VITE_HOST=0.0.0.0 significa que el servidor Vite estará escuchando en todas las interfaces de red de la máquina. VITE_PORT=3000 significa que el servidor Vite estará escuchando en el puerto 3000, puede cambiar este puerto por otro si su máquina tiene este puerto ocupado.

La variable VITE_BACKEND_URL se utiliza para especificar la URL del servidor backend. En este caso, el servidor backend está corriendo en <http://localhost:4000>, debe ser cambiado a la dirección IP de su máquina.

b. Variables Backend

```
1  DB_NAME="test"
2
3  # DB
4  DB_USER="manager_test"
5  DB_PASSWORD="RqoKdtp88Z94v7vL#XKVHPxwdb9dw"
6  DB_HOST="iniciativaser-db_test-1"
7  DB_PORT=3306
8  DB_DRIVER="mariadb"
9  JWT_SECRET_ADM="aAJSFKFK_@123DNMFFesi;"
10 JWT_SECRET="aKASUFN2k123h1ad@dmkle.;kn123FGB"
11
12 # configuración cors
13 ORIGIN="http://localhost:3000"
14
15
16 # configuración server
17 IP="0.0.0.0"
18 PORT=4000
```

Las primeras seis variables (DB_NAME, DB_USER, DB_PASSWORD, DB_HOST, DB_PORT, DB_DRIVER) están relacionadas con la configuración de la base de datos. Estas variables se utilizan para definir el nombre de la base de datos (DB_NAME), el usuario (DB_USER), la contraseña (DB_PASSWORD), el host (DB_HOST), el puerto (DB_PORT) y el controlador de la base de datos (DB_DRIVER) que tu aplicación utilizará para conectarse a la base de datos.

La variable JWT_SECRET es utilizada para firmar y verificar los tokens JWT en tu aplicación. JWT significa "JSON Web Token", que es un estándar de la industria para la creación de tokens de acceso que permiten la autenticación de usuarios y la transmisión segura de información.

La variable ORIGIN se utiliza para configurar el CORS (Cross-Origin Resource Sharing) en tu aplicación. CORS es una política de seguridad que permite o restringe las solicitudes de recursos a tu servidor desde un origen diferente.

Finalmente, las variables IP y PORT se utilizan para configurar la dirección IP y el puerto en el que tu servidor estará escuchando las solicitudes entrantes.

2. Crear los archivos "dockerfile" en la carpeta backend y frontend, y el archivo "docker-compose.yml" en la carpeta principal, ajuste estos archivos según los puertos usados en sus archivos .env

a. Dockerfile Backend

```
FROM node:14

WORKDIR /app

COPY Backend/package*.json ./

RUN npm install

COPY Backend ./

EXPOSE 3000

CMD [ "npm", "start" ]
```

Este archivo define cómo se construye una imagen del backend, utilizando Node.js en un contenedor docker.

FROM node:14: Utiliza la imagen de Node.js versión 14 como la imagen base para el contenedor.

WORKDIR /app: Establece el directorio de trabajo dentro del contenedor como /app.

COPY Backend/package.json ./: Copia el archivo package.json y package-lock.json desde el directorio Backend del host al directorio de trabajo /app dentro del contenedor.

RUN npm install: Instala las dependencias del proyecto utilizando npm dentro del contenedor.

COPY Backend ./: Copia todo el código restante del backend desde el directorio Backend del host al directorio de trabajo /app dentro del contenedor.

EXPOSE 3000: Expone el puerto 3000, que se utiliza para el servicio del backend.

CMD ["npm", "start"]: Define el comando que se ejecutará cuando se inicie el contenedor. En este caso, ejecuta el comando npm start

b. Dockerfile Frontend

```
1 FROM node:16-bullseye
2
3 WORKDIR /app
4
5 COPY package.json package-lock.json /app/
6
7 RUN npm install
8
9 COPY . /app
10
11 EXPOSE 3000
12
13 CMD ["npm", "run", "dev"]
```

Este archivo define cómo se construye una imagen para el frontend, utilizando Node.js en un contenedor Docker.

FROM node:16-bullseye: Utiliza la imagen oficial de Node.js versión 16 para el contenedor.

WORKDIR /app: Establece el directorio de trabajo dentro del contenedor como /app.

COPY package.json package-lock.json /app/: Copia los archivos package.json y package-lock.json desde el host al directorio de trabajo /app dentro del contenedor.

RUN npm install: Instala las dependencias del proyecto utilizando npm.

COPY . /app: Copia todo el código fuente del frontend desde el host al directorio de trabajo /app

EXPOSE 3000: Expone el puerto 3000

CMD ["npm", "run", "dev"]: Define el comando que se ejecutará cuando se inicie el contenedor. En este caso, ejecuta el script "npm run dev".

c. docker-compose.yml

```
version: '3'
services:
  db_test:
    image: mariadb
    hostname: test
    environment:
      MYSQL_ROOT_PASSWORD: pf$6#iAEQ#@y7vMnXiTXLLC#wiEHcn
      MYSQL_DATABASE: test
      MYSQL_USER: manager_test
      MYSQL_PASSWORD: RqoKdtp88Z94v7vL#XKVHPxWdb9dw
    volumes:
      - ./Base de datos/test.sql:/docker-entrypoint-initdb.d/init.sql
      - data:/var/lib/mysql
    ports:
      - 3306:3306
    restart: always
    networks:
      - red

  backend:
    build:
      context: .
      dockerfile: ./Backend/dockerfile
    ports:
      - 4000:4000
    depends_on:
      - db_test
    networks:
      - red
    restart: always
```

```
  frontend:
    build:
      context: ./Frontend
      dockerfile: dockerfile
    ports:
      - 3000:3000
    networks:
      - red

volumes:
  data:
    external: false

networks:
  red:
    name: red
```

Este archivo define tres servicios que trabajarán juntos para crear un entorno de desarrollo completo:

db_test: Este servicio utiliza la imagen de MariaDB como base para crear un contenedor de base de datos. Se configuran varias variables de entorno, como la contraseña del usuario root, la base de datos, el nombre de usuario y la contraseña. Además, se monta un volumen para inicializar la base de datos con un archivo SQL proporcionado (test.sql). Este servicio se reiniciará automáticamente (restart: always) y se conectará a una red llamada "red".

backend: Este servicio construye una imagen utilizando el Dockerfile ubicado en el directorio Backend. También expone el puerto 4000 y depende del servicio db_test. Se reinicia automáticamente (restart: always) y se conecta a la misma red que db_test.

frontend: Este servicio construye una imagen utilizando el Dockerfile ubicado en el directorio Frontend. Expone el puerto 3000 y no tiene dependencias explícitas. Se conecta a la red "red".

Además de los servicios, el archivo de docker-compose define dos volúmenes y una red:

Volumen "data": Este volumen no es externo (external: false) y se utiliza para almacenar los datos de la base de datos MariaDB.

Red "red": Esta red se utiliza para conectar todos los servicios entre sí, permitiendo que se comuniquen entre sí.

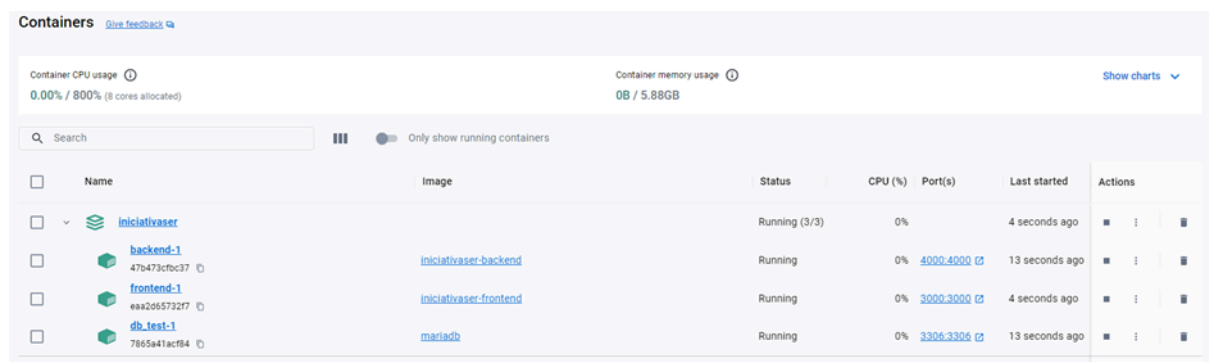
3. Para ejecutar el proyecto, primero ejecutamos o abrimos la aplicación Docker Desktop. Para que la página se suba en la aplicación, es necesario ejecutar el comando `docker-compose up` en la carpeta del proyecto.

```
IniciativaSer> docker-compose up
```

O bien, se puede hacer directamente desde la aplicación Visual Studio Code haciendo click derecho en el archivo, seleccionando la opción *Compose Up*:



Una vez levantado el proyecto, en Docker Desktop debería verse los siguientes elementos:



Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
iniciativaser		Running (3/3)	0%		4 seconds ago	
backend-1 47b473cfc37	iniciativaser:backend	Running	0%	4000:4000	13 seconds ago	
frontend-1 ea2065732f7	iniciativaser:frontend	Running	0%	3000:3000	4 seconds ago	
db_test-1 7865e41act94	mysql:db	Running	0%	3306:3306	13 seconds ago	

Esto significa que las imágenes se subieron correctamente y están en ejecución.

Finalmente, el proyecto se ejecutará en la ruta local definida `http://localhost:3000/`

