“Universidad Nacional de San Agustín”

Facultad de Ingeniería, Producción y Servicios

Escuela Profesional de Ciencia de la Computación

UNSA

UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

CLOUD COMPUTING

TRABAJO DE CONTENEDORES

Estudiante:

Ruiz Mamani, Eduardo German

Arequipa - Perú

2025

1. REPOSITORIO

https://github.com/EGRM23/CC_contenedores

2. CONFIGURACIÓN DE CONTENEDORES

2.1. ESTRUCTURA DEL PROYECTO

2.1.1. Contenedor frontend

```
FROM nginx:alpine
COPY . /usr/share/nginx/html
EXPOSE 80
```

Para el contenedor del frontend se usó la imagen de nginx:alpine y adicional a esto se usó un archivo html base.

2.1.2. Contenedor backend

```
FROM node:18
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["node", "server.js"]
```

Para el contenedor del backend se usó la imagen de node.js y se instaló las dependencias necesarias, adicional a eso se integró dentro del contenedor un archivo *server.js* que contenía la lógica del negocio que usará el trabajo

2.1.3. Contenedor database

```
FROM mysql:8.0
ENV MYSQL_ROOT_PASSWORD=rootpassword
ENV MYSQL_DATABASE=testdb
COPY init.sql /docker-entrypoint-initdb.d/
EXPOSE 3306
```

Para el contenedor de la database se usó de base la imagen de mysql y se declararon las variables necesarias para acceder a ella, se usó un archivo init.sql que contenía la creación de una tabla base con unos cuantos registros por defecto.

3. INTERACCIÓN ENTRE CONTENEDORES

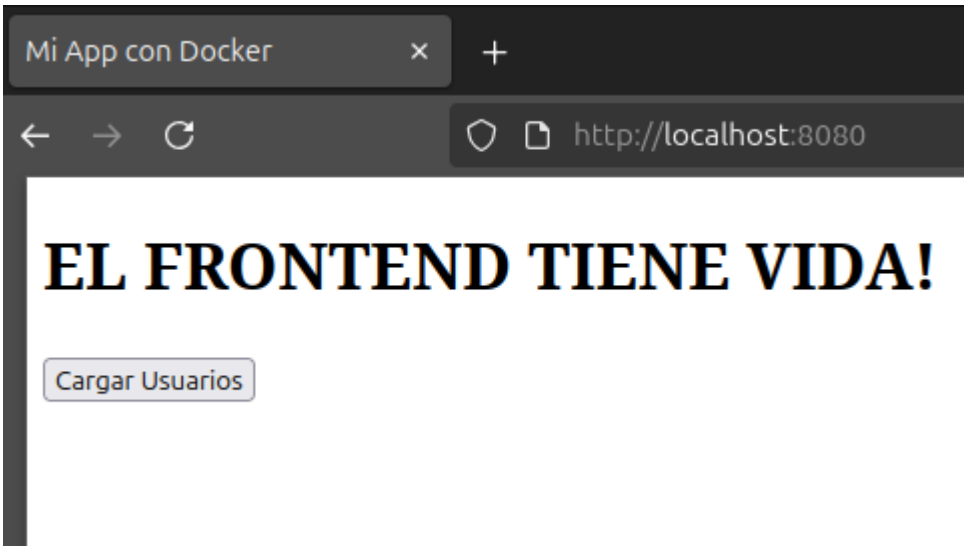
La interacción entre los contenedores se da a través del archivo docker-compose.yml donde se especifica cómo interactúan los contenedores entre sí, el código es el siguiente:

```
services:
 frontend:
   build: ./frontend
   ports:
     - "8080:80"
   depends_on:
     - backend

 backend:
   build: ./backend
   volumes:
   - ./backend:/app
   - /app/node_modules
   ports:
     - "3000:3000"
   depends_on:
     - db

 db:
   build: ./db
   environment:
     MYSQL_ROOT_PASSWORD: rootpassword
     MYSQL_DATABASE: testdb
   ports:
     - "3307:3306"
```

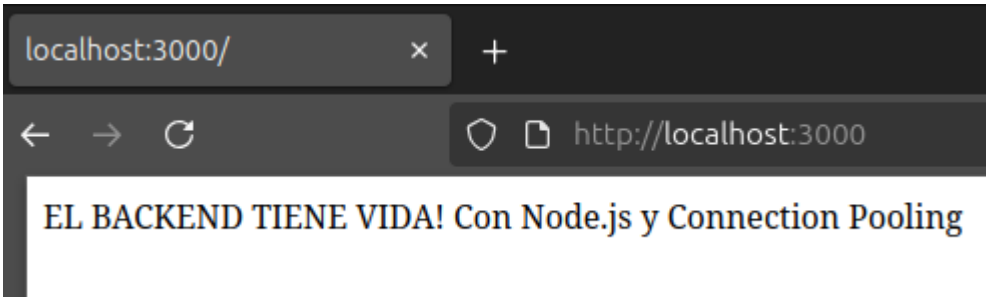
Como se ve son 3 servicios los que son parte del proyecto, en cada uno se especifica la dirección del container y se redirecciona los puertos que los servicios tienen dentro del container a otros dentro de la PC (para poder acceder a los servicios). Adicionalmente el comando depends_on significa que los servicios deben esperar a que se inicie otro servicio para que ellos puedan iniciar, como el caso del frontend que para iniciar debe esperar que antes inicie el backend.

4. FLUJO DE TRABAJO

El flujo de trabajo que sigue la aplicación es el siguiente, se comienza en la ventana del frontend, donde el usuario accede con la dirección http://localhost:8080, y se muestra la siguiente web:



Al hacer click en el botón *Cargar Usuarios* este se conecta con una función definida dentro del código donde redirecciona al servicio del backend en http://localhost:3000/:

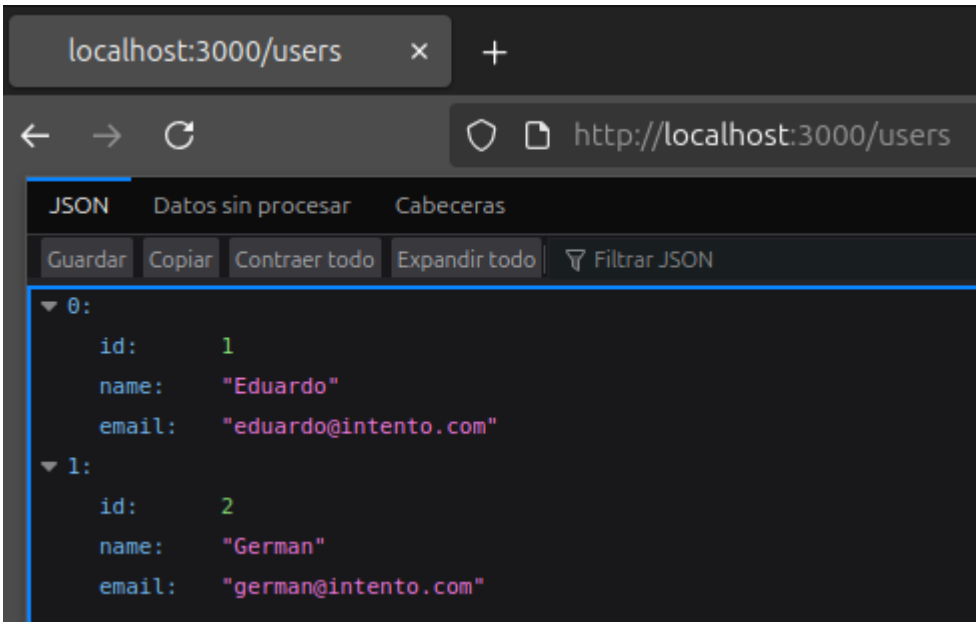


Aunque esto solo demuestra que el servicio está funcionando, la verdadera ruta a la que redirecciona es http://localhost:3000/users la cuál está declarada en server.js y es la que hará la petición a la base de datos, aquí en parte se ve la conexión entre contenedores que está haciendo docker, pues se conectan mediante los puertos y mediante el nombre del container que es como su identificador.

Después de que el backend recibe la petición, procede a hacer la conexión con la base de datos, haciendo uso del nombre del container que es *db* (esto se demuestra cuando declara host con ese nombre)

```
const pool = mysql.createPool({
 host: "db",
 user: "root",
 password: "rootpassword",
 database: "testdb",
 waitForConnections: true,
 connectionLimit: 10,
 queueLimit: 0,
 reconnect: true
});
```

Ejecuta la query, obtiene el resultado y devuelve un json:



Finalmente esto se refleja en el frontend donde se ve la lista de usuarios: