Universidad de las Fuerzas Armadas - ESPE.

Departamento de Ciencias de la Computación.

Taller Docker.

Aplicaciones Distribuidas.

Estudiante:
Shirley Stefania Otuna Rojano.

Docente:
Ing. Darío Javier Morales Caiza.

Agosto 2024.

TALLER DOCKER - PRUEBAS Y DESPLIEGUE DE MICROSERVICIOS Y APLICACIÓN FRONTEND UTILIZANDO DOCKER

19/08/2024

INFORMACIÓN GENERAL

1. Introducción

El presente informe documenta el proceso de despliegue y pruebas de un sistema compuesto por tres microservicios desarrollados en Spring Boot y una aplicación frontend construida con React. Para gestionar el despliegue de estos componentes, se ha utilizado Docker, una plataforma de contenedorización que facilita la creación, implementación y ejecución de aplicaciones mediante contenedores.

El sistema desarrollado se centra en la gestión de cursos, estudiantes y matrículas, donde cada funcionalidad está aislada en su propio microservicio. Docker se utilizó no solo para contenedizar los microservicios, sino también para gestionar sus bases de datos PostgreSQL y desplegar la interfaz de usuario. El uso de Docker Compose permitió la orquestación de todos estos componentes, asegurando que se comunicaran correctamente entre sí en un entorno aislado.

2. Objetivo General

Implementar un sistema basado en microservicios y una aplicación frontend utilizando Docker, mediante la creación, configuración y despliegue de contenedores Docker para cada componente del sistema, con el propósito de asegurar una correcta conectividad y funcionalidad integrada entre todos los servicios, así como verificar el correcto funcionamiento del sistema a través de pruebas automatizadas y manuales.

METODOLOGÍA

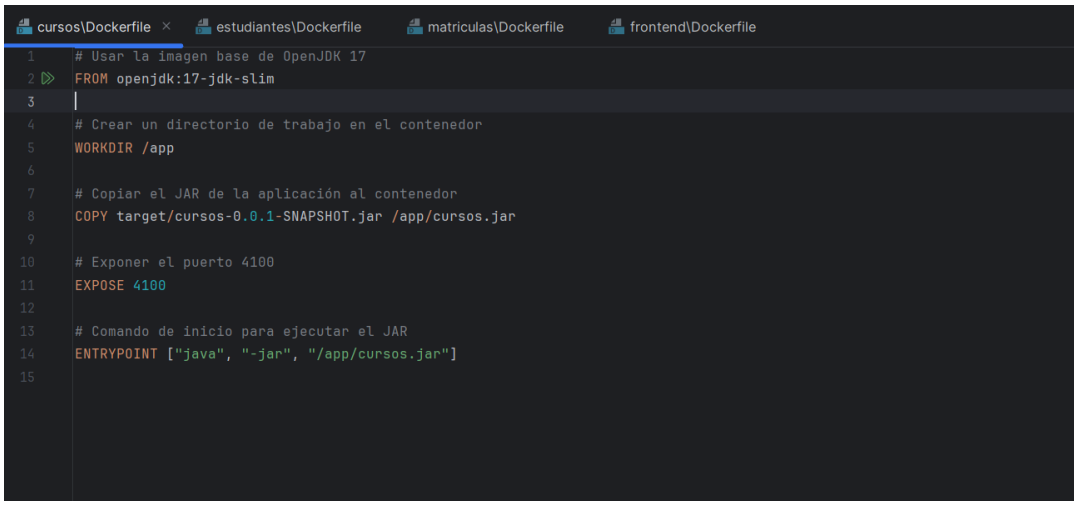
1. Preparación del Entorno

Paso 1: Crear Dockerfiles para los Microservicios

Se crearon Dockerfiles para cada microservicio (Cursos, Estudiantes, Matrículas). Cada Dockerfile define cómo construir y ejecutar el microservicio en un contenedor Docker.

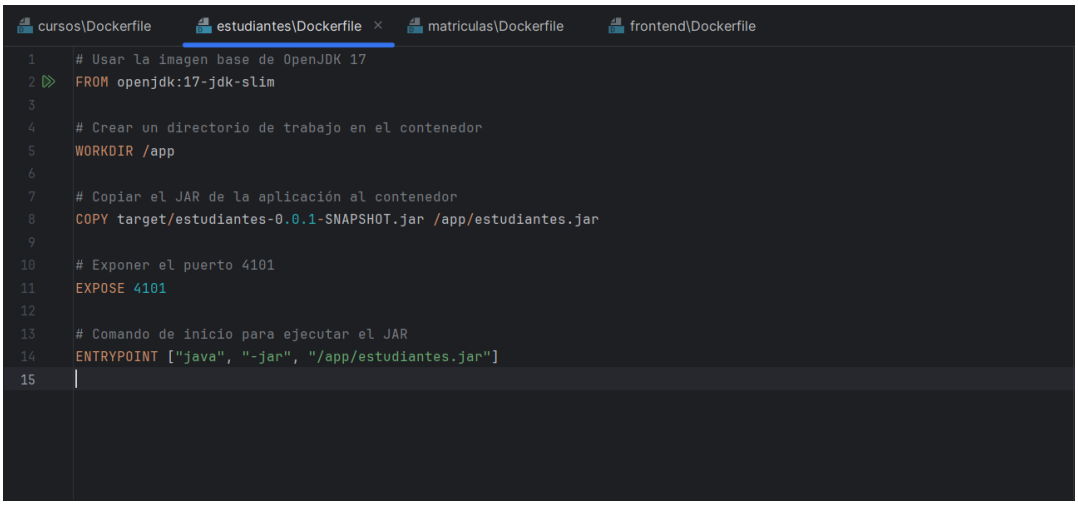
a. Dockerfile para cursos-service

Ubicación: cursos/Dockerfile



b. Dockerfile para estudiantes-service

Ubicación: estudiantes/Dockerfile



c. Dockerfile para matriculas-service

Ubicación: matriculas/Dockerfile

Paso 2: Crear Dockerfile para la Aplicación React

Se creó un Dockerfile para la aplicación frontend React, que maneja tanto la construcción de la aplicación como su despliegue utilizando Nginx.

a. Dockerfile para la aplicación React

Ubicación: frontend/Dockerfile

cursos\Dockerfile | estudiantes\Dockerfile | matriculas\Dockerfile | frontend\Dockerfile

```
# Fase 1: Construcción de la aplicación con Node.js
FROM node:18 as build

# Establecer el directorio de trabajo
WORKDIR /app

# Copiar los archivos del proyecto al contenedor
COPY package*.json ./

# Instalar las dependencias
RUN npm install

# Copiar el resto de los archivos del proyecto
COPY . .

# Construir la aplicación para producción
RUN npm run build

# Fase 2: Configuración de Nginx para servir la aplicación
FROM nginx:alpine

# Copiar los archivos construidos desde la fase anterior
COPY --from=build /app/build /usr/share/nginx/html

# Exponer el puerto en el que Nginx está sirviendo
EXPOSE 80

# Comando para iniciar Nginx
CMD ["nginx", "-g", "daemon off;"]
```

Paso 3: Crear Archivos de Configuración de PostgreSQL

Se crearon scripts SQL de inicialización para configurar las bases de datos PostgreSQL utilizadas por cada microservicio.

a. Configuración de la base de datos para cursos

Ubicación: cursos/init-cursos.sql

estudiantes\Dockerfile | EstudianteController.java | matriculas\Dockerfile | init-estudiantes.sql | init-cursos.sql | init-ma

*.sql files are supported by IntelliJ IDEA Ultimate — Try IntelliJ IDEA Ultimate

```
CREATE DATABASE cursos_db;
CREATE USER cursos_user WITH ENCRYPTED PASSWORD 'cursos_password';
GRANT ALL PRIVILEGES ON DATABASE cursos_db TO cursos_user;
```

b. Configuración de la base de datos para estudiantes

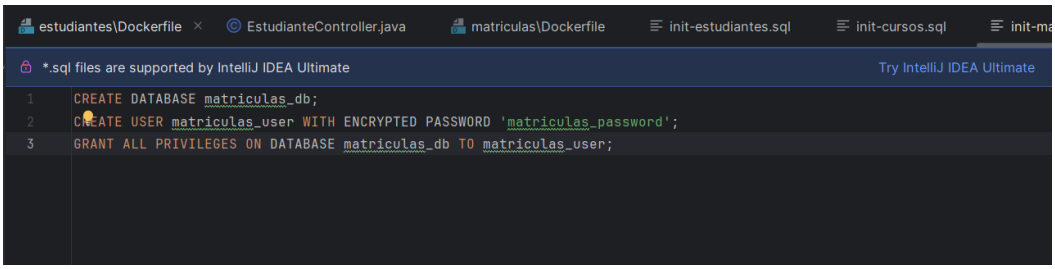
Ubicación: estudiantes/init-estudiantes.sql

*.sql files are supported by IntelliJ IDEA Ultimate — Try IntelliJ IDEA Ultimate

```
CREATE DATABASE estudiantes_db;
CREATE USER estudiantes_user WITH ENCRYPTED PASSWORD 'estudiantes_password';
GRANT ALL PRIVILEGES ON DATABASE estudiantes_db TO estudiantes_user;
```

c. Configuración de la base de datos para matriculas

Ubicación: matriculas/init-matriculas.sql



2. Construcción y Publicación de Imágenes Docker

Se construyeron las imágenes Docker para cada microservicio y la aplicación frontend, y se publicaron en Docker Hub.

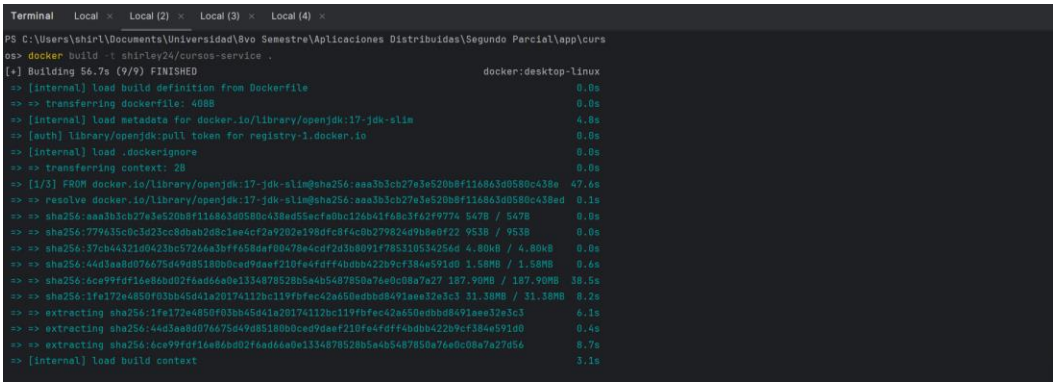
Paso 1: Construir las Imágenes Docker

Se utilizaron los siguientes comandos para construir las imágenes.

a. Construir la imagen para cursos-service

cd cursos

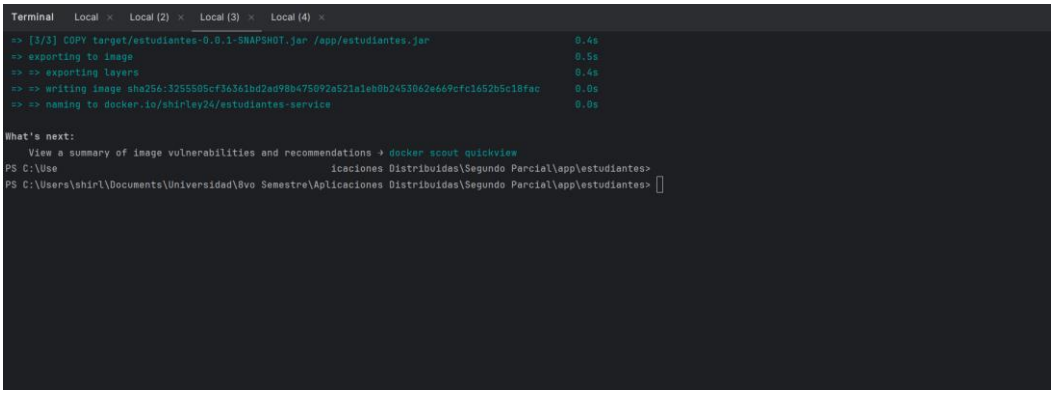
docker build -t shirley24/cursos-service .



b. Construir la imagen para estudiantes-service

cd estudiantes

docker build -t shirley24/estudiantes-service .



c. Construir la imagen para matriculas-service

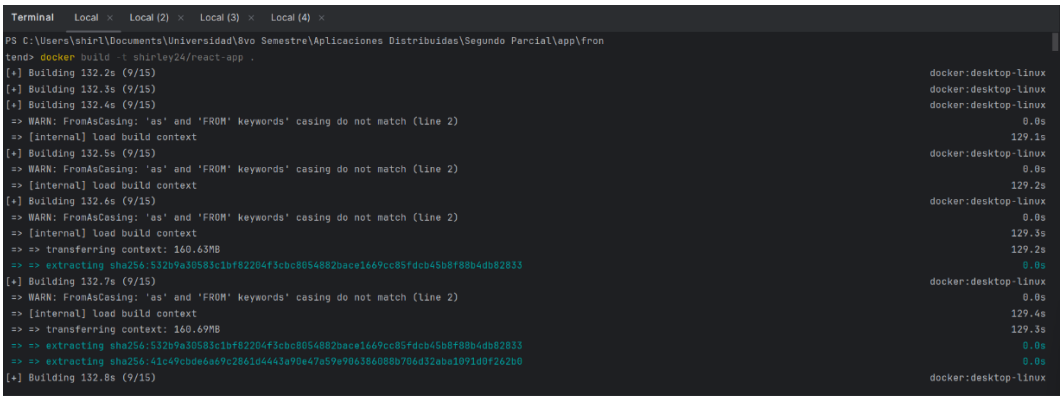
cd matriculas

docker build -t shirley24/matriculas-service .

d. Construir la imagen para la aplicación React

cd frontend

docker build -t shirley24/react-app .



Paso 2: Publicar las Imágenes en Docker Hub

Después de iniciar sesión en Docker Hub, se publicaron las imágenes:

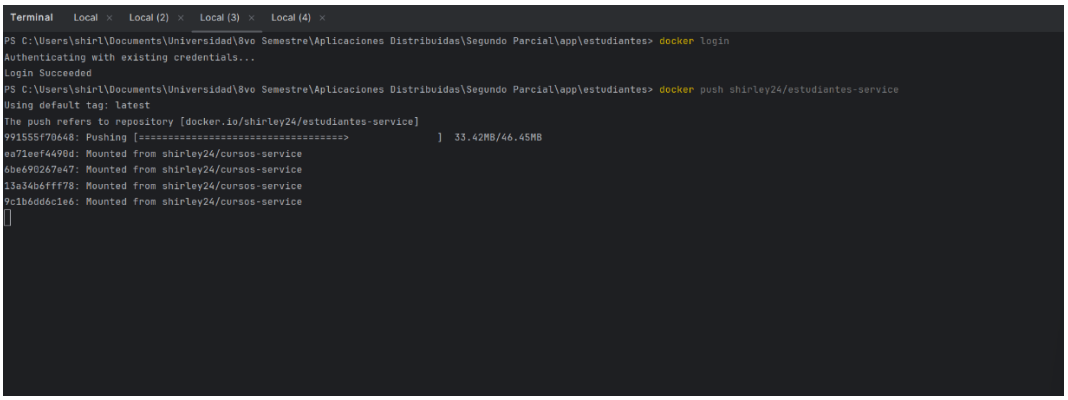
a. Iniciar sesión en Docker Hub

docker login

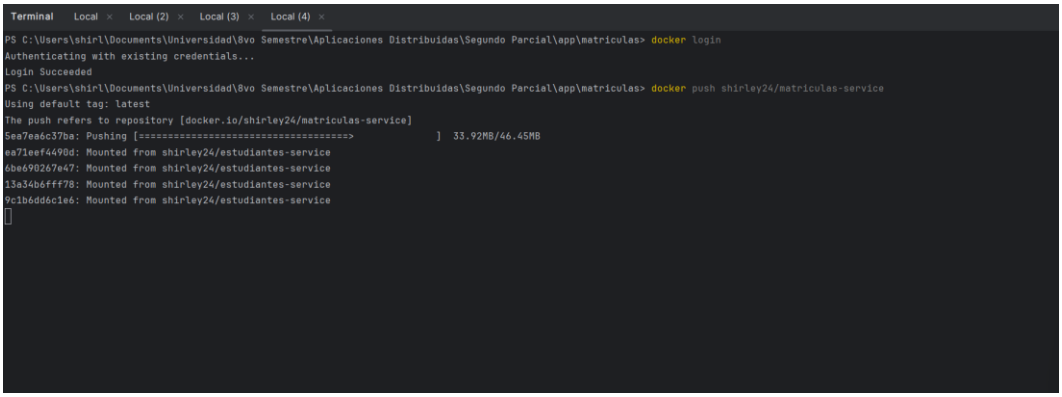
b. Publicar las imágenes

docker push shirley24 /cursos-service

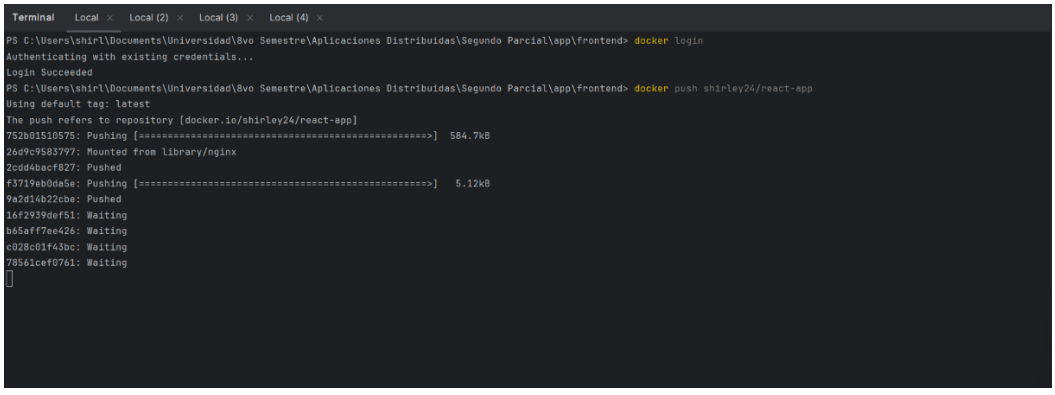
docker push shirley24/estudiantes-service



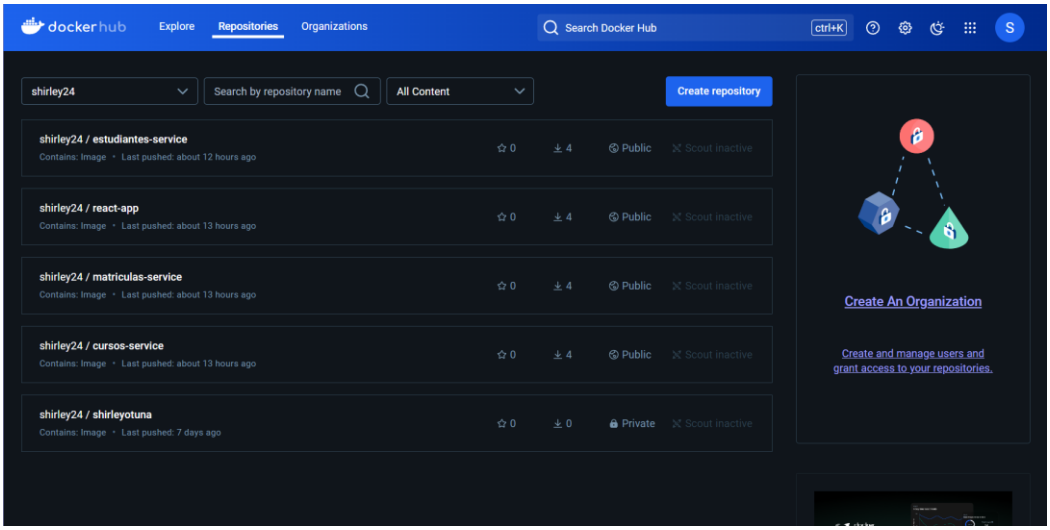
docker push shirley24/matriculas-service



docker push shirley24/react-app



c. Docker Hub



3. Despliegue del Sistema con Docker Compose

> Se configuró un archivo docker-compose.yml para coordinar el despliegue de todos los servicios y sus bases de datos.

a. Archivo docker-compose.yml

b. Ubicación: appdocker-compose.yml

```yaml
version: '3'

services:
  cursos-db:
    image: postgres:15
    container_name: cursos-db
    environment:
      POSTGRES_DB: cursos_db
      POSTGRES_USER: cursos_user
      POSTGRES_PASSWORD: cursos_password
    volumes:
      - ./cursos/init-cursos.sql:/docker-entrypoint-initdb.d/init-cursos.sql
    networks:
      - microservices-network
    ports:
      - "5432:5432"

  estudiantes-db:
    image: postgres:15
    container_name: estudiantes-db
    environment:
      POSTGRES_DB: estudiantes_db
      POSTGRES_USER: estudiantes_user
      POSTGRES_PASSWORD: estudiantes_password
    volumes:
      - ./estudiantes/init-estudiantes.sql:/docker-entrypoint-initdb.d/init-estudiantes.sql
    networks:
      - microservices-network
```

```yaml
services:
  estudiantes-db:
    ports:
      - "5433:5432"

  matriculas-db:
    image: postgres:15
    container_name: matriculas-db
    environment:
      POSTGRES_DB: matriculas_db
      POSTGRES_USER: matriculas_user
      POSTGRES_PASSWORD: matriculas_password
    volumes:
      - ./matriculas/init-matriculas.sql:/docker-entrypoint-initdb.d/init-matriculas.sql
    networks:
      - microservices-network
    ports:
      - "5434:5432"

  cursos-service:
    image: shirley24/cursos-service
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://cursos-db:5432/cursos_db
      SPRING_DATASOURCE_USERNAME: cursos_user
      SPRING_DATASOURCE_PASSWORD: cursos_password
    ports:
      - "4100:4100"
    networks:
      - microservices-network
```

```yaml
services:
  cursos-service:
    depends_on:
      - cursos-db

  estudiantes-service:
    image: shirley24/estudiantes-service
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://estudiantes-db:5432/estudiantes_db
      SPRING_DATASOURCE_USERNAME: estudiantes_user
      SPRING_DATASOURCE_PASSWORD: estudiantes_password
    ports:
      - "4101:4101"
    networks:
      - microservices-network
    depends_on:
      - estudiantes-db

  matriculas-service:
    image: shirley24/matriculas-service
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://matriculas-db:5432/matriculas_db
      SPRING_DATASOURCE_USERNAME: matriculas_user
      SPRING_DATASOURCE_PASSWORD: matriculas_password
    ports:
      - "4102:4102"
    networks:
      - microservices-network
```

```yaml
    depends_on:
      - matriculas-db

  react-app:
    image: shirley24/react-app
    ports:
      - "80:80"
    networks:
      - microservices-network

networks:
  microservices-network:
    driver: bridge
```

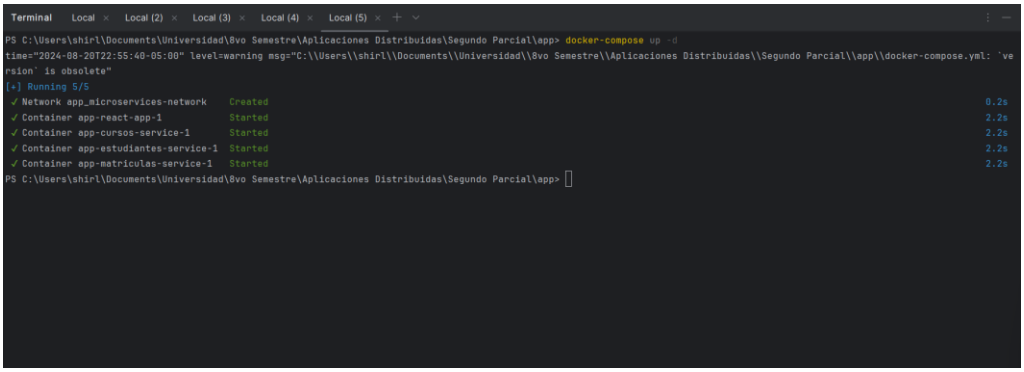
Paso 1: Desplegar los Servicios

Utilizando Docker Compose, se levantaron todos los servicios:

c. Navegar a la raíz del proyecto donde se encuentra el archivo docker-compose.yml.

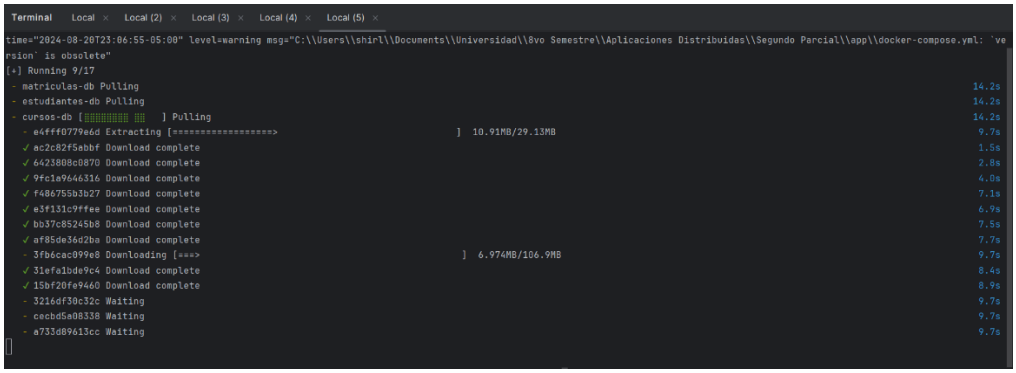
d. Ejecutar el siguiente comando para los servicios

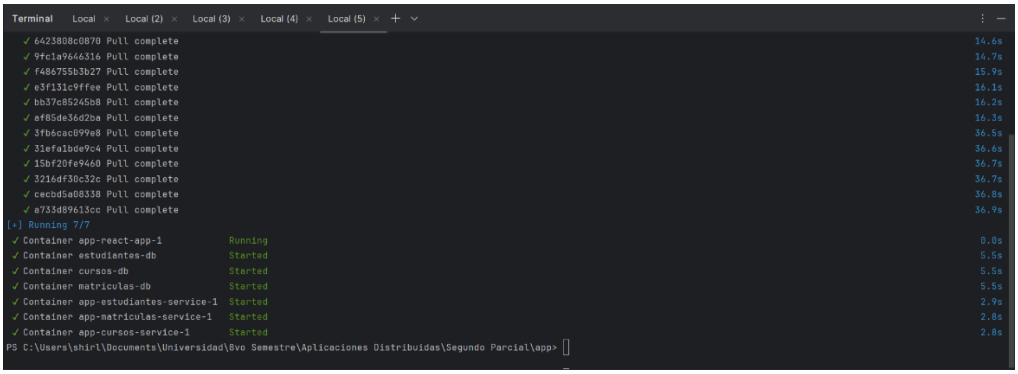
docker-compose up -d



e. Ejecutar el siguiente comando para las bases de datos

docker-compose up -d

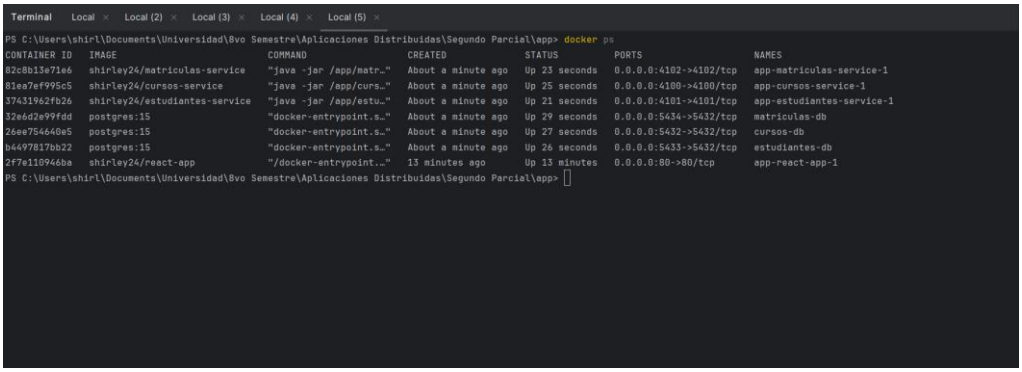




Paso 2: Verificar que los Contenedores Estén Corriendo

Se utilizó el comando docker ps para verificar que todos los contenedores estaban en ejecución correctamente:

a. **Ejecutar el siguiente comando:**
docker ps

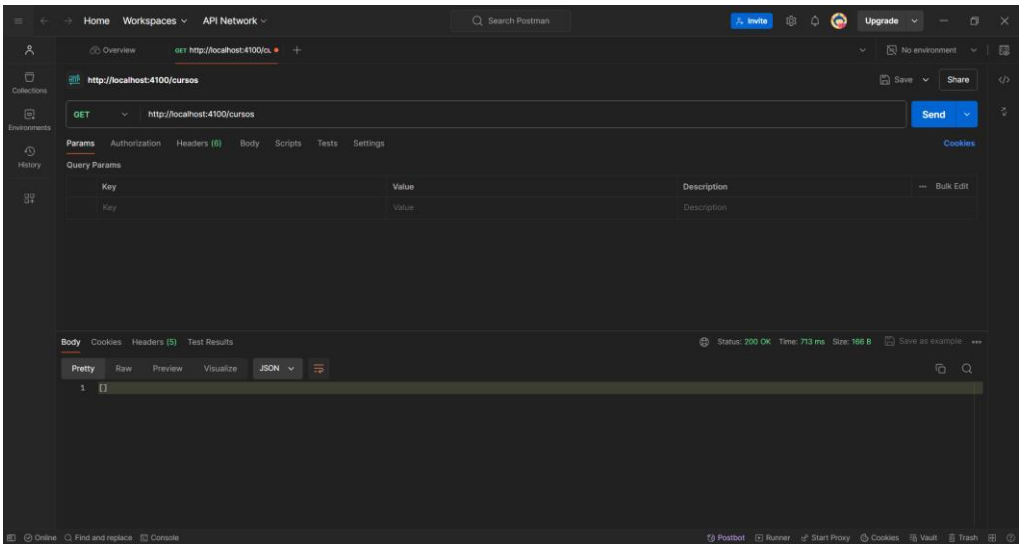


4. Pruebas del Sistema

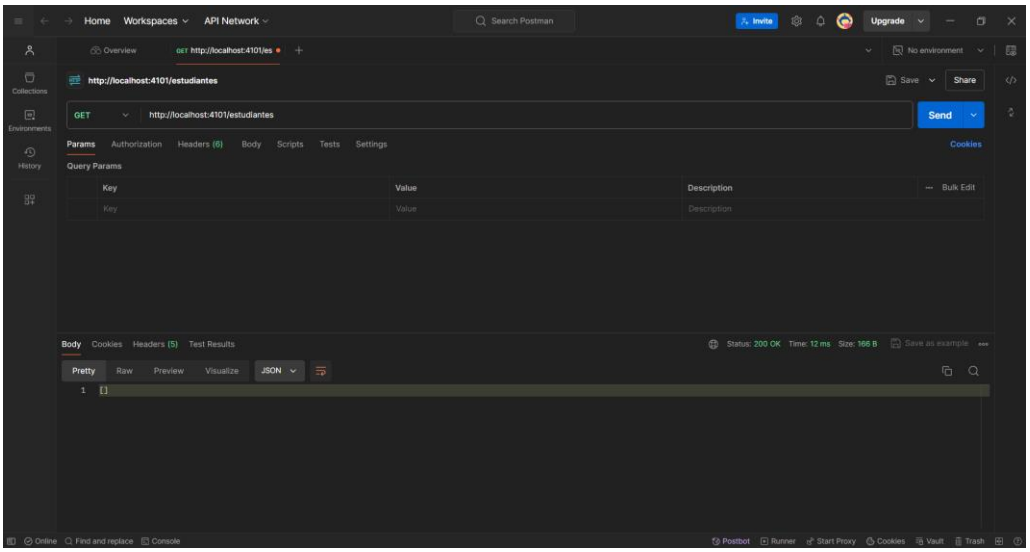
Paso 1: Verificación de Conectividad entre Microservicios

Las pruebas con Postman verificaron que los microservicios respondieron correctamente a las solicitudes API. Se probaron las operaciones CRUD para cada microservicio utilizando las siguientes rutas:

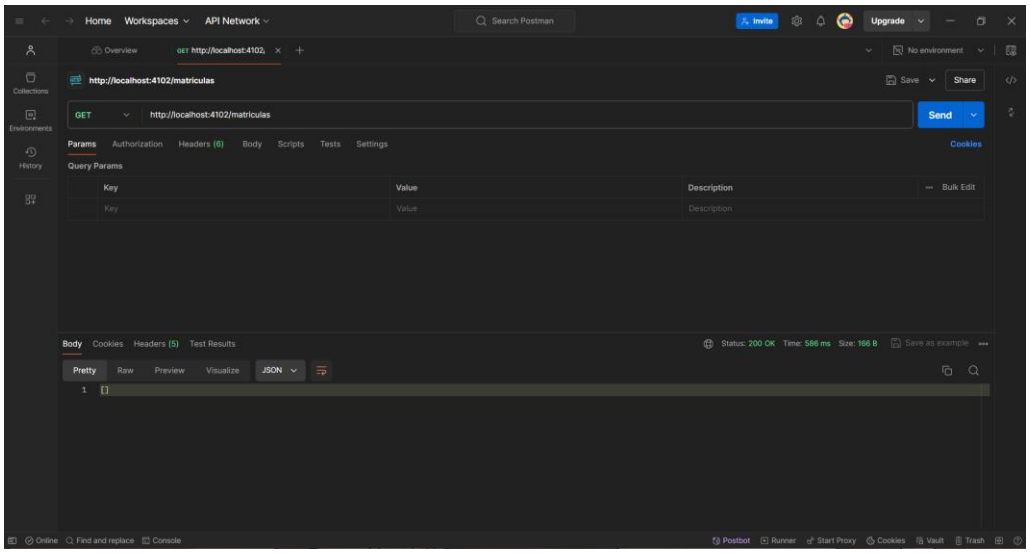
a. **Cursos**: http://localhost:4100/cursos



b. **Estudiantes**: http://localhost:4101/estudiantes

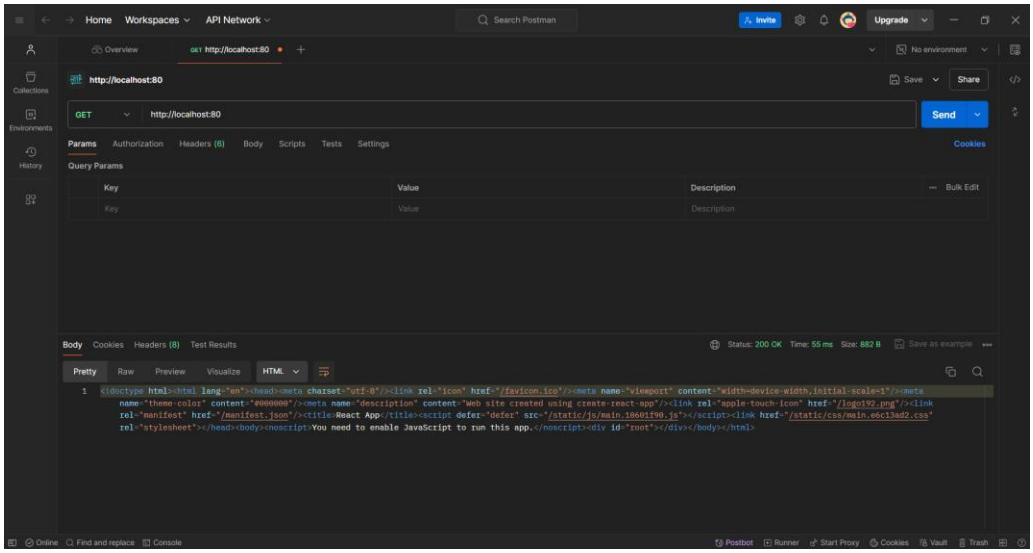


c. **Matrículas**: http://localhost:4102/matriculas



Paso 2: Pruebas de la Aplicación React

Se accedió a la aplicación frontend en http://localhost, donde se realizaron pruebas de funcionalidad utilizando la interfaz gráfica. Se verificó que las operaciones CRUD se reflejaran correctamente en la base de datos.



Paso 3: Pruebas de Integración

Se realizaron pruebas de integración para asegurarse de que los microservicios y la aplicación frontend funcionaran de manera coherente. Se probó un flujo completo donde un estudiante se matricula en un curso, y se verificó que todos los datos se gestionaran correctamente.

RESULTADOS

1. Verificación de Conectividad entre Microservicios

Las pruebas realizadas con Postman verificaron que los microservicios están expuestos en los puertos correctos y responden adecuadamente a las solicitudes API. Se probaron las operaciones CRUD básicas (crear, leer, actualizar, eliminar) en cada microservicio.

2. Pruebas de la Aplicación React

Se verificó que la aplicación React interactúa correctamente con los microservicios. Las operaciones CRUD fueron probadas a través de la interfaz gráfica, y los datos reflejados en la base de datos correspondieron correctamente a las acciones realizadas en la interfaz.

3. Pruebas de Integración

Las pruebas de integración confirmaron que los microservicios y la aplicación React funcionan de manera coherente e integrada. Se realizaron operaciones de matrícula de estudiantes en cursos y se verificó que los datos fueran almacenados y recuperados correctamente.

PROBLEMAS ENCONTRADOS Y SOLUCIONES

Durante las pruebas, se encontró un problema relacionado con la configuración incorrecta de las rutas en los microservicios, lo que causó errores 404. Esto se resolvió revisando y corrigiendo las rutas en los controladores de los microservicios.

Otro problema detectado fue la necesidad de reiniciar los contenedores después de aplicar configuraciones en el archivo docker-compose.yml para asegurar que los cambios surtieran efecto.

REFLEXIÓN: VENTAJAS Y DESAFÍOS DE DOCKER

1. Ventajas

Docker permite aislar y escalar cada microservicio de manera independiente, lo que facilita el mantenimiento y la actualización del sistema.

Las imágenes Docker pueden ser ejecutadas en cualquier entorno que soporte Docker, lo que simplifica el proceso de despliegue.

Docker Compose permite configurar y desplegar todos los servicios del sistema de manera coherente y repetible.

2. Desafíos

La configuración inicial de Dockerfiles y Docker Compose puede ser compleja, especialmente para sistemas con múltiples servicios interdependientes.

La gestión de volúmenes para la persistencia de datos en bases de datos puede requerir una planificación cuidadosa para evitar pérdida de datos.

CONCLUSIONES

En base al objetivo planteado, se **implementó** un sistema basado en microservicios y una aplicación frontend utilizando Docker, **mediante** la creación, configuración y despliegue de contenedores Docker para cada componente del sistema. Durante este proceso, se logró asegurar una correcta conectividad y funcionalidad integrada entre los servicios de cursos, estudiantes y matrículas, así como con la aplicación frontend.

Las pruebas realizadas, tanto automatizadas como manuales, permitieron verificar que todos los componentes del sistema funcionaran de manera coordinada, cumpliendo con los requerimientos de operación y comunicación esperados. Aunque se presentaron algunos desafíos, como errores en la configuración inicial de rutas y la necesidad de reiniciar los contenedores para aplicar correctamente ciertas configuraciones, estos problemas fueron solucionados eficazmente, demostrando la flexibilidad y robustez que Docker ofrece en el despliegue de aplicaciones complejas.

En conclusión, el uso de Docker para desplegar y gestionar un sistema basado en microservicios y una aplicación frontend no solo facilita la portabilidad y escalabilidad del sistema, sino que también permite mantener una consistencia en la configuración y operación de los servicios, cumpliendo así con el objetivo de asegurar una integración exitosa y el correcto funcionamiento del sistema en su conjunto.

ENLACE A IMÁGENES EN DOCKER HUB

a. **Cursos-service:** https://hub.docker.com/repository/docker/shirley24/cursos-service/general
b. **Estudiantes-service:** https://hub.docker.com/repository/docker/shirley24/estudiantes-service/general
c. **Matriculas-service:** https://hub.docker.com/repository/docker/shirley24/matriculas-service/general
d. **React-app:** https://hub.docker.com/repository/docker/shirley24/react-app/general