People’s Democratic Republic of Algeria Ministry of Higher Education and Scientific Research

Graduate School of Science and Technology of Computer Science and Digital

Project Report:

Thème:

Docker Compose

This Project is carried out by:

BENKERRI Ranim

Framed By:
Mme Khelouf Hanane

1-Introduction

This project focuses on developing a microservice-based application using Docker and Docker Compose, showcasing how different technology stacks can Type your text

be containerized and managed. The project aims to create a functioning web application with API and database services, all integrated through Docker. The technologies used include Flask for the web application, FastAPI for the API service, and PostgreSQL as the database service.

2-Technologies Used

- Docker and Docker Compose
- Flask (python) for the web application
- FastAPI (Python) for API service
- PostgreSQL for the database
- Adminer for database management and visualization

3-System Flow Diagram

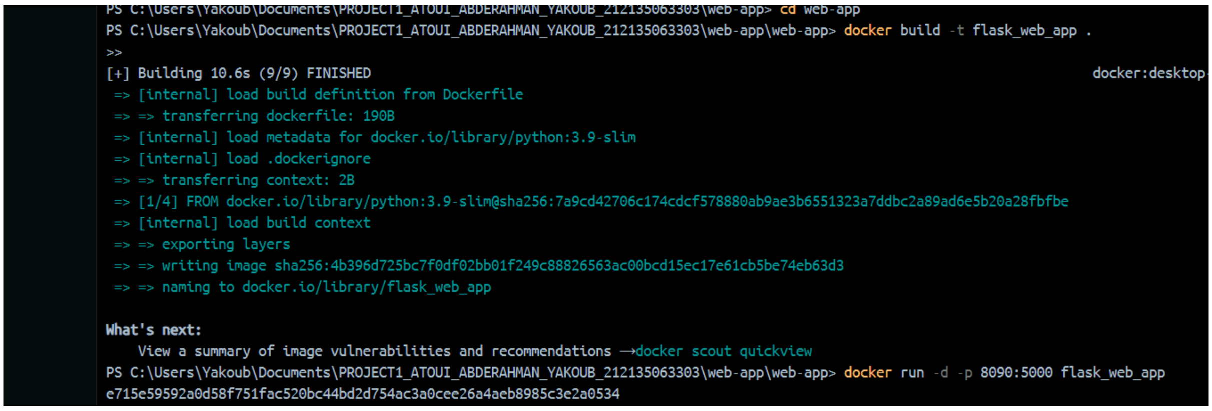
The overall system architecture comprises multiple services working in isolation but networked through Docker. The services include:

1. A web application running in Flask.
2. An API service developed with FastAPI.
3. A PostgreSQL database.
4. Adminer for database management.

4-Steps Executed in the Project

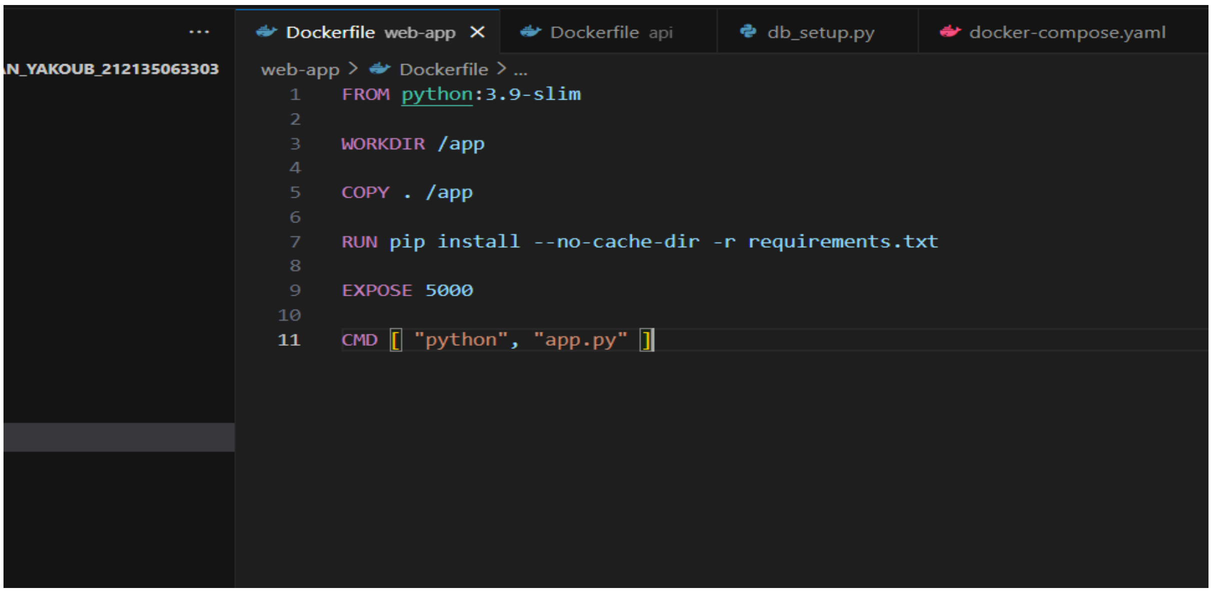
1. **Exercise 1:** Web-App Setup with Flask

- Created a Docker container for a Python-based Flask web application
- Configured the app to run on port 8090 on the host and 5000 in the container.
- Ensured that the web application was functional with endpoints like `/add` and `/all`.



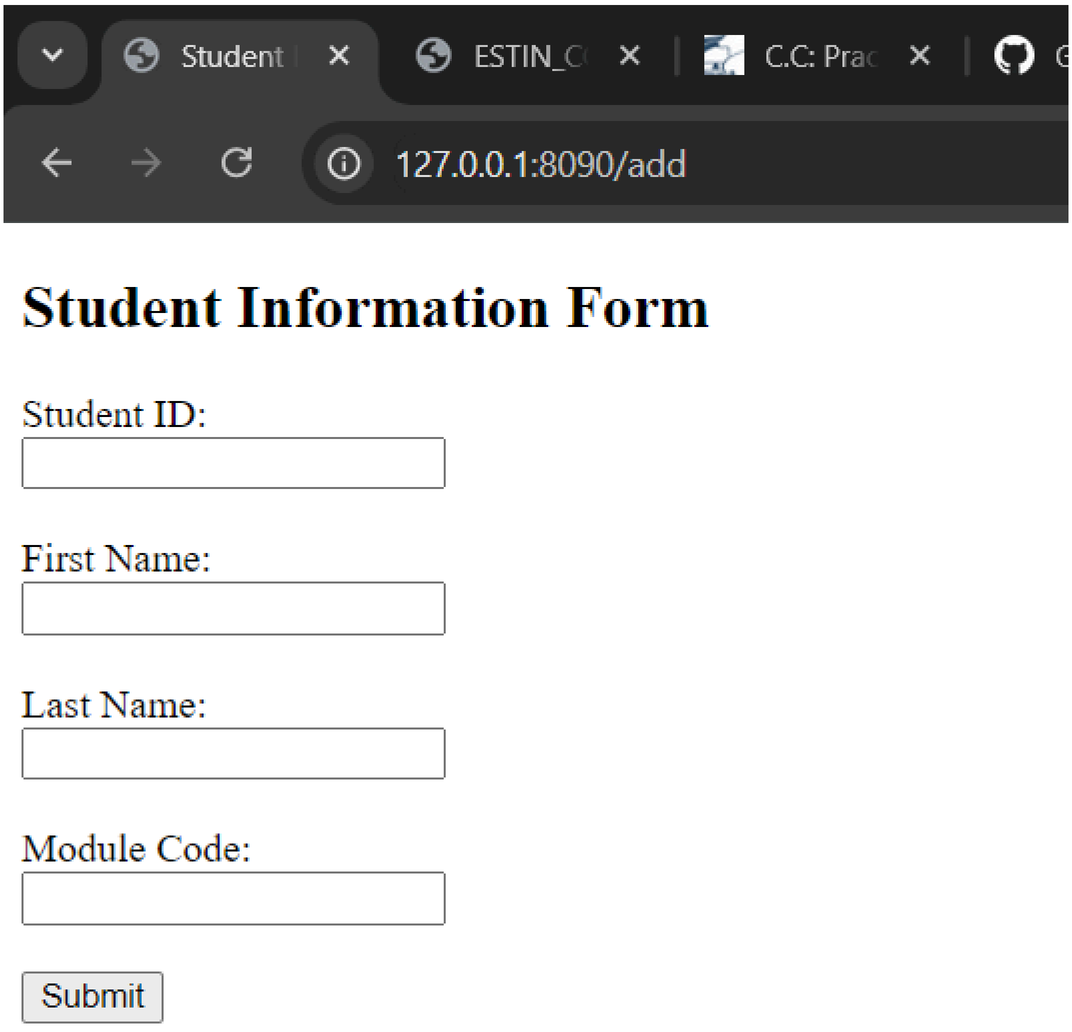
```
PS C:\Users\Yakoub\Documents\PROJECT1_ATOUI_ABDERAHMAN_YAKOUB_212135063303\web-app> cd web-app
PS C:\Users\Yakoub\Documents\PROJECT1_ATOUI_ABDERAHMAN_YAKOUB_212135063303\web-app\web-app> docker build -t flask_web_app .
>>
[+] Building 10.6s (9/9) FINISHED                                                                          docker:desktop
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 190B
 => [internal] load metadata for docker.io/library/python:3.9-slim
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/4] FROM docker.io/library/python:3.9-slim@sha256:7a9cd42706c174cdcf578880ab9ae3b6551323a7ddbc2a89ad6e5b20a28fbfbe
 => [internal] load build context
 => => exporting layers
 => => writing image sha256:4b396d725bc7f0df02bb01f249c88826563ac00bcd15ec17e61cb5be74eb63d3
 => => naming to docker.io/library/flask_web_app

What's next:
    View a summary of image vulnerabilities and recommendations →docker scout quickview
PS C:\Users\Yakoub\Documents\PROJECT1_ATOUI_ABDERAHMAN_YAKOUB_212135063303\web-app\web-app> docker run -d -p 8090:5000 flask_web_app
e715e59592a0d58f751fac520bc44bd2d754ac3a0cee26a4aeb8985c3e2a0534
```

Figure 1: Building my flask web app container & running the container



```
FROM python:3.9-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 5000

CMD [ "python", "app.py" ]
```

Figure 2: The dockerfile of web-app



Student Information Form

Student ID:

First Name:

Last Name:

Module Code:

Figure 3: The result of the web-app

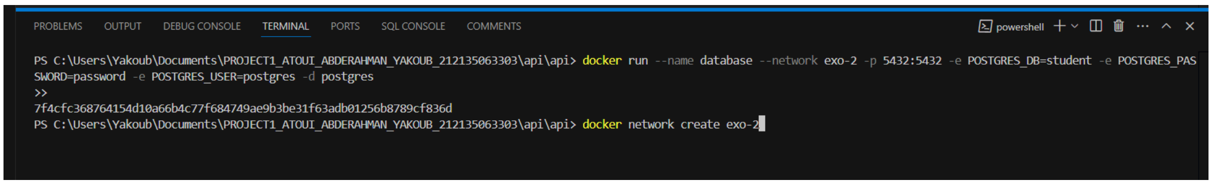
2. **Exercise 2:** API Setup with FastAPI

- Developed the API service with FastAPI, also containerized using Docker.

- Integrated the API with a temporary PostgreSQL database running in a separate container.

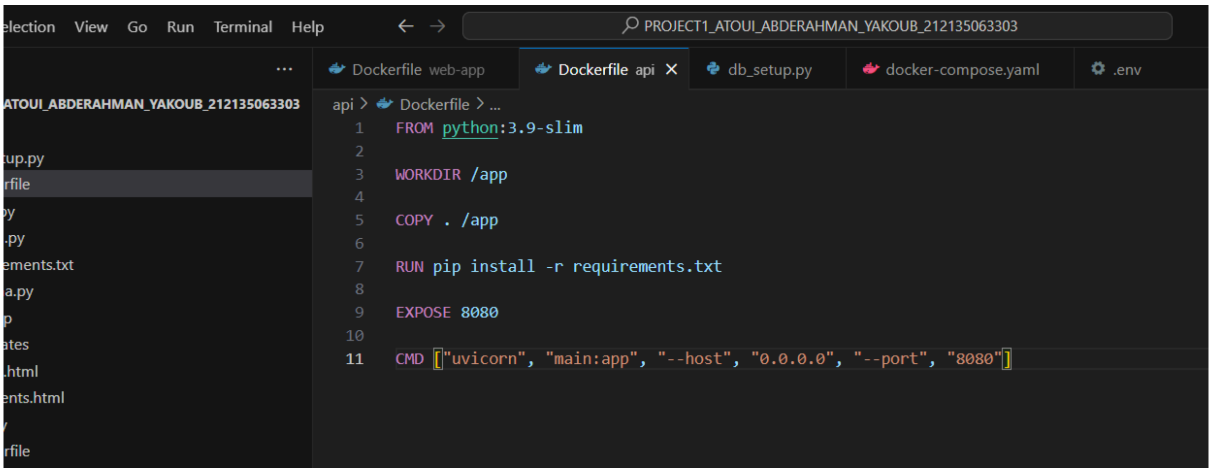
- Successfully exposed the API on port 8081, tested via the `/docs` FastAPI interface.

- Used **Docker network** to ensure seamless communication between the API service and the PostgreSQL database. This involved setting up a custom Docker network to allow both containers (API and database) to communicate without issues. This step was crucial for resolving connection challenges between the API and the database container.



```
PS C:\Users\Yakoub\Documents\PROJECT1_ATOUI_ABDERAHMAN_YAKOUB_212135063303\api\api> docker run --name database --network exo-2 -p 5432:5432 -e POSTGRES_DB=student -e POSTGRES_PAS
SWORD=password -e POSTGRES_USER=postgres -d postgres
>>
7f4cfc368764154d10a66b4c77f684749ae9b3be31f63adb01256b8789cf836d
PS C:\Users\Yakoub\Documents\PROJECT1_ATOUI_ABDERAHMAN_YAKOUB_212135063303\api\api> docker network create exo-2
```

Figure 4:Create a network & Running temporary db in the network

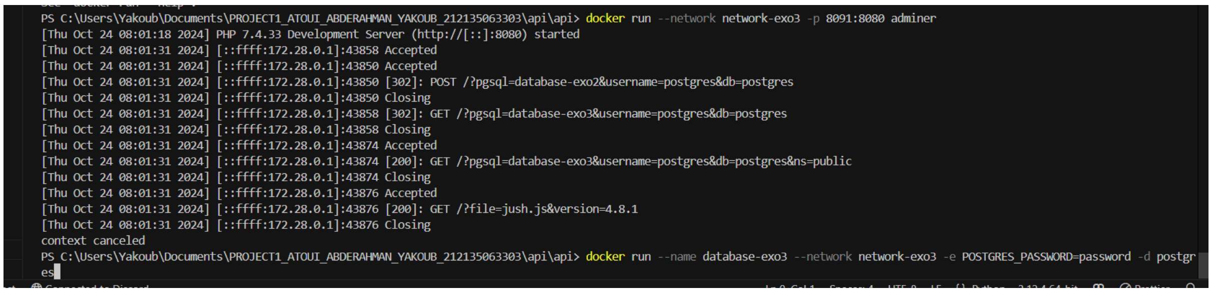

election View Go Run Terminal Help
PROJECT1_ATOUI_ABDERAHMAN_YAKOUB_212135063303
Dockerfile web-app
Dockerfile api
db_setup.py
docker-compose.yaml
.env
ATOUI_ABDERAHMAN_YAKOUB_212135063303
api > Dockerfile > ...

```
FROM python:3.9-slim

WORKDIR /app

COPY . /app

RUN pip install -r requirements.txt

EXPOSE 8080

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8080"]
```

Figure 5:Dockerfile of api

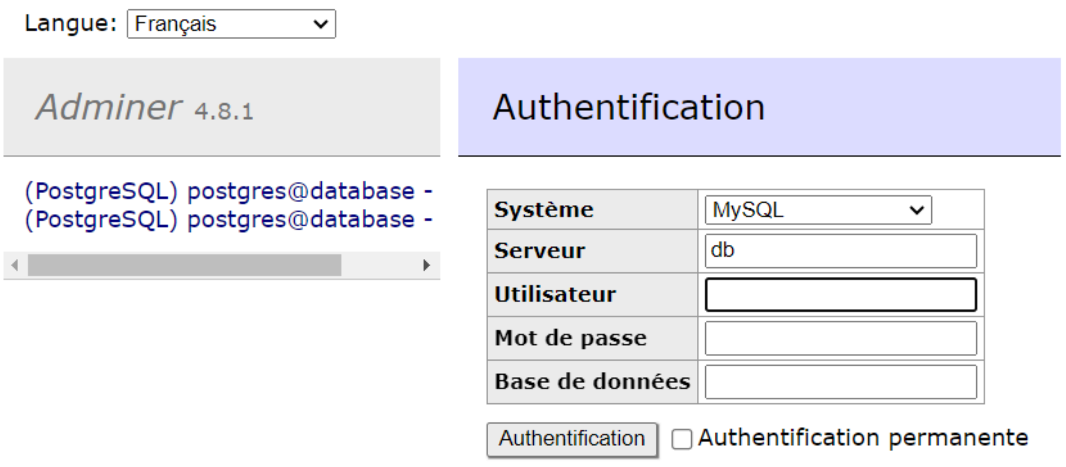
3. **Exercise 3:** Database Service Setup

- Configured a PostgreSQL database in a container.
- Introduced **Adminer** to visualize and manage the database from a browser at port 8091.
- Created a custom Docker network to allow inter-container communication, enabling the API to connect to the database.



```
PS C:\Users\Yakoub\Documents\PROJECT1_ATOUI_ABDERAHMAN_YAKOUB_212135063303\api\api> docker run --network network-exo3 -p 8091:8080 adminer
[Thu Oct 24 08:01:18 2024] PHP 7.4.33 Development Server (http://[::]:8080) started
[Thu Oct 24 08:01:31 2024] [::ffff:172.28.0.1]:43858 Accepted
[Thu Oct 24 08:01:31 2024] [::ffff:172.28.0.1]:43850 Accepted
[Thu Oct 24 08:01:31 2024] [::ffff:172.28.0.1]:43850 [302]: POST /?pgsql=database-exo2&username=postgres&db=postgres
[Thu Oct 24 08:01:31 2024] [::ffff:172.28.0.1]:43850 Closing
[Thu Oct 24 08:01:31 2024] [::ffff:172.28.0.1]:43858 [302]: GET /?pgsql=database-exo3&username=postgres&db=postgres
[Thu Oct 24 08:01:31 2024] [::ffff:172.28.0.1]:43858 Closing
[Thu Oct 24 08:01:31 2024] [::ffff:172.28.0.1]:43874 Accepted
[Thu Oct 24 08:01:31 2024] [::ffff:172.28.0.1]:43874 [200]: GET /?pgsql=database-exo3&username=postgres&db=postgres&ns=public
[Thu Oct 24 08:01:31 2024] [::ffff:172.28.0.1]:43874 Closing
[Thu Oct 24 08:01:31 2024] [::ffff:172.28.0.1]:43876 Accepted
[Thu Oct 24 08:01:31 2024] [::ffff:172.28.0.1]:43876 [200]: GET /?file=jush.js&version=4.8.1
[Thu Oct 24 08:01:31 2024] [::ffff:172.28.0.1]:43876 Closing
context canceled
PS C:\Users\Yakoub\Documents\PROJECT1_ATOUI_ABDERAHMAN_YAKOUB_212135063303\api\api> docker run --name database-exo3 --network network-exo3 -e POSTGRES_PASSWORD=password -d postgr
es
```

Figure 10:Configured PostgreSQL database in a container & Running admine



Langue: Français

Adminer 4.8.1

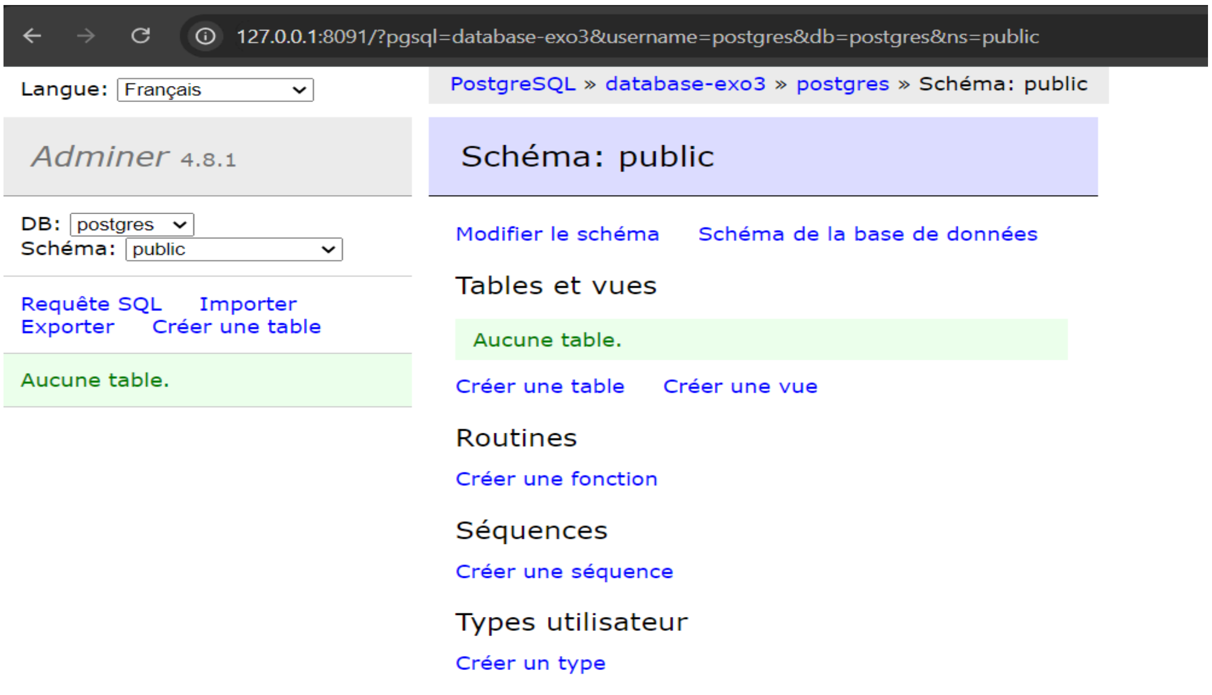
(PostgreSQL) postgres@database -
(PostgreSQL) postgres@database -

Authentification

Système	MySQL
Serveur	db
Utilisateur	
Mot de passe	
Base de données	

Authentification ☐ Authentification permanente

Figure 6:Adminer before configure the network


127.0.0.1:8091/?pgsql=database-exo3&username=postgres&db=postgres&ns=public

Langue: Français

Adminer 4.8.1

DB: postgres
Schéma: public

Requête SQL Importer
Exporter Créer une table

Aucune table.

PostgreSQL » database-exo3 » postgres » Schéma: public

Schéma: public

Modifier le schéma Schéma de la base de données

Tables et vues

Aucune table.

Créer une table Créer une vue

Routines

Créer une fonction

Séquences

Créer une séquence

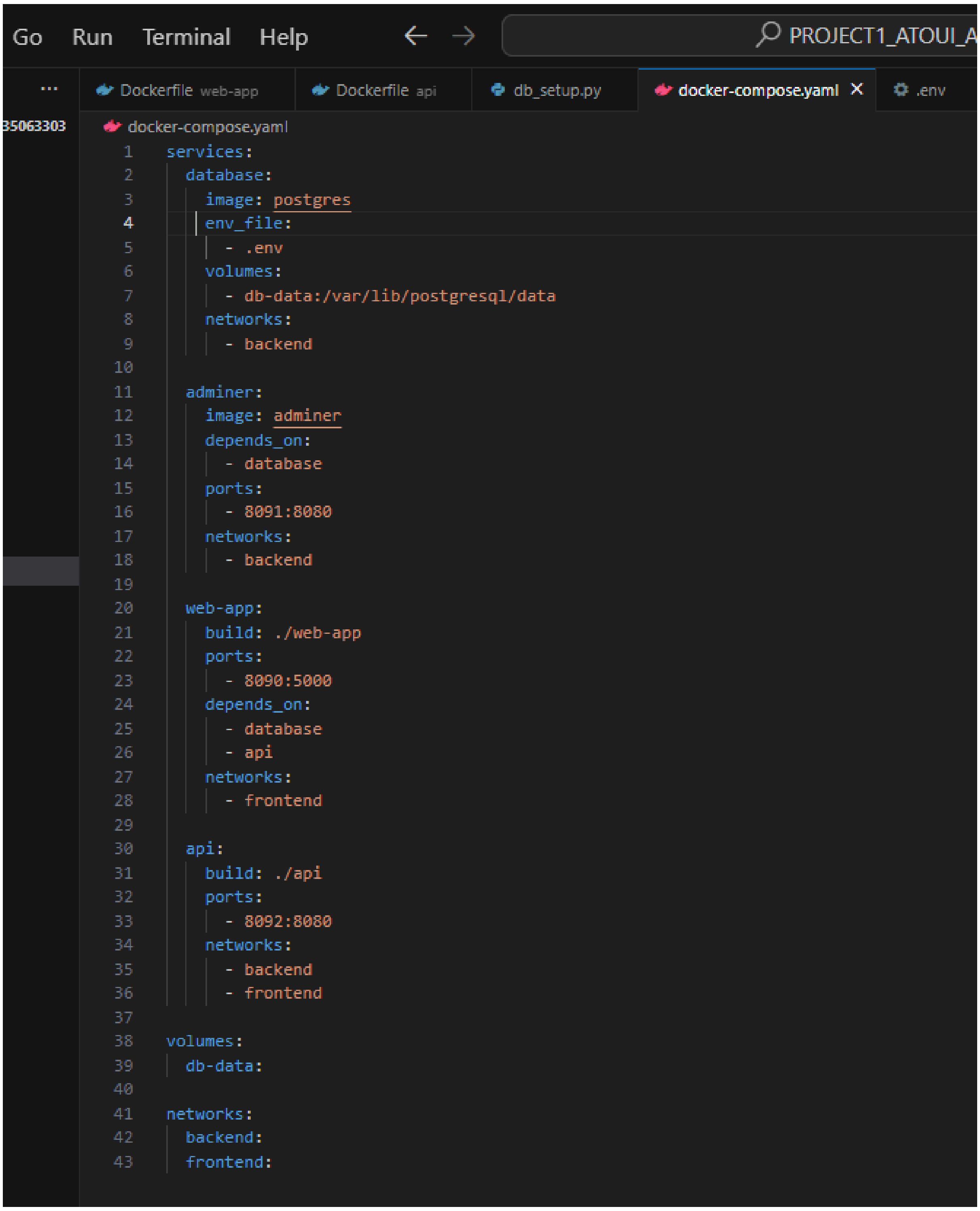
Types utilisateur

Créer un type

Figure 7:Adminer works

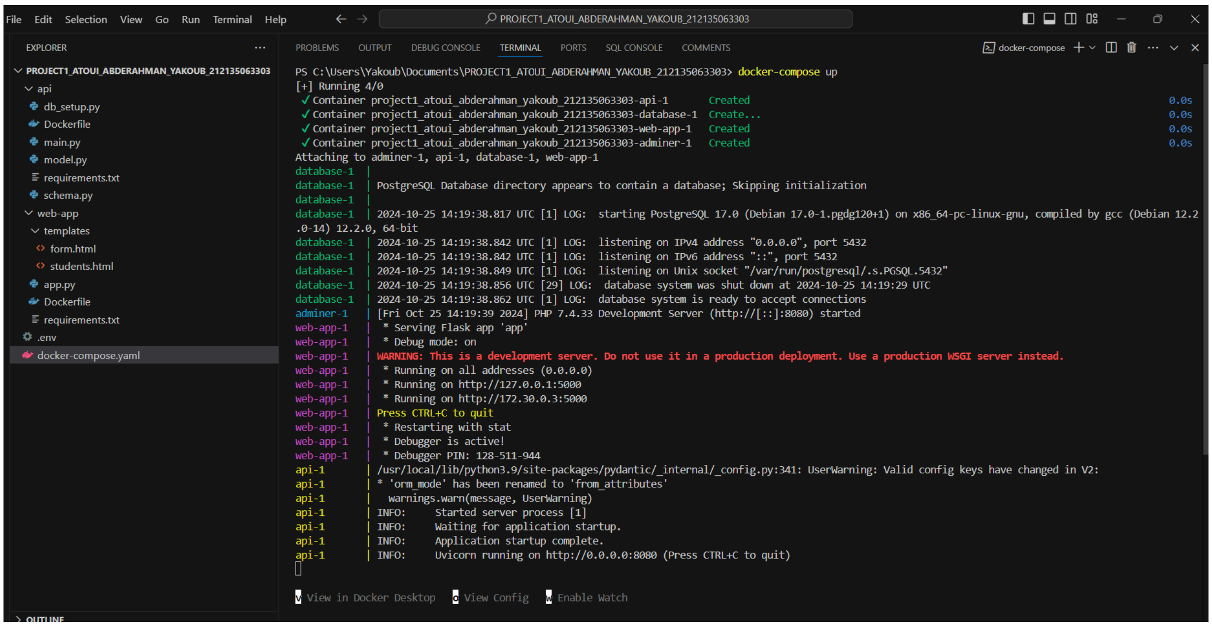
4. **Exercise 4:** Docker Compose

- Consolidated all services (web-app, API, database, Adminer) into a single Docker Compose setup.
- Implemented network and volume management in the `docker-compose.yaml` file
- Verified that the web application interacts with the API and the data persists in the database



```yaml
services:
  database:
    image: postgres
    env_file:
      - .env
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      - backend

  adminer:
    image: adminer
    depends_on:
      - database
    ports:
      - 8091:8080
    networks:
      - backend

  web-app:
    build: ./web-app
    ports:
      - 8090:5000
    depends_on:
      - database
      - api
    networks:
      - frontend

  api:
    build: ./api
    ports:
      - 8092:8080
    networks:
      - backend
      - frontend

volumes:
  db-data:

networks:
  backend:
  frontend:
```

Figure 8:Docker compose file


File Edit Selection View Go Run Terminal Help
PROJECT1_ATOUI_ABDERAHMAN_YAKOUB_212135063303
EXPLORER
PROJECT1_ATOUI_ABDERAHMAN_YAKOUB_212135063303
api
db_setup.py
Dockerfile
main.py
model.py
requirements.txt
schema.py
web-app
templates
form.html
students.html
app.py
Dockerfile
requirements.txt
.env
docker-compose.yaml
OUTLINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE COMMENTS
docker-compose

```
PS C:\Users\Yakoub\Documents\PROJECT1_ATOUI_ABDERAHMAN_YAKOUB_212135063303> docker-compose up
[+] Running 4/0
 ✓Container project1_atoui_abderahman_yakoub_212135063303-api-1       Created                    0.0s
 ✓Container project1_atoui_abderahman_yakoub_212135063303-database-1  Create...                  0.0s
 ✓Container project1_atoui_abderahman_yakoub_212135063303-web-app-1   Created                    0.0s
 ✓Container project1_atoui_abderahman_yakoub_212135063303-adminer-1   Created                    0.0s
Attaching to adminer-1, api-1, database-1, web-app-1
database-1  |
database-1  | PostgreSQL Database directory appears to contain a database; Skipping initialization
database-1  |
database-1  | 2024-10-25 14:19:38.817 UTC [1] LOG:  starting PostgreSQL 17.0 (Debian 17.0-1.pgdg120+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 12.2
.0-14) 12.2.0, 64-bit
database-1  | 2024-10-25 14:19:38.842 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
database-1  | 2024-10-25 14:19:38.842 UTC [1] LOG:  listening on IPv6 address "::", port 5432
database-1  | 2024-10-25 14:19:38.849 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
database-1  | 2024-10-25 14:19:38.856 UTC [29] LOG:  database system was shut down at 2024-10-25 14:19:29 UTC
database-1  | 2024-10-25 14:19:38.862 UTC [1] LOG:  database system is ready to accept connections
adminer-1   | [Fri Oct 25 14:19:39 2024] PHP 7.4.33 Development Server (http://[::]:8080) started
web-app-1   |  * Serving Flask app 'app'
web-app-1   |  * Debug mode: on
web-app-1   | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
web-app-1   |  * Running on all addresses (0.0.0.0)
web-app-1   |  * Running on http://127.0.0.1:5000
web-app-1   |  * Running on http://172.30.0.3:5000
web-app-1   | Press CTRL+C to quit
web-app-1   |  * Restarting with stat
web-app-1   |  * Debugger is active!
web-app-1   |  * Debugger PIN: 128-511-944
api-1       | /usr/local/lib/python3.9/site-packages/pydantic/_internal/_config.py:341: UserWarning: Valid config keys have changed in V2:
api-1       | * 'orm_mode' has been renamed to 'from_attributes'
api-1       |   warnings.warn(message, UserWarning)
api-1       | INFO:     Started server process [1]
api-1       | INFO:     Waiting for application startup.
api-1       | INFO:     Application startup complete.
api-1       | INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)

v View in Docker Desktop   o View Config   w Enable Watch
```

Figure 9:Building the compose and running

5-Conclusion

This project demonstrates the successful application of Docker and Docker Compose to manage microservices. The services were efficiently containerized and interconnected, following best practices like utilizing shared networks, environment variables, and persistent volumes to enhance scalability and maintainability.