NAME: Mohammad Hussam (2303.KHI.DEG.020)

PAIRING WITH **: MAVIA ALAM KHAN(2303.KHI.DEG.017)**

&

AQSA TAUHEED(2303.KHI.DEG.011)

ASSIGNMENT NO :4.4

Browse to:
tasks/4_microservices_development/day_4_best_practices/
app_that_doesnt_follow_best_practices/

Analyze the application - which Microservice best practices does it not follow?

Think about what needs to be improved first. Have a look at the areas_for_improvement.txt file for hints.

Improve the application.

SOLUTION :

1- The logs shouldn’t written to a file, but to the container output.

For this in main.py we modified it as :

```
# Configure logging to output to the container
root_logger = logging.getLogger()
root_logger.setLevel(logging.INFO)
stream_handler = logging.StreamHandler()
root_logger.addHandler(stream_handler)
```

By configuring the logger with a StreamHandler and a custom log message format, the logs will be output to the container console instead of being written to a file.

2- It should be stateless, so that:

- it can easily be restarted without loss of data,
- it is easy to spawn multiple instances of the application

To ensure the application can be easily restarted without loss of data, the SQLite database is stored in the data directory, which is mounted as a volume in the container.

```
if app.config['DEBUG']:
    app.config['TODO_DB'] = '/app/data/todo-dev.db'
else:
    app.config['TODO_DB'] = '/app/data/todo.db'
```

This means that during development (when DEBUG is True), the database file todo-dev.db will be used, and in production (when DEBUG is False), the database file todo.db will be used.

In the docker-compose.yml file, the ./data directory is mounted as a volume to the /app/data directory inside the container:

volumes: - ./data:/app/data

This ensures that any changes made to the database file within the container are persisted in the data directory on the host machine, allowing the application to be easily restarted without losing the data stored in the database.

3 - Requirements installation should be moved from runtime to build time.

Requirements installation is moved to the build time in the Dockerfile.

```
# Use an official Python runtime as the base image
FROM python:3.8-slim-buster

# Set the working directory in the container
WORKDIR /app

# Copy the application code to the container
COPY . .

# Install the application dependencies
RUN pip install --no-cache-dir -r requirements.txt

```

4 -App should be able to be executed both during development, with debugging enabled, and in production, with debugging disabled.

In main.py file ： The application can be executed with debugging enabled during development and with debugging disabled in production by setting the FLASK_ENV environment variable.

```
app = Flask(__name__)
app.config['DEBUG'] = os.environ.get('FLASK_ENV') == 'development'
```

5 -The application should be built in such a way that the database can easily be replaced (development with production instance).

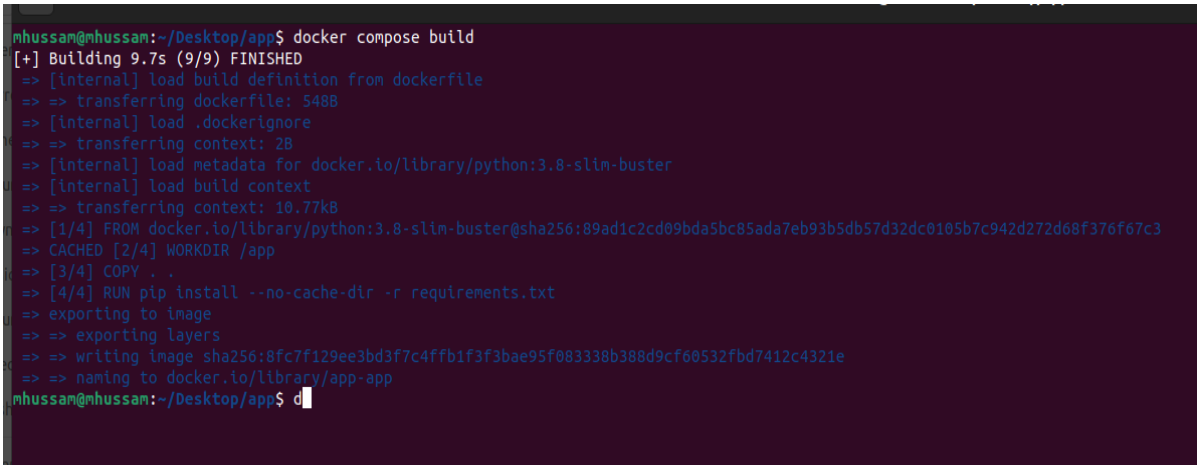
The database can be easily replaced by modifying the TODO_DB environment variable in the Flask application.

```python
# Set the database path based on the environment
if app.config['DEBUG']:
    app.config['TODO_DB'] = '/app/data/todo-dev.db'
else:
    app.config['TODO_DB'] = '/app/data/todo.db'
```

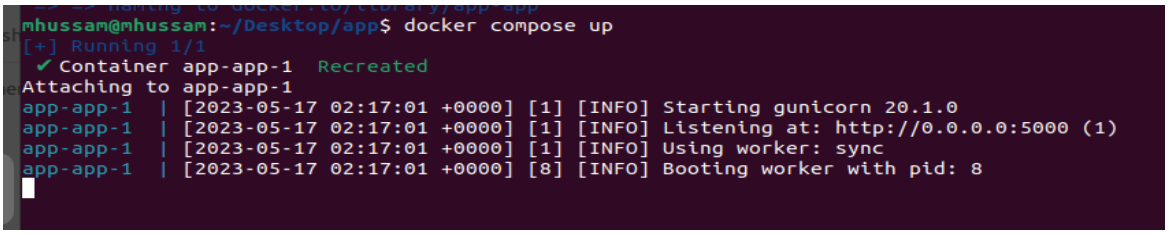
This allows you to use different database files depending on the environment. By changing the value of DEBUG, you can switch between the development and production databases without modifying the code.

OUTPUT:

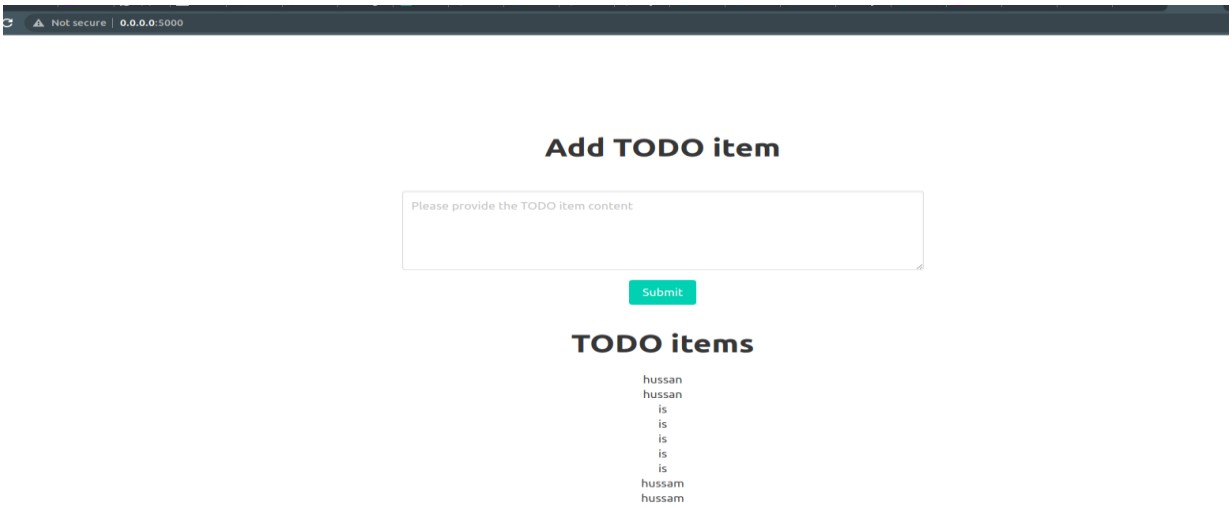
After setting all files , I used commnad docker compose build :



And now for running I used docker compose up :



Output on port:5000 as :



My data will never be lost if I stop port , when I restore it ,It will show todo items