NAME: MAVIA ALAM KHAN(2303.KHI.DEG.017)

PAIRING WITH **:Mohammad Hussam (2303.KHI.DEG.020)**

&

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.

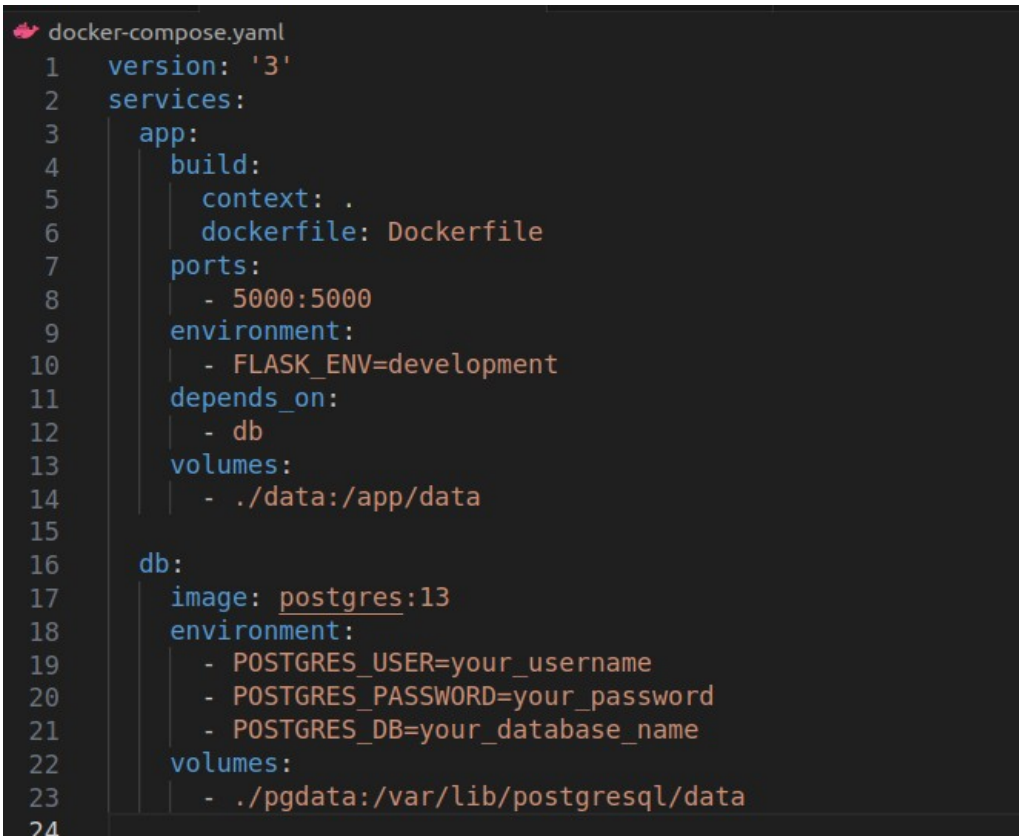
SOLUTION :

Here are the files :

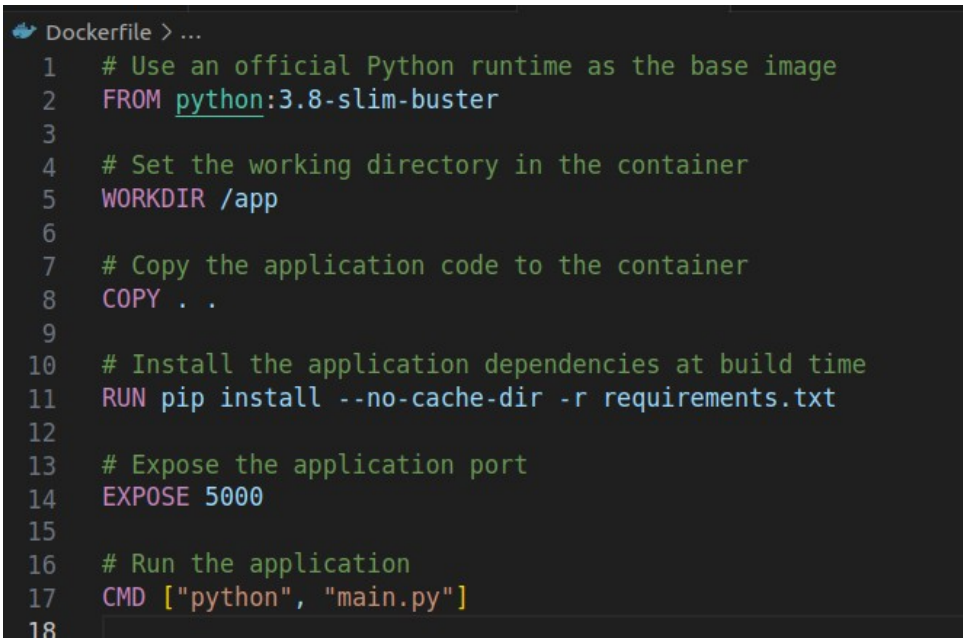
Main.py:

```python
import os
import logging
import psycopg2
from flask import Flask, render_template, request

app = Flask(__name__)
app.config['DEBUG'] = os.environ.get('FLASK_ENV') == 'development'
app.config['DB_HOST'] = os.environ.get('DB_HOST', 'db')
app.config['DB_PORT'] = int(os.environ.get('DB_PORT', 5432))
app.config['DB_NAME'] = os.environ.get('DB_NAME', 'your_database_name')
app.config['DB_USER'] = os.environ.get('DB_USER', 'your_username')
app.config['DB_PASSWORD'] = os.environ.get('DB_PASSWORD', 'your_password')

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

def add_todo_item(item):
    conn = psycopg2.connect(
        host=app.config['DB_HOST'],
        port=app.config['DB_PORT'],
        dbname=app.config['DB_NAME'],
        user=app.config['DB_USER'],
        password=app.config['DB_PASSWORD']
    )
    c = conn.cursor()
    c.execute('INSERT INTO todo (content) VALUES (%s)', (item,))
    conn.commit()
    conn.close()

def get_todo_items():
    conn = psycopg2.connect(
        host=app.config['DB_HOST'],
        port=app.config['DB_PORT'],
        dbname=app.config['DB_NAME'],
        user=app.config['DB_USER'],
        password=app.config['DB_PASSWORD']
    )
    c = conn.cursor()
    c.execute('SELECT content FROM todo')
    todo_items = [item[0] for item in c.fetchall()]
    conn.close()
    return todo_items

@app.route("/", methods=["GET", "POST"])
def main():
    if request.method == "POST":
        content = request.form["content"]
        add_todo_item(content)

    todo_items = get_todo_items()

    return render_template("index.html", todo_items=todo_items)

if __name__ == "__main__":
    if app.config['DEBUG']:
        app.run(host="0.0.0.0", port=int(os.environ.get("PORT", 5000)), debug=True)
    else:
        app.run(host="0.0.0.0", port=int(os.environ.get("PORT", 5000)), debug=False)
```

Docker-compose.yaml:



Docker file :



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

```
def add_todo_item(item):
    conn = psycopg2.connect(
        host=app.config['DB_HOST'],
        port=app.config['DB_PORT'],
        dbname=app.config['DB_NAME'],
        user=app.config['DB_USER'],
        password=app.config['DB_PASSWORD']
    )
    c = conn.cursor()
    c.execute('INSERT INTO todo (content) VALUES (%s)', (item,))
    conn.commit()
    conn.close()

def get_todo_items():
    conn = psycopg2.connect(
        host=app.config['DB_HOST'],
        port=app.config['DB_PORT'],
        dbname=app.config['DB_NAME'],
        user=app.config['DB_USER'],
        password=app.config['DB_PASSWORD']
    )
    c = conn.cursor()
    c.execute('SELECT content FROM todo')
    todo_items = [item[0] for item in c.fetchall()]
    conn.close()
    return todo_items
```

The application is stateless as it connects to a PostgreSQL database for data storage. This allows easy restarts without losing data and enables the spawning of multiple instances of the application

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

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

```
 9
10   # Install the application dependencies at build time
11   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.

if the DEBUG mode is enabled, the application runs with debug mode and the specified host and port. If the DEBUG mode is disabled, the application runs without debug mode and uses the specified host and port.

```
if __name__ == "__main__":
    if app.config['DEBUG']:
        app.run(host="0.0.0.0", port=int(os.environ.get("PORT", 5000)), debug=True)
    else:
        app.run(host="0.0.0.0", port=int(os.environ.get("PORT", 5000)), debug=False)
```

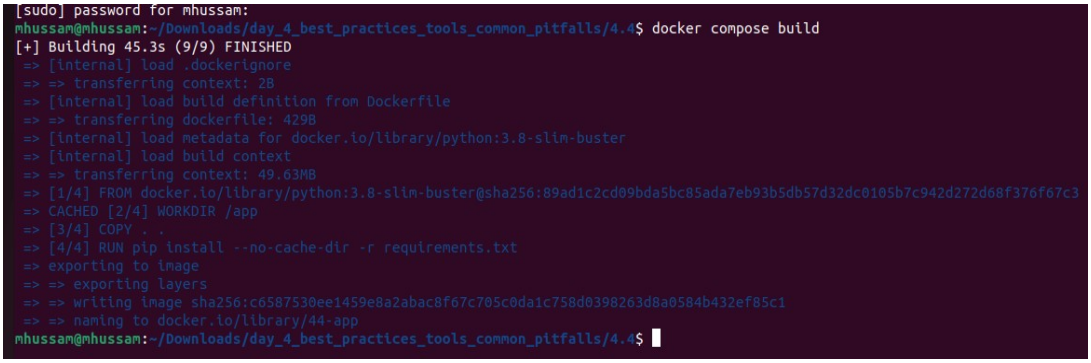
5 -The application should be built in such a way that the database can easily be replaced (development with production instance).

provided configuration for the PostgreSQL database in the Docker Compose file allows for easy replacement of the database between development and production instances.

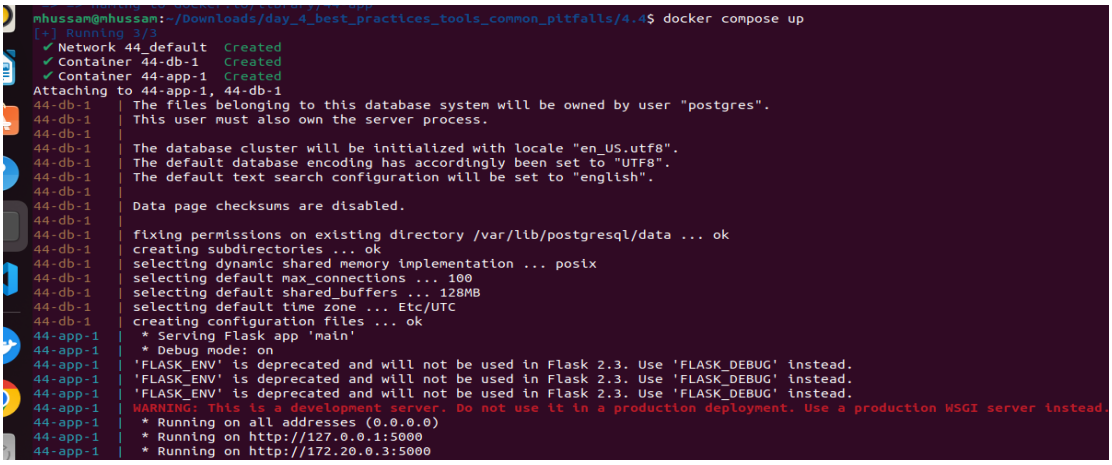
```
  db:
    image: postgres:13
    environment:
      - POSTGRES_USER=your_username
      - POSTGRES_PASSWORD=your_password
      - POSTGRES_DB=your_database_name
    volumes:
      - ./pgdata:/var/lib/postgresql/data
```

OUTPUT:

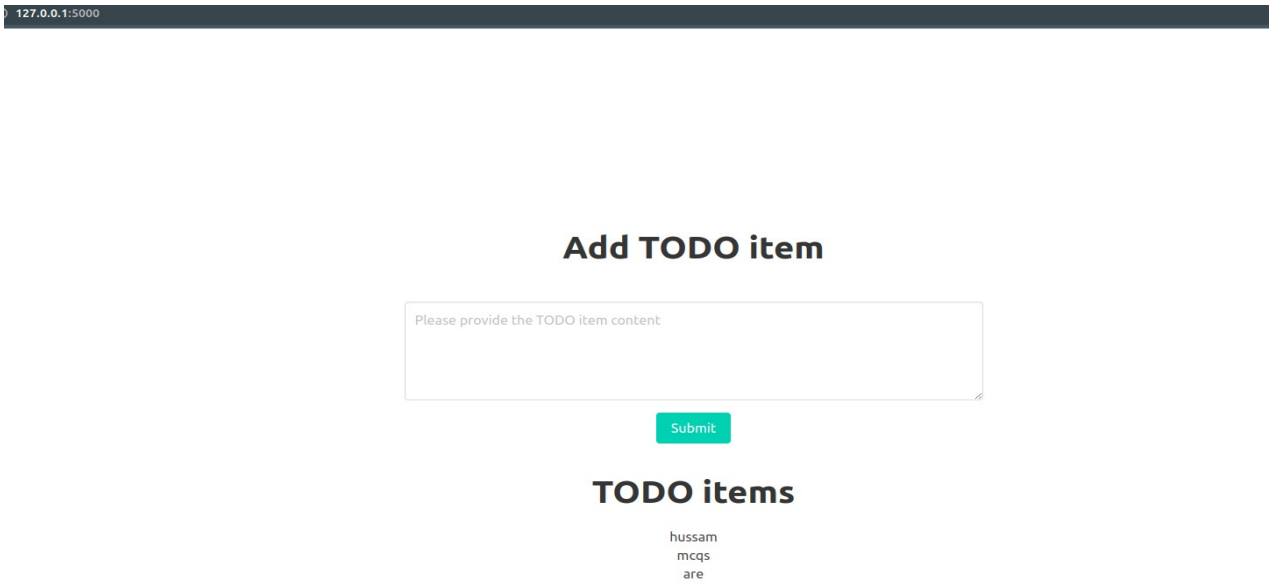
After setting all files , I used commnad docker compose build :



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