Full Stack Application with FastAPI and PostgreSQL

Submitted by: Shafeena Farheen

Program: M.Tech in Artificial Intelligence and Data Science

Role: AI Engineer/ML Engineer

---

1. Introduction

This project is about creating a full-stack application using FastAPI for the backend and PostgreSQL as the database. The application includes functionalities to load initial data into the database, provide a RESTful API for data interaction, and deploy the application using Docker.

2. Technologies and Tools Used

- **Programming Language:** Python 3.9
- **Web Framework:** FastAPI
- **Database:** PostgreSQL
- **ORM:** SQLAlchemy
- **Containerization:** Docker, Docker Compose
- **API Documentation:** Swagger UI

3. Project Setup and Execution

3.1 Directory Structure

lua

```
Task_Project/
|-- app/
|   |-- __init__.py
|   |-- database.py
|   |-- models.py
|   |-- schemas.py
|   |-- crud.py
```

```
|   |-- main.py
|   |-- load_data.py
|-- scripts/
|   |-- wait-for-it.sh
|-- data.json
|-- Dockerfile
|-- docker-compose.yml
|-- requirements.txt
|-- README.md
|-- .env
```

3.2 Docker Setup

docker-compose.yml:

yaml

```yaml
version: '3.8'

services:
  web:
    build: .
    ports:
      - "8000:80"
    depends_on:
      - db

  db:
    image: postgres:13
    environment:
      POSTGRES_USER: uniqueuser
      POSTGRES_PASSWORD: uniquepassword
      POSTGRES_DB: uniquedatabase
    ports:
      - "5432:5432"
```

Dockerfile:

```
FROM python:3.9-slim

WORKDIR /app
```

```
COPY . /app

COPY scripts/wait-for-it.sh /app/
RUN chmod +x /app/wait-for-it.sh

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

CMD ["sh", "-c", "/app/wait-for-it.sh db:5432 -- uvicorn app.main:app --host 0.0.0.0 --port 80"]
```

3.3 Application Configuration

The database configuration uses SQLAlchemy to connect to the PostgreSQL database using environment variables for credentials.

3.4 Loading Initial Data

This script loads initial data from a JSON file into the PostgreSQL database. It ensures that the necessary tables are created and populates them with the data.

3.5 Running the Application

Build and start the containers:
cmd

```
docker-compose up --build
```

Load initial data into the database:
cmd

```
docker-compose exec web sh -c "PYTHONPATH=/app python app/load_data.py"
```

1. **Access the API documentation:** Open a web browser and navigate to http://localhost:8000/docs.

3.6 API Endpoints

- **GET /documents:** Retrieve all documents.
- **POST /documents:** Add a new document.

Example Responses

GET /documents:

css

```
[ {    "type": "bank-draft",    "title": "Bank Draft",    "position": 0,    "id": 1  },  {    "type":
"bill-of-lading",    "title": "Bill of Lading",    "position": 1,    "id": 2  },  {    "type": "invoice",
"title": "Invoice",    "position": 2,    "id": 3  },  {    "type": "bank-draft-2",    "title": "Bank Draft
2",    "position": 3,    "id": 4  },  {    "type": "bill-of-lading-2",    "title": "Bill of Lading 2",
"position": 4,    "id": 5  }]
```

POST /documents Response:

json

```
{
  "type": "string",
  "title": "string",
  "position": 0,
  "id": 6
}
```

3.7 Documentation

README.md:

yaml

```
# Full Stack Application with FastAPI and PostgreSQL

## Setup Instructions

1. Clone the repository:
   git clone <repository_url>
   cd Task_Project

2. Create a .env file:
   POSTGRES_USER=myuser
```

POSTGRES_PASSWORD=mypassword
POSTGRES_DB=mydatabase

3. Build and start the Docker containers:
docker-compose up --build

4. Load initial data:
docker-compose exec web sh -c "PYTHONPATH=/app python app/load_data.py"

5. Access the API documentation:
Open your browser and navigate to http://localhost:8000/docs.

Project Structure

- app/: Contains the FastAPI application code.
- scripts/: Contains utility scripts.
- data.json: Contains initial data to be loaded into the database.
- Dockerfile: Docker configuration for the web service.
- docker-compose.yml: Docker Compose configuration.
- requirements.txt: Python dependencies.
- README.md: Project documentation.

cd C:\Users\safjag2011\Downloads\Task_Project

cd C:\Users\safjag2011\Downloads\Task_Project

docker-compose up --build

docker-compose exec web sh -c "PYTHONPATH=/app python app/load_data.py"

Terminal Look:

C:\Users\safjag2011\Downloads\Task_Project>docker-compose up --build # (Docker containers start up, logs appear...) # In another Command Prompt or terminal window: C:\Users\safjag2011\Downloads\Task_Project>docker-compose exec web sh -c "PYTHONPATH=/app python app/load_data.py" Data loaded successfully!

Verify docs:

http://localhost:8000/docs

GET /documents: Retrieve all documents.

- Click on the GET /documents endpoint.
- Click the Try it out button.
- Click the Execute button to retrieve the documents.


```
[
  { "id": 1, "type": "bank-draft", "title": "Bank Draft", "position": 0 },
  { "id": 2, "type": "bill-of-lading", "title": "Bill of Lading", "position": 1 },
  { "id": 3, "type": "invoice", "title": "Invoice", "position": 2 },
  { "id": 4, "type": "bank-draft-2", "title": "Bank Draft 2", "position": 3 },
  { "id": 5, "type": "bill-of-lading-2", "title": "Bill of Lading 2", "position": 4 }
]
```


Summary

- **Initial data has been successfully loaded into the database.**
- **You can verify and interact with the data through the API endpoints available at http://localhost:8000/docs.**

POST /documents: Add a new document.

- Click on the POST /documents endpoint.
- Click the Try it out button.

Enter the new document data in the request body:
json


```
{
  "type": "receipt",
  "title": "Receipt",
  "position": 5
}
```


- Click the Execute button to add the new document.

Example Response:

json

```
{
  "id": 6,
  "type": "receipt",
  "title": "Receipt",
  "position": 5
}
```

4. Conclusion

The project successfully demonstrates the development of a full-stack application using FastAPI and PostgreSQL, containerized using Docker. The API provides endpoints to interact with the data, and initial data is successfully loaded into the database. The setup is documented and can be reproduced by following the instructions provided.

Additional Resources

For more details, please visit the API

FastAPI - Swagger UI

FastAPI - Swagger UI

FastAPI - Swagger UI

Frontend:
cd C:\Users\safjag2011\Downloads\Task_Project\document-cards
npm start

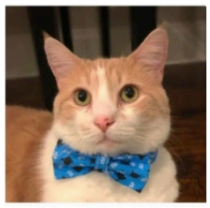
Display the content as 5 cards, 3 in the first row and 2 in the second row.
Assign a different thumbnail of your choice to each document type.

Display a placeholder spinner for each image that is loading.
Make the application so the cards can be reordered via drag and drop.
Make clicking on a card display the image as an overlay in the middle of the webpage.

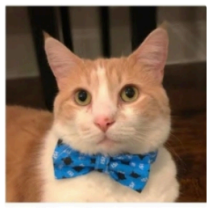
Bill of Lading 2

Bill of Lading

Invoice



Bank Draft 2



Bank Draft