AUTUMN INTERNSHIP PROJECT REPORT

FastAPI File Management using PostgreSQL, MinIO, and Caching

[Project Code: G19]

Report submitted by:

Shrabana Paul

B.S. in Data Science and Applications, IIT Madras

Email- 23f3003958@ds.study.iitm.ac.in

shrabanapaul9@gmail.com

For the successful completion of the internship in Data Science
(Tenure: 25th August 2025 - 31st October 2025)

Under the supervision of

Mr.Suprava Das
Assistant Software Developer
Institute of Data Engineering, Analytics and Science Foundation -
Technology Innovation Hub
Indian Statistical Institute, Kolkata

Report submitted to :

IDEAS - Institute of Data Engineering, Analytics and Science Foundation
Technology Innovation Hub
Indian Statistical Institute
Kolkata, West Bengal, India

Abstract

This project focuses on designing and implementing a robust and scalable **File Management System** using **FastAPI**, integrated with **PostgreSQL**, **MinIO**, and **FastAPI Caching**. The project was carried out as part of the Autumn Internship at the **Institute of Data Engineering, Analytics and Science Foundation [IDEAS-TIH], Indian Statistical Institute (ISI), Kolkata**. The objective was to create a backend framework capable of efficiently handling the entire lifecycle of data files—from upload and validation to storage, retrieval, and merging—while ensuring reliability, security, and scalability.

The developed system acts as a data ingestion and management pipeline for structured datasets (such as CSV and Excel files). When a user uploads a file through the FastAPI interface, its metadata is automatically stored in **PostgreSQL**, while the actual file is uploaded and preserved in **MinIO object storage**. The system supports real-time data merging, enabling users to combine multiple datasets based on a common identifier using **Pandas**, with the merged output temporarily cached in memory for faster access through **FastAPI Caching**.

This implementation demonstrates the integration of modern backend technologies to achieve seamless communication between APIs, databases, and cloud storage services. It showcases how modular and microservice-based architectures can simplify complex data management workflows commonly found in enterprise applications. By providing automated data validation, efficient file storage, and high-speed access mechanisms, this project contributes to the growing ecosystem of intelligent data engineering systems.

Keywords: FastAPI, PostgreSQL, MinIO, Caching, File Management System, API Integration, Data Engineering

Introduction

Project Background and Relevance

In today's data-driven environment, efficient file management systems are essential for handling large volumes of structured and semi-structured data. Many organizations face challenges in storing, organizing, and accessing datasets spread across multiple systems and formats. To address these challenges, this project—*"FastAPI File Management using PostgreSQL, MinIO, and Caching"*—was developed during the Autumn Internship at the **Institute of Data Engineering, Analytics and Science Foundation [IDEAS-TIH], Indian Statistical Institute (ISI), Kolkata**.

The system integrates **FastAPI** for developing RESTful APIs, **PostgreSQL** for structured metadata storage, **MinIO** for scalable object storage, and **FastAPI Caching** for temporary in-memory data handling. It automates the entire workflow of file upload, retrieval, merging, and caching—forming a foundation for building modern, scalable data management and ETL pipelines.

This project demonstrates how backend integration between APIs, databases, and object storage can streamline data engineering workflows, reduce redundancy, and ensure reliability in large-scale data processing systems.

Importance of MinIO for File Storage

In modern data engineering, object storage plays a key role in handling diverse and large datasets. **MinIO**, an open-source, S3-compatible storage system, was chosen for this project due to its ability to store raw data files efficiently and securely. It supports the **Extract-Load-Transform (ELT)** workflow, where data is preserved in its original format and transformed later as needed.

Benefits of using MinIO include:

1. **Data Preservation:** Maintains original uploaded files without modification.
2. **Scalability:** Handles large volumes of files efficiently.
3. **Flexibility:** Easily integrates with Python tools like Pandas for data processing.
4. **Security:** Provides access control and reliable object management.

Through this integration, the project delivers a unified, cloud-ready backend for managing and processing data files efficiently.

Project Objectives

The primary objectives of this internship project are:

1. **Develop a Robust File Management System**
   - Design and implement an API-driven file management system capable of handling CSV and Excel files.
   - Enable users to upload, validate, and retrieve files through a seamless FastAPI interface.

2. **Integrate Modern Backend Technologies**
   - Connect **FastAPI**, **PostgreSQL**, and **MinIO** to establish an efficient data management workflow.
   - Store metadata in PostgreSQL and raw data files securely in MinIO object storage.

3. **Implement Temporary Caching for Merged Datasets**
   - Use **FastAPI Caching** to temporarily store merged datasets in memory for faster data access.
   - Reduce repetitive computation by enabling short-term reuse of recently merged data.

4. **Automate Data Merging and Validation**
   - Perform file merging operations dynamically using **Pandas**, based on a common column (e.g., `customer_id`).
   - Validate file formats and ensure data integrity before and after merging.

5. **Ensure Scalability and Reproducibility**
   - Design the system to handle multiple users and large datasets efficiently.
   - Create a modular architecture that can be easily extended for future analytical or cloud-based integrations.

Methodology

This project focuses on building a robust and scalable **File Management System** using **FastAPI**, integrated with **PostgreSQL**, **MinIO**, and **FastAPI Caching**. The methodology involves multiple stages — from system design and environment setup to API implementation, database integration, and testing. The workflow ensures a smooth end-to-end process of file upload, storage, retrieval, merging, and caching for efficient backend management.

---

1 Overview

The project was developed to demonstrate how modern cloud-native backend services handle large data files with efficiency and scalability. The overall workflow includes:

1. Uploading files (CSV/Excel) through FastAPI.
2. Storing file metadata (name, format, upload time) in **PostgreSQL**.
3. Storing the actual files in **MinIO**, an open-source S3-compatible object storage system.
4. Merging selected files temporarily using **FastAPI Caching**.
5. Saving merged datasets permanently to MinIO for persistent storage.

The application follows a **modular design** — separating API logic, database connections, MinIO configuration, and caching mechanisms into distinct Python modules.

---

2 Endpoints Implemented

The following REST API endpoints were implemented:

1. **POST `/upload/` – File Upload**
    - Accepts user file uploads in CSV or Excel formats.
    - Extracts metadata (file name and format).
    - Stores the metadata in PostgreSQL and uploads the actual file to the MinIO bucket named `files`.
    - Returns a confirmation message on successful upload.

2. **GET `/files/` – Retrieve All Files**

   - Fetches and displays all stored files along with their IDs, names, and formats.
   - The information is retrieved directly from PostgreSQL.

3. **GET `/merge/` – Merge Two Files Temporarily**

   - Takes two file IDs as input.
   - Fetches both files from MinIO and loads them into **Pandas DataFrames**.
   - Merges the two datasets based on a common key (`customer_id`).
   - Stores the merged output temporarily in memory using **FastAPI Cache** and returns a **cache key** and a preview of the merged data.

4. **POST `/save-merged/` – Save Merged Dataset Permanently**

   - Accepts the cache key as input.
   - Retrieves the temporarily cached merged dataset.
   - Saves it permanently to MinIO as a new file.
   - Inserts its metadata (file name, format) into PostgreSQL and clears the cache entry.

These endpoints together demonstrate a complete backend pipeline for data ingestion, management, and processing.

---

3 Tools and Libraries Used

Tool / Library	Purpose
FastAPI	Backend API framework
Uvicorn	ASGI server to run the FastAPI app
PostgreSQL	Relational database to store file metadata
SQLAlchemy	ORM for database operations

MinIO	Object storage system for uploaded files
Pandas	Data manipulation and merging
FastAPI-Cache2	Temporary in-memory data storage
.env	Secure environment variable management

---

4 Workflow Summary

1. **Environment Setup:**
   The project environment was created using Python 3.12 and a virtual environment (`venv`).
   Dependencies were installed using `requirements.txt`.
2. **Database Integration:**
   PostgreSQL was connected using `SQLAlchemy`, and a table was created to store file metadata (file ID, name, format, and upload time).
3. **MinIO Configuration:**

A MinIO server was launched using PowerShell with:

```
minio server minio_data --console-address ":9090"
```

- A bucket named `files` was automatically created to store uploaded datasets.

4. **API Development:**
   Each endpoint was built in `main.py` using FastAPI, following clear function-based routing for upload, retrieval, merge, and save operations.

5. **Caching Mechanism:**
   Temporary merged datasets were stored using `InMemoryBackend` of **FastAPI-Cache2**, ensuring efficient retrieval before permanent storage.

6. **Testing and Validation:**

- The application was tested using the Swagger UI available at:
  http://127.0.0.1:8000/docs
- Successful file uploads and merges were verified through the MinIO Console at:
  http://localhost:9090

7. **Code Deployment and Repository:**
   All scripts were uploaded to a private GitHub repository for documentation and reproducibility:
   🔗 https://github.com/shrabanapaul9/fastapi_file_manager

---

5 Visual Workflow Summary

Below figure can be inserted to represent the process flow:

Flow Diagram :

```
User Upload → FastAPI → PostgreSQL (Metadata) + MinIO (File Storage) → Cache (Temporary Merge) → Save Merged File → MinIO
```

---

6 Summary

The system successfully demonstrates how FastAPI can serve as an efficient, modular, and scalable backend for data management.
It integrates seamlessly with **PostgreSQL** for relational storage, **MinIO** for object storage, and **FastAPI Caching** for temporary data persistence, ensuring high performance and modular code design.

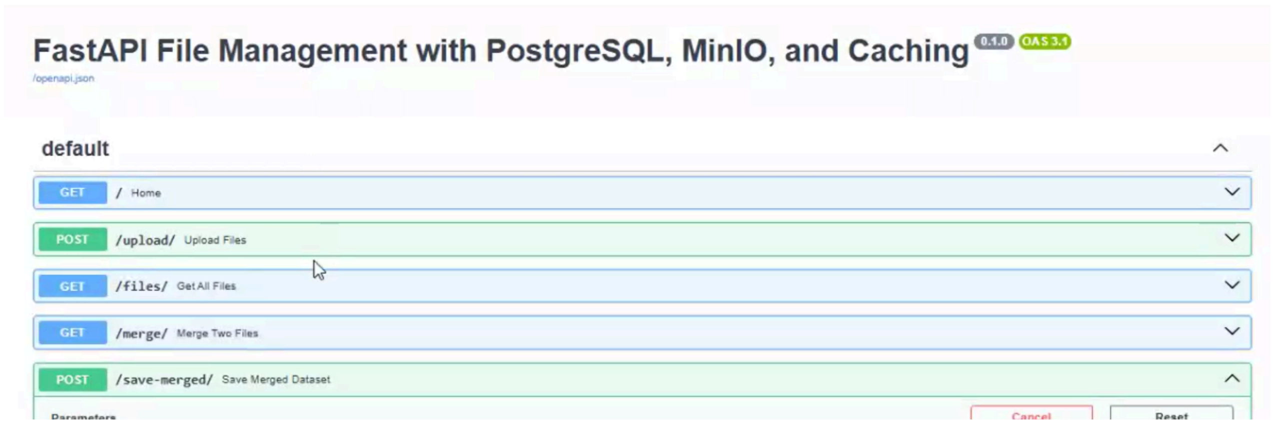
Data Analysis and Results

5.1 API Testing Environment

The testing was carried out locally using:

- **FastAPI's Swagger UI:** `http://127.0.0.1:8000/docs`
- **MinIO Console:** `http://localhost:9090`
- **PostgreSQL:** via pgAdmin and SQL shell for verifying database insertions

Each API endpoint was tested individually to ensure end-to-end functionality from data ingestion to storage.



FastAPI File Management with PostgreSQL, MinIO, and Caching 0.1.0 OAS 3.1

/openapi.json

default

GET / Home

POST /upload/ Upload Files

GET /files/ Get All Files

GET /merge/ Merge Two Files

POST /save-merged/ Save Merged Dataset

🖊️ Caption: "FastAPI Swagger Interface displaying all functional endpoints."

5.2 File Upload Verification

The `/upload/` endpoint successfully accepted multiple file uploads in CSV format.

Process Summary:

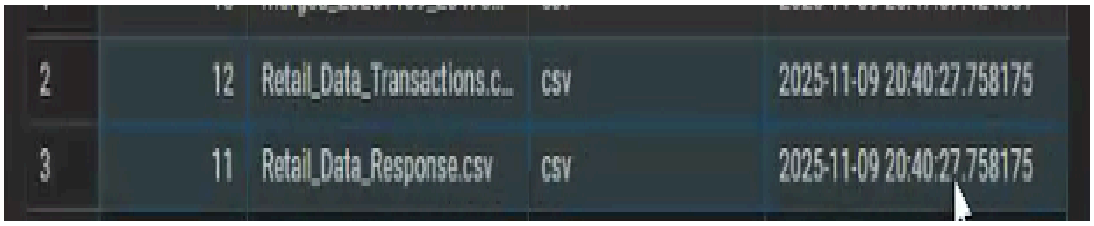
1. Selected two CSV files (`Retail_Data_Response.csv` and `Retail_Data_Transactions.csv`).
2. Uploaded both using Swagger interface.
3. The files were validated for extension and written to MinIO.
4. File metadata (ID, name, format) was stored in PostgreSQL.

API Response Example:

```
{
  "message": "Files uploaded successfully",

  "uploaded_files": [

    "Retail_Data_Response.csv",

    "Retail_Data_Transactions.csv"

  ]

}
```

Verification in PostgreSQL:

ID	File Name	Format
1	Retail_Data_Response.csv	csv
2	Retail_Data_Transactions.csv	csv



```
2    12  Retail_Data_Transactions.c...  csv    2025-11-09 20:40:27.758175
3    11  Retail_Data_Response.csv       csv    2025-11-09 20:40:27.758175
```

🖊️ Caption: “File metadata stored in PostgreSQL database.”

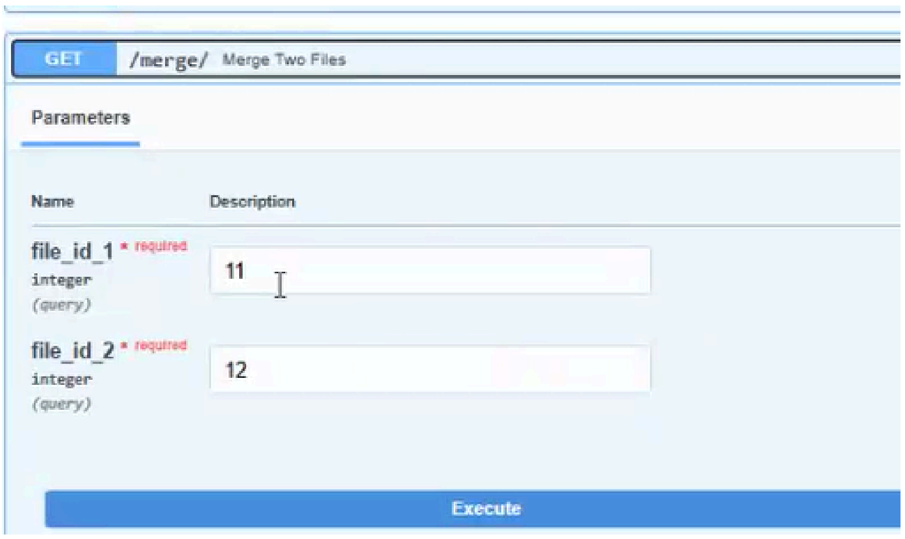
5.3 File Retrieval

The `/files/` endpoint retrieved all entries stored in PostgreSQL.

Sample Response:

```
{

  "files": [

    {"id": 11, "file_name": "Retail_Data_Response.csv", "file_format": "csv"},

    {"id": 12, "file_name": "Retail_Data_Transactions.csv", "file_format":
"csv"}

  ]
}
```

This validated successful linkage between the FastAPI backend and PostgreSQL database.



GET /merge/ Merge Two Files

Parameters

Name	Description
file_id_1 * required integer (query)	11
file_id_2 * required integer (query)	12

🖊 Caption: "List of stored files retrieved from PostgreSQL."

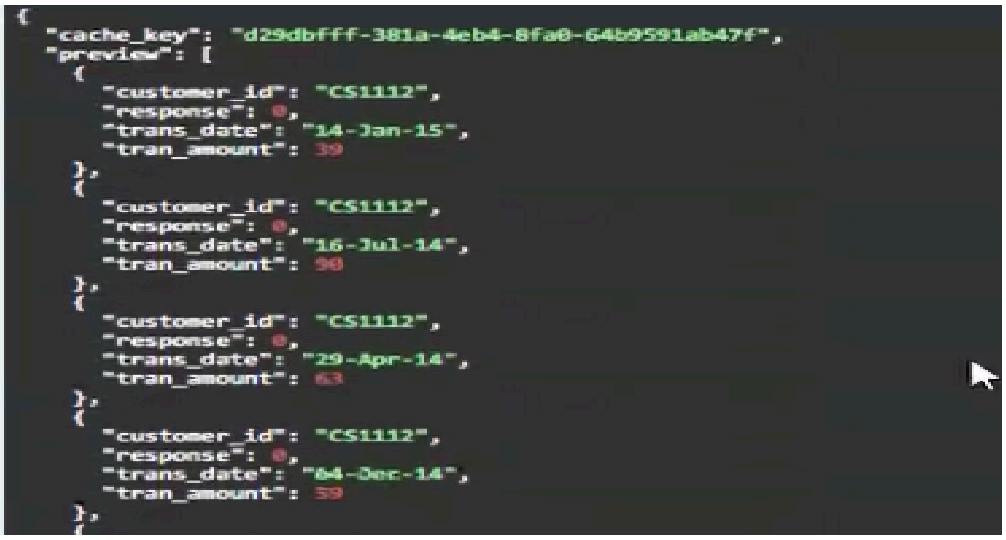
5.4 File Merging and Caching

The `/merge/` endpoint accepted two file IDs (`file_id_1=11` and `file_id_2=12`).
The system fetched both files from **MinIO**, read them using **pandas**, and merged them based on the common column `customer_id`.

A temporary merged dataset was generated and cached using **FastAPI-Cache2** (InMemoryBackend).
The cache key was automatically generated as a **UUID**.

Sample Response:

```
{

  "cache_key": "d29dbfff-381a-4eb4-8fa0-64b9591ab47f",

  "preview": [

    {"customer_id": "CS1112", "response": 0, "trans_date": "14-Jan-15",
"tran_amount": 39},

    {"customer_id": "CS1112", "response": 0, "trans_date": "16-Jul-14",
"tran_amount": 90},

    ...

  ]
}
```



```
{
  "cache_key": "d29dbfff-381a-4eb4-8fa0-64b9591ab47f",
  "preview": [
    {
      "customer_id": "CS1112",
      "response": 0,
      "trans_date": "14-Jan-15",
      "tran_amount": 39
    },
    {
      "customer_id": "CS1112",
      "response": 0,
      "trans_date": "16-Jul-14",
      "tran_amount": 98
    },
    {
      "customer_id": "CS1112",
      "response": 0,
      "trans_date": "29-Apr-14",
      "tran_amount": 63
    },
    {
      "customer_id": "CS1112",
      "response": 0,
      "trans_date": "04-Dec-14",
      "tran_amount": 59
    },
    {
```

🖉 Caption: "Merged dataset preview with generated cache key."

5.5 Saving the Merged Dataset Permanently

The `/save-merged/` endpoint was used to save the merged file permanently to **MinIO**.
It accepted the previously generated `cache_key`, retrieved the cached JSON data, and wrote it back as a new CSV file to the MinIO bucket `files`.

Sample Request Body:

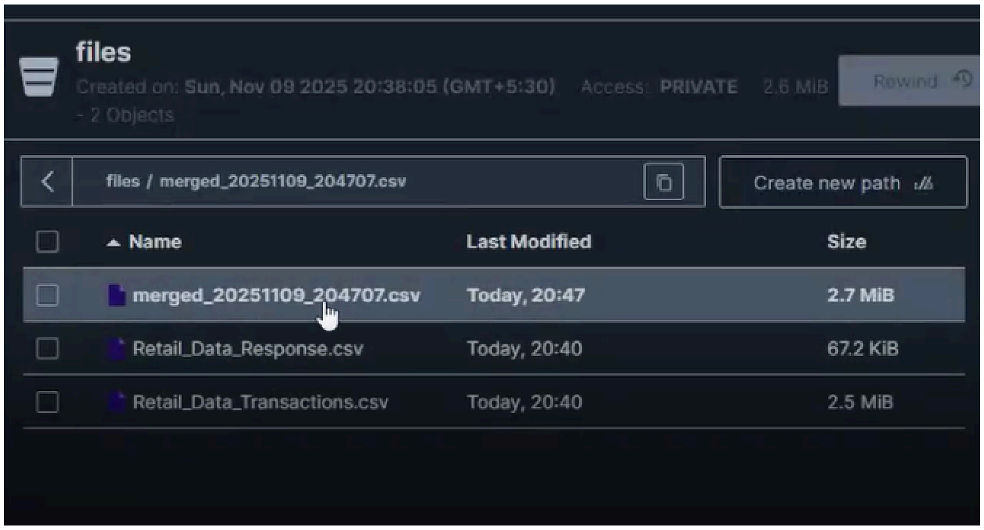
```
{

  "cache_key": "d29dbfff-381a-4eb4-8fa0-64b9591ab47f"

}
```

Sample Response:

```
{

  "message": "Merged dataset saved successfully!",

  "file_name": "merged_20251108_160300.csv"

}
```

Verification:

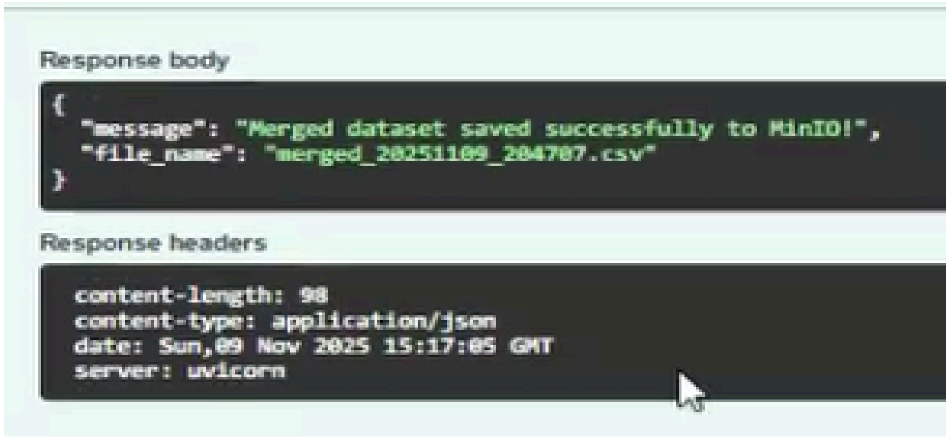
- The new merged file appeared in MinIO under the same bucket.
- The metadata for the new file was inserted into PostgreSQL with a new ID.



files
Created on: Sun, Nov 09 2025 20:38:05 (GMT+5:30) Access: PRIVATE 2.6 MiB - 2 Objects
Rewind
files / merged_20251109_204707.csv
Create new path

Name	Last Modified	Size
merged_20251109_204707.csv	Today, 20:47	2.7 MiB
Retail_Data_Response.csv	Today, 20:40	67.2 KiB
Retail_Data_Transactions.csv	Today, 20:40	2.5 MiB

🖉 Caption: “Permanent merged dataset stored in MinIO object storage.”



Response body

```
{
  "message": "Merged dataset saved successfully to MinIO!",
  "file_name": "merged_20251109_204707.csv"
}
```

Response headers

```
content-length: 98
content-type: application/json
date: Sun,09 Nov 2025 15:17:05 GMT
server: uvicorn
```

🖉 Caption: “Successful confirmation of merged dataset storage.”

5.6 System Workflow Verification

The entire process confirms that:

- Files can be uploaded, retrieved, and merged efficiently.
- Temporary caching improves performance during merge operations.
- Integration between FastAPI, PostgreSQL, and MinIO works seamlessly.

Final File Lifecycle:

```
Upload File → PostgreSQL (Metadata) + MinIO (Storage)

   ↓

Merge Files (Temporary Cache)

   ↓

Save Merged File → PostgreSQL + MinIO
```

Conclusion

The FastAPI File Management project successfully integrates API development, database management, cloud storage, and caching mechanisms. It provides a foundation for scalable backend applications where file handling and processing are automated. The project demonstrates the efficiency of modern microservice-based architectures in managing data workflows and can be extended to handle large-scale enterprise data operations.

Future improvements may include authentication, version control for uploaded files, and deployment on cloud platforms like AWS or Azure.

Appendices

Appendix 1: References

The following sources and tools were referred to during the development of this project:

1. FastAPI Documentation — https://fastapi.tiangolo.com
2. MinIO Documentation — https://min.io/docs
3. PostgreSQL Official Docs — https://www.postgresql.org/docs

4. FastAPI Cache2 GitHub Repository — https://github.com/long2ice/fastapi-cache

5. SQLAlchemy Documentation — https://docs.sqlalchemy.org

Appendix 2: GitHub Repository

All source codes, environment configuration files, and supporting scripts are available at:
🔗 **GitHub Link:** https://github.com/shrabanapaul9/fastapi_file_manager

---

Appendix 3: Supporting Documents

- A copy of this report, presentation slides, and all input datasets are stored in the same GitHub repository for evaluation and verification.

- Additional datasets used during testing (e.g., `Retail_Data_Response.csv` and `Retail_Data_Transactions.csv`) are included in the `/uploads` directory.

- MinIO server configurations and `.env` file details are provided in the `/app` folder for reference.