INTERNSHIP PROJECT REPORT

IDEAS – Institute of Data Engineering, Analytics and Science Foundation
Indian Statistical Institute, Kolkata

FastAPI-Based File Management and Dataset Processing System Using PostgreSQL, MinIO, and In-Memory Caching

Submitted by

Debangshu Sadhukhan

M.Sc. in Data Science (1st Year)

Ramakrishna Mission Residential College, Narendrapur

Under the Guidance of

Suprava Das

Internship Duration:
25th August 2025 – 31st October 2025

Report Submitted To:
IDEAS – Institute of Data Engineering, Analytics and Science Foundation, ISI Kolkata

1. Abstract

This report documents the design, implementation, and evaluation of a backend service to manage and process structured data files using FastAPI, PostgreSQL, MinIO, and in-memory caching. The service supports secure upload of CSV and Excel files, metadata management in PostgreSQL, object storage using MinIO, merging of two datasets with configurable join types, temporary caching of merged results for quick retrieval, and permanent storage of merged datasets back to MinIO. Implementation focuses on robustness (validation, error handling), maintainability (clear API contracts and modular code), and reproducibility (documentation, requirements, and GitHub repository). The system was tested with representative datasets, and performance observations and future improvements are discussed.

2. Introduction

2.1 Background and Motivation

Data engineers and analysts regularly work with tabular datasets in CSV and Excel formats. Real-world workflows often require consolidating disparate files, performing joins, and storing processed results. Traditional ad-hoc approaches (local folders and scripts) lack scalability, auditability, and concurrent access control. Cloud-native object storage (S3-compatible) and API-driven services provide a robust alternative. This project implements such a service using FastAPI (for API layer), MinIO (for object storage), PostgreSQL (for metadata), and an in-memory caching backend for quick temporary storage of merged results.

2.2 Relevance and Use Cases

Use-cases include:

- Analytical teams needing a reproducible pipeline for merging customer and transaction files.
- Small organizations wanting S3-like storage without managed cloud costs (MinIO on-prem).
- Teaching and demonstration of full-stack data engineering patterns (APIs + object storage + DB).

2.3 Technologies and Tools

Key technologies:

- **FastAPI** – API framework (async-friendly, automatic docs at /docs).
- **PostgreSQL** – metadata persistence.
- **MinIO** – S3-compatible object storage.
- **Pandas** – data loading, cleaning, merging.
- **fastapi-cache** (InMemoryBackend) – temporary caching of merged datasets.
- **psycopg2** – PostgreSQL driver.
- **uvicorn** – ASGI server for development.

3. Project Objectives

Primary objectives:

1. Design a REST API for file upload, listing, merging, and saving merged datasets.
2. Store file metadata (filename, format, size, uploader, timestamp) in PostgreSQL.
3. Store file content in MinIO (object store) with the original filename as object key.
4. Provide a merge endpoint that reads two objects from MinIO into Pandas, merges them by a specified column and join type, caches the merged result temporarily, and returns a preview and cache key.
5. Provide a save endpoint to persist cached merged results permanently to MinIO and register metadata in PostgreSQL.
6. Ensure robustness with validation, error handling, and safe temporary file handling.

4. Methodology

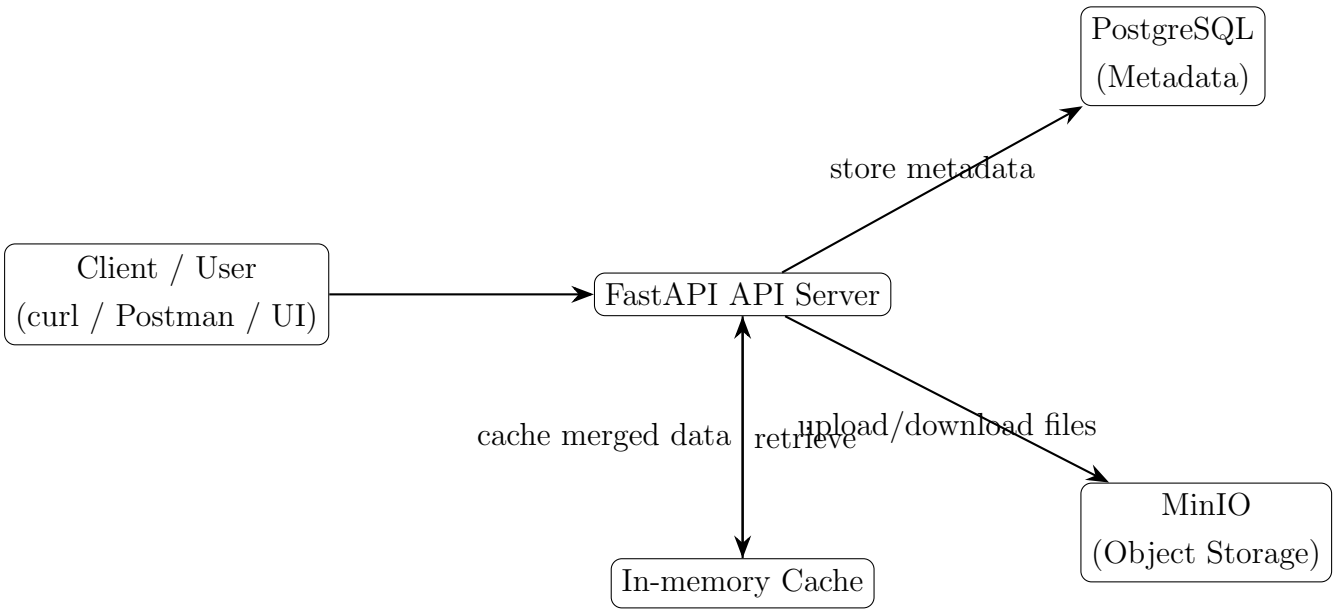
4.1 Design Principles

The system follows separation of concerns:

- API layer (FastAPI) handles request validation, authorization (if added), and orchestration.

- Storage layer (MinIO) is responsible for storing raw files and final outputs.
- Metadata layer (PostgreSQL) stores searchable metadata and supports audit.
- Caching layer stores transient merged outputs to avoid recomputation.

4.2 System Architecture



4.3 Database Schema

The primary table ‘files2‘ schema used in PostgreSQL:

Column	Type / Description
id	SERIAL PRIMARY KEY
name	VARCHAR – original filename (object key)
format	VARCHAR – ’csv’ / ’xlsx’
size	INTEGER – size in bytes
description	TEXT – optional description
uploaded_by	VARCHAR – uploader name
file_path	VARCHAR – MinIO object key
status	VARCHAR – ’active’/’merged’/’deleted’
upload_time	TIMESTAMP – insertion time

4.4 API Contracts

Endpoints (brief):

- **POST /upload** – multipart upload: file, uploaded_by, description. Returns file id.
- **GET /files** – list metadata of active files.
- **GET /merge** – params: file1_id, file2_id, join_column (optional), join_type (inner/left/right/outer). Returns preview and cache_key.
- **POST /save_merged** – form param: cache_key. Persists merged dataset to MinIO and inserts metadata.
- **DELETE /delete/file_id** – removes object from MinIO and deletes DB row.

4.5 Key Implementation Decisions

- Use of in-memory cache (fastapi-cache InMemoryBackend) for the assignment: simple, no external dependencies. For production, Redis is recommended.
- Store files in MinIO with original filename. To avoid collisions in production, prefix filenames with a UUID.
- Convert DataFrame to JSON for caching to ensure backend portability.
- Use ‘psycopg2‘ (synchronous) to keep the implementation straightforward. For fully async support, ‘asyncpg‘ + SQLAlchemy async could be used later.

5. Implementation Details

5.1 Environment

Essential environment variables (example ‘.env‘):

```
POSTGRES_DB=files_db
POSTGRES_USER=postgres
POSTGRES_PASSWORD=postgrespassword
POSTGRES_HOST=localhost
POSTGRES_PORT=5432

MINIO_ENDPOINT=localhost:9000
MINIO_ACCESS_KEY=minioadmin
MINIO_SECRET_KEY=minioadmin
MINIO_BUCKET=files
```

5.2 Requirements

Install required packages (requirements.txt):

```
fastapi
uvicorn[standard]
pandas
psycopg2-binary
minio
python-multipart
fastapi-cache
python-dotenv
openpyxl
```

5.3 Selected Code Snippets

Below are compact, key excerpts representative of the full implementation.

Upload endpoint (simplified)

```
@app.post("/upload")
async def upload_file(file: UploadFile = File(...),
                      uploaded_by: str = Form("Debangshu"),
                      description: str = Form("Uploaded via FastAPI")):
    if not file.filename.endswith(('.csv', '.xlsx')):
        raise HTTPException(status_code=400, detail="Only CSV or Excel files
            ↪ allowed")
    contents = await file.read()
    _minio_put_bytes(file.filename, contents, content_type=file.content_type)
    cursor.execute("""INSERT INTO files2 (name, format, size, description,
                      uploaded_by, file_path, status, upload_time)
                      VALUES (%s,%s,%s,%s,%s,%s,%s,NOW()) RETURNING id""",
                   (file.filename, ext, len(contents), description,
                    uploaded_by, file.filename, "active"))
    conn.commit()
    new_id = cursor.fetchone()[0]
    return {"message": "File uploaded", "file_id": new_id}
```

Merge endpoint (simplified)

```
@app.get("/merge")
```

```
async def merge_files(file1_id: int, file2_id: int,
                      join_column: Optional[str] = "customer_id",
                      join_type: Optional[str] = "inner"):
    # fetch metadata, get bytes from MinIO, read into DataFrames
    merged_df = pd.merge(df1, df2, on=join_column_norm, how=join_type)
    cache_key = str(uuid.uuid4())
    backend = FastAPICache.get_backend()
    await backend.set(cache_key, merged_df.to_json(orient="records"), expire
        ↪ =600)
    return {"cache_key": cache_key, "preview": merged_df.head().to_dict(orient
        ↪ ="records")}
```

Save merged endpoint (simplified)

```
@app.post("/save_merged")
async def save_merged(cache_key: str = Form(...)):
    backend = FastAPICache.get_backend()
    merged_json = await backend.get(cache_key)
    merged_df = pd.read_json(io.StringIO(merged_json), orient="records")
    tmp = tempfile.NamedTemporaryFile(delete=False, suffix=".csv")
    merged_df.to_csv(tmp.name, index=False)
    minio_client.fput_object(MINIO_BUCKET, os.path.basename(tmp.name), tmp.name
        ↪ )
    cursor.execute(... insert metadata ...)
    await backend.delete(cache_key)
    os.remove(tmp.name)
    return {"message": "Merged saved"}
```

6. Data Analysis and Results

6.1 Testing Dataset

To test functionality, sample datasets were used:

- `customers.csv` – columns: customer_id, name, email
- `orders.csv` – columns: order_id, customer_id, amount

6.2 Test Steps

1. Upload both files via POST /upload.

2. Verify metadata entries in PostgreSQL (SELECT * FROM files2).
3. Call GET /merge?file1_id=1&file2_id=2&join_column=customer_id.
4. Observe preview and receive cache_key.
5. Use POST /save_merged with cache_key to persist merged result.
6. Confirm merged file exists in MinIO and metadata inserted.

6.3 Representative Output

Preview returned by merge (example):

```
[
  {"customer_id": 101, "name": "Raj", "order_id": 1001, "amount": 250.0},
  {"customer_id": 102, "name": "Amit", "order_id": 1002, "amount": 340.0}
]
```

6.4 Performance Observations

- Local MinIO uploads were fast for small files (tens of MB).
- Cache retrieval on the same process was near-instant.
- For multi-instance deployment, replace in-memory cache with Redis to avoid cache misses across instances.

7. Conclusion

This project delivered a functional backend system that demonstrates file lifecycle management: upload, metadata storage, object storage, dataset merging, temporary caching, and permanent storage of merged outputs. The architecture and codebase are modular and ready for extension (authentication, Redis cache, async DB access). The project strengthened practical skills in FastAPI, MinIO integration, relational database handling, and data engineering workflows.

7.1 Limitations

- In-memory cache is process-local and not suitable for multiple instances.
- No authentication implemented — currently endpoints are open.
- Collision handling for identical filenames is simplistic (overwrites object).

7.2 Future Work

- Integrate Redis-backed fastapi-cache2 for distributed caching.
- Add OAuth2/JWT authentication and role-based access control.
- Add file versioning and unique object keys (UUID prefix).
- Add automated tests (pytest) and CI pipeline for the repository.
- Build a lightweight frontend (React) to interact with APIs.

Appendices

Appendix A: Installation and Run Instructions

Prerequisites: Python 3.10+, PostgreSQL, MinIO (or S3-compatible storage).

Steps:

1. Create and activate virtual environment:

```
python -m venv venv
& venv\Scripts\activate # Windows PowerShell
pip install -r requirements.txt
```

2. Ensure PostgreSQL is running and create database:

```
psql -U postgres -c "CREATE DATABASE files_db;"
```

3. Start MinIO (example Windows command):

```
"C:\Users\HP\Downloads\minio.exe" server C:\minio-data --console-
    ↪ address ":9001"
```

4. Set environment variables in a '.env' file (see earlier example).

5. Run the FastAPI app:

```
uvicorn main:app --reload --host 0.0.0.0 --port 8000
```

6. Open automatic docs at `http://localhost:8000/docs`.

Appendix B: GitHub Repository

Project repository: `https://github.com/debangshu9183/FastAPI-Based-File-Management-and-Da`

Appendix C: Sample API Usage (curl)

- Upload:

```
curl -X POST "http://localhost:8000/upload" \
  -F "file=@/path/to/customers.csv" \
  -F "uploaded_by=Debangshu" \
  -F "description=Customers file"
```

- Merge:

```
curl "http://localhost:8000/merge?file1_id=1&file2_id=2&join_column=
    ↪ customer_id&join_type=inner"
```

- Save merged:

```
curl -X POST -F "cache_key=YOUR_CACHE_KEY" "http://localhost:8000/
    ↪ save_merged"
```

Appendix D: Sample SQL for Table Creation

```
CREATE TABLE files2 (
    id SERIAL PRIMARY KEY,
    name VARCHAR NOT NULL,
    format VARCHAR NOT NULL,
    size INTEGER,
    description TEXT,
    uploaded_by VARCHAR,
    file_path VARCHAR,
    status VARCHAR DEFAULT 'active',
    upload_time TIMESTAMP DEFAULT NOW()
);
```

Appendix E: Full Example - main.py (Reference)

Below is an abbreviated but complete reference outline of the FastAPI application. For the full code, see the GitHub repository.

```
# main.py (outline)
import os, io, uuid, tempfile
from fastapi import FastAPI, UploadFile, File, Form, HTTPException
from minio import Minio
```

```
import pandas as pd
import psycopg2
from fastapi_cache import FastAPICache
from fastapi_cache.backends.inmemory import InMemoryBackend
# ... load env, init DB, init MinIO, create bucket, init cache

app = FastAPI()

@app.on_event("startup")
async def startup():
    FastAPICache.init(InMemoryBackend(), prefix="fastapi-cache")

@app.post("/upload")
async def upload_file(...):
    # validate, read bytes, upload to minio, insert metadata, return id

@app.get("/files")
def list_files():
    # query files2 and return list

@app.get("/merge")
async def merge_files(...):
    # fetch files from minio, load into pandas, normalize, merge, cache JSON

@app.post("/save_merged")
async def save_merged(cache_key: str = Form(...)):
    # retrieve cached JSON, write temp CSV, upload to minio, insert metadata,
        ↪ delete cache

@app.delete("/delete/{file_id}")
async def delete_file(file_id: int):
    # remove object from minio, delete db row
```

Note: The full production-ready file includes additional error handling, logging, and edge-case checks.

Acknowledgement: I thank IDEAS — ISI Kolkata and my mentor Suprava Das for guidance during the internship.