# **Data Upload and Query API**

This project is a simple backend system built with FastAPI that allows users to upload CSV files, validates the data against a predefined schema, stores the validated data in an SQLite database, and provides REST API endpoints to query and manage the stored information. It also includes comprehensive API activity logging.

**Features**

* **CSV File Upload:** Accept CSV files via a dedicated API endpoint.
* **Robust Data Validation:** Utilizes **Pandera** for strict schema validation, checking for:
  + Correct data types (integers, strings, booleans, dates)
  + Missing values (nullability checks)
  + Uniqueness constraints (e.g., for IDs)
  + Value ranges (e.g., age between 0 and 120)
  + Regex patterns (e.g., for email format)
* **Database Storage:** Stores validated CSV row data as JSON objects in an **SQLite** database using SQLAlchemy.
* **REST API Endpoints:**
  + POST /upload-csv/: Upload and process CSV files.
  + GET /data/: Retrieve all stored data with pagination.
  + GET /data/?filter\_column={column\_name}&filter\_value={value}: Filter retrieved data by any column within the stored JSON payload.
  + GET /data/{item\_id}: Retrieve a single data record by its unique ID.
  + DELETE /data/{item\_id}: Delete a single data record by its unique ID.
* **API Activity Logging:** A custom middleware logs every API request, including request ID, endpoint, method, status code, and processing time, to a dedicated log file (logs/api\_activity.log).
* **Interactive API Documentation:** Automatically generated Swagger UI documentation (OpenAPI) for easy testing and exploration of endpoints.

**Technologies Used**

* **Backend Framework:** [FastAPI](https://fastapi.tiangolo.com/) (Python)
* **Database:** [SQLite](https://www.sqlite.org/index.html) (via [SQLAlchemy ORM](https://www.sqlalchemy.org/))
* **Data Validation:** [Pandera](https://pandera.readthedocs.io/en/stable/)
* **Data Manipulation:** [Pandas](https://pandas.pydata.org/)
* **ASGI Server:** [Uvicorn](https://www.uvicorn.org/)
* **Dependency Management:** pip
* **Version Control:** Git / GitHub

## **Setup Instructions**

Follow these steps to get the project up and running on your local machine.

### **Prerequisites**

* Python 3.8+ (recommend Python 3.10 or newer)
* Git

### **1. Clone the Repository**

First, clone this GitHub repository to your local machine:

### **2. Create and Activate a Virtual Environment**

It's highly recommended to use a virtual environment to manage project dependencies.

# Create a virtual environment

python -m venv venv

# Activate the virtual environment

# On Windows:

.\venv\Scripts\activate

# On macOS/Linux:

source venv/bin/activate

You should see (venv) prepended to your terminal prompt, indicating the virtual environment is active.

### **3. Install Dependencies**

With the virtual environment activated, install all required packages from requirements.txt:

pip install -r requirements.txt

## **Running the Application**

Once dependencies are installed, you can start the FastAPI application using Uvicorn:

uvicorn main:app --reload

* The --reload flag enables auto-reloading of the server when code changes are detected, which is useful for development.
* The application will be accessible at http://127.0.0.1:8000.

## **API Endpoints**

You can interact with the API using tools like curl, Postman, or directly via the interactive Swagger UI.

### **Access Swagger UI**

Open your web browser and go to: http://127.0.0.1:8000/docs

This interface provides a complete overview of all endpoints, allows you to execute requests directly, and view responses.

### **Endpoints Details**

#### **1. Upload CSV File**

* **Endpoint:** POST /upload-csv/
* **Description:** Accepts a CSV file, validates its structure and content using the defined Pandera schema, and stores the valid rows in the database.
* **Request Body:** multipart/form-data with a File parameter named file.
* **Success Response (200 OK):**

{

"message": "Successfully uploaded and stored X rows.",

"filename": "your\_file.csv"

}

* **Error Responses (400 Bad Request, 500 Internal Server Error):** Returns detailed messages for invalid file formats, empty files, or data validation failures (e.g., missing values, type mismatches, regex failures).

#### **2. Retrieve All Uploaded Data (with Filtering and Pagination)**

* **Endpoint:** GET /data/
* **Description:** Retrieves all stored data items. Supports pagination and powerful filtering by any column within the JSON data.
* **Query Parameters:**
  + skip (int, optional, default: 0): Number of items to skip for pagination.
  + limit (int, optional, default: 100): Maximum number of items to return.
  + filter\_column (str, optional): The name of the column in the CSV data to filter by (e.g., name, age, is\_active).
  + filter\_value (str, optional): The value to filter the specified column by.
* **Example Usage:**
  + http://127.0.0.1:8000/data/ (Get all data)
  + http://127.0.0.1:8000/data/?skip=5&limit=10 (Pagination)
  + http://127.0.0.1:8000/data/?filter\_column=name&filter\_value=Alice (Filter by name)
  + http://127.0.0.1:8000/data/?filter\_column=age&filter\_value=30 (Filter by age)
  + http://127.0.0.1:8000/data/?filter\_column=is\_active&filter\_value=True (Filter by boolean status)
* **Success Response (200 OK):**
* JSON

{

"message": "Data retrieved successfully.",

"data": [

{

"id": 2,

"original\_filename": "sample.csv",

"row\_number": 2,

"data": {

"id": 2,

"name": "Bob",

"age": 24,

"email": "bob@example.com",

"is\_active": false,

"signup\_date": "2022-11-01"

},

"uploaded\_at": "2025-07-01T15:06:26.981358"

}

],

"count": 1

}

#### **3. Retrieve Single Data Item by ID**

* **Endpoint:** GET /data/{item\_id}
* **Description:** Retrieves a single data record using its unique database ID.
* **Path Parameter:** item\_id (integer): The unique ID of the data record.
* **Example Usage:** http://127.0.0.1:8000/data/2
* **Success Response (200 OK):**

{

"message": "Data item retrieved successfully.",

"data": {

"id": 2,

"original\_filename": "sample.csv",

"row\_number": 2,

"data": {

"id": 2,

"name": "Bob",

"age": 24,

"email": "bob@example.com",

"is\_active": false,

"signup\_date": "2022-11-01"

},

"uploaded\_at": "2025-07-01T15:06:26.981358"

}

}

#### **4. Delete Single Data Item by ID**

* **Endpoint:** DELETE /data/{item\_id}
* **Description:** Deletes a single data record using its unique database ID.
* **Path Parameter:** item\_id (integer): The unique ID of the data record to delete.
* **Example Usage:** http://127.0.0.1:8000/data/2
* **Success Response (200 OK):**

"message": "Data item with ID 2 deleted successfully."

}

* **Error Response (404 Not Found):** If the item\_id does not exist.

## **Project Structure**

data-upload-api/

├── venv/ # Python virtual environment (ignored by Git)

├── main.py # Main FastAPI application, defines endpoints and middleware

├── database.py # SQLAlchemy database setup and session management

├── models.py # SQLAlchemy ORM models for database tables

├── validation.py # Pandera DataFrame schema and validation logic

├── logger.py # Custom logging utility for API activity

├── data/

│ └── sample.csv # Example CSV file for testing uploads

├── logs/ # Directory for API activity logs (ignored by Git)

│ └── api\_activity.log # Log file generated by the application

├── requirements.txt # List of Python dependencies

└── .gitignore # Specifies files/directories to be ignored by Git

## **Logging**

All API requests are logged to logs/api\_activity.log. Each log entry includes:

* **Timestamp**
* **Log Level** (INFO)
* **Request ID:** A unique UUID for tracking a single request's lifecycle.
* **Endpoint:** The API path being accessed.
* **Method:** HTTP method (GET, POST, DELETE).
* **Status:** HTTP status code of the response (for completed requests).
* **Message:** A descriptive message about the request's state (started, finished, or specific actions/errors).
* **Processing Time:** How long the request took (for finished requests).

## **Sample Data**

A sample.csv file is provided in the data/ directory for easy testing of the upload functionality. Its structure adheres to the CSVDataSchema defined in validation.py.

Example sample.csv content:

id,name,age,email,is\_active,signup\_date

1,Alice,30,alice@example.com,True,2023-01-15

2,Bob,24,bob@example.com,False,2022-11-01

3,Charlie,35,charlie@example.com,True,2024-03-20

4,Diana,28,diana@example.com,True,2023-07-01

5,Eve,29,eve@example.com,False,2024-01-10

## **Possible Enhancements**

* **Error Handling Refinements:** More granular custom exception handling.
* **Authentication & Authorization:** Implement token-based authentication (e.g., OAuth2 with JWT) to secure API endpoints.
* **More Advanced Filtering/Querying:** Support for AND/OR conditions, range queries, or fuzzy search.
* **Update Endpoint:** Add a PUT or PATCH endpoint to modify existing data records.
* **Asynchronous Operations:** Explore using FastAPI's background tasks or external task queues (e.g., Celery) for large CSV processing.
* **Database Migration Tool:** Integrate a tool like Alembic for managing database schema changes.
* **Containerization:** Provide a Dockerfile for easy deployment with Docker.
* **Unit and Integration Tests:** Write automated tests to ensure code quality and functionality















