Data Upload and Query API with AI Assistant and Authentication

This is a robust backend system built with FastAPI that allows users to upload CSV files, validates the data, stores it in an SQLite database, and exposes REST API endpoints to query the stored data. It includes HTTP Basic Authentication for secure access, an AI assistant powered by Gemini for natural language queries, and comprehensive logging of all API activity.

Features

- **CSV File Upload:** Upload CSV files to dynamically named tables in an SQLite database.
- **Data Validation:** Basic validation for missing values and incorrect row lengths during CSV upload. Empty cells are stored as NULL.
- **SQLite Database:** Uses a file-based SQLite database (data.db) for data persistence.
- **REST API Endpoints:**
  - POST /upload_csv/{table_name}: Uploads a CSV file and stores its data.
  - GET /tables: Lists all tables created from CSV uploads.
  - GET /data/{table_name}: Retrieves all data from a specified table.
  - GET /data/{table_name}/query: Allows filtering, limiting, and offsetting data from a specified table using query parameters.
  - POST /ask_data_assistant: An AI assistant endpoint to answer natural language questions about the uploaded data.
- **HTTP Basic Authentication:** Securely protects all API endpoints, requiring a username and password for access.
- **AI Assistant Integration:** Leverages the Gemini API to provide intelligent answers based on your uploaded data's schema and sample content.
- **API Activity Logging:** Logs all API requests (method, path, client IP) and response statuses to api_activity.log and the console.
- **Automatic API Documentation (Swagger UI):** Interactive API documentation is automatically generated and available at /docs.

Technologies Used

- **Backend Framework:** FastAPI (Python)
- **Web Server:** Uvicorn
- **Database:** SQLite
- **File Handling:** python-multipart

- **Password Hashing:** passlib[bcrypt]
- **AI Integration:** requests (for Gemini API calls)
- **Data Processing:** Standard Python csv module

Setup and Installation

1. Clone or Download the Project:
   Save the main.py file to a directory of your choice (e.g., my_api_project).
2. Create a Virtual Environment (Recommended):
   It's good practice to use a virtual environment to manage project dependencies.
   python -m venv venv

3. **Activate the Virtual Environment:**
   - **Windows:**
     .\venv\Scripts\activate

   - **macOS/Linux:**
     source venv/bin/activate

4. Create requirements.txt:
   Create a file named requirements.txt in your project directory with the following content:
   fastapi==0.111.0
   uvicorn==0.30.1
   python-multipart==0.0.9
   passlib[bcrypt]==1.7.4
   bcrypt==3.2.0
   requests==2.32.3

5. Install Dependencies:
   Navigate to your project directory (where main.py and requirements.txt are located) in your terminal and install the required packages:
   pip install -r requirements.txt

   If pip doesn't work, try pip3.
6. Configure Gemini API Key:
   Open your main.py file and locate the line:
   GEMINI_API_KEY = ""

   Replace the empty string "" with your actual Gemini API Key. You can obtain one

from Google AI Studio: https://aistudio.google.com/app/apikey. **For production, always load this from environment variables.**

Running the Application

1. Start the FastAPI Server:
   From your project directory in the activated virtual environment, run:
   uvicorn main:app --reload

   - main: Refers to the main.py file.
   - app: Refers to the FastAPI() instance named app inside main.py.
   - --reload: This flag automatically restarts the server when you make changes to your code, which is useful for development.
2. Access the API Documentation:
   Once the server starts, open your web browser and go to:
   http://127.0.0.1:8000/docs

   This will display the interactive Swagger UI, where you can explore and test all the API endpoints.

API Endpoints and Usage

All endpoints, except for the initial access to /docs, are protected by HTTP Basic Authentication.

Authentication

When you access any protected endpoint (e.g., http://127.0.0.1:8000/tables), your browser will prompt you for a username and password.

- **Default Credentials:**
  - **Username:** testuser
  - **Password:** testpassword
  - **OR**
  - **Username:** admin
  - **Password:** adminpass

1. POST /upload_csv/{table_name}

- **Description:** Uploads a CSV file and stores its data in a new or existing SQLite table.
- **Authentication:** Required (HTTP Basic Auth).
- **Parameters:**
  - table_name (Path Parameter): The desired name for the database table.

        - **Input Examples:** users, products, orders, my_data
        - *Note: This name will be sanitized to be a valid SQL name.*
    - file (File Upload): The CSV file to upload.
        - **Input:** Select a .csv file from your local file system using the "Choose File" button in Swagger UI.
- **Example (using Swagger UI):**
    1. Go to http://127.0.0.1:8000/docs.
    2. Expand the POST /upload_csv/{table_name} endpoint.
    3. Click "Try it out".
    4. In the table_name field, enter a name like my_data.
    5. Click "Choose File" and select your CSV file.
    6. Click "Execute".
- **Sample my_data.csv content:**
  name,age,city
  Alice,30,New York
  Bob,24,Los Angeles
  Charlie,35,Chicago

2. GET /tables

- **Description:** Retrieves a list of all user-defined tables currently stored in the database.
- **Authentication:** Required (HTTP Basic Auth).
- **Parameters:** None.
- **Example (using Swagger UI):**
    1. Go to http://127.0.0.1:8000/docs.
    2. Expand the GET /tables endpoint.
    3. Click "Try it out".
    4. Click "Execute".
    - *Expected Response:* {"tables": ["my_data"]} (if you uploaded my_data.csv)

3. GET /data/{table_name}

- **Description:** Retrieves all data from the specified table.
- **Authentication:** Required (HTTP Basic Auth).
- **Parameters:**
    - table_name (Path Parameter): The name of the table to retrieve data from.
        - **Input Examples:** my_data, orders
        - *Note: This must be the name of an existing table.*
- **Example (using Swagger UI):**

1. Go to http://127.0.0.1:8000/docs.
2. Expand the GET /data/{table_name} endpoint.
3. Click "Try it out".
4. In the table_name field, enter my_data.
5. Click "Execute".
   - *Expected Response (JSON):* All rows from your my_data table.

4. GET /data/{table_name}/query

- **Description:** Queries data from the specified table with optional filters, limit, and offset. Filters are applied as key=value pairs.
- **Authentication:** Required (HTTP Basic Auth).
- **Parameters:**
  - table_name (Path Parameter): The name of the table to query.
    - **Input Examples:** my_data, orders
    - *Note: This must be the name of an existing table*.
  - limit (Query Parameter, Optional): Maximum number of rows to return.
    - **Input Examples:** 10, 50, 100 (Default)
  - offset (Query Parameter, Optional): Number of rows to skip for pagination.
    - **Input Examples:** 0 (Default), 10, 20
  - filters (Query Parameter, Optional): A comma-separated string of key=value pairs for filtering.
    - **Input Examples (comma-separated key=value pairs):**
      - city=New York
      - age=30
      - city=New York,age=30 (to combine filters)
      - For the orders table: status=Completed
      - For the orders table: total_amount=150.75
- **Example (using Swagger UI):**
  1. Go to http://127.0.0.1:8000/docs.
  2. Expand the GET /data/{table_name}/query endpoint.
  3. Click "Try it out".
  4. In the table_name field, enter my_data (or orders).
  5. In the filters input field, enter your desired filters as key=value pairs separated by commas.
  6. Optionally, adjust limit and offset.
  7. Click "Execute".
- **Example (Direct URL in browser):**
  http://127.0.0.1:8000/data/my_data/query?city=New%20York&age=30&limit=1

(Note: %20 is the URL encoding for a space).

5. POST /ask_data_assistant

- **Description:** Asks the AI assistant a natural language question about the uploaded data. The assistant will try to answer based on the schema and sample data from your tables.
- **Authentication:** Required (HTTP Basic Auth).
- **Parameters:**
  - question (Request Body, Optional): The natural language question to ask.
    - **Input Examples:**
      - How many products are in the Electronics category?
      - What is the average price of items in the Stationery category?
      - List all customers who have pending orders.
      - What is the total amount for all completed orders?
      - Give me the schema for the 'orders' table.
      - What tables do I have?
      - *If left empty, the assistant will prompt you to provide a question.*
- **Example (using Swagger UI):**
  1. Go to http://127.0.0.1:8000/docs.
  2. Expand the POST /ask_data_assistant endpoint.
  3. Click "Try it out".
  4. In the question field, type your natural language question.
  5. Click "Execute".

Logging

All API requests and their responses are logged to a file named api_activity.log in the same directory as main.py. Logs are also printed to the console where the uvicorn server is running.

Database File

The SQLite database file data.db will be created automatically in the same directory as main.py when the application starts for the first time or when you upload your first CSV.

Notes on Additional Features Implemented

This project goes beyond the core requirements by implementing several key enhancements:

- **HTTP Basic Authentication:** A secure authentication layer has been integrated to protect all API endpoints, ensuring that only authorized users with valid usernames and passwords can access the data and functionalities.
- **AI Assistant Integration:** A dedicated /ask_data_assistant endpoint has been added, leveraging the Gemini API. This allows users to interact with their uploaded data using natural language queries, providing a more intuitive and accessible way to gain insights. The assistant dynamically fetches table schemas and sample data to provide relevant context to the AI model.
- **Enhanced Query Filtering:** The GET /data/{table_name}/query endpoint has been improved to accept dynamic filters through a single filters query string parameter (e.g., filters=key1=value1,key2=value2). This makes the API more flexible and user-friendly, especially when interacting via the Swagger UI.
- **Comprehensive API Logging:** Middleware has been implemented to automatically log every incoming API request and its corresponding response status. This provides a detailed audit trail of API activity, which is crucial for monitoring, debugging, and security.
- **Automatic API Documentation:** FastAPI's built-in Swagger UI (/docs) is fully utilized, providing interactive and automatically generated documentation for all API endpoints. This simplifies testing and understanding of the API for developers.