CSV modification and Query Server (by Ziqi Wei)

Docker Hub Repository Link: https://hub.docker.com/repositories/williamzqwei

Overview and Project Description

The CSV Query Server is a web-based API service designed to interact with CSV data via HTTP requests. The service allows users to query data using SQL-like query strings and modify the data with SQL-like commands, such as INSERT, DELETE, and UPDATE. The server is implemented in Python using the Flask framework and is dockerized for easy deployment and scaling.

The project is structured with a clear separation of concerns, dividing the responsibilities into distinct modules for managing CSV files (**CSVFileManager**), parsing queries (**QueryParser**), filtering data (**DataFilter**), and modifying data (**DataModifier**). At its core, the CSVDatabase class integrates these components, providing a high-level interface for query processing and data manipulation.

Server Description

The system supports the following basic operations:

- Query: Retrieve data from the CSV file that matches specific conditions.
- Insert: Add new rows to the CSV file.
- Delete: Remove rows from the CSV file based on specific conditions.
- Update: Modify existing rows in the CSV file based on specific conditions.

Design Considerations

- Modularity: The system is designed with modularity in mind. Each class has a single responsibility, making the system easy to maintain and extend.
- Object-Oriented Design: The project utilizes an object-oriented approach to encapsulate related data and behavior, promoting code reuse and abstraction.
- Error Handling: Robust error handling is implemented to ensure the system behaves predictably and provides meaningful feedback to the user.
- Data Integrity: Changes to the data follows an immediate write-back policy. The system ensures that modifications are either fully applied or not at all, maintaining data consistency.

Error Handling

The API returns a status code of 400 for any malformed queries or modifications, accompanied by a descriptive error message to aid the user in correcting the issue. The system is tested with various edge cases and malformed data inputs to ensure it can handle unexpected or 'dirty' data gracefully. The system is designed to be robust against various failure modes, including invalid input and file I/O errors. Exception handling and logging provide insights into the system's state and behavior.

Verification and Validation Strategy

Unit tests are written for each component to verify their behavior in isolation. Integration tests ensure that the components work together as expected. Validation is performed via Postman tests and manual API testing.

Coding Style/Quality Programming Habits

The code follows PEP 8 style guidelines for Python, and best programming practices are adhered to, such as using meaningful variable names, writing concise and clear comments, and avoiding deep nesting.

Packaging (Dockerization)

The service is packaged as a Docker container, making it platform-independent and simplifying deployment and scaling. The Dockerfile specifies all the necessary steps to build the image, and the image is hosted on Docker Hub for public access.

Lesson Learned

- Design: Emphasized modular design for clear responsibility distribution, resulting in streamlined development and maintenance. Each module should be developed, tested, and maintained with a clear focus
- Development and Testing: Followed a development-first approach with immediate unit testing, leading to an iterative and robust build-out. Integration and API testing were pivotal before deployment.
- Docker Deployment: The shift to Docker revealed the complexities of containerized environments. Utilized Docker's port handling to replace Nginx, learning the importance of leveraging built-in features for efficient solutions.

Future Consideration

- Performance:
  - Implement multi-threading for processing query; introduce lock mechanism to protect shared data during modification
  - The system can be further optimized for performance, minimizing disk I/O by caching data in memory and writing changes at specified intervals (which is my initial implementation discarded due to the time constraint)
- Safety:
  - User authentication: Integrate user authentication using OAuth or JWT to ensure secure access.
  - SQL injection check: Implement protections against SQL injection to protect against malicious inputs
  - HTTPS: transition to HTTPS for data encryption to boost overall security
- Database: Use a more robust database solution to replace the CSV storage, enhancing scalability and query efficiency

Appendix:

Running the server_s Docker Image

This document provides instructions for pulling and running the server_s Docker image, which contains a Flask application designed to handle queries and modifications on CSV data.

Pulling the Image

Open your terminal and pull the server_s image from Docker Hub using the following command:
docker pull williamzqwei/server_s:latest

Running the Container with Custom CSV Data

To run the server_s container with your own CSV file, you need to mount the CSV file into the container. Replace /path/to/your/csvfile.csv with the absolute path to your CSV file on your host machine.

docker run -d -p 9527:5000 -p 7259:5000 -v /path/to/your/csvfile.csv:/app/data/data.csv server_s

This command will start the Docker container with your CSV file mounted, allowing the Flask application to interact with it directly.

Accessing the Application

After the container starts, the application will be accessible from your browser or any HTTP client on the following URLs:

- Query endpoint: http://localhost:9527/?query=<query_string>
- Modification endpoint: http://localhost:7259/?job=<modification_string>

Replace <query_string> with your actual query and <modification_string> with your actual modification command.

Stopping the Container

To stop the running container, use the following command:
docker stop server_s_instance

Removing the Container

If you wish to remove the container after stopping it, use the command:
docker rm server_s_instance

Additional Notes

- Make sure no other applications are running on ports 9527 and 7259 on your machine to avoid port conflicts.
- If you are running the container on a server or a machine that's not localhost, replace localhost with the appropriate IP address or hostname in the URLs provided above.