CS411 Final Project

24-Fall

Overview

This assignment tasks you with developing a web application that serves as both a technological and a professional challenge. You will construct a robust backend using Flask, SQLite, and SQLAlchemy, implement fundamental account management, and engage with at least one external API to fetch and manipulate data. The objective is to provide you with practical experience in API usage, data storage, and the subtleties of working within an established codebase.

Imagine being new to a software development team: you are placed in a scenario akin to walking into a room with strangers, handed a legacy codebase that is dense and perhaps poorly documented. Your challenge is to deconstruct this existing code and reassemble it to achieve new functionalities. This simulation is designed to mirror real-world situations where you must quickly adapt and innovate within the constraints of pre-existing systems.

Teams

You will work in teams of three or four. The teams will be assigned randomly. This simulates a real-world software development scenario where you will not be in a position to choose which of your colleagues to work with.

When you submit your project, you will be asked to rate whether your teammates were "professional." Did they communicate proactively? Did they follow through on their commitments? Your peer rating in the project and on the homeworks will contribute 10% to your overall grade in the course. The hope is that everyone will earn those points.

The expectation is that all team members will contribute roughly equal amounts. If the peer rating causes us to suspect otherwise, we will look at the git logs. If it becomes clear that not all team members were participating, we reserve the right to adjust grades accordingly.

- Therefore, if something happens to your git history such that, e.g. all commits wound up being from one person, please document that somewhere in your submission.

Submission Guidelines

- **Due Date:**
    - The assignment will be **due at 11:59PM on the final day of the semester, Tuesday December 10th**. We will be going **by git history** rather than gradescope submissions.
- **What to submit:**

    - Similar to HW3, everyone will be required to submit a link to a github repo / branch / pull request and a description of what you would like us to grade if necessary.
    - The repo should include:
        - Source code for your app
        - `Dockerfile`
        - `.env` file template (with actual secrets removed)
        - `requirements.txt` / `requirements.lock` files
        - A `README` file describing:
            - What the application does at a high level
            - A description of each route (example on ed discussion):
                - Route Name and Path
                - Request Type
                    - `GET`, `POST`, `PUT`, `DELETE`
                - Purpose
                - Request Format
                    - `GET` parameters
                    - `POST` / `PUT` / `DELETE` body
                - Response Format
                    - JSON keys and value types
                - Example
                    - Request in the form of JSON body or cURL command
                    - Associated JSON response

Code Reuse

- You are **encouraged to reuse** as much of **meal_max / playlist** as you can. **You don't have to cite anything** and you should copy-pasta as much as you can. **You may start with meal_max and replace** the **functionality as required** rather than starting from scratch and copying functionality over.
    - You are **not allowed to plagiarize from another team**'s project

Emphasis on Code Hygiene and Complexity

Code Hygiene:

- As you navigate and modify the legacy code, maintaining code hygiene becomes paramount. You are expected to write clear, maintainable code that adheres to industry standards and best practices. This includes proper use of comments, consistent coding style, and logical structuring.

Interacting with Non-Trivial Codebases:

- This project will expose you to a complex codebase that requires you to apply critical thinking and problem-solving skills extensively. The ability to understand and efficiently modify existing code is a valuable skill in software engineering, as it is common to join projects mid-development or take over legacy systems.

Requirements

- Secure Password Storage
  - **Database:**
    - Create a SQLite table to store usernames, salts, and hashed passwords. You can implement this using either SQLAlchemy directly or raw SQLite commands. Using SQLAlchemy is recommended for ease of code re-use.
  - **Routes**:
    - `/login`: Checks the provided password against the stored hash. This route *doesn't need to handle actual user sessions*.
    - `/create-account`: Allows new users to register by providing a username and password.
    - `/update-password`: Enables users to change their password.
- Functionality
  - **API Model**: Develop a model that interacts with at least one external API. The model should store relevant data from the API response in memory.
  - **Application (App)**:
    - **Health Check Route**: Implement a route to verify that your app is running.
    - **API Interaction**: Create at least five routes that utilize the API model to perform different functions. These routes should return JSON blobs appropriate for use by another application.
- Documentation and Testing
  - **Docstrings**: Every function must include a docstring describing its purpose and parameters.
  - **Logging**: Implement logging throughout your application to track operations, especially API interactions.
  - **Unit Tests**: Write comprehensive unit tests for all components.
  - **Smoke Test**: Conduct a *basic* smoke test to ensure the app launches and can perform essential functions without breaking.
  - **Docker**: Containerize your application using Docker. Include a `Dockerfile`.
  - **Environment Variables**: Use a `.env` file to manage secrets and other environment-specific settings. *Do not expose any secrets on github!*

Non-goals

- Login status
    - The "login" route does not need to change the actual status of the user or model. All it has to do is verify the password.
- Security
    - You do not need to secure your application in any way beyond safely managing API keys and other secrets.

Examples project ideas on ed discussion