Stack Overflow Clone CLI

General Overview and User Guide

The stack overflow clone application is a command-line interface program. Using this application, users can authenticate themselves or register for an account. They can post questions, search for posts, post answers to questions, and vote on posts they favour. This application nicely models the usefulness of creating a centralized database of posts that users can find useful and tailored to their professions and interests. We modelled The stack overflow clone application after the implementation of the widely popular stack overflow website. All the features discussed, accompanied by a solid graphical user interface, will create a robust platform for sharing knowledge.

Here are a set of instructions to get you started with running and using the application.

1. To run the application, you must install the package "PyInquirer," "python3.x," and "pip3". Since the application is tested on the CS undergraduate lab machines, python3 and pip3 are already installed. To install "PyInquirer" you can execute the following command:
    - "pip3 install PyInquirer"
2. To start the application, you need to run the command: "python3 __init__.py" within the application's root directory; this will initiate the application.
3. Once the app has started, go ahead and select "Create Account" if this the first time you are running the application.
4. Follow the prompts and enter the required fields.
5. Once done, the application brings you to the main menu page. Here you can select either "Post a question" or "Search for posts."
6. Select "Post a question" and follow the prompts. Once done, the application brings you back to the main menu page once again.
7. To log out of the application, you may select "Log Out."
8. Select "Search for posts" and follow the prompt and subsequent menu options.
9. The interactions with menu options that will follow will be identical to the ones described above. Make sure to follow the prompts and select your desired menu option.

Software Design

To handle our database, we have incorporated sqlite3, which is a powerful DBMS. Our sqlite3 database interacts with a CLI application that is powered by python. To build our command-line interface functionality, we have researched and incorporated a package called "PyInquirer." (For more details regarding the package consider visiting: PyInquirer)

Our design pattern follows the fundamentals of the Model, View, Controller design pattern.
The various scripts responsible for either models, views or controllers have been grouped accordingly. We designed the following directory architecture to demonstrate the layout of this design pattern in our project.

Models/

authModel.py --> Handles queries and communication between app and DBMS in the Auth menu.
MainModel.py --> Handles queries and communication between app and DBMS in the Main menu.
model.py --> Connects the application to DBMS; this serves as the base class for the other models.
postsModel.py --> Handles queries and communication between app and DBMS in the post menu that comes after the main menu options.

Views/

authView.py --> Handles the command line interface of the authorization menu.
MainView.py --> Handles the command line interface of the main menu.
view.py --> This provides global styling values for the command-line interface.
postsView.py --> Handles the command line interface of the post menu.

Controllers/

authController.py --> This is an event handler for the authorization menu command-line interface.
MainController.py --> This is an event handler for the main menu command-line interface.
postsController.py --> This is an event handler for the posts menu command-line interface.

The entry point to the application is the file: "__init__.py".

To aid with our design and implementation, we created a UML state diagram to abstract the interaction between the application, DBMS and user.
To view the diagram click on the following link: **UML State Diagram**

Testing Strategy

We have incorporated various methods of testing to ensure the application performs as expected. Here are all the testing methods we used.

1. **Coverage Testing:** We designed test cases to ensure that we executed every line of code within the program. Sample test cases may include:
    - Select "Create account" --> Follow prompts --> Select "Exit."
    - Select "Sign In" --> Follow prompts --> Select "Post a question" --> Select "Exit."
2. **UI Testing:** Interacting with the UI to ensure prompts are working as expected. This testing was conducted concurrently with the coverage testing as they shared the same test cases.
3. **Regression Testing:** Rerunning tests to ensure the program performs as expected after integrations and changes.

These test cases were designed to:

A) **Validate** to ensure the program runs correctly and meets the needs of users
B) **Verify** the program to ensure it complies with the requirements of the application

Work Breakdown Strategy

To manage collaboration between team members, we decided to make use of two widely used tools in software development:

- Git for version control
- Github.com to collaborate and push our commits to a remote repository

Upon starting a task, each developer is responsible to branch out from our main or release branch and create a new branch to commit to. Once commits are made to a branch and the branch is ready to be integrated into our system, the developer will push the branch to our remote repository on Github. The developer will proceed to make a pull request. The pull request is then accepted upon review. For more details feel free to check out our public project repository: GitHub Repository

Here's a break-down of the division of tasks among team members.

- Alireza Azimi (Sazimi):
    - Created main menu MVC foundation codes (~ 1.5 hrs)
    - Implemented "Post a question" functionality (~ 1 hr)
    - Implemented "Edit Post" functionality (~ 1.5 hrs)
    - Implemented error handling for queries (~ 1 hr)
    - Created and completed design document (~ 2 hrs)
    - Performed UI, regression and coverage testing for implanted features (~ 1.5 hrs)

- Brock Chelle (Bchelle):
    - Created UML State Diagram (~ 1 hr)
    - Implemented Auth Controller, Auth Model, Auth View (~3 hrs)
    - Implemented branch to vote on a post (~ 1hr)
    - Implemented branch to mark an answer as accepted (~ 1 hr)
    - Implemented branch to answer a question (~ 1hr)
    - Implemented logic to decide which actions a user can take based on post type, and user privilege (~ 1hr)
- Archit Siby (Siby):
    - Implemented the Search Branches (~4 hrs)
    - Implemented Formatting Results and Further Post Action Views (~ 1 hr)
    - Implemented the Privileged Users Branches (~ 2 hr)
    - Implemented Error-checking (~ 1 hr)
    - Implemented Checks for Injection Attacks(~ 0.5 hr)
    - Performed UI, regression and coverage testing for implanted features(~ 1 hr)

Known Issues

- Issue with command line user interface:
    - ❏ Link to issue in PyInquirer Repo
    - ❏ Due to this known issue a user cannot use a mouse pointer to interact with the command-line interface.
    - ❏ We are handling this issue so that it doesn't crash the program