**Stack Overflow Clone**

**General Overview and User Guide**

The stack overflow clone application is 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 favor.

This application nicely models the usefulness of creating a centralised database of posts that users can find useful and tailored to their professions and interests. The stack overflow clone application is modelled after the implementation of the widely popular stack overflow website. All the features discussed accompanied with 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.

To run the application, you must install the package “pyInquirer”, “python3.x” and “pip3”.

Installation details are provided in the README.

To start the application, you need to run the command: “python3 \_\_init\_\_.py” within the root directory of the application. This will initiate the application.

Once the app has started go ahead and select “Create Account” if this the first time you are running the application.

Follow the prompts and enter the required fields.

Once done you will be brought to a main menu page. Here you can select to either “Post a question” or “Search for posts”.

Select “Post a question” and follow the prompts. Once done you will be brought back to the main menu page once again.

To logout of the application you may select “Log Out”.

Select “Search for posts” and follow the prompt and subsequent menu options.

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 the functionality of our command line interface we have researched and incorporated a package called “pyInquirer”. (For more details regarding the package consider visiting: <https://pypi.org/project/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. The following directory architecture is designed to demonstrate the layout of this design pattern in our project.

**Models/**

**authModel.py** 🡪handles queries and communication between app and DBMS in Auth menu

**MainModel.py** 🡪 handles queries and communication between app and DBMS in Main menu

**model.py** 🡪 connects application to DBMS

**postsModel.py** 🡪 handles queries and communication between app and DBMS in post menu that comes after 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 🡪** 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 in order to abstract away the interaction between the application, DBMS and user.

Diagram, schematic

Description automatically generated

**Fig.1:** UML state diagram of the application

**Testing Strategy**

We have incorporated various methods of testing in order to ensure the application performs as expected. Some strategies that we have used can be broken down as follows:

1. **Coverage Testing:** We designed test cases that ensures every single line of code within the program has been executed. Sample test cases may include:

* Select "Create account” 🡪 Follow prompts 🡪 Select “Exit”
* Select “Sign In” 🡪 Follow prompts 🡪 Select “Post a question” 🡪 Select “Exit”

1. **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.
2. **Regression Testing:** Rerunning tests to ensure program performs as expected after integrations and changed

Using the mentioned strategies about 19 test cases were developed in total and were conducted in various stages of the development process.

These test cases were designed to:

1. **Validate** to ensure the program runs correctly and meets the needs of users
2. **Verify** the program to ensure it complies with the requirements

**Work break down strategy**

In order to manage collaboration between team members we decided to make use of two useful tools that are widely used today:

* 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 to 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 checkout our public project repository:<https://github.com/blchelle/stack-overflow-clone-cli>

Here’s a break-down of the division of tasks among team members. Each team member is identified through their CCID.

* 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)
* Created and completed design document (~ 2 hrs)
* Performed UI, regression and coverage testing for implanted features (~ 1.5 hrs)
* Brock Chelle (Bchelle):
* Archit Siby (Siby):