Reddit Clone CLI

General Overview and User Guide

The Reddit clone application is a command-line interface program. Using this application, users can log in using a user id or log in anonymously. They can post questions, search for questions, post answers to questions, vote on questions they favour, and list the answers to a question. This application nicely models the usefulness of creating a centralized database of posts that users can interact with anonymously and find helpful posts tailored to one's profession and interest. We modelled The Reddit clone application follows the implementation of the widely popular Reddit website. All the features discussed, accompanied by a solid graphical user interface, will create a robust platform for sharing ideas and information.

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," "pymongo," "python3.x," and "pip3". Since we tested the application on the CS undergraduate lab machines, python3, pip3, and pymongo are already installed. To install "PyInquirer," you can execute the following command: "pip3 install PyInquirer"

2. Prerequisite: Before interacting with the application, you may want to ensure you populate the database with documents and the collection of documents. You can populate the database by having the appropriate JSON files(e.g. Votes.json, Posts.json and Tags.json) in the application's Phase1/ directory. Once all JSON files are in the phase1 directory, you can populate the database by running: "python3 __init__.py" from within the phase1/ directory. It may take a few minutes to insert all the data.

3. Once the insertion is complete, you can start the program. Before starting the program, you need to ensure the MongoDB server is running and listening on a port. For example, if the MongoDB server is listening on port 27012, then you can run the program by:
    - Entering the phase2/ directory
    - Execute "python3 __init__.py 27012", you may replace the port number accordingly.

Software Design

For our database, we used MongoDB, an industry-standard NoSQL database system. This database interacts with a python CLI program. 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 (MVC) design pattern.

The various scripts responsible for either model, views or controllers are grouped accordingly. We designed the following directory architecture to demonstrate the layout of this design pattern in our project. Below, you'll find the project structure:

Phase1/

__init__.py → Script for handling insertion of data

Phase2/

__init__.py → Main entry point to the application

Models/

authModel.py --> Handles queries and communication between the app and MongoDB in the Auth menu.

MainModel.py --> Handles queries and communication between the app and MongoDB in the Main menu.

model.py --> Connects the application to MongoDB; this serves as the other models' base class.

postsModel.py --> Handles queries and communication between app and MongoDB 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.

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 "Login with user id" → Follow prompts → Select "Logout" → Select "Exit"
    - Select "Login with user id" → Follow prompts **-->** Select "Search for questions" → Follow prompts → Select a question → Select "back" → Select "Logout" → 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.

We designed the test cases 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 for branching out from our main or release branch and creating a new branch to complete their task. 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. If accepted, the reviewer merges the pull request.

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

- Alireza Azimi (sazimi):
  - Created post menu MVC foundation codes (~ .5 hrs)
  - Implemented search for question feature (1.5 hrs)
  - Implement select search result prompt (~ 1 hrs)
  - Format search result strings (~ 1hrs)
  - Created and completed design document (~ 1 hrs)
  - Performed UI, regression and coverage testing for implanted features (~ 1 hrs)

- Brock Chelle (bchelle):
  - Implemented Phase 1 of the project, including indexing (~3 hrs)
  - Implemented the Question-action: List Answers functionality (~2hrs)
  - Fixed several bugs during development (~1hr)

- Archit Siby (siby):
  - Starter Code for Phase 2, creating MVC design pattern. (~.5hrs)
  - Question and Answer insertion and following actions and checks (~2hrs)
  - Auth, Model and User Report (~1hr)
  - Main Menu controllers and Voting (~1hr)
  - Performed UI, regression and coverage testing for implanted features (~ 1 hrs)

- Team: Discussion and Fixing integration issues through joint coding sessions (~5hrs)