Mini Project 2

Group: Poulomi Ganguly (poulomi), Rahul Korde (rkorde)

General Overview and User Guide

This python project is a simple system that works in two phases. **Phase 1** allows a user to reset databases on a given MongoDB server running on a user-defined port and then populate it with the contents of 3 `.json` files:

1. `Posts.json`
2. `Votes.json`
3. `Tags.json`

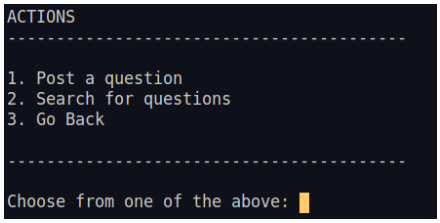
The entire process takes around a minute on given example data. Phase 1 can be run by the following command in the project directory after ensuring that all dependencies, including PyMongo is installed

```
python3 phase1.py <port no.>
```

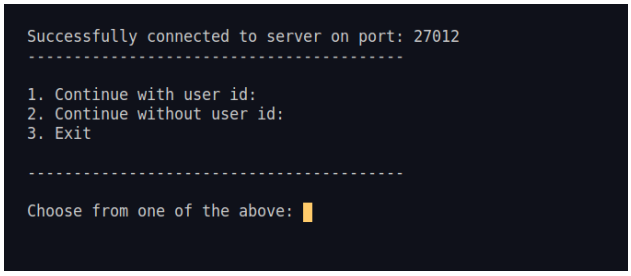
Phase 2 allows a user to access the database and operate on it. A user can choose to continue anonymously or with a numerical User ID for posting and searching questions. Phase 2 can be run with the following command in the project directory using the port number mentioned in the previous command:

```
python3 Phase2/main.py <port no.>
```

This should generate the following menu:

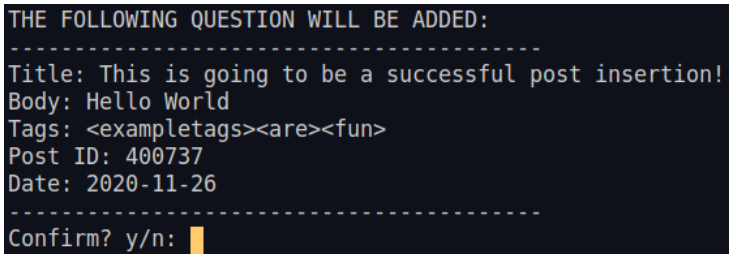


Choosing to continue with a user id (option 1) will generate a user report consisting of stats associated with the user. If the given user ID does not exist, a report will not be generated, however the user will be entered into the database upon having some post/vote activity. Alternatively, choosing to continue anonymously will lead the user straight to the next menu to choose between posting and searching questions. It will look like the following picture:



User can choose option 1 to post a question where they will be prompted to enter a non-empty title and body. User can also enter tags separated by spaces if they wish. This will be followed by a confirmation prompt accompanied with a draft version of the post and one can choose to not post the question by entering `n` or

confirm the insertion by entering y. A successful insertion will be displayed to indicate the same and the user will be brought back to the Actions menu.



User can also choose option 2 to search for posts by keywords. All posts matching any one or more of the keywords specified with spaces in between will be shown in no particular order. The user can then select any post by id (This post does not necessarily need to be shown in results) and then do the following:

1. Question actions: Invoked if the selected post is a question. All question fields will be displayed.

    * Answer a question: Write in a body text and upon entering, an answer will be linked to the selected question.

    * List all answers: This option will list all the answers linked to the selected question post. If an accepted answer exists, it will be displayed first with a star.

    * Vote Question: This option will ask for confirmation whether the user wishes to vote the selected question. This will increase the score of the question! **Note**: A user can only vote a post once.

2. Answer actions

    * Vote Answer: This option will ask for confirmation whether the user wishes to vote the selected answer. This will increase the score of the question! **Note**: A user can only vote a post once.

3. Back: Takes the user back to main menu.

Detailed Software Design

Phase 1:

1. **Connecting to server as client:** The port number is extracted from the command line and an attempt to connect to the server of the given port is made using `MongoClient`. If this step is unsuccessful the program throws an error.

2. **Dropping existing databases and loading JSON data:** If the server is already populated with a database that needs to be populated, it is dropped. Then, the json data is loaded by iterating over each of the json files and using the `<collection>.insert_many() command`. If it is a single record then `<collection>.insert_one()` is used for optimizing. Next, text indexing is added in to the searchable fields in `posts` collection, i.e: Body, Title and Tags. This allows for case-insensitive searches.

Phase 2:

1. **Posting questions and auto-generating ID's:** User given question fields are filled into a dictionary which is then inserted into the database using `.insert_one()` in *`postQuestion()`*. Tags are checked for whether they exist in the database, if not then a new one is created using `.insert_one()` and alternatively if they exist then the counter for it is incremented using the `.update()` method in *`updateTags()`*. Ids for both tags and question posts are auto generated in *`generateTagId()`* and *`generatePostId()`* by checking for the maximum id that exists in tags and posts collections respectively and then incrementing it by one.

2. **Searching:** Performed in *`search.py`* using text index search where the string of keywords separated by spaces is passed in as argument [Reference: https://docs.mongodb.com/manual/text-search/#text-operator]. Iterates over the cursor object that contains all matching posts and filters by PostTypeId to only show questions and their relevant fields.

3. **Answering:** Similar to posting questions, only the body field is taken from the user and the relevant details are entered into a dictionary which is then passed through an `.insert_one()` command in *`postAnswer()`*. ID generation is also similar to how question and tag ids were generated. Answer count is updated using `.update()`.

4. **Listing Answers:** All posts with ParentId matching the ID of selected question are fetched and then filtered by PostTypeId. If an accepted answer exists, it is printed first and its id is saved. The cursor object is then iterated over by filtering out the accepted answer's id to display all relevant fields while delimiting body field by 80 characters using array slicing.

5. **Voting question/answer**: Takes in given post as argument and confirms whether user wants to vote it. If confirmed, adds an entry to votes collection as well as updates the score of the selected post. If a user is not anonymous, it also checks for documents in `votes` collection where the UserId matches that of the user and the PostId matches that of the selected post. If a match is found, the vote does not go through. This all takes place in *`Vote.py`*

6. **Selection of post**: Always `.update()` the ViewCount of a post by one.

Testing Strategy

1. Return testing: Extensively used `try: except:` pairs to filter out any potential mishaps and used `while True():` loops for main menus in order to make sure that Back options return to their correct places instead of crashing the program

2. Posting Questions and answers: Debugged this by adding in multiple questions with varying lengths and types of content and ensured that it is never empty. Tag creation was checked in a similar manner by entering both existing and non-existing tags and checking with client whether they were created/counts were incremented. Answer insertion was crosschecked by checking whether answer count was updated in question post as well as whether the answer post existed in posts collection.

3. Searching: Tested by

* Making posts with exact same content except case-matched

* Making posts that had common terms

* Making sure duplicates weren't returned

* Searching for terms that matches no posts

    * Searching for terms that existed in multiple and singular fields in the same post

4. Listing posts/answers: Crosschecked using find command in mongo client terminal

5. Votes: Tested this extensively by adding multiple votes from anonymous users and checking if non-anon. Users could vote on both question and answer posts twice. Vote count updation was checked by listing the post.

Group Work Breakdown

Due to extenuating circumstances, all the work in this project was done by Poulomi Ganguly. Project was also maintained on a GitHub repo for regular updates and backup in case necessary.

`https://github.com/poulomi-g/MiniProject2`

Currently private, please ask for access.