Gist of Tidder

Tidder is a text-based, non-online Q&A program, similar to a forum. By first creating an account, users will then be able to:

- Search through existing posts
- Ask questions
- Answer questions
- Vote on a question or answer posts

Through the use of simple commands in the console.
Users can also choose to log out from the “navigation” screen. More details are available below.

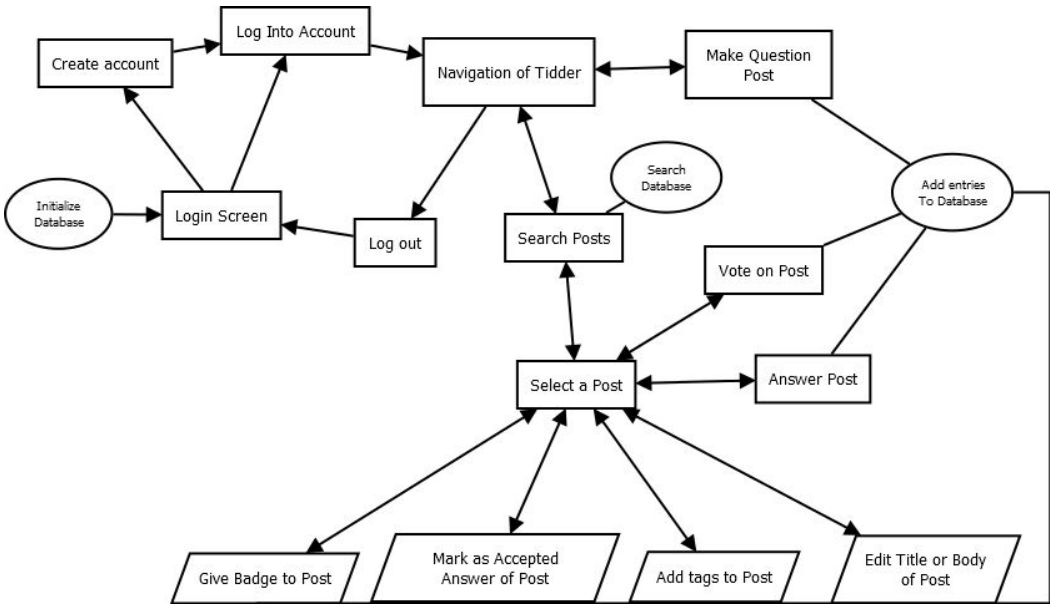
Simple Navigation

Tidder uses a CLI (command line interface) to interact with the user. First, to use Tidder, users must either create an account or, if an account exists, must log in. Once logged in, the user will be taken to a “navigation” page, allowing users to search through existing posts or create their own posts. To take “actions” on a post, a user must first search through posts and select a post. Once selected, a regular user will be allowed to either vote on the post or answer the post.

Once a post is selected, privileged users will also be allowed to edit the post, give tags to the post, mark the post (if it is an answer) as the accepted answer, or give a badge to a post.

More In-Depth Look at Tidder

The following is a flowchart describing how a user can interact with the program:



Boxes represent the interface a user will see when on that screen.
Arrows represent where a user can go and is allowed to go.
Circles represent background processes, mostly when data entered into the database.
Slanted boxes (parallelograms) represent functions only privileged users can access.

The functionality of Tidder can be broken down into four major components:

- The database which stores the information
- User login, which deals with creating an account and logging into Tidder
- The main function, which deals with “Navigation of Tidder.”
- Post operations, which encompass all actions a user can make on a post (including searching posts)

We will take a look at each of these components in detail.

Most In-Depth Look at Tidder

Tidder uses Python, with the implementation of the sqlite3 module. The use of SQLite is to create, store, and modify the database, which holds all the information in Tidder. The program uses four files to function, each of which is described by the functionality above. We will go through the program’s interface and describe what the program in the background is doing.

There are four files the program uses to run: main.py, user_login.py, post_operations.py, and Database.py.

We start from the terminal where the user has two options

1. Running only the database as it is:
   python3 main.py database.db
2. Run with sql data:
3. python3 main.py database.db data.sql

The first program to run will be main.py. Main.py is the interface of the program, where most “major” decisions take place. When this runs, it will immediately reference the other three files, the first of which is the database. The database.py file contains a class named Database. This class, along with its methods, is the primary interface between Python and SQLite. It is initialized with the Tidder database, including the tables users, privileged, badges, ubadges, posts, tags, votes, questions, and answers. The next file referenced will be user_login.py, which is the function to create an account, if needed, or to log in if an account already exists simply. Once logged in, next is where most of Tidder’s functionality lies: In the navigation screen. Users can either make a post or search posts - both functions being methods of a class named Post_user, a class created in a file named post_operations.py.

The class Post_user is where most of the methods involving interacting with posts are taking place. This class is the bulk of the programming involved in Tidder. The main() function, while navigating, will call this class for all of the “navigating.” Two of the largest methods in this class are makepost() and searchpost(). makepost() is a smaller method call, not needing any other class methods to run. searchpost(), however, contains much of the user’s capabilities, such as voting and answering. This describes the major functionality of Tidder.

Testing Strategies

Many bugs originally arose in the first iteration of the program. To make the program fool-proof, testing was done by intentionally trying to break the program, using the knowledge of how it was created. With each new change, added method or function, or any change to the user interface, the program was tested by only testing the feature that was currently being worked on.

The testing was strictly restricted to ensuring all the functionality that was described in the specification was met.

As an example, there is a method inside of Post_user named vote(). It is the method called when the user wants to vote on a post. The specification document tells us that a user can vote only once. When we created the voting method, users were allowed to vote infinitely, which was discovered by intentionally voting twice or multiple times.

We tested the new methods as we progressed through the program. We tried out different branches of possible results to make sure there are no gaps. With each little change or addition to the program, that addition or change was immediately tested, along with testing all the previous functions and functionality, to ensure none of the new changes broke any of the old code.

Group Work Strategies

Maksimus and Ferdous worked together to create Tidder. A general breakdown of the work each partner contributed is found below:

Maksimus:

- User login
- Main Function
- Database class
- Post_user class:
    - Searching posts
    - Selecting posts
    - Voting on posts
    - Answering posts
- Bug fixing

Ferdous:

- Bug fixing

- Post_user class:
- Database class
    - Editing posts
    - Adding tags to posts
    - Marking posts as accepted answer
    - Give badges to post

Email was the main method of coordination between us. We first decided who would work on what part of the project, attempting to divide the workload evenly. Next, over the course of two weeks, we emailed back and forth, exchanging code, detailing changes we have made and asking each other to test the program back and forth. Thus, how the creation of Tidder came to be.
Maksimus estimates that he spent around ~20-25 hours working on the project.
Ferdous estimates he spent around ~18-23 working on the project.

Report requirements -

(a) a general overview of your system with a small user guide,
(b) a detailed design of your software with a focus on the components required to deliver the major functions of your application,
(c) your testing strategy, and
(d) your group work break-down strategy

The general overview of the system gives a high-level introduction and may include a diagram showing the flow of data between different components; this can be useful for both users and developers of your application.

The detailed design of your software should describe the responsibility and interface of each primary function or class (not secondary utility functions/classes) and the structure and relationships among them. Depending on the programming language being used, you may have methods, functions or classes.

The testing strategy discusses your general strategy for testing, with the scenarios being tested, the coverage of your test cases and (if applicable) some statistics on the number of bugs found and the nature of those bugs.

The group work strategy must list the break-down of the work items among partners, both the time spent (an estimate) and the progress made by each partner, and your method of coordination to keep the project on track.

The design document should also include documentation of any decision you have made, which is not in the project specification or any coding you have done beyond or different from what is required.