Author

Name : - Yazad Mehernosh Pardiwala
Roll Number :- 21F3001724
Student email id :- 21f3001724@student.onlinedegree.iitm.ac.in

I am pursuing this course alongside a full time BMS degree from Mithibai college Mumbai, where I am in my 2nd year. I fully intend to complete till the BSc level & will almost certainly pursue the full BS degree as well.

Description

As I've understood it, this project requires me to create a application which allows users to create (as well as edit, delete & view) blogs, follow users & view their blogs as well, & like/dislike/comment on these blogs.

Technologies Used

- Flask – To create the app, run the app, & route functions to the various urls.
- Flask_sqlalchemy – To define the models for my database, connect to the database, & query it.
- Sqlite3 – For the actual database itself.
- Pillow – To handle the images recvd via the html forms for creating/editing a post.
- Os – To specify the directory from which to access/write to files.

DB Schema Design

The database consists of 7 tables in total – COMMENTS, FOLLOWER_FOLLOWING, LIKE_DISLIKE, POST, POST_USER, USER, & the default 'sqlite_sequence' table (which my app does query).

The USER table has an autoincrementing integer USER_ID as its primary key, a unique & mandatory string USER_NAME (the unique constraint seemed logical & precedented–Gmail & reddit are examples + it was useful for signing up users). It also has a text password which is not nullable, & two integer columns FOLLOWER_COUNT & POST_COUNT which default to 0 (I included these to avoid having to query other databases for just the number of posts / followers; in hindsight they aren't necessary & increase redundancy).

The POST table has an autoincrementing Integer primary key called POST_ID, a mandatory string POST_TITLE, POST_CONTENT which is not mandatory, & POST_LIKES & POST_DISLIKES columns which are not nullable & default to 0. The image for each post is handled by saving the images w/ a file name that uses the POST_ID.

The POST_USER table has POST_ID & USER_ID as foreign keys, & these act as the primary key for the table (in hindsight this was not needed since I intended for no more than one user to be able to write/access a post; using USER_ID a foreign key in the POST & defining a relationship that way would have been more efficient).

The COMMENTS table has an autoincrementing primary key COMMENT_ID, a TEXT field called COMMENT_CONTENT, POST_ID as an integer foreign key field & USER_NAME as a TEXT foreign key field. The POST_ID field is to mark which post the comment is for, & the use of USER_NAME instead of USER_ID (making USER_NAME unique in the USER table proved useful here) meant I did not need to define a relationship between COMMENTS & USER & query the same in order to find the author of a comment.

The FOLLOWER_FOLLOWING table has 2 mandatory integer fields – FOLLOWER_ID & FOLLOWED_ID, & both are foreign keys referencing USER.USER_ID. They indicate that FOLLOWER_ID has followed FOLLOWED_ID.

The LIKE_DISLIKE table has POST_ID & USER_ID as foreign keys from their respective tables (these also jointly act as the primary key for the table), & one other mandatory integer field LIKE_DISLIKE. In hindsight this could have been a Boolean field – I simply used a 0 here to indicate a dislike & a 1 for a like.

The sqlite_sequence is a table already defined – the name field stores the names of all the other tables & the sequence field stores the highest value that has been acquired by the primary keys of said tables.

'API' Design

I decided to make my app first using the more typical URL function mapping that Flask supports by itself, & did not use Flask_Restful to implement an API. As such my app consists of 21 URLs, & of these 8 can handle both GET & POST requests, while the rest only needed to handle GET requests. Detailing the functions to which these URLs have been assigned would be too lengthy for this document.

Architecture & Features

The app's setup in terms of its file structure is fairly similar to that of the app designed in the 3rd screencast in week 5. *'main.py'* is the file responsible for creating the app, importing the controllers, & running the app. The file *'config.py'* specifies the directory the database resides in, as well as if the app should open with debugging enabled or disabled. The database.py has a variable which calls upon the declarative_base() function (which as I underst& it returns the metadata needed to establish that all objects of this class will be tables of a database) & another which becomes an object of the SQLAlchemy() class (which again is used to connect link the models defined in the *'models.py'* file with the tables of an actual database). The *'models.py'* file has all of the models for the tables in my database, & the *'controllers.py'* file naturally has all of the controllers the app relies on to operate.
The 'main.py' file resides in the root directory, while the remaining python files are all in the 'application' folder. The database is in a folder *'db_directory'* which is also in the application folder (in hindsight it would have been more systematic to place this in the *'static'* folder, but since there weren't any drawbacks to leaving it where it was I did not make this change). The *'static'* folder gets used by the app to store any images that are to be displayed alongside the posts, while the *'templates'* folder stores all of the html pages/jinja2 templates (a total of 21 of them, including 2 test pages I used for debugging).

In terms of the features I have implemented –

- User login/signup are both accomplished by an html form using the post method.
- A user's profile shows their total number of posts, number of followers, & total number of people they have followed, along with a list of the titles of all their posts showing how many likes & dislikes each has.
- The feed shows the titles, images, & authors of all people followed by the user in the following order – all posts from the most recently followed person in rev. chronological order, all posts from the 2nd most recently followed person in rev. chronological, & so on.
- Users can create a UTF-8 encoded post, which handles the safe html tags, letting them use things like italic, headers, etc (accomplished by marking the variable as follows '{{POST.POST_CONTENT|safe}}' in the jinja2 templates). They can edit all aspects of their post, & they can naturally delete their post as well (the delete link opens a confirmation form which uses the post method).
- Users can search others users to follow/unfollow/block them. The search page lists all users (excepting the logged in one) in alphabetical order), & html the form used to follow/unfollow/block them has some front end validation via if-blocks to determine which is applicable.
- No API has been implemented.
- Every URL verifies that the USER currently accessing the page is the one who originally logged in; the login form verifies that a user with that username exists & that their password is correct; informs the user if either is not met. The signup form informs the user if such a user already exists, or if the password & confirmation do not match. There are many other cases of validation throughout.
- Users have the ability to like/dislike posts (the Like option is simply a link which sends a get request to a URL that is never seen by the user – it adds the like to the database & redirects the user back to the post, likewise for dislike). The same user may not like the same post multiple times (receives a message if they try), but they may change a like to a dislike & vice versa.
- Users have the ability to comment on blog posts & to delete their own comments.
- Bootstrap has been used for styling each page individually; there is no external style sheet.
- There isn't a login system/framework, & there is no means of exporting posts engagement.

Video
https://drive.google.com/file/d/1qTdu-jhTIwKvne3x0LwP7a3CbaPwtB65/view?usp=share_link