DESIGN DOCUMENT
FRITTER

Abstract:
The general structure of the system is as follows. There are users who can sign into the system and there are posts that are made by various users. The users are stored in a collection called “users collection” and the posts in a “posts” collection.

Key players:

- The User:
    - The user has a default _id , a user_name and a password as well as a name field (not used for authentication). This is created during the sign-in process and enables logging in even when the app is restarted (persistent).
- A Post:
    - Only a user who is authenticated can make a post. A post has a default *_id, a post field as well as a user_*name, indicating which user made this post.

Navigating the app:

The app, upon starting renders the **log_in page** (welcomepage in views). This enables a user to log in or to be redirected to a sign up page.

The **sign-up page** collects new user data, and if a user successfully gives valid information (a username not in the database and not equal to “”), redirects the user to the sign in page.

If unsuccessful, the page redirects to a **failure page** that enables the user to go back to the sign-up page to retry.

If a login is successful, the user is sent to the **make_new_post page**. There the user can post a non-empty post to the posts collection. There the user can choose to post something or to be directed to the *allposts page*.

If posting is successful, the user is redirected to a **success page** and from there he can go to the *allposts* page. If unsuccessful (only in the case where an un-authenticated user tried to post), he will be redirected to the *sign-in page* (because it makes sense to prompt a user to sign in to make changes).

The **allposts page** allows users to see posts by all users of the app. Each post has a link associated with it that says “edit post”. If a user is the user who make the post, he is is redirected to the update_post page. Else, he is redirected to the allposts page. This functionality essentially tells the user he is not allowed to do this edit.

The **update_post page** allows the user to do two things: delete post or update post. If an unauthenticated user navigates to this page, he will not be able to make changes and will be sent to the log-in page if he tries.

The **logout page** is basically accessible by clicking one of the logout links on pages sich as: make_new_post, allposts and edit post. This was made so that a user has an option to leave if he does not want to do a particular action, such as post a post or modify it.

Design challenges and decisions:

- Preventing multiple users from having the same username
    - This part involved checking the database for the username before creating a new user document in it. Thus, no two users can sign in with the same login.

- Preventing users with fields equal to "" from being added to the database, ensuring empty posts cannot be made, and ensuring a post is deleted.
    - This was an interesting edge case because a user could the simply click on sign-in on the sign-in page and be redirected to the post_new_post page. This would cause posts from a user named "" to be made and was not a desired behavior because anybody could just sign in like that. Thus the textboxes requiring user input, such as username and password were made to not accept empty as a valid input. The same logic is applied to posting as well as editing a post, a user cannot simply delete the post content and update it, he will have to delete the post to get rid of it. To see this behavior, simply try submitting something with empty field.
    - There were essentially two options to solve this. Either integrate checks for each text box for the fields and somehow tell the user when some input is wrong OR to enforce that the user can only make the request with valid inputs. The second option was chosed and was implemented in the ejs files (HTML) by requiring those fields.

- Preventing users who have not been authenticated from modifying/ deleting posts (security)
    - An unauthenticated user can, given the url with the post ID, navigate to an edit_post page. To prevent that user from making changes to the database (posts) was key so as to enforce security. Thus, sessions.user_id was used to check if the original post user_name matched that of the session user.

- Other decisions:
    - One can go to a URL and view posts (allposts) or even view the posting page (for now), later I plan to allow outside users to make a post to fritter as anonymous users. Which is why I chose to leave in this feature.
    - The pages were kept simple and made intuitive by allowing the users some options via a logout link or a go back link or essentially a way to be redirected elsewhere (in as logical a place as I could think of)

Grading highlights/help:
I particularly like the way in which I enforced no posts or users could contain “” fields. It was simple thing to do but changed the general flow of navigating.
I particularly like the pipelining of the pages. I tried to prevent weird behavior (such as unauthenticated users editing posts) as much as possible. Thus the pages were arranged in a logical order so as to ease navigation.

Modularity was enforced by making the methods in routes very independent from one another. This each method, if need be would have a GET and a POST version, and if needed they would have a corresponding view. The textboxes for posts were formatted using the CSS file and not individually to enable changing the overall look very easily.

Help: I was wondering how to do the subscription process as I did not make each user view a separate “homepage” based on his ID. IS it reasonable to look through the posts and see which ones the user can see based on who he is subscribed to?