Recipe Book – A recipe sharing community

Running the Application

The application is hosted on Heroku. To access the application, users may visit the following URL:

http://rb-recipebook.herokuapp.com/

A video that shows how to use the app is below:

http://youtu.be/jklY77ogqws

Application Usage

Recipe Book is a web app that allows the users to browse and upload recipes. The user interface is designed for an intuitive usage of the app's functionalities.

For the user interface, Jade is the templating engine used combined with bootstrap. The app is hosted on Heroku with MongoDB for the database. Other third party libraries used are Monk as a middleware for MongDB, Firefox's client-session for secure user log in and Multer for uploading images.

Anyone may browse the recent recipes. Users may see recipes of specific categories, using a panel of categories on the left. In using the search field, users may find recipes of specific tags. Tags are sets of keywords that is a recipe attribute provided by uploaders to help identify their app. Only logged in users may upload a recipe, rate them and leave a comment on them but anyone may browse the recipes.

For signed up users, they can keep track of what recipes they have uploaded, the comments they left on any recipes and their ratings for recipes. Once they are logged in, the profile button allows them to access all these data. In addition, they may also delete recipes they had uploaded, redo their recipe ratings and delete their own comments. When uploading recipes, users provide tags which are key words that help identify their recipe when searched for. These functionalities are straightforward on the user interface. Users with account may become website admins as well if the webmaster updates their account information.

Admins are special users who may delete anyone's account, recipes, or comments. The interfaces that allow these functionalities are only visible through an admin account.

Using the app is as straightforward as looking at the interface!

The basic features and design decisions in their implementation

The general implementation of the app is as follows. There are database tables for recipes, users, recipe comments, recipe tags and recipe ratings. The app generally queries the database to render the data into the views. This is done by event driven functions that are all stored in index.js. No AJAX implementation was done.

When users log in, data such as their recipes, comments and ratings are fetched from the database once it presents to be a part of the interface that the user is currently viewing.

User authentication

When users register for an account, the app authenticates with the database if the username or email has not previously been used yet. If the authentication is successful, the server sends a cookie to the user's browser which establishes the user's login session. The session is deleted from the server if the user logs out or if it times out.

On adding a new user, "/newuser" has a form which takes the user's credentials to be stored in the database "usercollection". "/viewprofile" is rendered after a successful registration. During the next times they log in "/index" is re-rendered displaying an indication of successful log in, but an error message is displayed otherwise.

User profiling

When a user signs up, the app stores the recipes they upload, any comment they leave and their basic information. These are all accessible through "/viewprofile" when users are logged in.

Users may also change their password. This is done replacing the stored user password in the database and re-rendering "/settings" to show users that their password had been successfully changed.

In addition, users may also delete any of their uploaded recipes. This is done by simply removing that recipe's entries in "recipecollection", "tagcollection" and "commentcollection" database collections.

Finally, users may delete their own comments, and other's comments if they are an admin, by deleting that comment's entry in "commentcollection", and then re-rendering "/viewrecipe" for users to see that the recipe they are viewing no longer includes the deleted comment.

Note that rating users was not implemented in realizing that it is not as valued as rating the recipes themselves.

Recipe pages

When users upload a recipe, the user fills in a form rendered by "/recipeform", and the data is stored in the recipes and tags database. If a user uploads an image for the recipe, an API, Multer, is used to copy that image onto the server and labels it uniquely.

As recipes accumulate comments, these are stored in a separate database table, this is the same table where users' comments are fetched for them to be able to view all their own comments. Recipes are viewed rendering "/viewrecipe" by querying the database whenever a user clicks on a recipe link.

Logged in users may rate a recipe, 1-5 stars. This is quite averaged with other ratings made for that recipe. If the user chooses to rate the recipe again, which they are allowed to do, their old rating will be removed, and the overall rating will reflect only the new one.

Finally, when users add a comment for the recipe, "/viewrecipe" is re-rendered to show the user that their comment has been added, after the comment is stored in "commentcollection" in the database.

Social Networking

Social networking is mostly established in the app as users are allowed to rate and comment on each other's recipes. Comments are stored in database as "commentcollection" which is also queried when users view (their own and other's) profiles to see the comments made by a specific user. In viewing the profile of a searched user "/viewprofile" is rendered displaying the recipes uploaded and comments made by that user.

Searching

When users search with a key word on the search field, the app looks for this key word in the tag attribute of all the recipes. Therefore, only recipe tags are searchable. The search results are then rendered to "/searchresults".

Admin

Admins are signed up users stored in the users database specially categorized by the admin attribute. Admin functionalities on the app's interface are accessible if the user has an admin status and is logged in. Admin functionalities are hidden on the interface and are made only visible depending on a boolean variable set on index.js that indicates the admin status of the logged in user. Therefore, admin functionalities are implemented by hiding special functionalities to regular users and a special user status. This explains why accessing a '/admin' page does not fit in the app structure.

Reputation system

The reputation system only applies to signed up users as only they can rate and leave comments on the recipes. When they leave a rating of 1-5 stars, this number is quite averaged with other ratings made for that recipe. If the user chooses to rate the recipe again, which they are allowed to do, the newly averaged rating of the recipe replaces the old rating made by that user.

Note again that rating users was not implemented in realizing that it is not valued in a recipe sharing community. This also explains why user earning points for uploading a recipe is not implemented. Recipe sharing communities only value recipes and hence only rate recipes.

Recommendations

Recipe recommendations are not implemented due to time constraints. Recommendations is supposed to display random recipes to logged in users that they may be interested in. The set of recipes a user may be interested in will be based the recipes they had recently searched for. For instance, if a logged in user had recently searched for key words “sugar-free” and “gluten free”, recommendations would display to users recipes having this tag.

Notes on testing the app

No testing framework was used because each functionality was done through the routing. There were no javascript functions to test on. However, throughout the development of the app, every time a feature was updated, we would test as many possible inputs as we could. This made overall development easier because we wouldn’t have to deal with multiple layers of bugs.

Notes on logging session security

Initially, user log in was implemented as follows. When a user logged into the app, the server would output a variable to our html indicating that the user was logged in. When the server needed to check who was logged in, it would pull the variable embedded in the html. This caused a major security issue where one could impersonate another user simply by editing the form in their browser which contained that variable.

This issue was fixed with the help of Firefox’s client-sessions API. Now, when a user logs in, a session variable is created. This contains the name of the user, and their admin status. The session is pushed to the clients’ browsers so the browser can render the relevant information, however since the server has their session information saved, it does not need to pull any malicious data from the client.