Matthew Fong

Rich Media 2 Project 2 - Tweeter Documentation

**Site purpose:**

For this project, I made a Twitter clone app. It is a small social media app experience where users create an account and can ‘tweet’ or post messages that other users can see or reply to.

**Profit method:**

My profit method for my app is through the use of ads. I display the ads on the right-hand side of the app so that the main app content is still displayed in the center of the screen.

**Templating language:**

In terms of templating languages, I used handlebars and React. I am using handlebars as the very basic skeleton of the app pages (login, app, and notFound). For React, I am rendering React components throughout the app experience. On the login screen, when I user switches tab views, the page is actually rendering React components for both the login and signup window. On the app page, if the user switches to the password change tab, the page is also rendering a React component to render the Password Change form onto the page. Otherwise, most, if not all, the user interaction features on the main app page uses React to render different views onto the page. Examples include, making/tweeting a message, clicking on the image icon to insert an image link, viewing all existing tweets, clicking the dropdown icon on a tweet to edit/delete that tweet, actually editing a tweet, which will render a change tweet form, deleting a tweet, which renders two more buttons onto the page, and finally, replying to tweets.

**MVC:**

In terms of MVC, I used express, which maps specific requests from the client browser to different functions on the server. Each specific request that comes from the client browser is received on the server-side controllers (Account or Tweet). Then the controller taps into the models to access the model’s database. After modifying an entry in the database, the controller returns the proper data/view back to the client browser to render onto the screen.

**Mongo:**

I used Mongo to store accounts and tweets. Each account has a username, displayname, password, createdDate, and salt properties. On the other hand, each tweet consists of the tweet message, the owner (account), the owner’s displayname, an image data property, favorites count, createdDate, and array of comments (or replies) to that tweet. The TweetSchema has a findAll(), findById(), and findByIdForAll() functions. The findAll() function is used to retrieve all existing tweets in the mongo database for any logged in user to view. The findById() function is used specifically to find a tweet by id that belongs to said user. This is mostly used when the user wants to edit/delete one of their own tweets. The findByIdForAll() function is used mostly for replying or favoriting a tweet. Any user can do either on a tweet but the server still needs to know which tweet to look for in the database, thus, the tweet’s id is necessary.

**Above and beyond:**

For above and beyond features, I wanted to focus on implementing features that are realistic and exists in real social media platforms. Other than editing and deleting data stored in mongodb, I wanted users to be able to favorite/like a tweet and add replies to a tweet. For editing and deleting a tweet, I wanted to make the UI buttons and interface as similar to Twitter’s as possible. Furthermore, in addition to tweeting a message, users can include an image in their tweet too. I felt having just text on the screen felt a bit bland and boring, so I wanted to allow users to include images to add more colors and personality to the page. However, my initial approach to implementing image tweets was extremely troublesome and time consuming. As a result, (and I won’t go down memory lane again) I opted in just having users post links to the image instead. Overall, I wanted the look and feel and user experience to feel as realistic and as close to a real social media platform’s experience.

Unfortunately, I did not include the dark/night mode feature in this project because of several reasons. Besides time constraints, I did not include the dark/light mode because of the structuring of my app pages. I had originally created a checkbox input that would change the theme of the app pages on change. However, because the checkbox input was independent of the other page elements, I did not want to grab all the HTML element references through JavaScript just to change their colors/background-colors.

For project 3, I plan on tackling the image data implementation again to allow users to upload images from their local directories and implementing dark/night mode (or maybe themes/skins for the app pages).