==========
Twitter Clone : Fritter
==========

Grading Directions

My Fritter app can be viewed live at twitter-daniman@rhcloud.com

Highlights

In my file system for this project, the main folders of interest are models, views, and routes. The models folder contains users.js and tweets.js, which contain the mongoose schema for the User and Tweet MongoDB Collections.

I think that the highlights of my code are the use of the MVC model (represented by the models, routes, and views folders), and my use of partials in the implementation of the UI.

I used partials for phase 1 of the project as well, but found that the UI that I created for phase 2 was significantly more complicated, and as a result, I had to create a file structure within the partials folder to keep the code clean and organized. I have a lot of ejs partials, and I think that they're all necessary and were good choices because it makes the logic cleaner → I also think that I did a good job of minimizing the logic in my ejs files.

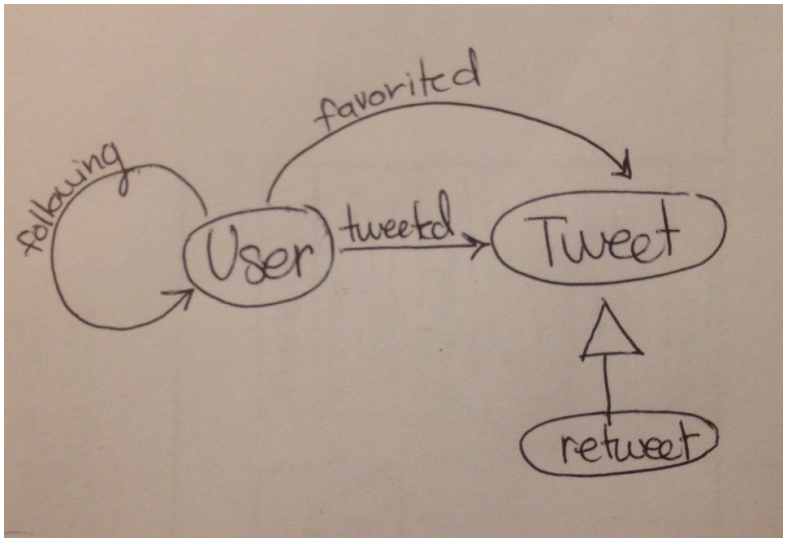
I kept my controller logic as minimal and clean as possible by creating my data schemas to be easily and quickly accessible, so I think that that is also a plus. I know that everybody probably used the Mongoose Populate method, but I was excited to discover that it existed and I think that my use of it in line 17 of routes/index.js was good – this method solved the main problem that I was having for phase 1 of the project, where I couldn't grab authoring data of the tweets to display in the tweet feed.

Design

Overview

This was an assigned project, not something that we got to choose, but I believe that the motivation for the instructors in assigning it was to give us an introduction to back-end web programming, as well as teach us the basics of data modeling by beginning with a simple app like Twitter.

The concept diagram for Fritter/Twitter is shown below.



Design Model

The concepts within Twitter are Users and Tweets. The concept of the Tweet is the most important, because most systems on the internet have Users, but a Tweet is unique in that it is a small message that's viewable to a lot of people all at once – tweets are especially great for news updates, etc.

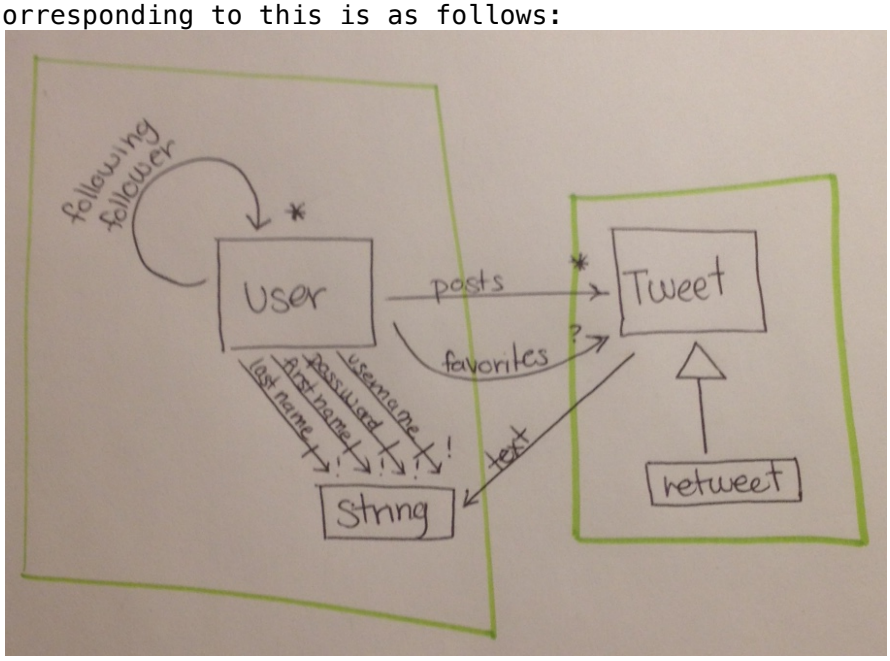
I designed Fritter to use a relational database with two collections: Users and Tweets. I then used Mongoose's populate method to draw connections between the two.

The schema for Tweet is fairly straightforward. Tweets have an *author*, which is just a string representing the UserID of the person who authored the tweet, and a *text* field representing the actual tweet message. Tweets also have two arrays, which support favoriting and retweeting. The favorites array holds the UserIDs of those people who have favorited the tweet, and likewise the retweets array hold the UserIDs of those people who have retweeted the tweet.

∗ A quick note about my implementation of Fritter is that retweeting is not actually supported in the UI of the app, though favoriting is. I wanted to be able to support retweeting and have it be a quick add to the project in the future, so I built the database to support it, however there is no actual button (or corresponding get/post requests) to retweet a tweet.

The schema for User is a little bit more complicated because a User has to keep track of a lot more things. The user must of course keep track of their name, username, and password; however, they must also keep track of their tweets (and favorited and retweeted tweets), and other users that either they are following, or are following them. Thus the User Schema has the subcategories of tweets, follows, and authentication.

The data model corresponding to this is as follows:



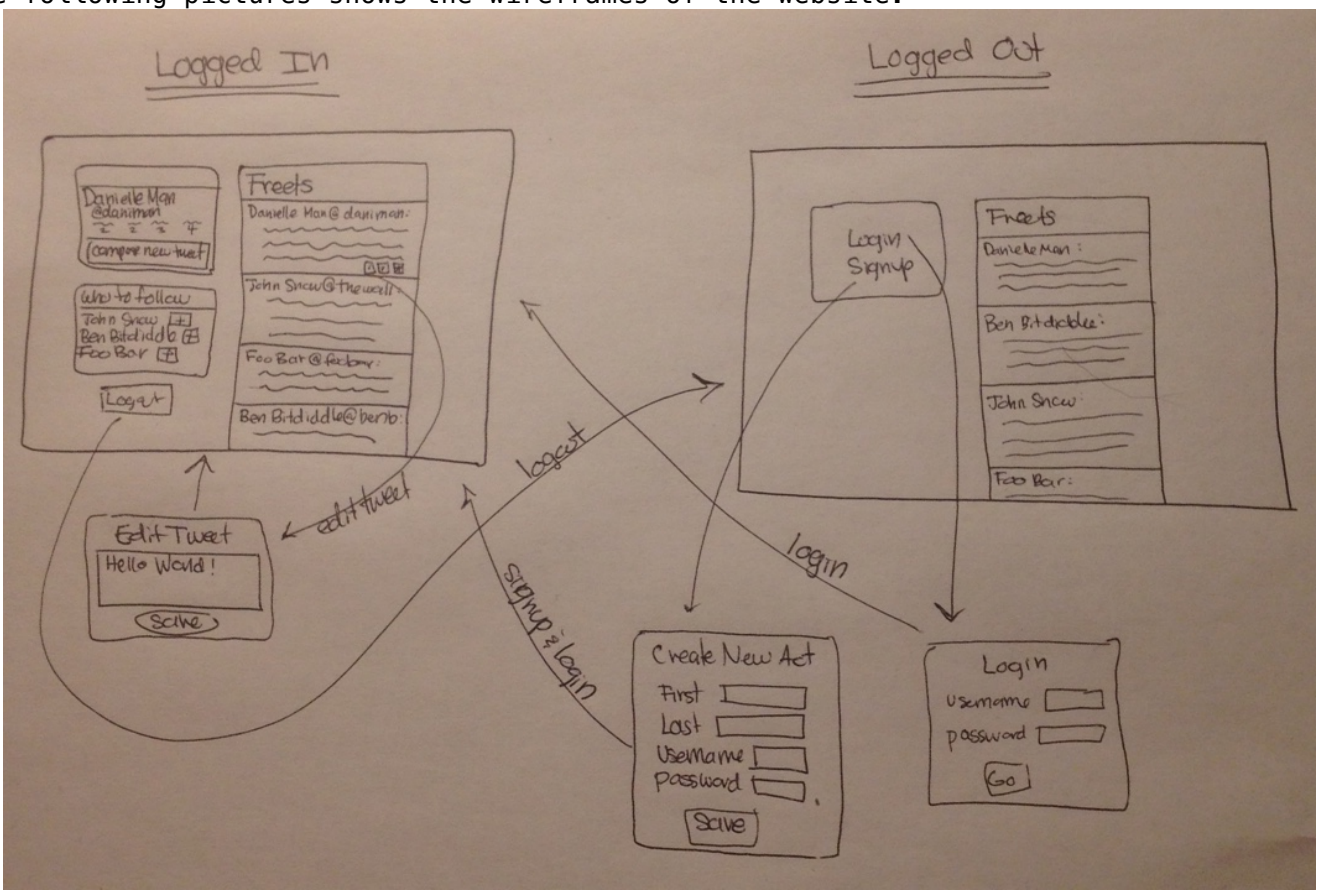
Behavior

We were not required to implement user authenticity safety, but I did put a lot of consideration into URL safety. I didn't want someone to be able to go to url/edit_tweet and be able to just edit any tweet, so I made sure that the pages I was rendering with my get/post calls were stayed true to the user. For the most part, my implementation is a single page app, with the exceptions being the login, signup, and tweet edit pages. Because of this, users are not able to edit each others accounts in any way.

When a user is logged in, the index page will give them a dashboard that shows their current status (number of followrers, etc). There is also another panel that pops up in

the bottom left corner, which gives the user a list of all the other users in the system. This allows the user to follow and unfollow other users at the click of a button. A user only sees the tweets of users that they are following in their tweet feed. A user who is not logged in will see the tweets of everyone in the world in one big feed.

The following pictures shows the wireframes of the website.



Challenges

I found the biggest challenges with the app to be the conceptualization and implementation of the data model.

In Phase 1, I had this problem where I could not populate my objects, but that was solved nicely with Mongoose's Populate method. With Phase 2, I ran into the problem of how to represent a subclass with Mongo and Mongoose, which don't strictly support subclasses. I believe that a retweet should be a subclass of a tweet, so though I didn't implement them in the UI, I wanted to support retweets now so that if I ever wanted to implement them in the future, I wouldn't have to do a complete data migration of my database. I considered a few different options for this. I could have had retweets be their own collection in Mongo, with a 'retweeter' and then a pointer to the original tweetID. This would have made displaying the tweets in the feed really complicated though, because not only would I have had to iterate over tweets, but I would have had to iterate over retweets and then combine the two into a list of posts that are sorted by time. In the end, I ended up adding an isRetweet field to my Tweet schema, that is either undefined if the tweet is original, or is a tweetID if the tweet is a retweet. If the tweet is a retweet, it will have the original tweetID so that we can pull how many times the original tweet has been favorited, etc., but it will be it's own new tweet in the Tweets collection, with the author being the user who retweeted it.

Another challenge that I had was that I wanted my database to be able to support a lot of the features that twitter does, even though I didn't have time to implement all of

them in the UI. These include modals popping up showing a list of the users that are following you, etc. when you click on certain parts of the page. As of right now, my UI shows you the number of tweets you've posted, how many tweets you've favorited, how many users you follow, and how many users are following you. I could have chosen for this to be a number in Mongo that is incremented up or down one by these actions, but I chose for it to be a list, so that in the future if I wanted to see who exactly was following who, that data would be available. As a result, I have a lot of arrays fields in both Users and Tweets. I think that this was important and necessary, considering future additions that I may want to add, etc. but merely incrementing a number would have have made my schema and a lot of my code much simpler and more elegant.