6.170 Project 2: Fritter 2.2

For the second project in 6.170, a Twitter clone, Fritter, was assigned. In this implementation, the website has backend persistence and frontend dynamics. The app is built with Node.js, Express and MongoDB. The implementation also uses EJS, jQuery, and Mongoose. The application is divided into separate sections of functionality. On Express initialization, a set of folders are created and these create an easy to follow division. From here, a */data* folder was added to hold the Mongoose schemas and models. The */routes* folder contains web routing divided by models they manipulated. Views were divided by main views in the */view* folder and an additional sub folder of partials to be used repeatedly.

The main features that were implemented in the first phase were the ability to create, edit and delete tweets. Tweets would then be associated with user instances that are created when a person sign up on Fritter. The Tweet Feed would then have all the tweets that were posted by all users. Whenever a user would log in, a session would be created and maintained in the MongoDB with the 'mongo-connect' module. This would ensure that as long as the user was signed in, his/her session would persist.

In the second phase, the abilities to "follow" and "unfollow" other users and to retweet other users' tweets were implemented. Here the user can now choose which tweets appear in the Tweet Feed rather than having all the tweets that exist. By following another user, his/her tweets would be included in the feed that is located on the left side. On the right hand side, the user is shown lists of people he could follow, who he's already following, and his current followers.

Instructions

The live Fritter site can be found at fritter-cmata.rhcloud.com.

The previous version of the site can be found at refritter-cmata.rhcloud.com.

From here you can, make a new profile by submitting a username, your name and a password and that is all you need to do to start tweeting on Fritter.

Grading Directions

Highlights

In the new addition to the code, I would like to point out the addition of a new schema and model. In the second phase, in order to simplify the query calls that were made to the MongoDB, I added an additional Mongoose model called Relations. Relations is a collection that kept track of the user and who was she was following and her followers. This allowed me to remove some of the additional complexities from updating fields in the model and from always having to access to the passwords in the User collection.

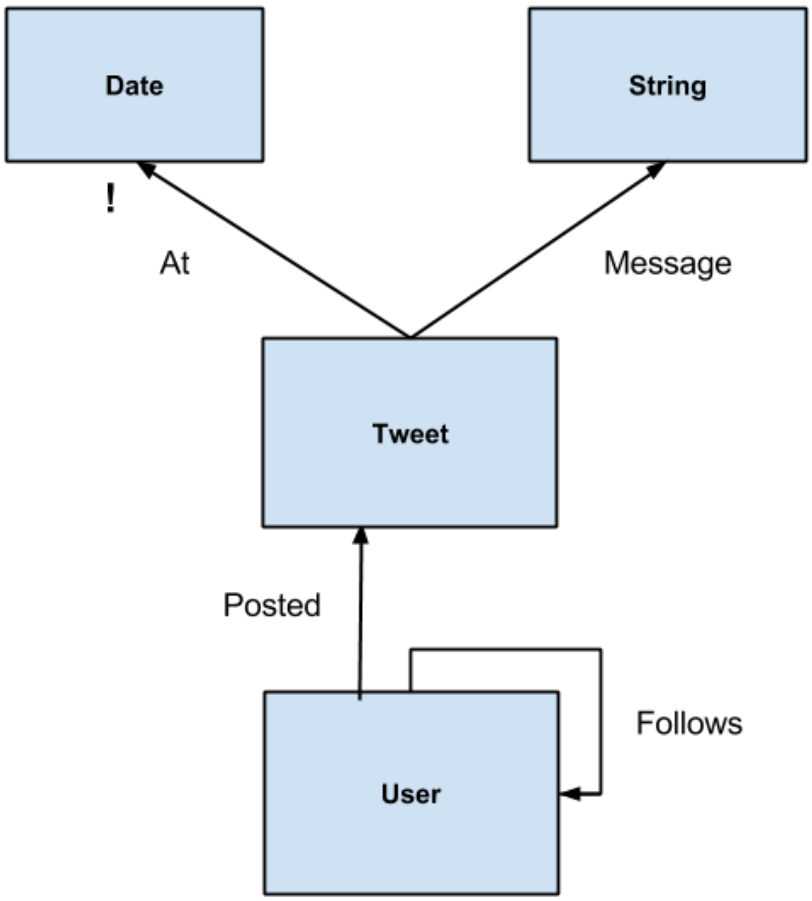
A highlight is the creation of retweets, which I defined to be the reuse of another person's tweet text with a small citation at the beginning. The logic is a reuse of the tweet, but would add a new tweet to the current user's tweets array. This a simple implementation that allows the user to able to retweet other user's tweets and the user will still be able to delete and edit the tweet as they'd like.

Another aspect that I wanted to highlight and also ask a question about was the creation of the home page. Here the homepage is the culmination of a series of queries. The first is to get the relations associated with this user, the second was to find all the tweets of interest (e.g. the user's and other users she follows) and then lastly, the people she could follow. This information is then passed to the view to be populated by the EJS logic. Here, I was most proud of figuring out the convoluted Mongoose API to get the correct queries created and returned like when finding the desired set of tweets. I also wanted to know if this is the best way to get all the queries. Is there a better way to get the results of many queries? Would it have been better to someone have each part of the view make its own calls to the database through AJAX or is nesting the only way? Furthermore, I would imagine there is a better way to do the queries so that the code doesn't have redundant lines of error handling.

Design

1. The first design challenge I had when implementing the second project was the decision of how to store the following and the followers. I had the option to make a new model, relations, or to choose to add the fields into the User model. I chose to make a new model as stated before because it simplified the query that I had to make to the Mongo database. Instead of adding an additional query nested within the already length chain, I reduced the query and simplified the logic necessary to get the followers and following arrays. Also, it reduced the amount of additional logic needed to find each of the arrays.

2. The second design challenge that I experienced was deciding on which additional feature to implement. I considered favoriting and retweeting. Here I thought both options would add a lot to the user experience of the website and would take a good amount of work to implement. I chose to implement the retweeting because it followed along the lines of a streamlined tweet feed that I had rather than having to complicate the home view with different tabs would have to get toggled if I did favoriting. I would've liked to implemented both if time would've allowed it.



(Previous Write Up)

Programming

Design

The main design parts follow directly from the models. The user model maintained the criteria for each user and was used in templating the views. The Tweet model maintained the structure of each tweet. From here, each of the routes were created to allow simplicity between displaying the models and editing the database. The three main views, home.ejs, index.ejs and login.ejs, all follow from displaying logged in user experience, the landing site and the login page.

The main challenges that I experienced were how to appropriately create a database model that would allow for reuse later. I had the option of using Monk or Mongoose to implement this. I had no experience with either and was not sure what the benefits or constraints would be for using either. In one of the lectures, Prof. Jackson spoke to the usefulness of Mongoose and I decided to heed his advice and use it. As it turns out, Mongoose lent itself well to this and allowing me flexibility into being able to quickly add new schemas and models to the app.

Another choice that I had to make was the decision of using Heroku or Openshift. The class materials said that Openshift would be supported but I had experience with deploying Rail applications to Heroku. I could learn another useful cloud deployment platform or use what I had already learned. I ultimately chose Openshift to gain more exposure, but I also got the added benefit of finding an online platform that I enjoyed using. I like the use of cartridges and the ability to add any as necessary.

Highlights

Using Mongoose, the programming required use of callbacks because of the asynchronous nature of the module. This allowed me to display use of callbacks throughout the routing logic that manipulated the database.

One of the biggest highlights was the use of Mongoose and using the lecture notes from “Using a Document Database” to maintain and simplify use with MongoDB. In the /data folder, the schemas and models for tweets and users are held in separate modular folders rather than one large schema and model file. This helped make it easy relating Mongo collections and will help in Project 2.2 to help and expand the functionality from here.

A required task was the creation of user accounts. Here the user information is stored in the database simply as strings (which isn’t secure, but wasn’t a requirement of this project). The User model requires a username, name and password. Each username is unique and cannot be reused once it has been taken in the database. This prevents others from posting tweets under someone’s username.

Another interesting functionality that was required and implemented was the use of sessions. Each user is created a session when they log into Fritter and it is maintained in MongoDB using the connect-mongo module. Once the session is created, it will persist in the server until the user chooses to log out. Once the user logs out, the session is destroyed in the database and the user is required to log back in to post, edit or delete tweets. A piece of middleware was added to check the authentication of the user. This middleware can be found in the /util folder. The middleware checks if the user that is logged in can access functionality or routes. If he/she is logged in properly, the site works well. If the user is not logged in, the user is constantly redirected to the root page to either log in or create an account.