Eric Gan
Project 2-2

Grading Directions:

My extra features for this project were adding retweets and creating a news feed for the user. There are also a few parts of my code that I would like to highlight.

1) #L24-51 in routes/newsfeed.js

   I thought one of the more difficult parts to code on my project was creating the news feed for my user. First I got the list of people the user was following, then for each of these users I got all their retweets and freets and stored them in an array. I prevented multiple of the same freets being appended for example if two users retweeted a freet then only one instance of it would show up on your feed. Also, I organized the freets in order that they were created; thus, if you edited your freet, the time it was last updated at would change, but it's position on the feed would not change.

2) #L67-91 in routes/users.js

   One of the extra features I implemented was allowing retweeting; however, this made deleting a freet more tricky. When a user deleted his/her freet, I made sure every instance of it was also deleted. Therefore, any place it got retweeted, it would also be removed from there. To do this I got the id of the freet to be able to find all the usernames in retweetsID, which is a list stored in every instance of a freet that carries the usernames of all the users who have retweeted this freet. Then for all these users I deleted this freet from their list of freets.

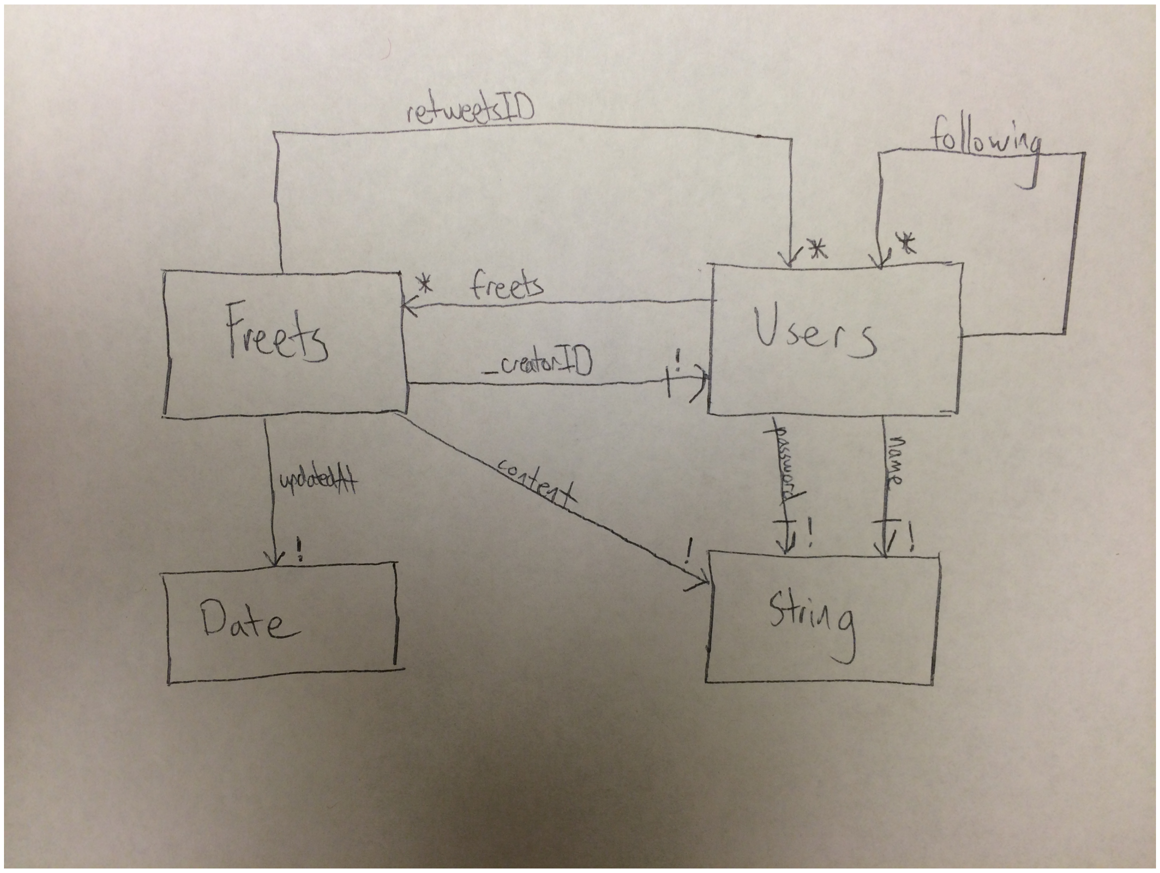
3) #L72-96 in routes/index.js

   I made usernames case insensitive to prevent the case where the users could have pretty much the same username, except that some letters were capitalized. Thus, whenever a user signed up, I made the username all lowercase and stored this in the database. Also, note that when a user logins I do the same thing of making his input all lowercase so I will be able to find it in the database. Finally, I disallowed duplicate usernames; thus, if the username already existed in the database I blocked the user from signing up with that username.

Design:

The first thing I had to decide on was how to use separation of concerns. For this I decided to use the Model View Controller. Mongoose made it easy to do this, as I could store the files where I described the schemas for my models in a models folder (I found this was a more definitive separation then when using Monk in the first part). Then my controllers were held in routes and my views were stored in the views folder. Also, I had to make design decisions on how to set up my databases. One thing I had to consider was how to model the relationship between freets and users. There were two ways I thought of doing this. The first was having the user have a freets field that held a list of freet ids (unique identifier) that I could use then to find the specific freets. I also thought about giving a freet a creatorID field that held the username of the user that wrote the freet. Note that username is a unique identifier as there can be no duplicates of usernames. The advantages of the former over the latter is that holding a list made in quicker to find all the freets of the user as we could just search through the users to find the user and then quickly return this field. However, if we didn't hold a list then we would have to iterate through all the freets in the collection and return any that had our username as the creatorID. On the other hand, the lists would require more storage space then when just using the creatorID. In the end, I decided on using both. I thought the extra storage was worth the faster run-time, especially when thinking for the future where that there would be many more freets in the freets collection. I also wanted the creatorID as it made it quick to associate a freet to its writer. If I didn't have this I would have to search through every users freets and see if the freet was in the list and even if I found it I couldn't be sure the user actually wrote it because retweets were also included in a users freets. Additionally, for modeling following, I realized we could have created a follow model that held two fields, the user being followed and the user being followed. This would make it convenient to find both relationships, who a user was following and also their followers. However, for this project we didn't really care about followers, just for who the user was following. Therefore, for each user I just held a field called following, which held usernames (unique identifiers) so that I could easily access all the users the user was following. This would be a lot quicker then having to iterate through all the records in a follow model, especially since we only really cared about the following aspect.

My drawing of my database is located on the next page.



Credit:

Used moment.js: http://momentjs.com/
Used livestamp.js: http://mattbradley.github.io/livestampjs/
Used Twitter Bootstrap: http://getbootstrap.com/2.3.2/
Used Font Awesome: http://fortawesome.github.io/Font-Awesome/
Used the TA's sample-express-app from GitHub: https://github.com/6170-fa14/sample-express-app/