Braeden Hall

101143403

COMP 2406 A

Stock Market Project

1. OpenStack Instructions
    1. Connect to my OpenStack instance, the IP Address is 134.117.132.74
    2. Enter the password: waterBOTTLE, exactly as written
    3. navigate to the ezstock directory
    4. ensure that the mongod service is running by typing: sudo systemctl start mongod
    5. run the command: node js/databaseInitializer.js, to get a fresh database *this must be done from the ezstock directory
    6. run the server with: node js/server.js *also must be done from the ezstock directory
    7. open a new terminal
    8. enter the folowing command: ssh -L 3000:localhost:3000 student@134.117.132.74, this will link port 3000 on your local machine to 3000 in the instance.
    9. navigate to localhost:3000 on your local machine
    10. test the system as you see fit, enjoy

- Accounts
    - There are 4 default accounts in the database from initialization, 1 is an admin account.
    - The admin password is pass1234
    - The remaining accounts are: BenjiG, MarketMat and TristanTrades
    - The passwords for these accounts are the same as the usernames
    - Obviously you are free to create as many accounts as you like. However, none of them will have admin access, so keep that password handy

2/3. Functionality and Extensions

General:

- sessions have a 3 hour hard expiration date after which you will be logged out, keep this in mind as you navigae the system
- I implemented a notification system for relevent events, to mark a notification as 'read' simply click on it in the dropdown list

- socket.io is used to update data on a page in real time as events happen in most if not all places it is required
    - the only area this not done is for notifications where a 5 minute interval is set
- the REST API is implemented as specified in the project requirements

Home Page:

- The home page provides the admin account the ability to advance the server to the next day
    - this updates each stock's historical data and looks to complete any orders remaining
- also allows the admin to bypass the default trading day system and allow trades to be made all day

User Accounts:

- new users have the ability to create an account by specifying a username and password
- existing users can login using their unique username and a password
- loggedin users can:
    - view and update their cash balance
    - view and edit thier stock portfolio
    - manage the orders they have currently in progress
    - view and manage their watchlists and subscriptions
    - I did not implement the transaction history section

Stocks:

- stocks can be filtered and searched through the given query parametres
- individual stocks can be selected to view more detailed information such as:
    - current price, bids, asks, shares traded etc.
    - historical data about the stock from past days, can be filtered to show a range of days
    - see the amount of the current stock they have in their portfolio
- on a specific stock's page users have the ability to:
    - place a buy or sell order for the stock, given that they have the capital required
    - add a stock to a watchlist
    - register a subscription to a stock

Watchlists:

- logged in users have the ability to add a stock a new or existing watchlist from the stock page
- users can view stocks in a specific watchlist from their profile page
    - stocks can be removed from a watchlist from this same page
- watchlists can only be created when adding a stock to a watchlist

    - I saw no reason to allow users to create an empty watchlist, though watchlist do persist after all stocks have been removed

Subscriptions:

- logged in users have the ability to register for event subscriptions to a stock from its main page
- users view, can remove, deactivate/reactivate or modify subscriptions from their profile page
- an active subscription will trigger a notification once per day provided the requirements are met
- the only subscription type currently implemented at the moment is a % change
    - negative percentage if the user wants to know if the stock drops, positive otherwise

Extensions:

- Database
    - I implemented a database because it adds a lot more scalability to the application. It provides the ability to store almost infinitely more data than if I were to store all my data in RAM. It is also a lot less finickey than trying to maintain updated files. Mongoose allows for much more ease of use for the application as well becuase the creating models to define datatypes prevents users from inserting bad data in to the application, this makes it more robust.
- GraphQL
    - I implemented a GraphQL route ("/graphql") on the server that allows you to make GraphQL queries to the server and recieve only the specified fields of data. I never use this in my application, but the original plan was to use GraphQL alongside React. I never got around to implementing react but since I had already built the GraphQL schema I figured it would be a waste not to include it. In theory GraphQL would have made queries faster because you are only requesting the data you need from the database (i.e. username, id, bids etc.) instead of recieving the entire document object. This means you would be transmitting less data across the internet and improve load times.
- socket.io
    - I use socket.io to update data on pages instead of intervals and timers in most places. I do this because it ensures that there are few to no useless queries to the server requesting data that has not been changed. This allows the server to spend more time handling more important queries, and improves scalability.

4. Design Decisions

I think I designed my system relatively well. I implemented a simple MVC architecture that effieciently completed what it need to do for the most part. The model is the data stored on the database and the view renders this data on the client side through the use of the template engine ejs. This makes the code very modular because the view pages don't care where the data comes from, whether it be a file database etc., as long as the objects have the correct properties the page will render correctly. This works in the opposite way as well, the data has no idea how it will be used, and it doesn't matter as it will not change how it is obtained.

I also did a good job using asynchronicity on the server wherever possible this allows the server to process multiple requests at the same time without stalling for one long action to complete. I used a database as welll insteasd of storing the data in RAM or in files as this allows for more scalability since there is space to store much more data. The use of mongoose also helps with making the system more robust through the design of models for data. When any data gets saved using the model.save() method the fields get validated, this means that it is difficult for a user to enter bad data that could cause the system to crash. I also devided my routes into small, reuseable, well named middleware functions so that it is easy to understand what each function does without necessarilly having to read the code.

As I discussed in the previous section, socket.io allows data to be transfered between server and client when events I triggered. I make use of this to update certain elements on the page when their values are changed in the database. This eliminates the need for repeated requests to the server for data that ,ay not have changed. Since there are less requests to the server it is able to execute the important requests at a faster rate, this makes load times on pages faster and improves the overall user experience. Where it was unreasonable to use socket.io to obtain data I made use of AJAX calls on the client side. This allowed me to request data in an asynchronus manner and update the page when it was ready, this caused pages to be more responsive with very little delay for the user.

5. Improvements

There are many improvements that I think could be made to the system that I think increase the general quality of my system. The first thing that I think could have been done differently is writing the client and server side code in TypeScript instead of Javascript. I think this would improve the understandablity of a lot of the code as well as increase the robustness of the system. TypeScript would require variables to be of a certain type when declared or passed to functions. This would decrease the chance of an error occuring because a string wasn't cast to an int correctly for example. It would also stop me from having to explicitly check types in certain scenarios, and as a programmer, less typing is always better.

The next big improvement that I think could have been made would be the use of a front-end framework such as React to implement the client side. This wouldn’t necessarily make the client side interface look better by itself, but it would provide the oppourtunity for more scalability and easier styling through the use of components. It would also make the connection between the client side Javascript and HTML easier to see and manipulate. I did experiment with the React framework Next JS, however I ran into trouble storing the session object on the Next JS server. I realize now that I should have probably tried using React with react-router instead so that I could render the app on the main server. In general, if I had used React I think I would have been more motivated to improve the UI of the system which would make for a better user experiennce, I could just never bring myself to spend time playing with regular CSS.

6. Modules

- graphql, express-graphql
  - these are all modules that deal with getting GraphQL to work with my server and allow queries to be made. As mentioned earlier, I don’t make use of this at any point in my cuurent implementation of the system. However, during the brief period while I was experimenting with Next JS this is how I made efficient queries. It allows you to only request certain fields of a document and combine different collections together into one object for easy access to any required data. These efficient queries improved scalablitity as the server spends less time handling large requests and sending large amounts of data to the user when they only need a few fields to render a page.
- bcrypt
  - I use the bcrypt module to hash user passwords. I felt that hashing passwords was important because if this system were to be deployed for real use storing plian-text passwords would be terrible practice. The bcypt module provides simple to use, effiecient but most importantly secure hashing functions that make it almost impossible to brute force passwords. It is much more efficient and secure than it would be had I tried to hash a password myself. Bcrypt gives users peace of mind knowing that their account data will not be hacked so they may trade happily.
- cors

    - Cors is another module that was included when I was playnig with Next JS. Cors allows connections to the server from different origins or urls/ports. It does this simply using one line of code instead of the many different lines you would have to write in order to include this functionality yourself (I know becuase I did it before I found out the module exists).
- socket.io
    - socket.io allows you to establish a connection between the client and the sever and send daat through that socket. I used it tom define events that trigger changes on client pages without the need to reload. With socket.io the data is sent/received as soon as the event occurs and only once, making sure that everything is always up to date. As discussed previously, I felt that socket.io was a better option than the alternative of using intervals because it elimanates the need to constantly make request to the server for data that may or may not have changed. Less requests to the server makes the more important operations run faster and decreases the chance of it crashing. This makes it more robust and keeps wait times down to make for a better user experience.

7. Overall Project

The thing I like most about my project is that I got to experiment more with mongoose and learn a lot of the cool things it can do for you. I really like being able to design models that the data must adhere to and even define methods in those models that can perform actions on that data. It reminds me a lot of object oriented programming in java and at the moment java is my favourite language. Overall this project taught me a lot about client server architecture and to design RESTful web applications. I think it provided a really good foundation for me to add upon. It has inspired me to want to learn new web technologies like React and TypeScript that I think would really take me to the next level. Even though this project is finished in terms of this course, I still plan to update and make it better as time goes on and I polish my skills. This project will probably end up being one of the key highlighted projects when I apply for co-ops next term.