# COMP 2406 Stock Website Project Report

## 1. OpenStack instructions

Instance information:

- Name: AmrNosir
- Public IP: 134.117.130.202
- Username: student
- Password: amr2406

To log in and start the server on a machine connected to the Carleton vpn:

1. ssh student@134.117.130.202
2. cd stocksWebsite
3. node server.js
4. In a separate terminal: ssh -L 9999:localhost:3000 student@134.117.130.202

The server can be reached using localhost:9999

Some sample data is automatically loaded using the files in /json. The example user is hardcoded as admin. The login details for example user are:

- Username: amrcash
- Password: abc123

## 2. Implementation Summary

I would like to point out that there is a significant portion of the features missing. My goal for this submission is to demonstrate that I do grasp the core concepts of the course and that I can implement them with good quality, instead of aiming for the highest bracket in the grading scheme. What has been implemented from the requirements successfully is:

- All the front-end functionality for how the user interacts with the website, and most of the related requests/responses
- Most user account features
- Account authentication through sessions
- Most stock page features
- Some watchlist features
- Some event subscription features
- A notification display portion
- Parts of the following public REST API: GET/stocks & GET/stocks/:symbol

What has not been implemented from the requirements:

- The admin features to simulate along
- Some of the back-end server logic that does:
    - Real time stock data updates
    - Order fulfillment algorithm

    - Notifications for trades and event subscriptions
    - Watchlist updates
- Buy/Sell orders with daily expiry
- Query parameters for the required REST API

## 3. Extensions

None that go beyond some design decisions.

## 4. Design Decisions

Each of the following paragraphs outlines one of my design decisions.

The server data is organized into multiple json files, in the future each file would be a table in a database. For convenient data retrieval within the current server design, all the files, except the one for users, are objects with each object's unique Id as a property. For the stocks file, the symbols are the unique Id. While orders, notifications, and subscriptions are user-specific, they have their own "tables" as they are often accessed for operations related to stocks or to other users. For example, when an order is placed by a user, it has to be registered to the relevant stock as well as the user's account. Without a separate orders "table" if part of the order is fulfilled or the user edits the order, the order data will have to be updated in two places. With a separate orders "table" however, updating the data there is sufficient. Given the previous reasoning notifications may not need a separate "table" depending on how real-time browser notifications are to be implemented, but that's pending review so it has its own file for now.

Asynchronous design was taken into account to improve user experience. All the tables use AJAX requests to retrieve their data. All post requests are also done through AJAX, and simply notify the user with the result.

All user-specific requests are handled using sessions that hold a userId assigned by the server. The goal was to maintain a secure framework that can be used as the website grows. The reason a unique userId is used in the sessions instead of the unique username was to create a backend that is isolated from username changes. This way, a future feature that allows users to change their usernames can be easily implemented without having to go through all their orders to update the username or having to refactor the database design. For added security, user objects cannot be sent to the client especially because there is no client-side feature that can benefit from it.

Two routers were created, one for the stocks and one for user related requests. The goal of this was to split the server structure into logical chunks that are easier to manage and separately develop which would increase its scalability. Since this project does not have a database implemented a database file is also used. These are the server/router responsibilities:

- **server.js**: Starting point, and is responsible for all authentication/authorization logic, as well as account creation. Also responsible for admin logic.
- **stocks-router.js**: Responsible for any url that starts with "/Stocks". It serves the stock related pages, handles stock related actions such as placing buy/sell shares orders and handles the JSON API

- **user-router.js**: Responsible for any url that starts with "/User". It serves the user account pages, handles user related actions such as updating balance, and serves the data for the different account features
- **database.js**: Initial server data is stored in json files. This file reads the files and exports the variables that contain the data. The separate routes can share changes by updating those variables

The pug template engine is being used to build all the pages. The goals were to increase code reuse using partials and to allow for the dynamic construction of stock-specific and user-related pages. These changes would improve the scalability of the system as now page rendering is independent from the size of the user/symbol database.

Care was taken to ensure the robustness of all implemented features:

- Type checks are performed in both the front-end JavaScript and in the server in case requests are not sent through the website.
- Users are redirected where logical. For example, when they try to go to the login/signup pages while they are already logged in.
- Simple error messages are also sent where appropriate to notify the user when something goes wrong. For example, if they try to withdraw more money than they have, or if they try to get the page of a symbol that doesn't exist.

## 5. Possible Improvements

Aside from order fulfilment, most of the unimplemented features would be implemented by copying and modifying already implemented features. So while those are the key improvements to get this project to actually fully work, they won't demonstrate any more understanding of the course concepts.

Otherwise, the most pressing limitation of the current design is the lack of data persistence if the server crashes or is restarted. This is due to the "database" running entirely in memory, which introduces a host of other issues, like risking running out of memory. All of these would be dealt with by using a proper database like MogoDb or Mongoose.

Security is another factor that can also be improved significantly. Currently, any password regardless of length is accepted, but that is not always a good idea. To protect user accounts from brute force attacks, and from the user's friends guessing an easy password, conditions like password length or special character requirements should be added to restrict accepted passwords. However, this will not be enough to secure the server side, which is arguably way more important. Currently, usernames and passwords are stored in plain text within the user object, so anybody with access to the database can easily compromise all user accounts. A potential way of dealing with this is to hash the passwords before storing them, that way login details are not human readable.

A feature that I sought to add since my first check in was a graph that displays a stock's price history. I found chart.js as a promising 3rd party way to implement it but time, among other things got in the way. Graphs would go a long way to improve the user experience when browsing stocks.

Presentation is important for driving traffic, so an html only website is unlikely to get any attention in 2020. Improving the visual presentation would go a long way in improving the user experience when browsing my website. Screen size adaptability and a prettier format would go a long way in that regard. These can be done through css and front-end frameworks. Other purely aesthetic features I can think of are:

- Loading animations for tables as the ajax requests for the rows execute
- Popup menus for simple actions like logging in, depositing/withdrawing cash, placing orders, etc.

While I have attempted to do so in some regards, there is definitely room for improvement in how the server sets status codes and how the client handles them.

Any of the extensions in the project document would also be great.

## 6. Modules Used

The following modules were used:

- express: To build the server
- express-session: To support user sessions
- pug: To dynamically render pages
- uuid: For unique Id generation

## 7. What I like about my project

I like that even though it's incomplete, it works decently well.