COMP 4513
Assignment #2: MERN *(version 2.1)*

Due Monday April 9, 2018 at 10am

Changes in yellow or green (March 9)

Overview

This group assignment provides an opportunity to display your front-end, back-end, and dev-ops capabilities. Groups can contain three or four people. Please don't ask to be a group of five, eight, eleven, or any other number larger than four. You can also work in a group of two or alone, but you will still have to do the same amount of work.

This assignment is an expansion of your first assignment.

Submitting

You will not be submitting your source code. Instead, I will mark your code from a GitHub repository. I will mark the functionality by visiting some type of live server (not cloud9). Thus, you will submit your assignment by emailing me a short message consisting of the group member names, the GitHub repository URL, and the web address where I will be able to find your working assignment.

Grading

The grade for this assignment will be broken down as follows:

Visual Design and Styling	15%
Programming Design	15%
Functionality (follows requirements)	70%

Data Files

I have provided you with several JSON data files that you will import into MongoDB. To make things easier, I have decided to provide just subsets of the data. The following files have been provided:

`users.json`	The users for the system, which includes id, first name, last name, email, salt, and password.
`companies.json`	Information about each company (only about 200 companies).
`prices.json`	Company stock data for each day in 2017.
`portfolio.json`	Portfolio (current stocks owned) for users.

I will eventually have all the necessary files available at:

`https://github.com/rconnolly/comp4513-w2018-assign2`

Back-End Requirements

1. To begin, you are going to want to import the JSON data into a MongoDB database.
2. You are going to need create Node/Express web services for retrieving data from MongoDB. You will need to create at least the following services:

    a. Given an email and password, return the id, first name, and last name if credential information is correct.

    In a real application, we wouldn't send a password as part of a GET request to a web service; here we are just to simplify this service.

    The actual password for each user is **funwebdev**, but has been salted and subjected to an MD5 hash function. That means to check for a correct login, you will have to perform two queries: the first to retrieve the **salt** field for the entered email (e.g., mdanelutv@wix.com) from the Users collection, and if that user exists, concatenate that salt value and the entered password, and run them through the MD5 function. If the result of that function matches the result in the Password field, then return the appropriate id and name information.

    I would recommend installing the **crypto-md5** package using npm. You can then perform the necessary hashing via: **md5(password + salt, 'hex')**

    b. Given a stock symbol, return the company information for it.

    c. Given a stock symbol and a month, return the price information for each day in the specified month. Some days (weekends and holidays) have no data, so this service will likely return some 15-20 price objects.

    d. Given a stock symbol, return the average close value for each month in the year.

    e. Given a stock symbol and a date, return the price information for that date.

    f. Given a stock symbol, return the latest price information. This will return the price information with the newest date (in our data that will be late December … you can hard-code that date for simplicity purposes).

    g. Given a user id, return all the portfolio information for that user. A given company can appear multiple times in a user's portfolio (perhaps representing separate individual purchases).

    h. Given a user id, return a percentage summary of the portfolio information for that user. That is, provide a list of each stock in that user's portfolio along with a percentage number that expresses the percentage of that stock in terms of the total number of stocks owned. For instance, user A, owns 200 of AMZN, 100 of GOOG, 300 of AAL, and 700 of ABC, then this service would return AMZN, .1538; GOOG, .0769; AAL, .2308; ABC, .5385 (though of course it would need to be formatted as JSON).

    i. Return list of all companies (just stock symbol and company name).

    Note: some of these are more complicated than others. Some will require using Mongo aggregate function [in mongoose, use aggregate() function instead of find()]. Because date data in the json file will be simply strings as far as Mongo and Mongoose are concerned, you will likely need to make use of regular expressions for your matching.

    Working with the Mongo aggregate function can be difficult. If you are struggling with getting your aggregate functions to work, you may decide to do the grouping and calculations **outside** of Mongo in your Node code. In such a case, I would recommend making use of the popular lodash package, which has many powerful array manipulation and search functions.

I would recommend implementing these services early. You can do so initially on Cloud9, but it probably makes sense to get these working on an external server early on. You may also want to create a test page with sample links for each of these services. Remember that REST service responses are viewable directly within the browser. You can also use the Chrome extension Postman.

3. You will need to create a chat server in Node that will use **socket.io** to push new chat messages to the user (if logged in). While you could combine the chat serving and the API serving in a single file, you may find it easier from a programming perspective to separate them.

Front-End Requirements

1. You must make use of the create-react-app starting files and structure. That is, you are creating a SPA on the front-end.
2. You must use the Bulma CSS Framework as the starting point for your design. However, you will need to use SASS to customize Bulma and come up with a different color scheme. You must also modify a few other, non-color variables. I will need to examine your SCSS file in which you have made your customizations. **Sass** command line tool is already installed in Cloud9. You can also install SASS using npm on your development machine.
3. Your site design should be optimized for mobile usage. That is, make sure everything looks good at a smaller browser size.
4. The first thing the user must experience is a log-in screen if the user is not already logged-in. The form must provide mechanism for entering email and password. It will display a Bulma notification if the credential information is incorrect. This notification must disappear once the user starts entering information into the login form. I would recommend adding in the log-in capabilities after you have most of the main functionality working. I will provide you with more guidance on how best to implement this in Node and react.
5. If user has logged on, then display a home page with the following options: Browse Portfolio, Browse Companies, Stock Visualizer, and About Us. The user's first and last name should be visible throughout the site on the navigation bar; as well, options to logout and chat should always be visible.
6. Browse Companies. For this view, display a list of companies (and their logos) sorted by name. Each company name will be a link/route to a Single Company view.
7. Single Company. For this view, display the logo and the company name. As well, display two tabs that allow the user to view either the Summary sub-view or the List sub-view.
    a. For the Summary sub-view, the other information for the company. Also display a bar chart of the average close price for each month. You are free to use any react-friendly JS charting library.
    b. For the List sub-view, display a drop-down list with the months of the year. When the user selects a month, display a table with the price information (date, low, high, close) for each day of the month that has data.

8. Browse Portfolio. For this view, use tabs that allow the user to view either the Summary sub-view or the List sub-view.

    a. For the List sub-view, display the user's portfolio information (i.e., the stock symbol, the company name, the number owned, and the current value) in a list. Just like in the first assignment, the user should be able to change the sort order by clicking on the column headings; repeated clicking will toggle between ascending and descending. The symbol and the name will be link/routes to Single Company view. For the current value, it is latest price * number owned.

    b. For the Summary sub-view (which should be the default), display the following information: total number of companies in portfolio, the total number of stocks in portfolio, and the current $ worth of the portfolio. Also display a pie chart displaying a percentage summary of the portfolio information for that user (see 2a in Back-End Requirements).

9. Stock Visualizer. For this view, display a line chart of the close values for a single month for up to three stocks. That is, the x-axis will contain the days, while the y-axis will be money. There should be four drop-down lists: one to select month, the others to select stocks. The drop-down should display symbol and name. Be sure to use different colors for each line.

10. About Us. For this view, list each person in group and their contributions. Provide link to your GitHub repo. Also provide links and info about all third-party libraries used. As well, be sure to provide summary of all the tools and anything else that a future potential employer or client would be interested in knowing about. That is, use this About Us as a way to sell your skills and knowledge to potential clients/employers.

11. Logout. If the user clicks on login, then nothing but the login screen should be accessible.

12. Chat. Whenever a new user logs in, the Node chat server (see 3 in Back-End Requirements) should push a message to the client notifying it of that fact. Display a notification telling the user that User X has logged in. If the user selects the Chat View, then display chat window that displays any received chat messages along with a textbox for entering a chat message and a Send button for submitting the message. When a message is submitted, your Node chat server will push it out to all clients. If a user is in Chat View it will be displayed in the message area. If the user is not in chat view, simply display a notification for 0.5 seconds (or so) with the message and then fade the notification away.

    I would implement this chat facility towards the end. You may need to host the chat server on its own server.

Dev-Ops Requirements

1. While in development, if you want to continue using Cloud9, you will need to find a way to have Node and create-react-app live together in the same workspace. Look online for help. It will require making some changes to the Express port, some changes to the package.json on the React side, and possibly installing some other packages.

   Alternately, you may decide early on to switch to a development environment with Node, Mongo, etc installed locally on your development machine. In that case, you can implement Node services early on and get them running on an external server near the beginning of the project.

2. You must host this on a live server. There are several free tier offerings available from Heroku, Amazon Web Services, Microsoft Azure, AppFog, Digital Ocean, Joyent, and many others. It will take time to get this working so please don't leave this to the last few days!

   Also, make sure the host you choose can't hit you with unexpected fees if suddenly your site gets hit with a surge of requests. Most free tiers will simply stop working if for some reason you start getting real-world request volumes, but to be safe, make sure your hosting option stays free! Alternately, if you decide to spring for a pay host, pick on with a flat monthly or yearly fee.

   I would recommend using Heroku, since it is a popular development-focused hosting environment. You can take one of the following strategies:

   - Use three servers: one to host Node web services, one to host chat server, and one to host react front end. This will be easier (I think) from a division of labour perspective. I will provide more information soon, but I believe the best approach to getting create-react-app up on Heroku is a specialized Heroku buildpack.
   - Have all three on a single server (look online for instructions).

   If using Heroku, you will also need to get a free account on some service (recommend mLab)to host your Mongo database. The free tier at mLab gives you just one database. When I completed this assignment, I created my database, and then imported each json file as separate collections. You will need to use the mongoimport command on your development machine. Interestingly, you can connect to your mLab Mongo database directly on your local machine. To do so requires using a database user name and password (which is not the same as your mLab username and password).

   Some of you may want to keep your site running after the course as a portfolio piece viewable by future potential employers or clients.

3. Your code (front-end and back-end) must be uploaded on GitHub. If you want it to be a public repo, that is fine. If you want it to be private, that is also fine, but be sure to add me as a contributor so I can view and mark it. If you want to have a private repo owned by the group and not by an individual, let me know, and I can provide you with one. If you decide to use three servers (one for web services, one for chat server, one for react client) and use Heroku, then you will need three separate github repos.

   Be sure to make frequent pushes of your code to GitHub.

Advice

I'm hoping this is a realistic and achievable assignment. I haven't had an opportunity yet to complete this assignment myself, so when I do, there is the possibility that I might make some future changes in order to make completing the assignment easier.

I do think the three sets of requirements can be worked on partially in parallel. That is, one person could do some of the dev-ops stuff, someone else could do some back-end stuff, and someone else could do front-end stuff.

I would recommend getting the back-end services completed pretty quickly since ultimately, the front-end requirements will need them.

The front-end requirements are likely to be the largest, so for fairness, everyone in the group should be involved in working on them.

Last time I taught this course, several groups had unexpected problems with getting their assignment to work on their host, so be sure you don't leave hosting too late.