COMP 3512
Assignment #1: Single-Page App *(version 1a)*

Due Sunday February 24, 2019 at midnightish

Overview

This assignment provides an opportunity for you to demonstrate your ability to generate a dynamically updateable single page web application using JavaScript. You will be creating something similar to the final exercise from your JavaScript 3 lab. That lab made use of a small subset of data in a small JSON file that contained all the information you needed. This lab uses a more realistic API.

Beginning

Starting files can be found at: `https://github.com/mru-comp3512-archive/w2019-assign1.git`

Submitting

You will need to share your workspace in c9 with me once it is complete. To do so, click the Share button (it has a gear icon) in the upper-right part of the Cloud9 workspace. Invite me via my email (rconnolly@mtroyal.ca) and be sure to give me RW access (otherwise I can't run anything). If you are having problems with cloud9, feel free to work locally and then submit your assignment to the I: drive; if you submit this way, please send me an email letting me know.

Grading

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

Visual Design	20%
Programming Design	15%
Functionality (follows requirements)	65%

Data Files

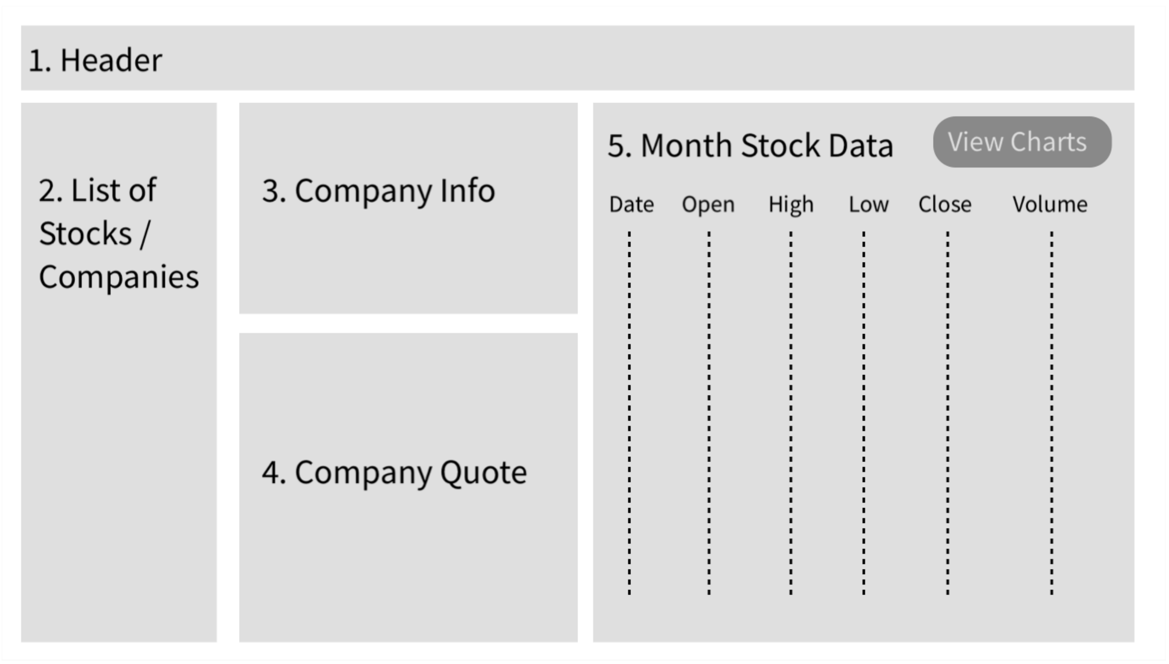
Images for the logos have been provided. Data is from one of my APIs and an external API from https://api.iextrading.com

Requirements

This assignment consists of a single HTML page (hence single-page application or SPA) with the following functionality:

1. Your assignment should have just a single HTML page which **must** be named `index.html`.
2. I expect your page will have significantly more additional CSS styling compared to `lab10-test02.html`. (Last year, many students received minimal design marks due to minimal effort in CSS styling). If you make use of CSS recipes you found online, you must provide references (i.e., specify the URL where you found it) via comments within your CSS file. I would recommend using online resources to look for color schemes if your uncertain of your ability to construct an attractive page. **Failure to properly credit other people's work in your CSS will likely result in a zero grade.**

3. You must write your own JavaScript. That is, no jQuery, no React, no other third-party JavaScript libraries other than one of the specified charting libraries. You have all the knowledge you need from the three JavaScript labs to complete the assignment. If you do find yourself, however, making use of some small snippet of code you found online (say more than 4-5 lines), then you must provide references (i.e., specify the URL where you found it) via comments within your code. **Failure to properly credit other people's work in your JavaScript will likely result in a zero grade.** There is no need to credit code you found in the lab exercises.
4. Most of the functionality in the app can be found in the two sketches shown on the next few pages. Each of these is described in more details below. The first shown below is the **Default View**.



5. **Header**. The page title should be COMP 3512 Assign1. The subtitle should be your name. Add an icon or label named Credits. When the user mouses over this icon/label, please display the following message: "Data provided for free by IEX. View IEX's Terms of Use". Make IEX a hyperlink to `https://iextrading.com/developer/`. Make Terms of Use a hyperlink to `https://iextrading.com/api-exhibit-a/`. This message should be formatted uniquely and should disappear after 5 seconds (use setTimeout).
6. **List of Stocks**. This must display a list of companies. Unlike the JavaScript 3 lab, in which you fetched and displayed info all contained within a single hierarchical JSON file, this assignment uses two discrete APIs: one for retrieving the company list, the other for retrieving company information, quotes, and month's stock data. The URL for the company list API will be:

   ```
   https://www.randyconnolly.com/funwebdev/services/stocks/companies.php
   ```

   This service returns just the most basic information about a few hundred companies. To improve the performance of your assignment, you must store the company data in local storage after you fetch it. Your page should thus check if this company list data is already saved in local storage: if it is then use local data, otherwise fetch it and store it in local storage. This approach improves initial performance by eliminating this first fetch in future uses of the application.

You must sort the companies by the `symbol` property. In the list, display both the company logo and the stock symbol; the logo will have to be quite small. Each list item (the logo+stock symbol) must be clickable.

When the user clicks on the logo+symbol in the list, then populate Company Info, Company Quote, and Monthly Stock Data sections with the data for the clicked stock/company.

Finally, above the list, provide a filter textbox. When the user types into the textbox (change event), filter the list so it only shows companies whose symbol begins with the same letters typed into the box. If there are no matches, it should be obvious.

7. **Company Info**. When the user clicks on a company in the list, then your program will need to fetch the company info from the following API:

   ```
   https://api.iextrading.com/1.0/stock/xxx/company
   ```

   where xxx is the `symbol` property value for the clicked-on company/stock.

   In this section, display the following information from the JSON: `symbol`, `companyName`, `exchange`, `industry`, `description`, `CEO`, `website`, `industry`, `sector`, and `tags`. The `website` should be an actual working link. Also display a larger version of the logo. I expect this info to be nicely formatted and laid out sensibly.
8. **Company Quote**. Displays current stock price information from the following API:

   ```
   https://api.iextrading.com/1.0/stock/xxx/quote
   ```

   where xxx is the `Symbol` property value for the clicked-on company/stock.

   In this section, display the following information from the JSON: `open`, `close`, `high`, `low`, `week52High`, and `week52Low`. These are currency values in US dollars so formatted them accordingly. Also include the `latestSource` field. I expect this info to be nicely formatted and laid out sensibly.
9. **Month Stock Data**. Display a month of data for the current company/stock from the following API:

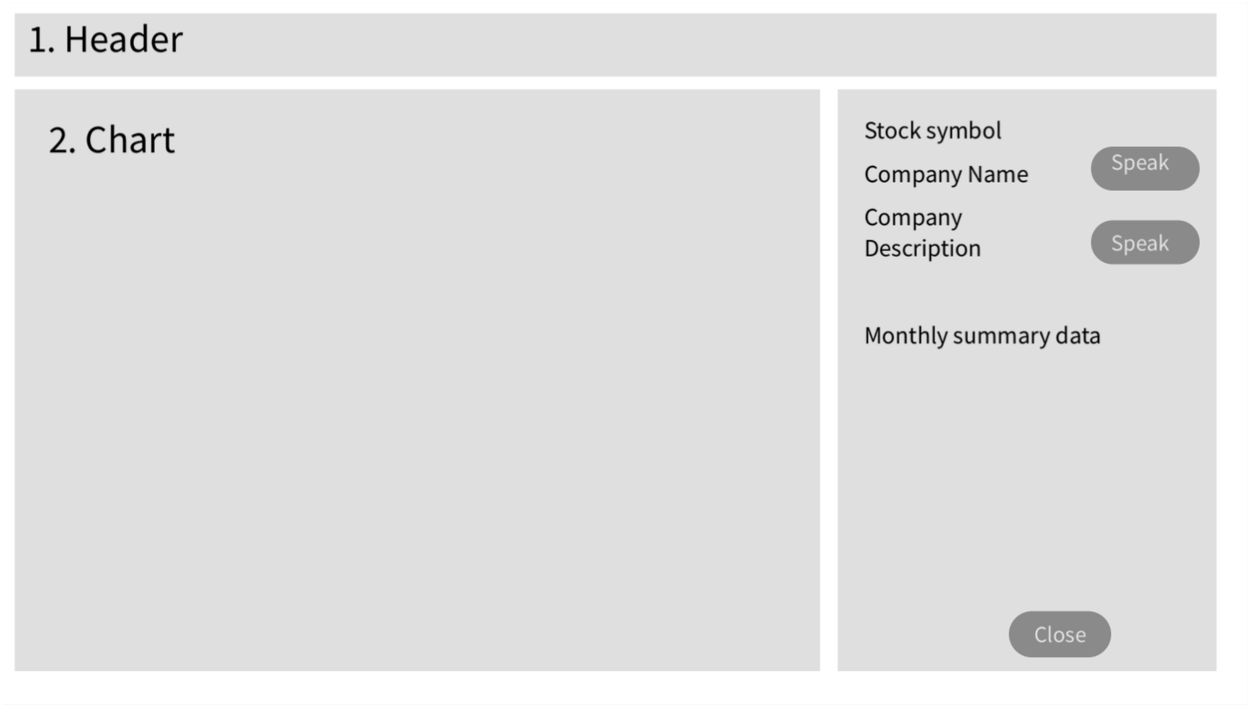
   ```
   https://api.iextrading.com/1.0/stock/xxx/chart/1m
   ```

   where xxx is the `Symbol` property value for the clicked-on company/stock.. Display the Date, `open`, `close`, `high`, `low`, and `volume` fields.

   The headings at the top of each column should be clickable. When the user clicks on a column heading, the month data will be redisplayed so that it is sorted on the value just clicked. When first displayed, sort by date. There should only be one initial fetch; the sorting should operate on the already fetched data.

   When the user clicks on the View Charts button, then sections 2, 3, 4, 5 on the page will be hidden and replaced with the Chart View, described next.

10. **Chart View**. When the user clicks on the View Charts button, sections 2, 3, 4, 5 will be hidden and replaced with just the chart view for the stock:



For the Chart View, display the following fields: `symbol`, `name`, and `description`. You will also need to calculate and display the following information from the monthly stock data: average close, minimum close, average close, average volume, total volume. I expect this information to be nicely formatted (e.g., currency as currency) and laid out sensibly.

The Speak buttons will use the speech synthesizer to say either the `companyName` or the `description`.

The Close button will hide this view and show instead the **Default View**.

For the chart, use either echarts (https://ecomfe.github.io/echarts-doc/public/en/index.html) or chart.js (https://www.chartjs.org/). These are analogous to Google Maps in that they are an external library with an API you will need to learn and discover. Both of these require either an empty <div> or an empty <canvas> element which will end up being the container for the chart generated by the JavaScript charting library.

The chart must be a line chart with the month dates on the x-axis and close price on the y-axis. I would recommend making the colors in your chart match the colors in your page's color scheme. Making the chart look nice is in your interest!

For extra credit, add the volume (probably as a bar chart) as a additional data series to your chart. This volume series will require its own separate y-axis.

Hints

1. Test the APIs out first in the browser. Simply copy and paste the URLs into the browser address bar. The JSON can be hard to understand because it doesn't have white space, so use an online JSON formatter (such as https://jsonformatter.curiousconcept.com) via copy and paste to see the JSON structure in an easier to understand format. Alternately, install a json viewer extension in chrome.
2. Remember that JSON and JavaScript are case sensitive. For instance, it is easy to type in `CompanyName` and it won't work because the JSON field is actually `companyName`.
3. Your HTML will include the markup for both **Default View** and the **Chart View**. Initially, the later will initially use CSS to set its `display` to `none`. Your JavaScript code will programmatically hide/unhide (i.e., change the `display` value) the relevant markup container for the two views.
4. Most of your visual design mark will be determined by how much effort you took to make the two views look well designed. Simply throwing up the data with basic formatting will earn you minimal design marks.
5. Most of your programming design mark will be based on my assessment of your code using typical code review criteria. For instance, did you modularize your code using functions? Are your functions too long? Is the code documented? Do you have code duplication (you shouldn't)? Did you make use of object-oriented techniques? Are variables and functions sensibly named? Is your code inefficient (e.g., fetching the same data repeatedly)? Are you using outdated JavaScript techniques (e.g., inline coding, `XmlHttpRequest`, etc)? Is the code excessively reliant on found code from Stack Overflow, etc?
6. We didn't get a chance to discuss this yet in class, but when constructing single-page applications in JavaScript, you sometimes need to "insert" data into dynamic HTML elements that you modify/create in JavaScript. In HTML5, this is supported via the `data-X` attribute.
7. For the Speak buttons and the Close button, be sure to only add click event handlers for these buttons only once when the page loads (and not every time you display a chart).