

CAB230 Web Computing

Assignment 1 Specification

The Stocks Client Application

(Official) Release Date: Monday, April 20 2020

Submission Date: Wednesday, May 13 2020

Weighting: 45% of Unit Assessment

Task: Individual Project

Introduction:

This assignment requires that you develop a React-based web application to allow users to view and analyse stock market statistics drawn from a database that we have created for this purpose and exposed to you via a REST API. We will talk about this API in detail below, and you will implement a version of this backend as your task for the second assignment.

The aims of this assignment are to:

- Allow you to build a sophisticated client web application using modern approaches
- Provide experience in querying REST APIs and presenting the results for your users
- Provide experience with modern web technologies including node.js and React

In this specification, we will not be too prescriptive, and you should feel free to exceed the specification if you wish to explore the technologies. But overall the assignment task is one best completed in stages, with each new step allowing you to look at a higher grade level.

The Data:

The dataset consists stock market data drawn from an American exchange from November 2019 through to March 2020. The more precise dates are given below in the format that you will use in subsequent queries:

from 2019-11-06 ***to*** 2020-03-24

We will all work with the same data, but we reserve the right to update our database with additional records prior to marking your assignments, so please don't be tempted to hard code your responses. The easiest way to see what is going on is to look at an example. The screenshot below shows a query on the backend database holding the data. For each *listed* company we will have an identifier which we call a *symbol*, and a snapshot of the trading prices and volume for that company over the given day. Note that the exchanges are not open on weekends and so there will be gaps in the timestamps shown. Here we are looking at (most

of) the records obtained when we query on the symbol AAL, which is used by American Airlines.

```
mysql> select * from stocks where symbol="AAL";
```

timestamp	symbol	name	industry	open	high	low	close	volumes
2020-03-24	AAL	American Airlines Group	Industrials	10.9	11.36	10.01	10.25	55494100
2020-03-23	AAL	American Airlines Group	Industrials	10.6526	12	10.3	10.38	62681800
2020-03-20	AAL	American Airlines Group	Industrials	11.6	12.16	10.01	10.29	71584500
2020-03-19	AAL	American Airlines Group	Industrials	14.24	14.28	10.17	11.65	78458800
2020-03-18	AAL	American Airlines Group	Industrials	15.785	18.49	14.79	15.58	58055800
2020-03-17	AAL	American Airlines Group	Industrials	12.4065	16.2	12	15.92	84916800
2020-03-16	AAL	American Airlines Group	Industrials	15.3	15.6	13.12	14.31	58376100
2020-03-13	AAL	American Airlines Group	Industrials	14.05	15.75	13.33	13.45	53968500
2020-03-12	AAL	American Airlines Group	Industrials	16.31	16.49	15.52	16.26	43778200
2020-03-11	AAL	American Airlines Group	Industrials	15.82	17.67	14.61	17	56972700
2020-03-10	AAL	American Airlines Group	Industrials	14.87	15.79	14.46	14.75	42558000
2020-03-09	AAL	American Airlines Group	Industrials	15.02	17.12	14.8	15.97	54505000
2020-03-06	AAL	American Airlines Group	Industrials	17.54	17.65	15.98	16.04	44656800
2020-03-05	AAL	American Airlines Group	Industrials	18.36	18.55	17.3	18.53	44362100
2020-03-04	AAL	American Airlines Group	Industrials	19.66	19.79	17.8	17.85	35402500
2020-03-03	AAL	American Airlines Group	Industrials	19.05	19.06	17.51	18.86	38738000
2020-03-02	AAL	American Airlines Group	Industrials	19.8	20.35	18.77	19.05	37166400
2020-02-28	AAL	American Airlines Group	Industrials	20.85	22.48	19.77	20.6	31897300
2020-02-27	AAL	American Airlines Group	Industrials	23.57	23.75	22.11	22.31	19631400
2020-02-26	AAL	American Airlines Group	Industrials	25.62	25.75	23.05	23.12	22296900
2020-02-25	AAL	American Airlines Group	Industrials	25.98	26.11	25	25.45	21713200
2020-02-24	AAL	American Airlines Group	Industrials	28.12	28.3	27.17	27.82	11256500
2020-02-21	AAL	American Airlines Group	Industrials	28.24	28.9	28.2024	28.51	10093400
2020-02-20	AAL	American Airlines Group	Industrials	28.84	28.88	28.31	28.33	8401020
2020-02-19	AAL	American Airlines Group	Industrials	29.04	29.345	28.49	28.63	10486500
2020-02-18	AAL	American Airlines Group	Industrials	30.01	30.27	29.14	29.2	5524790
2020-02-17	AAL	American Airlines Group	Industrials	30.03	30.4	29.73	30.09	9119430
2020-02-14	AAL	American Airlines Group	Industrials	30	30.78	29.99	30.47	9315380
2020-02-13	AAL	American Airlines Group	Industrials	29.06	29.94	28.97	29.84	8977740
2020-02-12	AAL	American Airlines Group	Industrials	28.19	28.81	27.98	28.79	15134300
2020-02-11	AAL	American Airlines Group	Industrials	28.06	28.55	27.78	28.38	6231750

The fields are described in more detail in the table below:

Field	Description	Example
Timestamp	A time stamp (year, month, day) of the interaction	2020-03-10
Symbol	The stock's symbol	AAPL
Name	The company name	Apple Inc.
Industry	The industry that describes the company	Information Technology
Open	The price at opening on that day	228.08
High	The highest price that day	251.83
Low	The lowest price that day	228.00
Close	The price at close on that day	229.24
Volume	The number of shares traded in that day	100,423,000

Table 1: The information in the dataset

We do not work with the database directly, but every successful response from the API will include some subset of the records held in the database. There are 49500 records from 495 listed companies. Your task will be to display the results from queries over that data set. If the expected response is a single record, you should tailor your display component to suit. If the result is an array of records – as will often be the case – then obviously we will expect that your application will include some use of a table display. We have provided you with a

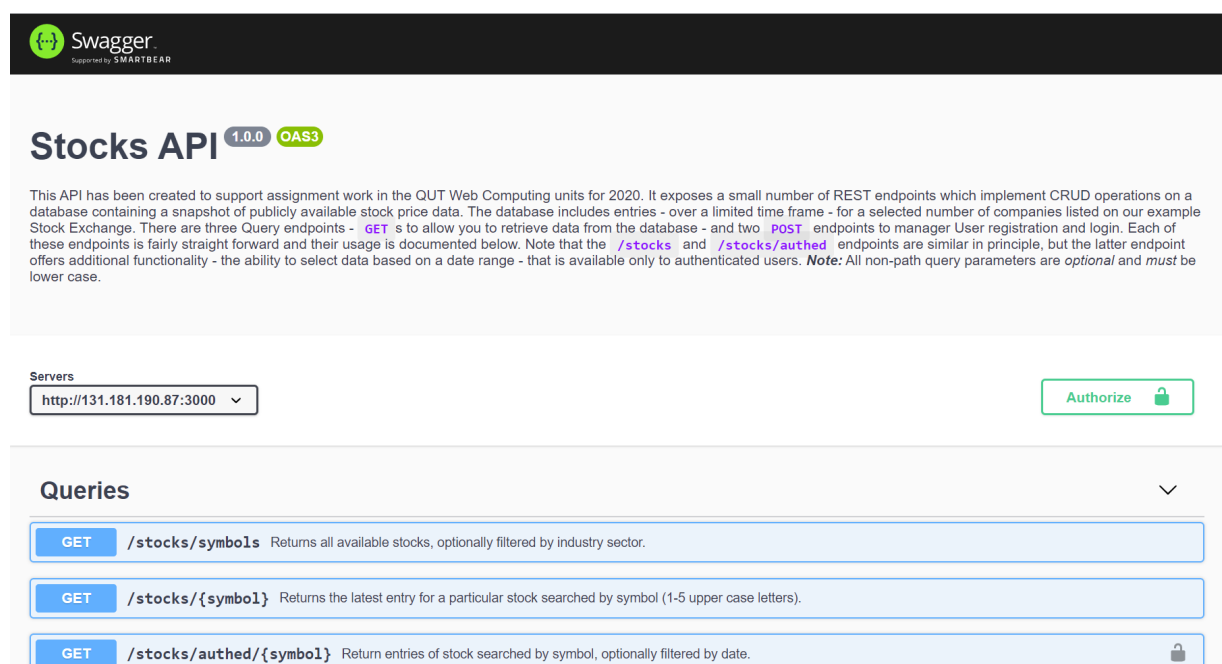
worksheet to help you, and the component readily supports advanced features such as filtering and sorting. We now consider the REST API in more detail.

The REST API:

The REST API is documented on the page at: <http://131.181.190.87:3000/>. The server is behind the QUT firewall, with remote access via the QUT VPN: <https://qutvirtual4.qut.edu.au/group/student/it-and-printing/wi-fi-and-internet-access/accessing-resources-off-campus>

We will not reproduce the endpoint documentation here – please use the Swagger docs provided as they will be maintained more regularly than the assignment spec and they have executable examples. Most of the endpoints are available to everyone, but there is one query route which requires authentication – think of this as a site membership. And if, like most of the class, you have never seen Swagger before, please have a brief look here: <https://swagger.io/docs/specification/about/>. Essentially Open API allows very professional documentation to be generated quickly and painlessly. It is really helpful.

Task highlight: Your application must allow the user to navigate cleanly between the query and user functions. The precise organisation will depend on your design, but these must be clearly separated in a menu. **Please see the Styling React Components worksheet for very good navigation examples using React.**



Swagger
Supported by SMARTBEAR

Stocks API 1.0.0 OAS3

This API has been created to support assignment work in the QUT Web Computing units for 2020. It exposes a small number of REST endpoints which implement CRUD operations on a database containing a snapshot of publicly available stock price data. The database includes entries - over a limited time frame - for a selected number of companies listed on our example Stock Exchange. There are three Query endpoints - `GET` s to allow you to retrieve data from the database - and two `POST` endpoints to manage User registration and login. Each of these endpoints is fairly straight forward and their usage is documented below. Note that the `/stocks` and `/stocks/authed` endpoints are similar in principle, but the latter endpoint offers additional functionality - the ability to select data based on a date range - that is available only to authenticated users. **Note:** All non-path query parameters are *optional* and *must* be lower case.

Servers

`http://131.181.190.87:3000` Authorize

Queries

- `GET` `/stocks/symbols` Returns all available stocks, optionally filtered by industry sector.
- `GET` `/stocks/{symbol}` Returns the latest entry for a particular stock searched by symbol (1-5 upper case letters).
- `GET` `/stocks/authed/{symbol}` Return entries of stock searched by symbol, optionally filtered by date.

As discussed in the overview shown, the endpoints are split into two categories: *Queries* and *Users*. If you have not registered with the system before, then you should *register* and then you will be able to *login*. We will discuss authentication properly below and in the lecture in week 7, but for now just assume that you are able to login to the site.

Task highlight: Your application must handle the registration and login processes. This basically requires that you implement two POST operations that have a very similar format. You should design or use a form component to handle these, though the responses from the server will be different.

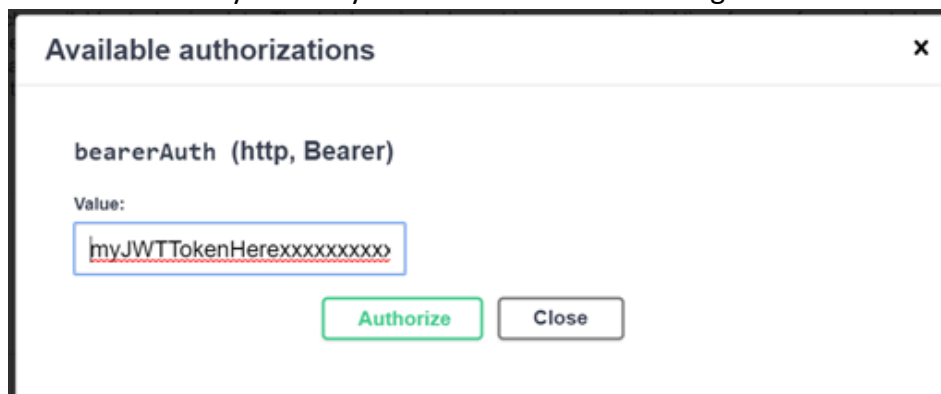
Once you have logged in, the response JSON will look something like the following, excepting that the token will be much longer and far more random (I have edited a real one).

```
{
  "token": "eyJxxxY5NzAzfytyyytdbappulxUo",
  "token_type": "Bearer",
  "expires_in": 86400
}
```

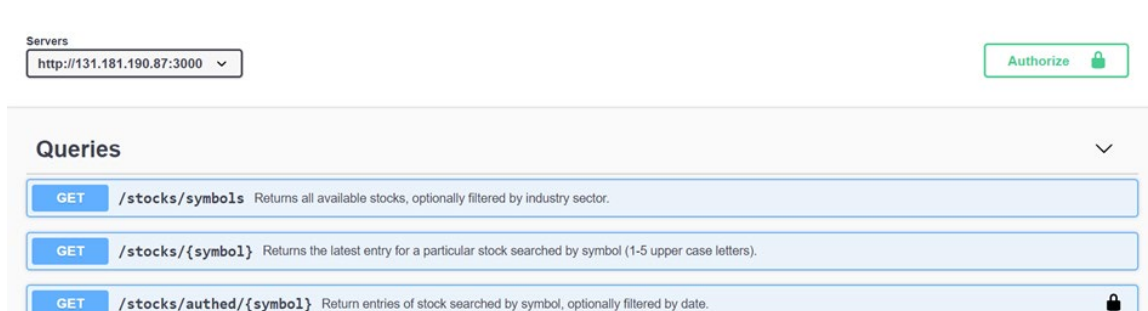
We will show you how to work with JWT tokens (<https://jwt.io/>) at the next lecture, but for now you will need this token to use the authenticated routes on the Swagger. If you take a look at the Swagger image above, you will see a green button with an open lock icon:



Click on this button and you will see the dialogue below. Copy the token from the JSON above and paste it where I have my fake `myJWTTokenHere` in the image:



Click on the green button and close the dialogue and you will see that the locks are now shut and the grey icon is now black. You are now able to use the authenticated route in Swagger.



We now consider the main query routes in turn. Each of them is a GET, and the first two are not authenticated. The error conditions and precise usage you can discover by playing with the Swagger docs. Here we are concerned with the types of data that come back and what we will expect you to do:



This first route is very simple and just returns an array of stocks. If you don't use the optional query parameter (*industry*) then you will get some 495 stock objects. The fields in this response are as shown below. I have edited the JSON to remove all the entries between Accenture and Xilinx. Note that this list has been filtered to show only those companies from the industry sector *Information Technology*.

```
[
  {
    "name": "Apple Inc.",
    "symbol": "AAPL",
    "industry": "Information Technology"
  },
  {
    "name": "Accenture plc",
    "symbol": "ACN",
    "industry": "Information Technology"
  },
  {
    "name": "Xilinx Inc",
    "symbol": "XLNX",
    "industry": "Information Technology"
  },
  {
    "name": "Xerox Corp.",
    "symbol": "XRX",
    "industry": "Information Technology"
  }
]
```


Task highlight: The obvious display for this response will be in a three-column formatted table. In designing this part of your application you will need to consider the inputs needed to support this query to the server – hitting this route in response to a button click for example – and the appropriate way to specify the text for the `industry` filter. Explore the results that come back when you use a prefix instead of the full search term. You then need to think about how you are going to handle the data when they reach your app – pagination, sorting by column, filtering by column – typing in partial values and seeing only those which match, styling, error conditions, no data found.

GET `/stocks/{symbol}`

This again is a simple unauthenticated route, but it relies critically on the symbol, which as you have seen above is a short upper case string of letters. The rule is that it must contain at least one and no more than five characters. The response below is for `/stocks/AAL`:

```
{
  "timestamp": "2020-03-23T14:00:00.000Z",
  "symbol": "AAL",
  "name": "American Airlines Group",
  "industry": "Industrials",
  "open": 10.9,
  "high": 11.36,
  "low": 10.01,
  "close": 10.25,
  "volumes": 55494100
}
```

Task highlight: Here the response is more limited. A table is the minimum requirement, but there are other approaches that might make more sense here. Your display could have more of the look and feel of a dashboard. Some charting to show the variation over the day might also be considered. If you had full control of the database, you might check whether this volume is big. But you don't have that available, so that might just be nicely formatted text. Spend some time designing on paper and then start to think about some mock-ups and then put it into practice.

GET `/stocks/authed/{symbol}` 

The final endpoint is authenticated. By default it has the same behaviour as the previous one. But if the user is logged in, then they are able to query a particular stock between two dates specified via the query parameters *from* and *to*. The result of such a query is an array of stock entries like those seen above. Here is the first part of the array that results when the stock symbol is again AAL and the dates are as shown in the Swagger example:

```
[
  {
    "timestamp": "2020-03-19T14:00:00.000Z",
    "symbol": "AAL",
    "name": "American Airlines Group",
    "industry": "Industrials",
    "open": 11.6,
    "high": 12.16,
    "low": 10.01,
    "close": 10.29,
    "volumes": 71584500
  },
  {
    "timestamp": "2020-03-18T14:00:00.000Z",
    "symbol": "AAL",
    "name": "American Airlines Group",
    "industry": "Industrials",
    "open": 14.24,
    "high": 14.28,
    "low": 10.17,
    "close": 11.65,
    "volumes": 78458800
  },
]
```

Task highlight: Here a table is the obvious basic approach. You might also think about the sorting and the filtering as before, but there are other approaches that might make more sense here. Your display could have more of the look and feel of a dashboard. You might also think about charting – showing us how the prices and volumes track over time. You need to think again about how we choose these date – is it a text box or a widget or whatever.

The Requirements:

The requirements for this assignment are ultimately pretty straightforward, but there are a lot of choices to be made. These decisions are obvious after you have done a few apps, but

here you are still putting it all together for the first time. At the most basic level, you must develop a client side web application that allows the user to work with each of the endpoints, but without the user ever really being aware of the underlying work. The user knows that they need to specify a symbol or an industry sector or whatever, but they don't want to know anything about the underlying calls. In the task highlights above I have given you some guidance on what we need to see, and you should think very carefully about the forms that allow us to enter the data and the components that display the responses, and the ways in which we move between them. We expect that you should be able to successfully process all of the endpoints. You should choose suitable interface elements to display and interact with the information. We will give you some latitude in this, and you can also get some ideas from the discussion below.

Some Guidance on Design:

Landing Page:

Below is an example initial landing page. This is at the very lowest level of expectations and is intended to get you started. There should be a welcome message like the one I have given, but chosen to suit your application, you might choose a hero image and your layout should be chosen to facilitate the information that will come when we hit the endpoints and use the display components you are developing. You should think about how you intend to present the information together.

[Home](#) | [Stocks](#) | [Quote](#) | [Price History \(restricted\)](#) [Register](#) | [Login](#)

Stock Prices

Welcome to the Stock Analyst portal. Click on Stocks to see the available companies, Quote to get the latest price information by stock symbol, or choose Price History to sample from the most recent one hundred days of information for a particular stock.

Routing:

For each route in this client side application you should think about how you are going to handle it – have a look at the React Router examples in the lectures and in the Styling Components worksheet. You may use a basic HTML menu but the use of React routing will attract more credit.

Choice of Widgets:

Each endpoint call needs parameters and you will need to choose how to specify them. In the case of the Industry Sector filtering, it may work to have a list because there aren't so many

alternatives, but the endpoint supports prefix queries like 'Consum' – giving results for both Consumer Discretionary and Consumer Staples. A simple list would not map well here as you wouldn't be taking full advantage of the endpoints. Your industry sector choices are as follows:

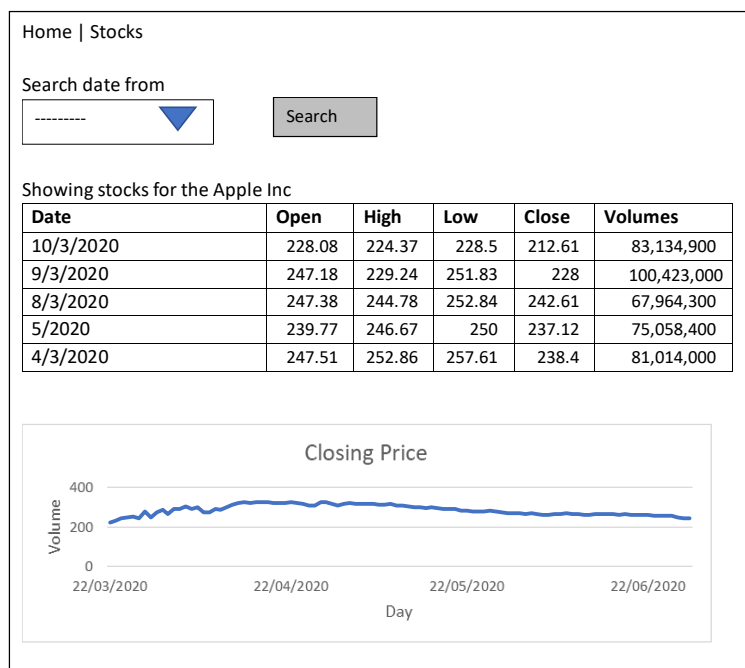
Health Care	Financials
Industrials	Real Estate
Consumer Discretionary	Materials
Information Technology	Energy
Consumer Staples	Telecommunication Services
Utilities	

Table Components:

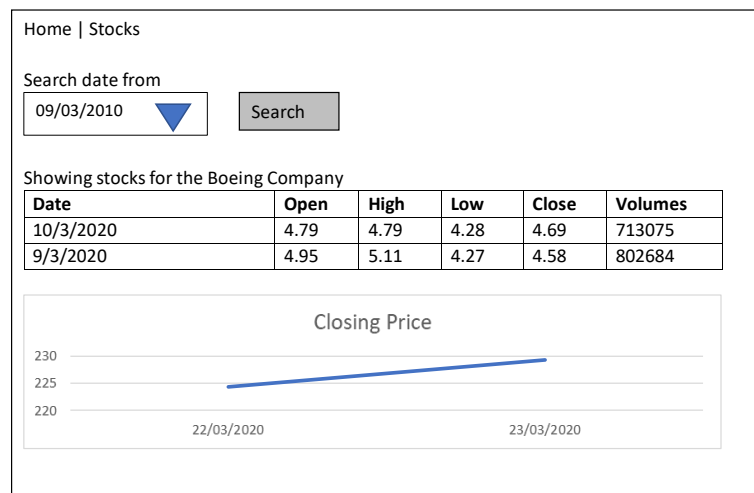
As discussed in the *Task Highlights*, the table structures are crucial to the successful display of the data from the server. Using a sophisticated table component like ag-Grid-react means that we can do a lot of things in the client application without going back to the server. So, if we perform a `/stocks/symbols` query – especially if it includes all of the available company records – what can we offer the user without going back to the server?

Charting:

For more sophisticated endpoints which offer a lot of data, line or bar charting or some other mechanism for showing the data will be very well received. You may use standard chart libraries to produce these, and as shown below there may well be a mix of table and chart displays.



In keeping with the earlier discussion, we will also look very favourably on applications which allow the user to select or filter on the client side and have the table and the charting update in response to these changes, as shown in a simple mock up below:



Grading:

The marking for this assignment will be governed by the CRA rubric, and this will take account of a number of aspects of the assignment process. These will include:

1. The overall level of functionality successfully implemented
2. The usability of the application
3. The robustness of the application
4. Evidence of a professional approach to design
5. Evidence of a professional approach to development and code quality
6. The quality of the professional report and submission video (TBC)*

Full details of this split will be found in the CRA document, which will also be made available in the assessment section of BB. Here we are concerned only with the marks for functionality, application usability and robustness. The precise marks awarded may be reduced as a result of features which are only partially implemented or error-prone or components which are poorly chosen and so on. But as an approximate guide, these are our expectations:

- **[Grade of 4 level]:** A simple React app with limited styling which implements the unauthenticated Query endpoints and presents the data cleanly using table components. User endpoints and the authenticated queries may not have been implemented successfully and the client side processing in the table components is very limited. A react-strap table component or similar would suffice here.
- **[Grade of 5 level]:** At this level we would expect successful implementation of the User endpoints and the authenticated Query route. Table component usage and client side processing would be expected to use the standard functionality provided by a component such as ag-Grid-react, and there should not be excessive querying of the server.
- **[Grade of 6 and 7 level]:** Here the expectation is that you have exceeded the grade of 5 level in that all of the basics are there and working smoothly. Navigation is handled using React Router, React forms are used for the data entry and there is evidence of a really good

match between your components and the services that they are using. We would expect some use of charting or other information graphics to show how the stock prices are varying and advanced use of the client side processing. The split between the grade of 6 and grade of 7 will involve a tradeoff between the features and the quality of the execution, and we will happily give you an opinion on your proposed application. For charting we recommend the use of chartJS (<https://www.chartjs.org/>), especially via the widely used React wrapper you can find here: <https://www.npmjs.com/package/react-chartjs-2>. d3 (<https://d3js.org/>) is a popular choice, but it is an advanced library and you shouldn't attempt it unless you have prior experience.

A reminder that these are guides only, and that the mark levels are based on successful implementation of the features mentioned. If unsure, please get in touch and discuss them with us.

Submission:

The submission instructions for this assignment cannot be finalised at the time of release – or at least at the time of unofficial release. It is our practice to assess these assignments via a face to face demo and we cannot use that approach in the middle of a pandemic. It is our intention that the submission will require a video demonstration of your application, with the option of us requiring a Zoom-based face to face. We cannot confirm the details of this at this stage as we are waiting on guidelines in respect of file sizes and platforms. We will have this in place by Week 8, well before the due date for the assignment.

Your submission will *definitely include* the following components:

The Code:

The code archive should be based on the React application structure that you inherit from create-react-app or codesandbox. We will not be impressed if this has been disrupted badly or there are additional folders with a seemingly random purpose and organisation.

Above all, however, we will need you to pay close attention to your ***node_modules***:

Node apps involve installation of packages, and that this leads to the installation of `node_modules`, and that eventually these take up rather a lot of space. Please delete them. And then look around through the directory structures again, and delete any others that you missed the first time. We don't want to see them, we don't want to store them, and you don't want to wait while they upload.

And in case you missed it the first time, please delete your `node_modules`.

Report:

We will expect a short report and user guide, generally running to 10 pages or so, including a lot of screenshots. The report is there to help us better understand your application, and to get your thoughts on the process, the bits that worked and the bits that didn't, and the

usability and correctness of the application. We will provide an example and a template for this report.

The report will include the following:

1. Introduction – telling us what was implemented and what wasn't, showing a few screenshots to illustrate the functionality. This will probably occupy a page or a bit more.
2. Technical description of the application. This section is to allow you to talk about the APIs used, to show us any tricky data flows, and to discuss technical issues that caused you problems. This is especially important if something doesn't actually work.
3. Application Design and Usability. This section is a quick discussion of the choices you made in designing the app, and some frank assessments of its usability.
4. Testing and limitations – test plan, results, compromises. We do not expect automated testing. We do expect you to verify that the application works for each of the use cases. Screen shots are your friend.
5. References
6. Appendix: brief user guide

The report should be entitled `report.pdf`

Video Demonstration (To be confirmed):

You will need to record a video demonstration of your app, showing all of the actions outlined in this document. In particular you will need to:

- Navigate to the landing page of your app
- Show us the use of each of the implemented end points and their principal use cases:
 - a. Show all the stocks with their associated detail
 - b. Show the ability to limit by industry sector
 - c. Show the ability to get information for a particular stock
 - d. Show the ability to register a new user and login
 - e. Show the ability to limit stock from a particular date
 - f. Demonstrate error handling

We will provide more specific information closer to time – instructions on editing, on formats and file sizes, and on the options for linking a higher resolution alternative file.

The video report should be entitled `video.mp4`

Final Submission:

The final submission instructions are not yet confirmed, but ideally we will require that you create a directory called `assign` which should include:

- The client application code directory
- The report
- The video demonstration

You should then zip up assign and uploaded to blackboard. We will mark the last attempt received and we will ignore all the others.

The CRA rubric will be released as a separate file.