HOMEWORK ASSIGNMENT 3

CSCI 571 – Spring 2024

Abstract

Ajax, JSON, Angular, Bootstrap, MongoDB, RWD, Node.js, and the FinnHub Stock API

Marco Papa

papa@usc.edu

Assignment 3: Ajax, JSON, Responsive Design and Node.js

Stock Search

(AJAX/JSON/HTML5/Bootstrap/Angular /Node.js/Cloud Exercise)

1. Objectives

- Get familiar with the AJAX and JSON technologies
- Use a combination of HTML5, Bootstrap and Angular on client side
- Use Node.js on server side
- Get familiar with Bootstrap to enhance the user experience using responsive design
- Get hands-on experience of Cloud services hosting NodeJS/Express
- Learn to use popular APIs such as the Finnhub API, Polygon.io API and Highcharts API
- Learn how to manage and access a NoSQL DBMS like MongoDB Atlas, in the cloud

2. Background

2.1 AJAX and JSON

AJAX (Asynchronous JavaScript + XML) incorporates several technologies

- Standards-based presentation using CSS
- Result display and interaction using the Document Object Model (DOM)
- Data interchange and manipulation using XML and JSON
- Asynchronous data retrieval using XMLHttpRequest
- JavaScript binding everything together

See the class slides on D2L Brightspace.

JSON, short for JavaScript Object Notation, is a lightweight data interchange format. Its main application is in AJAX web application programming, where it serves as an alternative to the use of the XML format for data exchange between client and server. See the class slides on D2L Brightspace.

2.2 Bootstrap

Bootstrap is a free collection of tools for creating responsive websites and web applications. It contains HTML and CSS-based design templates for typography, forms, buttons, navigation, and other interface components, as well as optional JavaScript extensions. To learn more details about Bootstrap please refer to the lecture material on Responsive Web Design (RWD). We recommend using **Bootstrap 4.6** through **5.3**, **Angular 12** through **17**, and **ng-bootstrap 11** through **16** in this assignment. See the RWD class slides on D2L Brightspace for the list of dependencies between these various versions.

2.3 Cloud Services

2.3.1 Google App Engine (GAE)

Google App Engine applications are easy to create, easy to maintain, and easy to scale as your traffic and data storage needs change. With App Engine, there are no servers to maintain. You simply upload your application and it's ready to go. App Engine applications automatically scale based on incoming traffic. Load balancing, microservices, authorization, SQL and NoSQL databases, memcache, traffic splitting, logging, search, versioning, roll out and rollbacks, and security scanning are all supported natively and are highly customizable.

To learn more about GAE support for Node.js visit this page:

https://cloud.google.com/appengine/docs/standard/nodejs/

2.3.2 Amazon Web Services (AWS)

AWS is Amazon's implementation of cloud computing. Included in AWS is Amazon Elastic Compute Cloud (EC2), which delivers scalable, pay-as-you-go compute capacity in the cloud, and AWS Elastic Beanstalk, an even easier way to quickly deploy and manage applications in the AWS cloud. You simply upload your application, and Elastic Beanstalk automatically handles the deployment details of capacity provisioning, load balancing, auto-scaling, and application health monitoring. Elastic Beanstalk is built using familiar software stacks such as the Apache HTTP Server, PHP, and Python, Passenger for Ruby, IIS for .NET, and Apache Tomcat for Java.

To learn more about AWS support for Node.js visit this page:

https://aws.amazon.com/getting-started/projects/deploy-nodejs-web-app/

2.3.3 Microsoft Azure

Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through a global network of Microsoft-managed data centers. It provides software as a service (SaaS), platform as a service (PaaS) and infrastructure as a service (IaaS) and supports many different programming languages, tools, and frameworks, including both Microsoft-specific and third-party software and systems.

To learn more about Azure support for Node.js visit this page:

https://docs.microsoft.com/en-us/javascript/azure/?view=azure-node-latest

2.4 Angular

Angular is a toolset for building the framework most suited to your application development. It is fully extensible and works well with other libraries. Every feature can be modified or replaced to suit your unique development workflow and feature needs. Angular combines declarative templates, dependency injection, end-to-end tooling, and integrated best practices to solve development challenges. Angular empowers developers to build applications that live on the web, mobile, or the desktop.

For this homework, Angular 12+ (Angular 12, through 17) can be used, but **Angular 12 is recommended**. Please note Angular 12+ will need familiarity with Typescript and component-based programming.

To learn more about Angular, visit this page:

https://angular.io/

2.5 Node.js

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js package ecosystem, **npm**, is the largest ecosystem of open-source libraries in the world.

To learn more about Node.js, visit:

https://Node.js.org/en/

Also, **Express.js** is strongly recommended. Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It is in fact the standard server framework for Node.js.

To learn more about Express.js, visit:

http://expressjs.com/

Important Note: All APIs calls should be done through your Node.js server

3. High-Level Description

In this exercise you will create a webpage that allows users to search for stocks using the Finnhub API and display the results on the search page. The application evolves from the previous homework.

A user will first open a page as shown below in **Figure 1**, where the user can enter a stock ticker symbol and select from a list of matching stock symbols using "autocomplete." A quote on a matched stock symbol can be performed. The description of the Search Box is given in **Section 3.1**. Instructions on how to use the API are given in **Section 4**. All implementation details and requirements will be explained in the following sections.

There are 4 routes for this application:

a) Home Route ['/'] redirected to ['/search/home']– It is the default route of this application.
b) Search Details Route ['/search/<ticker>'] – It shows the details of the <ticker> searched
c) Watchlist Route ['/watchlist'] – It displays the watchlist of the user.
d) Portfolio Route ['/portfolio'] – It displays the portfolio of the user.

When a user initially opens your web page, the initial search page should look like in **Figure 1**.

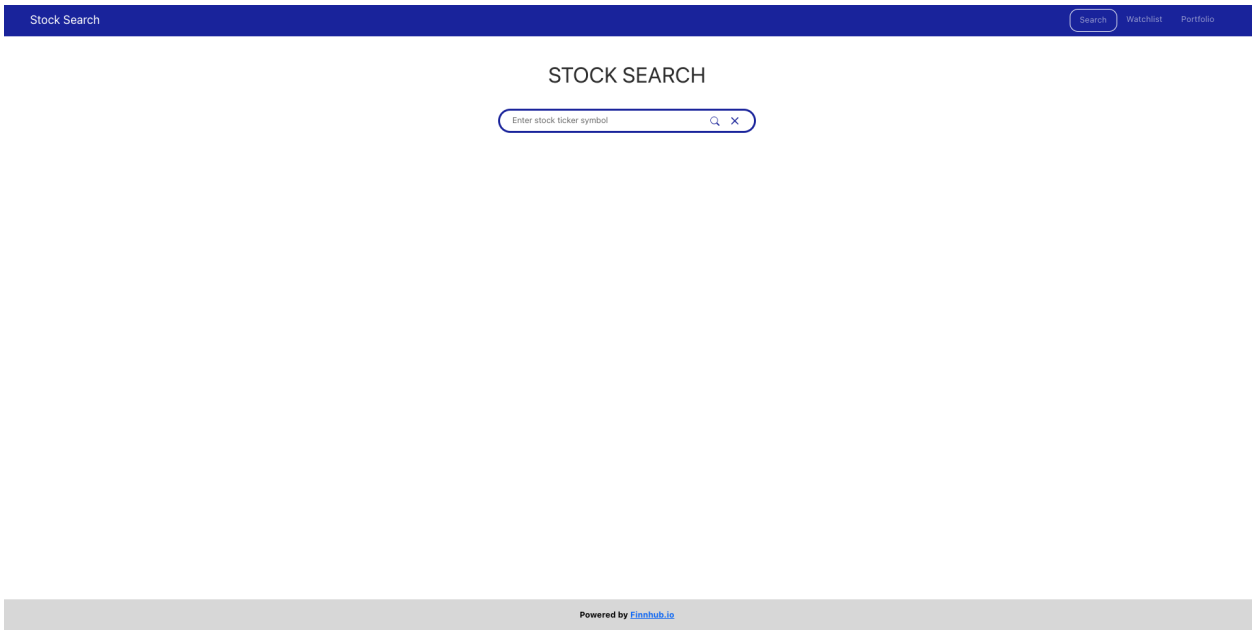


Figure 1: Initial Search Page

3.1 Search Page / Homepage

3.1.1 Design

You must replicate the **Search Bar** displayed in **Figure 1** using a **Bootstrap form**. The Search Bar contains three components.

1. **Stock Ticker**: This is a text box, which enables the user to search for valid stocks by entering keywords and/or accepting a suggestion of all possible tickers. Notice the "helper" text inside the search box.
2. **Search Button**: The "Search" button (which uses the widely used search icon), when clicked, will read the value from the textbox and send the request to the backend server. On a successful response, details for that stock will be displayed.
3. **Clear button:** The 'clear' (cross marked) button, would clear out the currently searched results page and show the initial search page.

3.1.2 Search Execution

Search can be executed in the following ways:

1. Once the user enters a ticker symbol and directly presses the Return key or clicks on the "Search" button, without using the auto-complete suggestion, your application should make an HTTP call to the Node.js script hosted on GA/AWS/Azure back end (the Cloud Services). The Node.js script on Cloud Services will then make a request to the Finnhub API services to get the detailed information. If the entered ticker is invalid and no data is found, an appropriate error message should be displayed. If valid stock data is found, the search results should be loaded.
2. Once the user starts typing a ticker symbol, autocomplete suggestions (See **Section 3.1.3** below) will populate below the search bar. A matched ticker can be selected. Upon clicking the dropdown selection, the search should start automatically, and execute identically as described in the previous paragraph.

3.1.3 Autocomplete

A Search Bar allows a user to enter a keyword (Stock ticker symbol) to retrieve information. Based on the user input, the text box should display a list of all the matching company's ticker symbols with the company's name (see **Figure 2**). The autocomplete JSON data is retrieved from the ***Finnhub Search API*** (refer to **Section 4.1.4**).

The autocomplete response is filtered using the criteria: type= 'Common Stock', Symbol doesn't contain '.'(dot)

These are examples of calling this API:

```
https://finnhub.io/api/v1/search?q=<COMPANY_NAME>&token=<API_TOKEN>
# or
```

```
https://finnhub.io/api/v1/search?q=<SYMBOL>&token=<API_TOKEN>

For example:
https://finnhub.io/api/v1/search?q=apple&token=<API_TOKEN>
```

The autocomplete function should be implemented using **Angular Material**. Refer to Section 5.3 for more details.

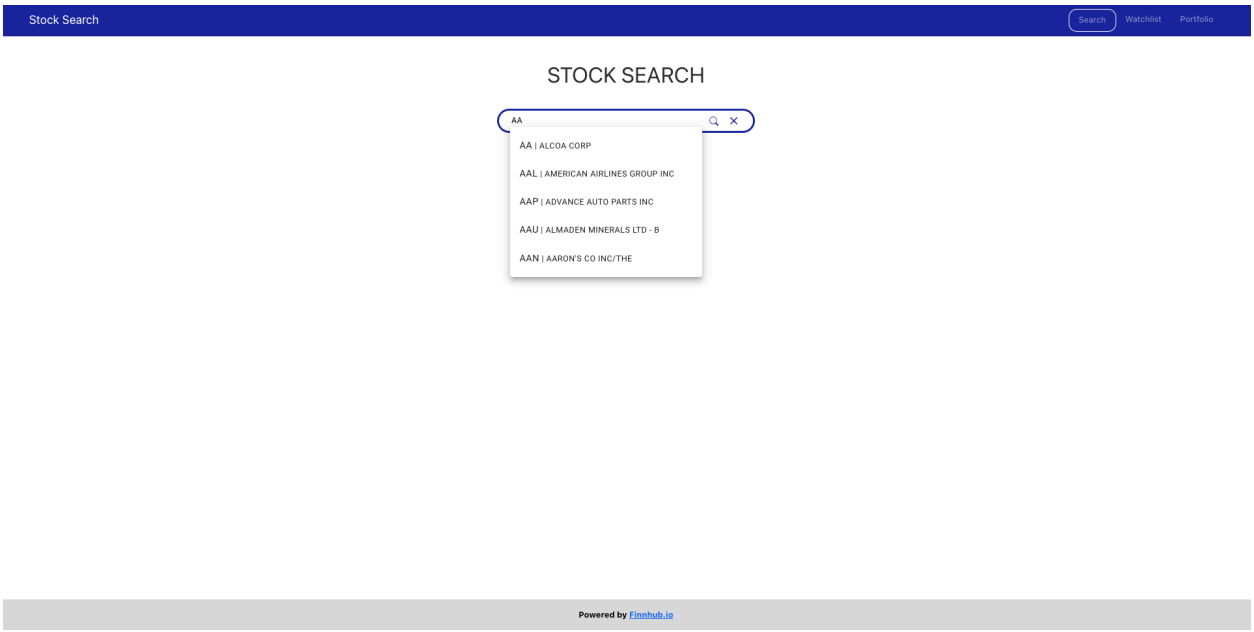


Figure 2: Autocomplete Suggestion

3.2 Search Results Page

3.2.1 Details of Searched Stock

After the user executes a search for a ticker, a page should route to /search/<ticker> path (example: /search/AMZN). The following components need to be displayed on successful search:

- Symbol, company name, trading Exchange (such as NASDAQ), and a Buy button on top left, The **Sell button should appear alongside the Buy button only when the portfolio has purchased stocks of a company. See Figure 3.3**;
- Last price, change, percent change, and date/time, on top right. The change items should be preceded by appropriately colored arrows;
- Company Logo and Indication of open / closed market in the top-center;
- Summary, Top News, Charts and Insights tabs.

IMPORTANT NOTE: If the user navigates to the watchlist route or the portfolio route, and navigates back to the search results page, the previously searched stock results should remain on

the search/<ticker> route. Also, results data should be retrieved from a state/service and not be fetched from a new search API call.

Please refer to **Figures 3.1**, **3.2, 3.3** below.

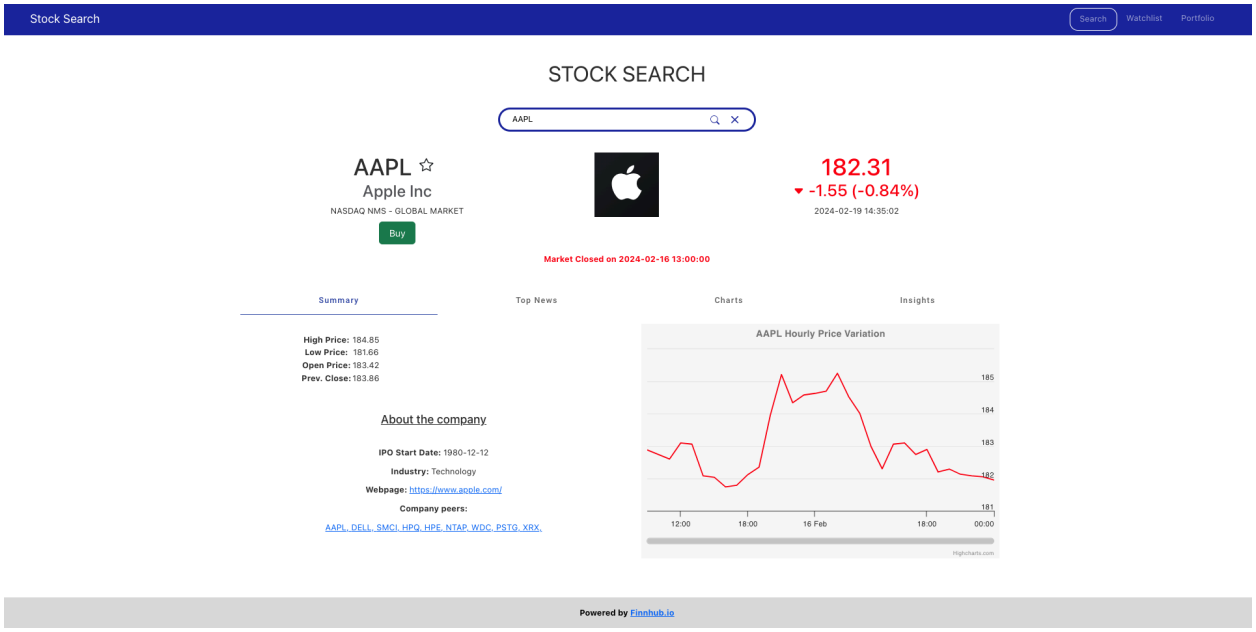


Figure 3.1: Search Details page overview (Market is Closed)

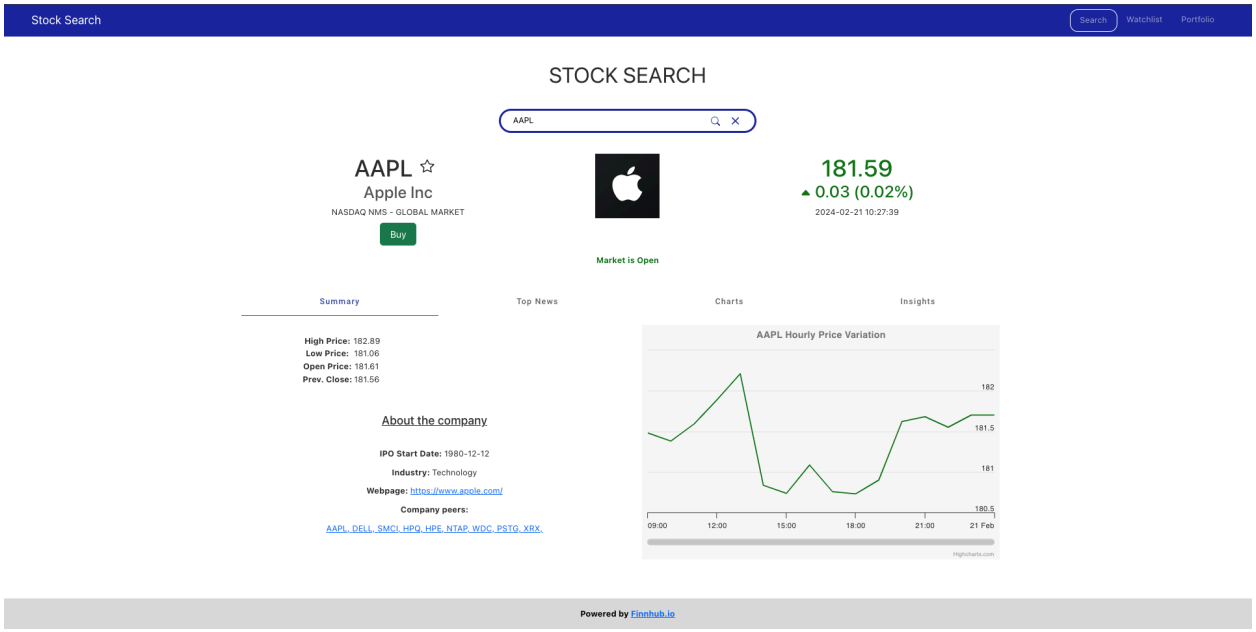


Figure 3.2: Search Details page overview (Market is Open)

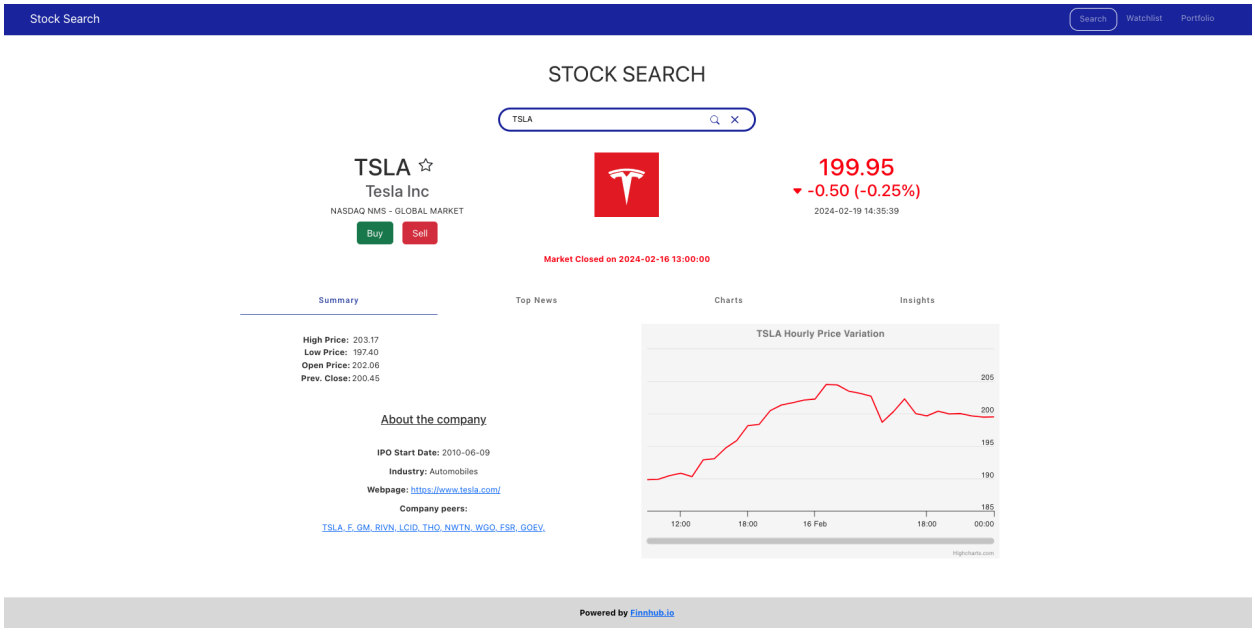


Figure 3.3: Search Details page overview (Dynamic Sell button when the portfolio has purchased stock of the company)

- When the user clicks on the star icon, the white star turns yellow, and that ticker should be stored in MongoDB Atlas. A self-closing alert should be displayed at the top and that stock should be visible on the Watchlist Page (see **Section 3.3**).
- When the user clicks on the **Buy** button, a modal window should open. which will display the details (stock symbol, current price, money in wallet, input for quantity of shares to buy and total price for the shares), as shown in Figure 3.5. Note that:
    - The **Buy** button should be disabled if the user inputs a quantity < 1 or the quantity field is empty or the quantity leading to total more than money in wallet (see **Figure 3.5**);
    - The **Buy** button will be enabled once the user enters a number greater than 0 and suitable quantity up to total equals (or less than) money in the wallet (see **Figure 3.4**).
- The **Sell** option is available on the homepage only when there is at least 1 quantity of stock available in the portfolio. Upon clicking the **Sell** button, similar behavior as the buy transaction should be implemented, along with the constraint being able to sell only the stocks owned.
- **Error messages** should be shown for an attempt to buy more than the possible quantity of stocks using the money in wallet/attempt to sell more than the number of stocks owned in Portfolio. See Figure 3.5 for reference.
- A self-closing transaction **alert message** should be shown for both buy and sell transactions as in **Figure 3.4a**.

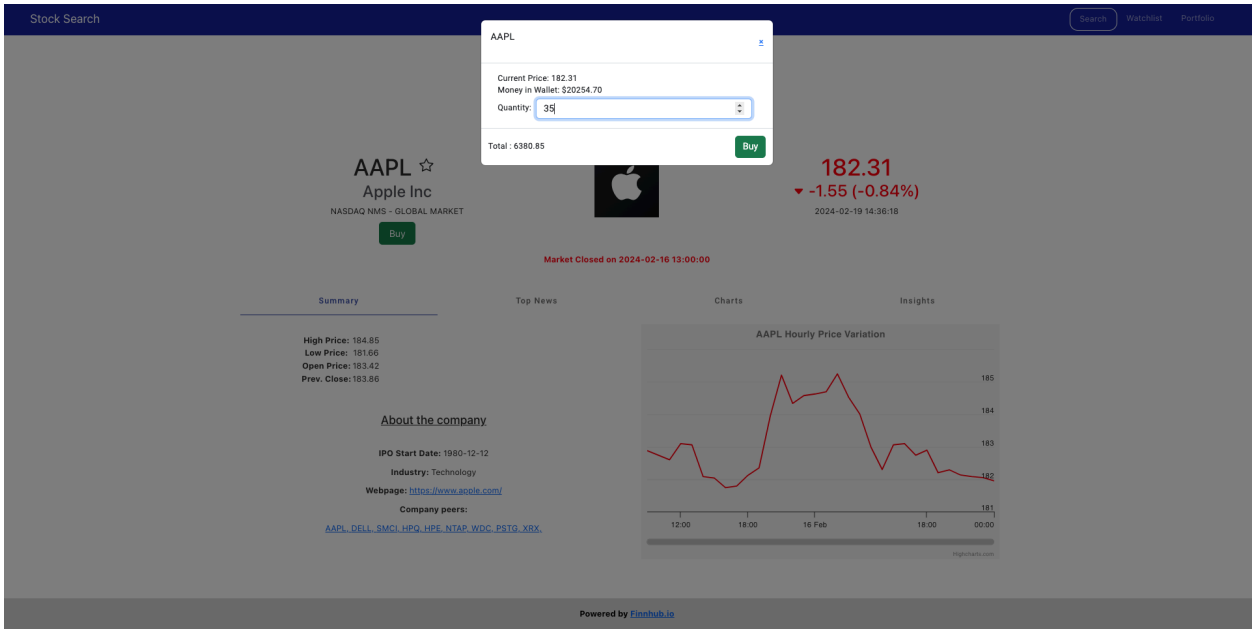


Figure 3.4: Buy Button enabled for valid input

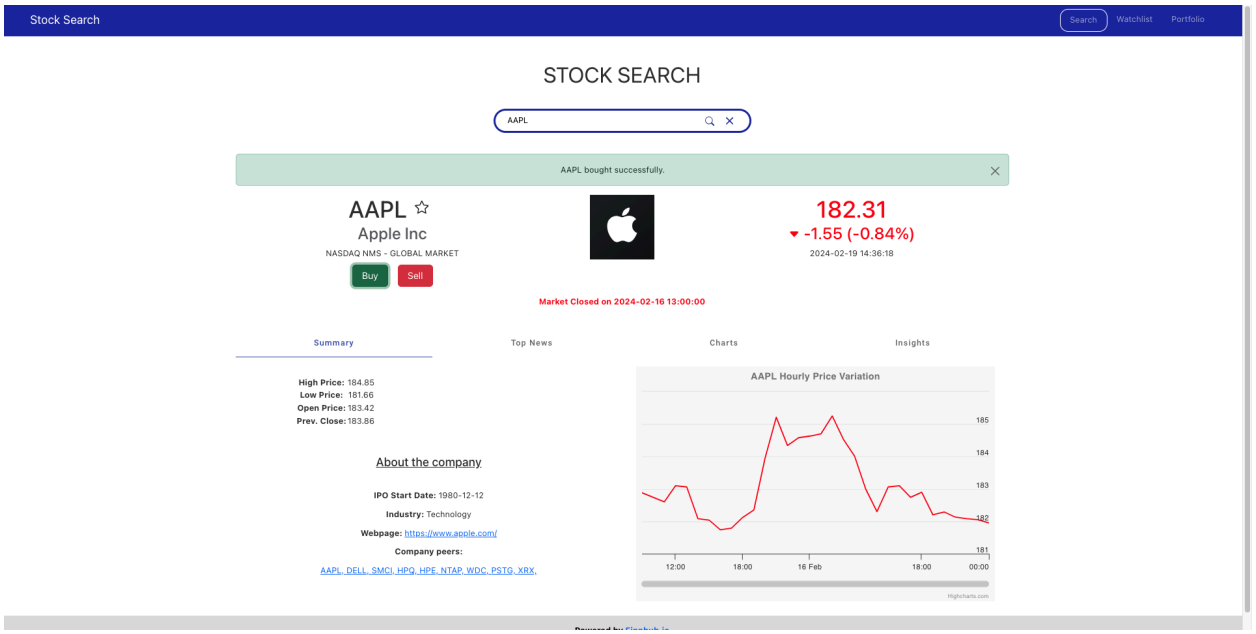


Figure 3.4a: Alert for buying stock successfully

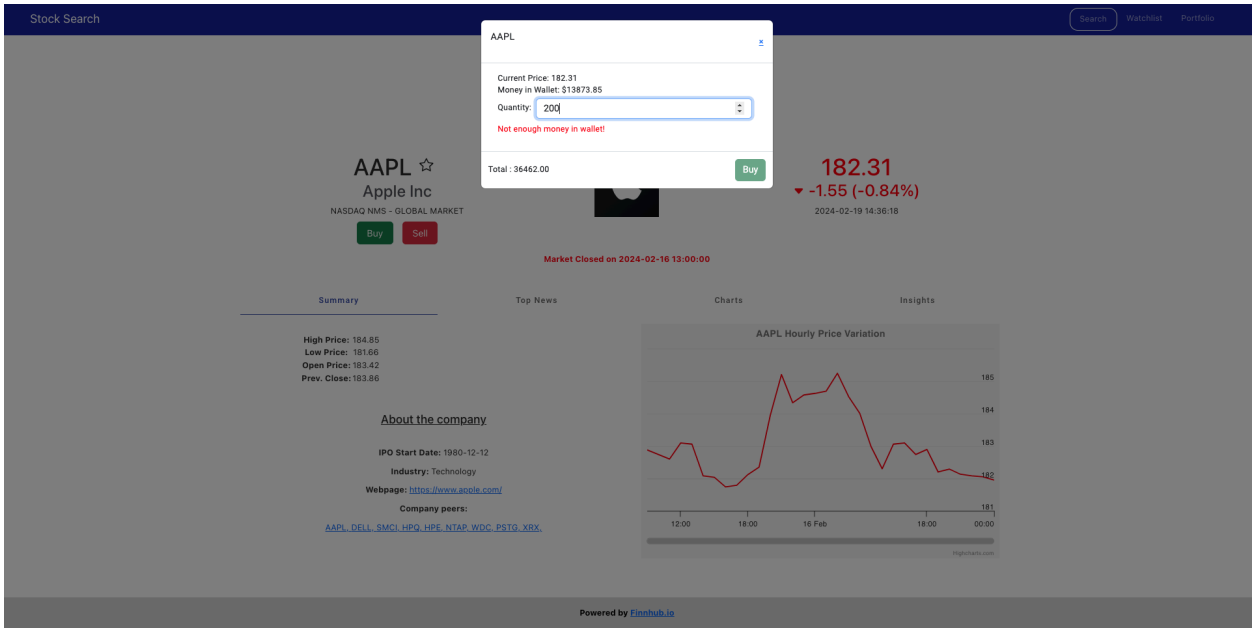


Figure 3.5: Buy button disabled for invalid input

For details and screenshots of similar sell transactions, refer to Section **3.4 (Portfolio).** As described, the page contains two major panels:

1) **Stock Details**: This panel displays all the values mentioned in **Table 3.1**. Last Timestamp should be only displayed beside the Market status if "Market is Close."

Fields	**Sample Values**	**API reference**
Ticker	VMW	From table 4.1, use 'ticker' key
Company's Name	VMware Inc.	From table 4.1, use 'name' key
Exchange Code	New York Stock Exchange, Inc	From table 4.1, use 'exchange' key
Last Price	126.99	From table 4.3, use 'c' key
Change	1.09	From table 4.3, use 'd' key
Change Percentage	0.86%	From table 4.3, use 'dp' key
Current Timestamp	2022-02-17 09:02:21	From table 4.3, use 't' key
Market Status	Open/Close	Calculation from timestamp

Table 3.1 – Stock Details

2) **Material Tabs** – This component helps users see different stock data on the same page, and the content of the tabs is detailed in the following sections.

IMPORTANT NOTE: All numerical values should be rounded off to 2 decimal places.

Data mentioned in the Stock Details section and Summary Tab, should auto-update every 15 seconds, when the stock market is open.

3.2.2 Summary Tab

This tab contains a summary of the stock, which includes:

- Details, as follows:
  - Calculate if the market is open, using the timestamp key in **Table 3.2.** The value of 't' is the last timestamp at which the stock details are available. Assume the market is closed if more than 5 minutes has elapsed from this 't' value. Assume the market is open, if otherwise.
  - If the market is open: display all the fields mentioned in **Table 3.2 and Table 3.3**, as shown in **Figure 3.2**.
  - If the market is closed: Display all the fields mentioned in **Table 3.2 and Table 3.3**, as shown in **Figure 3.1**.
- About the Company: values from **Table 3.3.**
- Chart for the last Working Day:
  - See **Section 4.1.2** to obtain hourly stock price data using resolution '5' and show variation for the last 6 hours.
  - If the market is open: show stock price variation from current time.
  - If the market is closed: show stock price variation from when the market was closed. (i.e., last working day).

Fields	Sample Values	API reference
High Price	128	From table 4.3, use 'h' key
Low Price	124.90	From table 4.3, use 'l' key
Open Price	125	From table 4.3, use 'o' key
Prev. Close	125.90	From table 4.3, use 'pc' key
Timestamp	1645218002	From table 4.3, use 't' key

Table 3.2: Fields used inside Summary Tab (Part 1)

Fields	Sample Values	API reference
IPO Start Date	2007-08-14	From table 4.1, use 'ipo' key
Industry	Technology	From table 4.1, use 'finnhubIndustry' key
Webpage	https://www.vmware.com/	From table 4.1, use 'weburl' key
Company Peers	MSFT, ORCL, NOW, VMW	From table 4.8, use response list

Table 3.3: Fields used inside Summary Tab (Part 2)

Important Note: The list of company peers should be clickable links which should navigate to the search results of that ticker.

3.2.3 Top News Tab

This tab shows top news for the given stock ticker symbol (see **Figure 3.6** and **Figure 3.7**). In particular:

- Show cards which contain Image and Title.
  - For Image use 'image' key from **Table 4.5**.
  - For Title use 'title' key from **Table 4.5**.
- When clicking on a card, open a Modal window as shown in **Figure 3.7**. For details regarding Modal check **Section 5.3**. Modal contains all the fields mentioned in **Table 3.4**.
- Users can share the news articles on Twitter and Facebook. For details on how to use it, check **Section 4.2**. Twitter and Facebook should open in a new browser tab, if clicked.
  - In Twitter, it should create a post having following content:
    - Title of the news article
    - URL of the news article.
  - In Facebook, it should create a post, which contains the URL of the news Article.
- Inside modal, when the user clicks on '***here***' in '*For more details click* ***here***', it should open the URL for the article in a new browser tab.

Fields	**API reference**
Source	From Table 4.5, use 'source' key.
Published Date	From Table 4.5, use 'datetime' key.
Title	From Table 4.5, use 'headline' key.
Description	From Table 4.5, use 'summary' key.
URL	From Table 4.5, use 'url' key.

Table 3.4: Fields used inside modal of Top News Tab

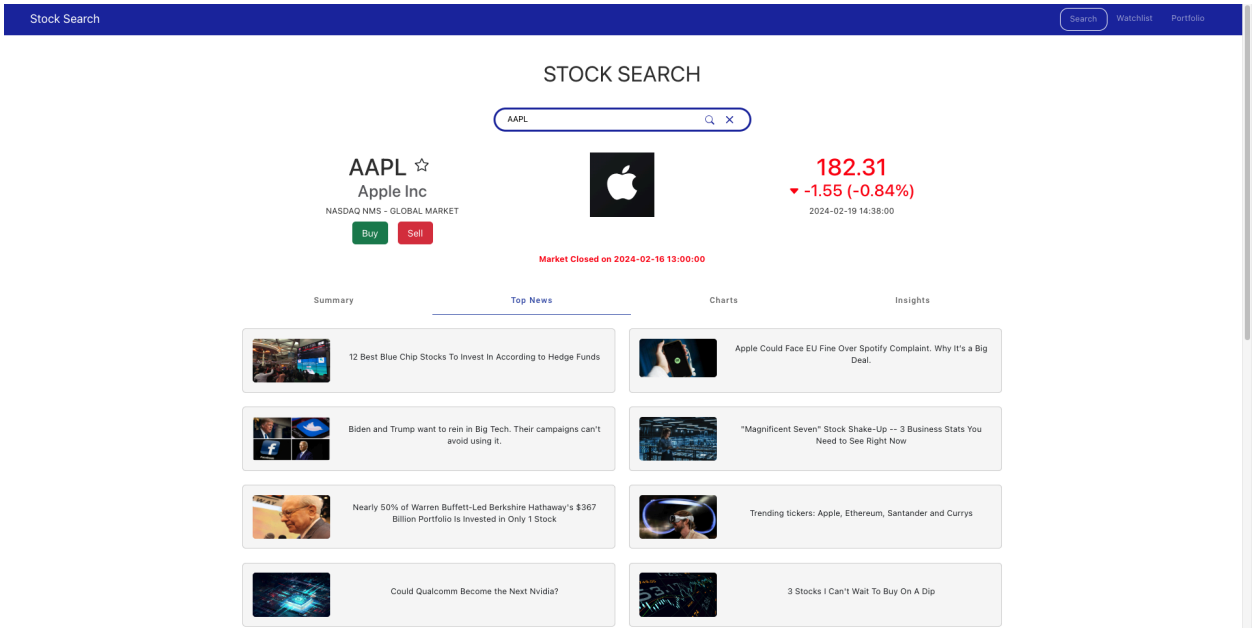


Figure 3.6: Top News Tab overview

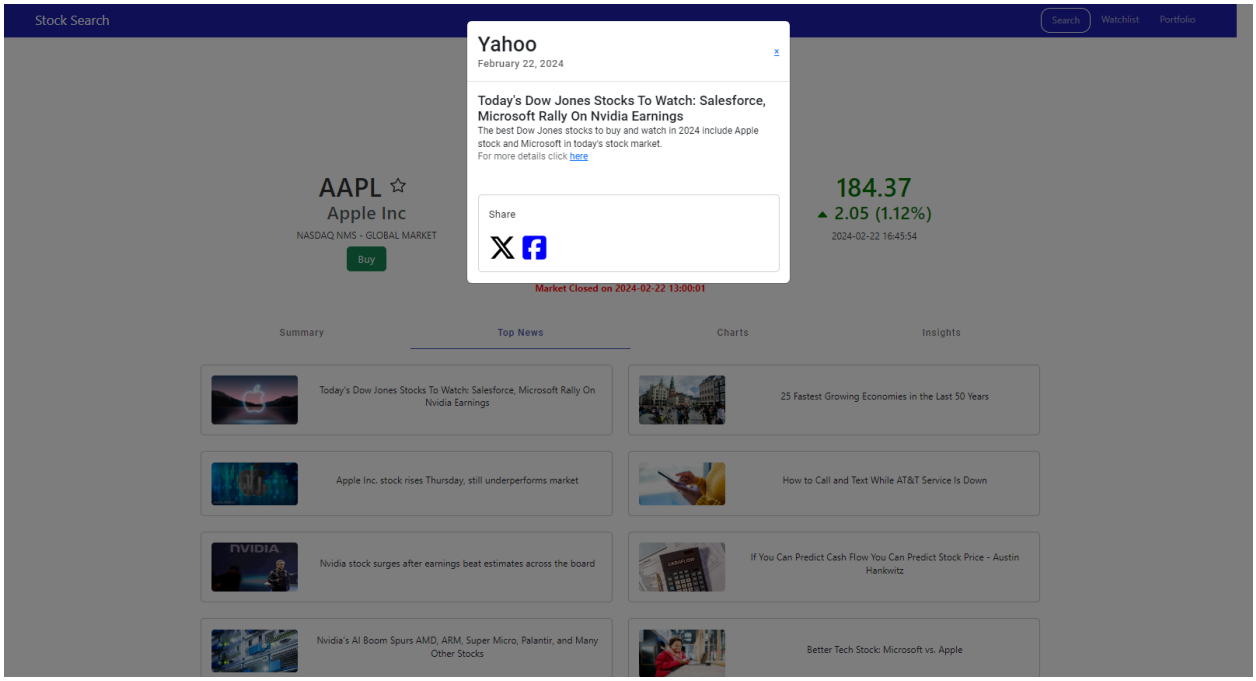


Figure 3.7: Top News Detailed Modal overview.

3.2.4 Charts Tab

This tab uses HighCharts to display historical stock market data on the related stock. In particular:

- See **Figure 3.8** for reference using **Section 4.1.3** implementation.
- For more details regarding *Highcharts* see **Section 5.5**.
- Display SMA and Volume by Price charts for data of the last 2 years.

Figure 3.8: Charts tab overview.

3.2.5 Insights Tab

This tab uses a table that contains fields as mentioned in **Table 3.5** and HighCharts to display recommendation trends and company earnings data on the related stock. In particular:

- See **Figure 3.9** for reference using **Section 4.1.7 and Section 4.1.9**.
- Aggregate the response data from **Section 4.1.7** for all MSPR data and display it in the table.
- For more details regarding Highcharts see **Section 5.5**.

Fields	**API reference**
Total - MSPR	From Table 4.7, aggregate the values of the ‘mspr’ key.
Positive – MSPR	From Table 4.7, aggregate all positive values of the ‘mspr’ key.
Negative – MSPR	From Table 4.7, From Table 4.7, aggregate all negative values of the ‘mspr’ key.
Total – Change	From Table 4.7, aggregate the values of the ‘change’ key.
Positive – Change	From Table 4.7, aggregate all positive values of the ‘change’ key.
Negative – Change	From Table 4.7, From Table 4.7, aggregate all negative values of the ‘change’ key.

Table 3.5: Fields used inside table of Insights Tab

IMPORTANT NOTE: 'Total' calculation to be made appropriately for all fields from the array of response obtained (see **Section 4.1.7**).

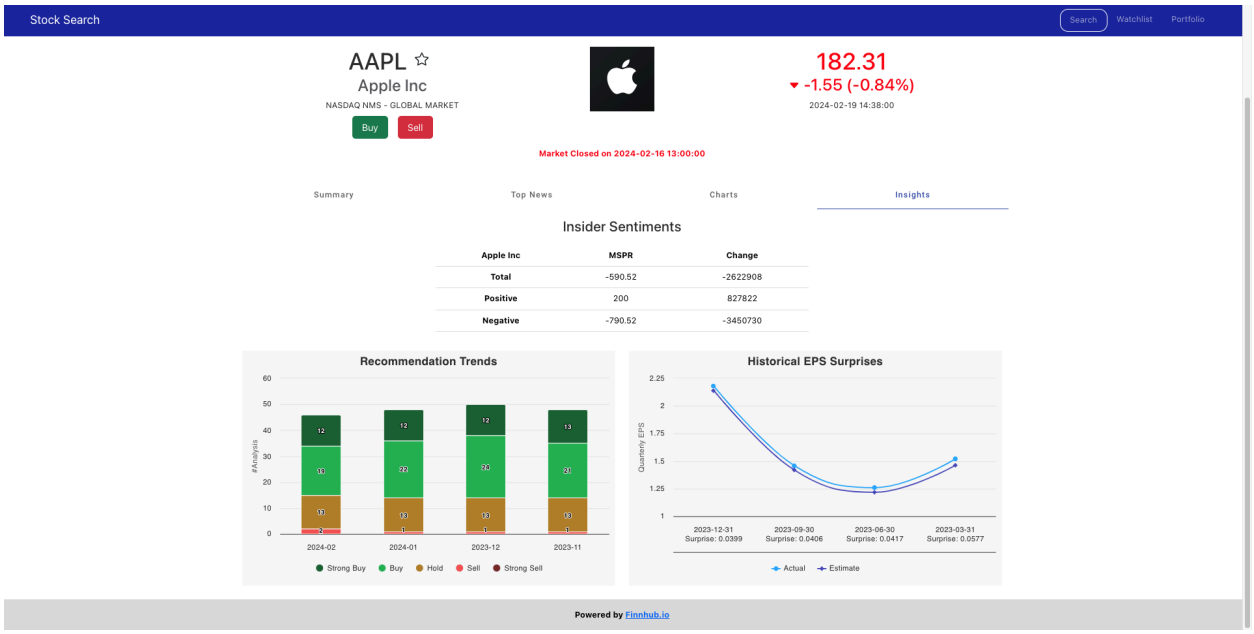


Figure 3.9: Insights tab overview.

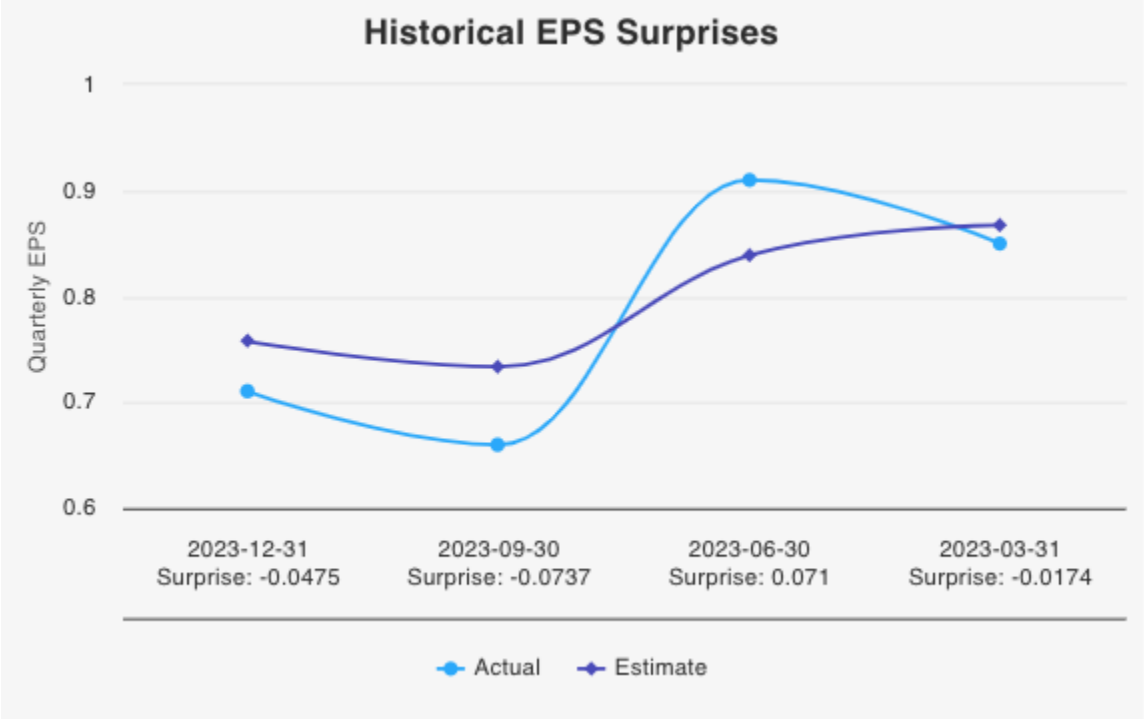


Figure 3.9a: Company Earning chart with non-null values

IMPORTANT NOTE: Display an error message, as shown below, if the user did not enter any input in the search box or no data is found for the entered input. Use the company profile response to determine no data.

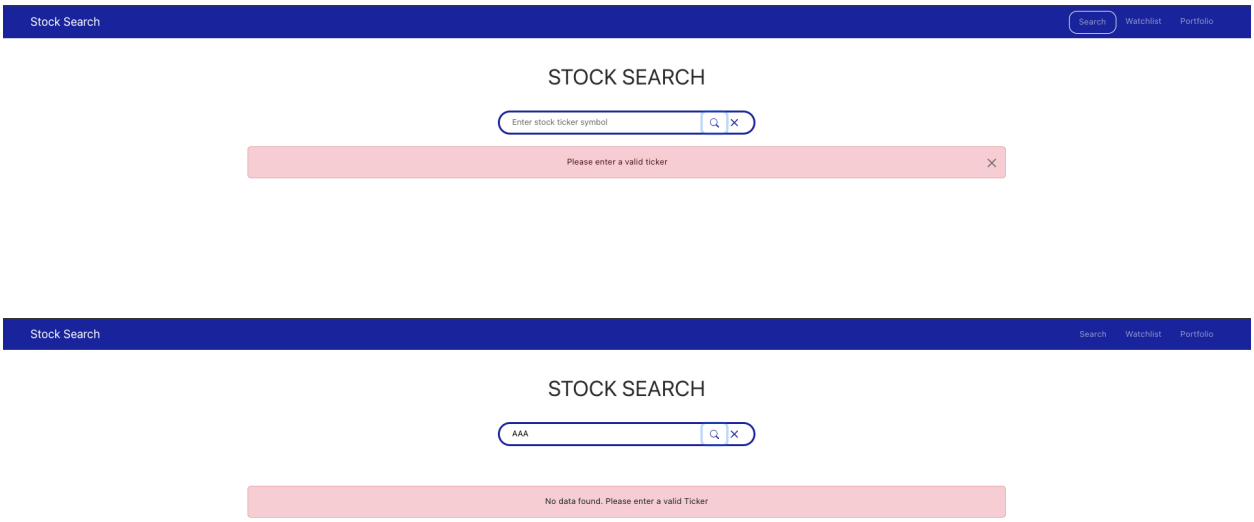


Figure 3.10: Error Alert.

3.3 Watchlist Menu

This menu will display all the stocks that are added to the watchlist by the user. This watchlist will be maintained in *MongoDB Atlas*. For more details on MongoDB Atlas, see **section 5.4** and **Figure 3.11**.

- If the change is positive, the color of the 'c', 'd' and 'dp' keys should be green
- If the change is negative, the color of the 'c', 'd' and 'dp' keys should be red
- If there is no change, the color of the 'c', 'd' and 'dp' keys should be black.
- When clicking on the close button on the right-top corner of the card (the "x"), it should remove the stock from the watchlist and display an updated watchlist.
- When clicking on the card, it should open the details route of that ticker (stock).
- If the watchlist is empty, it should display the alert as shown in **Figure 3.12**.

'c', 'd' and 'dp' key should be used from **Table 4.3**.

NOTE: 'percentage change' should be rounded off to 2 decimal places.

Upon clicking any area of the watchlist card, the user should be navigated to the search details page of that stock. Initially a loading "spinner" is displayed until the watchlist is pulled from the database.

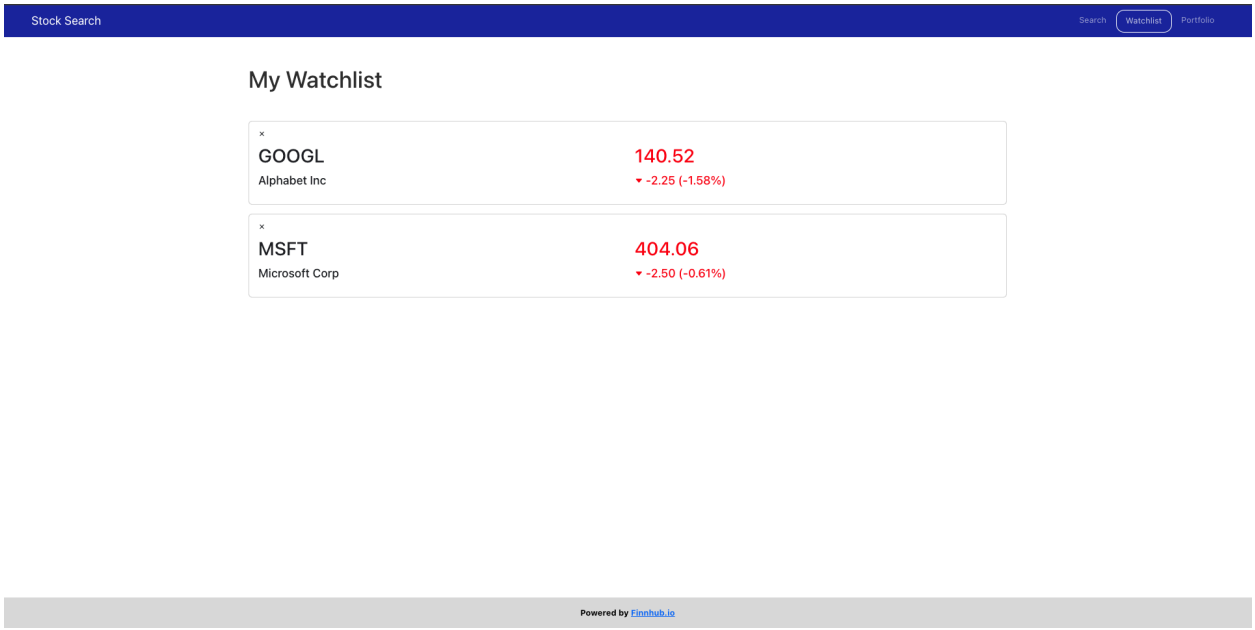


Figure 3.11: Watchlist menu page

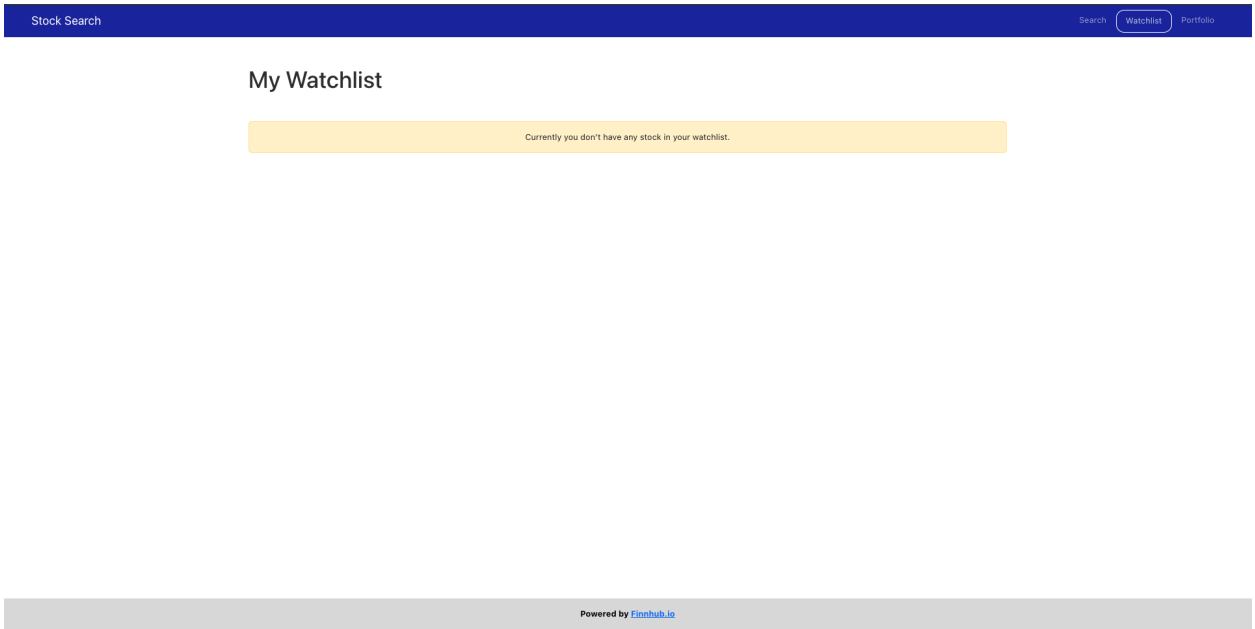


Figure 3.12: Watchlist Empty Alert

3.4 Portfolio Menu

This menu will display all the stocks that have been bought by the user (i.e., the current portfolio of the user). This portfolio will be maintained in the *MongoDB Atlas*. For more details on MongoDB Atlas, see **Section 5.4** and **Figure 3.13**.

To simulate real-world stock trading and allow the user to incur profits/losses, a wallet feature should be implemented. Initialize the cash balance in the wallet to be $25,000.00 using MongoDB Atlas. Users will use this cash balance to trade stocks.

Update the money spent and gained during buy and sell transactions, accordingly. The initial price of stock when buying it, and the current change in price should be used to update the balance in the wallet for profit and loss. Initially a loading spinner is displayed until the portfolio is pulled from the database.

In particular:

- If the current rate is greater than the rate at which user bought it, then color of the 'Change', 'Current Price' and 'Current Total' keys should be green;
- If the change is negative, the color of the 'Change', 'Current Price' and 'Current Total' keys should be red;
- If there is no change, the color of the 'Change', 'Current Price' and 'Current Total' keys should be black;
- When clicking on the Buy button, a modal should open as shown in **Figure 3.15**. The Buy button inside the modal should be disabled if the quantity entered by the user is not valid, as shown in **Figure 3.14**. Valid input should be (a) greater than 0, with (b) quantity that produces a total less than available cash in the wallet and (c) must be non-empty;
- When clicking on the Sell button, a modal should open as shown in **Figure 3.17 and 3.18**. The Sell button inside the modal should be disabled if the quantity entered by the user is not valid. Input is Valid if, 0<input<=Quantity and must be non-empty. Quantity is described in **Table 3.6**;
- When clicked on card's header part, it should open the search details route of that ticker (stock);
- If the portfolio is empty, it should display the alert as shown in **Figure 3.14**.

'c' key should be used from Table 4.3 for 'Current Price'. 'Current Change' and 'Current Total' should be calculated as shown in **Table 3.5**.

Quantity	Total Number of stocks bought by the user. It Should be more than 0, otherwise remove it from the portfolio in MongoDB Atlas.
Total Cost	Total cost is the sum of the total cost paid for all the purchases of the stock. For Example, if user has bought 10 stocks of AAPL in past, at the rate of 200, and today if user buys another 10 stocks of AAPL at the current price, i.e. 300, then the Total Cost for the user will be (10*200) + (10*300) = 5000. So Quantity is 20 and Total Amount is 5000.
Average Cost per Share	(Total Cost / Quantity)

Current Price	‘c’ key from the table 4.3
Change	(Average Cost per Share – Current Price) of the stock. Here, Current Price is ‘c’ key from the table 4.3
Market Value	(Current Price * Qty), here Current Price is ‘c’ key from **Table 4.3** and Qty is the number of stocks present in the user's portfolio.

Table 3.6: Fields used in Portfolio Cards.

Alerts should be displayed for successful buy and sell transactions, which should auto-close.

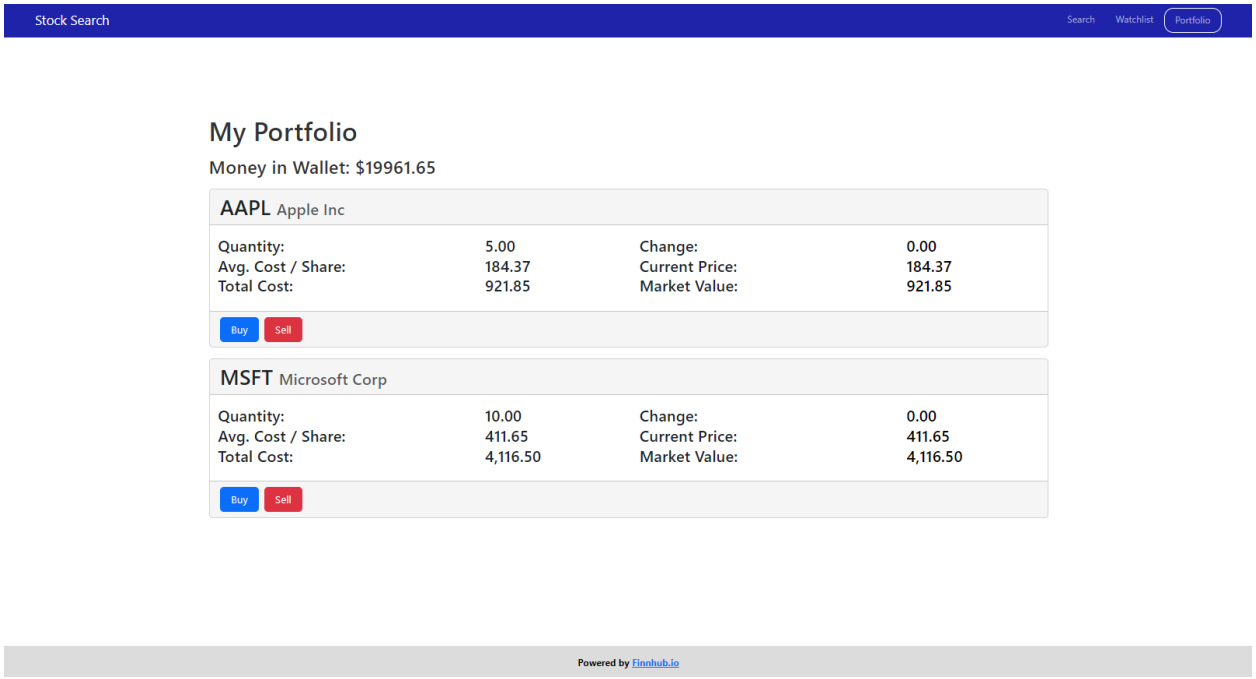


Figure 3.13: Portfolio

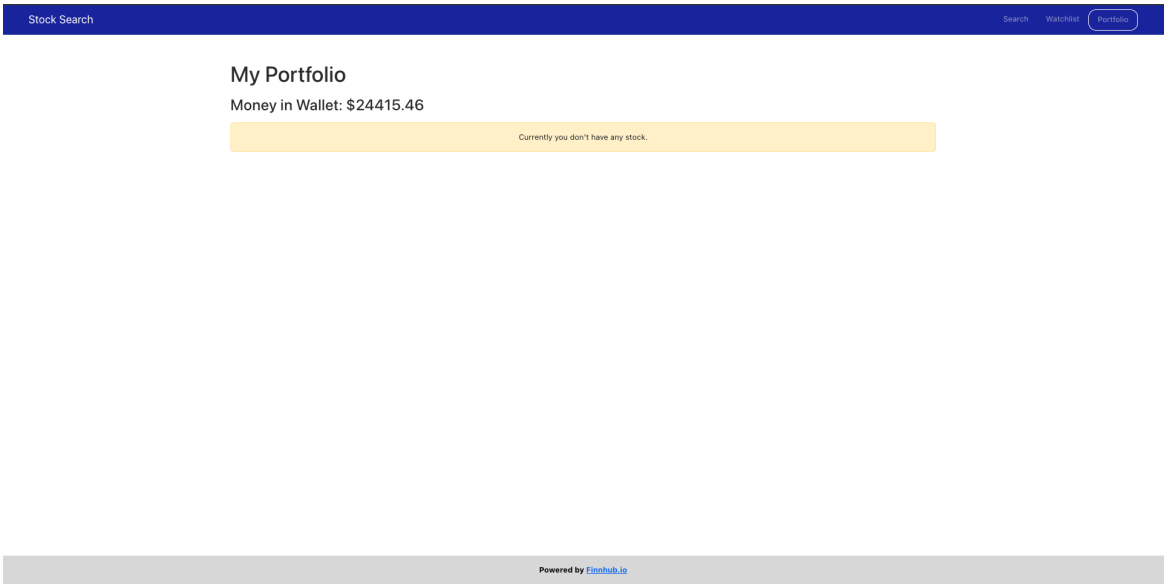


Figure 3.14: Portfolio Empty Alert

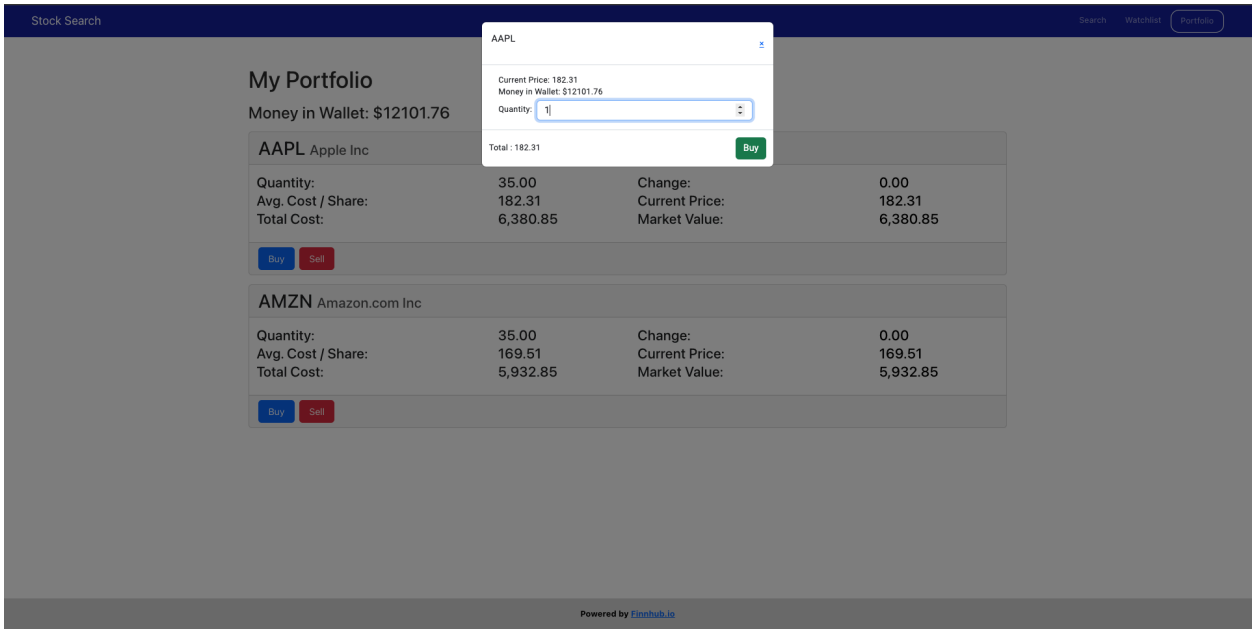


Figure 3.15: Modal for Buying Stock

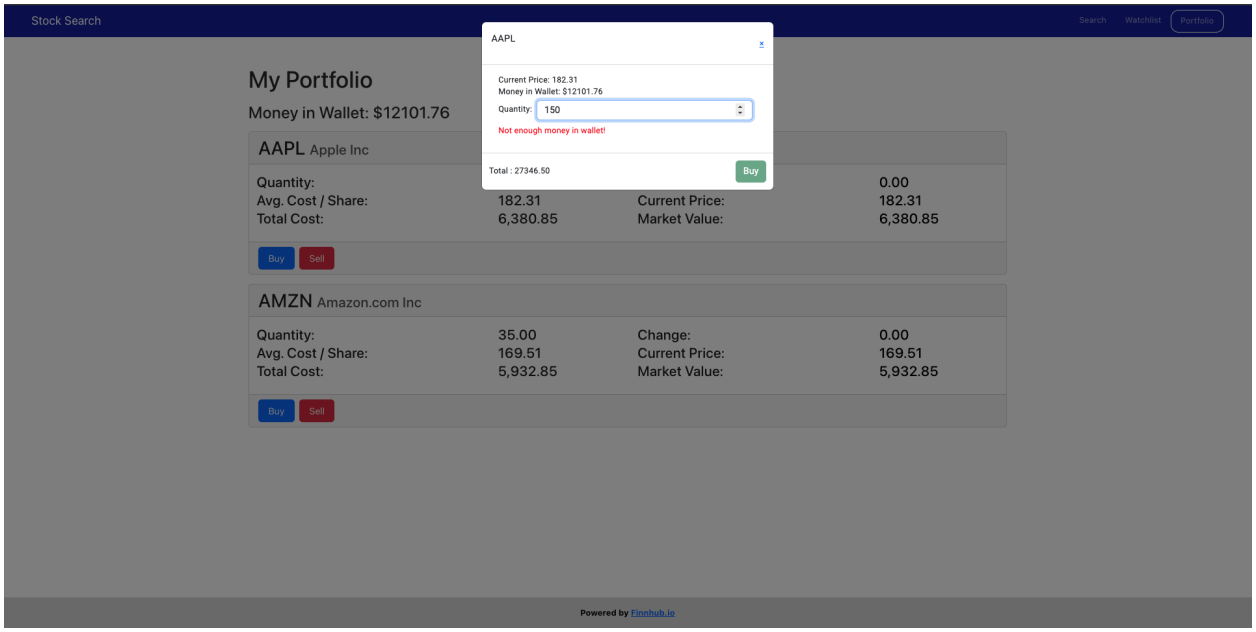


Figure 3.16: Input is invalid in Modal for Buying Stock

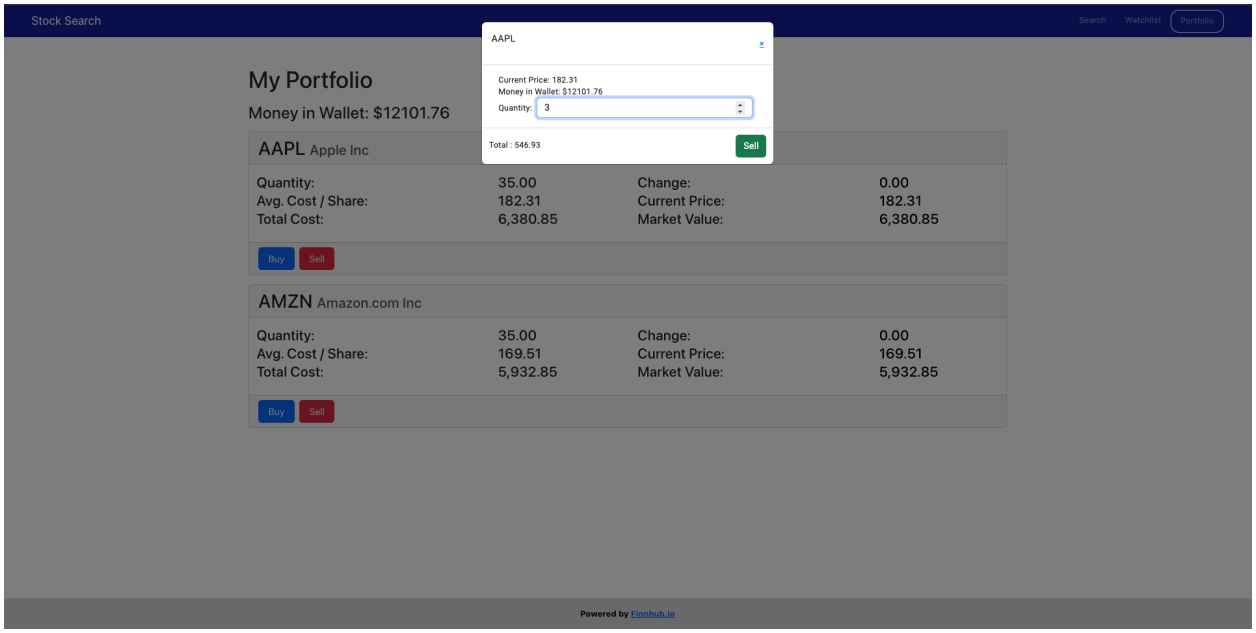


Figure 3.17: Modal for Selling Stock

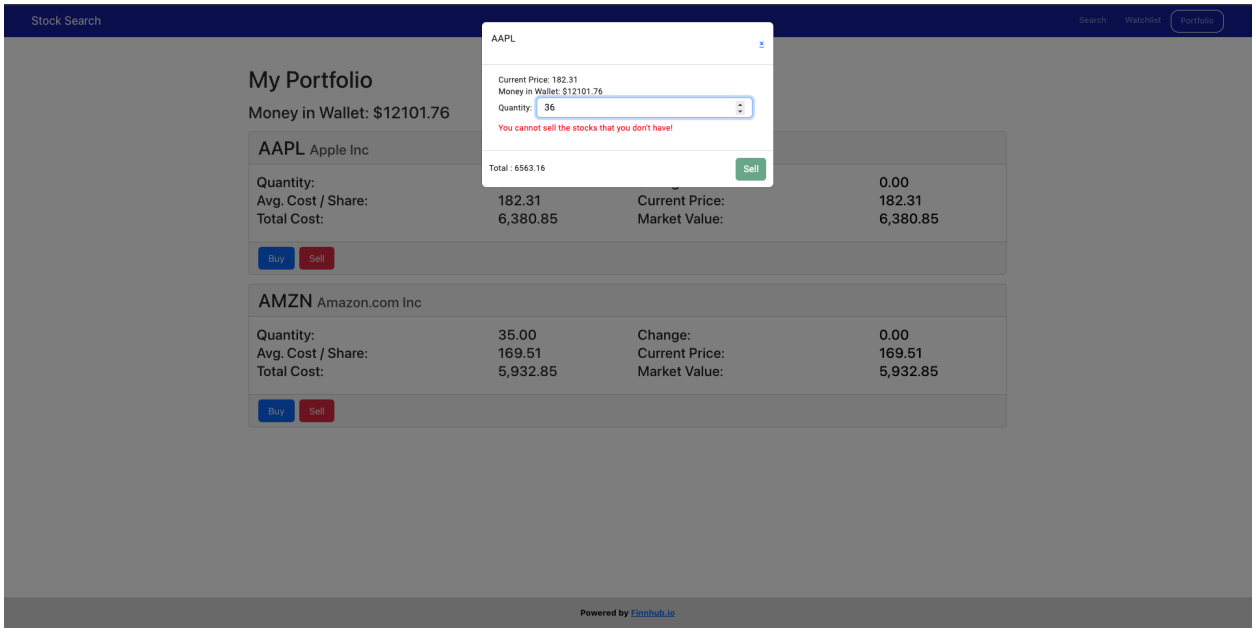


Figure 3.18: Input is invalid in Modal for Selling Stock

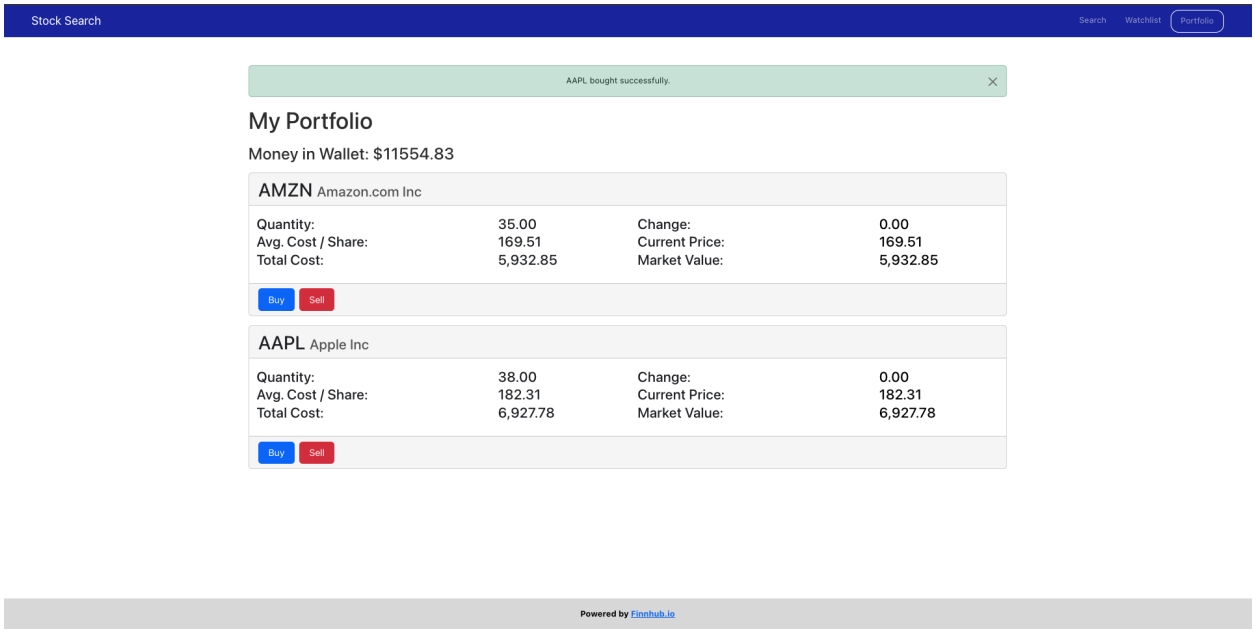


Figure 3.19: Auto closing alert message for buying more stocks from portfolio section

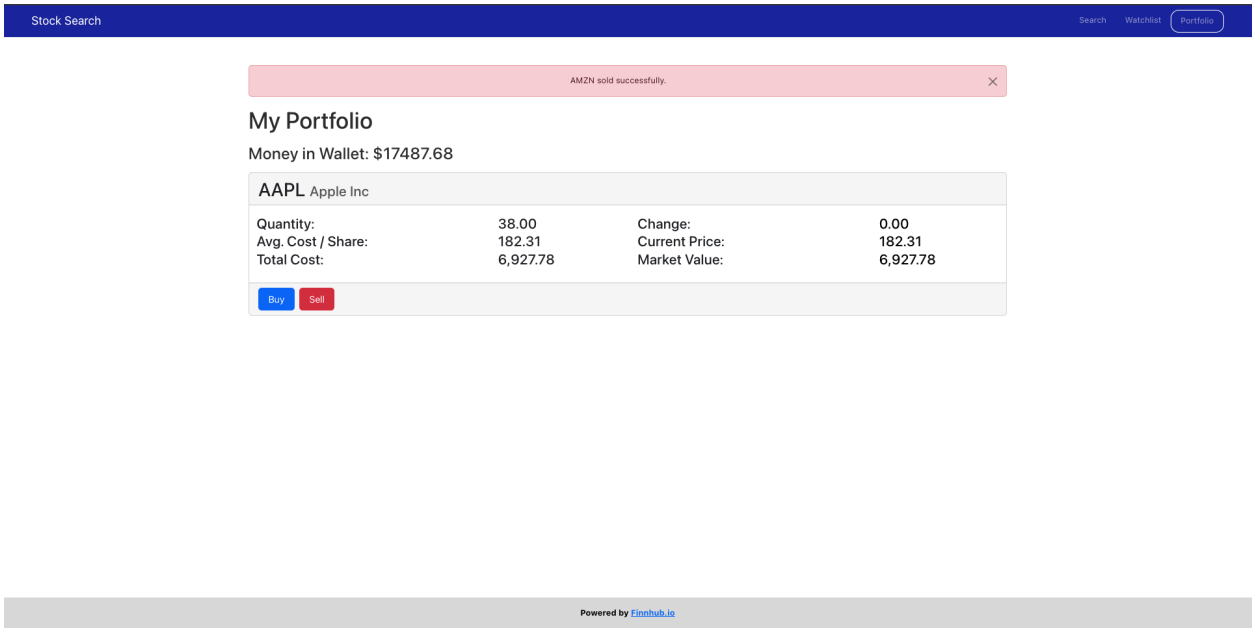
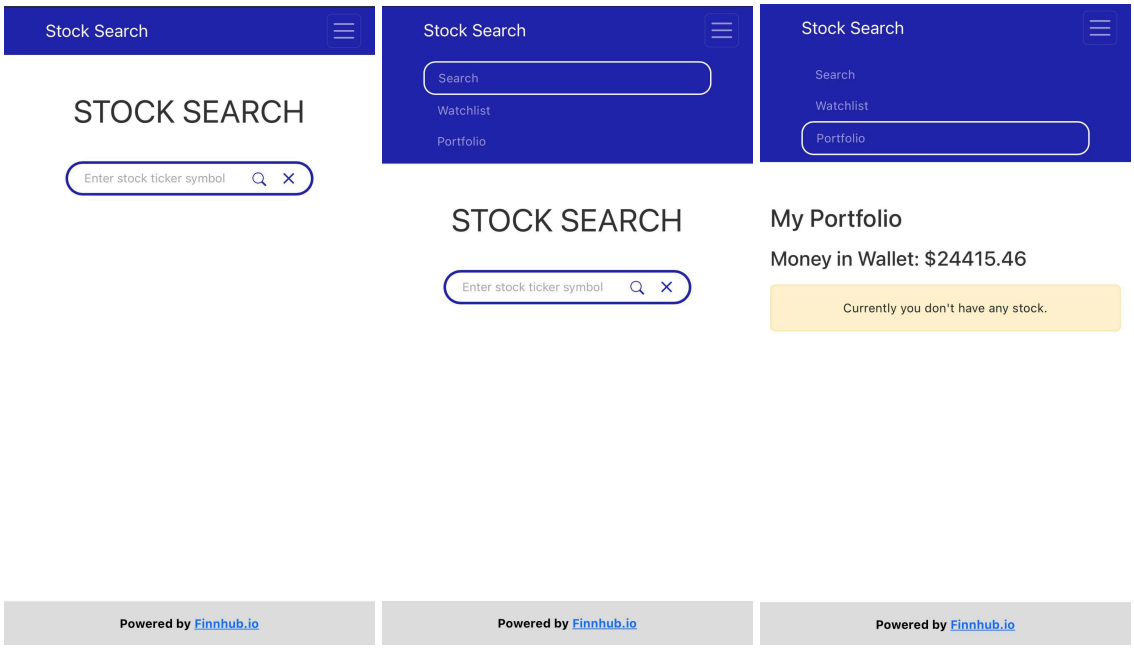
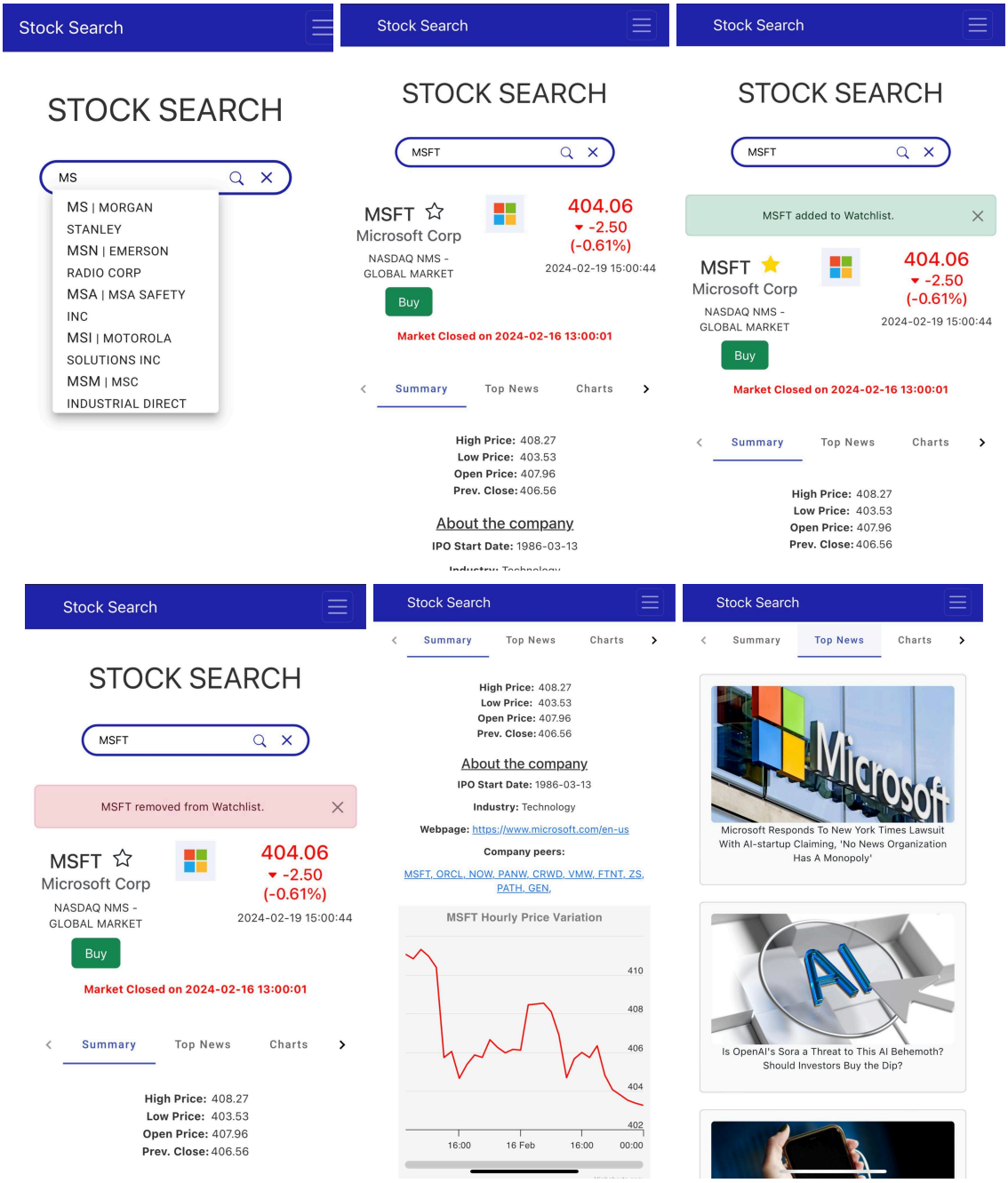


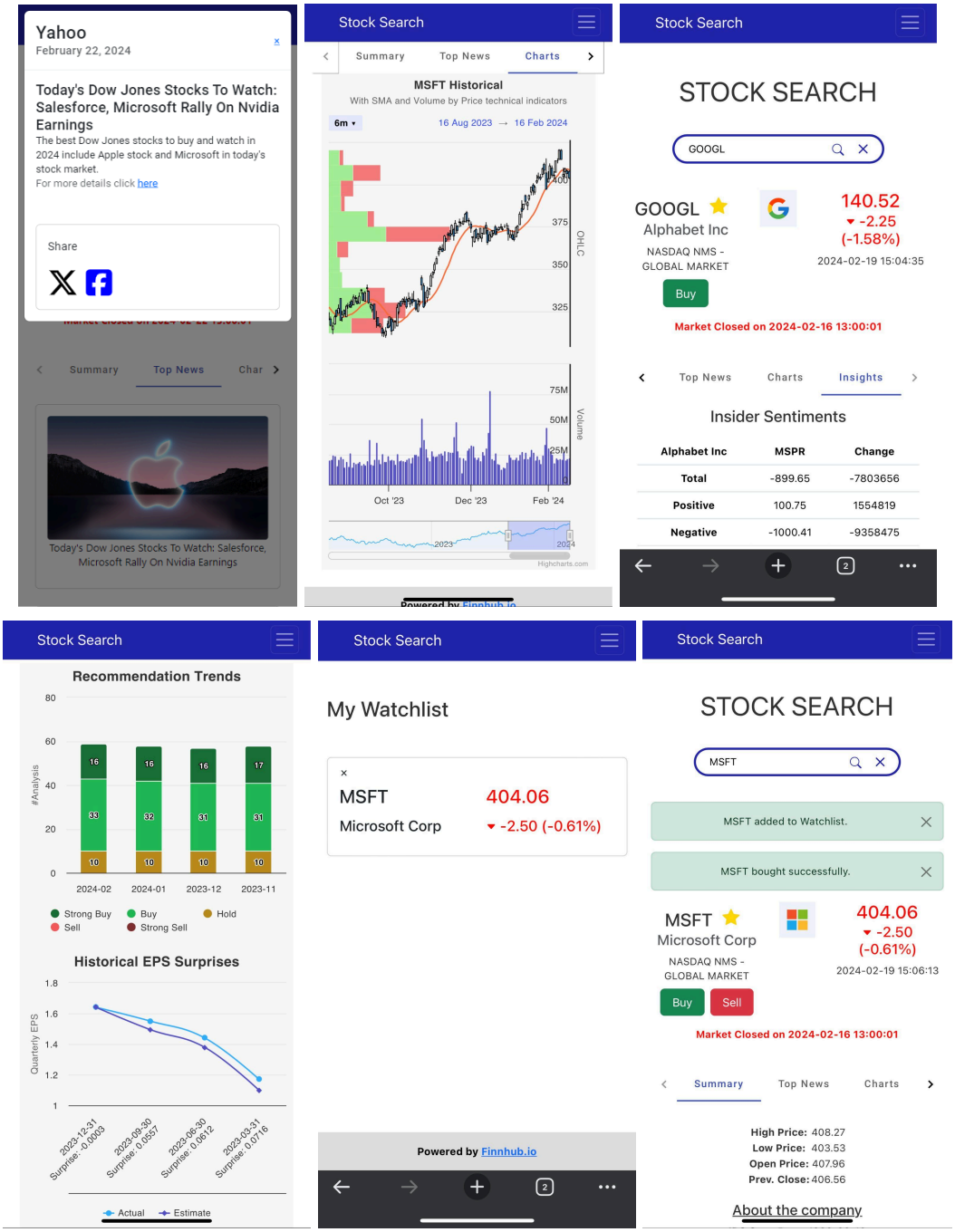
Figure 3.20: Auto closing alert message for selling stocks from portfolio section

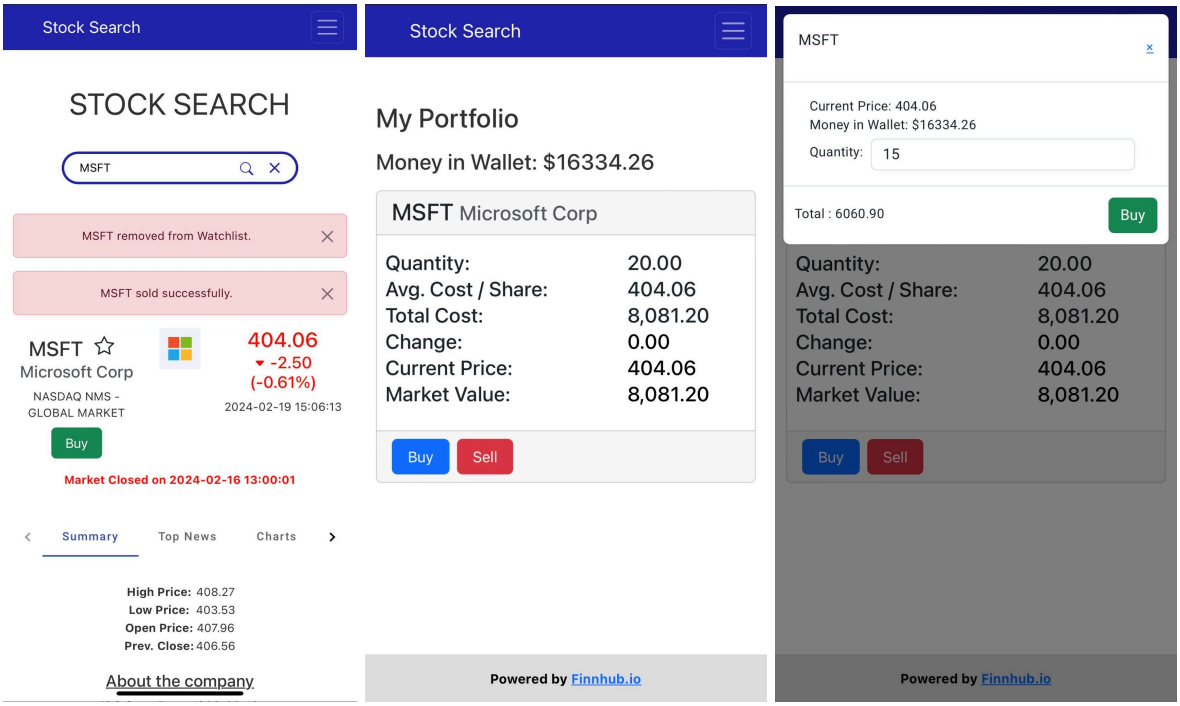
3.5 Responsive Design

The following are a few snapshots of the web app opened with Google Chrome is simulating a mobile device.




Stock Search
STOCK SEARCH
MS
MS | MORGAN STANLEY
MSN | EMERSON RADIO CORP
MSA | MSA SAFETY INC
MSI | MOTOROLA SOLUTIONS INC
MSM | MSC INDUSTRIAL DIRECT
Stock Search
STOCK SEARCH
MSFT
MSFT
Microsoft Corp
NASDAQ NMS - GLOBAL MARKET
Buy
404.06
-2.50
(-0.61%)
2024-02-19 15:00:44
Market Closed on 2024-02-16 13:00:01
Summary
Top News
Charts
High Price: 408.27
Low Price: 403.53
Open Price: 407.96
Prev. Close: 406.56
About the company
IPO Start Date: 1986-03-13
Stock Search
STOCK SEARCH
MSFT
MSFT added to Watchlist.
MSFT
Microsoft Corp
NASDAQ NMS - GLOBAL MARKET
Buy
404.06
-2.50
(-0.61%)
2024-02-19 15:00:44
Market Closed on 2024-02-16 13:00:01
Summary
Top News
Charts
High Price: 408.27
Low Price: 403.53
Open Price: 407.96
Prev. Close: 406.56
Stock Search
STOCK SEARCH
MSFT
MSFT removed from Watchlist.
MSFT
Microsoft Corp
NASDAQ NMS - GLOBAL MARKET
Buy
404.06
-2.50
(-0.61%)
2024-02-19 15:00:44
Market Closed on 2024-02-16 13:00:01
Summary
Top News
Charts
High Price: 408.27
Low Price: 403.53
Open Price: 407.96
Prev. Close: 406.56
Stock Search
Summary
Top News
Charts
High Price: 408.27
Low Price: 403.53
Open Price: 407.96
Prev. Close: 406.56
About the company
IPO Start Date: 1986-03-13
Industry: Technology
Webpage: https://www.microsoft.com/en-us
Company peers:
MSFT, ORCL, NOW, PANW, CRWD, VMW, FTNT, ZS, PATH, GEN,
MSFT Hourly Price Variation
410
408
406
404
402
16:00
16 Feb
16:00
00:00
Stock Search
Summary
Top News
Charts
Microsoft
Microsoft Responds To New York Times Lawsuit With AI-startup Claiming, 'No News Organization Has A Monopoly'
AI
Is OpenAI's Sora a Threat to This AI Behemoth? Should Investors Buy the Dip?


Yahoo
February 22, 2024
Today's Dow Jones Stocks To Watch: Salesforce, Microsoft Rally On Nvidia Earnings
The best Dow Jones stocks to buy and watch in 2024 include Apple stock and Microsoft in today's stock market.
For more details click here
Share
Summary
Top News
Today's Dow Jones Stocks To Watch: Salesforce, Microsoft Rally On Nvidia Earnings
Stock Search
Summary
Top News
Charts
MSFT Historical
With SMA and Volume by Price technical indicators
6m
16 Aug 2023 → 16 Feb 2024
OHLC
Volume
Oct '23
Dec '23
Feb '24
Highcharts.com
Powered by Finnhub.io
Stock Search
STOCK SEARCH
GOOGL
GOOGL
Alphabet Inc
NASDAQ NMS - GLOBAL MARKET
140.52
-2.25
(-1.58%)
2024-02-19 15:04:35
Buy
Market Closed on 2024-02-16 13:00:01
Top News
Charts
Insights
Insider Sentiments
Alphabet Inc | MSPR | Change
Total | -899.65 | -7803656
Positive | 100.75 | 1554819
Negative | -1000.41 | -9358475
Stock Search
Recommendation Trends
#Analysis
2024-02
2024-01
2023-12
2023-11
Strong Buy
Buy
Hold
Sell
Strong Sell
Historical EPS Surprises
Quarterly EPS
2023-12-31 Surprise: -0.0003
2023-09-30 Surprise: 0.0557
2023-06-30 Surprise: 0.0612
2023-03-31 Surprise: 0.0716
Actual
Estimate
Stock Search
My Watchlist
MSFT
Microsoft Corp
404.06
-2.50 (-0.61%)
Powered by Finnhub.io
Stock Search
STOCK SEARCH
MSFT
MSFT added to Watchlist.
MSFT bought successfully.
MSFT
Microsoft Corp
NASDAQ NMS - GLOBAL MARKET
404.06
-2.50
(-0.61%)
2024-02-19 15:06:13
Buy
Sell
Market Closed on 2024-02-16 13:00:01
Summary
Top News
Charts
High Price: 408.27
Low Price: 403.53
Open Price: 407.96
Prev. Close: 406.56
About the company



You must watch the video carefully to see how the page looks on mobile devices. All functions must work on mobile devices.

One easy way to test for mobile devices is to use Google Chrome Responsive Design Mode and Safari Develop – User Agent menu. An iPhone 12 Pro is used in the video.

3.5 Navbar

The Navigation bar must be present on top of the page, and visible at all times, as shown in all the figures above. You can use Bootstrap to create a navbar. It consists of following menu options:

1. Search
2. Watchlist
3. Portfolio

3.6 Footer

The Footer must be present at the end of each page, as shown in above figures. It should contain following line:

"Powered by Finnhub.io"

4. API's description

4.1 Finnhub API calls, similar to Assignment 2

In this assignment, we will use the Finnhub API. A comprehensive reference about this API is available at:

https://finnhub.io/docs/api/introduction

4.1.1 Company's Description

Reference: https://finnhub.io/docs/api/company-profile2

For **Company's Description**, use the following API. For more details refer **Figure 4.1**:

https://finnhub.io/api/v1/stock/profile2?symbol=<TICKER>&token=<API_Key >

URL parameter in API Call:

- Ticker: Ticker symbol of the stock. E.g.: MSFT
- Token: The API access Token. It is private, please do not share with anyone. See Assignment 2.

An example URL constructed from the parameters will look like this:

https://finnhub.io/api/v1/stock/profile2?symbol=AAPL&token=<API_Key >

Response:

Response Keys	**Details**
country	Country Name
currency	Currency Symbol
exchange	Company's Exchange
name	Company's Name
ticker	Company's Symbol
ipo	Company's Start Date
marketCapitalization	Company's MarketCap
shareOutstanding	Company's Shares
logo	Company's Logo
phone	Company's Contact No.
weburl	Company's Website Url
finnhubIndustry	Company's Industry

Table 4.1: Details regarding Company's Description API call

```
{
    "country": "US",
    "currency": "USD",
    "exchange": "NASDAQ NMS - GLOBAL MARKET",
    "finnhubIndustry": "Technology",
    "ipo": "1986-03-13",
    "logo": "https://finnhub.io/api/logo?symbol=MSFT",
    "marketCapitalization": 2245312,
    "name": "Microsoft Corp",
    "phone": "14258828080.0",
    "shareOutstanding": 7496.87,
    "ticker": "MSFT",
    "weburl": "https://www.microsoft.com/en-us"
}
```


Figure 4.1: Response received for Company's Description API call

4.1.2 Company's Historical Data

You should extract the content of Time Series Data from the returned JSON object to construct a chart which is responsible for displaying (close) price and volume. The chart is provided by **HighCharts.** Find more information about HighCharts at:

https://www.highcharts.com/demo

https://www.highcharts.com/demo/stock/area

https://www.highcharts.com/demo/stock/candlestick-and-volume

The historical data for the ticker can be obtained from ***Polygon.io "Aggregate (Bars)" Service***.

Please refer to the API documentation at:

https://polygon.io/docs/stocks/get_v2_aggs_ticker__stocksticker__range__multiplier__timespan__from__to

for more details on the Service.

API Sample:

GET: /v2/aggs/ticker/{stocksTicker}/range/{multiplier}/{timespan}/{from}/{to}

https://api.polygon.io/v2/aggs/ticker/AAPL/range/1/day/2023-01-09/2023-07-09?adjusted=true&sort=asc&apiKey=ctO8iVF_Gi19afBovU1ZSr6UIxqt8Fr3

When constructing the HTTP request required to obtain the Company Stock Chart, you need to provide 6 values:

1. The first value, the **stockTicker** the user entered in the form.
2. The second value, the **multiplier,** is the size of the timespan multiplier. Should be a **1**.
3. The third value, the **timespan,** should be **day**.
4. The fourth value, the **from** date, is the start of the aggregate time window. Either a date with the format YYYY-MM-DD or a millisecond timestamp. This should be **6 months and 1 day** prior to the current date. You can use Python DateUtils's "relativedelta" to calculate the date. Please refer to https://dateutil.readthedocs.io/en/stable/relativedelta.html for more details.
5. The fifth value, **to**, is the end of the aggregate time window. Either a date with the format YYYY-MM-DD or a millisecond timestamp. It should be the current date (**Today's date**).
6. The sixth value, the **query string**, should contain "*adjusted=true&sort=asc&apiKey=YOUR_KEY", where the **apiKey** is* your **Api Key** that you created. Do **not** use the **limit** parameter, as it will conflict with the **from/to** dates. Limit will default to 5,000, which is enough for 6 months of data.

Note: The Service only accepts dates in UNIX timestamps so you will need to convert dates to UNIX epoch time.

The **response** of this HTTP request is a **JSON-formatted object**. **Figure 4.2** shows a sample response from the request. You need to parse this JSON object and extract some fields as required.


```
{
  "c": 75.0875,
  "h": 75.15,
  "l": 73.7975,
  "n": 1,
  "o": 74.06,
  "t": 1577941200000,
  "v": 135647456,
  "vw": 74.6099
},
```


Figure 4.2: Sample JSON response from Polygon.io API's Aggregate (Bars) Endpoint

The data obtained from the API can then be mapped to the **HighCharts** dataset using the mapping below.

Chart Data	Data from polygon.io Aggregate (Bars) service JSON response
Date	The value of key *"t"*. The time stamp is given in the Unix epoch time.
Stock Price	The value of key *"c"*
Volume	The value of key *"v"*

Table 4.1: Mapping JSON data and HighCharts data

Note: Map each returned array element for each attribute, by its index. For example, for date (timestamp) t[0], close price is c[0] and volume is v[0].

For mapping the Stock price data to data for HighCharts, create an array of data points (x1, y1) where x1 will be the date and y1 will be the corresponding close stock price for that day. This array will then act as an input dataset for your HighCharts. Please refer to links in **Assignment 2** for more details.

Similarly create another array of points (x2, y2) where x2 will be the date and y1 will be the volume for that day. This array will be the second input for your HighCharts. Since you will be plotting Stock Price vs Date and Volume vs Date you will have two different datasets and two y-axis and a single x-axis.

Initially, the chart shows the historical stock price (in blue line with filling the area below, two digits after decimal) and volume (in gray bar) for the **past six months** by an interval of **one day**. **Figure 4.3** shows an example of the Stock Price/Volume chart.

Figure 4.3: An Example of Chart showing Stock Price/Volume for 6 months

The title of the chart for showing price/volume is "**Stock Price <Ticker> (YYYY-MM-DD)**", where "YYYY-MM-DD" is today's date. The subtitle of the chart should be "Source: Polygon.io" and should hyperlink to the Polygon.io website: https://polygon.io/. The title of the Y-axis is "**Stock Price**" when showing the stock price and the other Y-axis is "**Volume**".

The X-axis changes on the basis of the zoom level 6 months, 3 months, 1 month, 15 days, and 7 days. Please refer to **Section 3** for references on how to change the chart data on the basis of the zoom level**. Figure 4.4** shows an example of the Stock Price/Volume chart for 15 days zoom level. Ensure that the volume bars are small enough to not overlap the quote line chart.

Figure 4.4: An Example of Chart showing Stock Price/Volume for 15 days

4.1.3 Company's Latest Price of Stock (Stock Quote)

Reference: https://finnhub.io/docs/api/quote

For **Company's Latest Price**, use the following API. For more details refer to **Figure 4.5**:

https://finnhub.io/api/v1/quote?symbol=<TICKER>&token=<API_KEY>

URL parameter in API Call:

- Ticker: Ticker symbol of the stock. E.g.: MSFT
- Token: The API access Token. It is private, please do not share with anyone.

An example URL constructed from the parameters will look similar to this:

https://finnhub.io/api/v1/quote?symbol=MSFT&token=<API_KEY>

Response: We receive an array of objects, where each object contains following keys. We only need the following keys from the response.

Response Keys	Details
c	current price
d	change in price

dp	percentage change in price
h	high price of the day.
l	low price of the day.
o	open price of the day.
pc	Previous close price
t	Timestamp of last stock data

Table 4.2: Details regarding Company's Latest Price API call.

Market Status must be open if the difference between current Timestamp (current Timestamp will be created using new Date() in javascript) and 'timestamp' key is less than 60 seconds.


```
{
    "c": 291.98,
    "d": -7.52,
    "dp": -2.5109,
    "h": 296.8,
    "l": 291.41,
    "o": 296.36,
    "pc": 299.5,
    "t": 1645130548
}
```


Figure 4.5: Response received for Company's Latest Price API call

4.1.4 Autocomplete

Reference: https://finnhub.io/docs/api/symbol-search

For **Autocomplete**, use the following API. For more details refer Figure 4.6:

https://finnhub.io/api/v1/search?q=<QUERY>&token=<API_KEY>

URL parameter in API Call:

- Query: Search query of stock ticker . E.g.: TSL for TSLA
- Token: The API access Token. It is private, please do not share with anyone.

An example URL constructed from the parameters will look similar like:

https://finnhub.io/api/v1/search?q=AMZ&token=<API_KEY>

Response objects:

We receive a response in the form of an array of objects. From those objects we only need the following keys:

Response Keys	Details
count	number of results
result	array of search result
description	symbol description
displaySymbol	display symbol name
symbol	unique symbol used to identify this symbol used in /stock/candle endpoint.
type	security type

Table 4.3: Details regarding Autocomplete Response. API call


```
{
    "count": 34,
    "result": [
        {
            "description": "AZUCAR MINERALS LTD",
            "displaySymbol": "AMZ.V",
            "symbol": "AMZ.V",
            "type": "Common Stock"
        },
        {
            "description": "AZUCAR MINERALS LTD",
            "displaySymbol": "AMZ.NE",
            "symbol": "AMZ.NE",
            "type": "Common Stock"
        },
        {
            "description": "AMAZON.COM INC",
            "displaySymbol": "AMZ.MU",
            "symbol": "AMZ.MU",
            "type": "Common Stock"
        },
        {
            "description": "AMAZON.COM INC",
            "displaySymbol": "AMZ.DE",
            "symbol": "AMZ.DE",
            "type": "Common Stock"
        },
        {
            "description": "AMAZON.COM INC",
            "displaySymbol": "AMZ.HM",
            "symbol": "AMZ.HM",
            "type": "Common Stock"
        },
```


Figure 4.6: Response received for autocomplete API call

4.1.5 Company's News

Reference: https://finnhub.io/docs/api/company-news

For **Company's News**, use the following API. For more details refer to **Figure 4.7**.

https://finnhub.io/api/v1/company-news?symbol=<TICKER>&from=<DATE>&to=<DATE>&token=<API_KEY>

URL parameter in API Call:

- Ticker: Ticker symbol of the stock. E.g.: MSFT

- to: Date in YYYY-MM-DD
- from: Date in YYYY-MM-DD
- Token: The API access Token. It is private, please do not share with anyone. See Homework 2.

An example URL constructed from the parameters will look similar to this:

https://finnhub.io/api/v1/company-news?symbol=MSFT&from=2021-09-01&to=2021-09-09&token=<API_KEY>

Response objects:

Response Keys	**Details**
category	News category
datetime	Published timestamp in UNIX
headline	News headline
id	News id
image	Thumbnail image URL
related	Related stocks and companies mentioned
source	News source
summary	News summary
url	url of original article

Table 4.4: Details regarding Company's News API response


```
[
  {
    "category": "company",
    "datetime": 1631221522,
    "headline": "Arrival Might Be Worth A Speculative Look Here",
    "id": 70429020,
    "image": "https://static.seekingalpha.com/cdn/s3/uploads/getty_images/1249145859/medium_image_1249145859.jpg",
    "related": "MSFT",
    "source": "SeekingAlpha",
    "summary": "Arrival moves closer to production, with more demos and test trials occurring as microfactories in Rock Hill and Bicester move closer towards completion.",
    "url": "https://finnhub.io/api/news?id=ac5479a596494cf4d13a62b1459f5dc391485c5872a3f3b9e85420a13cc4d4c4"
  },
  {
    "category": "company",
    "datetime": 1631220499,
    "headline": "Microsoft rolls out new features for its Teams app",
    "id": 70430582,
    "image": "https://s.yimg.com/ny/api/res/1.2/mU2NHnevt8nU6TUKFBSYPA--/YXBwaWQ9aGlnaGxhbmRlcjt3PTEyMDA7aD02NzU-/https://s.yimg.com/hd/cp-video-transcode/prod/2021-09/09/613a7314ddb40d0fba1baecd/613a7314ddb40d0fba1baece_o_U_v2.jpg",
    "related": "MSFT",
    "source": "Yahoo",
    "summary": "Jared Spataro, Microsoft CVP of Modern Work, joined Yahoo Finance Live to discuss the new Teams app features and his outlook for the future of hybrid work.",
    "url": "https://finnhub.io/api/news?id=284ce70b7cd2233edcc61a10c6cebb7e160f76834e8ff667482af75996a64b7d"
  },
```


Figure 4.7: Response received for Company's News API call

4.1.6 Company's Recommendation Trends

Reference: https://finnhub.io/docs/api/recommendation-trends

For **Company's Recommendation Trends**, use the following API. For more details refer to **Figure 4.8**.

https://finnhub.io/api/v1/stock/recommendation?symbol=<TICKER>&token=<API_KEY>

URL parameter in API Call:

- symbol: Ticker symbol of the stock., e.g., MSFT
- token: The API access Token. It is private, please do not share with anyone. See Homework 2.

An example URL constructed from the parameters will look similar to this:

https://finnhub.io/api/v1/stock/recommendation?symbol=MSFT&token=<API_KEY>

Response objects:

Response Keys	Details
Buy	Recommendation count of buy category
Hold	Recommendation count of hold category
period	Update period
sell	Recommendation count of sell category
strongBuy	Recommendation count of strongbuy category
strongSell	Recommendation count of strongsell category
Symbol	Company symbol

Table 4.5: Details regarding Company's Recommendation API call


```
[
    {
        "buy": 32,
        "hold": 4,
        "period": "2022-02-01",
        "sell": 0,
        "strongBuy": 22,
        "strongSell": 0,
        "symbol": "MSFT"
    },
    {
        "buy": 29,
        "hold": 3,
        "period": "2022-01-01",
        "sell": 0,
        "strongBuy": 21,
        "strongSell": 0,
        "symbol": "MSFT"
    },
    {
        "buy": 27,
        "hold": 3,
        "period": "2021-12-01",
        "sell": 0,
        "strongBuy": 20,
        "strongSell": 0,
        "symbol": "MSFT"
    },
    {
        "buy": 25,
        "hold": 3,
        "period": "2021-11-01",
        "sell": 0,
        "strongBuy": 20,
        "strongSell": 0,
        "symbol": "MSFT"
    }
```


Figure 4.8: Response received for Company's Recommendation API call

4.1.7 Company's Insider Sentiment

Reference: https://finnhub.io/docs/api/insider-sentiment

For **Company's Insider Sentiment**, use the following API. For more details refer **Figure 4.9**:

https://finnhub.io/api/v1/stock/insider-sentiment?symbol=<TICKER>&from=2022-01-01&token=<API_KEY>

NOTE: 'from=2022-01-01' should be used as a default parameter while using company's social sentiment API calls, if not used can sometime return empty response.

URL parameter in API Call:

- Ticker: Ticker symbol of the stock. E.g.: MSFT
- From: Date in YYYY-MM-DD (2022-01-01 to be used)
- Token: The API access Token. It is private, please do not share with anyone. See Homework 2.

An example URL constructed from the parameters will look similar to this:

https://finnhub.io/api/v1/stock/insider-sentiment?symbol=MSFT&token=<API_KEY>

Response objects:

Response Keys	**Details**
data	array of entries containing month by month insider insight data
- symbol	Ticker symbol of the stock. E.g.: MSFT
- year	year of the data in this entry
- month	month of the data in this entry
- change	Net buying/selling from all insiders' transactions.
- mspr	Monthly share purchase ratio
symbol	Ticker symbol of the stock. E.g.: MSFT

Table 4.6: Details regarding Company's Insider Sentiment API call

```
{
  "data":[
    {
      "symbol":"TSLA",
      "year":2021,
      "month":3,
      "change":5540,
      "mspr":12.209097
    },
    {
      "symbol":"TSLA",
      "year":2022,
      "month":1,
      "change":-1250,
      "mspr":-5.6179776
    },
    {
      "symbol":"TSLA",
      "year":2022,
      "month":2,
      "change":-1250,
      "mspr":-2.1459227
    },
    {
      "symbol":"TSLA",
      "year":2022,
      "month":3,
      "change":5870,
      "mspr":8.960191
    }
  ],
  "symbol":"TSLA"
}
```


Figure 4.9: Response received for Company's Insider Sentiment API call

4.1.8 Company's Peers

Reference: https://finnhub.io/docs/api/company-peers

For **Company's Peers**, use the following API. For more details refer **Figure 4.10**:

https://finnhub.io/api/v1/stock/peers?symbol=<TICKER>&token=<API_KEY>

URL parameter in API Call:

- Ticker: Ticker symbol of the stock. E.g.: MSFT
- Token: The API access Token. It is private, please do not share with anyone. See Homework 2.

An example URL constructed from the parameters will look similar to this:

https://finnhub.io/api/v1/stock/peers?symbol=MSFT&token=<API_KEY>

Response array:

Response Keys	Details
response	List of company symbols

Table 4.7: Details regarding Company's Peers API call

```
[
    "MSFT",
    "ORCL",
    "NOW",
    "VMW",
    "FTNT",
    "PANW",
    "CRWD",
    "ZS",
    "PATH",
    "NLOK"
]
```


Figure 4.10: Response received for Company's Peers API call

4.1.9 Company's Earnings

Reference: https://finnhub.io/docs/api/company-earnings

NOTE: If any of the response values are null values for response keys, replace null values to 0. Sample null response received from Finnhub API (Figure 4.11.1) which should be handled and replaced with 0.

For **Company's Earnings**, use the following API. For more details refer **Figure 4.9.1 and Figure 4.11.2**:

https://finnhub.io/api/v1/stock/earnings?symbol=<TICKER>&token=<API_KEY>

URL parameter in API Call:

- Ticker: Ticker symbol of the stock. E.g.: MSFT
- Token: The API access Token. It is private, please do not share with anyone. See Homework 2.

An example URL constructed from the parameters will look similar to this:

https://finnhub.io/api/v1/stock/earnings?symbol=MSFT&token=<API_KEY>

Response objects:

Response Keys	Details
Actual	Actual earnings results
Estimate	Estimated earnings
Period	Reported period
Symbol	Company symbol

Table 4.8: Details regarding Company's Earnings API call

```
[
    {
        "actual": null,
        "estimate": 2.7008,
        "period": "2023-06-30",
        "surprise": null,
        "surprisePercent": null,
        "symbol": "MSFT"
    },
    {
        "actual": null,
        "estimate": 2.5295,
        "period": "2023-03-31",
        "surprise": null,
        "surprisePercent": null,
        "symbol": "MSFT"
    },
    {
        "actual": null,
        "estimate": 2.4229,
        "period": "2022-12-31",
        "surprise": null,
        "surprisePercent": null,
        "symbol": "MSFT"
    },
    {
        "actual": null,
        "estimate": 2.2157,
        "period": "2022-09-30",
        "surprise": null,
        "surprisePercent": null,
        "symbol": "MSFT"
    }
]
```


Figure 4.11.1: Null Response received for Company's Earnings API call


```
[
    {
        "actual": 2.48,
        "estimate": 2.3564,
        "period": "2022-06-30",
        "surprise": 0.1236,
        "surprisePercent": 5.2453,
        "symbol": "MSFT"
    },
    {
        "actual": 2.27,
        "estimate": 2.1163,
        "period": "2022-03-31",
        "surprise": 0.1537,
        "surprisePercent": 7.2627,
        "symbol": "MSFT"
    },
    {
        "actual": 2.17,
        "estimate": 1.9543,
        "period": "2021-12-31",
        "surprise": 0.2157,
        "surprisePercent": 11.0372,
        "symbol": "MSFT"
    },
    {
        "actual": 1.95,
        "estimate": 1.8142,
        "period": "2021-09-30",
        "surprise": 0.1358,
        "surprisePercent": 7.4854,
        "symbol": "MSFT"
    }
]
```


Figure 4.11.2: Response received for Company's Earnings API call

4.2 Social Networks

4.2.1 X (formerly known as Twitter)

Refer the following link for details:
https://developer.twitter.com/en/docs/twitter-for-websites/tweet-button/overview

4.2.2 Facebook

Refer the following link for details:
https://developers.facebook.com/docs/plugins/share-button/

5. Implementation Hints

5.1 ng-bootstrap Library

To get started with the ng-bootstrap toolkit, please see:
https://ng-bootstrap.github.io/#/home

ng-bootstrap will work with both Bootstrap 4 and 5, but with a specific version of Angular. Check compatibility and dependencies at:

https://ng-bootstrap.github.io/#/getting-started

Modules helpful for implementation:

Alerts - https://ng-bootstrap.github.io/#/components/alert/examples
Modal - https://ng-bootstrap.github.io/#/components/modal/examples

5.2 Bootstrap UI Components

Bootstrap provides a complete mechanism to make web pages responsive for different mobile devices. In this exercise, you will get hands-on experience with responsive design using the Bootstrap Grid System.

Bootstrap 4.6 grid:
https://getbootstrap.com/docs/4.6/layout/grid/

Bootstrap 5.2 grid:
https://getbootstrap.com/docs/5.2/layout/grid/

Some components that are useful for implementation:

Bootstrap Cards https://getbootstrap.com/docs/4.6/components/card/

Bootstrap Navbar https://getbootstrap.com/docs/4.6/components/navbar/

5.3 Angular

- Angular Set up – https://angular.io/
- Angular Material Installation - https://material.angular.io/guide/getting-started
- Angular Material Tabs - https://material.angular.io/components/tabs/overview
- Angular Material Spinner – https://material.angular.io/components/progress-spinner/overview
- Angular Material Autocomplete - https://material.angular.io/components/autocomplete/overview
- Angular Routing – https://angular.io/guide/routing-overview

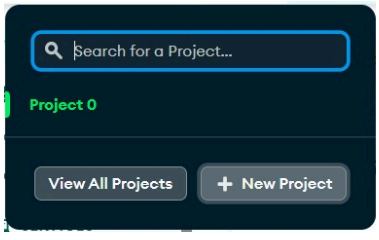
5.4 MongoDB Atlas

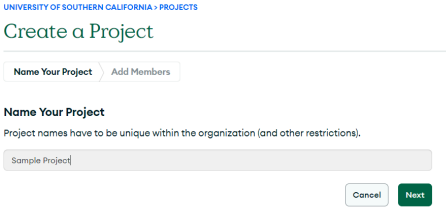
*MongoDB Atl*as is a source-available cross-platform document-oriented database program. It is classified as a NoSQL database program. MongoDB Atlas uses JSON-like documents with optional schemas. For more information, see: https://www.mongodb.com/docs/

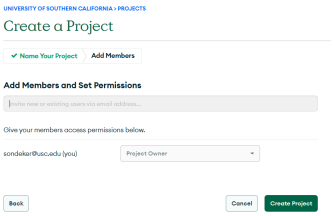
MongoDB on Google Cloud: https://www.mongodb.com/mongodb-on-google-cloud
MongoDB on AWS: https://www.mongodb.com/mongodb-on-aws
MongoDB on Azure: https://www.mongodb.com/mongodb-on-azure

Once you set up an account in MongoDB Atlas, you will have to create a project to store your databases. In a project you will create a database to store collections which hold the watchlist and protfolio information data in NoSQL format. Below are the steps to set up a project and create a database.

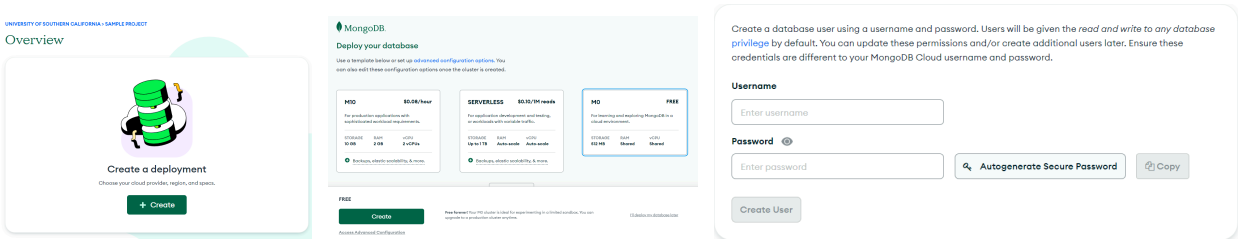
Follow these steps create a project in which you can store databases for your application.

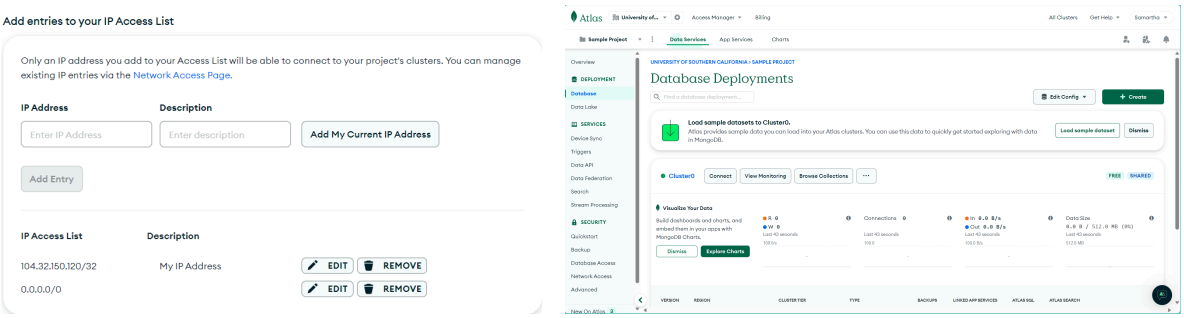




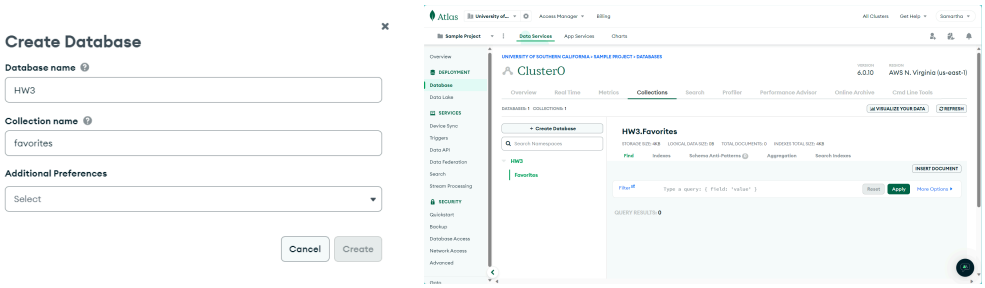


Follow these steps to create a deployment for the project by creating a cluster and providing user access and network access for the same. This would allow your application, running on a different server, to connect to MongoDB Atlas in the cloud.

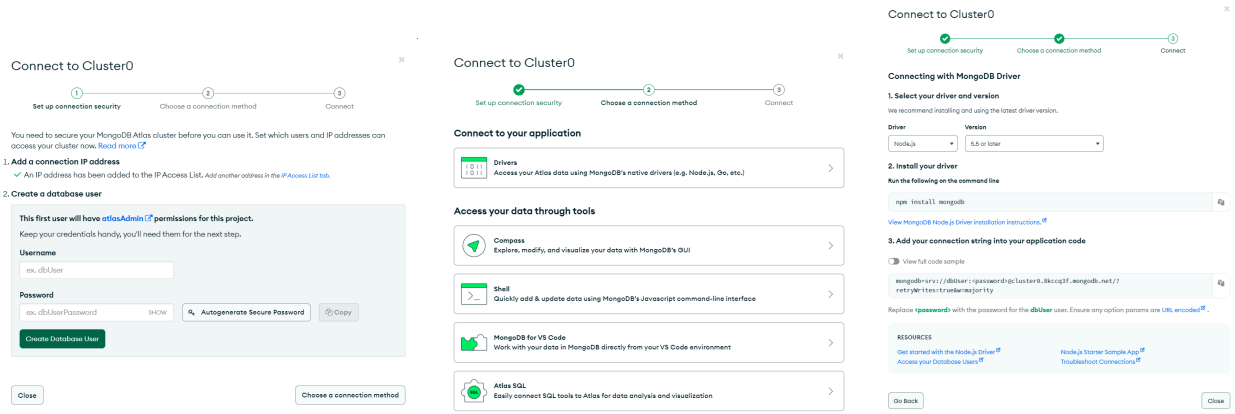




Follow these steps to create a database and a collection in that database. MongoDB stores data records as documents (specifically in BSON format) which are gathered in collections. A database stores one or more collections of documents.



Follow these steps to provide the instructions on how to connect to your MongoDB database from your application.



Once you have set up the database and the collection, you can connect to the database by adding the MongoDB node driver to your application. For more information how to add the driver and run queries on the database, refer to [MongoDB Node Driver — Node.js]

5.5 highcharts-angular

Refer the following documentation on **highcharts-angular** and how to use it.

https://www.npmjs.com/package/highcharts-angular

https://www.tutorialspoint.com/angular_highcharts/angular_highcharts_quick_guide.htm

https://github.com/highcharts/highcharts-angular

Chart type references:

https://www.highcharts.com/docs/chart-and-series-types/column-chart#stacked-column-chart

https://www.highcharts.com/docs/chart-and-series-types/spline-chart

https://www.highcharts.com/demo/stock/sma-volume-by-price

5.6 Icons

Reference: https://fontawesome.com/search

- https://icons.getbootstrap.com/icons/caret-up-fill/
- https://icons.getbootstrap.com/icons/caret-down-fill/
- https://icons.getbootstrap.com/icons/star/
- https://icons.getbootstrap.com/icons/star-fill/

- https://icons.getbootstrap.com/icons/x/
- https://fontawesome.com/icons/facebook-square?s=brands
- https://fontawesome.com/icons/twitter?s=brands

5.6 Deploy Node.js application on Cloud Services

Since this assignment is implemented with Node.js on Cloud Services, you should **select Nginx as your proxy server (if available)**, which should be the default option.

6. Files to Submit

In your course homework page on GitHub Pages, you should update the **Assignment #3 link** to refer to your new initial web page for this exercise. Your files must be hosted on the same service: Google Cloud, AWS or Azure. Graders will verify that this link is indeed pointing to one of the cloud services. Additionally, you need to provide **an additional link to** the URL of **the cloud** service where the AJAX call is made with sample parameter values. When this link is followed, JSON is expected as the output.

Also, submit your source code file to D2L Brightspace. Submit a **single ZIP file** that includes both the front-end and back-end code, plus any additional files needed to build your app. **The timestamp of the ZIP file will be used to verify if you have used any "grace days."**

****IMPORTANT**:**

- All explanations and clarifications provided in Piazza related to this homework are part of the homework description and grading guidelines. So please review all Piazza threads, before finishing the assignment. If there is a conflict between Piazza and this description and/or the grading guidelines, **Piazza always rules**.
- You **should not call any of the Finnhub or Polygon.io APIs directly from JavaScript**, bypassing the NodeJS proxy. Implementing any one of them in "client" JavaScript instead of NodeJS will result in a **4-point penalty.**
- **Appearance of all views, tables, and charts should be similar to the snapshots in this document and the reference videos as much as possible.**