

CSCI201 Group Project - Trojan Trade

Mentor: Devanshu Gupta

Group Number: 22

Weekly Meeting Time: Sundays at 1:00 PM

Group Member and Contact Information:

First Name	Last Name	Email
Jingzhen(Steve)	Wang	jwang977@usc.edu
Tianyu(Samuel)	Wang	twang508@usc.edu
Loui	Ikei	ikei@usc.edu
Xiayu	Li	xiayuli@usc.edu
Peter	Palanjan	ppalanji@usc.edu
Angela(Shuya)	Liu	shuyal@usc.edu
Sirui	Zhu	sirui-zhu@usc.edu

Project Proposal

Project Name:

Stock Trading Simulator

Background:

Due to the current economic downturn, the stock market is full of uncertainties and risks. For many people, investing in the stock market can seem daunting and overwhelming, as it requires a deep understanding of financial markets and a willingness to take calculated risks. To address this challenge, there is a growing need for educational tools and resources that can help people learn about the stock market and gain experience in buying and selling stocks in a simulated environment. A stock market simulation web application allows those traders to experience and practice stock trading in a safe and risk-free environment without risking real money, thus enabling them to become well-prepared for the real-world stock market.

Problem Statement:

The stock market is a complex and dynamic system that can be difficult to understand for those without experience in finance.

Target Audience:

The target audience should be those who are interested in trading stocks, but do not have sufficient knowledge or experience to use real money to trade. The website will use real data, giving users a genuine experience.

Overall Description of Project:

Our proposed web application aims to provide a platform for users to learn about the stock market and gain experience in buying and selling stocks through a simulated environment. It fetches information from the real stock market and allows users to practice stock trading. The application will allow users to create virtual accounts, view stock market data, and simulate buying and selling stocks. The application will also include a leaderboard that ranks users based on their performance in the simulation.

An example scenario for a user's experience could include a home page that provides the option for users to create a new account, log in using their existing personal accounts, or visit as guests. Guests are only allowed to view the stock but can't make any decisions on buying or selling. Authenticated users who log in with IDs and passwords can set up their account with an amount balance and decide to buy or sell at available trading time.

High-Level Requirements

We are designing a web-based application allowing users to practice stock trading.

The webpage should have a straightforward and concise interface, allowing users to register and create accounts and then authenticate themselves to access the trading platform. Users can input their usernames and passwords, and press the "Login" button to access their accounts. There should also be a "Create Account" button below that, once clicked, will jump to the account creation page and a "Guest Login" button for those who do not want to create an account.

The website should provide a virtual trading platform that allows users to simulate buying and selling stocks, view their portfolios, and track their performance. Real-time market data, such as stock prices, charts, and news, are also provided for making informed trading decisions. Each new user should be allocated a starting amount of currency (pending). The user should then be able to choose from a list of stocks available. Each stock should have its own page with certain data (24 Hour High and Low, Current Price, etc). The page should also include buy and sell buttons.

Educational resources, such as articles, are provided to help users learn about trading stocks and improve their skills. Additionally, users should be able to interact with each other and share knowledge and experiences in forums and chat rooms.

The website should allow administrators to manage users, including adding and deleting accounts, changing user information, and tracking user activity. All performance should have security measures to protect user data, prevent unauthorized access, and ensure the integrity of the trading platform.

Technical Specifications

Web Interface (15 Hours)

This will include basic front-end functionality, linking together many of the other aspects of the program. Specifically, we will include things like the login page buttons, buy and sell buttons, and general website interactions. The prices and user information will be gathered from the database and presented in a readable format. On pages like the buy and sell, the web interface will call APIs to generate graphs of a stock's history.

Data Scraping (4 Hours)

This functionality uses data scraping bot or API from open source to gain up-to-date stock market information. The data gained will be stored in our database. Our server will request the stock information every second. The data stored in the database will be shown on the index page of the interface. When the users request the information, whether they view/buy/sell the stock, they will request the information from the database.

Education (3 Hours)

The website will feature a curated assortment of educational resources including articles, graphics, and real-time stock information. The "education" tab of the web interface will feature 3 sections: a static "how to trade" article, a dynamic list of daily-updating articles regarding news in any of our listed stocks, and an updated list of stock prices for all the stocks listed on our service.

Database (10 Hours)

Our database mainly consists of four tables. The first one will be utilized to save user information like usernames and passwords for validation uses, as well as their current balance. In addition, current stock prices and related information will be stored in the second database. There should also be a table saving the users' last ten transactions for later display in the personal information section on the web interface, including data like names of the stocks, buying/selling prices, and the time they started transactions. And the last table should specify each user's current stock possession including stock information and the amount they have currently.

Detailed Design

1. Hardware and Software Requirements:

FRONTEND: HTML, CSS, JS, React

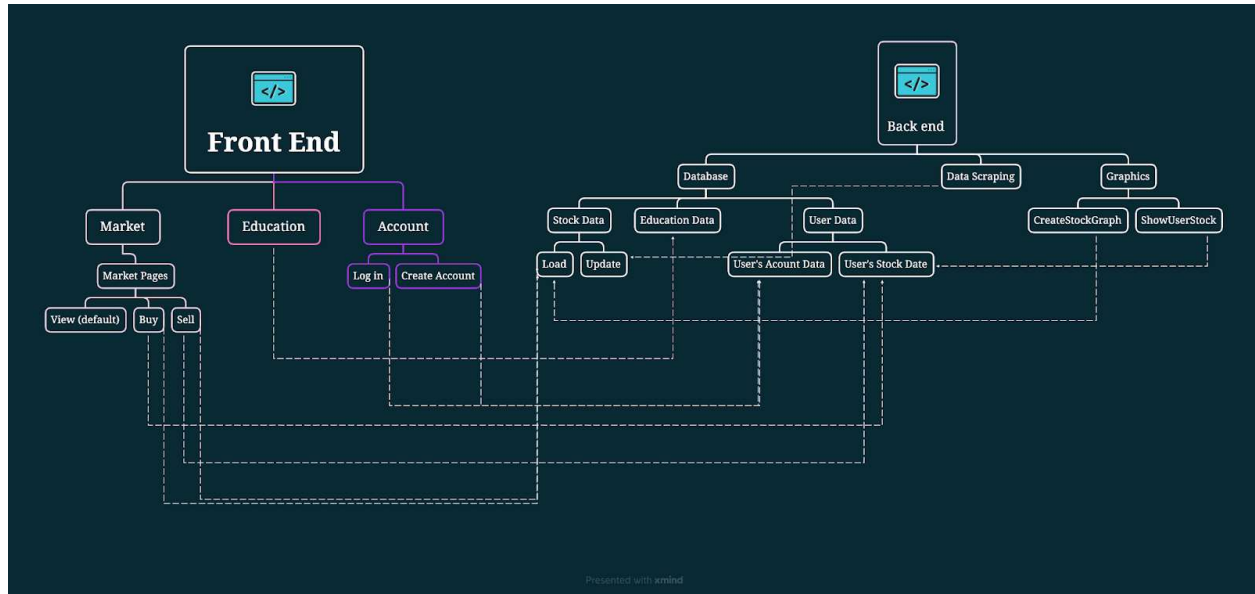
BACKEND: Java, Spring Framework

Database: MySQL, JDBC

Version Control: Git, Github

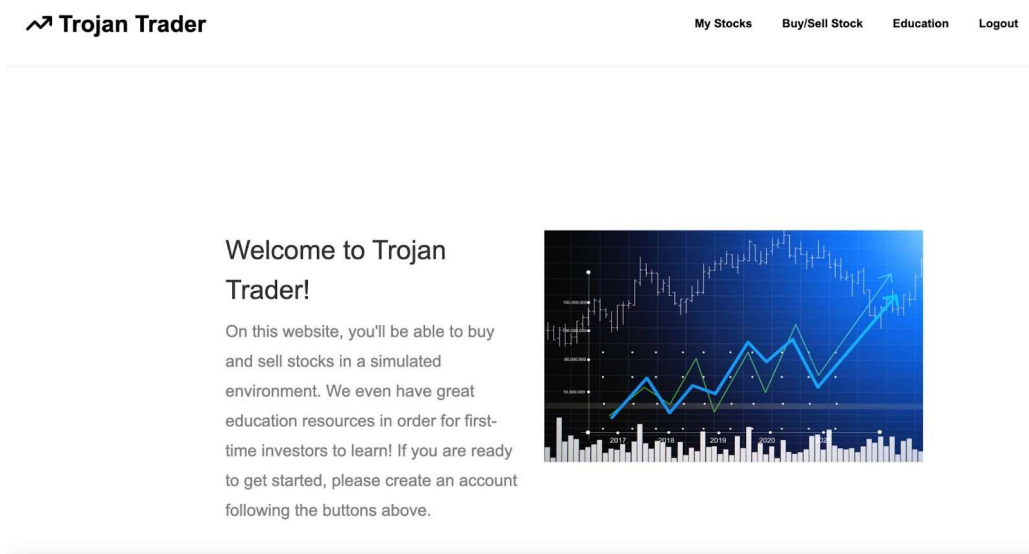
Hardware: Since this is a web application, machines that can run up-to-date browsers, like Google Chrome, will be able to access the entire application.

2. Class Diagram and Inheritance Hierarchy



3. Front End User Interface

a. Home Page



b. Login/Signup Page

Log in

Sign in

© Trojan Trader 2023


Create an account to begin trading!

Create Account

© Trojan Trader 2023

c. Trade Dashboard


Trojan Trader
My Stocks
Buy/Sell Stock
Education
Logout



Apple

Apple is a technology company based in California that sells consumer electronics, software, and services. It's known for its innovative design, user-friendly interface, privacy, and security.


Enter an Amount
Buy
Sell



Tesla

Tesla is an American company that designs, manufactures, and sells electric cars, energy storage systems, and solar products. It's known for its innovative technology and commitment to sustainable energy.


Enter an Amount
Buy
Sell



Amazon

Amazon is a Seattle-based technology company that primarily operates as an online retailer. It offers a wide range of products, services, and fast delivery. It prioritizes customer satisfaction.

Enter an Amount
Buy
Sell




Microsoft

Microsoft is an American technology company that develops and sells computer software, electronics, and personal computers. It's known for its innovative technology and empowering mission.

Enter an Amount
Buy
Sell


d. Education Page

Trojan Trader
My Stocks
Buy/Sell Stock
Education
Logout




How Stocks Work

This NerdWallet article provides guidance for beginners on how to start trading stocks. It covers essential steps such as opening a brokerage account, researching stocks, and creating a diversified portfolio.



Most Active Stocks


The Yahoo Finance "Most Active" page displays real-time data on the stocks with the highest trading volumes. Users can sort and filter by different metrics, including price, volume, and market cap.




How Stock Taxes Work

The NerdWallet article explains the tax implications of buying and selling stocks. Generally, profits are taxable, and capital gains on the sale of stocks are also usually taxed.


e. My Stock Page




AAPL
Apple Inc.




MSFT
Microsoft Inc.



TSLA
Tesla Inc.



AMZN
Amazon Inc.



twang508@usc.edu
Verified User

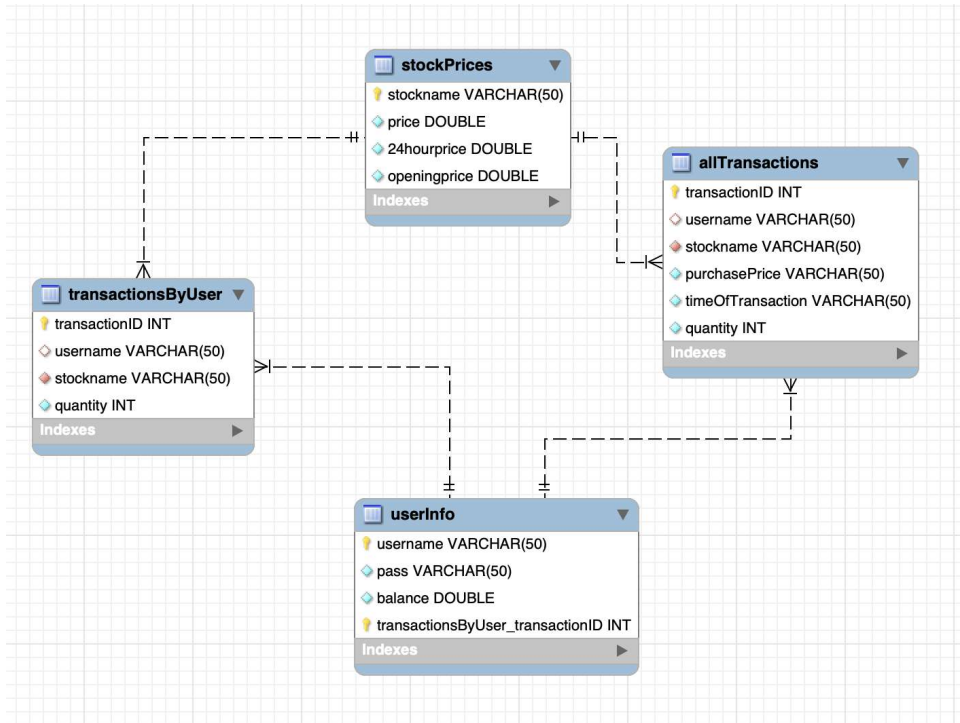
About

Exchange: NASDAQ

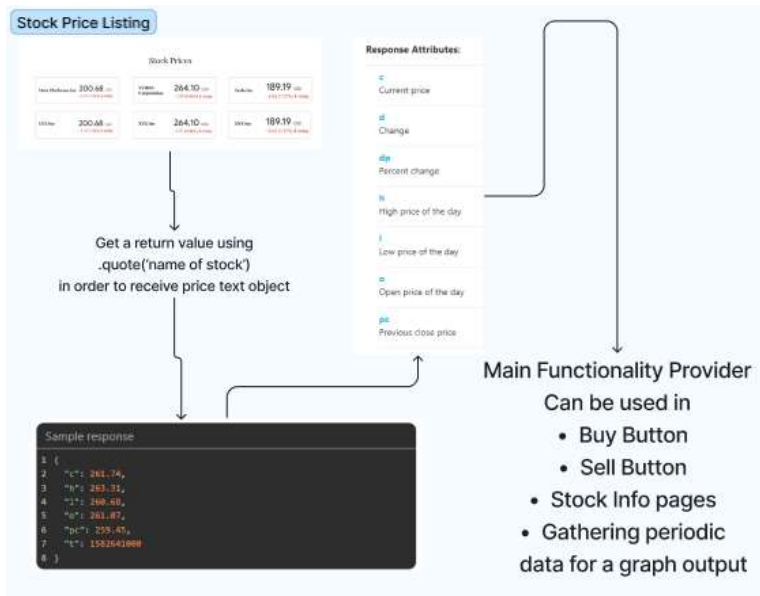
Number of Trades:

Balance: \$0.00

4. Database Schema



5. API



Stock Lookup Page

Use api call `finnhub_client.symbol_lookup('stock name')`

Sample response

```
1 {
2   "count": 4,
3   "result": [
4     {
5       "description": "APPLE INC",
6       "displaySymbol": "AAPL",
7       "symbol": "AAPL",
8       "type": "Common Stock"
9     },
10    {
11      "description": "APPLE INC",
12      "displaySymbol": "AAPL.SM",
13      "symbol": "AAPL.SM",
14      "type": "Common Stock"
15    },
16    {
17      "description": "APPLE INC",
18      "displaySymbol": "APC.DE",
19      "symbol": "APC.DE",
20      "type": "Common Stock"
21    },
22    {
23      "description": "APPLE INC",
24      "displaySymbol": "APC.BE",
25      "symbol": "APC.BE",
26      "type": "Common Stock"
27    }
28  ]
29 }
```

Use in

- Stock search bar
- To provide relevant search results

Long Term Data

`finnhubClient.companyBasicFinancials("AAPL", "all", (error, data, response))`

Sample response

```
1 {
2   "series": {
3     "annual": {
4       "currentRatio": [
5         {
6           "period": "2019-09-28",
7           "v": 1.5401
8         },
9         {
10          "period": "2018-09-29",
11          "v": 1.1329
12        }
13      ],
14      "salesPerShare": [
15        {
16          "period": "2019-09-28",
17          "v": 55.9645
18        },
19        {
20          "period": "2018-09-29",
21          "v": 53.1178
22        }
23      ],
24      "netMargin": [
25        {
26          "period": "2019-09-28",
27          "v": 0.2124
28        },
29        {
30          "period": "2018-09-29",
31          "v": 0.2241
32        }
33      ]
34    },
35    "metric": {
36      "180DayAverageTradingVolume": 32.50147,
37      "52WeekHigh": 310.43,
38      "52WeekLow": 149.22,
39      "52WeekLowDate": "2019-01-14",
40      "52WeekPriceReturnDaily": 101.96334,
41      "beta": 1.2989,
42    },
43    "metricType": "all",
44    "symbol": "AAPL"
45  }
46 }
```

Can be used to generate relevant graphs.

Live News

Sample code

```

1 const socket = new WebSocket('wss://ws.finnhub.io?token=');
2
3 // Connection opened -> Subscribe
4 socket.addEventListener('open', function (event) {
5   socket.send(JSON.stringify({'type': 'subscribe-news', 'symbol': 'AAPL'}));
6   socket.send(JSON.stringify({'type': 'subscribe-news', 'symbol': 'MSFT'}));
7   socket.send(JSON.stringify({'type': 'subscribe-news', 'symbol': 'GOOGL'}));
8 });
9
10 // Listen for messages
11 socket.addEventListener('message', function (event) {
12   console.log('Message from server: ', event.data);
13 });
14
15 // Unsubscribe
16 var unsubscribe = function(symbol) {
17   socket.send(JSON.stringify({'type': 'unsubscribe-news', 'symbol': symbol}));
18 }
19
20

```

Sample response

```

1 {
2   "data": [
3     {
4       "category": "company",
5       "datetime": 1679920200,
6       "headline": "Canalaska Uranium Announces Airborne Electromagnetic",
7       "id": 119440189,
8       "image": "https://media.1enfs.com/en/newsfile_64/8512ee5d9e134e",
9       "related": "CVVUF",
10      "source": "Yahoo",
11      "summary": "Multiple Priority Targets Associated with Regional",
12      "url": "https://finance.yahoo.com/news/canalaska-uranium-announ",
13    }
14  ],
15   "type": "News"
16 }

```

Relevant articles



Earnings Reports (to help new users make better decisions)

Sample code

```

1 const finnhub = require('finnhub');
2
3 const api_key = finnhub.ApiClient.instance.authentications['api_key'];
4 api_key.apiKey = '';
5 const finnhubClient = new finnhub.DefaultApi();
6
7 finnhubClient.companyEarnings('AAPL', {'limit': 10}, (error, data, res) {
8   console.log(data);
9 });
10

```

Sample response

```

1 {
2   (
3     {
4       "actual": 1.88,
5       "estimate": 1.9744,
6       "period": "2023-03-31",
7       "quarter": 1,
8       "surprise": -0.0944,
9       "surprisePercent": -4.7812,
10      "symbol": "AAPL",
11      "year": 2023
12    },
13    {
14      "actual": 1.29,
15      "estimate": 1.2957,
16      "period": "2022-12-31",
17      "quarter": 4,
18      "surprise": -0.0057,
19      "surprisePercent": -0.4399,
20      "symbol": "AAPL",
21      "year": 2022
22    },
23    {
24      "actual": 1.2,
25      "estimate": 1.1859,
26      "period": "2022-09-30",
27      "quarter": 3,
28      "surprise": 0.0141,
29      "surprisePercent": 1.2231,
30      "symbol": "AAPL",
31      "year": 2022
32    }
33  )
34 }

```

Response Attributes:

actual
Actual earning result.

estimate
Estimated earning.

period
Reported period.

quarter
Earnings quarter.

surprise
Surprise - The difference between actual and estimate.

surprisePercent
Surprise percent.

symbol
Company symbol.

year
Earnings year.

Testing Plan

UI/UX Testing

Our UI/UX testing will consist of white box testing, unit testing, component testing, and regression testing. The aim of UI testing is to ensure that the user interface of an application functions correctly and meets the users' expectations. UI testing is a critical aspect of software testing, as it is the primary point of interaction between the user and the application.

1. White Box Testing

The UI of this project provides an interactive interface for users to perform various stock market transactions, view their portfolios, and track their investment performance. Through UI testing, we want to ensure that the UI components, such as buttons, forms, links, and menus, are displayed correctly and function as expected; verify that the user interface is consistent across different browsers and devices; check that the user interface is intuitive and easy to use, and meets the user's expectations; identify and resolve user interface issues, such as incorrect data validation, broken links, incorrect formatting, and spelling errors.

2. Unit testing

1) Login testing:

- a) Login correctly will take the users to their trading homepage dashboard.
- b) Login incorrectly will display error messages like "invalid username or password" to the users and ask them to try again.
- c) Login incorrectly will allow the user to re enter their information below the error message.

2) Trading Testing:

- a) Buy/Sell successfully should change the users' balances correspondingly.
- b) Successful Buy/Sell actions should appear in the "last ten transactions" section correspondingly.

3) Result Testing:

- a) The stocks have the correct current price, purchase price, purchase date, and all other info.
- b) All available stocks are printed in marked color(Green for dropping and red for rising).
- c) Sorted from top to bottom alphabetically by the name of the stock.

4) Stock search testing:

- a) When you select a stock to add the check mark is filled properly.
- b) Unselecting a stock will unhighlight the box and if added will not add unselected stock that previously was chosen.
- c) The submission box works properly.

d) Previous selections will not affect current selections.

3. Regression Testing

- 1) We will do our final Regression Test at the UI/UX level. We will go through each aspect of the site and test extensively each component works. We will reuse most of our unit and component tests here and combine them into an extensive system-wide test. This will be after the release tests and will be treated as the final stage of testing. Possibly have another set of release tests and regression tests based on our timeline.

Database/Backend Testing

1. API Testing (White-box)

The main functionalities of our program will oftentimes be related to the various API calls we use. We will mainly be using finnhub.io as our interface to access various stock information. Since this API was developed and tested professionally (it is also used worldwide), we can have peace of mind knowing that the return values should work as expected. Thus, we only need a few unit tests (for sanity's sake) in terms of raw return values.

For any of the five top stocks, we will run them in our search feature, and confirm the output values on our website are accurate. After calling the `.quote()` function, we should check the GET return parameter and make sure the entire object is accurate.

Examples:

1. "C" is a valid current price
2. "D" is a valid change in price
3. "H" and "L" are valid highs and lows

Since, again, the API should be fully functional, our tests will implement white-box checks, as our source code implementation is the only way to get undesirable results. The API will be tested using the UI/UX as the interface.

After our initial implementation is done, we should run the API tests. After our whole project is complete, API will be tested again during regression testing.

2. Login Testing

Unit testing to ensure that each userID is correctly mapped to a specific user, ensuring that each ID in the system has an associated account and that every account has an associated userID.

Unit testing to verify that the backend stores user credentials securely, such as using password hashing, and that sensitive user data is not accessible to unauthorized users.

White box testing to ensure that there are no invalid IDs, such as those containing letters or unknown symbols.

White box testing to ensure correctly authentication of the user's credentials and grant access to the application's features only to authorized users

3. Trading Testing

Successful transactions(buy/sell) should update the balance column in the database based on userID and the price of corresponding stocks.

Successful transactions(buy/sell) actions should put or update the stocks information inside the users' owned stocks table.

Successful transactions(buy/sell) should put or update the actions information into the last-ten transactions table respectively.

Server Testing

The main focus of our Server testing will be to ensure that connections from Client to Server are working properly and that data is being transmitted from Server to Client in an efficient and swift manner. For this reason, we will be using various black box tests during our System Testing and Release

Testing phases:

1. Ensure server latency is below 100ms (at USC.)
2. Verify that Clients are able to connect to the database by way of the server to receive their personal user data (release test.)
3. Ensure that any dynamic graphics or text that may be generated as a result of server-size communication is appearing as intended.
4. Performance test to verify that the system can process multiple users (ex. ten) at the same time and run without any outstanding adverse effects or latencies.

Deployment

UI/UX

Launch the website on localhost port 8888 (user lands on page index.html).

Log in or sign up using the form on the index.html page and redirect the user to User_Page_Home.html.

Redirect to User_Page_Purchase.html by clicking the "Purchase Stock" button.

Add stocks to the cart by checking boxes on "User_Page_Stock" page and clicking the "Purchase" button to submit purchases.

Display the user portfolio page when users click the user icon.

Run testings.

Server

Create a web server with port 8888 to connect to the client.

Use JDBC to connect MySQL database with Java servlets.

Ensure that HTML/CSS forms can communicate with Java. Accept and identify the input information from the client.

Ensure that information gets correctly passed to the database.

UserID and password should be saved and usable.

Account and stock information should come up correctly.

Run black box tests to verify the functionality.

Data Collection

Send relevant collected data updates using API to the database (user balance updates using current price, 24-hour high, etc.)

We can push the data to git.

Ensure that local data matches the data scraped.

Run SQL scripts

Regression Testing for final deployment

Whenever we update different values in the local database, we must make sure we can update both locally and globally.

On initial deployment, some of the fields may not be filled, due to the timings of when to scrape data. Thus, we should initialize NULL.

Make sure on the initial runs of the website, the API only pushes data to existing databases

This means SQL creation scripts should always run first.

Backend/Database

Create the MySQL database on the server using DBMS MySQL workbench.

Import data like userID and stock information into their corresponding tables.

Connect DB to server side Java code using JDBC.

Run SQL scripts to verify the occurrence of data migration.

Run testings.