Team Alex, Jennifer, Vadim

1) **An issue with the stock page loading price and information**
    a) After initially searching up a stock price, or selecting a stock that the user owns, any subsequent searches for a new stock would continuously display the previous stock's information. The solution required the stock quote information to return the response and await on a variable waiting for the response.
2) **Displaying Financial Information**
    a) The FinnHub API also allowed for us to query financial information at any time frame, so we decided to query the previous quarter. However, the APIs response at times didn't always match the format we were expecting. We required data from the report object, an example of this issue would be:


```
{
    "cik": "320193",
    "data": [
        {
            "accessNumber": "0000320193-24-000006",
            "symbol": "AAPL",
            "cik": "320193",
            "year": 2024,
            "quarter": 1,
            "form": "10-Q",
            "startDate": "2023-10-01 00:00:00",
            "endDate": "2023-12-30 00:00:00",
            "filedDate": "2024-02-02 00:00:00",
            "acceptedDate": "2024-02-01 18:03:38",
            "report": {
```

```
{
    "cik": "320193",
    "data": [
       {
              "Misc": ...
       }
        {
            "accessNumber": "0000320193-24-000006",
            "symbol": "AAPL",
            "cik": "320193",
            "year": 2024,
            "quarter": 1,
            "form": "10-Q",
            "startDate": "2023-10-01 00:00:00",
```

```
"endDate": "2023-12-30 00:00:00",
"filedDate": "2024-02-02 00:00:00",
"acceptedDate": "2024-02-01 18:03:38",
"report": {
```


- We tried to place as many edge cases as we could but not all stocks were covered, so if the response was not in the expected format we would display, “Could not retrieve financial information”

3) **The Rotating Navigation Bar is not rotating far enough**
    a) Based on the Udemy course for a rotating nav bar, our implementation of the navigation worked properly until more HTML was placed on top of the nav bar, it would rotate to the point where both the close and open buttons were displayed. By analyzing the code and performing guesses and checks we fixed this by placing the close and open buttons farther apart and increasing the rotation amount and decreasing rotation speed
4) **Rotating Navigation Jumping**
    a) After opening or closing the rotating nav bar, the circle that had the buttons to open and close would jump up and down before completely closing. The fix we had for this was to change the css, and move the circle manually before starting the rotation
5) **Environment Variables**
    a) Our server.js contained all of our backend code and pipelines from and to our database and API calls to FinnHub. Due to this, we also had to reference our API key and our database password as plain text in the code. We understood this to be a major security hazard, we also knew about the existence of env variables, but we believed them to be similar to PATH variables and placed our API keys and passwords in this method. This was not the case and instead after further research, we found the dotenv module which can use a .env file and reference any key/value pairs within that file to fill in any information you need without requiring the plain text.
6) **Issues with CORS**
    a) In our previous project we had webpages that were statically linked to one another and had scripts attached to each html page that were querying our database, this required a CORS exception on the database side which would allow traffic from a specific URL. However, this project required node to host HTML, CSS and JS, and since node also has CORS restrictions it took a while before we realize that exceptions should be placed within our node server, server.js as well.
7) **Chat Bot**
    a) In the early stages of development, we created a direct copy of the linkedin learning chatbot which would communicate to a Slack chatroom. After getting familiar with it we decided to make a test page that the chat bot will communicate with, this caused alot of issues like require not being defined, modules must be in

a mjs file, and const not being defined. What we did to fix it was create a basic server.js file and bring in the required classes and methods from the linkedin learning course along with the end point of the queries to the NLP/LLM API recommendation from the course. This fixed all of the errors described previously and we were able to communicate with the chatbot.

8) **Hashing**
    a) When we first began the project we were storing passwords as plaintext for testing purposes, we then moved on to hashing the passwords with the same salt, after some research we learned we needed to use unique salts during every hash, but we were confused as to how and where we would store these unique salts so that we can check user passwords. After reading through the documentation for the module bcrypt we found that the salt is stored within the hash itself, and bcrypt can just get the salt from the hash apply to the user input compare it to the hash stored in the database and confirm whether it is a valid password

Hash format for bcrypt:

$algo$cost$salt+hash

Where algo is the algorithm identifier (e.g., 2a for bcrypt), cost is the number of rounds, and salt+hash is the concatenated salt and hash.

9) **Storing user variables**
    a) Our project hosts all of our webpages on a node.js webserver, because of this we needed to store data such as the user id, stock information, and the balances of the user as they move from page to page. Of course, we could continuously query our databases and API for the information again, but it would be computationally and monetarily inefficient since we were using the free version of FinnHub. We found a module that is built in javascript called SessinoStorage allowing you to store information locally on the user's browser allowing pages to access variables that was archived from another page

10) **Receiving Number values from API or DB queries**
    a) The project had to manipulate alot of numbers such as a user's account balance, stock price, and financial data, we believed it would be an easy task since sometimes you need to perform arithmetic operations on them and other times you just need to display them. It was not as simple as we originally thought, some scenarios would throw errors such as 'Cannot parse value' or 'Cannot perform expression on int'; from our research, it seemed that many if not all values that come in from the html or queries are not properly parsed and may be a type such that operations cannot be performed on them. To solve it we explicitly stated what type of value should be used by using parseInt() or parseFloat() and also force all decimal values to 2 decimal places to avoid confusion or better understanding.