2020

CAB230 Stocks API – Client Side

CAB230

Stocks API – Client Side Application

<John Nguyen /s>

<N10132767 /s>

5/13/2020

Contents

Introduction

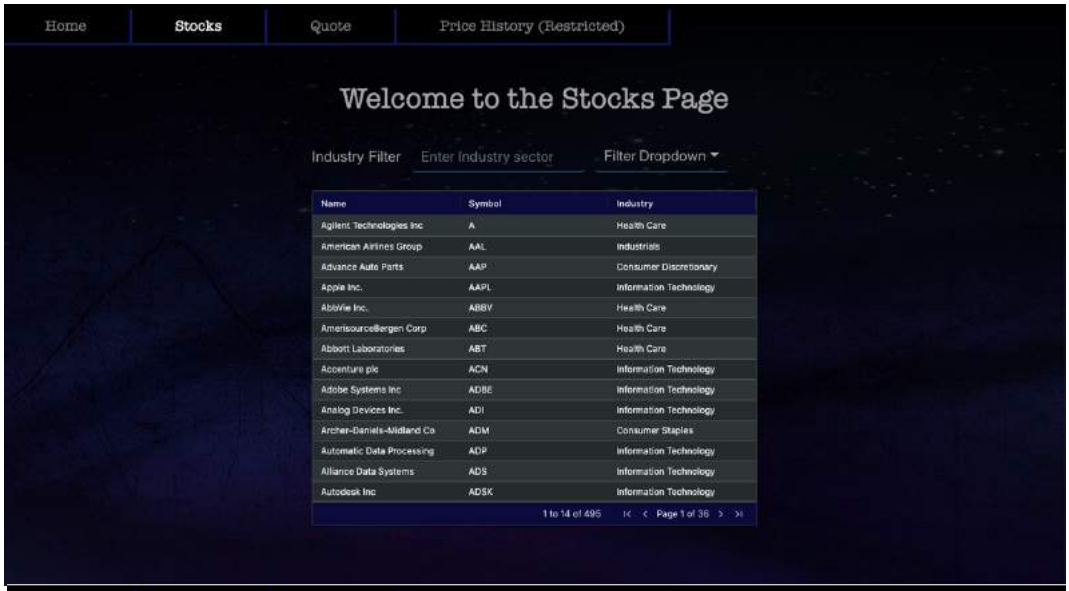
Purpose & description

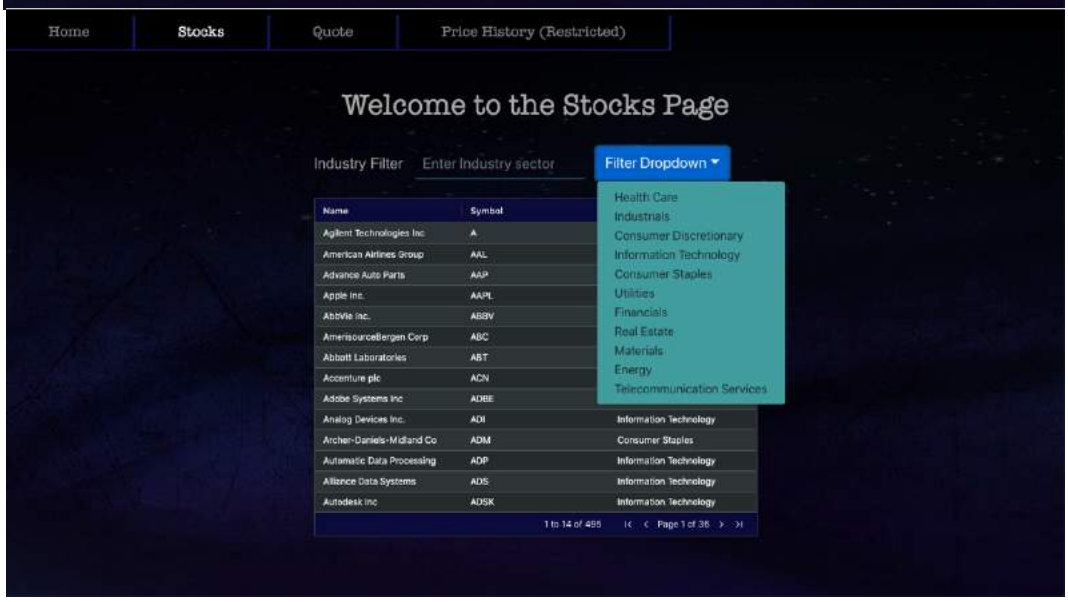
This assignment is a React-based web application which allows users to view and analyse stock market statistics drawn from a database that QUT have created for this purpose and exposed to us to via a REST API.

The aims of this assignment are to:

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

There is one specific functionality that would be considered a little unique. Regarding the querying the list of stocks and the stocks page, there includes a text search bar and a dropdown bar to filter the stock list by industry. These two functionalities are superfluous as the a-grid already has filter options for the industry sector. Although, the dropdown and search bar were included as it provides extra options to sort the stock list by industry sector.





Completeness and Limitations

- **[Grade of 4 level]:** A simple React app with limited styling which implements the unauthenticated Query endpoints and presents the data cleanly using table components. User endpoints and the authenticated queries may not have been implemented successfully and the client side processing in the table components is very limited. A react-strap table component or similar would suffice here.
- **[Grade of 5 level]:** At this level we would expect successful implementation of the User endpoints and the authenticated Query route. Table component usage and client side processing would be expected to use the standard functionality provided by a component such as ag-Grid-react, and there should not be excessive querying of the server.
- **[Grade of 6 and 7 level]:** Here the expectation is that you have exceeded the grade of 5 level in that all of the basics are there and working smoothly. Navigation is handled using React Router, React forms are used for the data entry and there is evidence of a really good match between your components and the services that they are using. We would expect some use of charting or other information graphics to show how the stock prices are varying and advanced use of the client side processing. The split between the grade of 6 and grade of 7 will involve a tradeoff between the features and the quality of the execution, and we will happily give you an opinion on your proposed application. For charting we recommend the use of chartJS (https://www.chartjs.org/), especially via the widely used React wrapper you can find here: https://www.npmjs.com/package/react-chartjs-2. d3 (https://d3js.org/) is a popular choice, but it is an advanced library and you shouldn't attempt it unless you have prior experience.

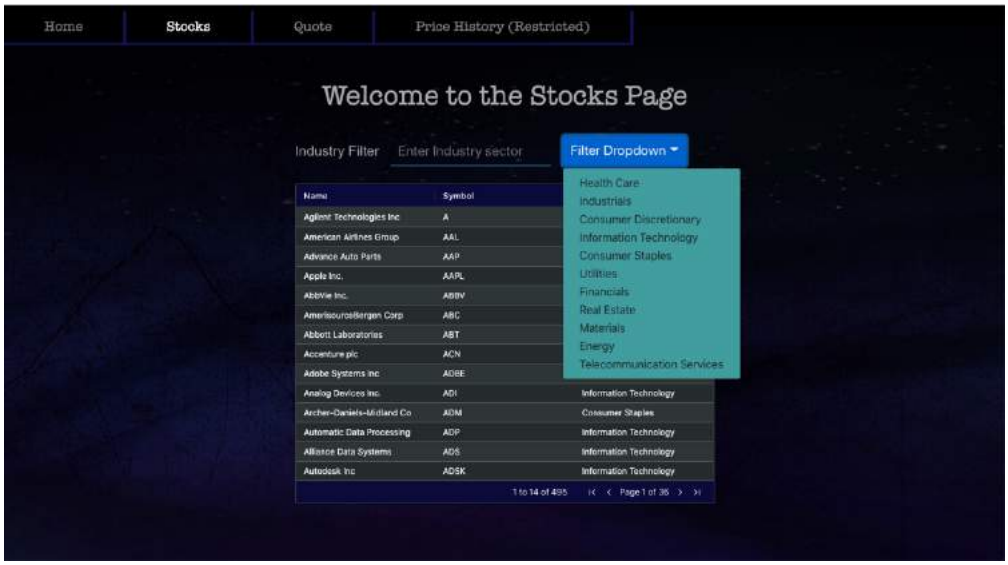
Claim: standards of this react application has met the criteria of Grade 6 and 7 level

The react application does have all the basic functionalities working smoothly. Navigation is handled using React Router, React forms have been used for filter search querying from the endpoints and user authentication. Charting has been included for the display of the closing price history over the last 100 days with a date query.

There are no limitations with the application.

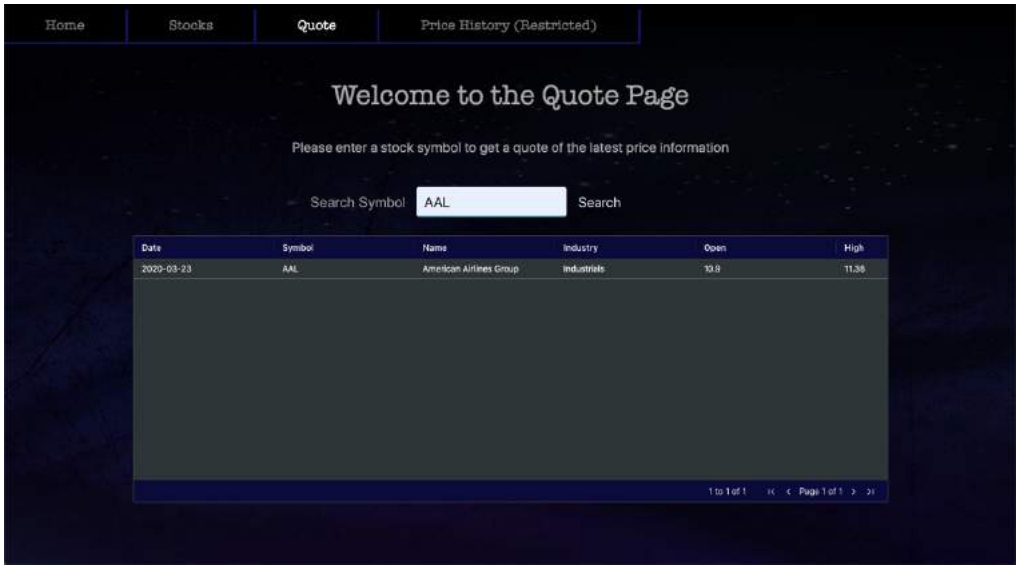
Use of End Points

/stocks/symbols



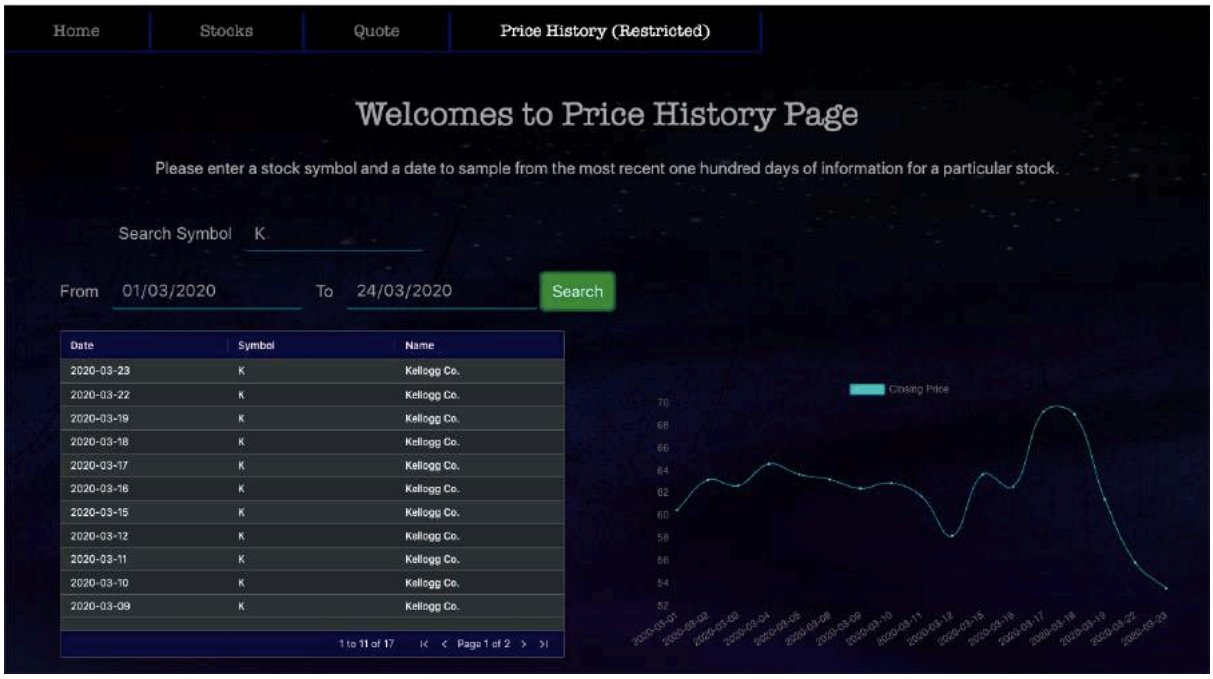
This is the Stocks page covers this section endpoint. Search bar form and a dropdown menu is provided so that the user can filter the list by industry sector.

/stocks/{symbol}



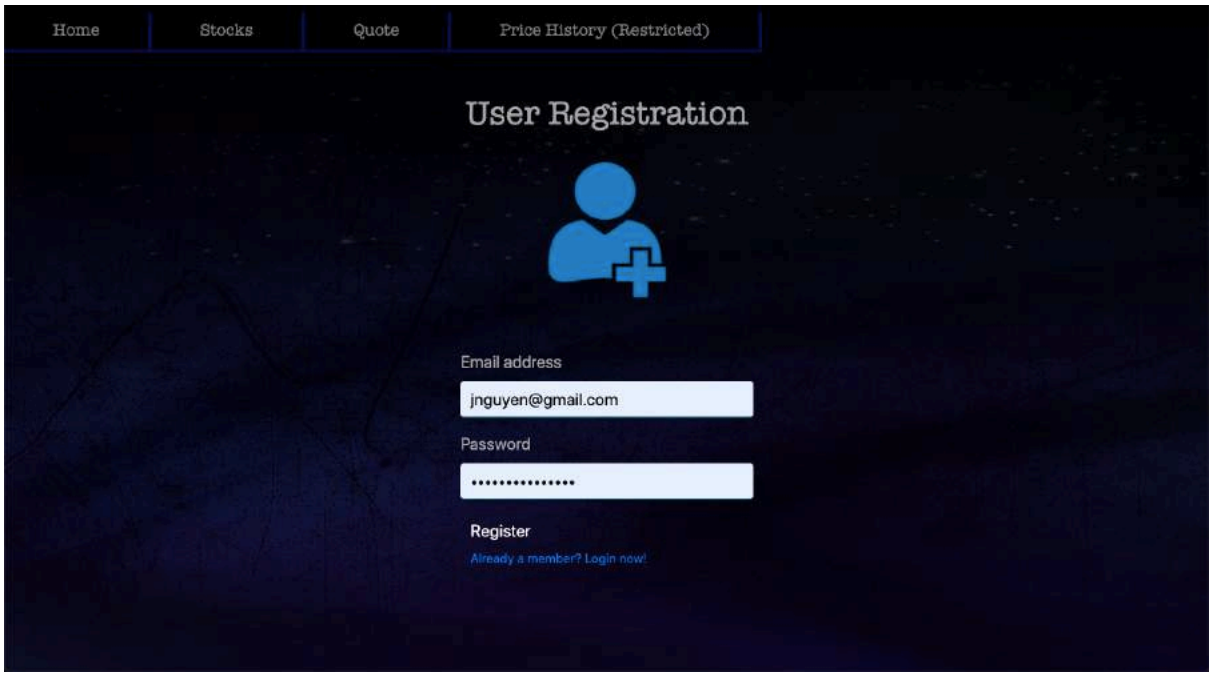
This is the Quote page covers this section endpoint. Search bar form is provided so that the user can get the latest entry for a particular stock, searched by symbol (1-5 characters)

/stocks/authed/{symbol}



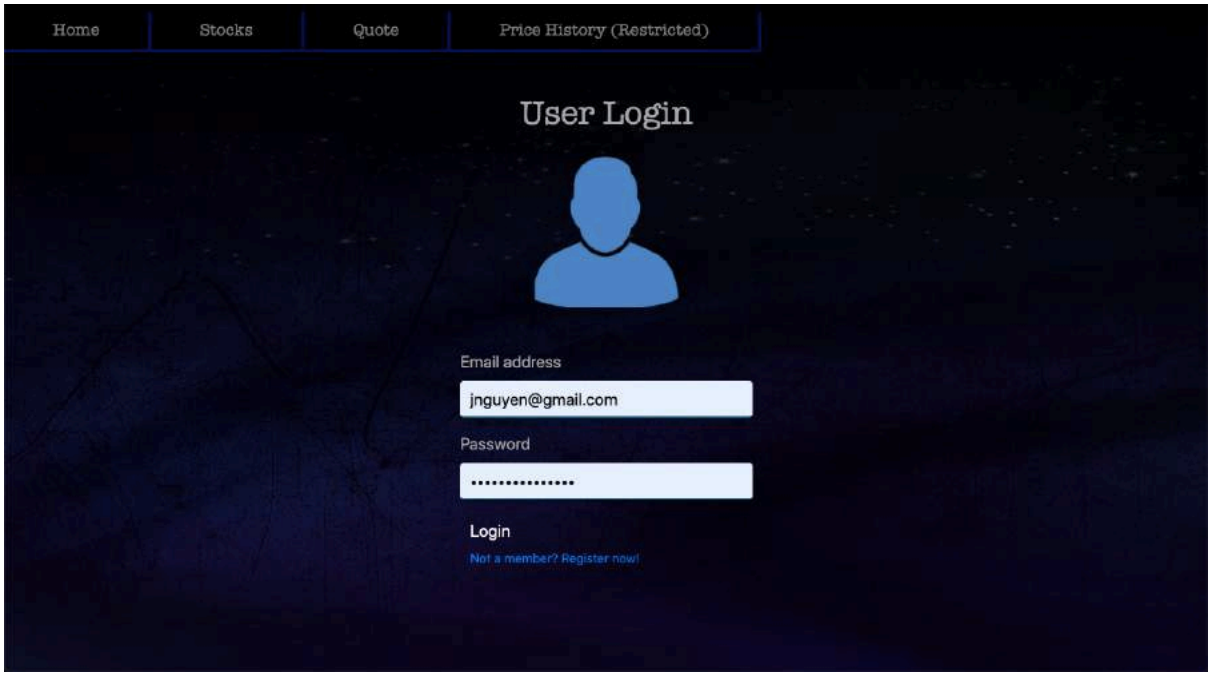
This is the Price-History page covers this section endpoint. Search bar forms including type text and date are provided so that users can get all the entries of the stock searched by symbol, optionally filtered by date. This page also requires user authentication to view contents of this page.

/user/register



This is the Register page covers this section endpoint. Email are provided so that users can register their account by entering an email and password.

/user/login



This is the Login page covers this section endpoint. Email are provided so that users can login their account by entering an email and password.

Modules used

Ag-grid-react

Module to provide fully-featured table components, including sorting and filtering.

https://www.ag-grid.com/react-grid/

React-bootstrap

Module to provide mainly for the navigation bar, forms, onclick buttons and the application CSS.

https://www.npmjs.com/package/react-bootstrap

Reactstrap

Module to provide the main dropdown menu features such as button and dropdown item etc.

https://www.npmjs.com/package/reactstrap

React-chartjs-2

Module to provide charting, namely a line graph for the closing price of the stocks.

https://www.npmjs.com/package/react-chartjs-2

React-hook-form

Module to handle the on submit process regarding user registration and login validation.

https://www.npmjs.com/package/react-hook-form

Axios

Module to handle the JWT authentication processes via Fetch POST for registration and login for the REST API. Reasoning and justification of the use of axios is listed in the appendices.

https://www.npmjs.com/package/axios

Application Design

Navigation and Layout

The design of the website has a dark midnight blue color theme which compliments the arid-Balham-dark theme for the graph. Alternatives were to use a brighter theme although a dark theme gives contrast to the vibrant colors of the graph, which is more appealing for users to view.

The navigation of the site relies mainly on a navigation bar with header links that take a user to a specific page that queries one of the endpoints from the API. Although there are important two navigation links not listed on the navigation bar, which are login and registration links. There's a login button on the main home page, that once clicked, redirects the user to the login page. User can then authenticate on the page or redirect to the registration page if they need to register.

Upon login authentication, the user is redirected to the home page where the login button is replaced with a logout button. Logging out removes authentication; thus, prevents access to the price history navigation link header.

Clicking the price history navigation link header without user authentication will reject access with an alert – subsequently, the user is redirected to the home page.

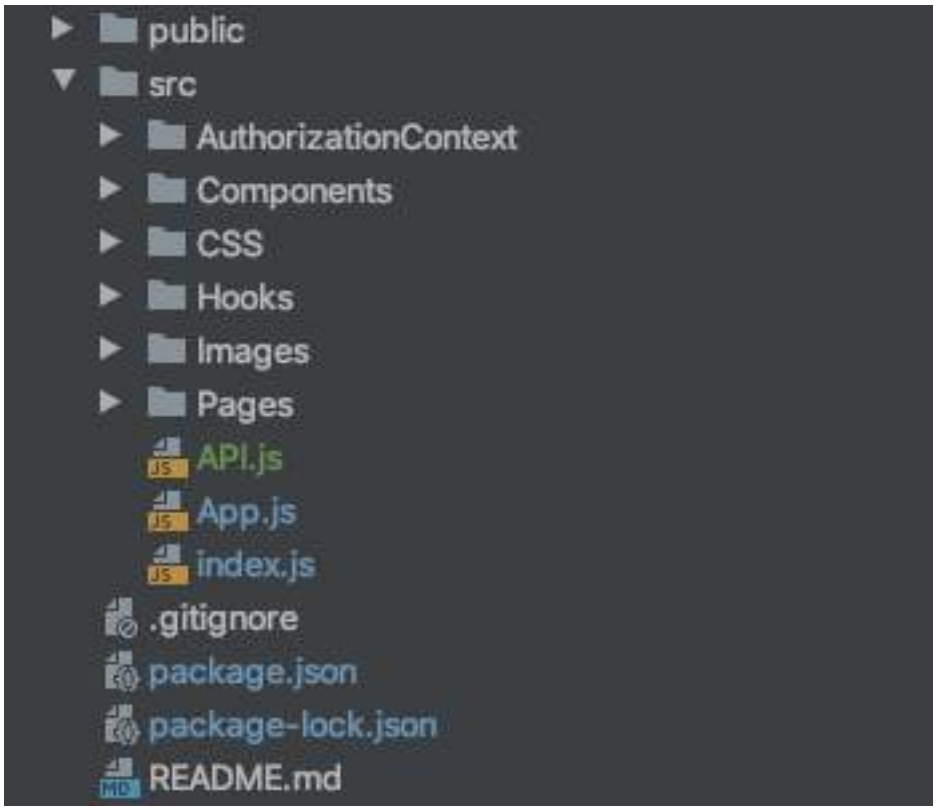
There was no critical compromising on the design. Although initially, the navigation links included a register and a login page, and the styling was difficult to float the two-header links to the far-right end of the screen without using margin styling. Margin styling worked but the design would look a little different on a different browser, and it was a little difficult to replace the header links with a logout link when a user authenticates. Even if it is possible to achieve a replacement, the logout link would need to redirect to a page – which is inconvenient as it would be more efficient to logout upon press. Instead, a login button is installed on the home page upon accessing the website. The login button redirects a user to a login page which includes a link to the registration page.

Technical Description

Architecture

The architecture of this application at source code level includes a main folder which is the application. Inside that folder includes a public and src folder. Other files include package.json and a readme.md

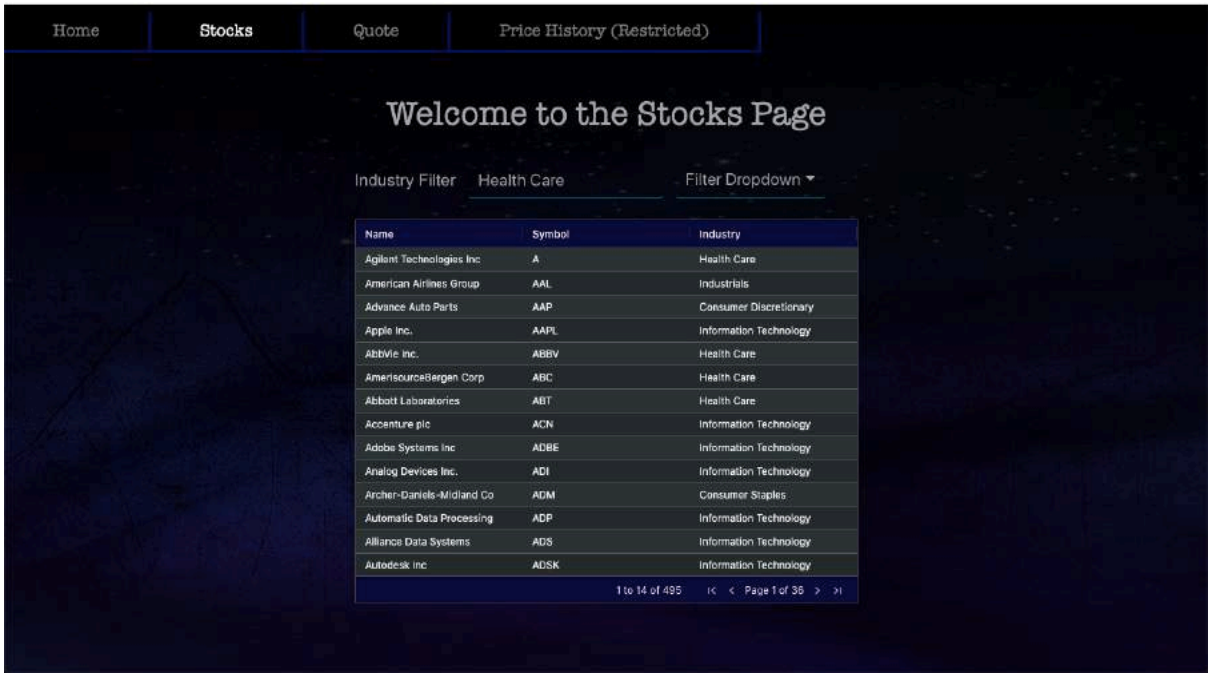
In the main src directory contains the main application files. The responsibilities of each section are allocated within folders. For example, the AuthorizationContext folder provides a main file which uses a context for main APP within index.js. This can provide information of the authentication process to do with the state of login, logout and registration. The components include the main components of app, such as forms and graphing. The CSS folder has the styling sheets for the components and pages. Hooks folder contains hooks for updating the state of the pages from querying an endpoint. Images folder contains images such as the hero image, login, logout and registration.



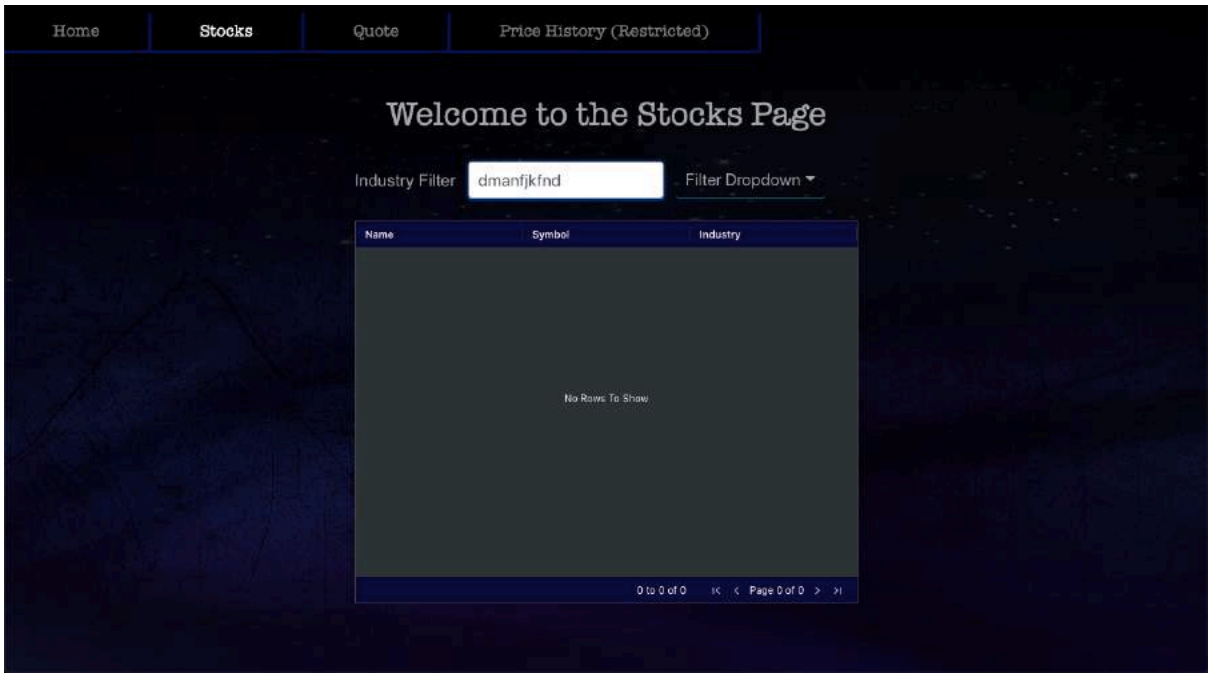
Test plan

/stocks/symbols endpoint

Positive case:

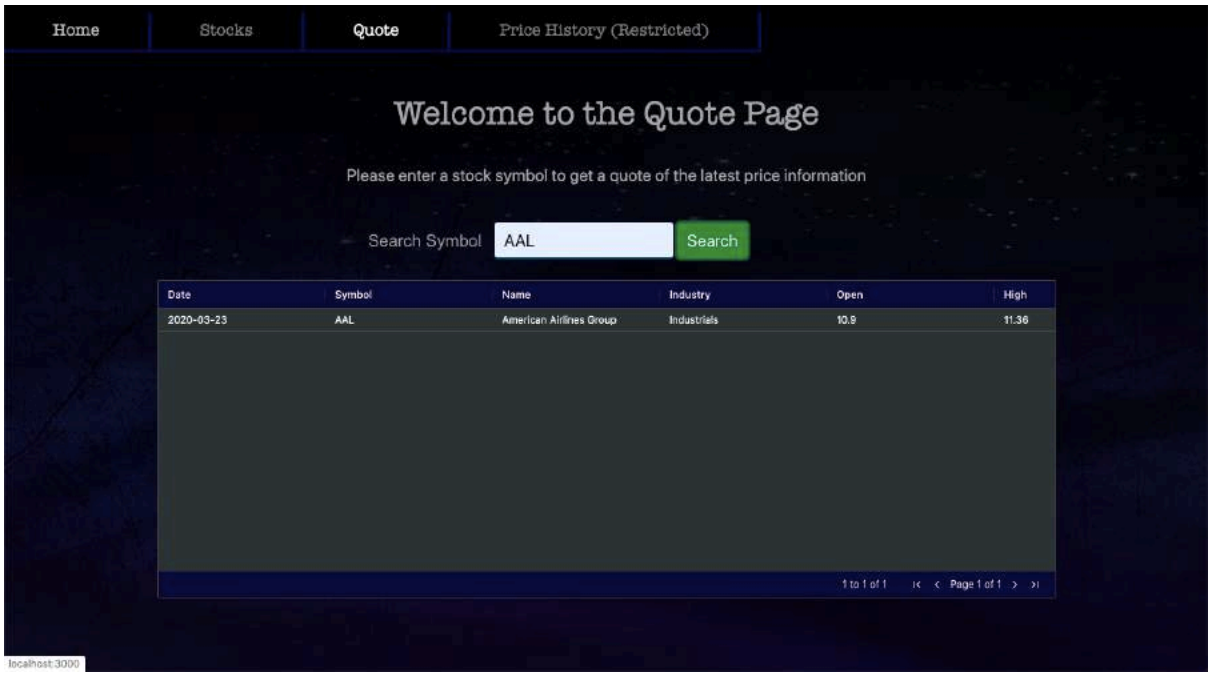


Negative case:

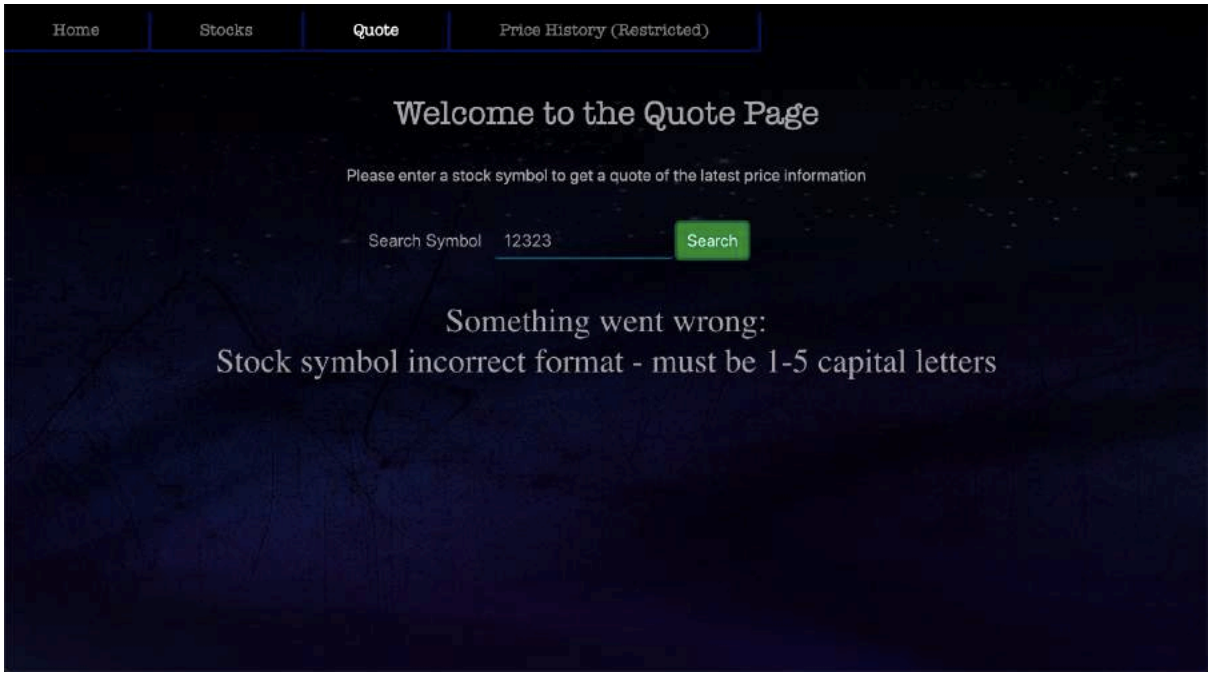


/stocks/{symbols} endpoint

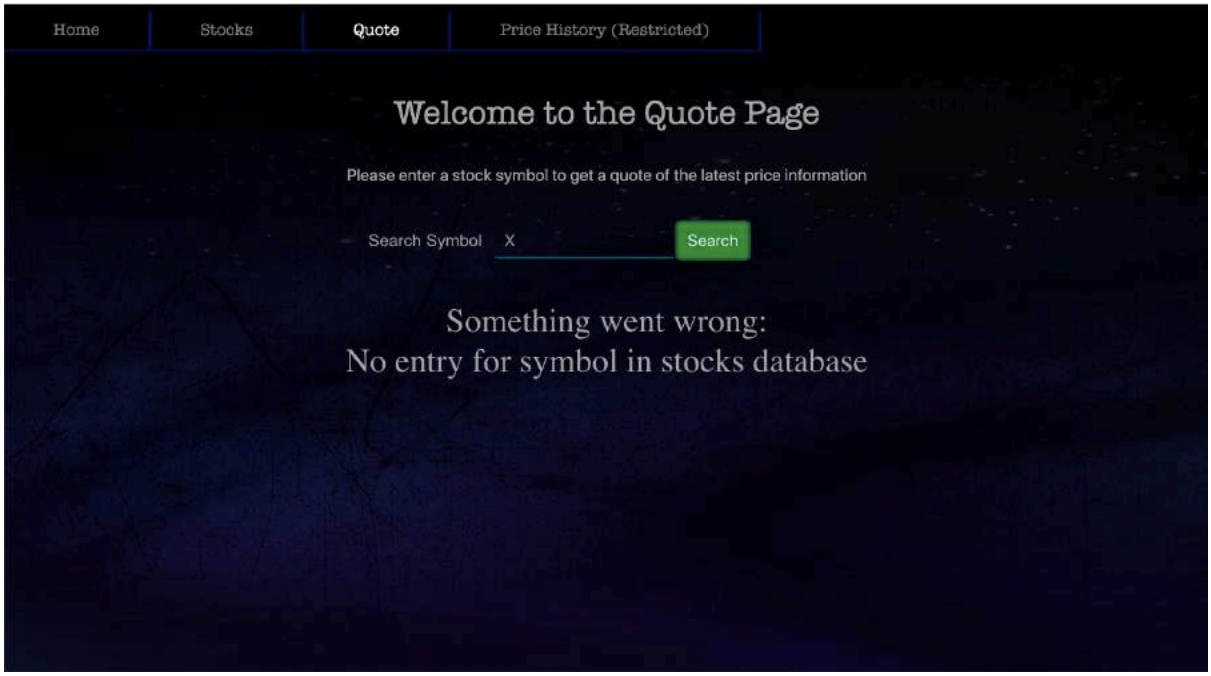
Positive case:



Negative case: When user enters an incorrect format/length of the stock symbol

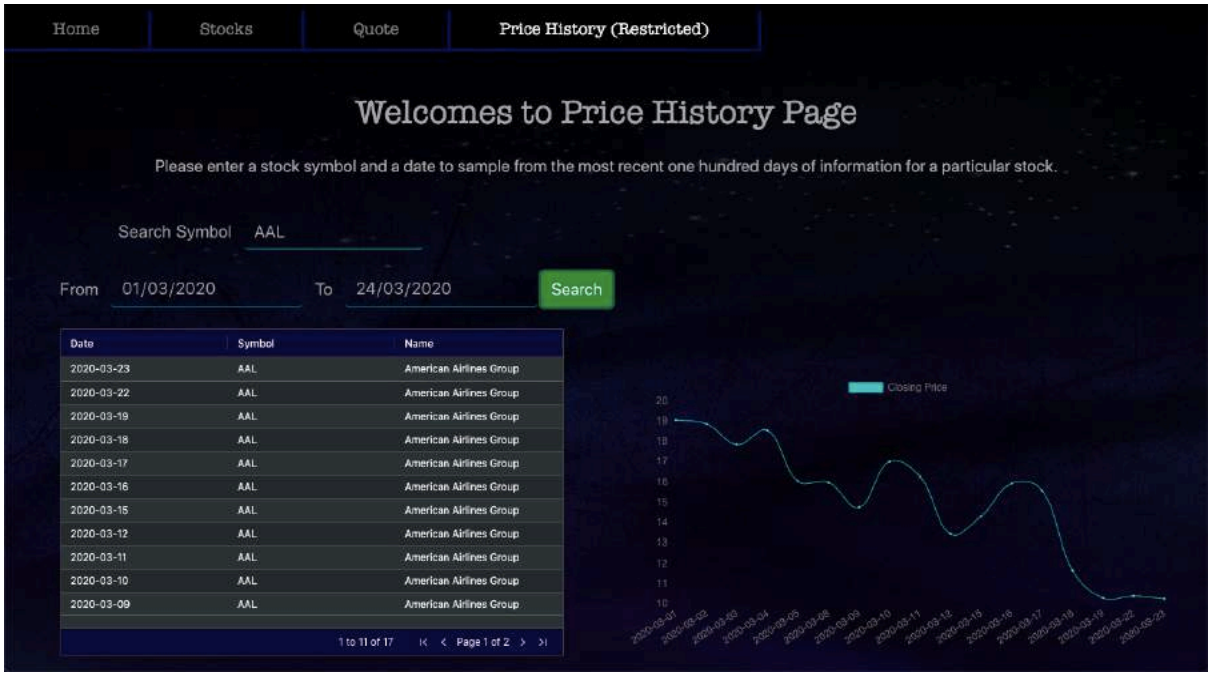


Negative case: When user presses incorrect symbol

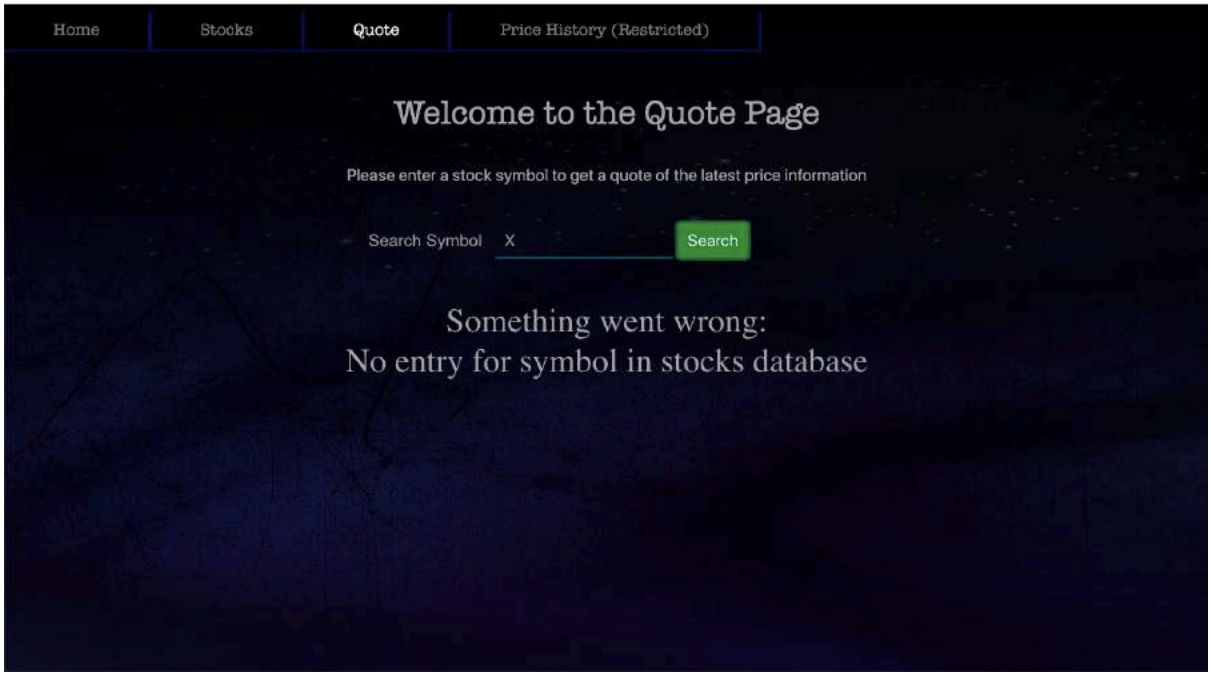


/stocks/authed/{symbols} endpoint

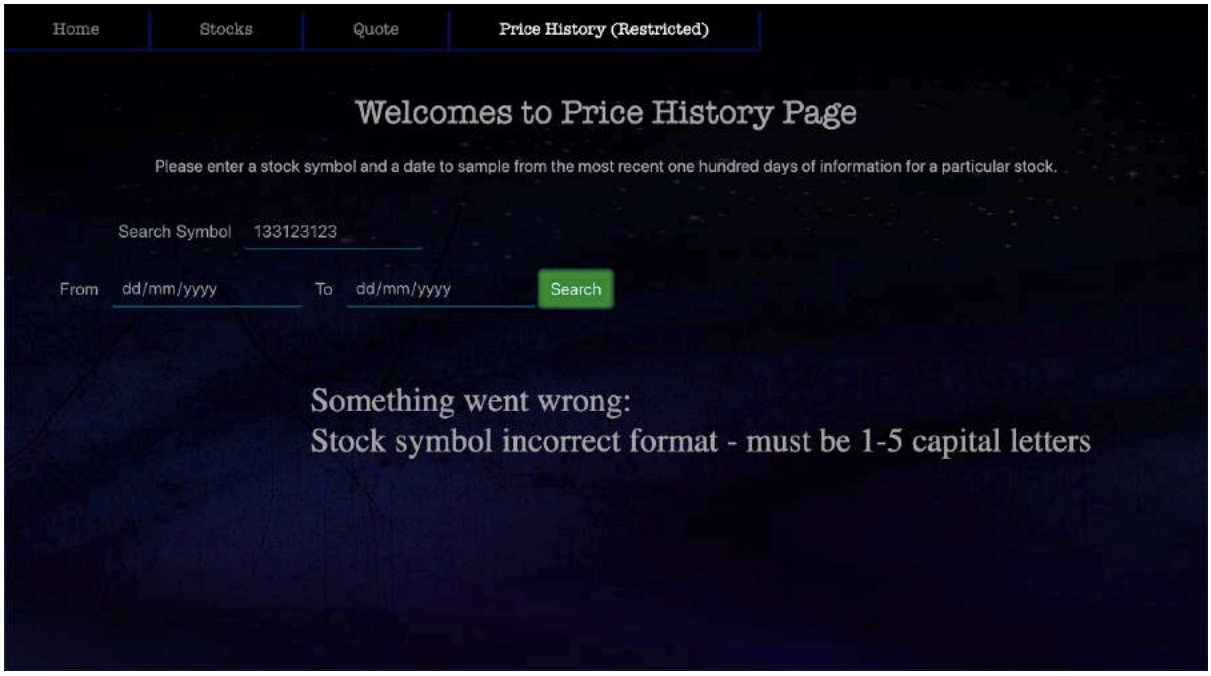
Positive case:



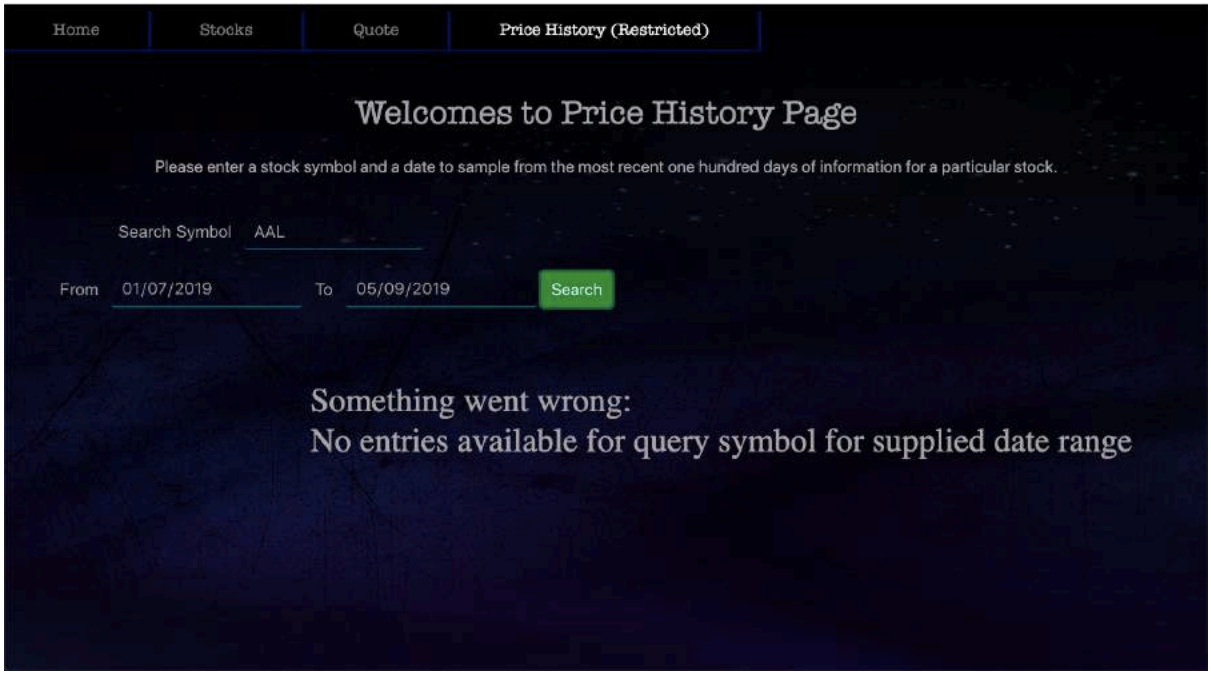
Negative case: When user presses incorrect symbol



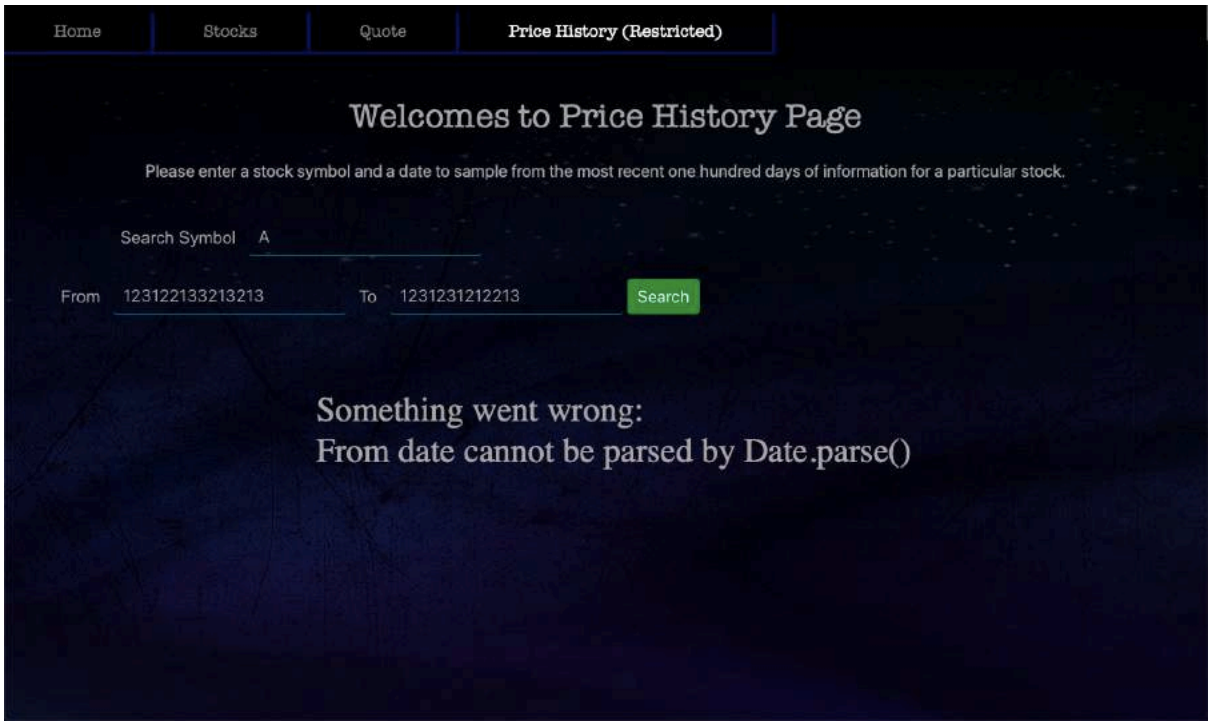
Negative case: When user enters an incorrect format/length of the stock symbol



Negative case: When user provides date and time query outside the scope of the stock dates

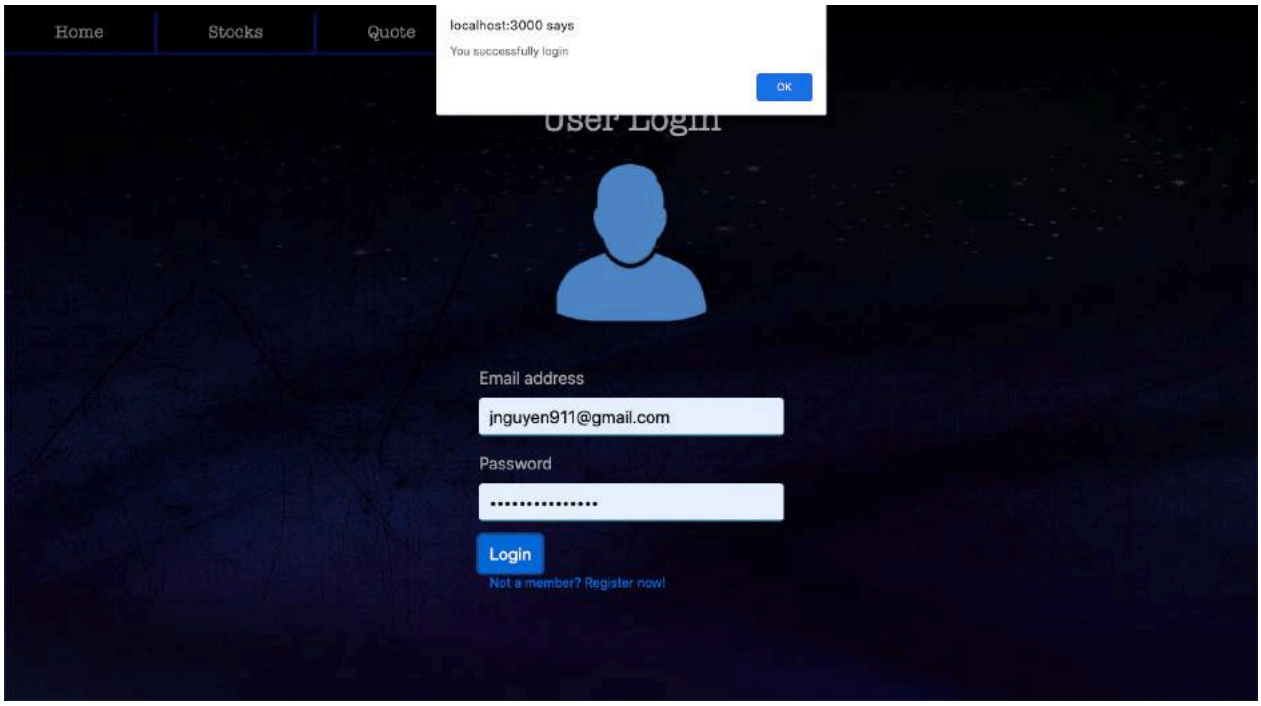


Edge case: Unlike chrome, on a safari browser the date forms don't provide a strict layout for inputting the date, nor does it provide a date picker. Thus, entering an incorrect date will result this:

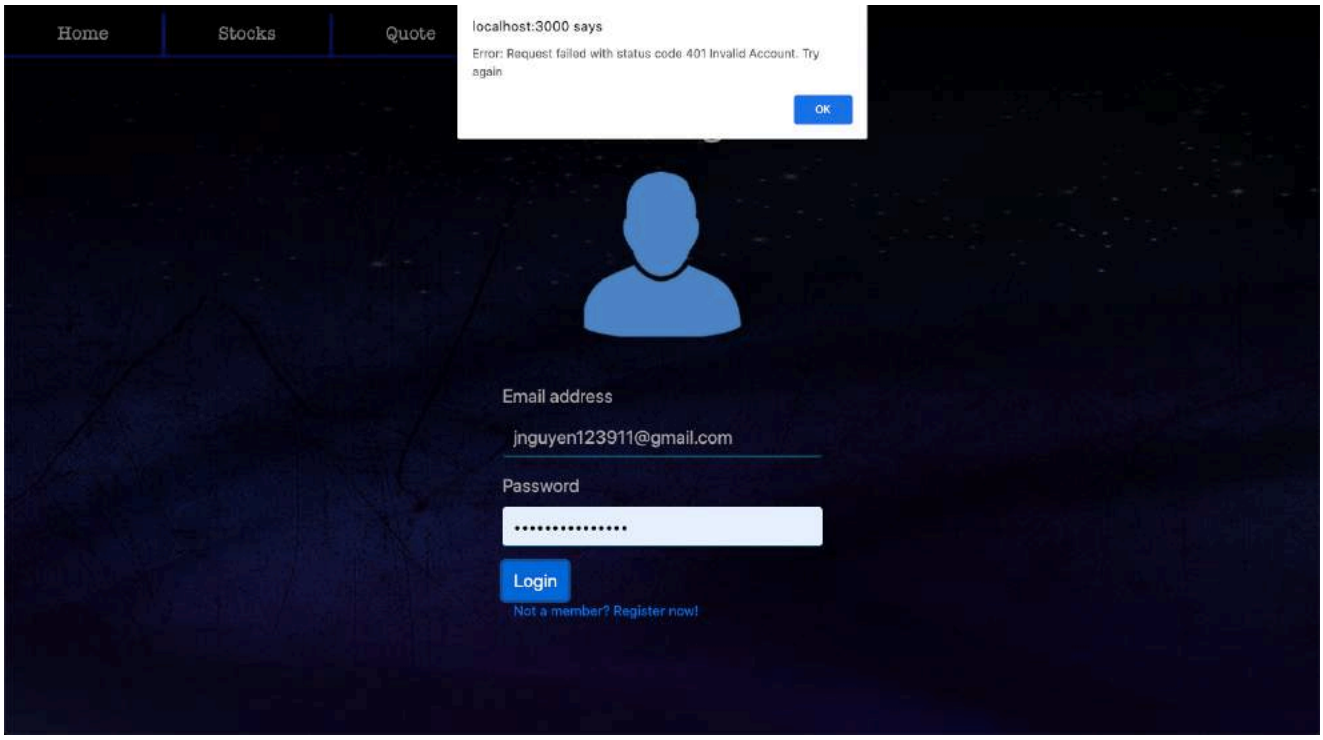


/user/login

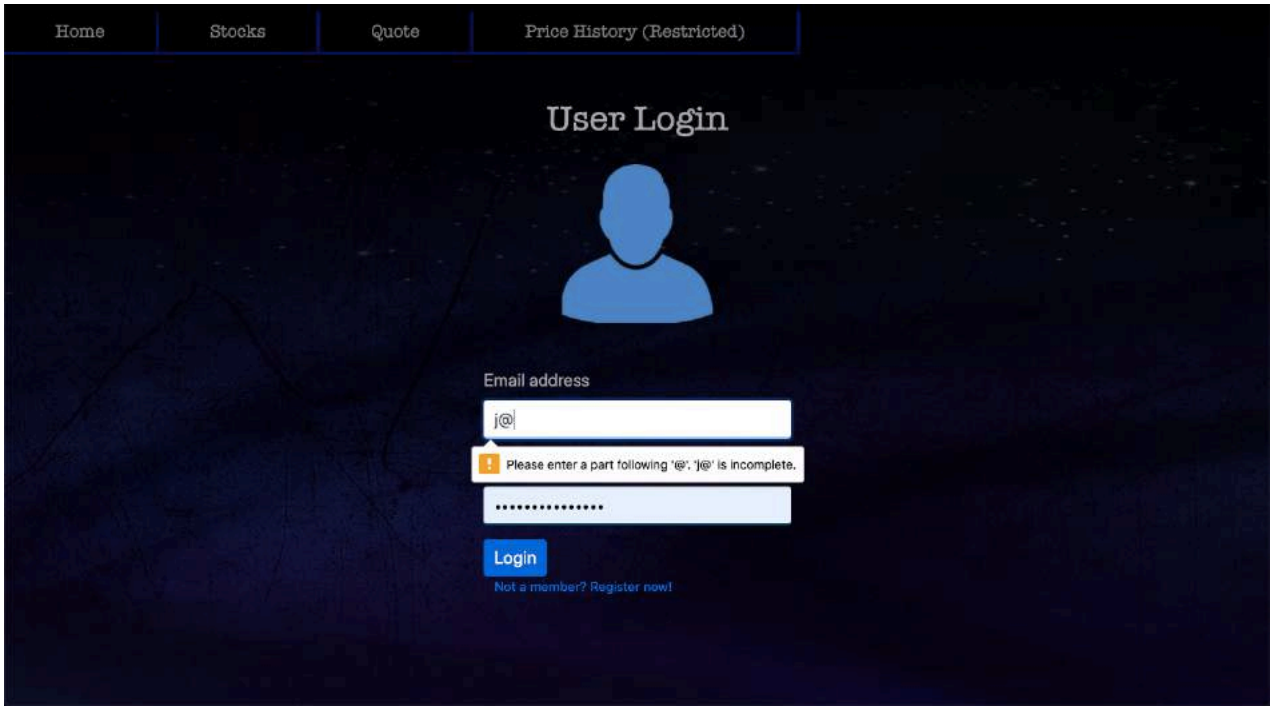
Positive case:



Negative case: when user types an invalid email account – does not exist or incorrect password

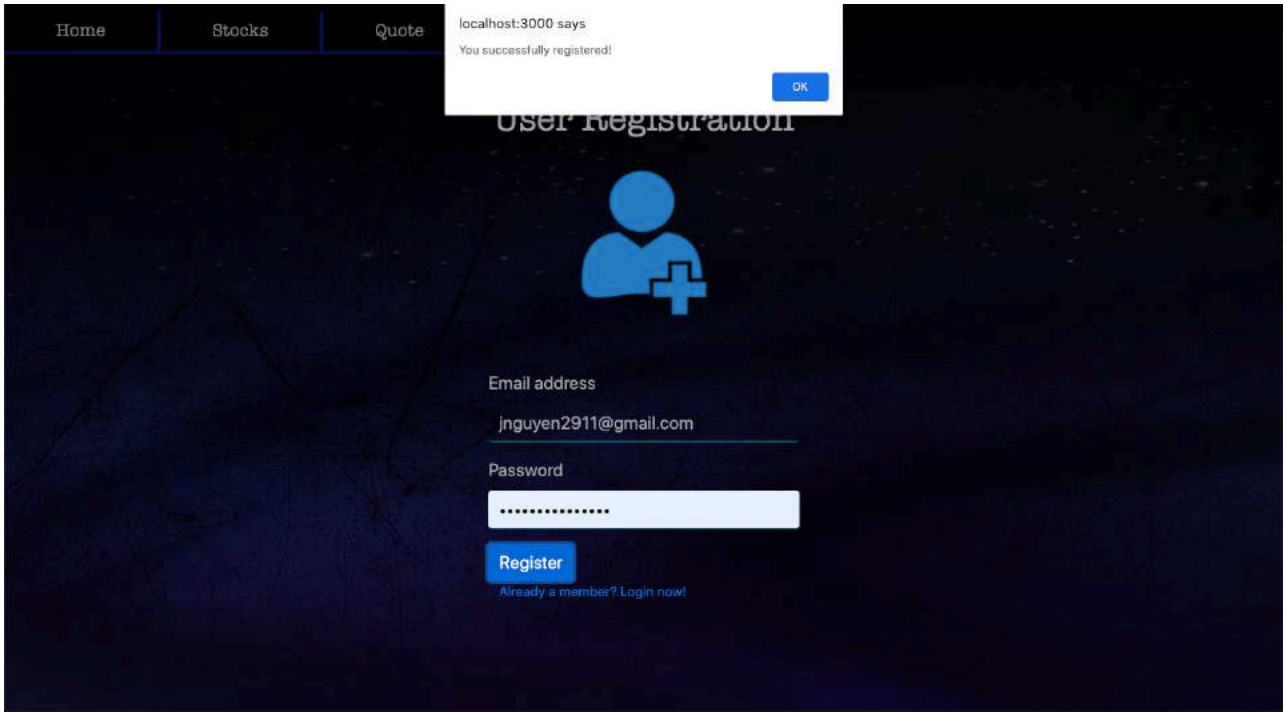


Negative case: when user enters incorrect format of email

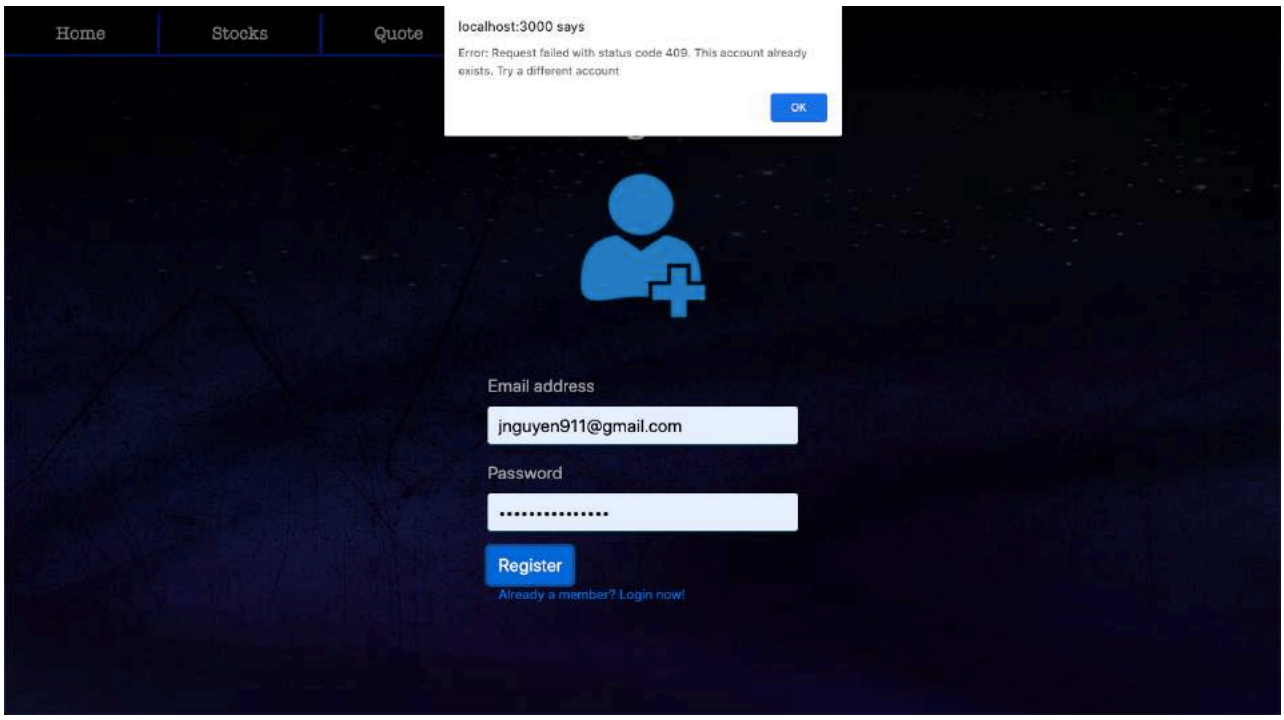


/user/register

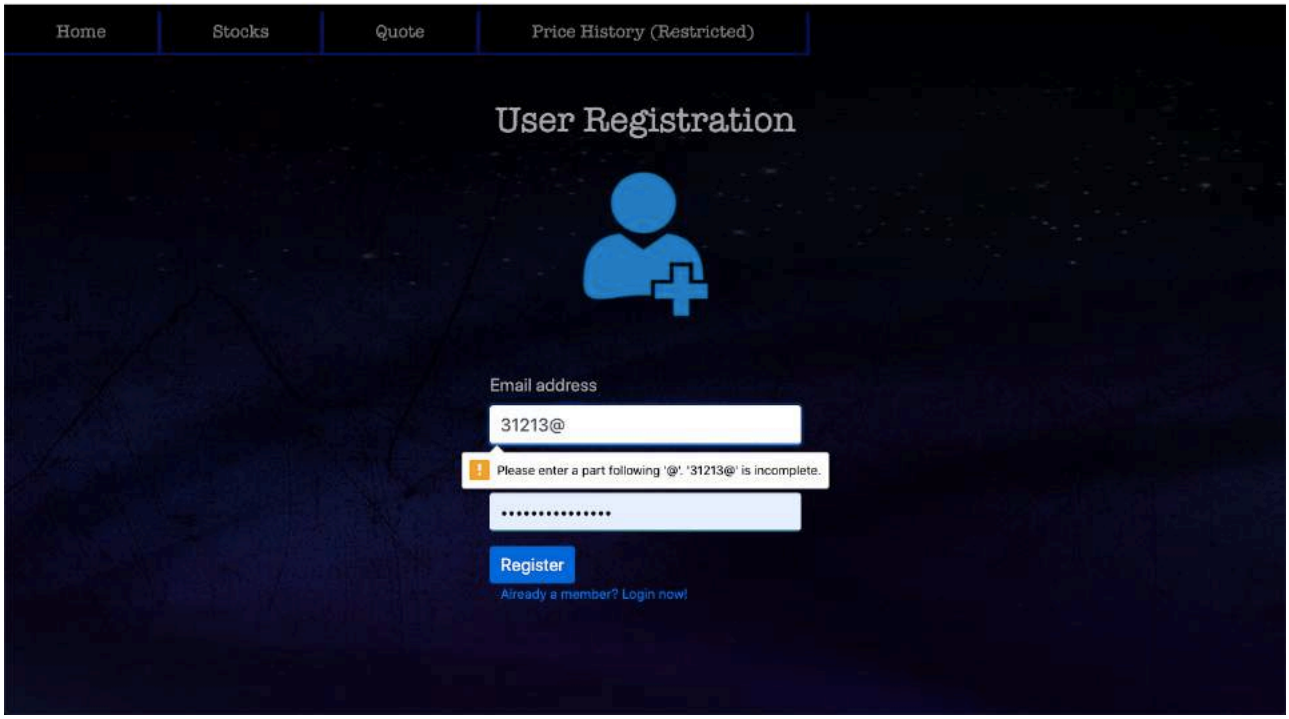
Positive case:



Negative case: When user enters an email that already exists



Negative case: When user enters an email by incorrect format



Difficulties / Exclusions / unresolved & persistent errors /

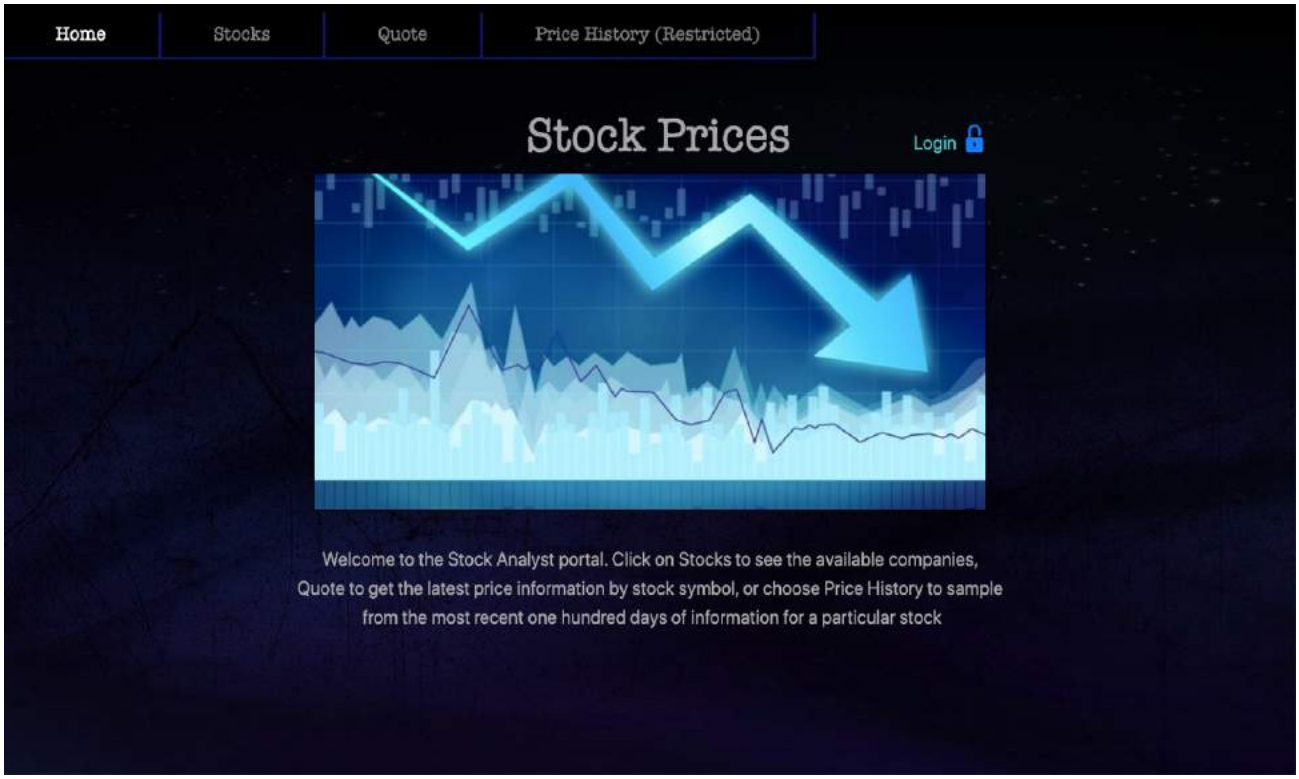
Some difficulties met were working with the JWT authentication forms, specifically for post querying for the user/login endpoint. When using the standard fetch Post method to convert the json data and returning the token value, I could not find a way to catch the error using standard fetch. See appendices for screenshots of the results.

The issue was resolved using axios with the async and await methodology. Using axios to catch using the error post query was well regarded and successful.

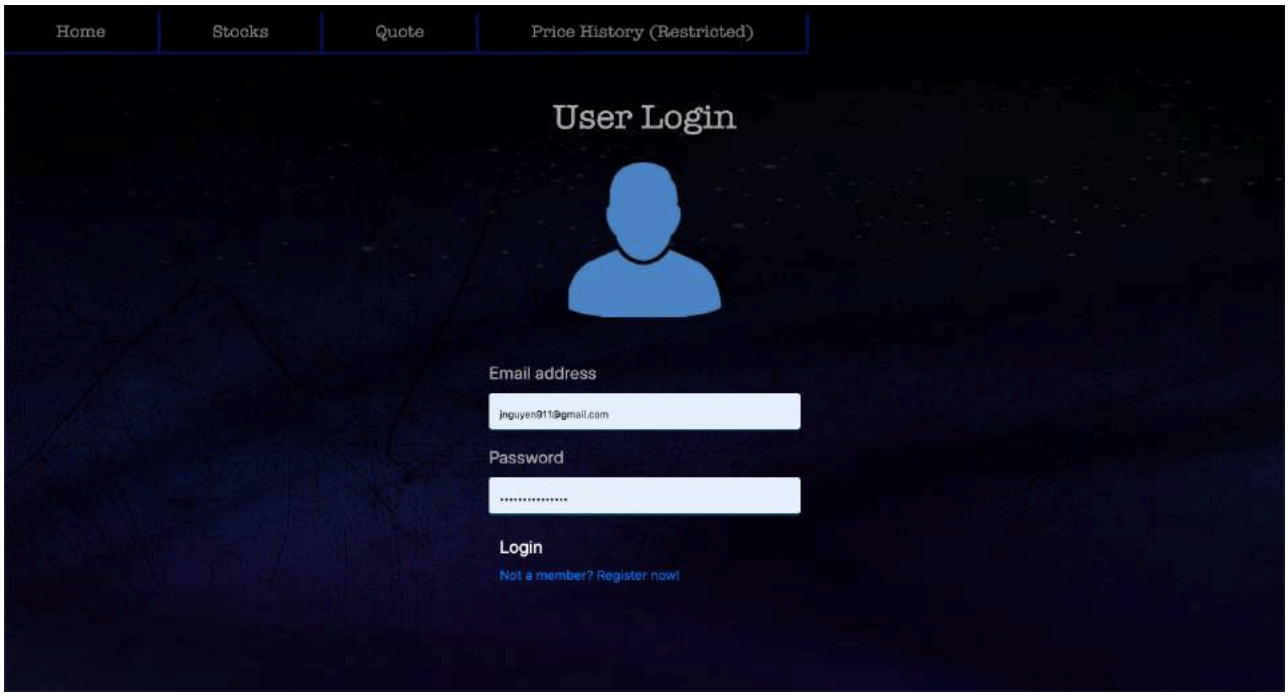
Most of the errors for this program other than the JWT forms have been quite trivial to be worthy of mention.

User guide

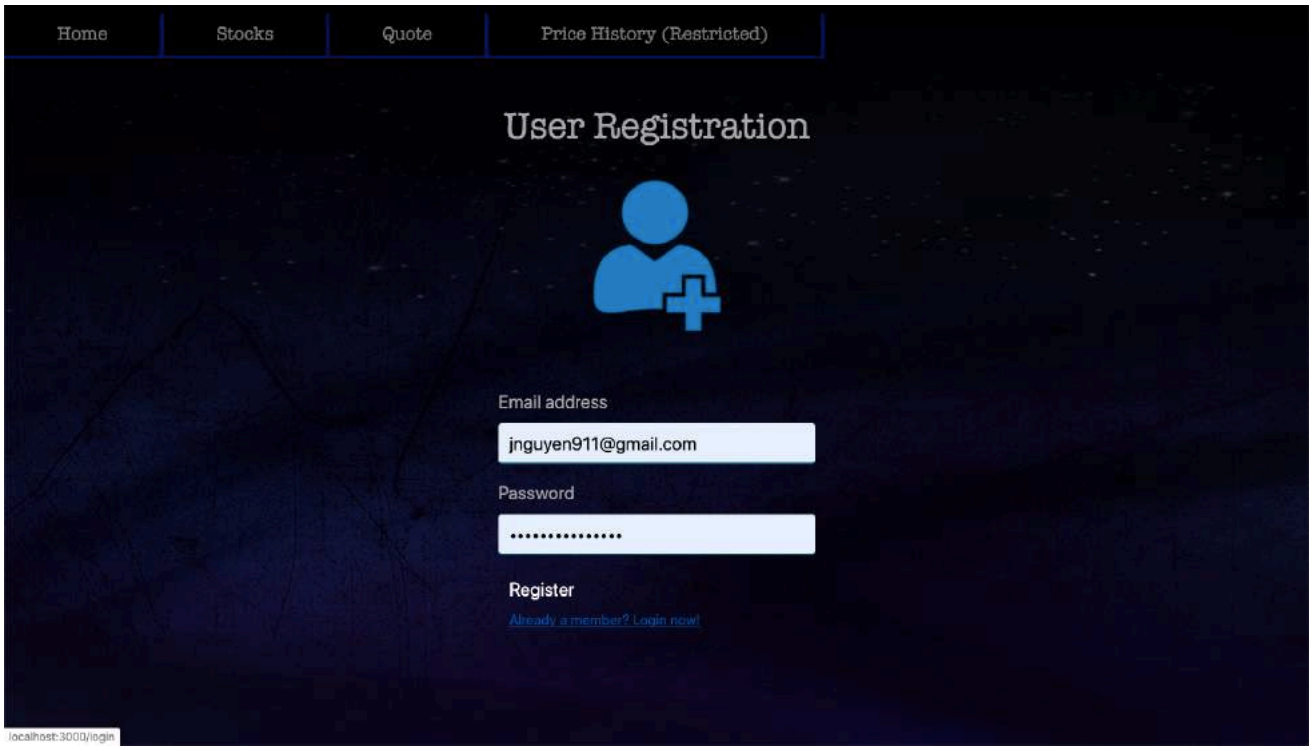
Upon visit, user lands onto the home page



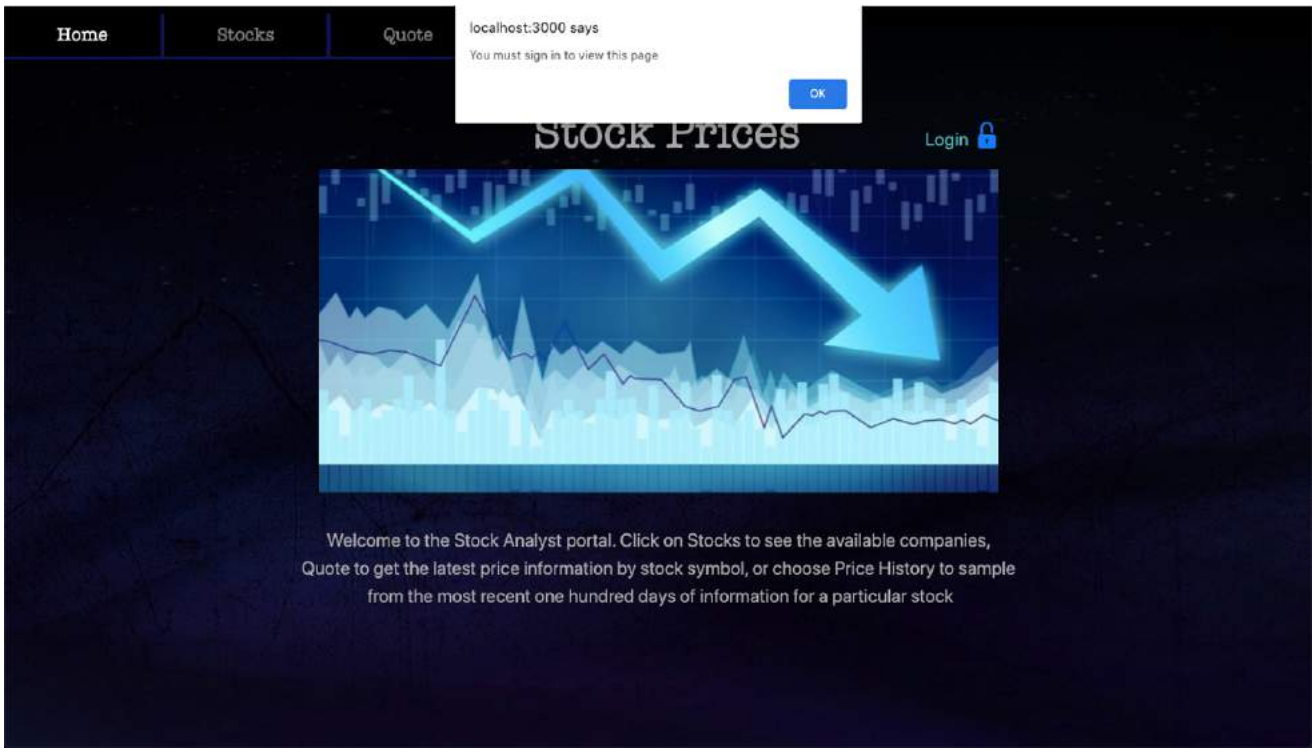
User can then press the login button to the login page



User can decide to register if their account is not registered, or they can login which they will be redirected back to the home page.

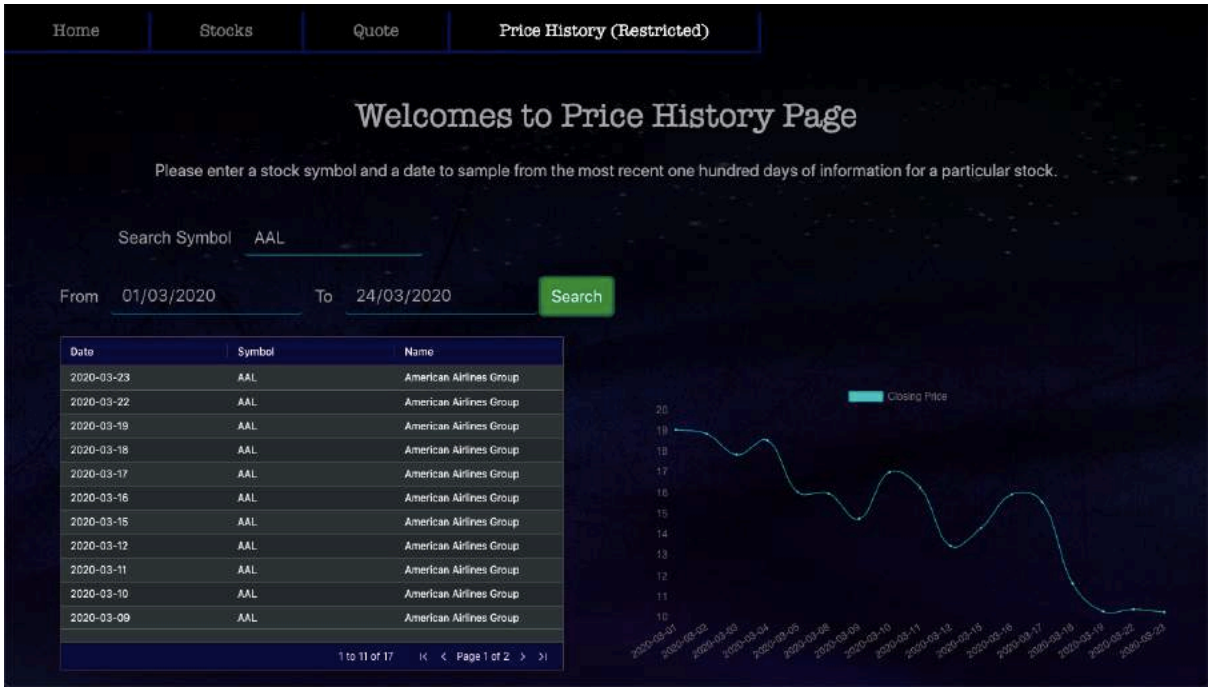


If the user is not logged in, they cannot access the price history page



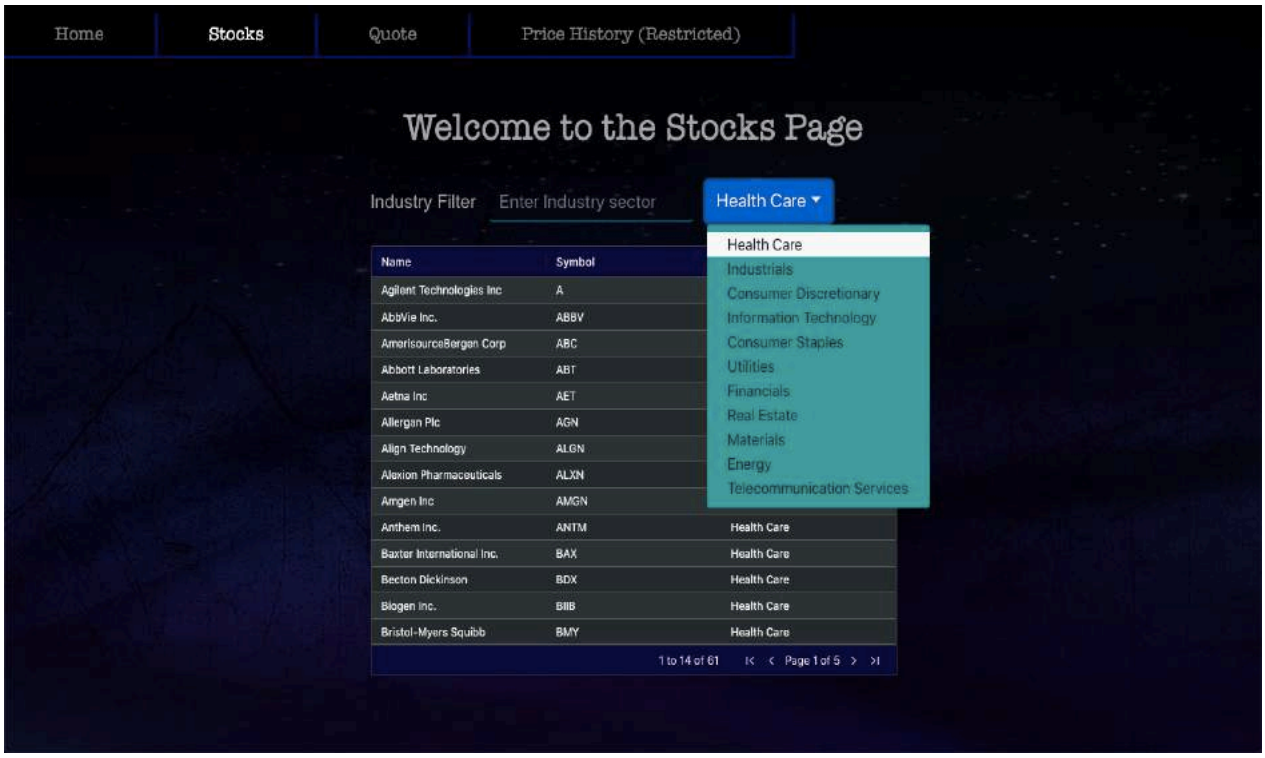
Otherwise if the user is logged in, they have access to the price history page.

User can make a query to display the price history of the stocks which includes a grid table and graph.

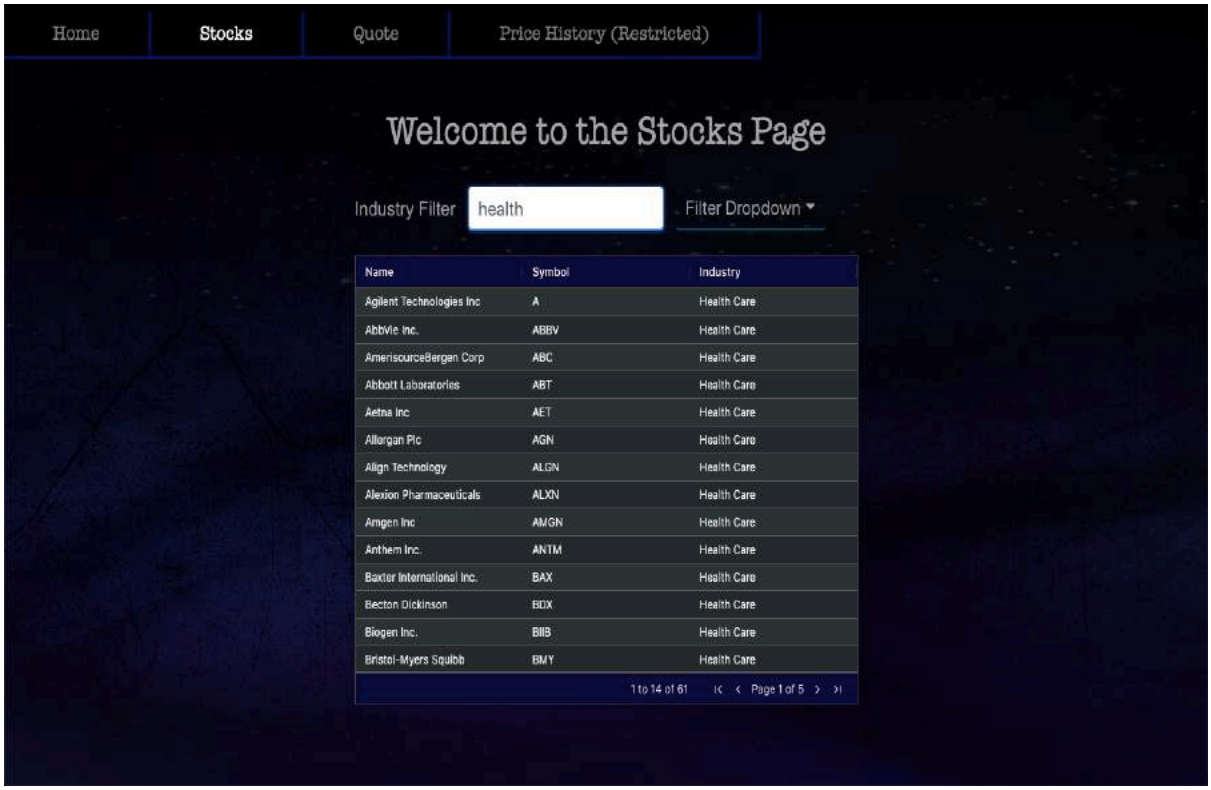


Clicking the stocks navigation link, user is redirected to the stocks page. User can then filter the stocks list by either inputting a search term through the search bar or using the dropdown button.

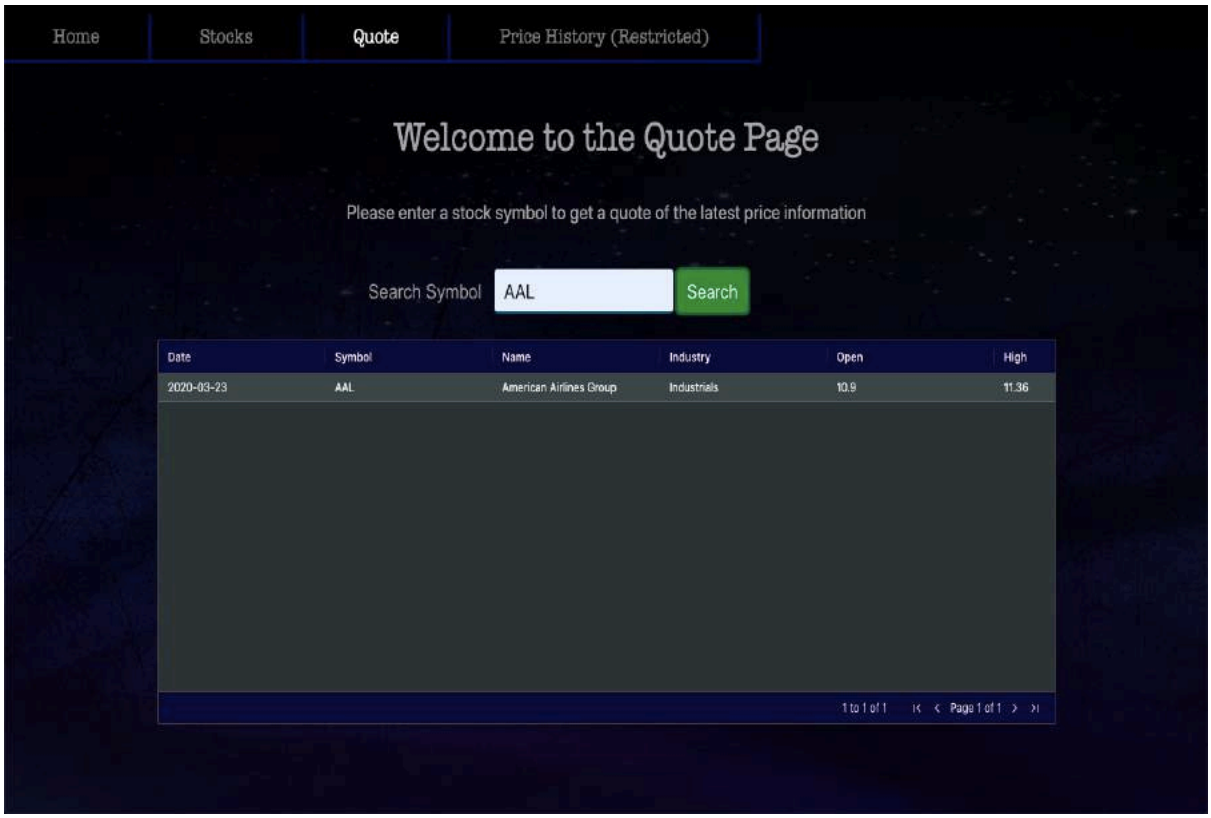
Filter by dropdown:



Filter by search bar:



When the user pressed on the quote navigation link, it should display a search form and a grid (a grid is displayed providing the user enters a valid search symbol



Appendices

JWT error catch problem using fetch: (resolved using Axios)

Using Fetch:

This particular fetch function fails to catch an error when post querying an invalid account to the API

```
//-----------------------------------POST-----------------------------------------------------------

// ValidateLogin uses Post method to login user through the API (retrieving token)
export const ValidateLogin = (email, password) => {
    const url = baseURL + '/user/login';

    // Using fetch to post user login email and password
    return (fetch(url, {
        method: 'POST',
        headers: { accept: "application/json", "Content-type":"application/json"},
        body: JSON.stringify({ email: email, password: password })
    })
    .then((res) => res.json())
    .then((res) => res.token)
    )
};
```

Using Axios:

This axios function successfully catches an error when post querying an invalid account to the API.

```
//-----------------------------------POST-----------------------------------------------------------

// ValidateLogin uses Axios Post method to login user through the API (retrieving toke
export const ValidateLogin = async (email, password) => {

    const MessageResponse = await axios({
        method: 'POST',
        baseURL,
        url: '/user/login',
        data: {
            email,
            password
        }
    });

    return MessageResponse.data.token;
};
```

Here is the hook which is used for the function mentioned above. I have implemented a state variable for the token and error value, but let's focus our attention to the error state, which is returned from this hook. If an error is caught, then we can assume the value of error is not null and changes to the error value. However, if the value remains null, then in this context we will assume that error has failed to be caught by the hook – since our query is set up to fail by setting an invalid account.

Notice that the values passed through to the ValidateLogin function are an unregistered email and password. So, the post query to the API endpoint should guarantee a fail.

```
import {ValidateLogin} from "../API";

function useLogin(){
    const [token, setToken] = useState([]);
    const [error, setError] = useState(null);

    const unregisteredEmail = "jnguyen1236699@gmail.com";
    const unregisteredPassword = "chucksoyandr911";
    useEffect(() => {
        ValidateLogin(unregisteredEmail, unregisteredPassword)
            .then((tokenValue) => {
                setToken(tokenValue)
            })
            .catch((err) => {
                setError(err)
            });
    }, []);
    return {
        token,
        error
    }
}
```

To access the error variable, we simply call the hook and grab it. Then we can print to console.

```
const { error } = useLogin()

console.log(error + "",)
```

Regarding the Fetch function, when observing the console IDE in browser developer tools, it is unsuccessful in catching the state of the error:

```
[HMR] Waiting for update signal from WDS...                    log.js:24
                                        react-dom.development.js:24994
Download the React DevTools for a better development experience: https://fb.
me/react-devtools
null                                                      HomePage.js:36
▶POST http://131.181.190.87:3000/user/login 401 (Unauthorized)    API.js:11
null                                                      HomePage.js:36
>
```

First the value of the error state is null, then after the post query to the login endpoint failed, it is still null.

Therefore, the error state could not be caught.

Regarding the Axios function, when observing the console IDE in browser developer tools, it is successful in catching the state of the error:

```
[HMR] Waiting for update signal from WDS...                    log.js:24
                                        react-dom.development.js:24994
Download the React DevTools for a better development experience: https://fb.
me/react-devtools
null                                                      HomePage.js:36
▶POST http://131.181.190.87:3000/user/login 401 (Unauthorized)    xhr.js:178
Error: Request failed with status code 401                HomePage.js:36
>
```

First the value of the error state null, then after the post query to the login endpoint failed, the value is changed

Therefore, the error state has been successfully caught.