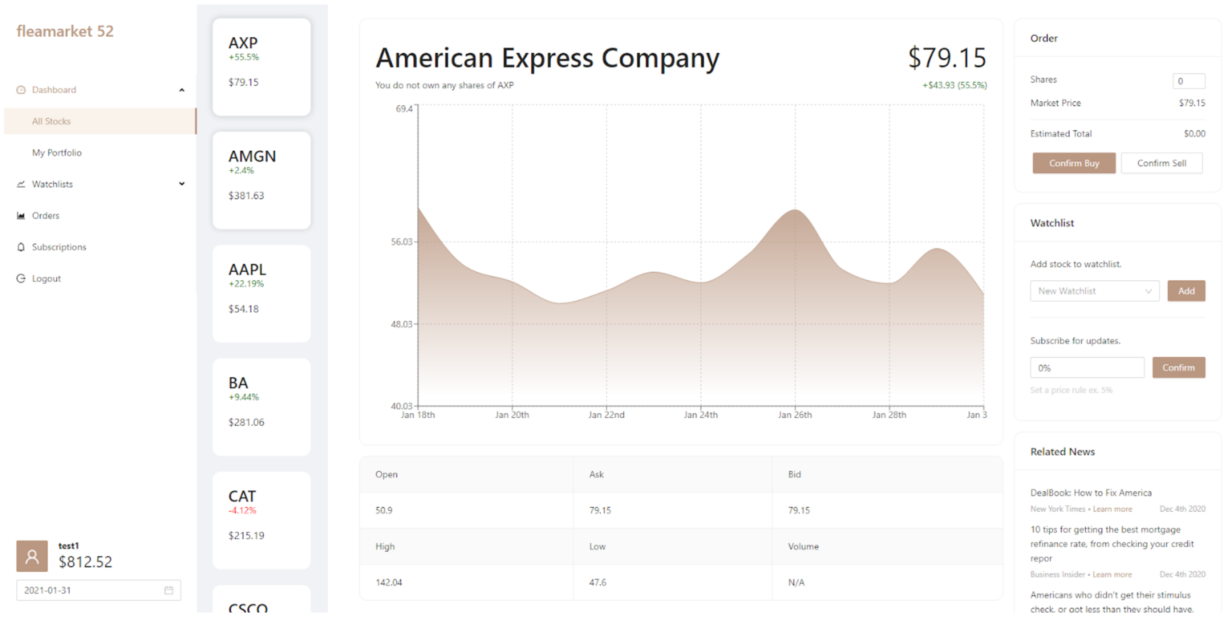
COMP 2406 | Shrish Mohapatra

FLEAMARKET 52
Stock Portfolio Application



Open	Ask	Bid
50.9	79.15	79.15
High	Low	Volume
142.04	47.6	N/A

DealBook: How to Fix America
New York Times • Learn more Dec 4th 2020

10 tips for getting the best mortgage refinance rate, from checking your credit repor
Business Insider • Learn more Dec 4th 2020

Americans who didn't get their stimulus check, or got less than they should have,

Introduction

Fleamarket 52 is a web based stock portfolio application designed to simulate stock market activity. Users can view, purchase, and sell stock shares using procedurally generated stock data. The application also allows users to create watchlists and subscribe to stock price changes. Few extensions include MongoDB integration, GraphQL API implementation, stylized interface with React, and a real-world News article feature. This report will discuss running the application, project features, technical details, and overall reflection.

Running Application

The following describes how to run the application on the OpenStack instance.

1. Connect to instance with: `ssh student@134.117.133.178`
   a. Password is `Fle@@20!?`
   b. Instance name is `ShrishMohapatra`
2. Run the following commands:

   ```
   cd src
   npm run cleanup      // deletes all collections
   npm run setup        // creates initial stock data
   npm start -- 3000    // server runs on port 3000
   ```

   Note that the setup command can take up to 10 seconds as the script fetches articles from the web for the 30 stocks stored in the database.

3. In another terminal create a tunnel with:

   ```
   ssh -L 9999:localhost:3000 student@134.117.133.178
   ```

4. Visit http://localhost:9999/ to view the application.

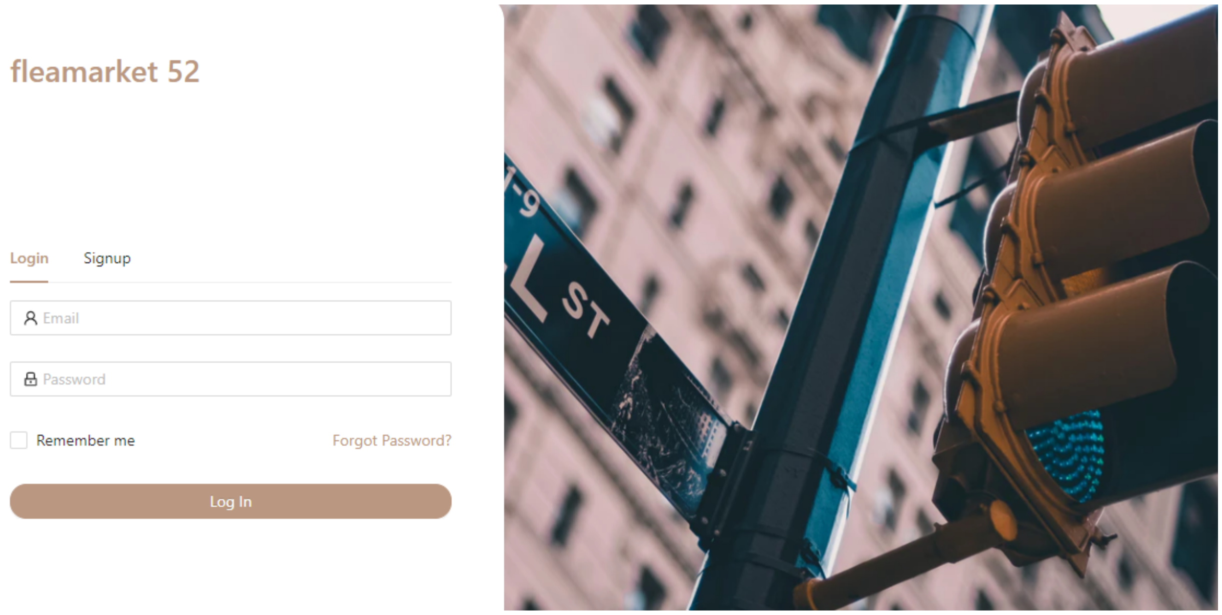
The node.js server has already been configured to serve the React client so there are no additional requirements to run the development server.

Features

This section will provide an overview of the various features that the application comes with. It will provide step by step guides and screenshots on how each feature works.

Authentication

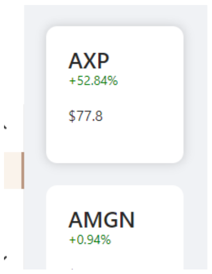
Fleamarket 52 comes with a secure authentication system. By default, when navigating to the app for the first time users will be prompted with the following screen:



There is currently an account made for testing however to before signing in one can test the authentication system through these various cases:

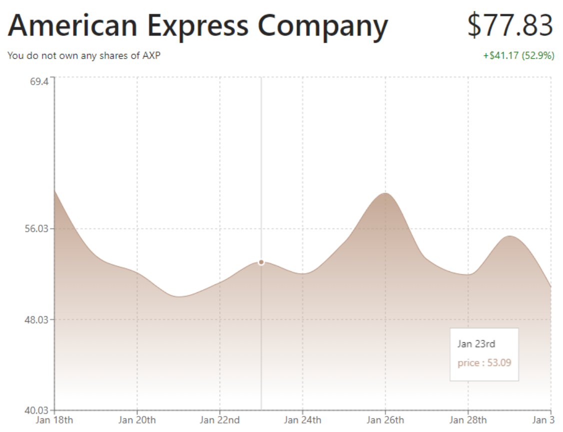
Case	Expected Response
Login with random credentials	Auth Error: User does not exists
Login with ‘test1’ but a random password	Auth Error: Incorrect password
Signup with ‘test1’	Auth Error: User already exists
Login with ‘test1’ and ‘password’	Redirects to home page
Signup with random credentials	Redirects to home page

Viewing Stocks

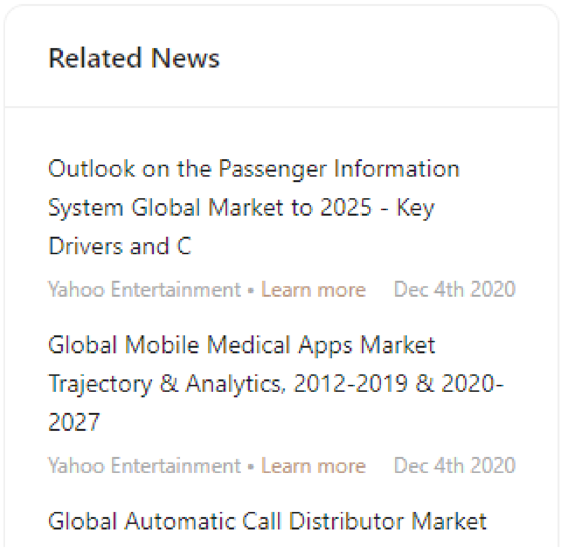


When a user first signs in they will be redirected to a dashboard page where they can start viewing stock data. There is a scrollable list view next to the navigation menu containing cards for all the stocks available. They are sorted alphabetically by ticker and contain key information such as price change % and current market price. Users can also search by the stock ticker.

Selecting a card will allow users to view more information regarding the stock's performance. The main source of data is the stock price graph. It displays the last 2 weeks of data in a stylish line graph that provides users context regarding the market price. Hovering over the graph will allow users to view the market price at a given time.



Key stock price information is also available such as open, ask, bid, high, low, and volume.



Related News

Outlook on the Passenger Information System Global Market to 2025 - Key Drivers and C

Yahoo Entertainment • Learn more Dec 4th 2020

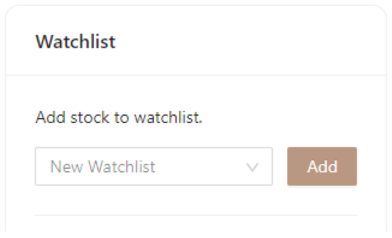
Global Mobile Medical Apps Market Trajectory & Analytics, 2012-2019 & 2020-2027

Yahoo Entertainment • Learn more Dec 4th 2020

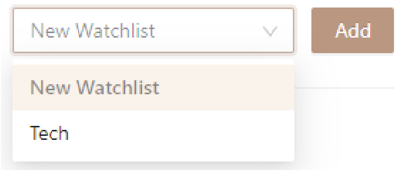
Global Automatic Call Distributor Market

To keep potential investors more informed about stocks, there is a *Related News* section which contains popular articles related to the stock. Please note that due to the nature of the project, and the ability to simulate dates in the future, these articles can only be fetched for the current day. Selecting dates in the future will not result in new articles to be generated.

Watchlists

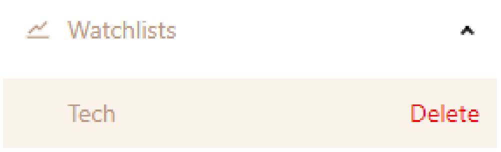


Users can also create watchlists to keep track of stocks they are interested in. Watchlists can be created by selecting a target stock from the stock bar, and then navigating to the Watchlist section. By default, clicking the Add button will prompt the user to specify a name for their new watchlist.



Stocks can also be added to previously created watchlists by selecting them from the dropdown. Note that if a stock has been added to a watchlist, it cannot be added again to the same watchlist, hence will not show in the dropdown.

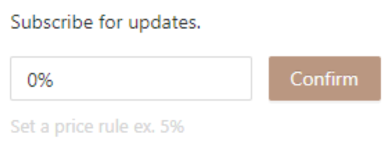
To view watchlists navigate to the Watchlists tab in the navigation menu and select a watchlist. The stock bar will update, only showing the stocks added. Users can also delete watchlists by selecting Delete.



Manage Funds

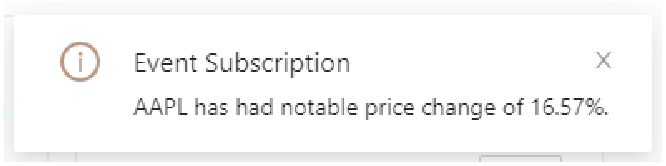
Users can deposit/withdraw funds from their account to finance stock purchases. Select Orders from the navigation menu and navigate to the Manage Funds section. Select Deposit or Withdraw, and specify the amount. Notifications will be sent regarding the status of the action.

Event Subscriptions



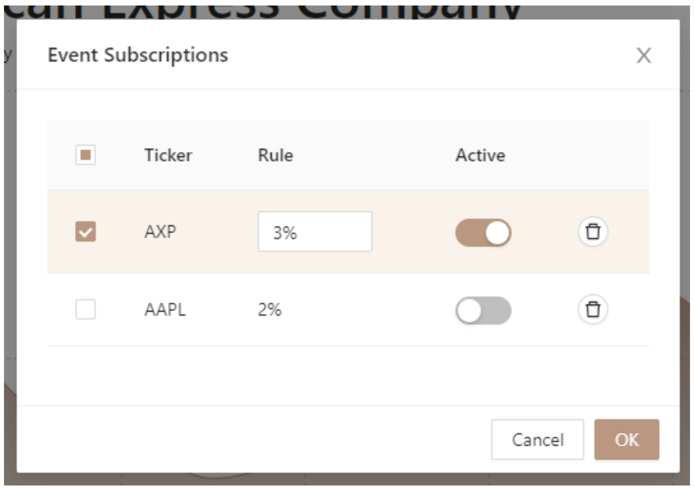
Users can also register to receive notifications for significant stock price activity. After selecting the target stock, navigate to the Watchlist section and specify a price change rule. Users will then receive notifications once a day when the condition is met.

These notifications include the stock ticker of interest along with the price change (including the direction +/-).



Event Subscription

AAPL has had notable price change of 16.57%.

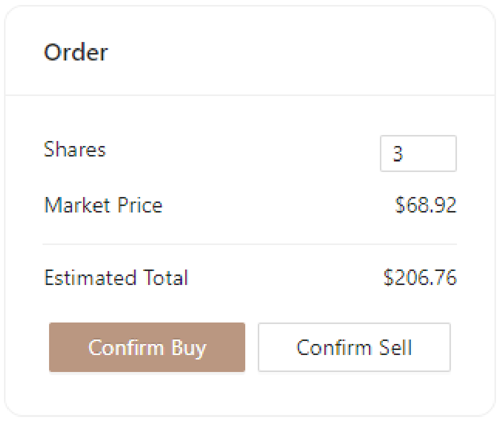


Event Subscriptions

	Ticker	Rule	Active	
✓	AXP	3%	●	
	AAPL	2%	○	

To view and manage subscriptions, select the Subscriptions tabs in the navigation menu. A modal will appear with a table of the current registered subscriptions. To update a subscription, select the checkbox next to the subscription entry. Users can change the rule, active status, and delete the subscription. Note that only selected rows will be updated.

Buying/Selling Stocks

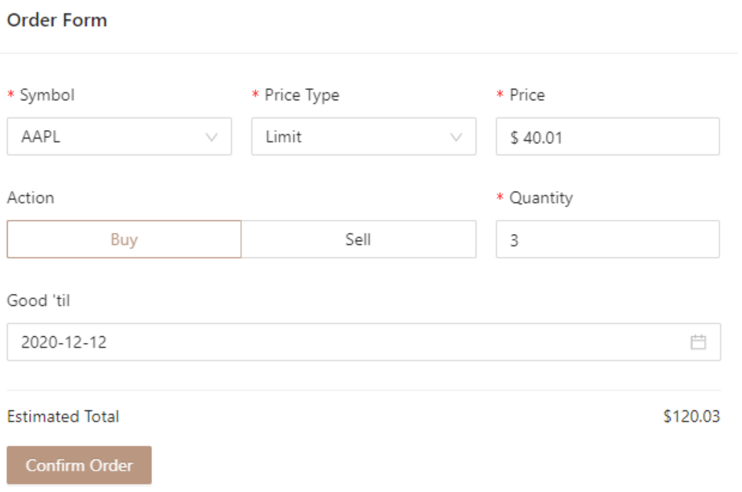


Order

Shares	3
Market Price	$68.92
Estimated Total	$206.76

There are two ways to buy and sell shares using Fleamarket 52. The first method involves selecting the target stock from the stock bar and navigating to the Order section. Using the form users can specify the number of shares they would like to buy/sell, with all orders being made using the market price. Notifications will be received notifying the user if the order was placed successfully or not.

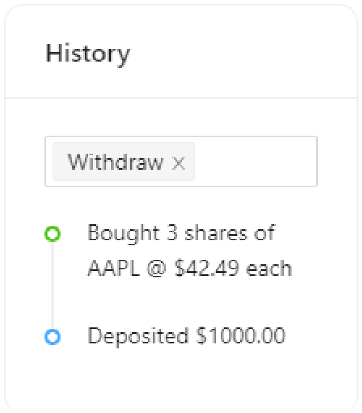
The second method involves selecting Orders from the navigation menu. Users can complete an order form specifying the target stock, price type, price for limit orders, buy/sell, quantity, and an expiry date. Default orders are good till cancelled.



Order Form

* Symbol	AAPL
* Price Type	Limit
* Price	$ 40.01
Action	Buy / Sell
* Quantity	3
Good 'til	2020-12-12
Estimated Total	$120.03

Past Transactions


History

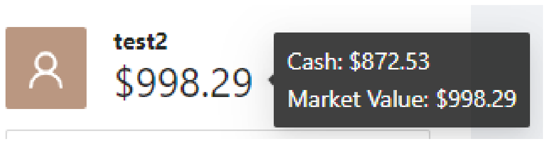
Withdraw ×

Bought 3 shares of AAPL @ $42.49 each

Deposited $1000.00

Users can view past orders and deposit/withdraw transactions on the Order page. All orders will appear in the sidebar with brief details regarding the ticker, quantity, price, and status. These orders can be searched by stock ticker as well. Users can view all transactions with the History section. A timeline is shown detailing past transactions in reverse chronological order. Users can also filter out certain actions to view key transactions.

Miscellaneous

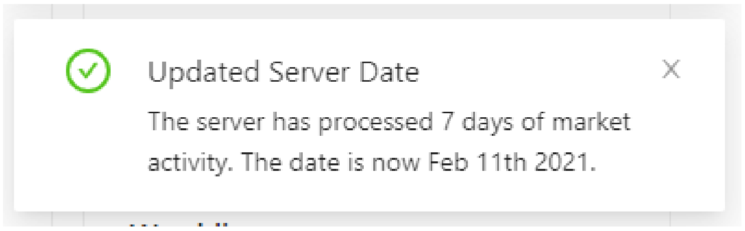

test2

$998.29

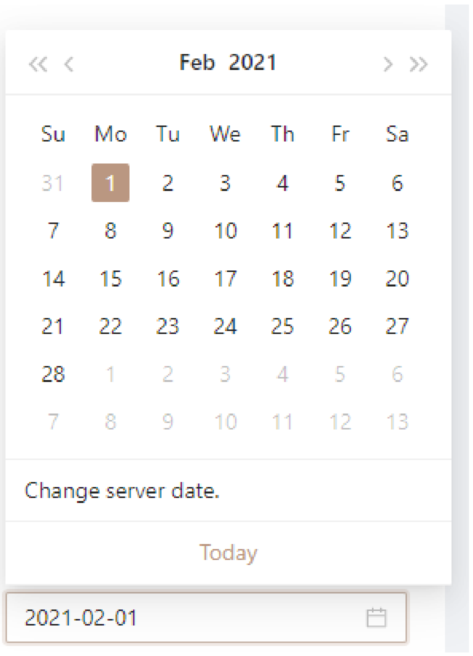
Cash: $872.53

Market Value: $998.29

Users can view their current cash balance and portfolio stock value by hovering over the balance in the Profile section located under the nav menu.


Updated Server Date

The server has processed 7 days of market activity. The date is now Feb 11th 2021.



Fleamarket 52 also allows users to simulate multiple days of stock market activity. Selecting the date picker from the Profile section will prompt users with a calendar to select a future date. Note that only future dates relative to the current will be accepted. A notification will be received when the server has completed simulating the stock activity for the specified duration (takes up to 10 seconds).

Market Simulation

The previously mentioned features were regarding the client-facing interface. However, the core functionality of the application depends on the server's ability to simulate the stock market. Fleamarket 52 uses a random walk algorithm to generate the next ask and bid prices for each stock. For instance, if the previous ask was $100, the algorithm would generate a future price between ($100 - random factor) and ($100 + random factor). This factor depends on the duration of time being simulated. By default, the script is attempting to simulate 10 seconds worth of activity. However, when simulating multiple days at a time, this factor increases to compensate.

The application also comes with an order fulfillment process in which the users' orders are completed based on conditions. For example, a sell limit order of $100 will only be completed if the price for the target stock rises above $100.

These features can all be found in the `simulate.js` utility script found in the directory `/src/utilities`. This script is called every 10 seconds with the client making frequent requests to check for the most recent data.

Technical Details

This section will discuss the technical aspects of the application such as the modules used, API setup for REST and GraphQL, server structure, and authentication system.

Modules & Frameworks

A variety of modules were used to create Fleamarket 52. Here is a brief breakdown of a few important modules used and the functionality they help provide.

Module	**Functionality**
Nodemon	Refresh server when changes are made to code.
Mongoose	Connect and interface with a MongoDB database.
Moment.js	Datetime module with useful methods for time manipulation.
Jsonwebtoken	Used in the authentication process to authorize users.
GraphQL	Data querying framework, a key component for the project's API.
Express	Web app framework for node backend.
CORS	Cross-origin resource sharing module, used when testing React client.
Bcrypt	Used for hashing passwords before saving to the database.
Axios	Module to simplify making HTTP requests.
React	Frontend UI library.
ApolloClient	Used to interface with GraphQL API on the server.
Ant Design	React UI library, similar to Material UI.
Recharts	A graph module, used to display stock data.

API Documentation (REST)

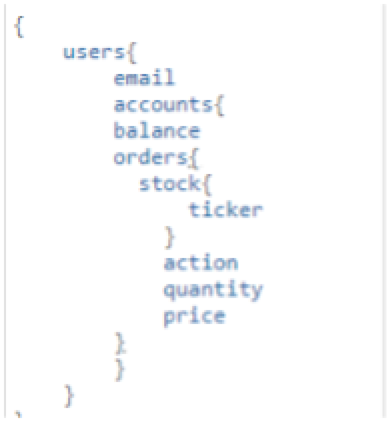
A variety of core functionality for the application is available through a REST API interface accessible here: `http://localhost:8000/api/rest/stock/{ROUTE}`. These routes can be tested through Postman. Here are a few of the endpoints available:

Route	**Params**	**Return**
GET /stocks	symbol, minprice, maxprice	Array of stock objects which include ticker, name, market.
GET /stocks/:symbol	startday, endday	Array of stock data objects which include daily ask, bid, high, low, volume, and date.
GET /stocks/:symbol/history	startday, endday	Array of stock order objects which include the action, quantity, price (in cents), completed date, expiry date, and username.

Note that all day parameters (startday, endday) are integers starting at 0. Since the server generates 14 days of stock data, day 0 corresponds to a date 14 days **before** the current date. Moreover, all price data (ex. ask, bid, high, etc.) are listed in cents (ex 12345 -> $123.45).

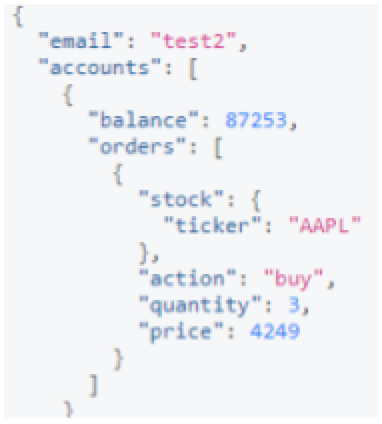
API Documentation (GraphQL)

The project also uses GraphQL to provide functionality for the React client. GraphQL comes with an interface (similar to Postman) to test queries and mutations. It also comes with dynamic documentation for the various endpoints, specifying the query/mutation parameters and return fields. It can be accessed here: `http://localhost:8000/api/graph`. The benefit of using a query language like GraphQL is the ability for clients to have more control regarding the data they wish to access. Moreover, relationships can be defined between models, allowing for powerful queries.



```
{
    users{
        email
        accounts{
        balance
        orders{
          stock{
            ticker
          }
          action
          quantity
          price
        }
        }
    }
```

Request



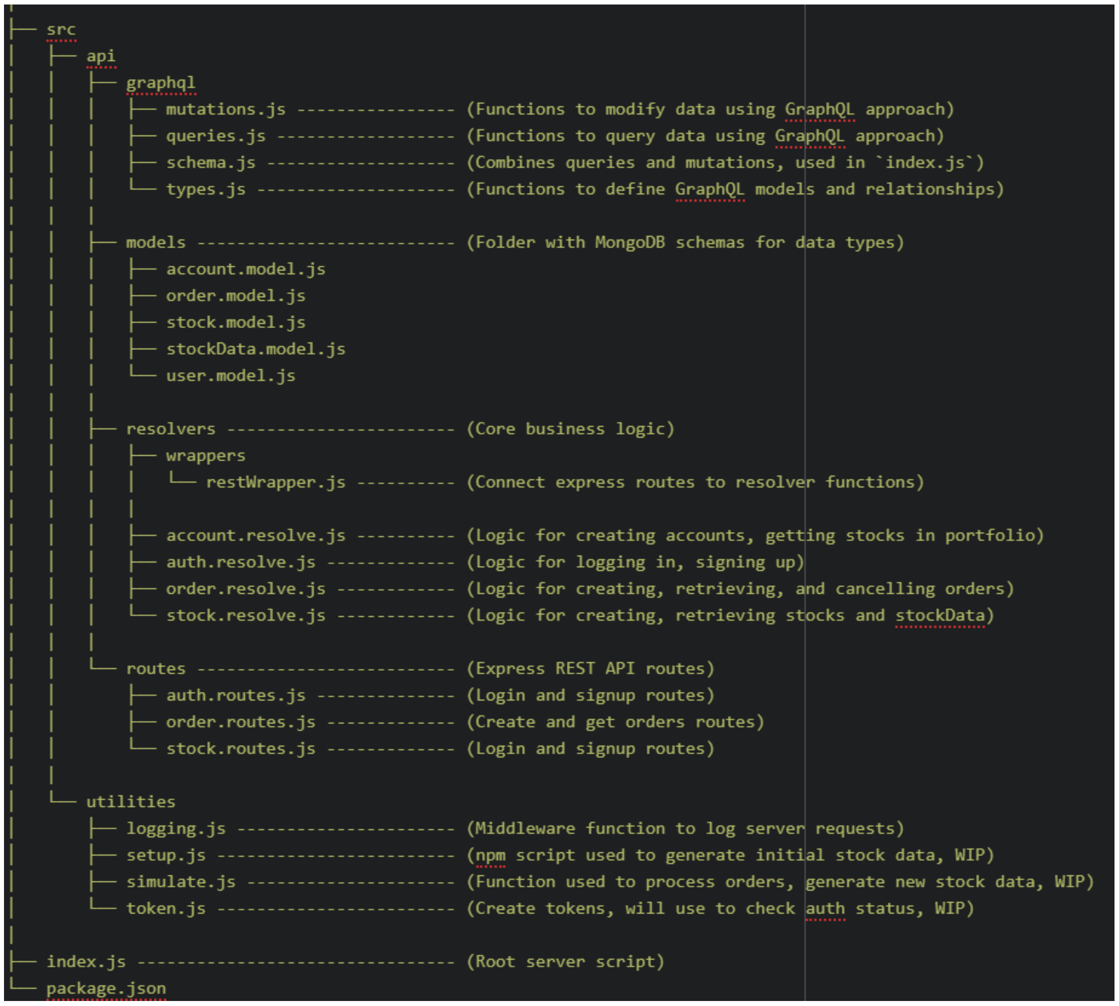
```
{
  "email": "test2",
  "accounts": [
    {
      "balance": 87253,
      "orders": [
        {
          "stock": {
            "ticker": "AAPL"
          },
          "action": "buy",
          "quantity": 3,
          "price": 4249
        }
      ]
    }
```

Response

The following example illustrates the benefit of defining these relationships. With a single request the client is able to retrieve all the user objects, along with the account objects related to the user, and the orders related to the account.

Server Structure

When developing the backend server for Fleamarket 52, a major emphasis was placed towards a modular design. The following is a brief overview of the organizational structure for the server:



```
├── src
│   ├── api
│   │   ├── graphql
│   │   │   ├── mutations.js ---------------- (Functions to modify data using GraphQL approach)
│   │   │   ├── queries.js ------------------ (Functions to query data using GraphQL approach)
│   │   │   ├── schema.js ------------------- (Combines queries and mutations, used in `index.js`)
│   │   │   └── types.js -------------------- (Functions to define GraphQL models and relationships)
│   │   │
│   │   ├── models -------------------------- (Folder with MongoDB schemas for data types)
│   │   │   ├── account.model.js
│   │   │   ├── order.model.js
│   │   │   ├── stock.model.js
│   │   │   ├── stockData.model.js
│   │   │   └── user.model.js
│   │   │
│   │   ├── resolvers ----------------------- (Core business logic)
│   │   │   ├── wrappers
│   │   │   │   └── restWrapper.js ---------- (Connect express routes to resolver functions)
│   │   │   │
│   │   │   ├── account.resolve.js ---------- (Logic for creating accounts, getting stocks in portfolio)
│   │   │   ├── auth.resolve.js ------------- (Logic for logging in, signing up)
│   │   │   ├── order.resolve.js ------------ (Logic for creating, retrieving, and cancelling orders)
│   │   │   └── stock.resolve.js ------------ (Logic for creating, retrieving stocks and stockData)
│   │   │
│   │   └── routes -------------------------- (Express REST API routes)
│   │       ├── auth.routes.js -------------- (Login and signup routes)
│   │       ├── order.routes.js ------------- (Create and get orders routes)
│   │       └── stock.routes.js ------------- (Login and signup routes)
│   │
│   └── utilities
│       ├── logging.js ---------------------- (Middleware function to log server requests)
│       ├── setup.js ------------------------ (npm script used to generate initial stock data, WIP)
│       ├── simulate.js --------------------- (Function used to process orders, generate new stock data, WIP)
│       └── token.js ------------------------ (Create tokens, will use to check auth status, WIP)
│
├── index.js -------------------------------- (Root server script)
└── package.json
```

I have defined the REST API endpoints in the `/api/routes` directory which maps URL paths to logic functions. This allows for a clear understanding of the interface for the REST API, and leaves out the details of the implementation. Similarly, I have implemented GraphQL types, queries, and mutations (found in the `/api/graphql` directory) which contain details of the interface and not the implementation.

While the REST routes and GraphQL methods represent the API's interface, I have created **resolver** functions which include details of the API's implementation/functionality (found in the `/api/resolvers` directory). These functions aim to resolve client requests by performing the actions necessary and return the required data. They have been further organized based on general features (ex. `auth.resolve.js` handles authentication logic, `order.resolve.js` handles stock order logic). This modular form of organization is useful as certain functionality could be used in other programs (ex. authentication).

Authentication System

A key focus for the project was implementing a secure authentication system. This was achieved by using JSON Web Tokens (JWTs). Here is a breakdown of the authentication process:

1. User uses credentials to either login/signup.
2. Server processes credentials, queries database, and returns a token if successful.
    a. Token is serialized with the user's ID, a secret, an an expiry date
3. Subsequent requests involve the client sending the token through the request headers.
    a. Server has middleware function to check for token in headers, check expiry, and provide a refreshed token if successful

With this process, the client is responsible for storing the access token so that it can make requests. Saving and sending the token repeatedly is much more secure than sending the user credentials frequently. The token expiry dates also enhance the project's security as they ensure that the client must verify their identity often.

Final Remarks

Fleamarket 52 fulfills the requirements for the stock broker application project by providing users a platform to buy and sell stock. It does this through the use of extensions such as React, MongoDB, GraphQL and News API integration. Moreover, the system is designed in a modular manner, encouraging future development and scalability. Future improvements could include linking the project to real stock price data rather than random simulation. The application's best feature from the frontend perspective is the stunning user interface. React allows for dynamic components and rendering which is difficult to replicate using standard HTML5 and JS. From the backend point of view, the GraphQL implementation stands out as it provides a new and efficient way of providing data through an API.