Version 1.1 - 09/21/2023

# Requirements Specification Document

## Real-time In-browser Stock Explorer (RISE)

Taiga Chang, Michael Chao, Isaiah Daye, Skylar Homan, Anoop Krishnadas

# 1. Introduction

## 1.1 Purpose of Project

This is a requirement specification document for our product, the Real-time In-browser Stock Explorer (RISE). We are a group in the Software Engineering class at Hofstra University that is planning on making a real-time stock web application that will display real-time stock information and allow users to track the stock value of companies listed on the NASDAQ stock exchange. This document will allow readers to understand the scope, objectives, and goals of our application. Along with the non-functional requirements we will also be showcasing functional requirements of our product, such as interaction models, and UI development.

## 1.2 Project Participants

Project Name: Real-time In-browser Stock Explorer (RISE)

Project Manager: Dr. Scott Jeffreys

Project Members: Taiga Chang, Michael Chao, Isaiah Daye, Skylar Homan, Anoop Krishnadas

## 1.3 Intended Audience

The audience of this project is intended to be newer users who are looking for an efficient method to access the stock price, information, and statistics of companies listed on the NASDAQ stock exchange. Some examples of potential users include everyday stock monitors, day traders, and surface level stock information searchers.

## 1.4 Intended Use

This application is intended to be an easy and effective way to browse stock information and details for any companies listed on the NASDAQ stock exchange. Our application will be classified as a “user-friendly” application made to be more accessible for newer users and investors.

## 1.5 Scope

This application will primarily provide the following services outlined in sections 3 and 4, which includes a stock lookup feature to browse stocks on the NASDAQ, detailed information regarding the queried stock, recent news regarding the particular stock, and a graph of the stock being queried with real-time updates.

# 2. Project Description

## 2.1 User Needs

This is a necessary utility for users as it is a layman's application. In the world of investing, stock trade, and financial markets where there are several moving factors and magnitudes of information to keep track of, our application provides an effective service that condenses publicly available information into "bite-sized" and condensed statistics.

We provide an interactive and new user friendly UI that flattens the learning curve and reduces the barriers to entry for many looking to invest or simply even learn about the stock-trade. While other platforms offer copious amounts of information, many of which is not often relevant for newer users, our application hand-picks the most useful information for beginner stock traders to access.

## 2.2 Stakeholder Needs

RISE's stakeholders include the developers (project team), beginner stock traders using our application to browse stock information, experienced traders using our application for surface-level monitoring of various lists of stocks, and even non-traders using our platform to access relevant stock details or news articles for a variety of other uses.

Our application is targeted for being simple, user-friendly, and easily scalable, meaning that our baseline provides condensed summaries of information that is incredibly useful for new users who are still constantly browsing and learning about a variety of stocks.

## 2.3 Dependencies

RISE will be dependent on two external APIs (to be selected); one for retrieving real-time stock information, and one for searching and displaying news articles. These APIs must be free or have a free version, and will be selected based on their response speed and how many requests per minute/day they allow. Our program may also be dependent on an external database framework such as MongoDB or Firebase, and/or an external website host such as Github Pages.

## 3. Functional User Requirements

### 3.1 Graph

The website will have a graphical representation of the stock being searched with a price to hour line graph of the stock, detailing the changes in stock price over time. The graph can be hovered over and the stock price at the corresponding time will be shown to the user. The graph should be updated every five seconds. Based on query rate and display update limitations, this update period should be reduced as low as possible without significantly limiting performance.

The horizontal axis of the graph will represent time, and can be adjusted in scale by pressing buttons to the side labeled with different time intervals (e.g. "live," "5 minutes," "daily," "weekly," etc.). The vertical axis will have the price of the stock, and will scale based on the highest and lowest values of the stock for the given time-scale.

### 3.2 Stock Information

Below the stock graph will be important trading information for the stock in the main display. The information will be Stock price, Sector, Industry, Share volume, Day's high, Day's low, Market cap, P/E ratio, Div yield, etc. The stock price will be refreshed every 5 seconds and be placed at the top. They will be placed under the sector and industry information.

### 3.3 Related News

The website will have related news articles to the company that will help users understand the status of a stock and the company it represents. There will be two articles (subject to availability) that are pulled from an existing news network and summarized below the stock information. The link to the full article will be posted on the bottom of the summary. The articles will be updated every 5 minutes so that they represent current news about the company.

### 3.4 Sidebar Stock-List

On the left side of the screen a column with rectangular blocks will be shown. The rectangular boxes will be a list of stocks that the user has added to make the stock a favorite. When the stock of a specific company is picked, the screen of the graph, information, and news will change to the company chosen.

The intended number of stocks on the sidebar is 8, but may increase or decrease depending on memory allocation and costs to performance with each successive stock on the list.

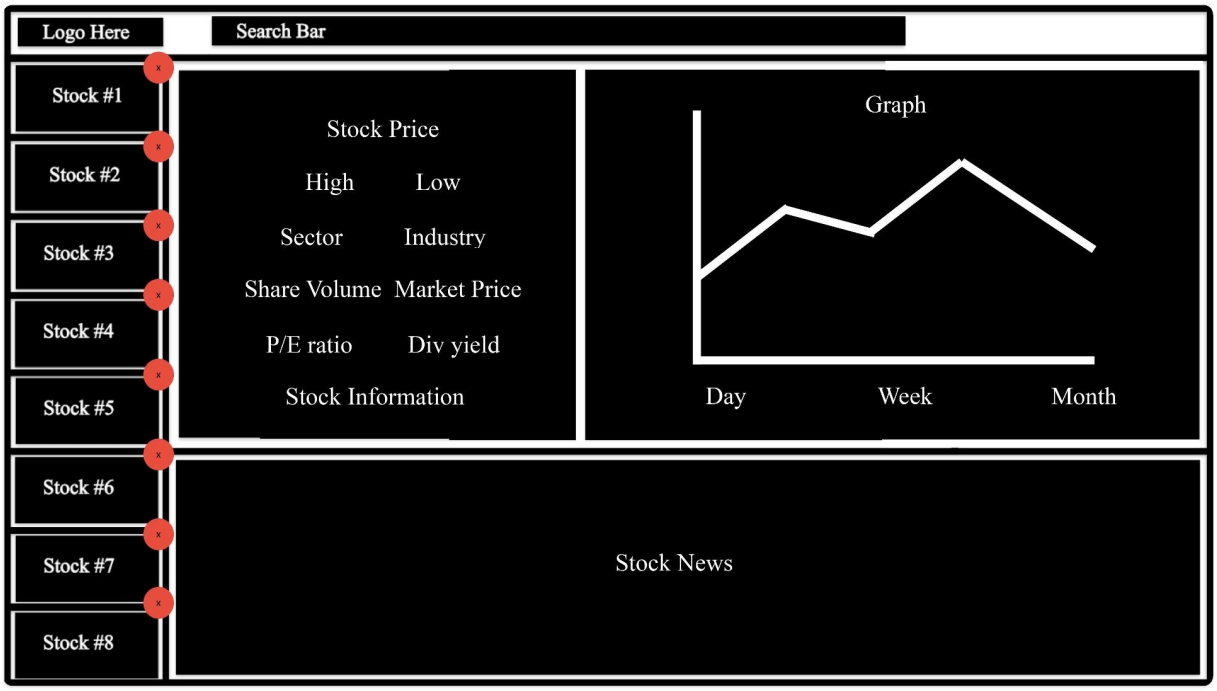
The rectangular box will have a red circular x mark on the right upper corner to indicate deleting the stock from the list of websites shown in the favorites section. Once all the stock spaces are filled, the user will need to delete a stock to make room for another.

### 3.5 Search Bar

There will be a search bar in the upper side of the screen that will let users see which stocks are listed on the website. When they type the name of the company or the abbreviated name, such as AAPL, in the search bar, a dropdown will appear below the search bar showing dynamic results for their search in alphabetical order.

### 3.6 Sample Website Diagram

Below is a diagram showing the placement of elements within the window.



# 4. Functional System Requirements

## 4.1 Database and API

When the application is started on the web server host's machine, a list of all NASDAQ stock symbols, along with their associated company names and trading information, will be retrieved using a publically available API and stored in a database. The API will continue to be used to retrieve real-time information (see 4.4), with the database storing this information as well to act as a cache for API queries. Whenever the user switches focus to a different stock, the database will first be queried for relevant information, to reduce latency from unnecessary API calls.

## 4.2 Search Bar

Along the top of the window will be a text entry field, which the user can type in to search for stocks in the application's database. Each character entered will dynamically update the search results, matching any company name and/or ticker symbol that starts with the entered characters. Searches will not return names or symbols which only contain a match partway into the name or symbol: for example, the search entry "ma" will match with "$MA" and "Mastercard," which each contain a word starting with "ma," but not "Amazon," which contains "ma" after the start of the word. Beginning a search with the $ character will restrict search results to only ticker symbols, but the search will still return ticker symbol results without a $.

Search results will be displayed in a box that drops down from the search bar when it is selected, with results stacked vertically. Each search result will display both the company name and ticker symbol. Clicking on a search result for a stock will clear the search bar and update the application's "focus," changing the stock info, graph, and news sections (4.4 - 4.6) to display information about the stock which was clicked.

For the search bar to function, we will need to maintain an underlying database containing all the stocks that our APIs support. We will be able to query efficiently as long as the database we create holds the company name/ticker symbol of any company we have access to the data for.

## 4.3 Sidebar Stock-List

The left-hand side of the window will have a sidebar made up of stacked rectangular boxes, which can be used to "pin" stocks for easy access. On startup, all boxes will be filled with a default selection of stocks, either chosen at random from the NASDAQ listings or manually selected before launching the web server. These boxes will display the stock's ticker symbol, current share price, and an up/down arrow indicating the recent trend of the

price. Clicking on one of the boxes will switch the focus info, graph, and news sections to display information about the stock the box represents, the same as if the user had selected it from the search bar.

In the upper right corner of each sidebar box will be an X button. When this button is clicked, the box will become empty, "unpinning" the stock it contained. With any stock focused, the user can click on an empty box to pin it, populating it with the focused stock's ticker symbol, current share price, and trend arrow and allowing the user to click it to change focus again. However, if the focused stock is already pinned in a different box, clicking another empty box will not pin it an additional time.

### 4.4 Stock Information

Next to the price graph will be a window containing additional information about the focused stock (see 3.2). Just like the price data used to update the graph, this information will be updated using calls to the API:

- Requests for unchanging or infrequently changing data (ie. sector, industry, and market cap) will be made as part of the initial API call at startup, and again once every hour that the program has been running (not necessarily on the hour).
- Requests for stock price will be made once every 5 seconds for all stocks focused and/or pinned in the sidebar.
- Requests for all other information will be made once every 5 seconds for the focused stock only.

For this project, no paid APIs will be used. If the API chosen for the product does not allow as many API calls as are required, the update frequency may be reduced during development from the ideal once per 5 seconds, down to a minimum of once every 15 seconds.

### 4.5 Graph

The main focus of the application window will be a line graph displaying the recent price movement of the focused stock versus the time at which that price was retrieved from the NASDAQ listings. The graph will be shifted to the left each time the price is updated, with the most recent price displayed along the left-hand side. Hovering over a data point with the cursor will display the price and time which it represents, which will be hidden again when the cursor is moved away.

The scale of the y-axis will be dynamically updated along with the price display. The axis will extend 10% higher than the highest displayed price point and 10% lower than the lowest displayed point, rounded to the nearest cent, with a minimum range of $0.00 to $1.00.

### 4.6 Related News

Two news articles will be pulled from (an) existing online news outlet(s) using a publicly available API (different from the one used to retrieve stock info) and displayed side by side below the graph and info sections. The articles selected for display will be the top two search results for the ticker symbol, the company name, or a combination of the two, to ensure the selected articles contain relevant information about the company. News articles will be updated every 5 minutes, rather than every 5 seconds like other data, and will also be stored in the database.

For each article, the application will display the headline/title, the date and time of publishing, the first 100 words, and a link which will open it in a new browser tab. If multiple news outlets are used to source articles, the application will also display the name or logo of the outlet which published each one.

# 5. Non-Functional Requirements

## 5.1 Reliability Requirements

The software should be optimized to have as little downtime as possible. Stock prices can be very volatile, and moment to moment fluctuations may influence a users decision on whether or not to buy or sell their stocks, so unexpected system downtime may have financial ramifications for the user. If the product is down too often or for too long, it will fail to display relevant, accurate information in real time.

The software can be optimized in the development process by undergoing a software editing/condensing process which removes redundant and inefficient components to the code and replaces it with more efficient methods. Better memory management, faster queries, and better data handling will all contribute to reducing the time from collecting the data from the API and converting it to a display for the user.

## 5.2 Response Time Requirements

In addition to minimizing downtime, rapid response time is also paramount for a real-time application. Data requests should be made at short, regular intervals to ensure that the display is as close to live as possible, and the system should be able to process the information from those requests as quickly as possible once it is returned. Data retrieved will be cached in a database so it can be quickly reloaded if necessary, without needing to wait on additional calls. Finally, by the time the next data requests are being sent, the user should already see the results of the old requests on the site; there should be as little lag as possible between the user's view and the software's current information.

## 5.3 Usability Requirements

The website should be intuitive and easy to use, even for users who are not familiar with stocks or trading. All primary site functions must be accessible using the mouse, without need for keyboard inputs or scrolling. When searching for specific stocks, keyboard input will be required in a designated text field. Site elements should be intuitively labeled as necessary, and colors and symbols should be used where appropriate to indicate their function (e.g. a magnifying glass to indicate the search bar; red X buttons to unpin stocks from favorites).

The purpose of making the software “user friendly” is because our software is intended to be used by a demographic of individuals who are not used to being active in stock trading, Our main goal is to provide a user experience that is so intuitive that an individual who knows nothing about stock trading could see our website and understand how the website works.

### 5.4 System Requirements

The system running the web server will require varying storage depending on the storage architecture for the site, such as how much it would need to always save and have that information on hand, and how much data would it have to hold before being able to release the data from its storage. If the storage were not properly allocated, the website may suffer from delays stemming from the system not being able to receive data, push data, and then reallocate space for new data.

The website should not require any special specifications for the user's system. Users should be able to access the site from any internet-connected device and expect comparable performance (allowing for external factors such as user connection speed).

Depending on if the software is scaled, the user’s browser cookies may need to be requested and accessed to restore the former state so that a user’s activity can be saved (improves the user experience).

### 5.5 Network Requirements

The web server's host machine and the user's machine must both be connected to the internet with a stable connection in order to send and receive real time updates. A fast and stable connection is most important for the host, as the software will need to send and receive many raw data requests and push the processed data to the user. It is not as important for the user to have a strong internet connection, though a faster connection will mean updates will be received sooner once the server sends them.

### 5.6 Performance Efficiency Requirements

The software should be optimized to require as few API calls and database updates as possible, as these actions are slow and too many may negatively impact performance. It is important to to minimize the amount of times that data requests are actually needed to ensure that the website runs smoothly. This will improve the application's reliability and response time, which in turn will help satisfy other non-functional requirements.

Furthermore, one of the major limitations to the speed with which the display can be updated is the data and memory management in the application. Should our program unnecessarily use excessive memory or if it should poorly handle the received data, then the update to the display will be delayed significantly.

# 6. Scalability

## 6.1 Potential for Scalability

Due to limitations on time, number of project members, and available resources to handle a variety of features, our product will not guarantee the features listed below upon initial completion. If time and resources allow, the features below are intended to be added as an aspect of the scalability of our product. While primarily targeted towards less experienced users and investors, our application could be scaled to make certain advanced features toggleable.

## 6.2 Additional Features

Some of the additional features intended to be added as a part of scalability are listed below. These features are intended to provide a greater quantity of information and has an intended audience of more advanced users who intend to find stock information on companies listed on the NASDAQ.

If possible, a demo of the app will be hosted online at a publicly accessible web address to allow users without the application and hosting framework installed to test it for themselves.

Persistent favorite lists would allow users to be able to access a selection menu to select a custom list of stocks to be displayed on the sidebar. This feature would require the creation of a database and a sign-in system so that users would be able to access a saved list of their preferred stocks between sessions. Otherwise, the lists would be wiped when the webpage reloads.

Saved activity would be a feature that could be added provided functionality does not come at a significant cost of efficiency, where browser cookies (or other data saving methods) would be used to preserve a user's activity in the website. Then, if a user were to return to the website, their previous list would be saved on the sidebar, and they would not need to fill it manually again.

Advanced Analytics mode is a toggleable feature that would allow users to see a larger variety of information and also access a stock comparison feature. This feature would allow users to select two stocks which will be displayed side-by-side with their respective information written below each stock graph for easy comparison by users.

Smoother UI would be a feature that prioritizes UI/UX over performance, and would include several mini-features such as a slide-bar to remove stocks from the side bar.

# 7. Workflows

## 7.1 Workflow Outline

While the following steps leading to a creating and managing a web application may appear intuitive, it is imperative to outline the necessary steps in order to accomplish such a goal. The general outline will detail what the goal of the step is, as well as what should be focused on when programming that stage of the process. The limitations and aspects to be wary of will be included in each section as well.

## 7.2 Data Collection

In order to produce a website to display stock data for users, our software must be able to access stock data. In order to do this, our project will require the collection of data from a minimum of two APIs.

The first API will contain real-time stock statistics, such as the current price, market open, daily high, etc. We will need to regularly query this API for real-time price updates to the stock in the display. We will only frequently pull information for one stock at a time, and the program will infrequently pull information for all the stocks in the display (sidebar list + main display).

The second API will provide access to news articles and an associated “expectation” on a particular stock. This API will be queried infrequently as it will not need to be updated unless the stock in the display is changed or if a certain amount of time has passed.

## 7.3 Data Management

Data Management is an essential part of the project development process as our project is dependent on efficiency. This efficiency is lost in the process of querying data from the API, interpreting the data, and moving the data to the database. In order to prevent this, there is a specific procedure with which the data should be manipulated.

Data should be extracted as a JSON file from the API after the query, This file is hard to use directly, and rather, should be “unwrapped” by several backend functions. The backend will need to separate the data into its barebone components and labels to be sent to a database for the frontend to query from. The process of moving the data to the database should be streamlined by having prior testing on the functions used to push the data. Error handling and resizing should all be considered during the development process.

### 7.4 Updating Display

The process of updating the display will necessitate that the data has already been moved to a database for the frontend to efficiently query from. The data should be labeled and accessed if the query information (stockname) matches the name of the stock data being temporarily stored. There will be several functions made to request data from the database and use it to construct the various display elements (graph, statistics, etc).

### 7.5 Error Handling

The project development team will be responsible for being aware of and handling all potential errors (that are controllable from the developers' side of the project). This means that a master list of error codes should be created so that the developers will be able to cross reference how to handle that particular error. Because functionality is an essential part of our web application, the website should not crash due to poor programming.

### 7.6 Tasks and Traceability

| Major Grouping: | Task Name: | Task Code: | Dependency: | Comments: |
|---|---|---|---|---|
| Project Planning (0.x) | Establish Meeting Dates and Synchronize Project Members' Workspaces | 0.1 | None | |
| | Create Github Repository and Establish Procedure | 0.2 | None | Procedure for Pushing to Main Branch |
| | Define Development Process and Editing Rules | 0.3 | None | |
| Data Collection (1.x) | Create API Testing Environment | 1.1 | None | |
| | Pull Data from APIs | 1.2 | 1.1 | |
| | Test Speed and Efficiency of Data Pull Calls | 1.3 | 1.2 | If inefficient or limited, change APIs |
| | Consider Potential Errors and Solutions | 1.4 | 1.3 | Master-list of error codes should be created for handling |
| Data Management (2.x) | Create Database for API Data | 2.1 | None | Associate the Data stored with the Company Name/Ticker Symbol from 2.7 |
| | Create Functions to Interpret JSON Data from APIs | 2.2 | None | Functions should output data to go directly to database |

| | | | | |
|---|---|---|---|---|
| | Test Functions on sample JSON and Add Error Handling | 2.3 | 2.2 | Do with self-made JSON file of same format |
| | Test Functions on Real API Pulls | 2.4 | 1.2, 2.3 | |
| | Push Interpreted Data into Database | 2.5 | 2.1, 2.4 | |
| | Test Speed and Efficiency of Pushing and Pulling Data from Database | 2.6 | 2.5 | If speed/efficiency is poor, make necessary changes |
| | Create Database for Stock Ticker Symbols and Company Names | 2.7 | None | Needs to exist permanently outside of program |
| | Consider Potential Errors and Solutions | 2.8 | 2.4, 2.6 | |
| Updating Display (3.x) | Create Website Framework | 3.1 | None | Add HTML |
| | Design Website Layout and Display | 3.2 | 3.1 | Add CSS Elements for positioning and basic visuals |
| | Outline Necessary Data for each Section of Display | 3.3 | None | |
| | Create Pull functions to Query Company Name/Ticker Symbol Database | 3.4 | 2.7 | |
| | Test Pull function Speed and Efficiency | 3.5 | 3.4 | |
| | Create Functions to Pull Data for Frontend | 3.6 | 2.1, 3.3, 3.4 | (Javascript, or other frontend language) |
| | Evaluate Function Efficiency and Error Handling | 3.7 | 3.6 | |
| | Move Data to Display and Test for Changes | 3.8 | 3.2, 3.6 | |
| | Add UI Features | 3.9 | 3.2 | Ensure that these features do not significantly impede efficiency |
| | Test Website and Edit for Functionality | 3.10 | 1.4, 2.8, 3.8, 3.9 | Review each step and ensure functionality |