Version 1.0 - 10/2/2023

Project Plan

Real-time In-browser Stock Explorer

(RISE)

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

1. Introduction

1.1 Purpose

This is a project planning document for our product, the Real-time In-browser Stock Explorer (RISE). The purpose of this document is to create a breakdown of the tasks involved in the project in order to identify risk areas and ensure each team member is on the same page in terms of the project timeline. This document will take the requirements outlined in the Requirements Specification Document for the RISE project team and schedule tasks for each team member. Team members will be assigned relative to the risk of the task, the amount of time available/needed to complete the given task, and based on team member availability. We will also create a project timeline, with traceable components with ES, EF, LS, LF, and a recognition of the slack and critical path along our project timeline.

1.2 Scope

This project plan will encompass the preliminary design, mapping, and task assignment phases necessary for the team. The project duration is from October 3rd, 2023 to November 30th, 2023. This project has 58 days from start to finish. Of these days, 39 are working days, 16 weekend days, and 3 federal holidays falling on weekdays.[1] Thus, the project must be completed within the 39 working days, allowing us to allot tasks within those dates.

1.3 Objectives

The overall objective of this project is to develop a real-time stock lookup web application. There are three primary phases in this project: Data Collection, Data Management, and Data Display. While the details are expanded upon in the Requirements Specification

[1] https://www.calendar-12.com/days_between_dates

Document, these three phases comprise the majority of the workload. We prioritized Data Collection first, because if we cannot efficiently query and interpret data from our APIs, the remainder of the project will be rendered unusable.

2. Project Organization and Overview

2.1 Team Info and Roles

The team consists of Taiga Chang, Michael Chao, Isaiah Daye, Skylar Homan, and Anoop Krishnadas. Isaiah has prior experience working with stock market APIs and databases, so he will primarily lead back-end development, retrieving and processing data. Skylar has prior experience with website design and hosting, so she will primarily lead front-end development, building the website interface and displaying data to the end user. Taiga has previous experience with the database MongoDB and also has experience with frontend, such as HTML and CSS with React. He will be focusing his efforts on backend in the beginning of the project and transition to frontend as the project progresses. Michael will divide their efforts between those areas as required by the project timeline. Anoop has had previous experience with front-end development, including HTML, CSS, and Javascript, so he will work along the critical path in backend until the Data Display section (Front-end work) begins.

2.2 Project Deliverables

The deliverables of this project include the following:

1. Fully functional User Interface providing real-time stock data for the NASDAQ
    a. The data can be expanded to include NYSE and others depending on the ticker symbol database creation and software efficiency

2. Graphical representation of stock data in a line graph format
3. Personal stock-list sidebar for users to save stocks to follow (acts like a hotbar)
    a. Can be expanded to allow users to maintain multiple lists, as well as saving user data in browser cookies to retain each user's list
4. Stock information section, compiling general information about the company reflected in the ticker symbol alongside 2 news articles
5. Interactive User Display allowing users to query stocks by name and ticker symbol to either be added to their list or displayed on the main graph

2.3 Project Milestones

In order to improve the traceability of the project and allow for visualization of progress along the project's timeline. The breakdown below will indicate the milestones of our project, and the task codes associated with the duration of each milestone. Though there may be overlap/dependencies, these milestones will serve as a functional reminder of what tasks have been completed and what remains to be done.

1. START	(0.0)
2. Team Organization	(0.1 - 0.3)
3. API Testing	(1.1 - 1.4)
4. Interpreting JSONs	(2.2 - 2.4)
5. Database Creation	(2.1, 2.5 - 2.8, 3.4)
6. Website Outlining	(3.1 - 3.4)
7. Backend => Frontend	(3.5 - 3.8)

8. Additional Features	(3.9)
9. Error Handling and Edits	(1.4, 2.8, 3.7, 3.A)
10. FINISH	(4.0)

2.4 Branching Policies

Each project member will initially clone a copy of the Github repository to their personal devices. From here, each member will be responsible for their actions by working on an individual branch from main. No member will be allowed to edit the Main Branch directly. Pull requests to merge a feature branch to main must be approved by at least three other members of the team.

Each branch must be named according to the task being undertaken as well as describe the goal of the branch. For example, a branch that intends to implement the Search Bar Interface must be labeled “3.4_Search_Bar”.

Project Members must update their local repositories and attempt to handle any merge collisions on their local machines prior to making their pull request. This will prevent further issues that may complicate the repository/necessitate editing in the main branch.

2.5 Project Management and Control

In an effort to streamline the development process and individual workflows, we will establish the following set of rules on utilizing Git and Github to outline the interactions between individuals.

Atomic Commits will be the methodology utilized by the development team. This means that team members must make their commits as small as reasonably possible (should be describable in one sentence). This will allow for efficient debugging and troubleshooting if a commit causes any issues. Furthermore, it provides traceability in the project.

Each team member must make their commits utilizing the task codes outlined below. For example, if a member is working on task 2.3, their commits should be written as follows:

Commit 2.3.1 - Tested JSON Interpreting Functions

Furthermore, team members should report to the other members when starting a new branch, the tasks they intend to complete, and what tasks are left unfinished in the branch after each commit. Team members should also outline issues they faced, nomenclature used, and formatting/organization for consistency throughout the project.

Team members must outline the functions and other parts that will be created in the new feature branch when opened. This will be tracked in the README file. Other team members should state in the communication channel when each function or part of the feature is finished, as well as update that in the README. Updates to the README including new tasks in the feature must be notified to the remaining members in the communication channel.

In the case of a merge/commit that breaks the software, the repository will be restored to its last stable version/release and the errors from the previous merge/commit must be interpreted.

3. Risk Analysis

3.1 Project Risks

The biggest potential risk to successful project completion comes from our dependencies. If the API, database, and/or website framework are unable to meet our speed and capacity requirements, they will need to be replaced with a different such framework. This could cost a huge amount of time learning and integrating a new framework if an inadequacy is discovered late into the project. To prevent this, we will prioritize selecting dependencies as soon as possible and testing them rigorously before they are integrated with one another, so that if anything needs to be replaced it will be discovered before it becomes a critical element of our system.

After dependencies, our next biggest risk comes from the link between the program's back end (database) and front end (website and data display). The back end database integration will be written in Python, using Pandas for array management, while at least some of the code for the front end will likely be written in Javascript to interact with the website. As a result, it will be necessary to send inputs and data from one language to another, requiring a consistent data format that both languages can read. Figuring out how to make this connection work must be a high priority as well, since without it the program cannot function.

Generic coding bugs are almost certain to occur no matter how much is done to prevent them. If not identified quickly, fixing uncaught bugs could easily slow the critical path and set our production timeline back significantly. Linting provided by our IDE (4.2) will help prevent basic bugs, but we will also incorporate regular testing into our development cycle to identify when new components or workflows introduce bugs and which parts of the program they stem from.

3.2 Product Risks

Our primary product risk is the possibility that our finished product will not be able to deliver real-time stock data due to our programs running slower than intended. To avoid this, our development decisions must prioritize program speed, so that the delay between receiving data back from an API call and that data being visible to the user in their browser is as low as possible.

While cross-OS compatibility for the back end framework is not a priority for our product, it is important that the product be runnable on all project members' machines, as otherwise some members may be unable to test the product and properly contribute to debugging, etc. This risk can be mitigated with regular testing of the system on all members' machines,

4. Resource Requirements

4.1 Hardware Requirements

The sole hardware requirement for our project is a computer which can host the web application and run the necessary program(s) to make API calls and update the database. Despite the need for database storage and fast processing speeds, an average modern computer should have enough RAM and a good enough processor to run the product without issue, as the size of our database should be small compared to modern average RAM capacity. However, if a virtual machine or similar environment has to be used in order to run the product on a particular system, it is important that it be given plenty of access to system resources in order to avoid throttled performance or program termination due to insufficient memory.

We must also ensure that each team member’s machine will be able to host the web application and run the programs for API calls, in order to avoid dependency on a single machine. Doing so may cause issues during the display and presentation process for the project.

4.2 Software Requirements

The software requirements involved would be with mainly the API we will choose to use; Pandas, a database framework; and Visual Studio Code, a code editor and IDE.

Our choice of API will depend on its speed and flexibility, to ensure we can use it for the purpose we need it for (pulling stock information/news/etc) while wasting as little waiting time as possible. Additionally, if the API's interface and return format is too complicated or if the it doesn't provide the response time and the information we are expecting, that would contribute towards the decision process.

Our database framework will include Pandas, a Python data library that we plan to use to store our information. We need this software to act as our database and store the information before we push it onto the project site. Pandas will enable us to store data and access it quickly to ensure the update time is fast on the website.

Finally, we have Visual Studio/VS Code, the IDE we will use to write scripts to allow for the pulling and pushing of the information onto the website. All project members are familiar with VS Code, and it allows us to write, save, and apply scripts to our project, as well as providing linting to help catch basic code errors with minimal effort.

In addition to this, all team members will be required to understand the Git Workflow, as well as maintaining a local repository cloned from Github on their personal machine. Though it is not explicitly foreign software, project members must ensure that their git and local repository are up to date upon each day of project work. This will keep the version as stable and up-to-date as possible.

5. Work Breakdown

5.1 Task Chart

The following Chart will be the Task Chart, outlining the major tasks, their associated task codes, and the dependencies of each task. Highlighted in yellow is the Critical Path.

Major Grouping:	Task Name:	Task Code:	Dependency:	Comments:
START	Project Start	0.0	None	
Project Planning (0.x)	Establish Meeting Dates and Synchronize Project Members' Workspaces	0.1	0.0	
	Create Github Repository and Establish Procedure	0.2	0.1	Procedure for pushing to main branch
	Define Development Process and Editing Rules	0.3	0.2	
Data Collection (1.x)	Create API Testing Environment	1.1	0.3	
	Pull Data from APIs	1.2	1.1	
	Test Speed and Efficiency of Data Pull Calls	1.3	1.2	If inefficient or limited, may have to change APIs
	Consider Potential Errors and Solutions	1.4	1.3	Masterlist of error codes should be created for handling
Data Management (2.x)	Create Database for API Data	2.1	0.3	Will need to associate data with the company name/ticker symbol (2.7)
	Create Functions to Interpret JSON Data from APIs	2.2	1.4	Functions should output data to go directly to database
	Test Functions on sample JSON and Add Error Handling	2.3	2.2	Test with self-made JSON file of same format
	Test Functions on Real API Pulls	2.4	1.3, 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	2.1	Needs to exist permanently, even when program is inactive
	Consider Potential Errors and Solutions	2.8	2.4, 2.6	
Data Display (3.x)	Create Website Framework	3.1	0.3	HTML
	Design Website Layout and Display	3.2	3.1	Add CSS for positioning and basic visuals
	Outline Necessary Data for each Section of Display	3.3	2.2	
	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	3.5, 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.A	1.4, 2.8, 3.8, 3.9	Review each step and ensure functionality
FINISH	Project End	4.0	3.A, 3.6, 3.7	

6. Scheduling

6.1 Scheduling Table

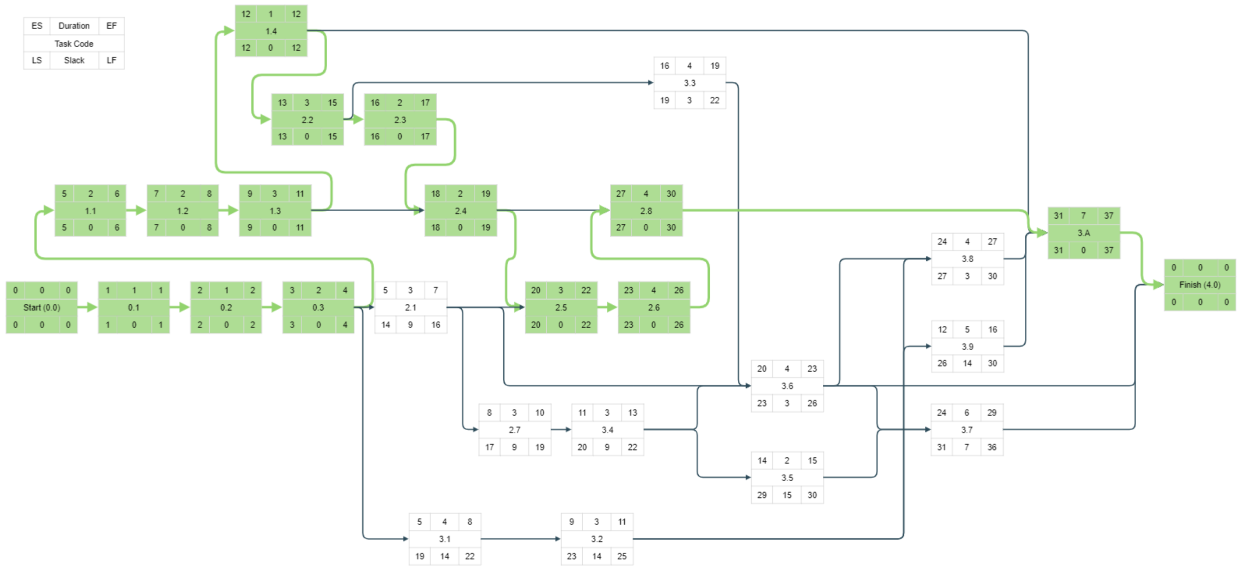
The following table is a scheduling table, containing each major task code along with the expected duration (time spent), alongside minimum and maximum durations, the priority/risk of the major task, as well as the team members assigned to that segment of the project. Of course, the team member assignment will have a more advanced distribution of work during actual project development, but the general members who will contribute in each section is listed.

Major Grouping:	Task Name:	Task Code:	Minimum Duration:	Time Spent (days):	Maximum Duration:	Priority / Risk	Team Member Assignment:
START	Project Start	0.0	0	0	0	N/A	N/A
Project Planning (0.x)	Establish Meeting Dates and Synchronize Project Members' Workspaces	0.1	1	1	2	Low	All Members
	Create Github Repository and Establish Procedure	0.2	1	1	1	Low	All Members
	Define Development Process and Editing Rules	0.3	1	2	2	Medium	All Members
Data Collection (1.x)	Create and Outline API Testing Environment	1.1	1	2	3	Low	Isaiah
	Pull Data from APIs	1.2	2	2	3	High	Isaiah, Taiga, Michael, Anoop

	Test Speed and Efficiency of Data Pull Calls	1.3	1	3	4	High	Michael, Anoop, Skylar
	Consider Potential Errors and Solutions	1.4	1	1	2	Medium	Anoop, Skylar
Data Management (2.x)	Create Database for API Data	2.1	1	3	4	Low	Taiga
	Create Functions to Interpret JSON Data from APIs	2.2	2	3	4	Low	Taiga, Skylar, Anoop
	Test Functions on sample JSON and Add Error Handling	2.3	1	2	3	Medium	Skylar, Michael
	Test Functions on Real API Pulls	2.4	1	2	2	High	Skylar, Isaiah, Michael
	Push Interpreted Data into Database	2.5	2	3	4	High	Isaiah, Michael, Anoop, Taiga
	Test Speed and Efficiency of Pushing and Pulling Data from Database	2.6	2	4	5	High	Anoop, Taiga, Isaiah
	Create Database for Stock Ticker Symbols and Company Names	2.7	2	3	5	Medium	Anoop, Isaiah
	Consider Potential Errors and Solutions	2.8	3	4	5	Medium	Anoop, Isaiah, Michael, Taiga
Data Display (3.x)	Create Website Framework	3.1	2	4	6	Low	Skylar, Anoop
	Design Website Layout and Display	3.2	2	3	5	Low	Skylar, Anoop
	Outline Necessary Data for each Section of Display	3.3	2	4	4	Medium	Skylar, Anoop, Isaiah
	Create Pull functions to Query Company Name/Ticker from Symbol Database	3.4	3	3	4	Medium	Anoop, Skylar, Michael
	Test Pull function Speed and Efficiency	3.5	1	2	3	Low	Anoop
	Create Functions to Pull Data for Frontend	3.6	3	4	6	High	Skylar, Anoop, Taiga, Michael
	Evaluate Function Efficiency and Error Handling	3.7	2	6	4	Medium	Anoop, Michael
	Move Data to Display and Test for Changes	3.8	2	4	5	High	Isaiah, Anoop, Skylar

	Add UI Features	3.9	4	5	6	Medium	All Members
	Test Website and Edit for Functionality	3.A	5	7	8	High	All Members
FINISH	Project End	4.0	0	0	0	N/A	All Members

6.2 Critical Path Chart



7. Monitoring and Testing

7.1 Reporting

Each member of the project team is responsible for the work they do. Thus, each member is also responsible for reporting the work that they have completed to all the other members of the team in a timely manner. This will ensure that all team members are up to date on the work that needs to be completed, work that is already completed, and the work that is currently in progress. The team members will also be responsible for staying up to date themselves with the project. This will allow for all team members to be ready when CEO Dr. Jeffreys asks for an update on the progress.

7.2 Monitoring

Monitoring of development will mostly be done through github. As mentioned above in the Project Management and Control Section (2.5), our project will have set procedures and goals. In the process of developing the product, we can track our progress based on our milestones reached, tasks completed, and duration along in our path, as well as how far we have strayed from our critical path and overall timeline.

We will consistently monitor and stay up to date with the level of progress at the start and end of each work week. This will allow us to be able to recognize the amount of work completed, the work that still needs to be done, and the future distribution of time and man-hours for each section of the project.

7.3 Testing

The beginning of the project will begin with unit testing with the API. We will run unit tests to see if we are able to obtain one company's stock information. Once we understand the API and the requested speeds we will begin more component testing with the Fortune 500 companies to make sure we can obtain their data. Along with this testing we will work to see if we can pull the data to the database to hold the stock information. The interface testing stage will begin when we see if our database is obtaining data from the API. Once this is tested the tests on the frontend will begin. There will be interface testing on the website to make sure the parameters are not broken when the interface is misused. When the backend and frontend testing is completed the system testing will begin. The components will be integrated and made sure the right data is pulled when the user requests data from a certain company.

Despite not running automated tests, we plan to have a consistent test format. For if statements, we will test all possible “types” of inputs, including fringe cases, as well as both the true and false input. For looping expressions such as for, while, and switch statements, we will make sure to test critical inputs to ensure that an exception or infinite loop is not reached.

We will also ensure to monitor and catch all possible errors and establish an error handling procedure as mentioned in the Requirements Specification Document. We plan on organizing our errors by labeling each case with an error code, and then publishing those codes in the README file.