-

Imagine a future where your smartphone isn't just smart; it's your dedicated stock trading guru! We've harnessed the power of cutting-edge AI to design an app that predicts stock movements, empowering you to buy and sell with confidence. Dive into our portfolio and explore how we're revolutionizing the world of stock trading.

*Profit Predictors*

Project Portfolio

*11/13/23*

[Introduction <Milestone 1: Proposal> 2](#_Toc125988975)

[The [Team Name] Team < Milestone 1: Proposal > 3](#_Toc125988976)

[System Requirements < Milestone 1: Proposal > 4](#_Toc125988977)

[Requirements [optional] 4](#_Toc125988978)

[Epics [optional] 4](#_Toc125988979)

[Epic #1 4](#_Toc125988980)

[User Stories 4](#_Toc125988981)

[User Story #1 4](#_Toc125988982)

[Project Management 5](#_Toc125988983)

[Continuity of Operations Plan (COOP) < Milestone 1: Proposal> 5](#_Toc125988984)

[Project Plan 0](#_Toc125988985)

[System Architecture Design and Development < Milestone 1: Proposal & Milestone 2: Architecture> 0](#_Toc125988986)

[System Implementation <Milestone 2: Architecture & Milestone 3: System Implementation> 0](#_Toc125988987)

[Project Postmortem <Postmortem> 0](#_Toc125988988)

[Project Wins 0](#_Toc125988989)

[Root Cause Analysis 0](#_Toc125988990)

[Lessons Learned 0](#_Toc125988991)

[System Design <Milestone 2: System Architecture> 1](#_Toc125988992)

[System Architecture <Milestone 2: System Architecture> 1](#_Toc125988993)

[Component Design 1](#_Toc125988994)

[Data Flow 1](#_Toc125988995)

[System Components <Milestone 3: System Implementation> 1](#_Toc125988996)

[Component [Component Name 1] 1](#_Toc125988997)

[Component [Component Name 2] 1](#_Toc125988998)

[Component [Component Name n] 1](#_Toc125988999)

[Design Pattern <Milestone 3: System Implementation> 1](#_Toc125989000)

[Design Pattern <Milestone 3: System Implementation> 2](#_Toc125989001)

[System Implementation <Milestone 3: System Implementation> 3](#_Toc125989002)

[Project portfolio template directives and placeholders (delineated by “[ ]” or “< >” and/or highlighted or optional sections not included) should be removed from the document prior to submission. Empty sections for inclusion in later submissions may remain in the document for early submissions.]

[IMPORTANT: All diagrams developed using Enterprise Architectures must include the following acknowledgement: “Thanks to SPARX Systems for LSU student and faculty use of Enterprise Architect for academic purposes”.]

# Introduction

In today's fast-paced financial world, stock market participants, from seasoned traders to novice investors, often struggle to make timely and informed decisions. The sheer volume of data, news, and market signals can be overwhelming, making it challenging to discern the best moments to buy or sell a stock. Furthermore, emotional biases often lead to suboptimal decisions, which can significantly impact investment outcomes.

Enter our AI-powered Stock Trading Assistant. Designed meticulously using C++, our system capitalizes on the efficiency and speed of this object-oriented language, ensuring rapid real-time analyses. The application amalgamates various cutting-edge technologies, such as machine learning for predictive analytics and cloud computing for scalable data storage. Users can benefit from intelligent stock predictions while automating their trading strategies, all on a user-friendly interface. By merging the prowess of C++ with advanced technologies, we provide traders a tool that cuts through the noise, reduces emotional biases, and ushers in a new era of informed trading.

A blue hexagon with white letter c and plus symbols

Description automatically generated

`Core Features:

* Intelligent Stock Prediction:
  + Description: Utilizing advanced machine learning algorithms, the app would analyze historical and real-time data to predict potential price movements of stocks.
* Automated Trading System:
  + Description: Users can set their trading parameters, risk tolerance, and investment goals. Once set, the system will autonomously execute buy and sell orders based on its predictions and the user's presets.

Viable Features:

* Personalized Investment Dashboard:
  + Description: A visually appealing dashboard where users can monitor their portfolio's performance, view the AI's recent trade decisions, and get personalized insights and recommendations based on their trading history and market trends.

Adaptive Risk Management:

* + The feature would allow users to set their risk tolerance level (e.g., low, medium, high) when trading. The AI would then adjust its trading recommendations and strategies based on this input. For instance, a user with a "low" risk tolerance would see recommendations for more stable stocks and ETFs, while those with a "high" risk tolerance might receive suggestions for more volatile stocks or sectors. The AI would continuously learn from the market's behavior and the user's feedback to fine-tune these risk parameters, ensuring the recommendations align with the user's comfort zone and financial goals.

# The Profit predictors Team

Team Structure:

1. William Graham

* Role: Project Manager
* Responsibilities:
  + Oversee the project's overall direction and progress.
  + Coordinate meetings and ensure effective communication among team members.
  + Assign tasks and ensure milestones are met on time.

1. Riley Oest

* Role: Lead Developer (Backend)
* Responsibilities:
  + Design and develop the app's backend architecture.
  + Integrate the AI algorithms with the app.
  + Coordinate with the frontend team to ensure seamless integration.

1. Don Juan

* Role: AI & Machine Learning Specialist
* Responsibilities:
* Design and train the AI models for stock prediction.
* Constantly refine and update the models based on new data and feedback.
* Work closely with the backend team for AI integration.

1. Landon Clarke

* Role: Frontend Developer
* Responsibilities:
* Design and implement the app's user interface.
* Ensure a responsive and user-friendly experience.
* Collaborate with the backend team for data display and integration.

1. Mark Echols

* Role: Quality Assurance & Tester
* Responsibilities:
* Conduct rigorous testing of the app, both frontend and backend.
* Identify bugs and coordinate with developers for solutions.
* Ensure the app maintains a high standard of quality and reliability.

1. Andrew Holden

* Role: Financial Analyst
* Responsibilities:
* Provide insights into stock market trends and data.
* Collaborate with the AI specialist to provide domain knowledge.
* Assist in validating the accuracy of AI predictions.

1. Giovanni

* Role: Integration Specialist & API Developer
* Responsibilities:
* Develop and manage APIs for interfacing with external data sources, ensuring real-time stock data feeds into the app.
* Coordinate with both frontend and backend teams to ensure seamless data flow and integration.
* Address integration-related challenges, ensuring the app's various components communicate effectively.

### User Story #1

[*User Story Statement, using the following format:*

As an investor, I want to get AI-driven risk assessments for specific stocks based on historical data and current market conditions, so I can manage my investments wisely.

# Project Management

## Continuity of Operations Plan (COOP)

Effective communication and coordination are the lifeblood of our project, and as such, we are committed to ensuring a seamless flow of operations even in the face of unforeseen challenges.

Our primary mode of communication will be through a dedicated channel, where all updates, queries, and discussions will take place. Weekly Zoom meetings will be scheduled for team check-ins, ensuring we stay on track and address any pressing issues. For version control and code collaboration, we will utilize GitHub, ensuring that every member is consistently updated with the latest developments.

In light of the potential challenges posed by situations like the COVID-19 pandemic, all team members are equipped to work remotely and have access to all necessary tools and platforms from their homes. If a team member cannot attend in-person meetings, they can join virtually or catch up with recorded sessions, ensuring no one misses out on critical updates.

In case a member becomes temporarily unavailable due to unforeseen circumstances, tasks will be reallocated among the team based on the skill set. Each member will document their roles and current tasks comprehensively so that in their absence, another team member can take over with minimal friction. If, however, a member becomes permanently unavailable, we will convene an emergency meeting to redistribute responsibilities, ensuring the project's continuity and timely delivery.

Our COOP is designed to maintain our project's momentum, regardless of the obstacles we might face, ensuring that we consistently deliver on our commitments.

Group Gethib link: [hkaiserteaching/csc3380-fall-2023-project-group-3: csc3380-fall-2023-project-group-3 created by GitHub Classroom](https://github.com/hkaiserteaching/csc3380-fall-2023-project-group-3)

## Project Plan

### System Architecture Design and Development < Milestone 1: Proposal & Milestone 2: Architecture>

[Milestone 1 (Proposal): The Project Plan WBS provides a list of activities/tasks to be undertaken to complete Milestone 2 (Architecture). The WBS activity chart should include task dependencies, estimated level of effort, and expected start and completion dates.

Milestone 2 (Architecture): The WBS activity chart for the milestone should be updated to include actual level of effort and start and completion dates.]

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Activity** | **Pre #** | **Estimated**  **Effort** | **Actual**  **Effort** | **Estimated**  **Start Date** | **Estimated**  **Finish Date** | **Actual**  **Start Date** | **Actual**  **Finish Date** |
| 1 | Investigate and implement twitter API |  | 80% | 5% | 9/13/23 | 10/5/23 | 9/15/23 | 9/27/23 |
| 2 | Implement accurate web scraping |  | 80% | 95% | 9/13/23 | 10/5/23 | 9/17/23 | 10/1/23 |
| 3 | Develop a basis and a front end |  | 70% | 70% | 9/13/23 | 10/5/23 | 9/13/23 | 10/6/23 |

### System Implementation <Milestone 2: Architecture & Milestone 3: System Implementation>

[Milestone 2 (Architecture): The Project Plan WBS provides a list of activities/tasks to be undertaken to complete Milestone 3 (System Implementation). The WBS activity chart should include task dependencies, estimated level of effort, and expected start and completion dates.

Milestone 3 (System Implementation): The WBS activity chart for the milestone should be updated to include actual level of effort and start and completion dates.]

## 

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Activity** | **Pre #** | **Estimated**  **Effort** | **Actual**  **Effort** | **Estimated**  **Start Date** | **Estimated**  **Finish Date** | **Actual**  **Start Date** | **Actual**  **Finish Date** |
| 1 | Alpaca ai data integration |  | 90% | 100% | 10/6/23 | 10/25/23 | 10/12/23 | 11/13/13 |
| 2 | Implement sentiment Ai |  | 90% | 100% | 10/6/23 | 10/25/23 | 10/9/23 | 11/7/23 |
| 3 | Implement front end around back end functions |  | 90% | 100% | 10/6/23 | 10/25/23 | 10/6/23 | 11/8/23 |
| 4 | Implement back testing |  | 90% | 95% | 10/8/23 | 10/27/23 | 10/9/23 | 11/4/23 |
| 5 | Implement qt frame work for the front end |  | 75% | 85% | 10/20/23 | 11/13/23 | 10/25/23 | 11/13/123 |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

# System Design <Milestone 2: System Architecture>

The system design for our AI-powered stock trading platform emphasizes user-centricity and real-time responsiveness, ensuring trades have both the insights and the tools they need to make informed decisions in the volatile world of tock trading.

## System Architecture <Milestone 2: System Architecture>

The system architecture of the Profit Predictor algorithmic trading AI consists of the User, the AI, the Alpaca trading API, and a series of function to generate a confidence factor. The User passes a series of however many stocks they would like to trade to the AI, the AI then calls the series of functions starting at getting the stock names. These names are then passed to a data function which gets the data from the names given to the function through the Alpaca API. The names are also passed to a sentiment data function to gather news on the stock which contributes to the confidence factor. After the confidence factor is generated, it is passed to the AI which makes a decision whether to sell, hold, or buy a stock depending on the confidence factor.

### Component Design

A diagram of a flowchart

Description automatically generated

Created by: Landon Clarke + William Graham

[*Architecture overview, to include user I/O, external data sources, and major system components.* ]

### Data Flow

A diagram of a flowchart

Description automatically generated

Created by: William Graham + Riley Oest

[*Architecture data flow discussion: a high-level description of the data between both internal major components and external data sources.*]

## System Components <Milestone 3: System Implementation>

[*Include a component sub-section for each component in the architecture diagram. Each component subsection will include a class diagram*]

### Component[GUI setup]

In our implementation of the Graphical User Interface (GUI), we focused on creating an intuitive and responsive user experience. Our GUI presents users with a clean layout of windows, icons, buttons, and menus, all designed with a modern aesthetic. The interaction is seamless, with each component responding to user inputs like mouse clicks or touch gestures through efficient event handling. We've incorporated a variety of widgets and controls, such as text boxes and sliders, ensuring that they are both functional and visually appealing. The layout management is finely tuned to keep the interface user-friendly and accessible. We paid special attention to providing immediate and clear feedback for user actions, enhancing the overall interaction experience. The graphics rendering is smooth, leveraging the system's hardware capabilities to ensure a fluid and dynamic display. Underneath the polished exterior, our GUI is powered by robust code, meticulously crafted to handle various user inputs and system interactions efficiently. Furthermore, it's seamlessly integrated with the operating system, facilitating a wide range of functionalities while ensuring stability and performance.

UML made by don juan

### A screenshot of a computer Description automatically generated

### Component [Get andData processing]

In our data processing implementation with Alpaca's API, we began by collecting stock market data, focusing on both real-time and historical information. This data then underwent preprocessing to clean and format it into a structured, analyzable form. The next step involved transforming the data, normalizing it, and creating derived metrics to highlight relevant patterns for our trading strategies. We applied statistical methods and machine learning algorithms in the analysis phase to extract market trends and insights, which were then integrated into our trading algorithm. An essential part of the process was backtesting our strategies using historical data to validate their effectiveness. Based on these results, we continually refined our data processing methods and trading algorithms, ensuring their accuracy and efficiency. This comprehensive approach, emphasizing robust error handling and performance monitoring, laid a solid foundation for our algorithmic trading strategies.

A screenshot of a computer

Description automatically generated

Data Processing UML was Created by Riley Oest

### 

[*An EA class diagram of the component that includes method parameters. Include the name of the team member that created the diagram in EA.*]

### Component [Modeling]

After processing the data, we are then able to build a predictive financial forecasting model. This is accomplished by compiling a couple different models which only differ in the number of input days of data for the next day’s prediction. The best of these models (with the optimal number of input days) is then used to predict the close price for the next day. We also use this complied (best) model to plot the predicted trend of the price of chart for the time period selected. This plot is then rendered into html and saved to the disk for later displaying in the GUI window.

Modeling UML made by Mark EcholsA diagram of a model

Description automatically generated

### Component [Techincal indicators ]

Technical indicators are a foundation for algorithmic trading. Using a set of rules, we can automate buy and sell signals based on different indicators. The technical indicators in use are:

* Bollinger Bands
* Dual Thrust
* Heikin-Ashi
* Awesome Oscillator
* MACD Oscillator
* RSI

Using these indicators, an initial capital $ amount, and a position size, we can track the profit/loss of the indicator signals by back testing on a specific time frame.

A diagram of a data flow

Description automatically generated

UML Made by Andrew Holden

### Component [Back Testing]

After acquiring the technical indicator signals, we are then ready to back test. Back testing is valuable to algorithmic traders so they can validate the success of a strategy. Here, we provide back test results for al indicators by embedding a python class which renders the plots of indicators in html and saves the html files to a disk. The plots account for the profit/loss of each indicator given an initial capital and position size to use.

A diagram of a computer program

Description automatically generated with medium confidence

UML made by giovanni

### Component [Sentiment Analysis]// have implemented but did not use

The sentiment analysis algorithm works by acquiring relevant news pertaing to a given tickers through Alpaca’s API. The algorithm goes through each day of data we have saved, and find relevant news for each day. This is a lot of queries for datasets that span more than a year, so we decided to leave it out of the final implementation. However, the algorithm works by collecting the titles of all relevant news for each day and calculating a compound score of sentiment using the Natural Language Toolkit from Python’s nltk library. The sentiment score used for each day in our application averages the compound score for days that have more than one relevant news headline. The scores are the appended to the data from for later analysis.

A diagram of a service

Description automatically generated

UML made by William Graham

## Design Pattern <Milestone 3: System Implementation>

[*Class diagram of design pattern incorporated into the project. Pattern must be specific to the project and not a general design pattern class diagram. The project must include at least design patterns covered in class. Include the name of the team member that created the diagram in EA.*]

Made by: William Graham

A diagram of a flowchart

Description automatically generated

A diagram of a flowchart

Description automatically generated

## Project Postmortem <Postmortem>

### Project Wins

Quantitative trading application in C++: Our application serves as a good start for any algorithmic trader looking to use C++ and/or Python. It includes AI forecasting, technical analysis, optimized data processing, and back testing for technical indicators.

· Seamless Integration: Thanks to the setup\_and\_run.sh shell script, anyone can easily clone the repo and run it, without having to go through and install dependencies. There are also scripts in get\_path.py and getPath.cpp which pull the user path for later use in file reading/writing.

· Efficient File Management: A key strength of our project is the effective handling of file operations. We implemented a system where files are saved using dynamic naming conventions ensuring each file is uniquely identified. This approach not only streamlines the process of saving and retrieving multiple files but also significantly reduces the risk of errors during file access.

· DataFrame.cpp: In python, there is a convenient way to display tabular data (pandas.DataFrame). In this project, we created a similar concept which allowed us to display

the data in a readable format from the GUI. The class we implemented allows for operations like applying formulas to columns, removing/adding columns, and printing.

· TechnicalIndicators.cpp: Using this class and DataFrame.cpp, we were able to calculate metrics and signals for each respective technical indicator and append them to our data frame in C++. Since we have the DataFrame class to perform operations on tabular data, we were able to easily and effectively (with the speed of C++) add respective columns for each indicator.

· InidicatorBacktestCpp.cpp: Using this class, we were able to embed Python code which calculated the back test results and saved the html plot results to the disk.

· Gui.cpp: Compiling the GUI in C++ allowed for a faster response in displaying our software. We were able to implement inputs for ticker, start date, end date, initial capital, and position size to calculate the results. This was especially a win for our project as we were able to nicely display the data frame in a scrollable window, the text output from the model compilation in another window, and the html plots from the back test results in a third window.

### Root Cause Analysis

* Occasional Inaccuracies in Sentiment Analysis:
  + Why: The sentiment analysis tool sometimes misinterpreted the context or the true sentiment of complex financial news and social media posts.
  + Why: This was due to limitations in the natural language processing algorithms, which struggled with nuanced language and industry-specific jargon.
  + Why: The training datasets used for the NLP models might not have been comprehensive enough to cover the wide range of expressions and terminologies used in financial news and social media.
* Lag in Data Processing Speed:
  + Why: There were instances where our data processing pipeline experienced delays, particularly during periods of high market volatility.
  + Why: This lag was partly due to the volume of data being processed and the complexity of our analytical models.
  + Why: The infrastructure and computing resources allocated for data processing were not optimally configured to handle sudden spikes in data volume and computational demand.
* Limited only runs once:
  + Why: Backtesting did not always accurately predict the performance of our trading strategies in live market conditions.
  + Why: This discrepancy arose because historical data and market conditions can never perfectly replicate future market scenarios.
  + Why: Our backtesting models possibly lacked certain dynamic elements of the market, such as real-time reactions to unexpected global events or changes in market regulations.

### Lessons Learned

   Lag in Plotting HTML Back Test Plots:

o   Mitigation Strategy: Instead of displaying all of the indicator plots in one scrollable element, we decided to have “previous” and “next” buttons to display them individually. This allowed for plotting even on very large datasets. ·       Not using Sentiment Analysis: o   Mitigation Strategy: Since the sentiment analysis didn’t improve the model accuracy too much, it was decided that it was negligible and that we could leave the computation out and therefore increasing the speed of the software.

·       Plotting figures with Python.Plotly:

o   Mitigation Strategy: To embed Python code into our C++ application we made a class IndicatorBacktestCpp.cpp that calls a python class and runs the supported functions to save html files to the disk.

# System Implementation <Milestone 3: System Implementation>

[*In the table below, include a row for each component in your System Architecture diagram. In the second column, list the programming language(s) used to implement the component and the what % of that programming language is used in the implementation. In the third column, list the team member(s) that implement the component and what % of that implementation was completed by that team member. IMPORTANT NOTE: All architectural components must be implemented by an object-oriented programming language: Java, C++, or C#.*]

|  |  |  |
| --- | --- | --- |
| **Architectural Component** | **Programming Language(s) %** | **Team Member(s) %** |
| *Run\_py.py* | *C++ (100%)* | Riley Oest [100%] |
| *Get\_data.py* | *[C++(100%)]* | Riley Oest [60%]  Landon Clark[40 %] |
| *Get\_path.py* | *[C++(100%)}* | *Riley Oest[100%]* |
| *Sentiment\_analysis* | *[C++[100 %)]* | *Riley Oest[60%]*  *William graham [40%]* |
| Indicator\_backtest | *C++[100%]* | *Riley Oest[60%]*  *Andrew Holden[40%]* |
| Modeling.py | *C++(100%)* | *Riley Oest[60%]*  *William Graham[40%]* |
| getPath.cpp | *C++(100%)* | *Riley Oest[80%]*  *Giovanni[20%]* |
| DataFrame.cpp | *C++(100%)* | *Riley Oest[70%]*  *Mark Echols[30%]* |
| Gui.cpp | *C++(100%)* | Riley 20%, Landon 20%, Mark 20%, Donovan 40% |
| IndicatorBacktestCpp.cpp | *C++(100%)* | *Riley Oest [70%]*  *William Graham[30%]* |
| TechnicalIndicators.cpp | *C++(100%)* | *Riley Oest[100%]* |