NEA: TRADING BOT + GUI

NAME: GODAMUKA WITHANAGE NETHMINI HIRUNIKA MACKENZY

CANDIDATE NUMBER: -

CENTER NUMBER: -

TOC

Analysis

Foreword

Trading algorithms have become increasingly popular in recent years, and they offer several advantages. One significant benefit is their ability to process vast amounts of data and execute trades at high speeds, which can be challenging for human traders. Algorithms can analyze multiple indicators, market conditions, and historical data simultaneously, leading to potentially more informed and efficient trading decisions. Another advantage is the reduction of emotional biases in trading. Humans can be influenced by emotions such as fear, greed, or excitement, which can lead to irrational trading decisions. Algorithms, on the other hand, operate based on predefined rules and strategies without being influenced by emotions, leading to more disciplined and consistent trading.

However, trading algorithms also have their drawbacks. One concern is that sudden and unexpected events, such as economic or political developments, can significantly impact markets, and algorithms may struggle to adapt quickly enough. Moreover, algorithmic trading is not readily available and used by the general public. The purpose of my project is to offer algorithmic trading in a way that counters both these problems.

A trading algorithm aims to automate the following process of placing an order (buy/sell), by running calculations on the latest stock price data. The following steps occur when placing a market order:

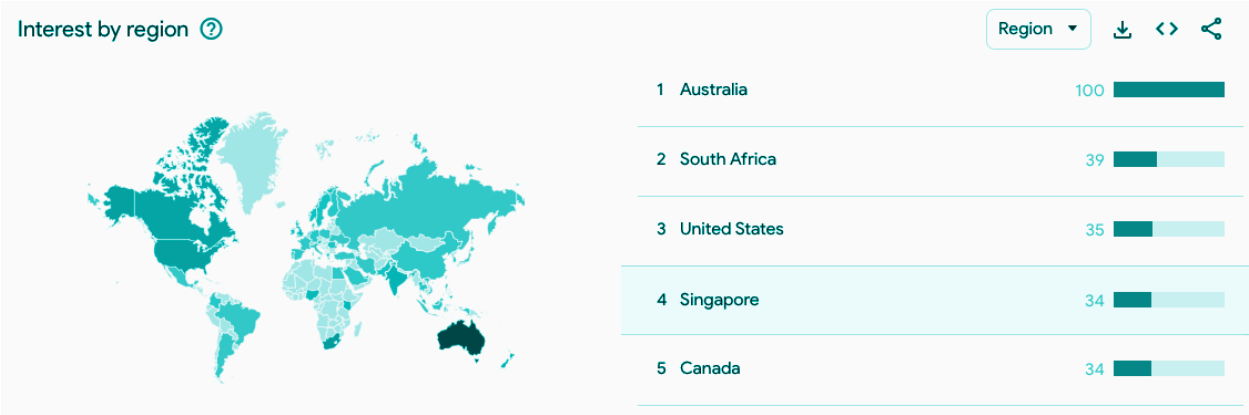
I. Authenticate with platform API using your API key
II. Define the stock symbol, quantity, and order type (e.g., market, limit)
III. Create an order object with necessary parameters (symbol, qty, side, type)
IV. Submit the order using the Python client's method (e.g., place_order)
V. Handle any errors and confirm the order status

Audience

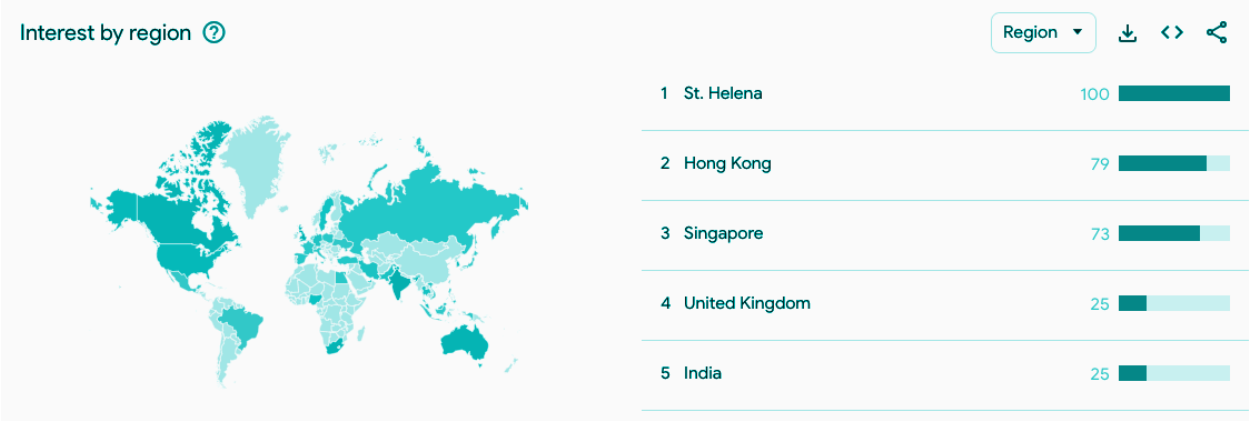
Algorithmic trading is usually used in industry by institutional investors, hedge funds, proprietary trading firms, or firms that specialize in HFTs. Some big players are Renaissance Technologies ($165B in AUM), AQR Capital Management ($61B in AUM), and Citadel Securities ($32B in AUM). While there is an increase in individuals using personalized algorithmic trading, as of today it is not the norm for the average investor. About 35%-50% of the commodity trading volume is generated by algorithmic trading, and similarly, nearly 40% of options trading was via trading algorithms. I aim to create a trading bot, with a user-friendly graphical interface, that will give the user control over the strategies, and simplify algorithmic trading for anyone, with no need for a background in either computer science or economics.

Current System

Is information about the stock/crypto/forex markets, and access to these markets (in terms of trades) and algorithmic trading available to everyone at a fair level? Demographics for such things may depend on environmental factors, like access to the internet, and level of education or knowledge about markets. Google Trends search statistics for words like ‘day trading’ and ‘algorithmic trading’ come mainly from more developed countries.

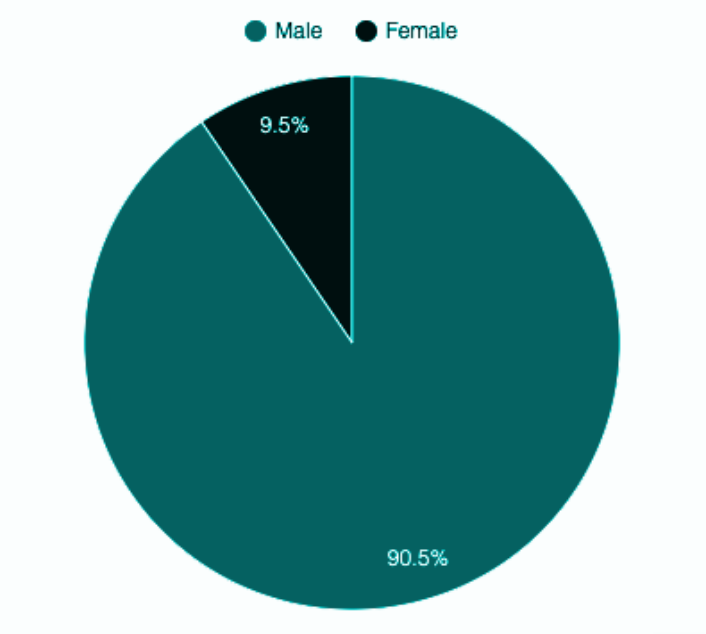


Searches for 'day trading' 2004 - 2023

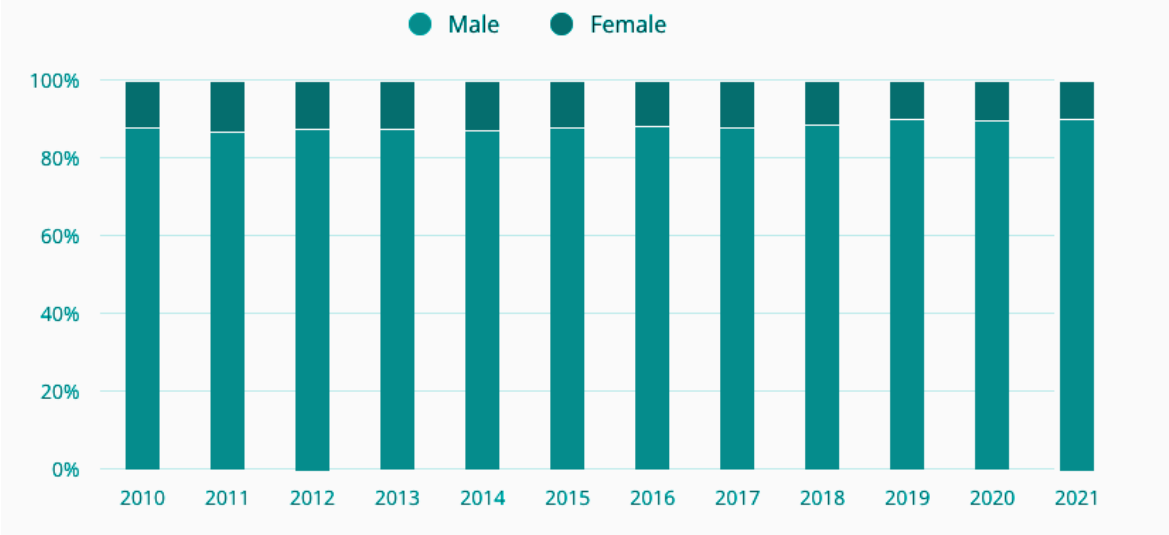


Searches for algorithmic trading' 2004 - 2023

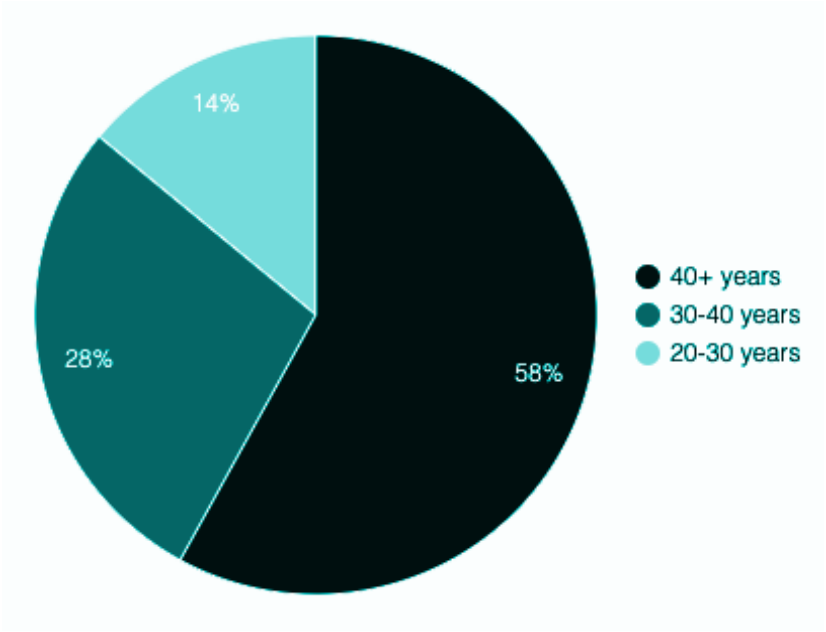
Outside of geography, what other factors about a person makes them more likely to become a day trader? There are several studies about day trading that were carried out in the US, which I will use as a case study to take a closer look at and better understand the demographic. When it comes to gender, for example, day trading is a very male dominated industry.



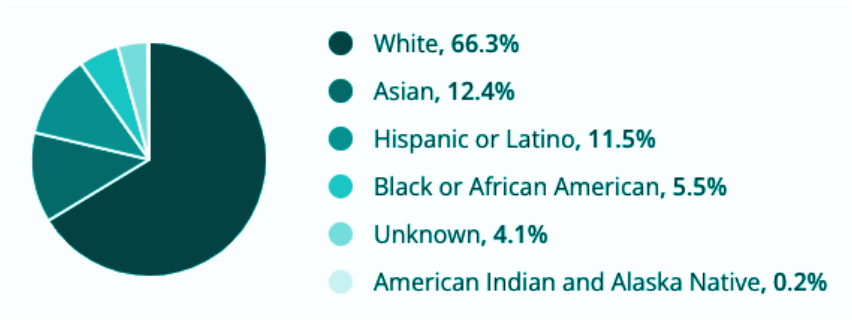
Further statistics show that women in day trading are actually decreasing over time.



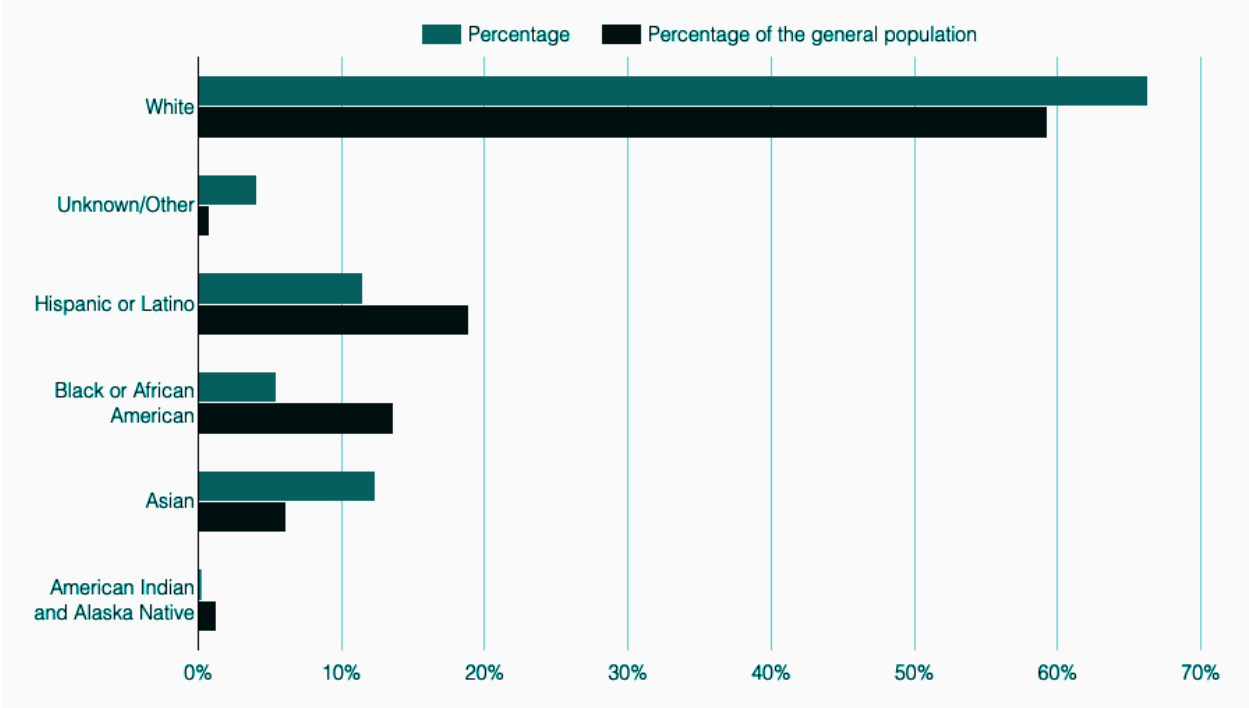
There are major discrepancies in age too, the average age being 43, leaning more towards the older population.



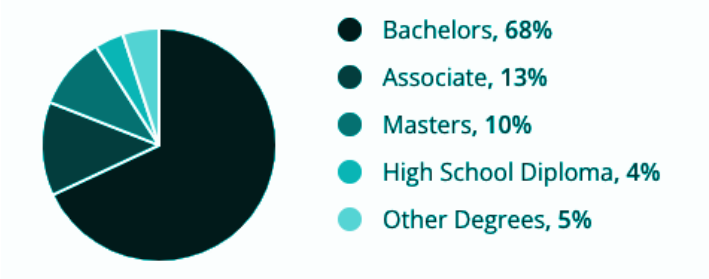
The percentage of white people in day trading compared to the rest is greatly overrepresented.



To put this into perspective, here are the different ethnicities by percentage of the US population, compared to the percentage participating in day trading.



The activity seems to discriminate between different levels of education as well- most traders have at least a bachelor's degree.



Another thing to note is that most people that pick up trading tend to come from the industry of finance, and rarely elsewhere. All statistics from zippia.

I believe such an app would help anyone - regardless of race, gender or background - be introduced into the world of trading by simplifying the process, and allowing them to maintain control over their own money rather than lending it to companies to invest it for them.

Gathering Insight

Interview 1

Interviewer: What are your thoughts on trading algorithms in general?

Interviewee: Well, I think algorithmic trading is a fascinating field, particularly for those who have a background in both finance and computer science. The potential for automating trades and capitalizing on small market inefficiencies can be quite attractive. It's exciting to think of an AI handling some aspects of trading that traditionally require human intuition.

Interviewer: That's a fair point. But where do you see the downside of trading algorithms?

Interviewee: There's always a downside, isn't there? The major concerns are probably the lack of transparency and the risk of over-reliance. It's sometimes hard to understand why a certain decision is made by the algorithm, and this can create trust issues. I think it would be helpful if it produced detailed reports of the trading decisions made and their outcomes.

Interviewer: That makes sense. In an ideal world, how could trading algorithms be improved to better suit your needs?

Interviewee: For me, the key would be more transparency and control. I'd like to understand the logic behind the decisions that the algorithm makes. I'd also like the ability to set boundaries on its operations. For example, being able to cap losses or limit the number of trades it can execute in a given time period. It would also be useful if the algorithm could predict stock prices with the use of machine learning.

Interviewer: Those are some insightful points. I'll definitely take them into consideration. Thanks again for your time.

Interview 2

Interviewer: What are your thoughts on trading algorithms?

Interviewee: They are great for quick decision-making and they take emotion out of trading. But they can also amplify market volatility. Flash crashes in the past are a testament to that. They cannot make decisions based on events that occur in the real world, like covid, for example.

Interviewer: True. What changes would make them more useful for you?

Interviewee: I'd appreciate it if they had more customization. Allowing me to adjust risk tolerance, set specific trading strategies or sectors, would make them more appealing. Also, they could incorporate social and environmental metrics, considering the rise of ethical investing.

Interviewer: Noted. Thanks for your feedback!

Research

Languages

Python: Python is a popular language in this area due to its simplicity and the variety of scientific and mathematical libraries available, such as pandas, NumPy, and scikit-learn. It's important to note that Python is not ideal for HFT or intra-minute trading, due to its execution speed, however can be used as a glue language to trigger code that runs in other languages (for example, NumPy is implemented in C).

R: R is another popular language for statistical analysis and data visualization. It may be used in combination with Python, especially for specialized statistical and graphical capabilities.

SQL: When working with large datasets, SQL for database management could be very useful.

SQLite: A lightweight, self-contained, serverless, and open-source relational database management system that comes with Python as a built-in module

C++/Java: Some high-frequency trading firms use lower-level languages like C++ or Java for their algorithms, due to their faster execution speed compared to Python. However, for most retail algorithmic traders, Python is sufficiently fast.

I chose Python since it is the language I have most experience with, and due to its extensive libraries specially for machine learning like TensorFlow, PyTorch, and Keras. For the implementation of databases, I decided to use SQLite due to its simplicity and ease of use.

APIs & Brokers

Interactive Brokers: Interactive Brokers offers an extensive API that supports multiple languages (including Python, Java, and C#), which makes it a popular choice for algorithmic trading. They also provide access to a wide range of markets and financial instruments. Known for its robust API, global access to stocks, options, futures, forex, bonds, and funds from a single integrated account, and relatively low trading costs. Interactive Brokers' Trader Workstation (TWS): Interactive Brokers offers a paper trading account that lets you use the full range of features in TWS. However, their API is known to have a bit of a learning curve.

Alpaca: Alpaca is a relatively new brokerage that is specifically focused on providing an easy-to-use API for algorithmic trading. They offer commission-free trading and support for several programming languages through their API. Alpaca is a popular choice for individual traders developing their own trading algorithms. Alpaca is a commission-free API-first stock brokerage platform. However, as of now, Alpaca only offers access to U.S. markets.

Alpa vantage: is a financial data provider that offers access to a wide range of real-time and historical market data, including stock quotes, technical indicators, cryptocurrencies,

and more. more impressively, AV also offers the latest market news & sentiment that can be filtered as needed.

Polygon: offers an API that developers can use to access market data, trade execution, and other financial services. Alpaca has integrated Polygon's real-time market data into their API, allowing Alpaca users to access this market information.

Here, I opted for Alpaca, which offers paper trading with an API, allowing me to test my algorithm in a simulated environment with no risk. It's free to use and is compatible with Python, making it a good option for me. The biggest problem with Alpaca would be their new python SDK for 2023, which has limited documentation compared to their previous one.

IDEs

Jupyter Notebook: This is an open-source web application that is great for data analysis and machine learning tasks. It allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It's not quite a full IDE, but it's excellent for exploratory work.

PyCharm: This is a full-featured IDE developed by JetBrains. It offers features like smart code navigation, fast and safe refactoring, and debugging. It has a professional version that is paid, and a community version that is free.

Visual Studio Code (VS Code): This is a free and open-source IDE developed by Microsoft. It supports Python development out of the box and has a rich ecosystem of extensions for other languages and development tasks. It also has integrated terminal and debugging tools, which are quite useful.

As for IDEs, I decided to use Jupyter Notebook for the data science part of the project, like testing statistical algorithms because of its relatively straightforward data visualization. I chose Pycharm as my main IDE due to features like the management of virtual environments, version control integration (git), and intelligent code assistance.

Key Terms

SMA: The simple moving average is a commonly used statistical calculation that helps smooth out fluctuations in data and identify trends. It is calculated by adding together a specified number of data points and then dividing the sum by the number of points.

EMA: Exponential Moving Average. It is a type of moving average that places more weight on recent data points, making it more responsive to recent price changes compared to the Simple Moving Average (SMA).

Close: refers to the final price at which a financial asset (such as a stock, commodity, or currency) is traded on a specific market for the day. It signifies the value at the end of the trading session and is used as a reference point for analyzing market trends and making investment decisions.

Bar: refers to a representation of a specific time interval or period of trading activity. It is a way of organizing and summarizing market data, typically for a specific financial instrument such as a stock, commodity, or currency pair. Usually consists of open, close, high, low and volume.

Log returns: Log returns, often referred to as logarithmic returns, are a way to measure the percentage change in the value of a financial asset over a period. They use the natural logarithm to transform absolute price changes into a relative scale, making them additive rather than multiplicative. This helps in analyzing investment performance and risk.

Strategies

First I had to choose what type of asset I was going to trade whether it be stocks, ETFs, crypto, commodities, futures, or currencies. Because I was already familiar with trading stocks in simulations, I chose stocks.

Time Frame

Predicting Short-Term vs Long-Term Price Movements:

- Short-term price prediction and trading are often considered more difficult. This is due to the high amount of 'noise' in short-term data, and because price movements over short periods can be influenced heavily by unpredictable factors (news events, etc.). You'd also need to deal with the transaction costs more frequently in short-term trading, which could erode your profits if not managed well.
- Long-term trading strategies, such as trend following or buy-and-hold, could be considered easier in terms of analysis and execution. They typically involve fewer transactions and thus lower transaction costs. However, they require patience and the ability to withstand short-term losses and periods of underperformance.

Next, I needed to decide on what time frames the bot would support. I wanted to support both long-term and short-term trading, and therefore decided on two separate bots with separate specialities.

Common Strategies

The effectiveness of a strategy over certain time frames depends on many factors like market conditions, the specific securities you're trading, and the parameters of the strategy

itself. That being said, certain types of strategies are generally better suited to certain timeframes:

- Short-Term Timeframes (1m - 15m bars):
    - Scalping: This strategy attempts to profit from small price changes. It's a decision-making process that requires quick thinking and the ability to act on changes in the market swiftly.
    - Mean-reversion: This is based on the concept that the high and low prices of an asset are a temporary phenomenon that revert to their mean value (average value) periodically. An example of this strategy is "Bollinger Bands".
    - Momentum (short term): This is similar to the longer-term momentum strategies, but shorter-term strategies rely more on the speed of the movement in prices rather than the size of the movement.
    - Statistical Arbitrage Strategies: These involve complex mathematical models to identify trading opportunities. They typically involve a large number of securities and trades and aim to profit from pricing inefficiencies.
    - High-frequency Trading Strategies: These involve making a large number of trades in a very short time period, often milliseconds. They exploit small price inefficiencies and are highly technology-dependent.
- Long-Term Timeframes (15m - daily bars):
    - Trend following (SMA): This type of strategy is based on the idea that prices tend to move in a particular direction over time. Traders using this strategy attempt to buy a security early in its trend and sell when they believe the trend is about to reverse.
    - Breakout strategies: These are strategies where trades are entered when the price moves above a certain level. It's assumed that the price will continue in the same direction after the breakout.
    - Momentum (long term): These strategies are based on the trend of a stock over a specific timeframe. The idea is to buy securities that have been rising in price and sell those that have been falling.
    - SVR (machine learning): effective for non-linear regression analysis, leveraging the power of support vector machines to capture intricate relationships in the data, making it suitable for various forecasting tasks, including time series prediction.
    - Sentiment Analysis Strategies: These strategies try to gauge market sentiment using various sources of data, such as social media, news, etc., and make trading decisions based on this sentiment.

I plan on using multiple strategies (underlined) and an overlap of buy signals to place a buy order, and a union of sell signals to place a sell order.

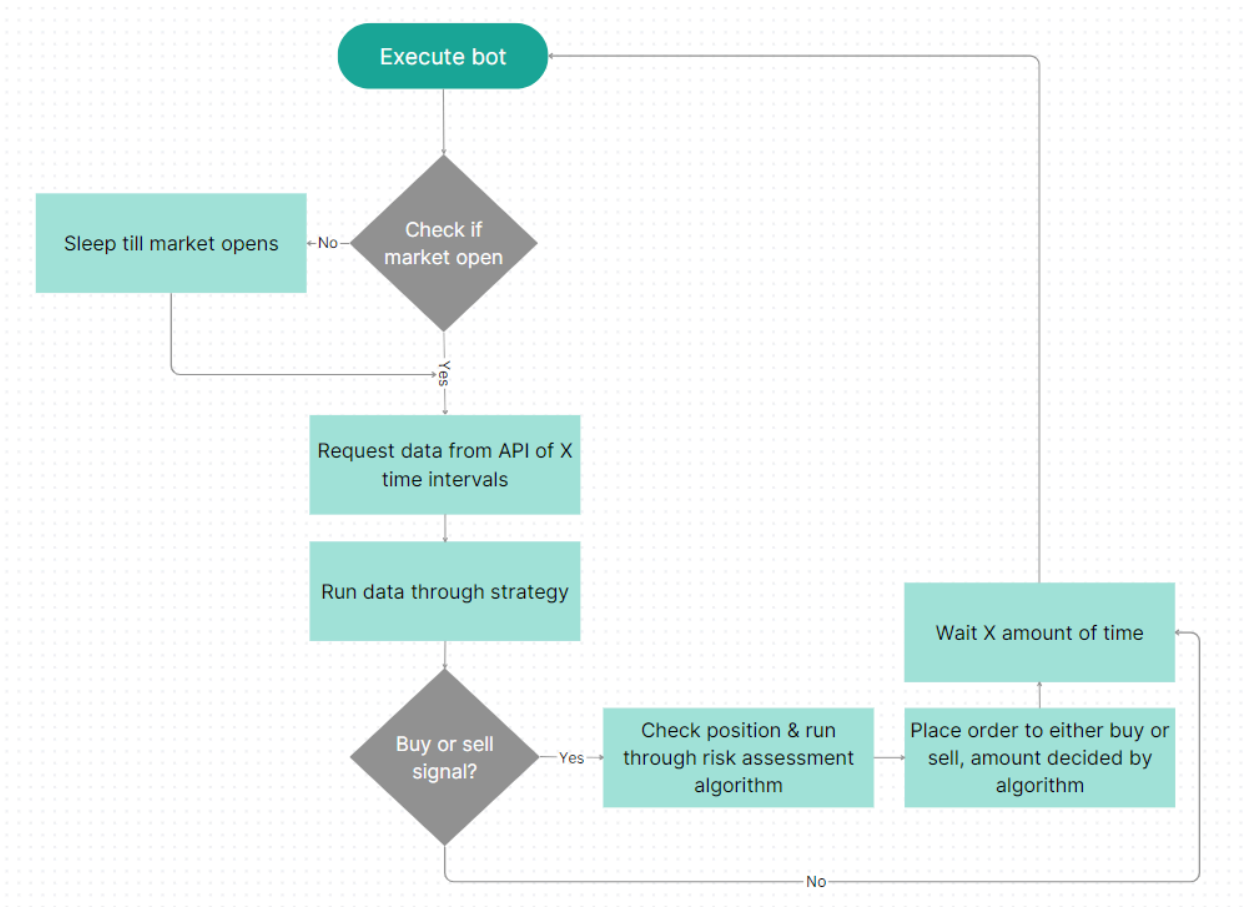
Risk

However, there are many more sophisticated methods to calculate position size based on factors such as:

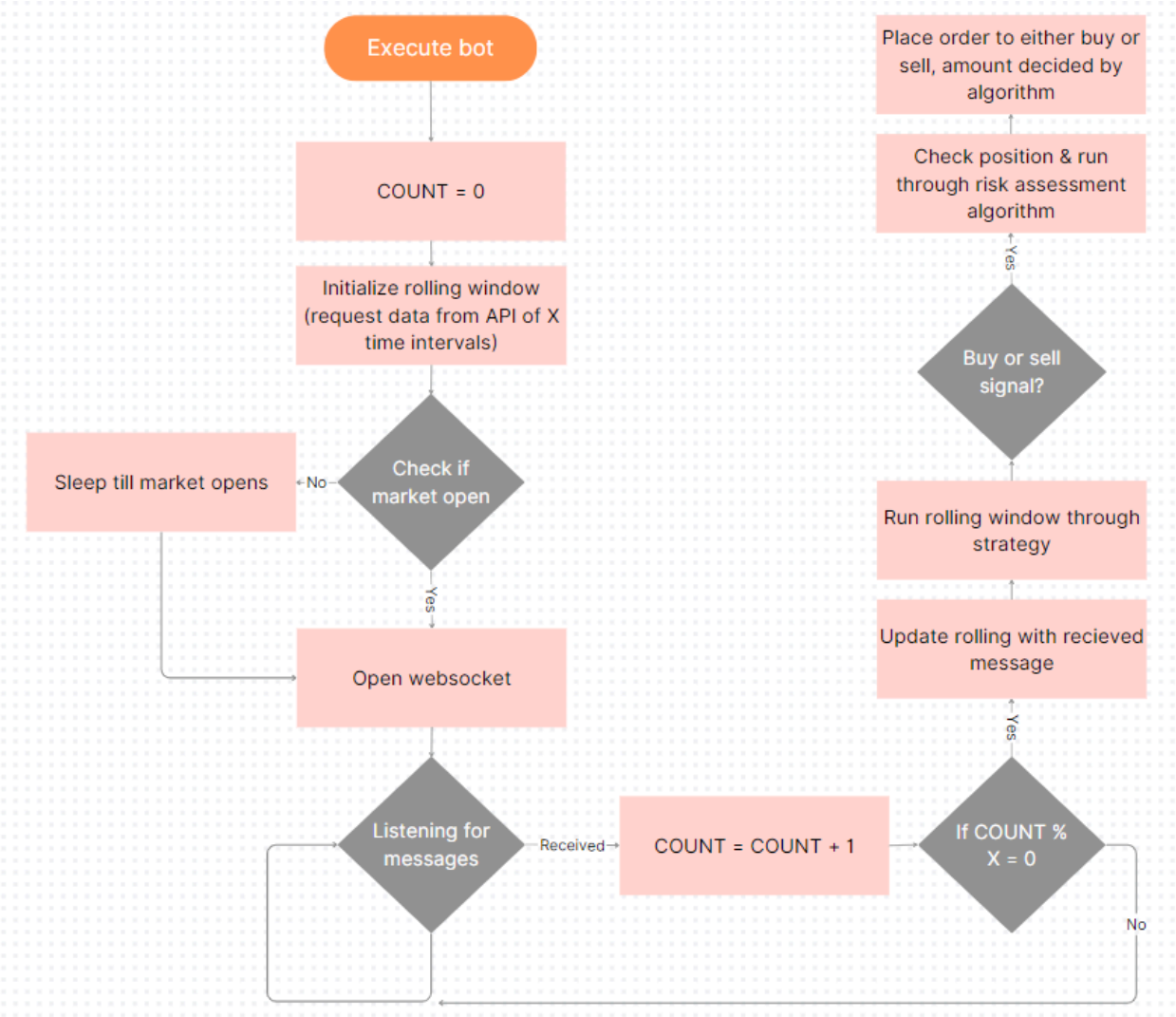
- Fixed Fractional: This approach involves investing a fixed fraction of your current account balance per trade. This helps manage risk by reducing the size of your trades as your account balance decreases.
- Equal Dollar Amount: In this method, the same dollar amount is invested in each trade. This results in buying more shares of a cheaper stock and fewer shares of a more expensive one.
- Risk Parity: In a risk parity strategy, positions are sized based on their risk, typically measured by volatility or standard deviation. More volatile stocks would have smaller position sizes, and less volatile ones would have larger position sizes.
- Kelly Criterion: This is a mathematical formula designed to maximize the growth of your portfolio over the long run. It considers both the winning and losing probabilities and the winning and losing payout ratios to calculate the optimal position size.
- Volatile Unit Size: This method adjusts the position size based on the volatility of the stock. If the stock price is highly volatile, fewer shares are bought and vice versa.
- Portfolio Optimization: Techniques like Modern Portfolio Theory (MPT) or mean-variance optimization can also be used to calculate optimal position sizes based on expected returns, volatilities, and correlations between different assets.

And finally for risk, I decided to use a mixture of methods like fixed fractional and risk parity to make my own risk function.

Design



Simplified overview of the historical bot



Simplified overview of the live bot

File overview for either bot (package):

- Mark as package (__init__.py): This file will be empty.
- Configuration (config.py): Takes inputs and assigns all variables a value. e.g. which stocks the user wants, time frames, etc along with the user's API keys. This will take no imports but will be imported into most files.
- Data request (data_grab.py): This file can contain the functions interacting directly with the Alpaca API. This might include functions to get price data, etc. This page will import only config.py and be imported into multiple files.
- Strategy Classes (x_strategy.py): I will define each strategy as a class in this file. Each strategy might have methods like generate_signals that take in price data and return a boolean indicating whether to buy or sell.

- Risk Management (risk.py): This can contain functions related to position sizing and risk management. For instance, it could have a function that calculates the number of shares to buy based on a risk tolerance and account balance.
- Main Bot Script (main.py): This file ties everything together. It should import the functions/classes from the other files, run the main loop, and include the logic for when to trade.
- Market check (market.py): This file could include a function to check if the market is currently open, or even a function to determine how long the bot should sleep if it isn't.

Objectives

1. APIs

1.1. Must use the user's API keys to request data //

1.1.1. Request stock data using Python SDK based API //

1.1.2. Request financial data using HTTP request-based API //

1.2. Must execute trades using the user's Alpaca account via python SDK-based API

2. Trading bots

2.1. User-defined time frame (1m - daily or more)

2.2. Iterates through multiple user-selected stocks

2.3. Each strategy must be tested for outperformance against buy and hold to be valid

2.4. Only buy/sell if all/any valid strategies confirm for that specific stock (AND/OR)

2.5. Multiple strategies implemented (selection must be adjustable)

2.6. Should be able to handle market being open/closed

2.6.1. Should be able to handle market opening/closing while running

2.7. Risk calculation returns a position size between maximum and minimum //

2.8. For longer time frames bots should be based on historical data //

2.8.1. At least one strategy must use NN/machine learning //

2.8.2. Position size must be based on correlation, volatility and sentiment //

2.9. For shorter time frames bots should be based on streamed data through a websocket connection //

2.9.1. Maintain a 'rolling window' dataframe //

2.9.2. Position size must be based on correlation and volatility //

2.10. Bot should have the option to be terminated through the press of a button

2.11. Information should be logged to inform the user

2.11.1. Transactions should be logged (placed a buy order for 7 shares of AAPL, etc.)

2.11.2. Bot status should be logged (sleeping, etc.)

2.11.3. Account balance should be logged

2.11.4. Log must have the option to be cleared through the press of a button

3. Detailed performance reports

3.1. At least two reports (graphs) for each analyzed stock-strategy combination

3.1.1. Strategy vs buy and hold for specific stock-strategy combination

3.1.2. Graph of calculated statistics for specific stock-strategy combination

4. GUI / Interface

4.1. 'Home' page as the welcome screen

4.1.1. Negative values of performance should be in red

4.1.2. Positive values of performance should be in green

4.1.3. Market status along with a countdown (till open/close) must be displayed.

4.1.4. Relevant user account information must be displayed (e.g. name, account balance, total profit/loss)

4.1.5. Relevant financial data must be displayed.

4.1.5.1. Watchlist of major market indexes (e.g. QQQ, SPY)

4.1.5.2. A graph of the S&P 500 [5Y]

4.1.5.3. Sentiment data for all major sectors

4.1.6. Financial data about a stock must be accessible upon user request.

4.1.6.1. Qualitative information about the stock

4.1.6.2. Performance values for the stock

4.1.6.3. Latest news articles relevant to the stock

4.1.6.4. Historical close price for the stock [5Y]

4.2. 'Bots' page as the second tab

4.2.1. Bot configuration must let the user manage which strategies are being used and which stocks are being traded.
4.2.1.1. Maximum stocks per configuration must be capped at 3
4.2.2. User must be able to delete configurations
4.3. User's will be transferred to the 'Running' page when a bot is running
4.3.1. This page should display the configurations of the bot that is currently running
4.3.2. 'Running' page should be disabled when a bot is not running
4.3.3. When a bot is running, other tabs (Home, Bots) should be disabled

5. Account system

5.1. A sign-up/login system must be implemented.
5.1.1. Usernames must be unique
5.1.2. Alpaca API keys must be validated before accepted.
5.1.3. Alpha Vantage API key must be validated before accepted.
5.1.4. If the username is found but the password is incorrect, the user should be notified.
5.2. Sensitive information must be protected
5.2.1. Passwords must be hashed before storing in .db file
5.2.2. API Keys must be encrypted before storing in .db file
5.3. User information must be saved to their respective ID
5.3.1. User details must be saved to their respective ID
5.3.2. User configurations must be stored to their respective ID
5.3.3. Only user configurations that belong to a specific user must be displayed
5.4. A user should be able to delete their account along with any information associated with it.
5.4.1. A confirmation message must be displayed if forget account is pressed

6. Validation & user experience

6.1. Should be able to handle invalid inputs (or lack of inputs) anywhere
6.1.1. Relevant and descriptive error messages must be displayed
6.2. Should be able to handle connection errors (regarding the websocket, etc.). //
6.3. Any required non-py files must be created if they don't already exist (.db, .txt, .key, etc.).
6.4. Users should only be allowed to interact with one input window of each type at once.

6.5. Should be able to handle API request timeouts.

6.5.1. Relevant and descriptive timeout messages should be displayed

6.6. Capitalisation should not matter (unless stocks are being entered for the bot).

6.7. While a bot is running, any actions interactions done by the user must not break the program (e.g. quit, forget account).

6.8. Upon pressing 'quit' the program should properly close any running bots, and close the program safely.

6.8.1. Pressing the x button to close the window should have this same effect.

6.8.2. Pressing the x button on the account system should also safely quit the program.

Documented Design

APIs

Alpaca

Interacting with Alpaca will mean using Alpaca's python SDK, the downloadable module 'alpaca-py'. Alpaca-py provides an interface for interacting with the API products Alpaca offers. These API products are provided as various REST, WebSocket and SSE endpoints.

For placing orders:

```
from alpaca.trading.client import TradingClient
from alpaca.trading.requests import MarketOrderRequest
from alpaca.trading.enums import OrderSide, TimeInForce

trading_client = TradingClient('api-key', 'secret-key', paper=True)

# preparing order data
market_order_data = MarketOrderRequest(
                    symbol="BTC/USD",
```

```
                    qty=0.0001,
                    side=OrderSide.BUY,
                    time_in_force=TimeInForce.DAY
                )

# Market order
market_order = trading_client.submit_order(
                order_data=market_order_data
                )
```

Bots of different time frames will use different order types:

- For longer time frames, GTC will be used. This means the order stays active until it's executed or manually canceled by the trader, potentially indefinitely.
- For shorter time frames, IOC will be used. This Requires an order to be executed immediately or canceled if it can't be filled immediately.

The parameter `paper` determines whether the trades are executed in simulation (if set to True) or in the real market (if set to False or the parameter isn't specified at all). For the purpose of this NEA, `paper` will always be set to True.

For requesting data:

```
from alpaca.data.historical import CryptoHistoricalDataClient
from alpaca.data.requests import CryptoBarsRequest
from alpaca.data.timeframe import TimeFrame
from datetime import datetime

# no keys required for crypto data
client = CryptoHistoricalDataClient()

request_params = CryptoBarsRequest(
                        symbol_or_symbols=["BTC/USD", "ETH/USD"],
                        timeframe=TimeFrame.Day,
                        start=datetime.strptime("2022-07-01", '%Y-%m-%d')
                        )

bars = client.get_crypto_bars(request_params)
```

```
# convert to dataframe
bars.df
```

Alpha Vantage

Alpha Vantage on the other hand uses direct HTTP request-based. This approach involves constructing the API URL and making HTTP requests directly, usually with a library like requests in Python. These requests return data in a json format.

Example data fetch:

```
import requests # replace the "demo" api key below with your own key from https://www.alphavantage.co/support/#api-key

url = 'https://www.alphavantage.co/query?function=TIME_SERIES_INTRADAY&symbol=IBM&interval=5min&apikey=demo'

r = requests.get(url)

data = r.json()

print(data)
```

Example JSON response:

```
{
    "Meta Data": {
        "1. Information": "Intraday (5min) open, high, low, close prices and volume",
        "2. Symbol": "IBM",
        "3. Last Refreshed": "2023-11-10 19:55:00",
        "4. Interval": "5min",
        "5. Output Size": "Compact",
        "6. Time Zone": "US/Eastern"
    },
    "Time Series (5min)": {
        "2023-11-10 19:55:00": {
            "1. open": "148.7100",
            "2. high": "148.9800",
            "3. low": "148.7100",
            "4. close": "148.9800",
```

```
            "5. volume": "53"
        },
        "2023-11-10 19:50:00": {
            "1. open": "148.8100",
            "2. high": "148.9700",
            "3. low": "148.8100",
            "4. close": "148.9700",
            "5. volume": "51"
        },
    }
}
```

Overview

Structure

There will be two bots to choose from: historical and live. Both work using different methods. These bots will be treated as a 'package', through the addition of an empty __init__.py file in their respective folders. Everything will be wrapped up in a tkinter GUI, with added functionality. I will be using a custom module of tkinter, namely, CustomTkinter. This will have the same functionality as tkinter, with only the appearance of the widgets being different (more modernized).

Modules used in this project will include:

- Standard library:
    - datetime
    - time
    - os
    - json
    - sqlite3
    - threading
    - multiprocessing
    - math
- External:
    - requests
    - numpy
    - pandas

- yfinance
- matplotlib
- pytz
- keras
- sklearn
- websocket
- customtkinter

Overall file layout:

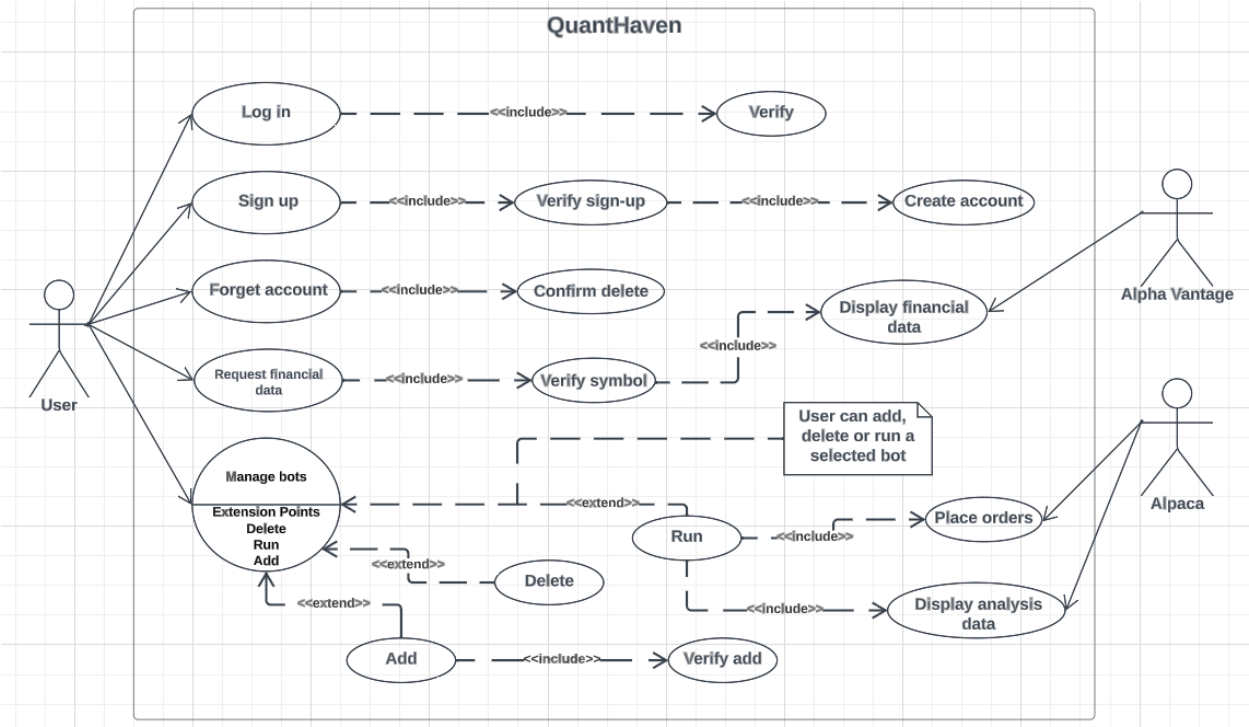
- pythonProject:
  - main.py: Account system and app root window.
  - home.py: Home page, imported by main.py.
  - popup.py: Grabs data for the pop up window in the home page through API. Imported by home.py.
  - home_datagrab.py: Grabs data for home page through API. Imported by home.py.
  - bot.py: Bots page. Displays bot configurations for the current user and lets them manage it (add, delete, run). Imported by main.py.
  - run.py: This page is disabled until 'run' is pressed on the Bots page. Displays performance reports (matplotlib) and logs transactions. Imported by main.py.
  - security.py: Includes any methods relating to the vernam encryption. Imported into anywhere that requires API key encryption or decryption.
  - task.py: Running this program runs the desired bot. Chooses between historical and live bot. Imported into main.py.
  - keys.key: Stores keys needed for the vernam cipher in security.py.
  - database.db: Stores all three tables used in the account system.
  - bot_configuration.txt: Used so that multiple files can retrieve the bot configuration of the bot that is currently running. Overwritten when a different bot is run.
  - historicalbot (package):
    - __init__.py: Empty
    - hist_config.py: Includes a single function that retrieves the necessary data for the bot to run. Imported into multiple files.
    - hist_datagrab.py: Requests data from alpaca and alpha vantage.
    - hist_main.py: Main loop. Imports all other files. Trade orders are placed here. Returns performance reports and transaction log.
    - hist_market.py: Checks if the market is open, calculates the time needed to sleep for.
    - hist_risk.py: Calculates the number of stocks to buy based on volatility, correlation (with rest of the portfolio), and sentiment.

- hist_x_strategy.py: Generates buy/sell signals, returns performance values needed for risk calculations, returns two performance reports (pandas df).

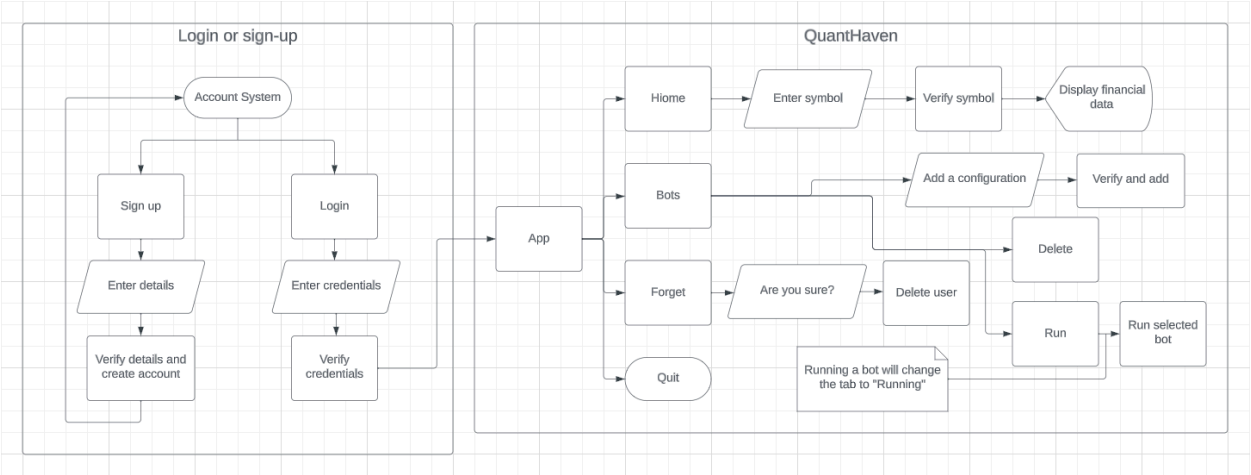
- livebot (package):
  - __init__.py: Empty
  - live_config.py: Includes a single function that retrieves the necessary data for the bot to run. Imported into multiple files.
  - live_datagrab.py: Requests data from alpaca.
  - live_main.py: Main loop. Imports all other files. Trade orders are placed here. Returns performance reports and transaction log.
  - live_market.py: Checks if the market is open, calculates the time needed to sleep for.
  - live_risk.py: Calculates the number of stocks to buy based on volatility and correlation (with rest of the portfolio).
  - live_x_strategy.py: Generates buy/sell signals, returns performance values needed for risk calculations, returns two performance reports (pandas df).

GUI

Below is a UML use case diagram to show the relationship between the user and the APIs.



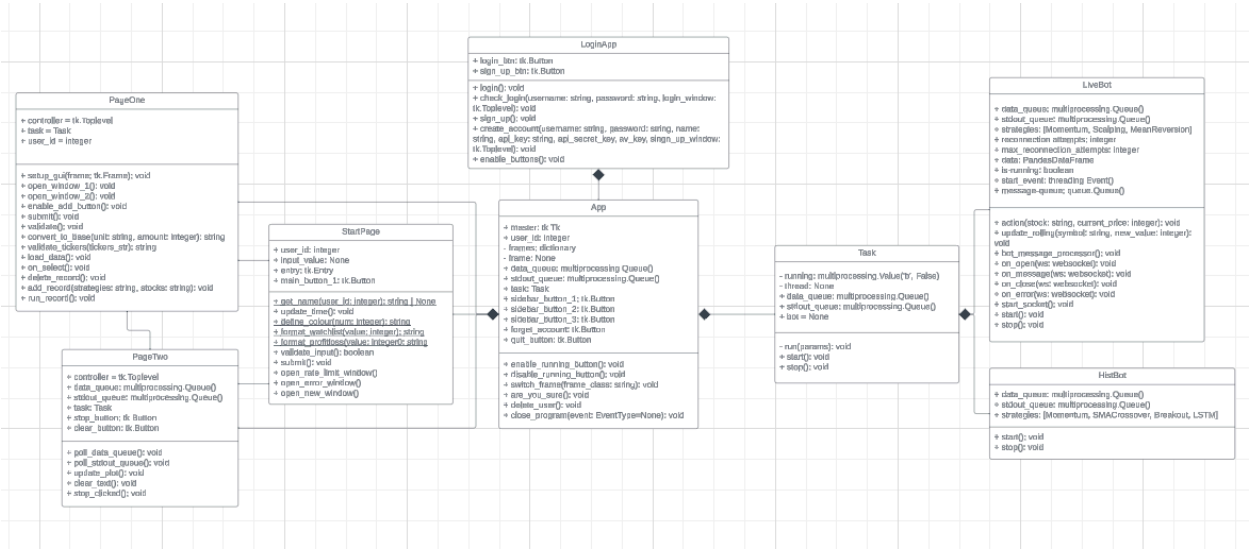
Alpaca is used for requesting data for analysis and placing orders. Alpha vantage will be used to get data for display in the home page, and for sentiment data in the historical bot. When the user first opens the app, they will be met with a signup/login screen, where they can either create a new account or proceed with existing credentials. After 'Login' is pressed, any windows related to the account system will close, and the main application will open. This is represented in the user flow diagram below.



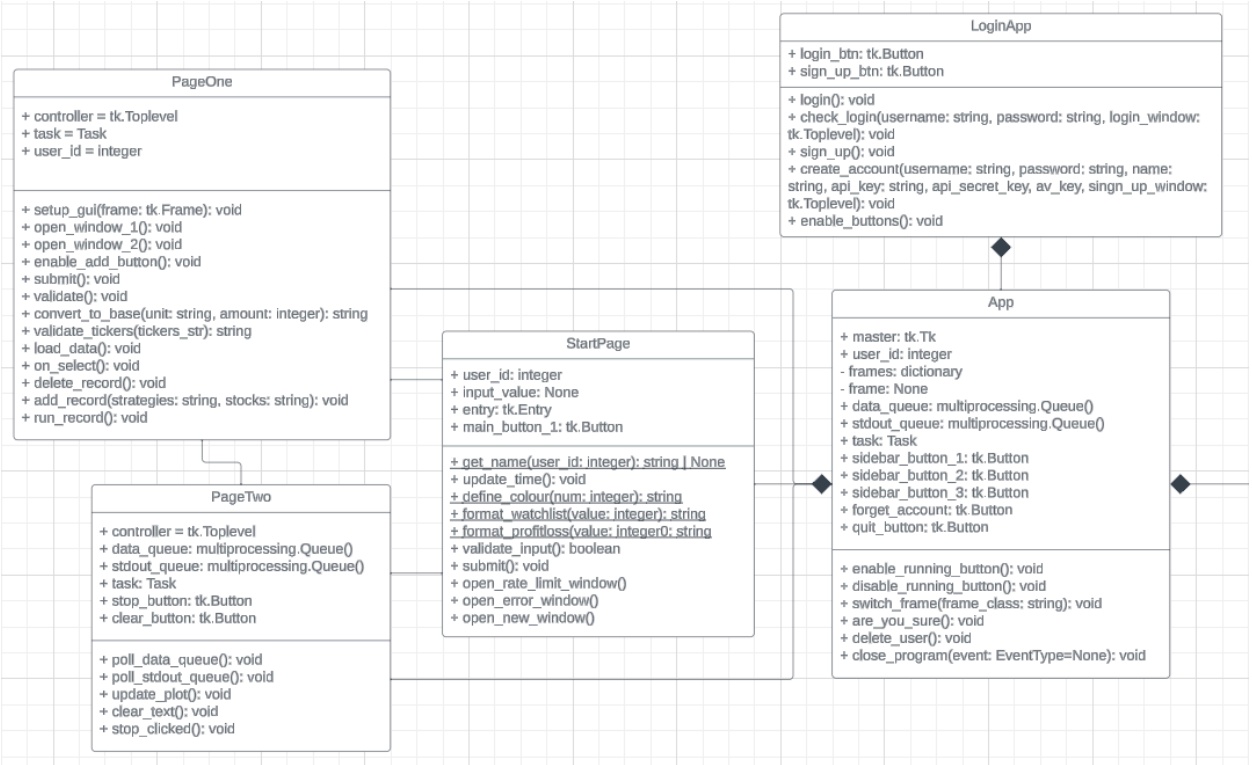
The main app will have a sidebar to navigate between two pages: 'Home' and 'Bots'. 'Running' will be initially disabled. The Home page will display data about the user's alpaca account (balance, net profit, etc.), and complementary financial data about top gainers, and major index funds like S&P 500 or NASDAQ. The Home page will allow a user to enter a symbol, which, if valid, will open a pop up window of financial data about the stock, and an error message otherwise. The Bots page will display a table using tkinter's 'Treeview'. The user will be able to add a configuration (row) by entering a time frame (time unit and amount). A popup will then open with checkboxes of strategies and an entry for symbols (separated by ', '). The window will be different (different strategies) depending on whether the time frame indicates a live or historical bot. After selecting a row, a user must be able to delete, or run a bot.

Once a bot is selected to run, the tab will switch to 'Running'. All other tabs will be disabled. until the 'Stop' button is pressed on the Run page. The user will still be able to forget the account (which will delete the user, their bot configurations, stop the bot and close the program), or press 'Quit' (which will stop the bot and close the program). Pressing the x button to close the window will have the same effect. The Run page will display performance reports from the bot along with transaction logs (e.g. 'Placed a buy order for 7 shares of AAPL').

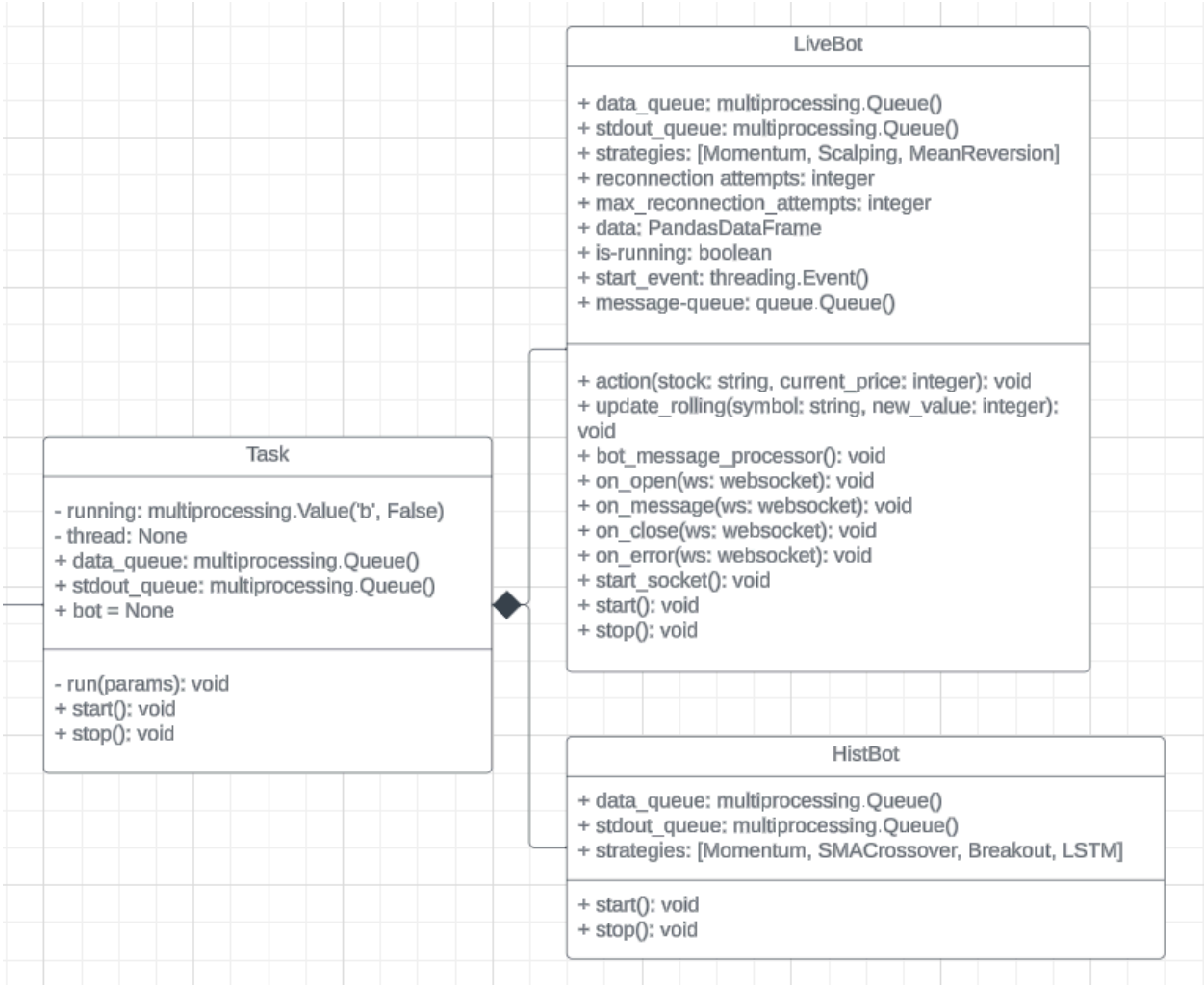
My solution will incorporate object-oriented programming, as I believe this paradigm compliments the nature of my solution, combined with the fact that the customtkinter library relies on this too. And so, I will use UML class diagrams to represent parts of my code like shown below.



Overview



GUI (left side)



Bot (right side)

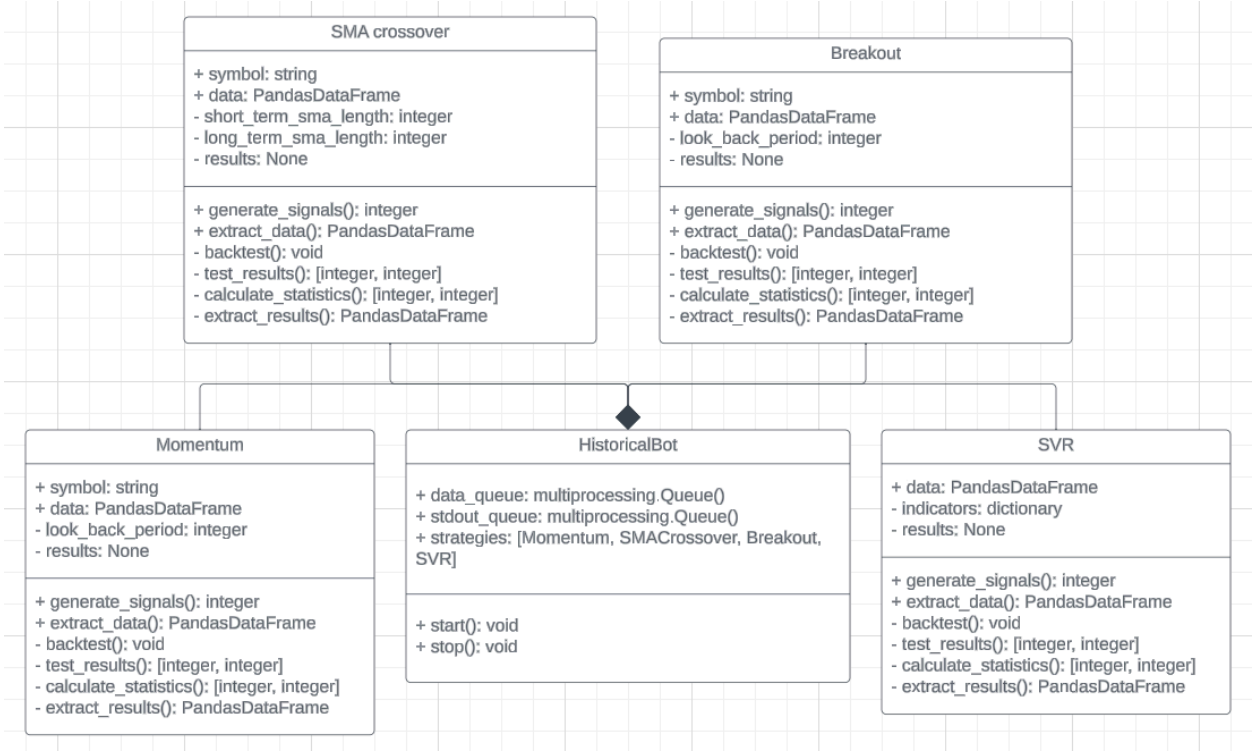
The LoginApp class is the tkinter root window. The App class is a Toplevel from this root window. It will import the three 'pages' and raise and hide them when requested by the user. Below is pseudo code to represent this logic.

```
FUNCTION switch_frame(frame_class):

1. GET frame from _frames with key frame_class, default to None

2. IF frame is None:

    a. IF frame_class is 'PageOne':

        - GET current_user_id using get_user_id(username, password)

        - CREATE PageOne instance and assign to frame

    b. ELSE IF frame_class is 'PageTwo':

        - CREATE PageTwo instance and assign to frame
```

```
    c. ELSE:
        - CREATE StartPage instance and assign to frame
    d. STORE frame in _frames with key frame_class
3. IF _frame exists:
    - HIDE _frame using grid_forget()
4. SET _frame to frame
5. DISPLAY _frame at specified grid position
```

Starting/stopping `Task()` will start/stop a required bot (between live and historical), and will use the multiprocessing module to run alongside the GUI without it freezing. `Task()` acts as a way for the GUI to communicate with the bot packages. Two queues will be utilized here: stdout_queue and data_queue. The data queue will be used to pass performance data in the form of pandas dataframes, to be displayed on the GUI in the Run page (matplotlib). The stdout queue will be used to print transaction logs onto the Run page by inserting strings into a 'Textbox' widget.

Historical bot



Within the app, the historical bot is defined as a bot with a time frame greater than 15 minutes. The Historical bot uses historical data and works by requesting historical bars with

a time interval of x, every x amount of time that goes by. All available strategies will be imported into the main.py file, and the ones selected by the user will be used. This will be done through using globals to return the strategy classes that match the selection imported from config like so: `self.strategies = [globals()[name] for name in self.configurations[3]].`

config.py will read from the bot_configuration.txt file (for bot settings) and the details table (for API keys). The start function of hist_main enters into a while loop, checks if the market is open, or sleeps til the market is open, then continues. Each stock is then looped through, and for each stock, each strategy is looped through. Strategies are first validated through backtesting of the same time frame, to see if they would give any profit. Then, if all of the valid strategies returned a buy signal (1), a buy order would be placed. If they all returned a sell signal, (-1), a sell order would be placed (For the purpose of testing 'all' will be replaced with 'any' to increase the number of trades). The account balance is outputted after each loop of a stock. The historical bot will exclusively take time stamps of when a loop starts to reduce it from the time to sleep till the next loop (for `x` amount of time). This is important because some strategies may use machine learning, which even for smaller data sets may take a significant amount of time to return data.

Live bot

Historical bars have a 15 minute delay, which is significant when the time frame is smaller. This is the reason for the live bot adopting a completely different structure. The Live bot uses streamed data through a websocket connection and works by first requesting historical bars with x time interval (df of fixed size), then appending every `x`th quote/bar received, and deleting the oldest bar. Effectively a rolling window of data with no delay.

The main.py for the live bot will be slightly more complex than that for the historical one. Threading will be used. For loops (looping through the stocks with strategies nested within) share similar logic to the historical bot, and are placed within a function, say, `action()`. Multiple functions control the authentication and connection to the websocket upon `start()`. They will subscribe to the stocks selected by the user. This means every minute, quotes will arrive for each stock subscribed to. The `on_message()` procedure will put the `x`th received message onto a queue. The queue is then processed at `bot_message_processor()` where it checks if it is in fact a quote message, then retrieves the symbol and close price. the rolling window is then updated with this information, and right after, `acton()` is called.

Strategies are slightly differently structured compared to the historical bot, as each live strategy will take the rolling window as a parameter as the data to be analyzed, while for historical strategies, datagrab.py is imported into each strategy and data is requested from there.

Strategies

All strategies will follow the same structure. This structure is represented in the table below for the xth strategy class.

strategyx The strategyx module allows the program to import the xth strategy, to analyze data, generate signals and performance statistics for a given stock.			
Procedure/Function	**Description**	**Parameters**	**Output/return value**
validation()	Unique to historical strategies, for purposes of error handling.	None	Boolean
generate_signals()	Generates a buy, sell or hold (1, -1 or 0) signal based on data analysis	None	Integer
extract_data()	Extracts certain columns (to put into queue) to later graph the indicators relevant to the strategy	None	Pandas DataFrame
backtest()	Calculates the strategies returns for the given time period	None	Updates dataframe
test_results()	Does the strategy outperform against buy and hold? Returns outperformance value.	None	Float

calculate_statistics()	Returns the average return and volatility (standard deviation) for the given time frame.	None	Float
extract_results()	Extracts certain columns (to put into queue) to later graph the strategy's returns against buy and hold.	None	Pandas DataFrame

Here is a generalized structure in pseudocode:

```
CLASS StrategyClass
    FUNCTION __init__(symbol, OPTIONAL parameters related to specific strategy)
        SET symbol TO self.symbol
        FETCH data FOR symbol AND SET TO self.data
        SET strategy-specific parameters
        INITIALIZE results AS None
    FUNCTION generate_signals()
        CALCULATE necessary indicators using parameters
        GENERATE buy and sell signals based on strategy's logic
        COMBINE signals to get overall signal
        REMOVE rows with missing data
        RETURN most recent signal
    FUNCTION extract_data()
        EXTRACT relevant data columns (such as close price, and calculated indicators)
        STORE extracted data in a structured format
        RETURN extracted data
    FUNCTION backtest()
```

```
        CALCULATE returns based on strategy's logic
       REMOVE rows with missing data
        SET 'position' column based on buy and sell signals
    FUNCTION test_results()
        CALCULATE strategy's returns based on backtest results
        REMOVE rows with missing data
        CALCULATE cumulative returns for strategy and possibly a benchmark
like buy-and-hold
        STORE results in self.results
        RETURN performance metrics
    FUNCTION calculate_statistics()
        IF no data OR 'strategy' not found in data columns
            PRINT error message
            EXIT function
        CALCULATE key statistics (e.g., annualized return, standard
deviation, etc.)
        RETURN calculated statistics
    FUNCTION extract_results()
        IF results is None
            PRINT error message
            EXIT function
        EXTRACT strategy performance metrics or results
        RETURN extracted results
END CLASS
```

For live strategies, a new `data` parameter will be required, as data won't be fetched within the class itself.

SVR (Support Vector Regression)

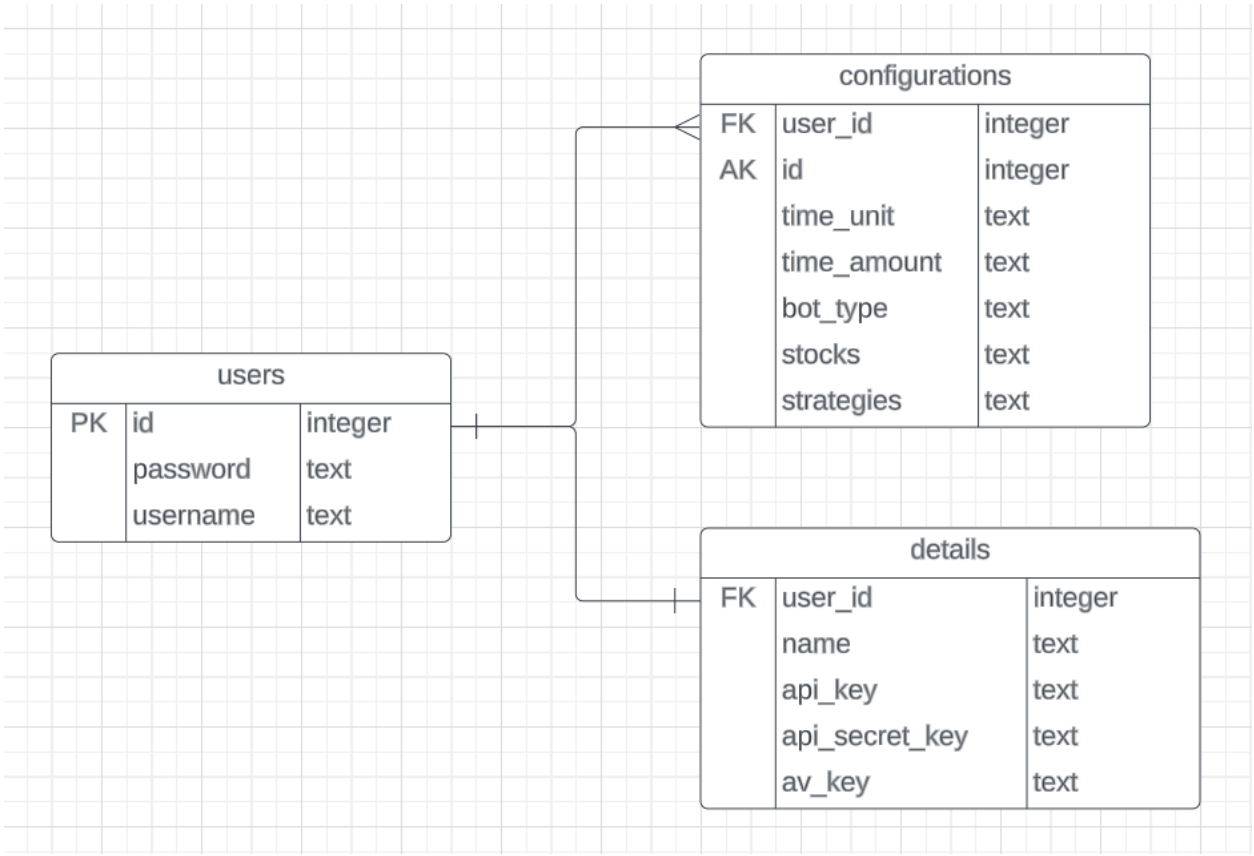
The historical bot will integrate at least one machine learning strategy. This may use SVR. While the structure still stays the same, the `generate_signals()` function will be very different (although returning the same values).

```
FUNCTION generate_signals():
    // Calculate technical indicators based on indicator list
    FOR EACH indicator, length IN indicators:
        IF indicator IS "RSI":
            CALCULATE RSI for data using the given length and STORE in 'RSI'
column
        ELSE IF indicator IS "EMAF":
            CALCULATE EMA for data using the given length and STORE in 'EMAF'
column
        ELSE IF indicator IS "EMAM":
            CALCULATE EMA for data using the given length and STORE in 'EMAM'
column
        ELSE IF indicator IS "EMAS":
            CALCULATE EMA for data using the given length and STORE in 'EMAS'
column
    // Process the data
    SHIFT 'close' column by -1 and STORE in 'targetnextclose'
    REMOVE columns 'volume', 'trade_count', 'vwap', 'symbol' from data
    REMOVE any rows with missing values from data
    // Split data into independent and dependent variables
    X = all columns of data EXCEPT 'targetnextclose'
    y = 'targetnextclose' column of data
    // Split X and y into training and test datasets
    SPLIT X and y into X_train, X_test, y_train, y_test with 20% as test
data, using random state 42
    // Initialize and train the model
    CREATE a pipeline with MinMaxScaler and SVR with C=1.0, epsilon=0.2
    TRAIN the model using X_train and y_train
    // Predict using the trained model
    y_pred = PREDICT using X_test
    // Store results in a dataframe
```

```
    CREATE a dataframe with columns 'Close' from y_test and 'Pred Close' from
y_pred
    // Generate buy and sell signals based on prediction
    BUY signal is when 'Pred Close' for next day > 'Close'
    SELL signal is when 'Pred Close' for next day < 'Close'
    COMBINE buy and sell signals into a single 'signal' column
    REMOVE any rows with missing values from results
    RETURN the last value in 'signal' column
END FUNCTION
```

Account system

The app will utilize a simple, three table account system using sqlite. This is represented by the ER diagram below.



Data dictionary is shown below.

User Table:

Column Name	Data Type	Description
id (PK)	Integer	Primary Key (auto incremented)
username	Text	User's username
password	Text	User's password

Details Table:

Column Name	Data Type	Description
user_id (FK)	Integer	Foreign Key to User Table (id)
name	Text	User's name
api_key	Text	API key associated with the user
api_secret_key	Text	API secret key associated with the user
av_key	Text	AV key associated with the user

Configurations Table:

Column Name	Data Type	Description
user_id (FK)	Integer	Foreign Key to User Table (id)
id (AK)	Integer	Auto-incremented, unique identifier for the bot
time_unit	Text	Unit of time for bot operation (e.g., Hour, Day)
time_amount	Integer	Amount of specified unit (trading frequency)
bot_type	Text	Type of bot (historical or live)
stocks	Text	Stocks associated with the bot (e.g., AAPL, MSFT)
strategies	Text	Strategies employed by the bot (e.g, Momentum, SMACrossover)

The `users` table will store a user's password and username, and generate a user id using auto increment. This table will be used to check credentials when logging in. The `details` table will hold information about a user, and will have a foreign key `user_id`, linked to id in the `users` table. These tables have a one to one relationship. This table will store a user's name (needed for display on the home page), and api keys, which will be encrypted. These

api keys will need to be retrieved whenever data is requested or a trade order is being placed. The `configurations` table has a one to many relationship with the `users` table, with the foreign key `user_id` and alternate key `id` which is auto incremented and is unique for each row. Each user can have multiple configurations saved. There can be multiple users at a time. The foreign key is used to know which user's configurations to display, and also needed when clearing a user's data when the user presses "forget account".

For clearing a user's configurations along with the user, "ON DELETE CASCADE" will need to be used in the creation of the configurations table. This will look something like this:

```
CREATE TABLE IF NOT EXISTS configurations(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER,
    time_unit TEXT,
    time_amount TEXT,
    bot_type TEXT,
    stocks TEXT,
    strategies TEXT,
    FOREIGN KEY(user_id) REFERENCES users(id) ON DELETE CASCADE
    )
""")
```

The bot configuration table in more detail:

- `id`: Unique identifier for each configuration, auto incremented.
- `user_id`: Foreign key referencing users (user_id)
- `time_unit`: This will either be 'Minute', 'Hour', or 'Day'.
- `time_amount`: This will be an integer.
- `bot_type`: This will be calculated depending on time_unit and time_amount. If the time frame is < 15 minutes, the value stored will be 'Live' and otherwise 'Historical'.
- `stocks`: A string listing the stocks involved in this configuration. Each stock is expected to be separated by a comma then space (', ').
- `strategies`: A string listing the strategies involved in this configuration. Each strategy will be separated by a comma then space (', '). Strategies will be chosen through checkbox selection then converted to string.

Visual aspect

As the aim of the project is to make trading more accessible, it was vital that the GUI was intuitive and simple. I went for a clean, modern and minimal aesthetic with a turquoise

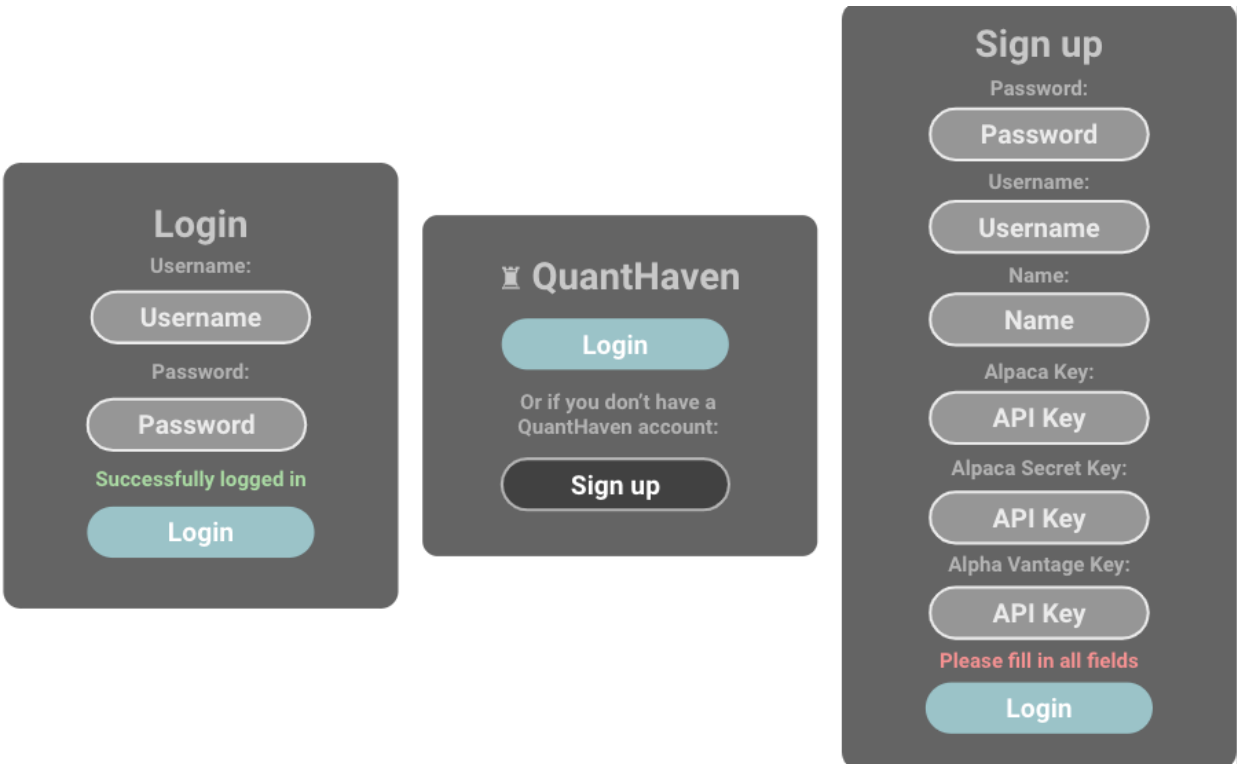
theme. I chose 'QuantHaven' as a placeholder name, with a Unicode rook "♖" (U+265C) as a placeholder logo. I also stuck to a consistent color scheme to make the app as user friendly as possible. To emphasize on the minimal aspect, the dedicated font for this project is Roboto Bold.

Colours

```
[ ] #A2C4C9 - light cyan 2
[ ] #6FA8DC - dark cyan 2
[ ] #000000 - black
[ ] #434343 - dark gray 4
[ ] #666666 - dark gray 3
[ ] #999999 - dark gray 2
[ ] #B7B7B7 - dark gray 1
[ ] #CCCCCC - gray
[ ] #D9D9D9 - light gray 1
[ ] #EFEFEF - light gray 2
[ ] #F3F3F3 - light gray 3
[ ] #B6D7A8 - light green 2
[ ] #EA9999 - light red 2
```

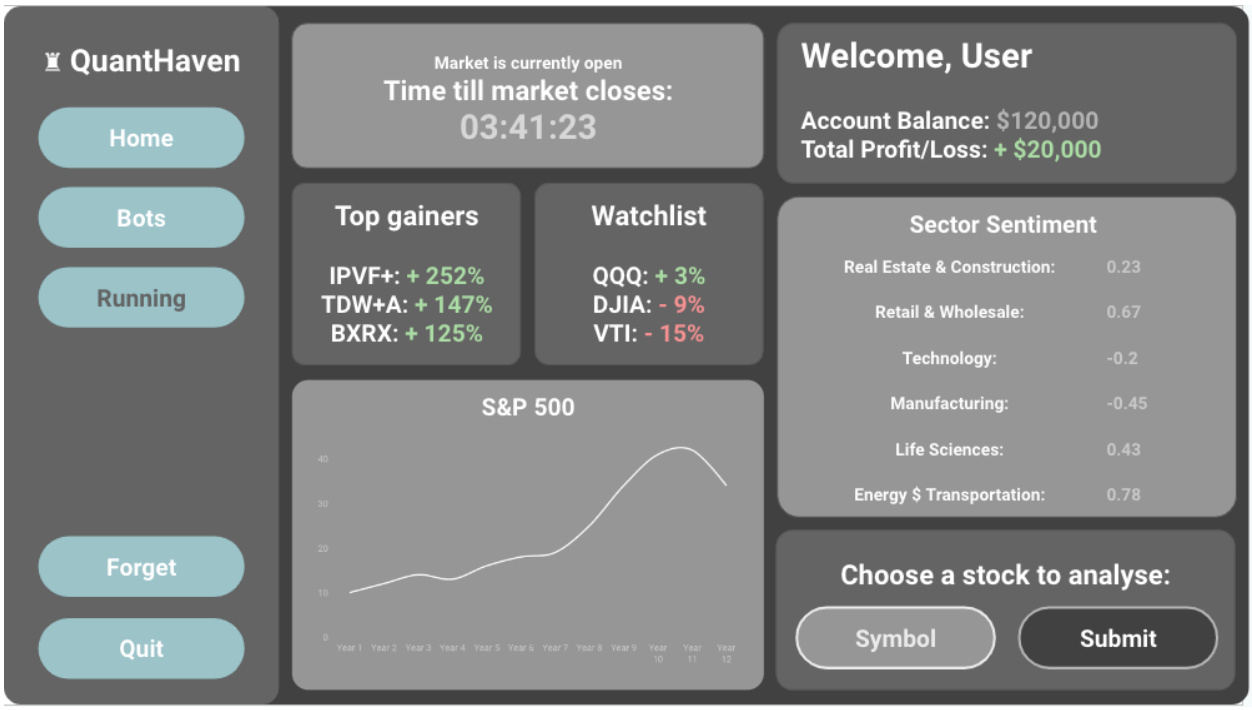
Windows will not be resizable, and will be displayed in the middle of the screen. Main window geometry: 1100 x 580 pixels. Pop up windows will be of variable sizes. The following images show drawings of what each window will look like/be designed after.

1. Account Pages

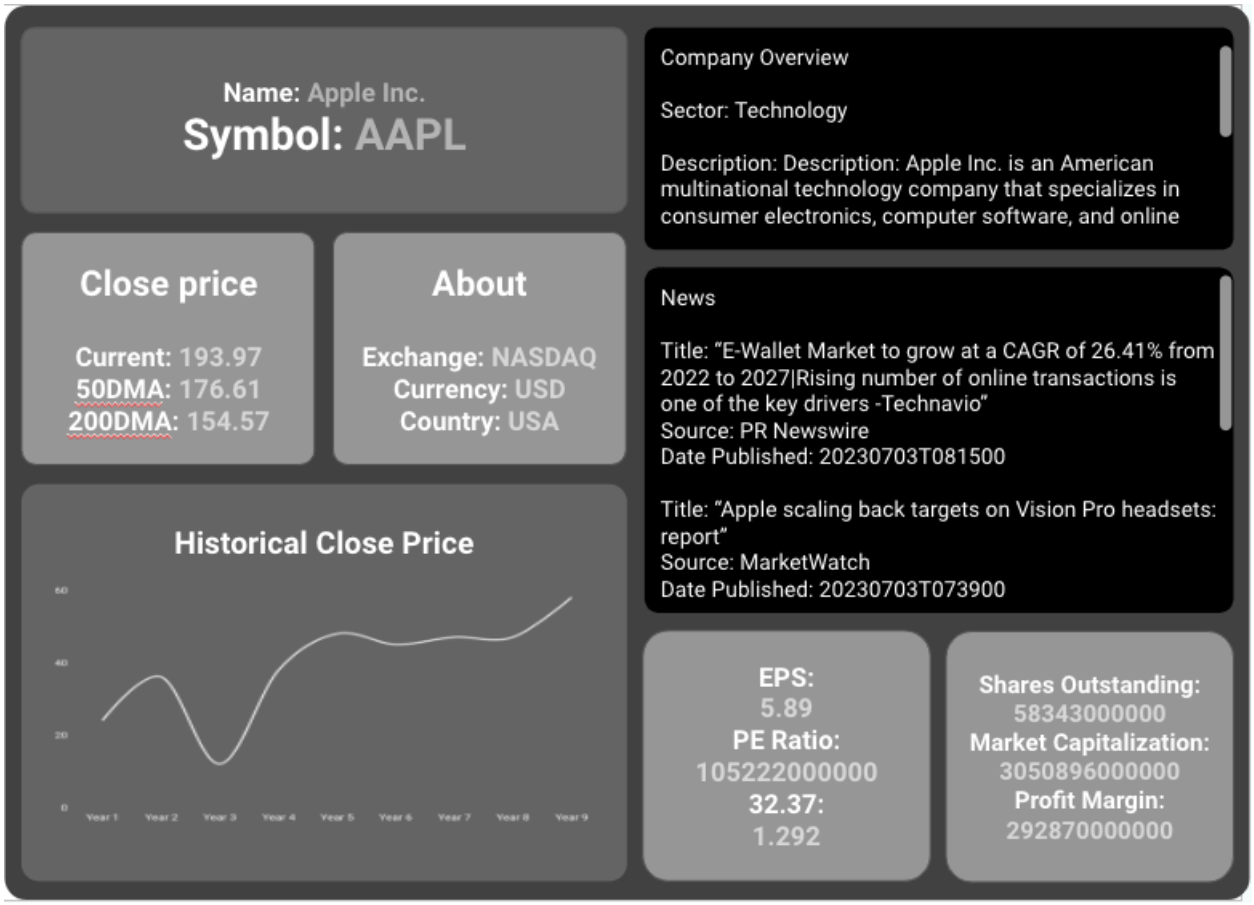


All entries will be validated. Multiple login/sign up pages cannot be open at the same time (once one is open, the button will be disabled). All account pages will hide once logged in.

2. The Home Page

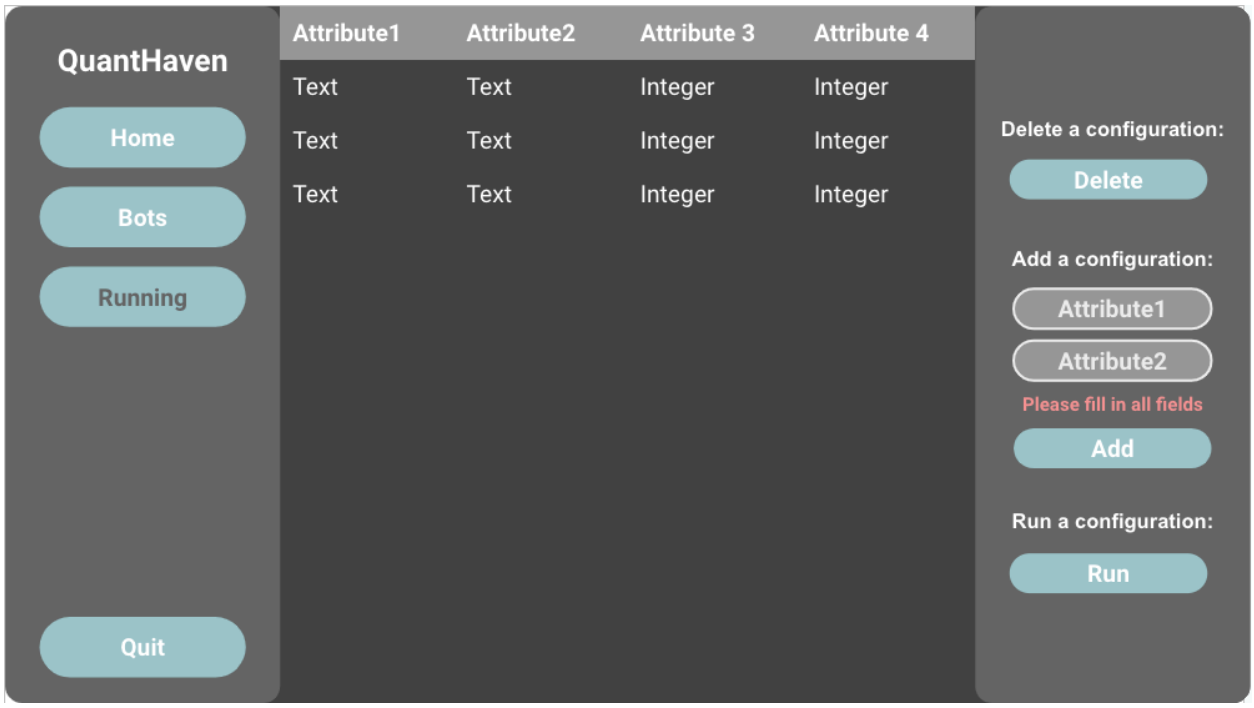


Countdown is updated live, all other widgets are updated only once as the app is launched. This is because once all pages are created, they are hidden and raised rather than destroyed and recreated. The Home page will have 3 API calls all together: 1 Alpaca API call for account balance, 2 Alpha Vantage API calls for financial data (Top gainers endpoint, News and sentiment endpoint). Name will be displayed in title case. The popup window displayed when a symbol is submitted is shown below. Matplotlib will be used for graphs.



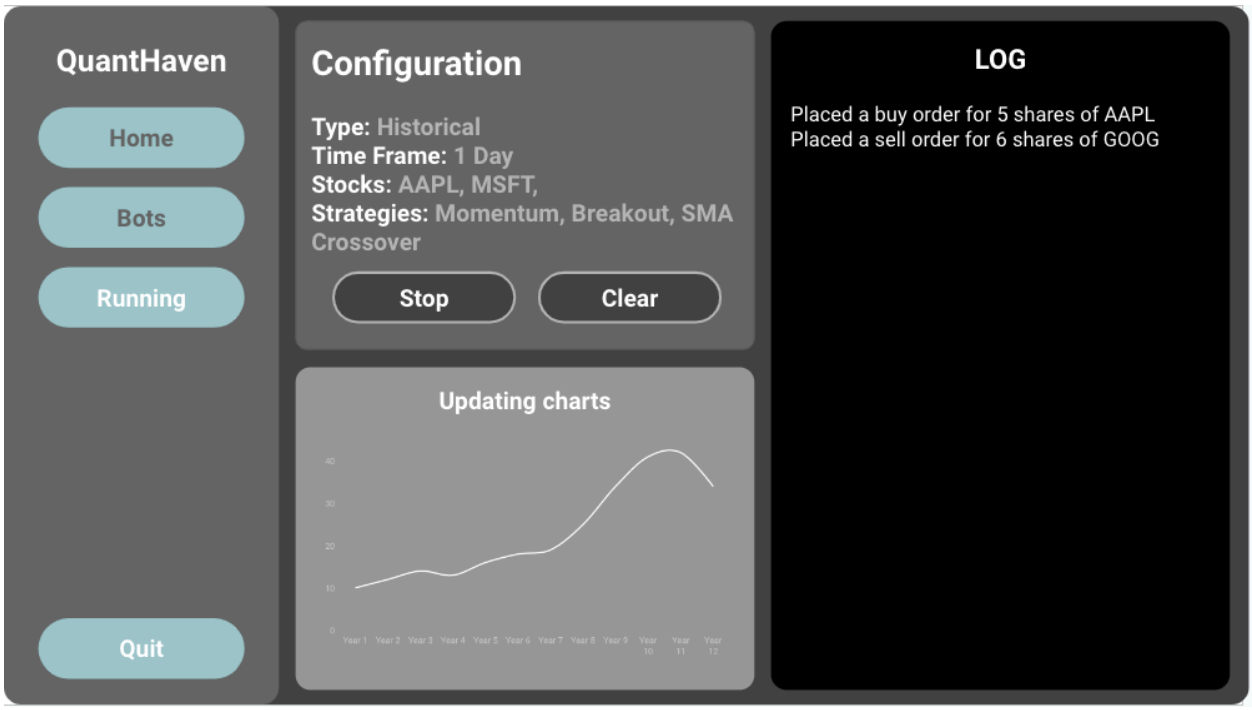
2 API calls all together: 2 Alpha Vantage API calls (Company overview endpoint, News and sentiment endpoint). Pop up window geometry: 800 x 580 pixels. The scrollable boxes will use customtkinter's 'Textbox' widget.

3. The Bots Page



Once the add button is pressed, it will be disabled until the popup window is closed. Running is disabled till a bot is run. When a bot is running, Home and Bots will be disabled. Regardless of the way the time frame is entered by the user, it will always be converted to a base value in minutes, and this is the value that will be used by the program. The table will use tkinter's 'TreeView' widget. Stocks will be limited to three per bot to account for the app's processing capacity (this can be changed).

4. The Running Page

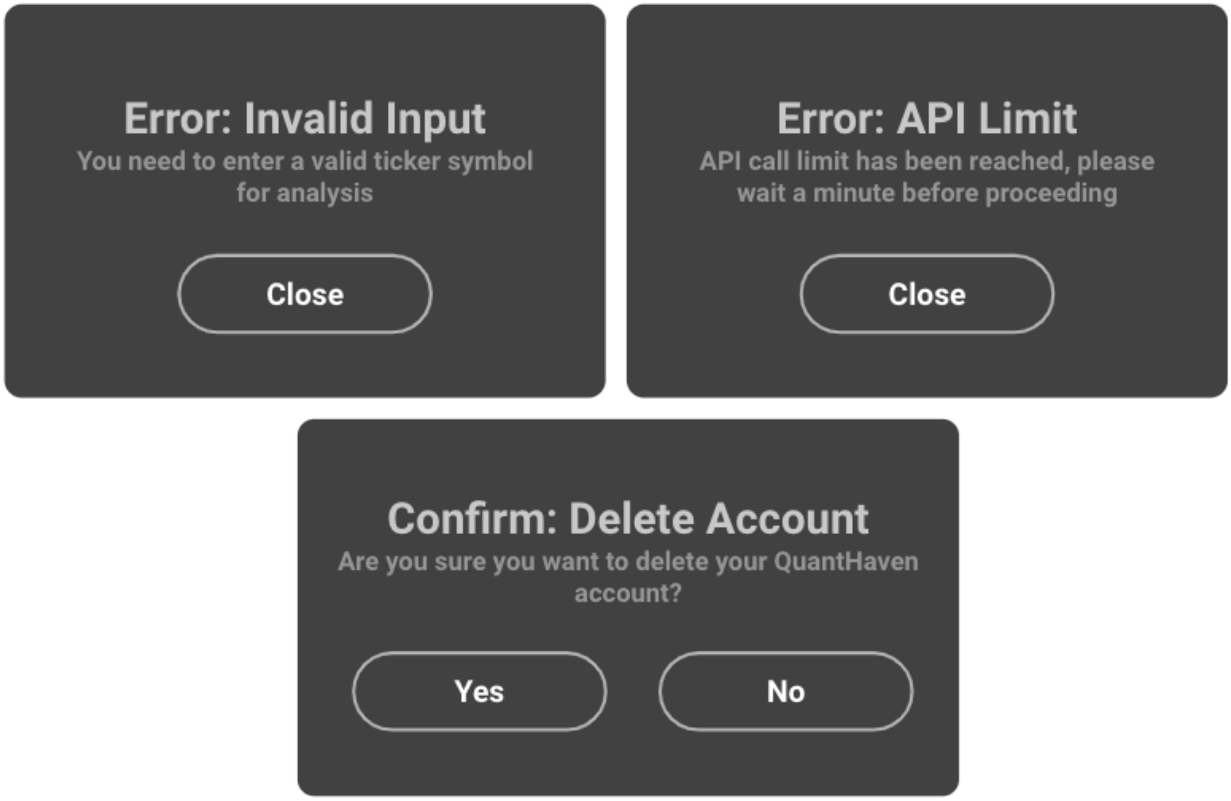


The 'Stop' button will stop the bot and return the user to the Bots page. Clear will clear the log to remove the need for scrolling if the user finds the log is filled with messages. The data_queue Queue will be polled for incoming dataframes, and graphed on the bottom left frame seen above. Every new dataframe will override the old one.

5. Error Pages

Some errors will be popup windows while some will be a variable label on the GUI frame. All error and confirmation messages:

- [Home] You need to enter a valid ticker symbol for analysis - If submit is pressed without entering anything or with an invalid stock.
- [Home, Bots] API call limit has been reached, please wait a minute before proceeding x 2
- [Login, Sign-up, Bots] You need to fill in all fields x 4
- [Bots] You need to pick at least one strategy x 2
- [Bots] You need to enter valid ticker symbols x 2
- [App] Are you sure you want to delete your QuantHaven account?



Security & Integrity

Hashing

```
FUNCTION generate_salt(length = 5):
    // Generate a salt from random integers
    salt = EMPTY_STRING
    FOR i FROM 0 TO length - 1:
        number = 1000 * ABSOLUTE_VALUE((i ^ 0.5) - FLOOR(i ^ 0.5))
        character = CHR(number % 256)
        salt = salt + character
    END FOR
    RETURN salt
```

```
FUNCTION hash_password(input_string, salt = NULL):
    IF salt IS NULL:
        salt = CALL generate_salt()
    combined_input = input_string + salt
    // Large prime number
    prime = 7919
    ascii_values = EMPTY_LIST
    FOR char IN combined_input:
        ADD ASCII(char) TO ascii_values
    END FOR
    hash_value = 0
    FOR i, val IN ascii_values WITH INDEX:
        hash_value = hash_value + val * (prime ^ i)
    END FOR
    hash_value = hash_value % 1000000
    // Store the salt alongside the hash using a delimiter (e.g., $)
    RETURN ZERO_PAD(hash_value, 6) + "$" + salt
FUNCTION verify(stored_hash, input_string):
    hash_part, salt = SPLIT stored_hash BY "$"
    RETURN hash_part EQUALS SPLIT(CALL hash_password(input_string, salt) BY
"$")[0]
```

In the hash_password function, salts are random values appended to inputs to ensure identical inputs yield distinct hashes. The generate_salt method produces this randomness using the fractional parts of square roots. The chosen prime number, 7919, amplifies unpredictability: each character's ASCII value in the salted input is multiplied by the prime raised to its position, with the results aggregated to form a hash value. The modulo operation (% 1000000) ensures the hash is of a fixed length (6 digits). This hash is stored alongside its generating salt, separated by a $. For verification, the salt is extracted and combined with the user's input to recreate and verify the hash, ensuring originality and authenticity.

Encryption

The Vernam cipher, also known as the one-time pad, is a symmetric encryption method that is theoretically unbreakable when used correctly. It involves taking a plaintext and a random key (the "pad") of the same length and performing an XOR operation between them to produce the ciphertext. This will be used to encrypt the API keys. The following functions will be stored in the security.py file, and imported when needed.

```
FUNCTION generate_key(length):
    key = EMPTY_STRING
    FOR i FROM 0 TO length - 1:
        random_byte = GET_RANDOM_BYTE() // OS-specific call to generate
random byte
        character = CHR(random_byte)
        key = key + character
    END FOR
    RETURN key

FUNCTION store_key(identifier, key):
    OPEN FILE NAMED KEY_FILE FOR APPENDING AS file:
        WRITE identifier + ":" + key + NEWLINE TO file
    END OPEN

FUNCTION retrieve_key(identifier):
    OPEN FILE NAMED KEY_FILE FOR READING AS file:
        FOR EACH line IN file's LINES:
            IF line STARTS WITH identifier + ":":
                RETURN SPLIT line BY ':' AT INDEX 1
            END IF
        END FOR
    END OPEN
    RAISE ERROR "No key found for identifier " + identifier

FUNCTION encrypt_data(string_to_encrypt, identifier):
```

```
    key = CALL generate_key(LENGTH OF string_to_encrypt)
    CALL store_key(identifier, key)
    encrypted_string = EMPTY_STRING
    FOR EACH pair (p, k) IN ZIP(string_to_encrypt, key):
        encrypted_char = CHR(ASCII(p) XOR ASCII(k))
        encrypted_string = encrypted_string + encrypted_char
    END FOR
    RETURN encrypted_string

FUNCTION decrypt_data(encrypted_string, identifier):
    key = CALL retrieve_key(identifier)
    decrypted_string = EMPTY_STRING
    FOR EACH pair (c, k) IN ZIP(encrypted_string, key):
        decrypted_char = CHR(ASCII(c) XOR ASCII(k))
        decrypted_string = decrypted_string + decrypted_char
    END FOR
    RETURN decrypted_string
```

The store_key function writes the keys to an external file named keys.key. Each line in this file follows the format:

```
Identifier:key
```

The identifier will be the user_id + 'X' where X is either A (Alpaca key), B (Alpaca secret key) or C (Alpha Vantage key). This will be provided each time something is encrypted/decrypted. The keys.key file will looks like this:

```
1A:random_key_for_string1
1B:random_key_for_string2
1C:random_key_for_string3
```

This file will be secured using access rights.

Error handling

- Ideal error handling for connections failures:

    - For the historical bot, API requests may fail upon a connection error. The primary function for retrieving data from APIs, `request_data()`, will return None if such an error is encountered. Each strategy will be equipped with appropriate error handling as the `validation()` function will first check if self.data is None and the user will be notified about the connection error.
    - For the live bot, the websocket is likely to fail upon a connection error. This will be handled by calling the built-in function `on_error()` for the websocket library. There should be a limit set to the attempts at reconnecting (this can be adjusted, but will be set to 5 as a placeholder).
    - Upon failure, a bot will stop. The program will not crash, but the bot itself will need to be restarted.
- Validating user inputs:
    - Within the main application and GUI, all user inputs will be checked using respective procedures:
        - main.py - `validate_alpha_vantage_key()` and `check_credentials()`
        - home.py - `validate_input()` and `submit()`
        - bot.py - `submit()`, `validate()` and `validate_tickers()`
- Handling API call limits:
    - With a free key, making certain request types too frequently can lead to expected errors (especially relevant to Alpha Vantage rather than Alpaca). To handle this, procedures like `handle_api_limit()` will make sure that repeated calls (in required situations) are at least 60 seconds apart. Such procedures would open a `open_rate_limit_window()` and only allow the user to proceed after this wait time is completed.

Key structures and algorithms

Calculating performance values

Correlation

```
FUNCTION get_correlation_data():
    // Retrieve configurations
    configurations = GET_CONFIG()
    // Extract the stocks from the configurations
    stocks = configurations AT INDEX 2
    // Initialize an empty list to store average correlations for each stock
    correlation_data = EMPTY_LIST
```

```
    // Request data for the given stocks
    df = REQUEST_DATA(stocks)
    // Loop through each stock in the stocks list
    FOR EACH stock IN stocks:
        // Calculate the correlation matrix for the entire DataFrame
        correlation_matrix = CALCULATE_CORRELATION(df)
        // Extract the correlation values for the current stock
        stock_correlations = correlation_matrix FOR COLUMN stock
        // Calculate the average correlation for the current stock
        average_correlation = AVERAGE_OF stock_correlations
        // Add the calculated average correlation to the correlation_data
list
        ADD average_correlation TO correlation_data
    END FOR
    // Return the list of average correlations for each stock
    RETURN correlation_data
```

The `get_correlation_data` function retrieves stock configurations and then fetches related stock data. For each stock, it calculates its correlation with other stocks in the dataset. The function computes the average correlation for each stock compared to the rest and returns a list of these average correlation values. This is done under the assumption that a portfolio must be diversified, and investing in stocks that move in the same direction are to be avoided.

Volatility

This will be scaled to the next higher unit of time. Volatility (standard deviation) will be calculated from either historical data or the rolling window like so: `std = self.data["strategy"].std() * scaling_factor . self.data["strategy"].std()` calculates the standard deviation of the strategy's returns as an average of the next higher unit of time. For example, a 3 minute bar configuration will return an hourly average volatility.

Sentiment

A `get_sentiment_data()` function will retrieve sentiment scores for a list of stocks from Alpha Vantage. After fetching configurations and constructing a specific URL for each stock, it gathers news article sentiment data. Using a nested method, the function will compute a

weighted sentiment score for each article, factoring in both the article's relevance and a time decay (with a half-life of 7 days). The function will return a list of aggregated sentiments for each stock, derived from the combined weighted sentiments and relevances of the related articles.

The time decay will work somewhat like this (data is retrieved from json format):

```
SET time_published TO CONVERT_STRING_TO_DATETIME(item["time_published"], FORMAT "%Y%m%dT%H%M%S")

SET days_since_published TO DIFFERENCE_IN_DAYS(today, time_published)

SET decay_factor TO POWER(0.5, days_since_published DIVIDED BY 7.0) // Using half-life of 7 days
```

The method to gradually decrease the significance or weight of a piece of data as time progresses. This is especially useful when newer data is considered more important than older data, so is relevant in the context of sentiment from news analysis.

Calculating position size

The position calculation for the live bot will utilize a linked list along with a little recursion to return an integer position size.

Node Structure:

- The system utilizes a basic structure called 'Node'.
- Each Node holds a weight (its importance), a value (its actual data), and a reference to the next Node (like in a linked list).

Computing Score:

- A recursive function traverses through the linked list of nodes.
- For each node, it multiplies its weight with its value and adds the result to the score of the next node.
- This process continues until the end of the linked list is reached.

Calculating Position Size:

- Defaults and constants:
    - If volatility isn't provided, it defaults to a moderate value of 0.5.
    - A constant defines the maximum fraction of the balance that can be allocated.
- Determine weights:
    - The importance (weights) of factors is determined dynamically.
    - Volatility: Its weight is influenced directly by its own value.
    - Correlation: Its weight is adjusted based on volatility.
    - Balance and price: They are given fixed weights.

    - The sum of all weights should be 1.
- Normalization:
    - The given factors (volatility, correlation, balance, and price) are normalized to fit within a specific range.
    - A linked list of Nodes is constructed using these normalized values and their respective weights.
- Compute combined score:
    - The score is calculated by traversing the linked list using the recursive function described earlier.
- Risk management:
    - A risk factor is derived from volatility. The higher the volatility, the riskier the investment is perceived to be.
    - The position size is influenced by this risk factor.
- Scaling and allocation adjustment:
    - The calculated position size is scaled down by a predetermined factor.
    - It's then ensured that the position size does not exceed the maximum allowed allocation of the balance.
    - The position size should also not be less than 1.
- The final, adjusted position size is then returned.

The scaling factor and maximum allocation will influence the scale of the risk, and can be changed by the user. Lowering the value of the scaling factor will increase the resulting position size, and the maximum allocation will act as a safety cap. Also instead of linearly mapping sentiment, I'll use the tanh function to get a non-linear normalized sentiment value. As there is a large window for calculation with the historical bot, the position size calculation for this bot is slightly more complex; it will traverse a balanced binary search tree, and take an extra parameter of sentiment. This will work like so:

Node Structure:

- A 'Node' represents an element in the balanced tree.
- Each node holds:
    - A weight (indicating importance).
    - A value (actual data).
    - A key (for maintaining order).
    - Its height in the tree.
    - Pointers to left and right children.

BalancedTree Structure:

- Contains a main root node.
- Offers operations for:
    - Retrieving and updating node height.
    - Checking and maintaining balance during insertion.

- Calculating the overall score.

Calculating Position Size:

- Setting defaults for volatility and 0 for a sentiment value of None.
- Determining factor importance (weights) same as for Live.
- Normalization of factors same as for Live. Sentiment is mapped between 0 and 1.
- Construct the balanced tree:
  - Insert each factor based on its weight, value, and a unique key.
  - The tree self-balances during insertions, ensuring optimal traversal times.
- Score computation:
  - Traverse the tree to compute a score based on the weights and values of the nodes.
- Position size calculation and final adjustments will be the same as for Live.

The subroutine is designed to use dynamic weights and normalization, and risk factor to align with common principles of risk management in trading.

Updating live countdown

```
FUNCTION update_time:
    SET now to CURRENT DATE AND TIME in US/Eastern timezone
    SET current_day to DAY OF THE WEEK from now (0 for Monday, 6 for Sunday)

    IF self.total_seconds is 0 OR not initialized THEN
        IF current_day is Saturday OR Sunday THEN
            SET market status to "Market is closed"
            SET status message to "Time till market opens:"
            IF current_day is Sunday THEN
                SET days_till_monday to 1
            ELSE
                SET days_till_monday to 2
            END IF
            SET next_open to TIME at start of market on Monday
            SET self.total_seconds to TIME DIFFERENCE between next_open and
now
        ELSE
```

```
            IF current_time is between market_open AND market_close THEN
                SET market status to "Market is open"
                SET status message to "Time till market closes:"
                SET self.total_seconds to TIME DIFFERENCE between
market_close today and now
            ELSE
                SET market status to "Market is closed"
                SET status message to "Time till market opens:"
                IF current_time is after market_close THEN
                    SET next_day to tomorrow's date
                    SET next_open to TIME at start of market tomorrow
                    SET self.total_seconds to TIME DIFFERENCE between
next_open and now
                ELSE
                    SET self.total_seconds to TIME DIFFERENCE between
market_open today and now
                END IF
            END IF
        END IF
        UPDATE market display with market status
        UPDATE status display with status message
    END IF
    DECREMENT self.total_seconds by 1
    CONVERT self.total_seconds to hours, minutes, and seconds format
    UPDATE countdown display with the formatted time
    CALL update_time again after 1 second
END FUNCTION
```

The `update_time` function should be optimized to reduce resource usage by employing a time-based trigger mechanism. Instead of continuously polling or checking the market status or time, it recalculates the time difference only when it's necessary and then updates the countdown by decrementing self.total_seconds every second. This countdown is

updated once a second using the self.after method, ensuring that the function only executes when needed.

bot_configuration.txt

The bot_configuration.txt file is used to share the current bot's settings with any files that need it. This file will be created and written into when first run, and overwritten when another bot is run. The information in this text file will be written on to a single line and be separated using '$' as a splitter. So for example: `1$Minute$3$Live$AAPL, IBM$Momentum, Scalping$2`.

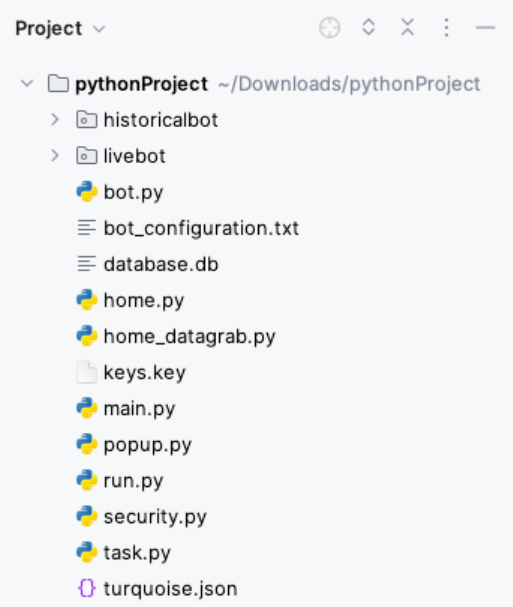
Determining size of database

The approximate size of the databases being worked with are determined by using a `limit` value (currently set to 500) and a `frequency` (this is the timeframe input by the user, converted to minutes) value. Alpaca's `StockBarsRequest` takes a start date, and returns data till the present. A function `calculate_days_back` will calculate the number of days back (start date) for `x` time frame to have around 500 bars.

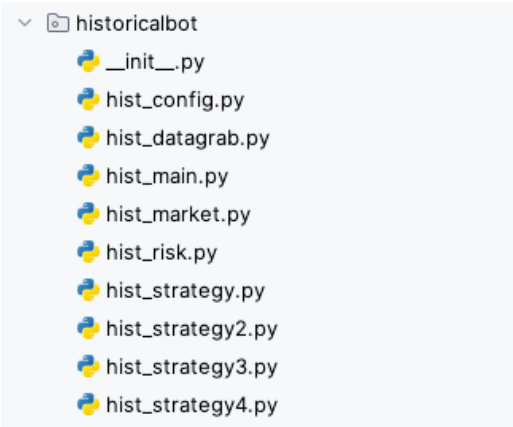
```
FUNCTION calculate_days_back(bars, frequency):

    SET trading_minutes_per_day to 6.5 multiplied by 60

    SET total_minutes to bars multiplied by frequency

    SET days to total_minutes divided by trading_minutes_per_day

    RETURN rounded down days plus 1

END FUNCTION
```

Technical Solution

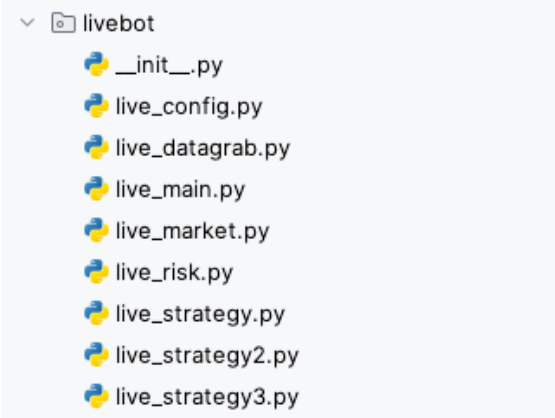
File structure



All files



Package 1: historicalbot



Package 2: livebot

GUI

main.py

```python
# IMPORTING API
from alpaca.trading.client import TradingClient
# IMPORTING MODULES
import customtkinter
import requests
import sqlite3
import multiprocessing
# IMPORTING USER-DEFINED MODULES
from home import StartPage
from bot import PageOne
from run import PageTwo
from task import Task
import security


1 usage
def generate_salt(length=5):
    # Generate a salt from random integers
    salt = ''.join(chr(i % 256) for i in [int(1000 * abs((i ** 0.5) - int(i ** 0.5))) for i in range(length)])
    return salt


2 usages
def hash_password(input_string, salt=None):
    if salt is None:
        salt = generate_salt()

    combined_input = input_string + salt

    # Operations involving large prime numbers are not easily reversible
    prime = 7919
    ascii_values = [ord(char) for char in combined_input]
    hash_value = 0

    for i, val in enumerate(ascii_values):
        hash_value += val * (prime ** i)

    hash_value = hash_value % 1000000

    # Store the salt alongside the hash using a delimiter (e.g., $)
    return f"{str(hash_value).zfill(6)}${salt}"


1 usage
def verify(stored_hash, input_string):
    hash_part, salt = stored_hash.split('$')
    return hash_part == hash_password(input_string, salt).split('$')[0]


customtkinter.set_appearance_mode("Dark")
customtkinter.set_default_color_theme("turquoise.json")
```

```
49
50  # Connect to SQLite database
51  conn = sqlite3.connect('database.db')
52  # Create a cursor
53  c = conn.cursor()
54  # Create users table
55  c.execute("""CREATE TABLE IF NOT EXISTS users (
56              id INTEGER PRIMARY KEY AUTOINCREMENT,
57              username text,
58              password text
59              )""")
60  # Create details table
61  c.execute("""CREATE TABLE IF NOT EXISTS details (
62              user_id INTEGER PRIMARY KEY,
63              name text,
64              api_key text,
65              api_secret_key text,
66              av_key text,
67              FOREIGN KEY(user_id) REFERENCES users(id)
68              )""")
69  # Commit and close connection
70  conn.commit()
71  conn.close()
72
73
    1 usage
74  class App(customtkinter.CTkToplevel):
75      def __init__(self, master, user_id):
76          super().__init__(master)
77          self.master = master
78          self.user_id = user_id
79          self._frames = {}
80          self._frame = None
81          self.bot_time_frame = customtkinter.StringVar()
82          self.bot_type = customtkinter.StringVar()
83          self.bot_stocks = customtkinter.StringVar()
84          self.bot_strategies = customtkinter.StringVar()
85          self.data_queue = multiprocessing.Queue()
86          self.stdout_queue = multiprocessing.Queue()
87          self.task = Task(self.data_queue, self.stdout_queue)
88          self.switch_frame('StartPage')
89
90          # Configure window
91          self.title("⧗ QuantHaven")
92          screen_width = 1440
93          screen_height = 900
94          window_width = 1000
95          window_height = 580
96          x_position = (screen_width - window_width) // 2
```

```
        y_position = (screen_height - window_height) // 2
        self.geometry(f"{window_width}x{window_height}+{x_position}+{y_position}")
        self.resizable(False, False)

        # Configure grid layout (4x4)
        self.grid_columnconfigure(1, weight=1)
        self.grid_columnconfigure((2, 3), weight=0)
        self.grid_rowconfigure((0, 1, 2), weight=1)
        self.configure(fg_color="#434343")

        # Create sidebar frame with widgets
        self.sidebar_frame = customtkinter.CTkFrame(self, fg_color="#666666", width=140, corner_radius=0)
        self.sidebar_frame.grid(row=0, column=0, rowspan=4, sticky="nsew")
        self.sidebar_frame.grid_rowconfigure(4, weight=1)
        self.logo_label = customtkinter.CTkLabel(self.sidebar_frame,
                                                 text="♜ QuantHaven",
                                                 font=customtkinter.CTkFont(size=22, weight="bold"))
        self.logo_label.grid(row=0, column=0, padx=20, pady=(20, 10))
        self.sidebar_button_1 = customtkinter.CTkButton(self.sidebar_frame,
                                                        text="Home",
                                                        font=customtkinter.CTkFont(size=12, weight="bold"),
                                                        height=30,
                                                        corner_radius=15,
                                                        command=lambda: self.switch_frame('StartPage'))

        self.sidebar_button_1.grid(row=1, column=0, padx=20, pady=10)
        self.sidebar_button_2 = customtkinter.CTkButton(self.sidebar_frame,
                                                        text="Bots",
                                                        font=customtkinter.CTkFont(size=12, weight="bold"),
                                                        height=30, corner_radius=15,
                                                        command=lambda: self.switch_frame('PageOne'))
        self.sidebar_button_2.grid(row=2, column=0, padx=20, pady=10)
        self.sidebar_button_3 = customtkinter.CTkButton(self.sidebar_frame,
                                                        text="Running",
                                                        font=customtkinter.CTkFont(size=12, weight="bold"),
                                                        height=30,
                                                        corner_radius=15,
                                                        command=lambda: self.switch_frame('PageTwo'))
        self.sidebar_button_3.grid(row=3, column=0, padx=20, pady=10)
        self.forget_account = customtkinter.CTkButton(self.sidebar_frame,
                                                      text="Forget",
                                                      font=customtkinter.CTkFont(size=12, weight="bold"),
                                                      height=30,
                                                      corner_radius=15,
                                                      command=lambda: self.are_you_sure())
        self.forget_account.grid(row=7, column=0, padx=20, pady=10)
        self.quit_button = customtkinter.CTkButton(self.sidebar_frame,
                                                   text="Quit",
                                                   font=customtkinter.CTkFont(size=12, weight="bold"),
```

```
                                                height=30,
                                                corner_radius=15,
                                                command=lambda: self.close_program())
        self.quit_button.grid(row=8, column=0, padx=20, pady=(10, 20))

        # Set default values
        self.sidebar_button_3.configure(state="disabled")
        self.bind("<Destroy>", self.close_program)

    1 usage (1 dynamic)
    def enable_running_button(self):
        self.sidebar_button_3.configure(state="normal")
        self.sidebar_button_1.configure(state='disabled')
        self.sidebar_button_2.configure(state='disabled')

    1 usage (1 dynamic)
    def disable_running_button(self):
        self.sidebar_button_3.configure(state="disabled")
        self.sidebar_button_1.configure(state='normal')
        self.sidebar_button_2.configure(state='normal')

    6 usages (2 dynamic)
    def switch_frame(self, frame_class):  # Raise requested frame
        frame = self._frames.get(frame_class, None)

        if frame is None:
            if frame_class == 'PageOne':
                frame = PageOne(master=self, task=self.task, user_id=self.user_id, controller=self)
            elif frame_class == 'PageTwo':
                frame = PageTwo(master=self,
                                task=self.task,
                                data_queue=self.data_queue,
                                stdout_queue=self.stdout_queue,
                                controller=self)
            else:
                frame = StartPage(master=self, user_id=self.user_id)
            self._frames[frame_class] = frame

        if self._frame is not None:
            self._frame.grid_forget()

        self._frame = frame
        self._frame.grid(row=0, column=1, rowspan=4, columnspan=3, sticky="nsew")

    1 usage
    def are_you_sure(self):
        are_you_sure = customtkinter.CTkToplevel(self)
        are_you_sure.title("Login")
        screen_width = 1440
        screen_height = 900
        window_width = 300
        window_height = 200
```

```
        x_position = (screen_width - window_width) // 2
        y_position = (screen_height - window_height) // 2
        are_you_sure.geometry(f"{window_width}x{window_height}+{x_position}+{y_position}")
        are_you_sure.resizable(False, False)
        are_you_sure.configure(fg_color="#434343")
        customtkinter.CTkLabel(are_you_sure,
                               text="Confirm: Delete Account",
                               font=customtkinter.CTkFont(size=22, weight="bold")).place(relx=0.5,
                                                                                        rely=0.2,
                                                                                        anchor='center')
        customtkinter.CTkLabel(are_you_sure,
                               text_color="#CCCCCC",
                               text="Are you sure you want to delete your QuantHaven account?",
                               font=customtkinter.CTkFont(size=14, weight="bold"),
                               wraplength=200).place(relx=0.5, rely=0.4, anchor='center')
        customtkinter.CTkButton(are_you_sure,
                                text="Yes",
                                fg_color="transparent",
                                border_width=2,
                                font=customtkinter.CTkFont(size=12, weight="bold"),
                                width=80,
                                height=30,
                                corner_radius=15,
                                command=self.delete_user).place(relx=0.3, rely=0.7, anchor='center')

        customtkinter.CTkButton(are_you_sure,
                                text="No",
                                fg_color="transparent",
                                border_width=2,
                                font=customtkinter.CTkFont(size=12, weight="bold"),
                                width=80,
                                height=30,
                                corner_radius=15,
                                command=are_you_sure.destroy).place(relx=0.7, rely=0.7, anchor='center')

    1 usage
    def delete_user(self):
        conn = sqlite3.connect('database.db')
        c = conn.cursor()
        # Delete the user's data entries
        c.execute('DELETE FROM configurations WHERE user_id = ?', (self.user_id,))
        c.execute('DELETE FROM details WHERE user_id = ?', (self.user_id,))
        # Delete the user
        c.execute('DELETE FROM users WHERE id = ?', (self.user_id,))
        conn.commit()
        conn.close()
        # Choosing to delete your user data also closes the app
        self.close_program()

    3 usages
    def close_program(self, event=None):
```

```
        self.unbind("<Destroy>")
        self.task.stop()
        self.destroy()
        self.master.quit()


1 usage
def check_credentials(api_key, secret_key):
    trading_client = TradingClient(api_key, secret_key, paper=True)
    try:
        account_info = trading_client.get_account()
        print(account_info)
        return True
    except Exception as e:
        print(e)
        return False


1 usage
def validate_alpha_vantage_key(api_key):
    test_url = f"https://www.alphavantage.co/query?function=GLOBAL_QUOTE&symbol=SPY&apikey={api_key}"
    response = requests.get(test_url)

    if response.status_code == 200:  # Successful request will return 200
        json_response = response.json()
        if json_response.get("message") == "forbidden.":

            print("Invalid Alpha Vantage API key.")
            return False
        elif "Global Quote" in json_response:
            print("Valid Alpha Vantage API key.")
            return True
        else:
            print("Received unexpected JSON response from Alpha Vantage API.")
            return False
    else:
        print(f"Received unexpected status code {response.status_code} from Alpha Vantage API")
        return False


1 usage
class LoginApp(customtkinter.CTk):
    def __init__(self):
        super().__init__()
        self.title("Account System")
        self.screen_width = 1440
        self.screen_height = 900
        window_width = 250
        window_height = 250
        x_position = (self.screen_width - window_width) // 2
        y_position = (self.screen_height - window_height) // 2
        self.geometry(f"{window_width}x{window_height}+{x_position}+{y_position}")
```

```
        self.resizable(False, False)
        self.configure(fg_color="#434343")

        self.conn = sqlite3.connect('database.db')
        self.c = self.conn.cursor()

        # Create login and sign up buttons
        customtkinter.CTkLabel(self,
                               text="✖ QuantHaven",
                               font=customtkinter.CTkFont(size=22, weight="bold")).place(relx=0.5,
                                                                                        rely=0.2,
                                                                                        anchor='center')
        self.login_btn = customtkinter.CTkButton(self,
                                                 text="Login",
                                                 font=customtkinter.CTkFont(size=12, weight="bold"),
                                                 height=30,
                                                 corner_radius=15,
                                                 command=self.login)
        self.login_btn.place(relx=0.5, rely=0.4, anchor='center')
        customtkinter.CTkLabel(self,
                               text_color="#CCCCCC",
                               text="Or if you don't have a QuantHaven account:",
                               font=customtkinter.CTkFont(size=14, weight="bold"),
                               wraplength=180).place(relx=0.5, rely=0.6, anchor='center')

        self.sign_up_btn = customtkinter.CTkButton(self,
                                                   text="Sign Up",
                                                   fg_color="transparent",
                                                   border_width=2,
                                                   font=customtkinter.CTkFont(size=12, weight="bold"),
                                                   height=30,
                                                   corner_radius=15,
                                                   command=self.sign_up)
        self.sign_up_btn.place(relx=0.5, rely=0.8, anchor='center')

    1 usage
    def login(self):
        self.login_btn.configure(state='disabled')
        login_window = customtkinter.CTkToplevel(self)
        login_window.title("Login")
        window_width = 250
        window_height = 300
        x_position = (self.screen_width - window_width) // 2
        y_position = (self.screen_height - window_height) // 2
        login_window.geometry(f"{window_width}x{window_height}+{x_position}+{y_position}")
        login_window.resizable(False, False)
        login_window.configure(fg_color="#434343")
        customtkinter.CTkLabel(login_window,
                               text="Login",
                               font=customtkinter.CTkFont(size=22, weight="bold")).place(relx=0.5,
```

```
                                                                    rely=0.2,
                                                                    anchor='center')

    customtkinter.CTkLabel(login_window,
                           text_color="#CCCCCC",
                           text="Username:",
                           font=customtkinter.CTkFont(size=14, weight="bold")).place(relx=0.5,
                                                                    rely=0.3,
                                                                    anchor='center')
    username_entry = customtkinter.CTkEntry(login_window,
                                            height=30,
                                            corner_radius=15,
                                            placeholder_text="Username",
                                            font=customtkinter.CTkFont(size=12, weight="bold"))
    username_entry.place(relx=0.5, rely=0.4, anchor='center')

    customtkinter.CTkLabel(login_window,
                           text_color="#CCCCCC",
                           text="Password:",
                           font=customtkinter.CTkFont(size=14, weight="bold")).place(relx=0.5,
                                                                    rely=0.5,
                                                                    anchor='center')
    password_entry = customtkinter.CTkEntry(login_window,
                                            show="\u2022",

                                            height=30,
                                            corner_radius=15,
                                            placeholder_text="Password",
                                            font=customtkinter.CTkFont(size=12, weight="bold"))
    password_entry.place(relx=0.5, rely=0.6, anchor='center')

    self.login_error_label = customtkinter.CTkLabel(login_window,
                                                    text="",
                                                    font=customtkinter.CTkFont(size=14, weight="bold"))
    self.login_error_label.place(relx=0.5, rely=0.7, anchor='center')
    customtkinter.CTkButton(login_window,
                            text="Login",
                            font=customtkinter.CTkFont(size=12, weight="bold"),
                            height=30,
                            corner_radius=15,
                            command=lambda: self.check_login(username_entry.get(),
                                                             password_entry.get(),
                                                             login_window)).place(relx=0.5,
                                                                                  rely=0.8,
                                                                                  anchor='center')
    login_window.bind("<Destroy>", self.enable_buttons)

1 usage
def check_login(self, username, password, login_window):
    self.c.execute("SELECT * FROM users WHERE username=?", (username,))
```

```
385         user = self.c.fetchone()
386
387         if username == "" or password == "":
388             self.login_error_label.configure(text="Fields cannot be blank", text_color="#EA9999")
389             return
390
391         if user:
392             if verify(user[2], password):
393                 # Passwords match
394                 self.login_error_label.configure(text="Successfully logged in", text_color="#B6D7A8")
395                 # CLOSE  WINDOW
396                 login_window.withdraw()
397                 self.withdraw()  # Hide the login window
398                 App(self, user[0])
399             else:
400                 # Passwords do not match
401                 self.login_error_label.configure(text="Password does not match user", text_color="#EA9999")
402                 user = None
403         else:
404             self.login_error_label.configure(text="User not found", text_color="#EA9999")
405
        1 usage
406     def sign_up(self):
407         self.sign_up_btn.configure(state='disabled')
408         sign_up_window = customtkinter.CTkToplevel(self)

409         sign_up_window.title("Sign Up")
410         window_width = 250
411         window_height = 500
412         x_position = (self.screen_width - window_width) // 2
413         y_position = (self.screen_height - window_height) // 2
414         sign_up_window.geometry(f"{window_width}x{window_height}+{x_position}+{y_position}")
415         sign_up_window.resizable(False, False)
416         sign_up_window.configure(fg_color="#434343")
417         sign_up_window.grid_rowconfigure(0, weight=1)
418         sign_up_window.grid_rowconfigure(16, weight=1)
419         sign_up_window.grid_columnconfigure(0, weight=1)
420         sign_up_window.grid_columnconfigure(2, weight=1)
421         customtkinter.CTkLabel(sign_up_window,
422                                text="Sign up",
423                                font=customtkinter.CTkFont(size=22, weight="bold")).grid(row=1, column=1)
424
425         customtkinter.CTkLabel(sign_up_window,
426                                text_color="#CCCCCC",
427                                text="Username:",
428                                font=customtkinter.CTkFont(size=14, weight="bold")).grid(row=2, column=1)
429         username_entry = customtkinter.CTkEntry(sign_up_window,
430                                                 height=30,
431                                                 corner_radius=15,
432                                                 placeholder_text="Username",
```

```
                                        font=customtkinter.CTkFont(size=12, weight="bold"))
    username_entry.grid(row=3, column=1)

    customtkinter.CTkLabel(sign_up_window,
                           text_color="#CCCCCC",
                           text="Password:",
                           font=customtkinter.CTkFont(size=14, weight="bold")).grid(row=4, column=1)
    password_entry = customtkinter.CTkEntry(sign_up_window,
                                            show="\u2022",
                                            height=30,
                                            corner_radius=15,
                                            placeholder_text="Password",
                                            font=customtkinter.CTkFont(size=12, weight="bold"))
    password_entry.grid(row=5, column=1)

    customtkinter.CTkLabel(sign_up_window,
                           text_color="#CCCCCC",
                           text="Name",
                           font=customtkinter.CTkFont(size=14, weight="bold")).grid(row=6, column=1)
    name_entry = customtkinter.CTkEntry(sign_up_window,
                                        height=30,
                                        corner_radius=15,
                                        placeholder_text="Name",
                                        font=customtkinter.CTkFont(size=12, weight="bold"))

    name_entry.grid(row=7, column=1)

    customtkinter.CTkLabel(sign_up_window,
                           text_color="#CCCCCC",
                           text="Alpaca Key:",
                           font=customtkinter.CTkFont(size=14, weight="bold")).grid(row=8, column=1)
    api_key_entry = customtkinter.CTkEntry(sign_up_window,
                                           show="\u2022",
                                           height=30,
                                           corner_radius=15,
                                           placeholder_text="API Key",
                                           font=customtkinter.CTkFont(size=12, weight="bold"))
    api_key_entry.grid(row=9, column=1)

    customtkinter.CTkLabel(sign_up_window,
                           text_color="#CCCCCC",
                           text="Alpaca Secret Key:",
                           font=customtkinter.CTkFont(size=14, weight="bold")).grid(row=10, column=1)
    api_secret_key_entry = customtkinter.CTkEntry(sign_up_window,
                                                  show="\u2022",
                                                  height=30,
                                                  corner_radius=15,
                                                  placeholder_text="API Key",
                                                  font=customtkinter.CTkFont(size=12, weight="bold"))
```

```
481         api_secret_key_entry.grid(row=11, column=1)
482
483         customtkinter.CTkLabel(sign_up_window,
484                                text_color="#CCCCCC",
485                                text="Alpha Vantage Key",
486                                font=customtkinter.CTkFont(size=14, weight="bold")).grid(row=12, column=1)
487         av_key_entry = customtkinter.CTkEntry(sign_up_window,
488                                               show="\u2022",
489                                               height=30,
490                                               corner_radius=15,
491                                               placeholder_text="API Key",
492                                               font=customtkinter.CTkFont(size=12, weight="bold"))
493         av_key_entry.grid(row=13, column=1)
494
495         self.sign_up_error_label = customtkinter.CTkLabel(sign_up_window,
496                                                           text="",
497                                                           font=customtkinter.CTkFont(size=14, weight="bold"))
498         self.sign_up_error_label.grid(row=14, column=1)
499
500         customtkinter.CTkButton(sign_up_window,
501                                 text="Sign Up",
502                                 font=customtkinter.CTkFont(size=12, weight="bold"),
503                                 height=30,
504                                 corner_radius=15,
505
505                                 command=lambda: self.create_account(username_entry.get(),
506                                                                     password_entry.get(),
507                                                                     name_entry.get(),
508                                                                     api_key_entry.get(),
509                                                                     api_secret_key_entry.get(),
510                                                                     av_key_entry.get(),
511                                                                     sign_up_window)).grid(row=15, column=1)
512         sign_up_window.bind("<Destroy>", self.enable_buttons)
513
    1 usage
514     def create_account(self, username, password, name, api_key, api_secret_key, av_key, sign_up_window):
515
516         if username == "" or password == "" or name == "" or api_key == "" or api_secret_key == "" or av_key == "":
517             self.sign_up_error_label.configure(text="Fields cannot be blank", text_color="#EA9999")
518             return
519
520         self.c.execute("SELECT * FROM users WHERE username=?", (username,))
521         if self.c.fetchone():
522             self.sign_up_error_label.configure(text="Username already exists", text_color="#EA9999")
523             return
524
525         if not check_credentials(api_key, api_secret_key):
526             self.sign_up_error_label.configure(text="Alpaca keys invalid", text_color="#EA9999")
527             return
528
```

```
        if not validate_alpha_vantage_key(av_key):
            self.sign_up_error_label.configure(text="Alpha vantage invalid", text_color="#EA9999")
            return

        # HASH
        hashed_password = hash_password(password)

        self.c.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, hashed_password))
        self.conn.commit()
        self.c.execute("SELECT id FROM users WHERE username=?", (username,))
        user_id = self.c.fetchone()[0]

        # ENCRYPT
        encrypted_api_key = security.encrypt_data(api_key, str(user_id) + 'A')
        encrypted_api_secret_key = security.encrypt_data(api_secret_key, str(user_id) + 'B')
        encrypted_av_key = security.encrypt_data(av_key, str(user_id) + 'C')

        self.c.execute("INSERT INTO details (user_id, name, api_key, api_secret_key, av_key) VALUES (?, ?, ?, ?, ?)",
                       (user_id, name, encrypted_api_key, encrypted_api_secret_key, encrypted_av_key))
        self.conn.commit()
        self.sign_up_error_label.configure(text="Account created successfully", text_color="#B6D7A8")
        self.sign_up_btn.configure(state='normal')
        # CLOSE SIGN UP PAGE
        sign_up_window.withdraw()


    2 usages
    def enable_buttons(self, event=None):
        self.login_btn.configure(state='normal')
        self.sign_up_btn.configure(state='normal')


if __name__ == "__main__":
    loginapp = LoginApp()
    loginapp.mainloop()
```

home.py

```python
# IMPORTING MODULES
import customtkinter
import yfinance as yf
import datetime
import time
from pytz import timezone
import matplotlib.figure as figure
from matplotlib import font_manager
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import sqlite3
# IMPORTING USER-DEFINED MODULES
from home_datagrab import home_data
from popup import get_news, get_overview
import security

market_open = datetime.time(9, 30, tzinfo=timezone('US/Eastern'))  # 9:30 AM Eastern Time
market_close = datetime.time(16, 0, tzinfo=timezone('US/Eastern'))  # 4:00 PM Eastern Time


2 usages
class StartPage(customtkinter.CTkFrame):
    def __init__(self, master, user_id):
        customtkinter.CTkFrame.__init__(self, master)
        self.user_id = user_id
        self.entry_used_time = 0

        self.total_seconds = 0

        home = home_data(self.user_id)
        account_balance = home[0]
        total_profit_or_loss = home[1]
        sentiment_values = home[2]
        top_3 = home[3]
        watchlist_stats = home[4]
        watchlist_keys = list(watchlist_stats.keys())

        # Configure grid
        self.grid_columnconfigure((0,1), weight=1)
        self.grid_rowconfigure((0, 1, 2, 3), weight=1)
        self.configure(fg_color="#434343")

        # COLUMN 1

        # Frame 1
        self.frame1 = customtkinter.CTkFrame(self, fg_color="#999999", width=200, height=100)
        self.frame1.grid(row=0, column=0, padx=(20, 10), pady=(20, 0), sticky="nsew")

        self.market = customtkinter.CTkLabel(master=self.frame1,
                                             text="",
                                             font=customtkinter.CTkFont(size=12, weight="bold"))
```

```
        self.market.place(relx=0.5, rely=0.25, anchor='center')  # center alignment
        self.status = customtkinter.CTkLabel(master=self.frame1,
                                             text="",
                                             font=customtkinter.CTkFont(size=18, weight="bold"))
        self.status.place(relx=0.5, rely=0.4, anchor='center')  # center alignment
        self.countdown = customtkinter.CTkLabel(master=self.frame1,
                                                text_color="#CCCCCC",
                                                text="",
                                                font=customtkinter.CTkFont(size=38, weight="bold"))
        self.countdown.place(relx=0.5, rely=0.7, anchor='center')  # center alignment
        self.update_time()

        # Frame 2 left
        self.frame2 = customtkinter.CTkFrame(self, fg_color="transparent", width=200, height=100)
        self.frame2.grid(row=1, column=0, padx=(20, 10), pady=(20, 0), sticky="nsew")

        # Frame 2 right
        self.frame2l = customtkinter.CTkFrame(self.frame2, fg_color="#666666", width=100, height=100)
        self.frame2l.place(relx=0, rely=0, relwidth=0.48, relheight=1.0)
        self.top3 = customtkinter.CTkLabel(master=self.frame2l,
                                           text="Top gainers",
                                           font=customtkinter.CTkFont(size=18, weight="bold"))
        self.top3.place(relx=0.5, rely=0.2, anchor='center')
        self.top3_1 = customtkinter.CTkLabel(master=self.frame2l,
```

```
        self.top3_1 = customtkinter.CTkLabel(master=self.frame2l,
                                             text=top_3[0][:-6],
                                             font=customtkinter.CTkFont(size=12, weight="bold"))
        self.top3_1.place(relx=0.3, rely=0.4, anchor='center')
        self.top3_1_s = customtkinter.CTkLabel(master=self.frame2l,
                                               text_color="#B6D7A8",
                                               text=top_3[0][-6:],
                                               font=customtkinter.CTkFont(size=12, weight="bold"))
        self.top3_1_s.place(relx=0.7, rely=0.4, anchor='center')
        self.top3_2 = customtkinter.CTkLabel(master=self.frame2l,
                                             text=top_3[1][:-6],
                                             font=customtkinter.CTkFont(size=12, weight="bold"))
        self.top3_2.place(relx=0.3, rely=0.6, anchor='center')
        self.top3_2_s = customtkinter.CTkLabel(master=self.frame2l,
                                               text_color="#B6D7A8",
                                               text=top_3[1][-6:],
                                               font=customtkinter.CTkFont(size=12, weight="bold"))
        self.top3_2_s.place(relx=0.7, rely=0.6, anchor='center')
        self.top3_3 = customtkinter.CTkLabel(master=self.frame2l,
                                             text=top_3[2][:-6],
                                             font=customtkinter.CTkFont(size=12, weight="bold"))
        self.top3_3.place(relx=0.3, rely=0.8, anchor='center')
        self.top3_3_s = customtkinter.CTkLabel(master=self.frame2l,
                                               text_color="#B6D7A8",
                                               text=top_3[2][-6:],
```

```
                                        font=customtkinter.CTkFont(size=12, weight="bold"))
        self.top3_3_s.place(relx=0.7, rely=0.8, anchor='center')

        self.frame2r = customtkinter.CTkFrame(self.frame2, fg_color="#666666", width=100, height=100)
        self.frame2r.place(relx=0.52, rely=0, relwidth=0.48, relheight=1.0)
        self.watchlist = customtkinter.CTkLabel(master=self.frame2r,
                                                text="Watchlist",
                                                font=customtkinter.CTkFont(size=18, weight="bold"))
        self.watchlist.place(relx=0.5, rely=0.2, anchor='center')
        self.spy = customtkinter.CTkLabel(master=self.frame2r,
                                          text=f"{watchlist_keys[0]}:",
                                          font=customtkinter.CTkFont(size=12, weight="bold"))
        self.spy.place(relx=0.3, rely=0.4, anchor='center')
        self.spy_s = customtkinter.CTkLabel(master=self.frame2r,
                                            text_color=self.define_colour(watchlist_stats[watchlist_keys[0]]),
                                            text=self.format_watchlist(watchlist_stats[watchlist_keys[0]]),
                                            font=customtkinter.CTkFont(size=12, weight="bold"))
        self.spy_s.place(relx=0.7, rely=0.4, anchor='center')
        self.qqq = customtkinter.CTkLabel(master=self.frame2r,
                                          text=f"{watchlist_keys[1]}:",
                                          font=customtkinter.CTkFont(size=12, weight="bold"))
        self.qqq.place(relx=0.3, rely=0.6, anchor='center')
        self.qqq_s = customtkinter.CTkLabel(master=self.frame2r,
                                            text_color=self.define_colour(watchlist_stats[watchlist_keys[1]]),

                                            text=self.format_watchlist(watchlist_stats[watchlist_keys[1]]),
                                            font=customtkinter.CTkFont(size=12, weight="bold"))
        self.qqq_s.place(relx=0.7, rely=0.6, anchor='center')
        self.dia = customtkinter.CTkLabel(master=self.frame2r,
                                          text=f"{watchlist_keys[2]}:",
                                          font=customtkinter.CTkFont(size=12, weight="bold"))
        self.dia.place(relx=0.3, rely=0.8, anchor='center')
        self.dia_s = customtkinter.CTkLabel(master=self.frame2r,
                                            text_color=self.define_colour(watchlist_stats[watchlist_keys[2]]),
                                            text=self.format_watchlist(watchlist_stats[watchlist_keys[2]]),
                                            font=customtkinter.CTkFont(size=12, weight="bold"))
        self.dia_s.place(relx=0.7, rely=0.8, anchor='center')

        # Frame 3
        self.frame3 = customtkinter.CTkFrame(self, fg_color="#999999", width=200, height=200)
        self.frame3.grid(row=2, column=0, rowspan=2, padx=(20, 10), pady=(20, 20), sticky="nsew")

        sp500 = yf.download(tickers="SPY", period="5y", interval="1d")
        close = sp500['Close']

        # Create FontProperties object for title
        title_font_prop = font_manager.FontProperties(weight='bold', size=14, family='Roboto')
        # Create FontProperties object for labels
        label_font_prop = font_manager.FontProperties(weight='bold', size=8, family='Roboto')
```

```
145
146         fig = figure.Figure(figsize=(4, 2), dpi=100)
147         fig.set_facecolor("#999999")
148         ax = fig.add_subplot(111)
149         ax.set_facecolor("#999999")
150         ax.plot_date(close.index, close, linestyle='-', marker=None, linewidth=1, color='white')
151         ax.grid(False)
152         ax.spines['left'].set_visible(False)
153         ax.spines['right'].set_visible(False)
154         ax.spines['bottom'].set_visible(False)
155         ax.spines['top'].set_visible(False)
156
157         # Set title with bold text
158         ax.set_title('S&P 500', color='white', fontproperties=title_font_prop)
159
160         ax.tick_params(colors='white')
161         for label in ax.get_xticklabels():
162             label.set_fontproperties(label_font_prop)
163         for label in ax.get_yticklabels():
164             label.set_fontproperties(label_font_prop)
165
166         canvas = FigureCanvasTkAgg(fig, master=self.frame3)
167         canvas.draw()
168         canvas.get_tk_widget().place(relx=0.55, rely=0.5, anchor='center')
```

```
169
170         # COLUMN 2
171
172         # Frame 4
173         self.frame4 = customtkinter.CTkFrame(self, fg_color="#666666", width=200, height=100)
174         self.frame4.grid(row=0, column=1, padx=(10, 20), pady=(20, 0), sticky="nsew")
175
176         self.welcome = customtkinter.CTkLabel(master=self.frame4,
177                                               text=f"Welcome, {self.get_name(user_id).title()}",
178                                               font=customtkinter.CTkFont(size=18, weight="bold"))
179         self.welcome.place(relx=0.5, rely=0.25, anchor='center')
180         self.balance = customtkinter.CTkLabel(master=self.frame4,
181                                               text="Account balance:",
182                                               font=customtkinter.CTkFont(size=18, weight="bold"))
183         self.balance.place(relx=0.3, rely=0.5, anchor='center')
184         self.balance = customtkinter.CTkLabel(master=self.frame4,
185                                               text_color="#B7B7B7",
186                                               text=f"${(int(account_balance))}",
187                                               font=customtkinter.CTkFont(size=18, weight="bold"))
188         self.balance.place(relx=0.7, rely=0.5, anchor='center')
189         self.profitloss = customtkinter.CTkLabel(master=self.frame4,
190                                                  text="Total profit/loss:",
191                                                  font=customtkinter.CTkFont(size=18, weight="bold"))
192         self.profitloss.place(relx=0.3, rely=0.75, anchor='center')
```

```
        self.profitloss = customtkinter.CTkLabel(master=self.frame4,
                                                 text_color=self.define_colour(int(total_profit_or_loss)),
                                                 text=self.format_profitloss(int(total_profit_or_loss)),
                                                 font=customtkinter.CTkFont(size=18, weight="bold"))
        self.profitloss.place(relx=0.7, rely=0.75, anchor='center')

        # Frame 5
        self.frame5 = customtkinter.CTkFrame(self, fg_color="#999999", width=200, height=200)
        self.frame5.grid(row=1, column=1, rowspan=2, padx=(10, 20), pady=(20, 0), sticky="nsew")

        self.sentiment = customtkinter.CTkLabel(master=self.frame5,
                                                text="Sector sentiment",
                                                font=customtkinter.CTkFont(size=18, weight="bold"))
        self.sentiment.place(relx=0.5, rely=0.125, anchor='center')
        self.technology = customtkinter.CTkLabel(master=self.frame5,
                                                 text="Technology:",
                                                 font=customtkinter.CTkFont(size=12, weight="bold"))
        self.technology.place(relx=0.4, rely=0.25, anchor='center')
        self.technology_s = customtkinter.CTkLabel(master=self.frame5,
                                                   text=str(sentiment_values[5]),
                                                   text_color="#CCCCCC",
                                                   font=customtkinter.CTkFont(size=12, weight="bold"))
        self.technology_s.place(relx=0.8, rely=0.25, anchor='center')
        self.energy_transportation = customtkinter.CTkLabel(master=self.frame5,
                                                            text="Energy & Transportation:",
                                                            font=customtkinter.CTkFont(size=12, weight="bold"))
        self.energy_transportation.place(relx=0.4, rely=0.375, anchor='center')
        self.energy_transportation_s = customtkinter.CTkLabel(master=self.frame5,
                                                              text=str(sentiment_values[0]),
                                                              text_color="#CCCCCC",
                                                              font=customtkinter.CTkFont(size=12, weight="bold"))
        self.energy_transportation_s.place(relx=0.8, rely=0.375, anchor='center')
        self.manufacturing = customtkinter.CTkLabel(master=self.frame5,
                                                    text="Manufacturing:",
                                                    font=customtkinter.CTkFont(size=12, weight="bold"))
        self.manufacturing.place(relx=0.4, rely=0.5, anchor='center')
        self.manufacturing_s = customtkinter.CTkLabel(master=self.frame5,
                                                      text=str(sentiment_values[1]),
                                                      text_color="#CCCCCC",
                                                      font=customtkinter.CTkFont(size=12, weight="bold"))
        self.manufacturing_s.place(relx=0.8, rely=0.5, anchor='center')
        self.life_sciences = customtkinter.CTkLabel(master=self.frame5,
                                                    text="Life Sciences:",
                                                    font=customtkinter.CTkFont(size=12, weight="bold"))
        self.life_sciences.place(relx=0.4, rely=0.625, anchor='center')
        self.life_sciences_s = customtkinter.CTkLabel(master=self.frame5,
                                                      text=str(sentiment_values[2]),
                                                      text_color="#CCCCCC",
```

```
                                                font=customtkinter.CTkFont(size=12, weight="bold"))
        self.life_sciences_s.place(relx=0.8, rely=0.625, anchor='center')
        self.real_estate = customtkinter.CTkLabel(master=self.frame5,
                                                  text="Real Estate & Construction:",
                                                  font=customtkinter.CTkFont(size=12, weight="bold"))
        self.real_estate.place(relx=0.4, rely=0.75, anchor='center')
        self.real_estate_s = customtkinter.CTkLabel(master=self.frame5,
                                                    text=str(sentiment_values[3]),
                                                    text_color="#CCCCCC",
                                                    font=customtkinter.CTkFont(size=12, weight="bold"))
        self.real_estate_s.place(relx=0.8, rely=0.75, anchor='center')
        self.retail_wholesale = customtkinter.CTkLabel(master=self.frame5,
                                                       text="Retail & Wholesale:",
                                                       font=customtkinter.CTkFont(size=12, weight="bold"))
        self.retail_wholesale.place(relx=0.4, rely=0.875, anchor='center')
        self.retail_wholesale_s = customtkinter.CTkLabel(master=self.frame5,
                                                         text=str(sentiment_values[4]),
                                                         text_color="#CCCCCC",
                                                         font=customtkinter.CTkFont(size=12, weight="bold"))
        self.retail_wholesale_s.place(relx=0.8, rely=0.875, anchor='center')

        # Frame 6
        self.frame6 = customtkinter.CTkFrame(self, fg_color="#666666", width=200, height=50)
        self.frame6.grid(row=3, column=1, padx=(10, 20), pady=(20, 20), sticky="nsew")



        self.label1 = customtkinter.CTkLabel(master=self.frame6,
                                             text="Choose a stock to analyse:",
                                             font=customtkinter.CTkFont(size=22, weight="bold"))
        self.label1.place(relx=0.5, rely=0.3, anchor='center')
        self.entry = customtkinter.CTkEntry(master=self.frame6,
                                            placeholder_text="Symbol",
                                            font=customtkinter.CTkFont(size=12, weight="bold"), height=30,
                                            corner_radius=15)
        self.entry.place(relx=0.25, rely=0.7, anchor='center')
        self.main_button_1 = customtkinter.CTkButton(master=self.frame6,
                                                     text="Submit",
                                                     fg_color="transparent",
                                                     border_width=2,
                                                     font=customtkinter.CTkFont(size=12, weight="bold"),
                                                     height=30, corner_radius=15, command=self.submit)
        self.main_button_1.place(relx=0.75, rely=0.7, anchor='center')

        self.input_value = None

    1 usage
    def get_name(self, user_id):
        # Connect to the SQLite database
        conn = sqlite3.connect('database.db')
        c = conn.cursor()
```

```
        # Execute the query
        c.execute("SELECT name FROM details WHERE user_id = ?", (user_id,))
        # Fetch the result
        result = c.fetchone()
        # Close the connection
        conn.close()
        # If a result was found, return the name; otherwise, return None
        if result:
            return result[0]
        else:
            return None

    2 usages
    def update_time(self):
        now = datetime.datetime.now(timezone('US/Eastern'))
        current_day = now.weekday()  # Monday is 0 and Sunday is 6

        # If we hit 0 or the state is not initialized, we need to recalculate
        if self.total_seconds <= 0:
            if current_day > 4:  # Saturday or Sunday
                market = "Market is closed"
                status = "Time till market opens:"
                days_till_monday = 1 if current_day == 6 else 2
                next_open = datetime.datetime.combine(
                    now.date() + datetime.timedelta(days=days_till_monday), market_open

                )
                self.total_seconds = int((next_open - now).total_seconds())
            else:
                if now.time() >= market_open and now.time() <= market_close:
                    market = "Market is open"
                    status = "Time till market closes:"
                    self.total_seconds = int(
                        (datetime.datetime.combine(now.date(), market_close) - now).total_seconds()
                    )
                else:
                    market = "Market is closed"
                    status = "Time till market opens:"
                    if now.time() > market_close:
                        next_day = now + datetime.timedelta(days=1)
                        next_open = datetime.datetime.combine(next_day.date(), market_open)
                        self.total_seconds = int((next_open - now).total_seconds())
                    else:
                        self.total_seconds = int(
                            (datetime.datetime.combine(now.date(), market_open) - now).total_seconds()
                        )

            self.market.configure(text=market)
            self.status.configure(text=status)

```

```
        # Reduce the total_seconds by 1 for each call
        self.total_seconds -= 1

        hours, remainder = divmod(self.total_seconds, 3600)
        minutes, seconds = divmod(remainder, 60)
        countdown = "{:02d}:{:02d}:{:02d}".format(hours, minutes, seconds)

        self.countdown.configure(text=countdown)

        # Always reschedule every second since we're counting down
        self.after(1000, self.update_time)

    4 usages
    def define_colour(self, num):
        if num >= 0:
            colour = "#B6D7A8"  # Green
        elif num < 0:
            colour = "#EA9999"  # Red
        else:
            colour = "#EA58F9"  # Magenta
        return colour

    3 usages
    def format_watchlist(self, value):
        if value >= 0:
            formatted = f" + {value}%"

        elif value < 0:
            formatted = f" - {str(value)[1:]}%"
        else:
            formatted = "Error"
        return formatted

    1 usage
    def format_profitloss(self, value):
        if value >= 0:
            formatted = f" + ${value}"
        elif value < 0:
            formatted = f" - ${str(value)[1:]}"
        else:
            formatted = "Error"
        return formatted

    1 usage
    def validate_input(self):
        try:
            data = yf.Ticker(self.input_value).history(period='1d')  # Get recent 1-day trading data
            if data.empty:
                return False
            else:
                return True
        except Exception as e:
            print(f"Exception occurred: {e}")
```

```
385             return False
386
        1 usage
387     def submit(self):
388         current_time = time.time()
389
390         if current_time - self.entry_used_time >= 60:
391             self.input_value = self.entry.get().upper()
392             if self.validate_input():
393                 self.entry_used_time = current_time  # Update entry_used_time when a new window is opened
394                 self.open_new_window()
395             else:
396                 self.open_error_window()
397         else:
398             print("Rate Limit:", "Please wait for a minute before making another API call.")
399             self.open_rate_limit_window()
400
        1 usage
401     def open_rate_limit_window(self):
402         self.entry.delete(0, customtkinter.END)
403         rate_limit_window = customtkinter.CTkToplevel(self)
404         rate_limit_window.title("Login")
405         screen_width = 1440
406         screen_height = 900
407         window_width = 300
408         window_height = 200

409         x_position = (screen_width - window_width) // 2
410         y_position = (screen_height - window_height) // 2
411         rate_limit_window.geometry(f"{window_width}x{window_height}+{x_position}+{y_position}")
412         rate_limit_window.resizable(False, False)
413         rate_limit_window.configure(fg_color="#434343")
414         customtkinter.CTkLabel(rate_limit_window,
415                                text="Error: Rate Limit",
416                                font=customtkinter.CTkFont(size=22, weight="bold")).place(relx=0.5,
417                                                                                         rely=0.2,
418                                                                                         anchor='center')
419         customtkinter.CTkLabel(rate_limit_window,
420                                text_color="#CCCCCC",
421                                text="Please wait for a minute before making another API call",
422                                font=customtkinter.CTkFont(size=14, weight="bold"),
423                                wraplength=200).place(relx=0.5, rely=0.4, anchor='center')
424         customtkinter.CTkButton(rate_limit_window,
425                                 text="Close",
426                                 fg_color="transparent",
427                                 border_width=2,
428                                 font=customtkinter.CTkFont(size=12, weight="bold"),
429                                 height=30,
430                                 corner_radius=15,
431                                 command=rate_limit_window.destroy).place(relx=0.5, rely=0.7, anchor='center')
432
```

```
    1 usage
433 def open_error_window(self):
434     self.entry.delete(0, customtkinter.END)
435     error_window = customtkinter.CTkToplevel(self)
436     error_window.title("Login")
437     screen_width = 1440
438     screen_height = 900
439     window_width = 300
440     window_height = 200
441     x_position = (screen_width - window_width) // 2
442     y_position = (screen_height - window_height) // 2
443     error_window.geometry(f"{window_width}x{window_height}+{x_position}+{y_position}")
444     error_window.resizable(False, False)
445     error_window.configure(fg_color="#434343")
446     customtkinter.CTkLabel(error_window,
447                            text="Error: Invalid Input",
448                            font=customtkinter.CTkFont(size=22, weight="bold")).place(relx=0.5,
449                                                                                     rely=0.3,
450                                                                                     anchor='center')
451     customtkinter.CTkLabel(error_window,
452                            text_color="#CCCCCC",
453                            text="You need to enter a valid ticker symbol for analysis",
454                            font=customtkinter.CTkFont(size=14, weight="bold"),
455                            wraplength=200).place(relx=0.5, rely=0.45, anchor='center')
456     customtkinter.CTkButton(error_window,

457                             text="Close",
458                             fg_color="transparent",
459                             border_width=2,
460                             font=customtkinter.CTkFont(size=12, weight="bold"),
461                             height=30,
462                             corner_radius=15,
463                             command=error_window.destroy).place(relx=0.5, rely=0.7, anchor='center')
464
    1 usage
465 def open_new_window(self):
466     print("window button pressed")
467
468     new_window = customtkinter.CTkToplevel(self)
469
470     new_window.title("⧗ QuantHaven")
471     screen_width = 1440
472     screen_height = 900
473     window_width = 820
474     window_height = 580
475     x_position = (screen_width - window_width) // 2
476     y_position = (screen_height - window_height) // 2
477     new_window.geometry(f"{window_width}x{window_height}+{x_position}+{y_position}")
478     new_window.resizable(False, False)
479
480     symbol = self.input_value
```

```
        self.entry.delete(0, customtkinter.END)

        conn = sqlite3.connect('database.db')
        c = conn.cursor()
        c.execute("SELECT av_key FROM details WHERE user_id=?", (self.user_id,))
        record = c.fetchone()
        c.close()
        conn.close()
        AVKEY = security.decrypt_data(record[0], str(self.user_id) + 'C')
        overview = get_overview(symbol, AVKEY)
        news = get_news(symbol, AVKEY)

        # Configure grid
        new_window.grid_columnconfigure((0, 1), weight=1)
        new_window.grid_rowconfigure((0, 1, 2, 3), weight=1)
        new_window.configure(fg_color="#434343")

        # COLUMN 1

        # Frame 1
        new_window.frame1 = customtkinter.CTkFrame(new_window, fg_color="#666666", width=200, height=100)
        new_window.frame1.grid(row=0, column=0, padx=(20, 10), pady=(20, 0), sticky="nsew")

        new_window.symbol_l = customtkinter.CTkLabel(master=new_window.frame1,
                                                     text="Symbol:  ",
                                                     font=customtkinter.CTkFont(size=28, weight="bold"))
        new_window.symbol = customtkinter.CTkLabel(master=new_window.frame1,
                                                   text=symbol,
                                                   text_color="#B7B7B7",
                                                   font=customtkinter.CTkFont(size=28, weight="bold"))
        new_window.name_l = customtkinter.CTkLabel(master=new_window.frame1,
                                                   text="Name:  ",
                                                   font=customtkinter.CTkFont(size=18, weight="bold"))
        new_window.name = customtkinter.CTkLabel(master=new_window.frame1,
                                                 text=overview[0],
                                                 text_color="#B7B7B7",
                                                 font=customtkinter.CTkFont(size=18, weight="bold"),
                                                 wraplength=200)

        def place_labels():
            new_window.frame1.update_idletasks()
            center_point = (new_window.frame1.winfo_width() / 2) - 60
            total_width = new_window.symbol_l.winfo_reqwidth() + new_window.symbol.winfo_reqwidth()
            total_width_2 = new_window.name_l.winfo_reqwidth() + new_window.name.winfo_reqwidth()
            start_x1 = center_point - total_width / 2
            start_x2 = start_x1 + new_window.symbol_l.winfo_reqwidth()
            start_x1_2 = center_point - total_width_2 / 2
            start_x2_2 = start_x1_2 + new_window.name_l.winfo_reqwidth()
```

```
        new_window.name_l.place(x=start_x1_2, rely=0.2)  # adjust y as needed
        new_window.name.place(x=start_x2_2, rely=0.2)
        new_window.symbol_l.place(x=start_x1, rely=0.5)
        new_window.symbol.place(x=start_x2, rely=0.5)

    place_labels()

    # Frame 2
    new_window.frame2 = customtkinter.CTkFrame(new_window, fg_color="transparent", width=200, height=100)
    new_window.frame2.grid(row=1, column=0, padx=(20, 10), pady=(20, 0), sticky="nsew")

    def get_current_price(s):
        stock = yf.Ticker(s)
        hist = stock.history(period="1d")
        current_price = round(hist['Close'][0], 2)
        return current_price

    # Frame 2 left
    new_window.frame2l = customtkinter.CTkFrame(new_window.frame2, fg_color="#999999", width=100, height=100)
    new_window.frame2l.place(relx=0, rely=0, relwidth=0.48, relheight=1.0)
    new_window.close = customtkinter.CTkLabel(master=new_window.frame2l,
                                              text="Close prices",
                                              font=customtkinter.CTkFont(size=18, weight="bold"))
    new_window.close.place(relx=0.5, rely=0.2, anchor='center')

    new_window.currentprice_l = customtkinter.CTkLabel(master=new_window.frame2l,
                                                       text="Current:",
                                                       font=customtkinter.CTkFont(size=12, weight="bold"))
    new_window.currentprice_l.place(relx=0.3, rely=0.4, anchor='center')
    new_window.currentprice = customtkinter.CTkLabel(master=new_window.frame2l,
                                                     text=str(get_current_price(symbol)),
                                                     text_color="#CCCCCC",
                                                     font=customtkinter.CTkFont(size=12, weight="bold"))
    new_window.currentprice.place(relx=0.7, rely=0.4, anchor='center')
    new_window.MA50_l = customtkinter.CTkLabel(master=new_window.frame2l,
                                               text="50DMA:",
                                               font=customtkinter.CTkFont(size=12, weight="bold"))
    new_window.MA50_l.place(relx=0.3, rely=0.6, anchor='center')
    new_window.MA50 = customtkinter.CTkLabel(master=new_window.frame2l,
                                             text=overview[11],
                                             text_color="#CCCCCC",
                                             font=customtkinter.CTkFont(size=12, weight="bold"))
    new_window.MA50.place(relx=0.7, rely=0.6, anchor='center')
    new_window.MA200_l = customtkinter.CTkLabel(master=new_window.frame2l,
                                                text="200DMA",
                                                font=customtkinter.CTkFont(size=12, weight="bold"))
    new_window.MA200_l.place(relx=0.3, rely=0.8, anchor='center')
    new_window.MA200 = customtkinter.CTkLabel(master=new_window.frame2l,
                                              text=overview[12],
```

```
                                        text_color="#CCCCCC",
                                        font=customtkinter.CTkFont(size=12, weight="bold"))
        new_window.MA200.place(relx=0.7, rely=0.8, anchor='center')

        # Frame 2 right
        new_window.frame2r = customtkinter.CTkFrame(new_window.frame2, fg_color="#999999", width=100, height=100)
        new_window.frame2r.place(relx=0.52, rely=0, relwidth=0.48, relheight=1.0)
        new_window.about = customtkinter.CTkLabel(master=new_window.frame2r,
                                        text="About:",
                                        font=customtkinter.CTkFont(size=18, weight="bold"))
        new_window.about.place(relx=0.5, rely=0.2, anchor='center')
        new_window.exchange_l = customtkinter.CTkLabel(master=new_window.frame2r,
                                            text="Exchange:",
                                            font=customtkinter.CTkFont(size=12, weight="bold"))
        new_window.exchange_l.place(relx=0.3, rely=0.4, anchor='center')
        new_window.exchange = customtkinter.CTkLabel(master=new_window.frame2r,
                                            text=overview[2],
                                            text_color="#CCCCCC",
                                            font=customtkinter.CTkFont(size=12, weight="bold"))
        new_window.exchange.place(relx=0.7, rely=0.4, anchor='center')
        new_window.currency_l = customtkinter.CTkLabel(master=new_window.frame2r,
                                            text="Currency",
                                            font=customtkinter.CTkFont(size=12, weight="bold"))
        new_window.currency_l.place(relx=0.3, rely=0.6, anchor='center')

        new_window.currency = customtkinter.CTkLabel(master=new_window.frame2r,
                                            text=overview[3],
                                            text_color="#CCCCCC",
                                            font=customtkinter.CTkFont(size=12, weight="bold"))
        new_window.currency.place(relx=0.7, rely=0.6, anchor='center')
        new_window.country_l = customtkinter.CTkLabel(master=new_window.frame2r,
                                            text="Country:",
                                            font=customtkinter.CTkFont(size=12, weight="bold"))
        new_window.country_l.place(relx=0.3, rely=0.8, anchor='center')
        new_window.country = customtkinter.CTkLabel(master=new_window.frame2r,
                                            text=overview[4],
                                            text_color="#CCCCCC",
                                            font=customtkinter.CTkFont(size=12, weight="bold"))
        new_window.country.place(relx=0.7, rely=0.8, anchor='center')

        # Frame 3
        new_window.frame3 = customtkinter.CTkFrame(new_window, fg_color="#666666", width=200, height=200)
        new_window.frame3.grid(row=2, column=0, rowspan=2, padx=(20, 10), pady=(20, 20), sticky="nsew")

        data = yf.download(tickers=symbol, period="5y", interval="1d")
        close = data['Close']

        title_font_prop = font_manager.FontProperties(weight='bold', size=14, family='Roboto')
        label_font_prop = font_manager.FontProperties(weight='bold', size=8, family='Roboto')
```

```
625
626     fig = figure.Figure(figsize=(4, 2), dpi=100)
627     fig.set_facecolor("#666666")
628     ax = fig.add_subplot(111)
629     ax.set_facecolor("#666666")
630     ax.plot_date(close.index, close, linestyle='-', marker=None, linewidth=1, color='white')
631     ax.grid(False)
632     ax.spines['left'].set_visible(False)
633     ax.spines['right'].set_visible(False)
634     ax.spines['bottom'].set_visible(False)
635     ax.spines['top'].set_visible(False)
636
637     ax.set_title('Historical Close Price', color='white', fontproperties=title_font_prop)
638
639     ax.tick_params(colors='white')
640     for label in ax.get_xticklabels():
641         label.set_fontproperties(label_font_prop)
642     for label in ax.get_yticklabels():
643         label.set_fontproperties(label_font_prop)
644
645     canvas = FigureCanvasTkAgg(fig, master=new_window.frame3)
646     canvas.draw()
647     canvas.get_tk_widget().place(relx=0.55, rely=0.5, anchor='center')
648
649     # COLUMN 2
650
651     # Frame 4
652     new_window.textbox1 = customtkinter.CTkTextbox(new_window, fg_color="#000000", width=200, height=100)
653     new_window.textbox1.grid(row=0, column=1, padx=(10, 20), pady=(20, 0), sticky="nsew")
654     new_window.textbox1.insert(
655         "0.0", "COMPANY OVERVIEW\n\n" + f"Sector: {overview[5].title()}\n\n" + f"Description: {overview[1]}"
656     )
657     new_window.textbox1.configure(state="disabled")
658
659     # Frame 5
660     new_window.textbox2 = customtkinter.CTkTextbox(new_window, fg_color="#000000", width=200, height=150)
661     new_window.textbox2.grid(row=1, column=1, rowspan=2, padx=(10, 20), pady=(20, 0), sticky="nsew")
662     new_window.textbox2.insert(
663         "0.0",
664         "NEWS\n\n" +
665         f"Title: {news[0].title()}\n" +
666         f"Source: {news[1]}\n" +
667         f"Date Published: {news[2]}\n\n" +
668         f"Title: {news[3].title()}\n" +
669         f"Source: {news[4]}\n" +
670         f"Date Published: {news[5]}\n\n" +
671         f"Title: {news[6].title()}\n" +
672         f"Source: {news[7]}\n" +
```

```
        f"Date Published: {news[8]}\n\n" +
        f"Title: {news[9].title()}\n" +
        f"Source: {news[10]}\n" +
        f"Date Published: {news[11]}\n\n" +
        f"Title: {news[12].title()}\n" +
        f"Source: {news[13]}\n" +
        f"Date Published: {news[14]}"
    )
    new_window.textbox2.configure(state="disabled")

    # Frame 6
    new_window.frame6 = customtkinter.CTkFrame(new_window, fg_color="transparent", width=200, height=100)
    new_window.frame6.grid(row=3, column=1, padx=(10, 20), pady=(20, 20), sticky="nsew")

    padding = 0.1
    scale_factor = (1 - 2 * padding) / 6

    # Frame 6 left
    new_window.frame6l = customtkinter.CTkFrame(new_window.frame6, fg_color="#999999", width=100, height=100)
    new_window.frame6l.place(relx=0, rely=0, relwidth=0.48, relheight=1.0)

    new_window.eps_l = customtkinter.CTkLabel(master=new_window.frame6l,
                                              text="EPS:",
                                              font=customtkinter.CTkFont(size=12, weight="bold"))

    new_window.eps_l.place(relx=0.5, rely=padding + scale_factor * 0.5, anchor='center')
    new_window.eps = customtkinter.CTkLabel(master=new_window.frame6l,
                                            text=overview[8],
                                            text_color="#CCCCCC",
                                            font=customtkinter.CTkFont(size=12, weight="bold"))
    new_window.eps.place(relx=0.5, rely=padding + scale_factor * 1.5, anchor='center')
    new_window.pe_ratio_l = customtkinter.CTkLabel(master=new_window.frame6l,
                                                   text="PE Ratio:",
                                                   font=customtkinter.CTkFont(size=12, weight="bold"))
    new_window.pe_ratio_l.place(relx=0.5, rely=padding + scale_factor * 2.5, anchor='center')
    new_window.pe_ratio = customtkinter.CTkLabel(master=new_window.frame6l,
                                                 text=overview[7],
                                                 text_color="#CCCCCC",
                                                 font=customtkinter.CTkFont(size=12, weight="bold"))
    new_window.pe_ratio.place(relx=0.5, rely=padding + scale_factor * 3.5, anchor='center')
    new_window.beta_l = customtkinter.CTkLabel(master=new_window.frame6l,
                                               text="Beta:",
                                               font=customtkinter.CTkFont(size=12, weight="bold"))
    new_window.beta_l.place(relx=0.5, rely=padding + scale_factor * 4.5, anchor='center')
    new_window.beta = customtkinter.CTkLabel(master=new_window.frame6l,
                                             text=overview[10],
                                             text_color="#CCCCCC",
                                             font=customtkinter.CTkFont(size=12, weight="bold"))
    new_window.beta.place(relx=0.5, rely=padding + scale_factor * 5.5, anchor='center')
```

```
721
722         #  Frame 6 right
723         new_window.frame6r = customtkinter.CTkFrame(new_window.frame6, fg_color="#999999", width=100, height=100)
724         new_window.frame6r.place(relx=0.52, rely=0, relwidth=0.48, relheight=1.0)
725
726         new_window.shares_outstanding_l = customtkinter.CTkLabel(master=new_window.frame6r,
727                                                                  text="Shares Outstanding:",
728                                                                  font=customtkinter.CTkFont(size=12, weight="bold"))
729         new_window.shares_outstanding_l.place(relx=0.5, rely=padding + scale_factor * 0.5, anchor='center')
730         new_window.shares_outstanding = customtkinter.CTkLabel(master=new_window.frame6r,
731                                                                text=overview[13],
732                                                                text_color="#CCCCCC",
733                                                                font=customtkinter.CTkFont(size=12, weight="bold"))
734         new_window.shares_outstanding.place(relx=0.5, rely=padding + scale_factor * 1.5, anchor='center')
735         new_window.market_capitalisation_l = customtkinter.CTkLabel(master=new_window.frame6r,
736                                                                     text="Market Capitalisation:",
737                                                                     font=customtkinter.CTkFont(size=12, weight="bold"))
738         new_window.market_capitalisation_l.place(relx=0.5, rely=padding + scale_factor * 2.5, anchor='center')
739         new_window.market_capitalisation = customtkinter.CTkLabel(master=new_window.frame6r,
740                                                                   text=overview[6],
741                                                                   text_color="#CCCCCC",
742                                                                   font=customtkinter.CTkFont(size=12, weight="bold"))
743         new_window.market_capitalisation.place(relx=0.5, rely=padding + scale_factor * 3.5, anchor='center')
744         new_window.profit_margin_l = customtkinter.CTkLabel(master=new_window.frame6r,

745                                                             text="Profit Margin:",
746                                                             font=customtkinter.CTkFont(size=12, weight="bold"))
747         new_window.profit_margin_l.place(relx=0.5, rely=padding + scale_factor * 4.5, anchor='center')
748         new_window.profit_margin = customtkinter.CTkLabel(master=new_window.frame6r,
749                                                           text=overview[9],
750                                                           text_color="#CCCCCC",
751                                                           font=customtkinter.CTkFont(size=12, weight="bold"))
752         new_window.profit_margin.place(relx=0.5, rely=padding + scale_factor * 5.5, anchor='center')
753
```

home_datagrab.py

```python
# IMPORTING API
from alpaca.trading.client import TradingClient
# IMPORTING MODULES
import requests
import yfinance as yf
import sqlite3
# IMPORTING USER-DEFINED MODULES
import security


2 usages
def home_data(user_id):
    conn = sqlite3.connect('database.db')
    c = conn.cursor()
    c.execute("SELECT api_key, api_secret_key, av_key FROM details WHERE user_id=?", (user_id,))
    record = c.fetchone()
    c.close()
    conn.close()

    API_KEY = security.decrypt_data(record[0], str(user_id) + 'A')
    API_SECRET_KEY = security.decrypt_data(record[1], str(user_id) + 'B')
    AVKEY = security.decrypt_data(record[2], str(user_id) + 'C')

    trading_client = TradingClient(API_KEY, API_SECRET_KEY, paper=True)


    # BALANCE AND PROFIT/LOSS

    account_balance = float(trading_client.get_account().cash)
    account_equity = 100000
    total_profit_or_loss = -(float(account_equity) - float(account_balance))

    # SECTOR SENTIMENT

    topics = {'energy_transportation': 'Energy & Transportation',
              'manufacturing': 'Manufacturing',
              'life_sciences': 'Life Sciences',
              'real_estate': 'Real Estate & Construction',
              'retail_wholesale': 'Retail & Wholesale',
              'technology': 'Technology'
              }

    sentiment_values = []

    for topic in topics:
        url3 = f'https://www.alphavantage.co/query?function=NEWS_SENTIMENT&topics={topic}&apikey={AVKEY}'
        r3 = requests.get(url3)
        data_latest = r3.json()

        def calculate_weighted_sentiment(data, current_topic):
```

```
        sentiment_sum = 0.0
        relevance_sum = 0.0

        for item in data["feed"]:

            relevance = 0.0

            for topic_item in item["topics"]:
                if topic_item["topic"] == topics[current_topic]:
                    relevance = float(topic_item["relevance_score"])

            sentiment = float(item["overall_sentiment_score"])

            sentiment_sum += sentiment * relevance
            relevance_sum += relevance

        if relevance_sum == 0.0:
            return None

        return sentiment_sum / relevance_sum

    val = calculate_weighted_sentiment(data_latest, topic)
    sentiment_values.append(round(val, 2))


# TOP 3 GAINERS

top_3 = []

url1 = f'https://www.alphavantage.co/query?function=TOP_GAINERS_LOSERS&apikey={AVKEY}'
r1 = requests.get(url1)
data1 = r1.json()

top_gainers = data1['top_gainers'][:3]

for gainer in top_gainers:
    top_3.append(f"{gainer['ticker']}: + {round(float(gainer['change_percentage'][:-1]))}%")

# WATCHLIST

watchlist = ['SPY', 'QQQ', 'DIA']
watchlist_stats = {}

for symbol in watchlist:
    ticker = yf.Ticker(symbol)
    hist = ticker.history(period="2d")  # Get last 2 days' historical market data
    latest = hist['Close'].iloc[-1]  # The latest closing price
    previous = hist['Close'].iloc[-2]  # The previous closing price
    percent_change = round(((latest - previous) / previous) * 100, 2)

    watchlist_stats[symbol] = percent_change

final = [account_balance, total_profit_or_loss, sentiment_values, top_3, watchlist_stats]

return final
```

popup.py

```python
import requests
from datetime import datetime

# OVERVIEW

2 usages
def get_overview(symbol, AVKEY):
    url1 = f'https://www.alphavantage.co/query?function=OVERVIEW&symbol={symbol}&apikey={AVKEY}'
    r1 = requests.get(url1)
    data_latest1 = r1.json()

    name = data_latest1["Name"]
    description = data_latest1["Description"]
    exchange = data_latest1["Exchange"]
    currency = data_latest1["Currency"]
    country = data_latest1["Country"]
    sector = data_latest1["Sector"]
    market_capitalisation = data_latest1["MarketCapitalization"]
    pe_ratio = data_latest1["PERatio"]
    eps = data_latest1["EPS"]
    profit_margin = data_latest1["ProfitMargin"]
    beta = data_latest1["Beta"]
    MA50 = data_latest1["50DayMovingAverage"]
    MA200 = data_latest1["200DayMovingAverage"]
    shares_outstanding = data_latest1["SharesOutstanding"]


    overview = [name, description, exchange, currency, country, sector, market_capitalisation, pe_ratio, eps,
                profit_margin, beta, MA50, MA200, shares_outstanding]
    return overview

# NEWS

2 usages
def get_news(symbol, AVKEY):
    news = []

    url3 = f'https://www.alphavantage.co/query?function=NEWS_SENTIMENT&tickers={symbol}&limit=5&apikey={AVKEY}'
    r3 = requests.get(url3)
    data_latest3 = r3.json()

    for item in data_latest3["feed"]:
        title = item["title"]
        news.append(title)
        source = item["source"]
        news.append(source)
        date = item["time_published"]
        dt = datetime.strptime(date, "%Y%m%dT%H%M%S")
        date_str = dt.strftime("%Y-%m-%d")
        news.append(date_str)
    return news

```

bot.py

```
from tkinter import ttk
import time
import customtkinter
import sqlite3
import yfinance as yf


2 usages
class PageOne(customtkinter.CTkFrame):
    def __init__(self, master, task, user_id, controller=None):
        customtkinter.CTkFrame.__init__(self, master)
        self.controller = controller
        self.task = task
        self.user_id = user_id
        self.window_opened = None
        self.MAX_STOCKS = 3
        self.last_api_call_time = 0
        self.entry_used_time = 0
        self.total_seconds = 0

        # Configure grid
        self.screen_width = 1440
        self.screen_height = 900
        self.grid_columnconfigure(0, weight=1)
        self.grid_rowconfigure(0, weight=1)

        self.configure(fg_color="#666666")

        # Set style
        style = ttk.Style(self)
        style.theme_use("clam")
        # Eliminate borders and lines
        style.layout("Treeview", [('Treeview.treearea', {'sticky': 'nswe'})])
        # Configure Treeview style
        style.configure("Treeview",
                        background="#000000",  # Black background
                        fieldbackground="#000000",  # Black field background
                        foreground="#ffffff",  # White text
                        borderwidth=0,  # No border
                        highlightthickness=0,  # No highlight
                        bd=0,  # No border width
                        font=('Roboto', 14))
        # Configure Treeview Heading style
        style.configure("Treeview.Heading",
                        background="#434343",  # Dark gray heading background
                        foreground="#ffffff",  # White heading text
                        borderwidth=0,  # No border
                        font=('Roboto', 14, 'bold'))

        # Create the table
```

```
        self.table = ttk.Treeview(self,
                                  columns=("id", "time_unit", "time_amount", "bot_type", "stocks", "strategies"),
                                  show='headings')
        self.table.heading("#1", text="ID")
        self.table.heading("#2", text="Time Frame")
        self.table.heading("#3", text="Bar Frequency")
        self.table.heading("#4", text="Bot Type")
        self.table.heading("#5", text="Stocks")
        self.table.heading("#6", text="Strategies")
        self.table.grid(row=0, column=0, sticky="nsew", rowspan=8)
        self.table.column('#1', stretch=False, minwidth=0, width=0)

        # Connect to the SQLite database
        self.conn = sqlite3.connect('database.db')
        self.conn.execute("PRAGMA foreign_keys = ON")
        self.c = self.conn.cursor()

        self.c.execute("""
            CREATE TABLE IF NOT EXISTS configurations(
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                user_id INTEGER,
                time_unit TEXT,
                time_amount TEXT,
                bot_type TEXT,
                stocks TEXT,
                strategies TEXT,
                FOREIGN KEY(user_id) REFERENCES users(id) ON DELETE CASCADE
            )
        """)

        # Frame
        widget_frame = customtkinter.CTkFrame(self, fg_color="#666666")
        widget_frame.grid(row=0, column=1, padx=25, sticky="nsew")

        # GUI setup
        self.setup_gui(widget_frame)

        # Load the data into the table
        self.load_data()

    1 usage
    def setup_gui(self, frame):
        frame.grid_rowconfigure(0, weight=1)
        frame.grid_rowconfigure(12, weight=1)

        # Delete widgets
        delete_label = customtkinter.CTkLabel(frame, text="Delete configuration:", font=("Roboto", 12, "bold"))
        self.delete_button = customtkinter.CTkButton(frame,
                                                     text="Delete",
```

```
                                            command=self.delete_record,
                                            font=customtkinter.CTkFont(size=12, weight="bold"),
                                            height=30,
                                            corner_radius=15,
                                            state="disabled")

        delete_label.grid(row=1, column=0, sticky="n", pady=(0, 10))
        self.delete_button.grid(row=2, column=0, sticky="n", pady=(0, 20))

        # Add widgets
        add_label = customtkinter.CTkLabel(frame, text="Add a configuration:", font=("Roboto", 12, "bold"))
        add_unit_label = customtkinter.CTkLabel(frame, text="Time  Unit:", font=("Roboto", 10, "bold"))
        self.unit = customtkinter.StringVar()
        self.unit_entry = customtkinter.CTkEntry(frame,
                                                 font=("Roboto", 12, "bold"),
                                                 height=30,
                                                 corner_radius=15,
                                                 textvariable=self.unit)
        add_amount_label = customtkinter.CTkLabel(frame, text="Time  Amount:", font=("Roboto", 10, "bold"))
        self.amount = customtkinter.StringVar()
        self.amount_entry = customtkinter.CTkEntry(frame,
                                                   font=("Roboto", 12, "bold"),
                                                   height=30,
                                                   corner_radius=15,

                                                   textvariable=self.amount)

        add_label.grid(row=3, column=0, sticky="n", pady=(0, 10))
        add_unit_label.grid(row=4, column=0, sticky="n")
        self.unit_entry.grid(row=5, column=0, sticky="n", pady=(0, 10))
        add_amount_label.grid(row=6, column=0, sticky="n")
        self.amount_entry.grid(row=7, column=0, sticky="n")

        self.error_label = customtkinter.CTkLabel(frame, text="", text_color="#EA9999", font=("Roboto", 12, "bold"))
        self.add_button = customtkinter.CTkButton(frame,
                                                  text="Add",
                                                  command=self.validate,
                                                  font=customtkinter.CTkFont(size=12, weight="bold"),
                                                  height=30,
                                                  corner_radius=15)

        self.error_label.grid(row=8, column=0, sticky="n")
        self.add_button.grid(row=9, column=0, sticky="n", pady=(10, 20))

        # Run widgets
        run_label = customtkinter.CTkLabel(frame, text="Run configuration:", font=("Roboto", 12, "bold"))
        self.run_button = customtkinter.CTkButton(frame,
                                                  text="Run",
                                                  command=self.handle_api_limit,
```

```
                                            font=customtkinter.CTkFont(size=12, weight="bold"),
                                            state="disabled",
                                            height=30,
                                            corner_radius=15)

        run_label.grid(row=10, column=0, sticky="n", pady=(0, 10))
        self.run_button.grid(row=11, column=0, sticky="n", pady=(0, 0))

        self.table.bind("<<TreeviewSelect>>", self.on_select)

    1 usage
    def open_window_1(self):
        print("window button 1 pressed")
        self.error_label.configure(text="")
        self.window_1 = customtkinter.CTkToplevel(self)
        self.window_opened = "window 1"

        self.window_1.title("Customisation")
        window_width = 250
        window_height = 350
        x_position = (self.screen_width - window_width) // 2
        y_position = (self.screen_height - window_height) // 2
        self.window_1.geometry(f"{window_width}x{window_height}+{x_position}+{y_position}")
        self.window_1.resizable(False, False)
        self.window_1.configure(fg_color="#434343")

        self.window_1.grid_rowconfigure(0, weight=1)
        self.window_1.grid_rowconfigure(9, weight=1)
        self.window_1.grid_columnconfigure(0, weight=1)
        self.window_1.grid_columnconfigure(2, weight=1)

        customtkinter.CTkLabel(master=self.window_1,
                               text="Customisation",
                               font=("Roboto", 18, "bold")).grid(row=1, column=1, pady=(0, 10))
        self.var1_1 = customtkinter.StringVar()
        customtkinter.CTkCheckBox(master=self.window_1,
                                  text="Momentum",
                                  variable=self.var1_1,
                                  onvalue="Momentum",
                                  offvalue="",
                                  font=("Roboto", 12, "bold")).grid(row=2,
                                                                    column=1,
                                                                    padx=(20, 0),
                                                                    pady=(0, 10), sticky="w")
        self.var1_2 = customtkinter.StringVar()
        customtkinter.CTkCheckBox(master=self.window_1,
                                  text="SMACrossover",
                                  variable=self.var1_2,
                                  onvalue="SMACrossover",
                                  offvalue="",
```

```
                            font=("Roboto", 12, "bold")).grid(row=3,
                                                              column=1,
                                                              padx=(20, 0),
                                                              pady=(0, 10),
                                                              sticky="w")
        self.var1_3 = customtkinter.StringVar()
        customtkinter.CTkCheckBox(master=self.window_1,
                                  text="Breakout",
                                  variable=self.var1_3,
                                  onvalue="Breakout",
                                  offvalue="",
                                  font=("Roboto", 12, "bold")).grid(row=4,
                                                                    column=1,
                                                                    padx=(20, 0),
                                                                    pady=(0, 10),
                                                                    sticky="w")
        self.var1_4 = customtkinter.StringVar()
        customtkinter.CTkCheckBox(master=self.window_1,
                                  text="SVR",
                                  variable=self.var1_4,
                                  onvalue="SVRStrategy",
                                  offvalue="",
                                  font=("Roboto", 12, "bold")).grid(row=5,
                                                                    column=1,
                                                                    padx=(20, 0),
                                                                    pady=(0, 15),
                                                                    sticky="w")

        self.stocks_entry_1 = customtkinter.CTkEntry(master=self.window_1,
                                                     placeholder_text="Stocks",
                                                     font=("Roboto", 12, "bold"),
                                                     height=30,
                                                     corner_radius=15)
        self.stocks_entry_1.grid(row=6, column=1)

        self.window_error_label_1 = customtkinter.CTkLabel(master=self.window_1,
                                                           text="",
                                                           text_color="#EA9999",
                                                           font=("Roboto", 12, "bold"))
        self.window_error_label_1.grid(row=7, column=1, pady=(0, 10))

        customtkinter.CTkButton(master=self.window_1,
                                text="Done",
                                command=self.submit,
                                font=customtkinter.CTkFont(size=12, weight="bold"),
                                height=30,
                                corner_radius=15).grid(row=8, column=1, pady=(0, 10))
        self.window_1.bind("<Destroy>", self.enable_add_button)
```

```
241
        1 usage
242     def open_window_2(self):
243         print("window button 2 pressed")
244         self.error_label.configure(text="")
245         self.window_2 = customtkinter.CTkToplevel(self)
246         self.window_opened = "window 2"
247         self.window_2.title("Customisation")
248         window_width = 250
249         window_height = 300
250         x_position = (self.screen_width - window_width) // 2
251         y_position = (self.screen_height - window_height) // 2
252         self.window_2.geometry(f"{window_width}x{window_height}+{x_position}+{y_position}")
253         self.window_2.resizable(False, False)
254         self.window_2.configure(fg_color="#434343")
255         self.window_2.grid_rowconfigure(0, weight=1)
256         self.window_2.grid_rowconfigure(8, weight=1)
257         self.window_2.grid_columnconfigure(0, weight=1)
258         self.window_2.grid_columnconfigure(2, weight=1)
259
260         customtkinter.CTkLabel(master=self.window_2,
261                                text="Customisation",
262                                font=("Roboto", 18, "bold")).grid(row=1, column=1, pady=(0, 10))
263         self.var2_1 = customtkinter.StringVar()
264         customtkinter.CTkCheckBox(master=self.window_2,
265                                   text="Momentum",
266                                   variable=self.var2_1,
267                                   onvalue="Momentum",
268                                   offvalue="",
269                                   font=("Roboto", 12, "bold")).grid(row=2,
270                                                                     column=1,
271                                                                     padx=(20, 0),
272                                                                     pady=(0, 10),
273                                                                     sticky="w")
274         self.var2_2 = customtkinter.StringVar()
275         customtkinter.CTkCheckBox(master=self.window_2,
276                                   text="Scalping",
277                                   variable=self.var2_2,
278                                   onvalue="Scalping",
279                                   offvalue="",
280                                   font=("Roboto", 12, "bold")).grid(row=3,
281                                                                     column=1,
282                                                                     padx=(20, 0),
283                                                                     pady=(0, 10),
284                                                                     sticky="w")
285         self.var2_3 = customtkinter.StringVar()
286         customtkinter.CTkCheckBox(master=self.window_2,
287                                   text="Mean Reversion",
288                                   variable=self.var2_3,
```

```
                                onvalue="MeanReversion",
                                offvalue="",
                                font=("Roboto", 12, "bold")).grid(row=4,
                                                                  padx=(20, 0),
                                                                  column=1,
                                                                  pady=(0, 15),
                                                                  sticky="w")

        self.stocks_entry_2 = customtkinter.CTkEntry(master=self.window_2,
                                                     placeholder_text="Stocks",
                                                     font=("Roboto", 12, "bold"),
                                                     height=30,
                                                     corner_radius=15)
        self.stocks_entry_2.grid(row=5, column=1)

        self.window_error_label_2 = customtkinter.CTkLabel(master=self.window_2,
                                                           text="",
                                                           text_color="#EA9999",
                                                           font=("Roboto", 12, "bold"))
        self.window_error_label_2.grid(row=6, column=1, pady=(0, 10))

        customtkinter.CTkButton(master=self.window_2,
                                text="Done",
                                command=self.submit,

                                font=customtkinter.CTkFont(size=12, weight="bold"),
                                height=30, corner_radius=15).grid(row=7, column=1, pady=(0, 10))
        self.window_2.bind("<Destroy>", self.enable_add_button)

    2 usages
    def enable_add_button(self, event=None):
        self.add_button.configure(state='normal')

    1 usage
    def open_rate_limit_window(self):
        rate_limit_window = customtkinter.CTkToplevel(self)
        rate_limit_window.title("Login")
        screen_width = 1440
        screen_height = 900
        window_width = 300
        window_height = 200
        x_position = (screen_width - window_width) // 2
        y_position = (screen_height - window_height) // 2
        rate_limit_window.geometry(f"{window_width}x{window_height}+{x_position}+{y_position}")
        rate_limit_window.resizable(False, False)
        rate_limit_window.configure(fg_color="#434343")
        customtkinter.CTkLabel(rate_limit_window,
                               text="Error: Rate Limit",
                               font=customtkinter.CTkFont(size=22, weight="bold")).place(relx=0.5,
                                                                                        rely=0.2,
                                                                                        anchor='center')
```

```
337             customtkinter.CTkLabel(rate_limit_window,
338                                    text_color="#CCCCCC",
339                                    text="Please wait for a minute before making another API call",
340                                    font=customtkinter.CTkFont(size=14, weight="bold"),
341                                    wraplength=200).place(relx=0.5, rely=0.4, anchor='center')
342             customtkinter.CTkButton(rate_limit_window,
343                                     text="Close",
344                                     fg_color="transparent",
345                                     border_width=2,
346                                     font=customtkinter.CTkFont(size=12, weight="bold"),
347                                     height=30,
348                                     corner_radius=15,
349                                     command=rate_limit_window.destroy).place(relx=0.5, rely=0.7, anchor='center')
350
        2 usages
351         def submit(self):
352             if self.window_opened == "window 1":
353                 options_1 = [self.var1_1.get(), self.var1_2.get(), self.var1_3.get(), self.var1_4.get()]
354                 checked_options_1 = [option for option in options_1 if option != ""]
355                 result_1 = ', '.join(checked_options_1)
356                 stocks_str = self.stocks_entry_1.get()
357                 if result_1 == "":
358                     self.window_error_label_1.configure(text="Please check at least one box")
359                 elif not stocks_str:
360                     self.window_error_label_1.configure(text="Please enter at lest one stock")

361                 elif self.validate_tickers(stocks_str) == "Too many":
362                     self.window_error_label_1.configure(text="Too many stocks entered.")
363                 elif self.validate_tickers(stocks_str) == "Invalid":
364                     self.window_error_label_1.configure(text="Stock(s) invalid")
365                 else:
366                     self.add_record(result_1, stocks_str)
367                     self.window_1.destroy()
368                     self.add_button.configure(state='normal')
369             elif self.window_opened == "window 2":
370                 options_2 = [self.var2_1.get(), self.var2_2.get(), self.var2_3.get()]
371                 checked_options_2 = [option for option in options_2 if option != ""]
372                 result_2 = ', '.join(checked_options_2)
373                 stocks_str = self.stocks_entry_2.get()
374                 if result_2 == "":
375                     self.window_error_label_2.configure(text="Please check at least one box")
376                 elif not stocks_str:
377                     self.window_error_label_2.configure(text="Please enter at lest one stock")
378                 elif self.validate_tickers(stocks_str) == "Too many":
379                     self.window_error_label_2.configure(text="Too many stocks entered.")
380                 elif self.validate_tickers(stocks_str) == "Invalid":
381                     self.window_error_label_2.configure(text="Stock(s) invalid")
382                 else:
383                     self.add_record(result_2, stocks_str)
384                     self.window_2.destroy()
```

```
385                 self.add_button.configure(state='normal')
386             else:
387                 print("error no window opened")
388
        1 usage
389     def validate(self):
390         time_unit = self.unit_entry.get().title()
391         time_amount = self.amount_entry.get().title()
392
393         if not time_unit or not time_amount:
394             self.error_label.configure(text="Please enter a time frame")
395         else:
396             minutes = self.convert_to_base(unit=time_unit, amount=time_amount)
397             if minutes == "Unsupported unit":
398                 self.error_label.configure(text="Unsupported unit")
399             if minutes == "Amount must be an integer":
400                 self.error_label.configure(text="Amount must be an integer")
401             elif int(minutes) <= 15:
402                 self.open_window_2()
403                 self.add_button.configure(state='disabled')
404             else:
405                 self.open_window_1()
406                 self.add_button.configure(state='disabled')
407
        1 usage
408     def convert_to_base(self, unit, amount):

409         try:
410             x = int(amount)
411             if unit == "Day":
412                 return x * 24 * 60
413             elif unit == "Hour":
414                 return x * 60
415             elif unit == "Minute":
416                 return x
417             else:
418                 return "Unsupported unit"
419         except ValueError:
420             return "Amount must be an integer"
421
        4 usages
422     def validate_tickers(self, tickers_str):
423         tickers = tickers_str.split(', ')
424
425         if len(tickers) > self.MAX_STOCKS:
426             print(f"Error: Too many tickers entered.")
427             return "Too many"
428
429         for ticker in tickers:
430             try:
431                 data = yf.Ticker(ticker).history(period='1d')  # Get recent 1-day trading data
432                 if data.empty:  # If the data is empty, ticker is invalid
```

```
                    return "Invalid"
            except Exception as e:  # If any exception occurs, ticker is invalid
                print(f"Exception occurred: {e}")
                return "Invalid"

    return "Valid"

3 usages
def load_data(self):
    # Remove all existing rows from the table
    for row in self.table.get_children():
        self.table.delete(row)
    # Query the database
    self.c.execute(
        "SELECT id, time_unit, time_amount, bot_type, stocks, strategies FROM configurations WHERE user_id = ?",
        (self.user_id,)
    )
    rows = self.c.fetchall()
    # Insert data into the table
    for row in rows:
        self.table.insert('', 'end', values=row)

1 usage
def on_select(self, event):
    selected = self.table.selection()
    if selected:

        self.run_button.configure(state="normal")
        self.delete_button.configure(state="normal")
    else:
        self.run_button.configure(state="disabled")
        self.delete_button.configure(state="disabled")

1 usage
def delete_record(self):
    selected_id = self.table.selection()[0]  # Get selected item's ID
    selected_item = self.table.item(selected_id)  # Get selected item's details
    selected_values = selected_item['values']
    id_to_delete = selected_values[0]
    # Check if the user owns this data
    self.c.execute('SELECT user_id FROM configurations WHERE id = ?', (id_to_delete,))
    owner_id = self.c.fetchone()[0]
    if owner_id == self.user_id:
        # If the user owns this data, delete it
        self.c.execute("DELETE FROM configurations WHERE id=?", (id_to_delete,))
        self.conn.commit()
        self.load_data()
    else:
        print("Error user does not own data to delete")

2 usages
def add_record(self, strategies, stocks):
    unit_to_add = self.unit_entry.get().title()
```

```
        amount_to_add = self.amount_entry.get().title()
        stocks_to_add = stocks
        strategies_to_add = strategies
        if self.window_opened == "window 1":
            bot_type_to_add = "Historical"
        else:
            bot_type_to_add = "Live"
        # Autoincrement for id
        self.c.execute("INSERT INTO configurations (time_unit, time_amount, bot_type, stocks, strategies, user_id) "
                       "VALUES (?, ?, ?, ?, ?, ?)",
                       (unit_to_add, amount_to_add, bot_type_to_add, stocks_to_add, strategies_to_add, self.user_id)
                       )
        self.conn.commit()
        self.load_data()
        # Clear the entry fields
        self.unit_entry.delete(0, customtkinter.END)
        self.amount_entry.delete(0, customtkinter.END)

    1 usage
    def handle_api_limit(self):
        current_time = time.time()

        if current_time - self.last_api_call_time >= 60:
            self.last_api_call_time = current_time
            self.entry_used_time = current_time

            self.run_record()
        else:
            if current_time - self.entry_used_time >= 60:
                self.entry_used_time = current_time
                self.run_record()
            else:
                print("Rate Limit:", "Please wait for a minute before making another API call.")
                self.open_rate_limit_window()

    2 usages
    def run_record(self):
        selected_id = self.table.selection()[0]  # Get selected item's ID
        selected_item = self.table.item(selected_id)  # Get selected item's details
        selected_values = selected_item['values']
        selected_values.append(self.user_id)
        result_string = '$'.join(map(str, selected_values))
        with open('bot_configuration.txt', 'w') as file:
            file.write(result_string)
        self.controller.bot_time_frame.set(f'{selected_values[2]} {selected_values[1]}')
        self.controller.bot_type.set(selected_values[3])
        self.controller.bot_stocks.set(selected_values[4])
        self.controller.bot_strategies.set(selected_values[5])
        self.controller.enable_running_button()
        self.controller.switch_frame('PageTwo')
        self.task.start()

```

run.py

```python
import customtkinter
from matplotlib import dates
import pandas as pd
import queue
from matplotlib import font_manager
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg


2 usages
class PageTwo(customtkinter.CTkFrame):
    def __init__(self, master, task, data_queue=None, stdout_queue=None, controller=None):
        customtkinter.CTkFrame.__init__(self, master)

        self.controller = controller
        self.task = task
        self.data_queue = data_queue
        self.stdout_queue = stdout_queue
        self.grid_columnconfigure((0,1), weight=1)
        self.grid_rowconfigure((0,1), weight=1)
        self.configure(fg_color="#434343")

        # Frame 1
        self.frame1 = customtkinter.CTkFrame(self, fg_color="#666666", width=200, height=150)
        self.frame1.grid(row=0, column=0, padx=(20, 10), pady=(20, 0), sticky="nsew")


        self.container = customtkinter.CTkFrame(self.frame1, fg_color="#666666", width=200, height=100)
        self.container.place(relx=0.5, rely=0.45, anchor='center')
        self.container.grid_rowconfigure(0, weight=1)
        self.container.grid_rowconfigure(7, weight=1)

        self.config_label = customtkinter.CTkLabel(master=self.container,
                                                   text='Configurations',
                                                   font=customtkinter.CTkFont(size=24, weight="bold"))
        self.config_label.grid(row=1, column=0, columnspan=2, sticky="w", padx=(10, 0), pady=(0, 10))
        self.label1 = customtkinter.CTkLabel(master=self.container,
                                             text='Time Frame:',
                                             font=customtkinter.CTkFont(size=15, weight="bold"))
        self.label1.grid(row=2, column=0, sticky="w", padx=(10, 0), pady=(0, 5))
        self.label1_data = customtkinter.CTkLabel(master=self.container,
                                                  font=customtkinter.CTkFont(size=15, weight="bold"),
                                                  textvariable=self.controller.bot_time_frame,
                                                  text_color="#B7B7B7")
        self.label1_data.grid(row=2, column=1, sticky="w", padx=(10, 0), pady=(0, 5))
        self.label2 = customtkinter.CTkLabel(master=self.container,
                                             text='Type:',
                                             font=customtkinter.CTkFont(size=15, weight="bold"))
        self.label2.grid(row=3, column=0, sticky="w", padx=(10, 0), pady=(0, 5))
        self.label2_data = customtkinter.CTkLabel(master=self.container,
```

```
                                        font=customtkinter.CTkFont(size=15, weight="bold"),
                                        textvariable=self.controller.bot_type, text_color="#B7B7B7")
        self.label2_data.grid(row=3, column=1, sticky="w", padx=(10, 0), pady=(0, 5))
        self.label3 = customtkinter.CTkLabel(master=self.container,
                                             text='Stocks:',
                                             font=customtkinter.CTkFont(size=15, weight="bold"))
        self.label3.grid(row=4, column=0, sticky="w", padx=(10, 0), pady=(0, 5))
        self.label3_data = customtkinter.CTkLabel(master=self.container,
                                        font=customtkinter.CTkFont(size=15, weight="bold"),
                                        textvariable=self.controller.bot_stocks,
                                        wraplength=200,
                                        text_color="#B7B7B7")
        self.label3_data.grid(row=4, column=1, sticky="w", padx=(10, 0), pady=(0, 5))
        self.label4 = customtkinter.CTkLabel(master=self.container,
                                             text='Strategies:',
                                             font=customtkinter.CTkFont(size=15, weight="bold"))
        self.label4.grid(row=5, column=0, sticky="w", padx=(10, 0), pady=(0, 20))
        self.label4_data = customtkinter.CTkLabel(master=self.container,
                                        font=customtkinter.CTkFont(size=15, weight="bold"),
                                        textvariable=self.controller.bot_strategies,
                                        wraplength=200,
                                        text_color="#B7B7B7")
        self.label4_data.grid(row=5, column=1, padx=(10, 0), pady=(0, 20))
        self.stop_button = customtkinter.CTkButton(master=self.frame1,

                                         text="Stop",
                                         font=customtkinter.CTkFont(size=14, weight="bold"),
                                         fg_color="transparent",
                                         border_width=2,
                                         height=30,
                                         corner_radius=15,
                                         command=self.stop_clicked)
        self.stop_button.place(relx=0.25, rely=0.85, anchor='center')
        self.clear_button = customtkinter.CTkButton(master=self.frame1,
                                          text="Clear",
                                          font=customtkinter.CTkFont(size=14, weight="bold"),
                                          fg_color="transparent",
                                          border_width=2,
                                          height=30,
                                          corner_radius=15,
                                          command=self.clear_text)
        self.clear_button.place(relx=0.75, rely=0.85, anchor='center')

        # Frame 2
        self.frame2 = customtkinter.CTkFrame(self, fg_color="#999999", width=200, height=150)
        self.frame2.grid(row=1, column=0, padx=(20, 10), pady=(20, 20), sticky="nsew")

        self.title_font_prop = font_manager.FontProperties(weight='bold', size=14, family='Roboto')
        self.label_font_prop = font_manager.FontProperties(weight='bold', size=8, family='Roboto')
```

```
97
98          # Plot setup
99          self.fig, self.ax = plt.subplots(figsize=(3.5, 2.5))
100         self.ax.set_facecolor("#999999")
101         self.fig.set_facecolor("#999999")
102         self.ax.grid(False)
103         for spine in self.ax.spines.values():
104             spine.set_visible(False)
105
106         self.ax.axis('off')  # Hide the axes initially
107
108         self.canvas = FigureCanvasTkAgg(self.fig, self.frame2)
109         self.canvas.get_tk_widget().place(relx=0.5, rely=0.5, anchor='center')
110
111         # Frame 3
112         self.textbox1 = customtkinter.CTkTextbox(self, fg_color="#000000", width=200, height=200)
113         self.textbox1.grid(row=0, column=1, rowspan=2, padx=(10, 20), pady=(20, 20), sticky="nsew")
114
115         self.poll_data_queue()
116         self.poll_stdout_queue()
117
            2 usages
118     def poll_data_queue(self):
119         try:
120             x = self.data_queue.get_nowait()

121             print("Received dataframe of length:", x.shape[0])
122             self.update_plot(x)
123         except queue.Empty:
124             pass
125         # Check the queue again in 1 second
126         self.after(1000, self.poll_data_queue)
127
            2 usages
128     def poll_stdout_queue(self):
129         try:
130             message = self.stdout_queue.get_nowait()
131             self.textbox1.insert("end", message)  # Insert the message to the end of the textbox
132             self.textbox1.see("end")
133         except queue.Empty:
134             pass
135         self.after(1000, self.poll_stdout_queue)
136
            1 usage
137     def update_plot(self, df):
138         self.ax.clear()
139         self.ax.axis('on')  # Show the axes
140         line_styles = ['-', '--', ':', '-.']
141
142         # Ensure the 'timestamp' column is a datetime object
143         df.index = pd.to_datetime(df.index)
144
```

```
        # Plot each column with different line styles
        for index, column in enumerate(df.columns):
            style = line_styles[index % len(line_styles)]
            self.ax.plot(df.index, df[column], label=column, linestyle=style, color='white')

        # Set title and labels
        self.ax.set_title("Received Data", color='white', fontproperties=self.title_font_prop)
        self.ax.set_xlabel("Timestamp", color='white', fontproperties=self.label_font_prop)
        self.ax.set_ylabel("Value", color='white', fontproperties=self.label_font_prop)

        # Determine x-axis tick interval and formatting
        locator = dates.AutoDateLocator()

        # Depending on the range, determine the appropriate format
        time_range = df.index[-1] - df.index[0]
        if time_range <= pd.Timedelta(hours=1):  # less than 1 hour
            formatter = dates.DateFormatter('%H:%M')
        elif time_range <= pd.Timedelta(days=1):  # less than 1 day but more than 1 hour
            formatter = dates.DateFormatter('%H:%M')
        else:  # more than a day
            formatter = dates.DateFormatter('%m-%d')

        self.ax.xaxis.set_major_locator(locator)
        self.ax.xaxis.set_major_formatter(formatter)


        # Legend adjustments
        legend = self.ax.legend(loc='upper left', frameon=False)
        for text in legend.get_texts():
            text.set_color('white')
            text.set_fontproperties(self.label_font_prop)

        # Handle background and grid
        self.ax.set_facecolor("#999999")
        self.ax.grid(False)
        for spine in self.ax.spines.values():
            spine.set_visible(False)
        self.ax.tick_params(colors='white')

        # Apply font properties to tick labels
        for label in self.ax.get_xticklabels():
            label.set_fontproperties(self.label_font_prop)
        for label in self.ax.get_yticklabels():
            label.set_fontproperties(self.label_font_prop)

        # Ensure that labels and titles fit
        self.fig.tight_layout()
        self.canvas.draw()


    1 usage
    def clear_text(self):
        self.textbox1.delete('1.0', 'end')

    1 usage
    def stop_clicked(self):
        self.controller.switch_frame('PageOne')
        self.controller.disable_running_button()
        self.task.stop()
```

security.py

```
import os

KEY_FILE = 'keys.key'


1 usage
def generate_key(length):
    return ''.join(chr(os.urandom(1)[0]) for _ in range(length))


1 usage
def store_key(identifier, key):
    with open(KEY_FILE, 'a') as file:
        file.write(f"{identifier}:{key}\n")


1 usage
def retrieve_key(identifier):
    with open(KEY_FILE, 'r') as file:
        for line in file.readlines():
            if line.startswith(f"{identifier}:"):
                return line.strip().split(':')[1]
    raise ValueError(f"No key found for identifier {identifier}")


3 usages
def encrypt_data(string_to_encrypt, identifier):
    key = generate_key(len(string_to_encrypt))

    store_key(identifier, key)
    return ''.join(chr(ord(p) ^ ord(k)) for p, k in zip(string_to_encrypt, key))


9 usages
def decrypt_data(encrypted_string, identifier):
    key = retrieve_key(identifier)
    return ''.join(chr(ord(c) ^ ord(k)) for c, k in zip(encrypted_string, key))

```

task.py

```python
# IMPORT MODULES
import multiprocessing
# IMPORT USER-DEFINED PACKAGES
from historicalbot.hist_main import HistoricalBot
from livebot.live_main import LiveBot


2 usages
class Task:
    def __init__(self, data_queue, stdout_queue):
        self._running = multiprocessing.Value('b', False)
        self._thread = None
        self.data_queue = data_queue
        self.stdout_queue = stdout_queue
        self.bot = None

    1 usage
    def _run(self):
        while self._running.value:
            print("Task is running")
            with open('bot_configuration.txt', 'r') as f:
                content = f.read()
                content_list = content.split('$')
            if content_list[3] == 'Historical':
                self.bot = HistoricalBot(self.data_queue, self.stdout_queue)
                self.bot.start()

            elif content_list[3] == 'Live':
                self.bot = LiveBot(self.data_queue, self.stdout_queue)
                self.bot.start()
            else:
                self.stop()

    1 usage (1 dynamic)
    def start(self):
        if self._running.value:  # Check if the task is already running
            print("Task is already running")
        else:
            self._running.value = True
            self._thread = multiprocessing.Process(target=self._run)
            self._thread.start()

    4 usages (2 dynamic)
    def stop(self):
        if self.bot:
            self.bot.stop()
        self._running.value = False
        print("Task stopped")
        if self._thread is not None:
            self._thread.terminate()
            self._thread = None
```

Historical Bot

hist_config.py

```python
import os
import sqlite3
import security


10 usages
def config():  # Retrieve configuration details along with API keys
    current_directory = os.path.dirname(os.path.abspath(__file__))
    config_path = os.path.join(current_directory, '..', 'bot_configuration.txt')
    db_path = os.path.join(current_directory, '..', 'database.db')

    configurations = []
    with open(config_path, 'r') as f:
        content = f.read()
        content_list = content.split('$')
        configurations.append(content_list[1])
        configurations.append(content_list[2])
        stocks_list = content_list[4].split(', ')
        configurations.append(stocks_list)
        strats_list = content_list[5].split(', ')
        configurations.append(strats_list)
        configurations.append(content_list[6])

    time_unit_string = configurations[0]
    amount = int(configurations[1])

    stocks = configurations[2]
    strategies = configurations[3]
    user_id = int(configurations[4])

    conn = sqlite3.connect(db_path)
    c = conn.cursor()
    c.execute("SELECT api_key, api_secret_key, av_key FROM details WHERE user_id=?", (user_id,))
    record = c.fetchone()
    c.close()
    conn.close()

    API_KEY = security.decrypt_data(record[0], str(user_id) + 'A')
    API_SECRET_KEY = security.decrypt_data(record[1], str(user_id) + 'B')
    AVKEY = security.decrypt_data(record[2], str(user_id) + 'C')

    limit = 500

    def convert_to_base(unit, amount):  # Convert to minutes
        if unit == "Day":
            return amount * 24 * 60  # 24 hours in a day, 60 minutes in an hour
        elif unit == "Hour":
            return amount * 60  # 60 minutes in an hour
        elif unit == "Minute":
            return amount  # Already in minutes
```

```
        else:
            raise ValueError(f"Unsupported unit: {unit}")

    in_minutes = convert_to_base(time_unit_string, amount)

    def calculate_days_back(bars, frequency):
        trading_minutes_per_day = 6.5 * 60  # 6.5 hours per day * 60 minutes per hour
        total_minutes = bars * frequency  # Total minutes represented by the bars
        days = total_minutes / trading_minutes_per_day  # Days represented by the bars
        return int(days) + 1

    for_the_past_x_days = calculate_days_back(limit, in_minutes)

    config = [time_unit_string, amount, stocks, strategies, API_KEY, API_SECRET_KEY, AVKEY, in_minutes,
              for_the_past_x_days]
    return config

```

hist_datagrab.py

```
# IMPORTING API
from alpaca.data.historical import StockHistoricalDataClient
from alpaca.data.requests import StockBarsRequest
from alpaca.data.timeframe import TimeFrame, TimeFrameUnit
# IMPORTING MODULES
from datetime import datetime, timedelta
import requests
# IMPORTING USER-DEFINED MODULES
from historicalbot.hist_config import config


2 usages
def get_timeframe_unit(unit_string):  # Convert string to an actual value
    timeframe_unit = getattr(TimeFrameUnit, unit_string)
    return timeframe_unit


9 usages
def request_data(stock):  # Reqeust close price for a specific stock
    try:
        configurations = config()
        time_unit_string = configurations[0]
        amount = configurations[1]
        API_KEY = configurations[4]
        API_SECRET_KEY = configurations[5]
        for_the_past_x_days = configurations[8]
```

```
        client = StockHistoricalDataClient(API_KEY, API_SECRET_KEY)
        current_date = datetime.now()
        date_x_days_ago = current_date - timedelta(days=for_the_past_x_days)
        formatted_date = date_x_days_ago.strftime('%Y-%m-%d')
        request_params = StockBarsRequest(
            symbol_or_symbols=stock,
            timeframe=TimeFrame(amount, get_timeframe_unit(time_unit_string)),
            start=datetime.strptime(f"{formatted_date}", '%Y-%m-%d'),
        )
        bars = client.get_stock_bars(request_params).df
        data = bars["close"].to_frame()
        data = data.reset_index().pivot(index='timestamp', columns='symbol', values='close')
        data = data.ffill()  # NaN values get forward filled
        return data
    except requests.exceptions.ConnectionError as e:
        # Handle the connection error
        return None

2 usages
def request_all_data(stock):  # Request all data for a specific stock
    try:
        configurations = config()
        time_unit_string = configurations[0]
        amount = configurations[1]
        API_KEY = configurations[4]

        API_SECRET_KEY = configurations[5]
        for_the_past_x_days = configurations[8]
        # DECLARING GLOBAL VARIABLES
        client = StockHistoricalDataClient(API_KEY, API_SECRET_KEY)
        current_date = datetime.now()
        date_x_days_ago = current_date - timedelta(days=for_the_past_x_days)
        formatted_date = date_x_days_ago.strftime('%Y-%m-%d')
        request_params = StockBarsRequest(
            symbol_or_symbols=stock,
            timeframe=TimeFrame(amount, get_timeframe_unit(time_unit_string)),
            start=datetime.strptime(f"{formatted_date}", '%Y-%m-%d'),
        )
        df = client.get_stock_bars(request_params).df
        df_reset = df.reset_index()
        df_reset.set_index('timestamp', inplace=True)
        df_reset = df_reset.ffill()  # Forward fill assumed to best account for market behaviour for NaN values
        return df_reset
    except requests.exceptions.ConnectionError as e:
        # Handle the connection error
        return None

2 usages
def calculate_current_price(stock):  # Return the current price of a stock
    df = request_data(stock).copy()
    value = df.iloc[-1:].values[0]
    return value

```

hist_main.py

```python
# IMPORTING API
from alpaca.trading.client import TradingClient
from alpaca.trading.requests import MarketOrderRequest
from alpaca.trading.enums import OrderSide, TimeInForce
# IMPORTING OTHER MODULES
import time
# IMPORTING USER-DEFINED MODULES
from historicalbot.hist_market import check_if_open, sleep
from historicalbot.hist_strategy import Momentum
from historicalbot.hist_strategy2 import SMACrossover
from historicalbot.hist_strategy3 import Breakout
from historicalbot.hist_strategy4 import SVRStrategy
from historicalbot.hist_risk import get_correlation_data, calculate_position_size, get_sentiment_data
from historicalbot.hist_datagrab import calculate_current_price
from historicalbot.hist_config import config


2 usages
class HistoricalBot:
    def __init__(self, data_queue, stdout_queue):
        self.configurations = config()
        self.in_minutes = self.configurations[7]
        self.data_queue = data_queue
        self.stdout_queue = stdout_queue
        self.API_KEY = self.configurations[4]

        self.API_SECRET_KEY = self.configurations[5]
        self.stocks = self.configurations[2]
        self.strategies = [globals()[name] for name in self.configurations[3]]

        self.trading_client = TradingClient(self.API_KEY, self.API_SECRET_KEY, paper=True)
        self.risk_per_trade = 0.01
        self.min_risk = 0.005
        self.max_risk = 0.02

        self.correlation_list = get_correlation_data()
        self.sentiment_data = get_sentiment_data()
        self.is_running = False

    2 usages (1 dynamic)
    def start(self):
        self.is_running = True
        while self.is_running:
            if check_if_open():
                start_time = time.time()
                for stock in self.stocks:

                    account_balance = float(self.trading_client.get_account().cash)
                    self.stdout_queue.put(f"Account balance: {int(account_balance)}. ")

                    try:
```

```
                position = self.trading_client.get_open_position(stock)
                current_holding = position.qty
            except:
                current_holding = 0

            current_price = calculate_current_price(stock)

            signals = []
            volatility_for_stock = []
            correlation_for_stock = self.correlation_list[self.stocks.index(stock)]
            sentiment_for_stock = self.sentiment_data[self.stocks.index(stock)]

            for Strategy in self.strategies:
                strategy_instance = Strategy(stock)
                validation = strategy_instance.validation()
                if validation:
                    pass
                else:
                    self.connection_error()
                    break
                generated_signal = strategy_instance.generate_signals()
                data = strategy_instance.extract_data()
                self.data_queue.put(data)
                strategy_instance.backtest()

                if not strategy_instance.test_results()[1] > 0:
                    self.stdout_queue.put(
                        f'Skipping {Strategy.__name__} for {stock} due to negative performance. '
                    )
                    continue
                signals.append(generated_signal)
                _, annual_std = strategy_instance.calculate_statistics(self.in_minutes)
                volatility_for_stock.append(annual_std)
                results = strategy_instance.extract_results()
                self.data_queue.put(results)

            if not signals:
                self.stdout_queue.put(f"All strategies skipped for stock {stock}. Moving to next stock. ")
                continue

            average_volatility = sum(volatility_for_stock) / len(volatility_for_stock)

            if all(signal == 1 for signal in signals) and int(current_holding) == 0:
                num_shares = calculate_position_size(account_balance, current_price, average_volatility,
                                                     correlation_for_stock, sentiment_for_stock)
                market_order_data = MarketOrderRequest(
                    symbol=stock,
                    qty=num_shares,
                    side=OrderSide.BUY,
```

```
                    type='market',
                    time_in_force=TimeInForce.GTC,
                )
                market_order = self.trading_client.submit_order(
                    order_data=market_order_data
                )
                self.stdout_queue.put(f'Placed a buy order for {num_shares} shares of {stock}. ')

            elif all(signal == -1 for signal in signals) and int(current_holding) > 0:
                market_order_data = MarketOrderRequest(
                    symbol=stock,
                    qty=current_holding,
                    side=OrderSide.SELL,
                    type='market',
                    time_in_force=TimeInForce.GTC,
                )
                market_order = self.trading_client.submit_order(
                    order_data=market_order_data
                )
                self.stdout_queue.put(f'Placed a sell order for {current_holding} shares of {stock}. ')

            else:
                self.stdout_queue.put("No action. ")
                pass
```

```
            end_time = time.time()
            execution_time = end_time - start_time
            sleep_duration = max(60*self.in_minutes - execution_time, 0)  # Necessary due to the machine learning
            # strategies
            time.sleep(sleep_duration)
        else:
            self.stdout_queue.put("Sleeping till market opens. ")
            sleep()

1 usage
def connection_error(self):
    self.stdout_queue.put("Connection error. Please restart the program. ")
    self.is_running = False

2 usages (2 dynamic)
def stop(self):
    self.is_running = False
```

hist_market.py

```python
# IMPORTING API
from alpaca.trading.client import TradingClient
# IMPORTING MODULES
import datetime
import pytz
import time
# IMPORTING USER-DEFINED MODULES
from historicalbot.hist_config import config

# DECLARING GLOBAL VARIABLES
market_open_time = datetime.time(9, 30)
market_close_time = datetime.time(16, 0)
market_timezone = pytz.timezone('America/New_York')


2 usages
def sleep():  # Sleep till market opens
    configurations = config()
    API_KEY = configurations[4]
    API_SECRET_KEY = configurations[5]
    trading_client = TradingClient(API_KEY, API_SECRET_KEY, paper=True)
    now = datetime.datetime.now(market_timezone)
    clock = trading_client.get_clock()
    next_open = clock.next_open.astimezone(market_timezone)
    time_until_open = next_open - now

    sleep_duration = time_until_open.total_seconds() + 5
    print(f"Sleeping for {time_until_open} until market opens...")
    time.sleep(sleep_duration)


2 usages
def check_if_open():  # Check if market is currently open
    current_datetime = datetime.datetime.now(market_timezone)
    current_day = current_datetime.weekday()  # Monday is 0 and Sunday is 6

    # Check if it's a weekend
    if current_day > 4:  # Saturday or Sunday
        return False

    # Check if the current time is outside of market hours
    start_datetime = current_datetime.replace(hour=market_open_time.hour, minute=market_open_time.minute, second=0,
                                              microsecond=0)
    end_datetime = current_datetime.replace(hour=market_close_time.hour, minute=market_close_time.minute, second=0,
                                            microsecond=0)

    if start_datetime <= current_datetime <= end_datetime:
        return True
    return False

```

hist_risk.py

```python
# IMPORTING MODULES
import requests
from datetime import datetime
import math
# IMPORTING USER-DEFINED MODULES
from historicalbot.hist_datagrab import request_data
from historicalbot.hist_config import config


2 usages
def get_correlation_data():  # How correlated is the stock to the other stocks in the portfolio?
    configurations = config()
    stocks = configurations[2]
    correlation_data = []
    df = request_data(stocks)
    for stock in stocks:
        correlation_matrix = df.corr()
        stock_correlations = correlation_matrix[stock]
        average_correlation = stock_correlations.mean()
        correlation_data.append(average_correlation)
    return correlation_data


2 usages
def get_sentiment_data():
    configurations = config()

    stocks = configurations[2]
    AVKEY = configurations[6]

    try:
        sentiment_data = []

        for stock in stocks:
            url = f'https://www.alphavantage.co/query?function=NEWS_SENTIMENT&tickers={stock}&apikey={AVKEY}'
            r = requests.get(url)
            data_latest = r.json()

            def calculate_weighted_sentiment(data, ticker):
                sentiment_sum = 0.0
                relevance_sum = 0.0
                today = datetime.now()

                for item in data["feed"]:
                    for ticker_sentiment in item["ticker_sentiment"]:
                        if ticker_sentiment["ticker"] == ticker:
                            sentiment = float(ticker_sentiment["ticker_sentiment_score"])
                            relevance = float(ticker_sentiment["relevance_score"])

                            # Calculate time decay factor
                            time_published = datetime.strptime(item["time_published"], "%Y%m%dT%H%M%S")
```

```
                    days_since_published = (today - time_published).days
                    decay_factor = pow(0.5, days_since_published / 7.0)  # Half-life of 7 days

                    sentiment_sum += sentiment * relevance * decay_factor
                    relevance_sum += relevance * decay_factor

            if relevance_sum == 0.0:
                return None

            return sentiment_sum / relevance_sum

        val = calculate_weighted_sentiment(data_latest, stock)
        sentiment_data.append(val)

    return sentiment_data
except Exception as e:
    print(f"Error occurred: {e}")  # If request fails, return neutral sentiment
    return [0] * len(stocks)


1 usage
class TreeNode:
    def __init__(self, weight, value, key):
        self.weight = weight
        self.value = value

        self.key = key
        self.height = 1
        self.left = None
        self.right = None


1 usage
class AVLTree:
    def __init__(self):
        self.root = None

    4 usages
    def _height(self, node):
        if not node:
            return 0
        return node.height

    5 usages
    def _update_height(self, node):
        if node:
            node.height = 1 + max(self._height(node.left), self._height(node.right))

    1 usage
    def _balance_factor(self, node):
        if not node:
            return 0
        return self._height(node.left) - self._height(node.right)

```

```
    3 usages
97  def _left_rotate(self, z):
98      y = z.right
99      T2 = y.left
100     y.left = z
101     z.right = T2
102     self._update_height(z)
103     self._update_height(y)
104     return y
105
    3 usages
106 def _right_rotate(self, z):
107     y = z.left
108     T3 = y.right
109     y.right = z
110     z.left = T3
111     self._update_height(z)
112     self._update_height(y)
113     return y
114
    7 usages
115 def insert(self, root, key, weight, value):
116     if not root:
117         return TreeNode(weight, value, key)
118
119     if key < root.key:
120         root.left = self.insert(root.left, key, weight, value)

121     else:
122         root.right = self.insert(root.right, key, weight, value)
123
124     self._update_height(root)
125     balance = self._balance_factor(root)
126
127     # Left heavy
128     if balance > 1:
129         if key < root.left.key:
130             return self._right_rotate(root)
131         else:
132             root.left = self._left_rotate(root.left)
133             return self._right_rotate(root)
134
135     # Right heavy
136     if balance < -1:
137         if key > root.right.key:
138             return self._left_rotate(root)
139         else:
140             root.right = self._right_rotate(root.right)
141             return self._left_rotate(root)
142
143     return root
144
```

```
    3 usages
    def compute_score(self, root):
        if not root:
            return 0
        left_score = self.compute_score(root.left)
        right_score = self.compute_score(root.right)
        return left_score + root.weight * root.value + right_score


2 usages
def calculate_position_size(balance, price, volatility, correlation, sentiment):
    # Default values if volatility or sentiment is None
    if volatility is None:
        volatility = 0.5  # Middle ground
    if sentiment is None:
        sentiment = 0  # Neutral sentiment

    # Dynamic Weights
    total_weight = 1
    w_volatility = 0.2 + 0.1 * volatility
    w_correlation = 0.2 - 0.1 * volatility
    w_sentiment = 0.2
    w_balance = 0.2
    w_price = total_weight - (w_volatility + w_correlation + w_sentiment + w_balance)

    # Normalization

    normalized_sentiment = (math.tanh(sentiment) + 1) / 2  # Using tanh for non-linear mapping
    normalized_volatility = volatility
    normalized_correlation = 1 - correlation
    normalized_balance = balance / max(balance, 1)
    normalized_price = price / max(price, 1)

    # Create AVL Tree
    tree = AVLTree()
    tree.root = tree.insert(tree.root, 1, w_volatility, normalized_volatility)
    tree.root = tree.insert(tree.root, 2, w_correlation, normalized_correlation)
    tree.root = tree.insert(tree.root, 3, w_sentiment, normalized_sentiment)
    tree.root = tree.insert(tree.root, 4, w_balance, normalized_balance)
    tree.root = tree.insert(tree.root, 5, w_price, normalized_price)

    # Compute combined score using AVL Tree
    score = tree.compute_score(tree.root)

    # Risk Management
    risk_factor = 0.5 + 0.5 * volatility
    position_size = score * balance * (1 - risk_factor) / price

    # Scaling down
    SCALING_FACTOR = 20  # Adjust based on user preference
    position_size = position_size / SCALING_FACTOR


    # Adjust for Maximum Allocation
    MAX_ALLOCATION = 0.001
    position_size = min(position_size, balance * MAX_ALLOCATION)
    position_size = max(1, position_size)

    return int(position_size)

```

hist_strategy.py

```python
# IMPORTING LIBRARY MODULES
import numpy as np
import pandas as pd
# IMPORTING USER-DEFINED MODULES
from historicalbot.hist_datagrab import request_data


1 usage
class Momentum:
    def __init__(self, symbol, lookback_period=10):
        # Initialize the Momentum class with a stock symbol and an optional lookback period for calculations
        self.symbol = symbol
        self.data = request_data(symbol)  # Fetches the stock data
        self.lookback_period = lookback_period
        self.results = None  # A placeholder to store results

    1 usage (1 dynamic)
    def validation(self):
        if self.data is None:
            return False
        else:
            return True

    2 usages (2 dynamic)
    def generate_signals(self):
        # Calculate momentum as the difference between the current price and the price 'lookback_period' days ago
        self.data['momentum'] = self.data[self.symbol] - self.data[self.symbol].shift(self.lookback_period)

        # Generate buy signals where momentum is positive, represented as 1
        self.data['buy_signal'] = np.where((self.data['momentum'] > 0), 1, 0)
        # Generate sell signals where momentum is negative, represented as -1
        self.data['sell_signal'] = np.where((self.data['momentum'] < 0), -1, 0)
        # Combine buy and sell signals to get an overall signal (1 for buy, -1 for sell, 0 otherwise)
        self.data['signal'] = self.data['buy_signal'] + self.data['sell_signal']
        self.data.dropna(inplace=True)  # Remove rows with missing data
        return self.data['signal'].iat[-1]  # Return the most recent signal

    2 usages (2 dynamic)
    def extract_data(self):
        # Extract relevant data for analysis: close price and momentum
        y_close_price = self.data[self.symbol]
        y_momentum = self.data['momentum']
        extracted_data = pd.DataFrame({
            'Close_Price': y_close_price,
            'Momentum': y_momentum
        })
        return extracted_data

    2 usages (2 dynamic)
    def backtest(self):
        # Calculate logarithmic returns based on the stock's closing prices
        self.data["returns"] = np.log(self.data[self.symbol].div(self.data[self.symbol].shift(1)))
        self.data.dropna(inplace=True)  # Remove rows with missing data
        # Create a 'position' column that holds the difference between buy and sell signals
```

```
        self.data['position'] = self.data['buy_signal'] - self.data['sell_signal']

    2 usages (2 dynamic)
    def test_results(self):
        # Calculate the strategy's returns by multiplying the returns with the previous day's position
        self.data["strategy"] = self.data["returns"] * self.data["position"].shift(1)
        self.data.dropna(inplace=True)  # Remove rows with missing data
        # Calculate cumulative returns for both the strategy and a buy-and-hold approach
        self.data["returnsbh"] = self.data["returns"].cumsum().apply(np.exp)
        self.data["returnsstrategy"] = self.data["strategy"].cumsum().apply(np.exp)
        perf = self.data["returnsstrategy"].iloc[-1]  # Final performance of the strategy
        outperf = perf - self.data["returnsbh"].iloc[-1]  # Outperformance over buy-and-hold
        self.results = self.data  # Store results in the instance variable
        return round(perf, 6), round(outperf, 6)  # Return rounded performance and outperformance

    2 usages (2 dynamic)
    def calculate_statistics(self, minute_interval):
        if self.data is None or 'strategy' not in self.data.columns:
            print("Strategy results not found. Please run the backtest first.")
            return

        if minute_interval < 60:
            timeframe = 'Minute'
            multiplier = minute_interval
            scaling_unit = 60  # 60 minutes in an hour
        elif 60 <= minute_interval < 1440:

            timeframe = 'Hour'
            multiplier = minute_interval / 60
            scaling_unit = 24  # 24 hours in a day
        else:
            timeframe = 'Day'
            multiplier = minute_interval / 1440
            scaling_unit = 252  # Approximately 252 trading days in a year

        scaling_factor = np.sqrt(scaling_unit / multiplier)

        # Calculate the average return for the specific timeframe
        ret = np.exp(self.data["strategy"].mean() * scaling_factor)

        # Calculate the volatility, scaled to the next higher unit of time
        std = self.data["strategy"].std() * scaling_factor

        return ret, std

    2 usages (2 dynamic)
    def extract_results(self):
        if self.results is None:
            print("No results to extract. Run the test first.")
            return None
        y_returnsbh = self.results['returnsbh']
        y_returnsstrategy = self.results['returnsstrategy']

        extracted_data = pd.DataFrame({
            'ReturnsBH': y_returnsbh,
            'ReturnsStrategy': y_returnsstrategy
        })
        return extracted_data
```

hist_strategy2.py

```python
# IMPORTING LIBRARY MODULES
import numpy as np
import pandas as pd
# IMPORTING USER-DEFINED MODULES
from historicalbot.hist_datagrab import request_data


1 usage
class SMACrossover:
    def __init__(self, symbol, short_term_sma_length=15, long_term_sma_length=60):
        self.symbol = symbol
        self.data = request_data(symbol)
        self.short_term_sma_length = short_term_sma_length
        self.long_term_sma_length = long_term_sma_length
        self.results = None

    1 usage (1 dynamic)
    def validation(self):
        if self.data is None:
            return False
        else:
            return True

    2 usages
    def calculate_sma(self, window):
        return self.data.rolling(window=window).mean()


    2 usages (2 dynamic)
    def generate_signals(self):
        short_term_sma = self.calculate_sma(self.short_term_sma_length)
        long_term_sma = self.calculate_sma(self.long_term_sma_length)
        self.data['short_sma'] = short_term_sma
        self.data['long_sma'] = long_term_sma
        self.data['buy_signal'] = np.where(
            (short_term_sma < long_term_sma) & (short_term_sma.shift(1) > long_term_sma.shift(1)), 1, 0)
        self.data['sell_signal'] = np.where(
            (short_term_sma > long_term_sma) & (short_term_sma.shift(1) < long_term_sma.shift(1)), -1, 0)
        self.data['signal'] = self.data['buy_signal'] + self.data['sell_signal']
        self.data.dropna(inplace=True)
        return self.data['signal'].iat[-1]

    2 usages (2 dynamic)
    def extract_data(self):
        y_close_price = self.data[self.symbol]
        y_short_sma = self.data['short_sma']
        y_long_sma = self.data['long_sma']
        extracted_data = pd.DataFrame({
            'Close_Price': y_close_price,
            f'{self.short_term_sma_length}-Day SMA': y_short_sma,
            f'{self.long_term_sma_length}-Day SMA': y_long_sma
        })
        return extracted_data

```

```
    2 usages (2 dynamic)
49  def backtest(self):
50      self.data["returns"] = np.log(self.data[self.symbol].div(self.data[self.symbol].shift(1)))
51      self.data.dropna(inplace=True)
52      self.data['position'] = self.data['buy_signal'] - self.data['sell_signal']
53
    2 usages (2 dynamic)
54  def test_results(self):
55      self.data["strategy"] = self.data["returns"] * self.data["position"].shift(1)
56      self.data.dropna(inplace=True)
57      self.data["returnsbh"] = self.data["returns"].cumsum().apply(np.exp)
58      self.data["returnsstrategy"] = self.data["strategy"].cumsum().apply(np.exp)
59      perf = self.data["returnsstrategy"].iloc[-1]
60      outperf = perf - self.data["returnsbh"].iloc[-1]
61      self.results = self.data
62      return round(perf, 6), round(outperf, 6)
63
    2 usages (2 dynamic)
64  def calculate_statistics(self, minute_interval):
65      if self.data is None or 'strategy' not in self.data.columns:
66          print("Strategy results not found. Please run the backtest first.")
67          return
68
69      if minute_interval < 60:
70          timeframe = 'Minute'
71          multiplier = minute_interval
72          scaling_unit = 60  # 60 minutes in an hour

73      elif 60 <= minute_interval < 1440:
74          timeframe = 'Hour'
75          multiplier = minute_interval / 60
76          scaling_unit = 24  # 24 hours in a day
77      else:
78          timeframe = 'Day'
79          multiplier = minute_interval / 1440
80          scaling_unit = 252  # Approximately 252 trading days in a year
81
82      scaling_factor = np.sqrt(scaling_unit / multiplier)
83
84      # Calculate the average return for the specific timeframe
85      ret = np.exp(self.data["strategy"].mean() * scaling_factor)
86
87      # Calculate the volatility, scaled to the next higher unit of time
88      std = self.data["strategy"].std() * scaling_factor
89
90      return ret, std
91
    2 usages (2 dynamic)
92  def extract_results(self):
93      if self.results is None:
94          print("No results to extract. Run the test first.")
95          return None
96      y_returnsbh = self.results['returnsbh']

97      y_returnsstrategy = self.results['returnsstrategy']
98      extracted_data = pd.DataFrame({
99          'ReturnsBH': y_returnsbh,
100         'ReturnsStrategy': y_returnsstrategy
101     })
102     return extracted_data
```

hist_strategy3.py

```python
# IMPORTING LIBRARY MODULES
import numpy as np
import pandas as pd
# IMPORTING USER-DEFINED MODULES
from historicalbot.hist_datagrab import request_data


1 usage
class Breakout:
    def __init__(self, symbol, lookback_period=20):
        self.symbol = symbol
        self.data = request_data(symbol)
        self.lookback_period = lookback_period
        self.results = None

    1 usage (1 dynamic)
    def validation(self):
        if self.data is None:
            return False
        else:
            return True

    2 usages (2 dynamic)
    def generate_signals(self):
        self.data['high'] = self.data[self.symbol].rolling(self.lookback_period).max()
        self.data['low'] = self.data[self.symbol].rolling(self.lookback_period).min()
        self.data['buy_signal'] = np.where((self.data[self.symbol] > self.data['high'].shift(1)), 1, 0)

        self.data['sell_signal'] = np.where((self.data[self.symbol] < self.data['low'].shift(1)), -1, 0)
        self.data['signal'] = self.data['buy_signal'] + self.data['sell_signal']
        self.data.dropna(inplace=True)
        return self.data['signal'].iat[-1]

    2 usages (2 dynamic)
    def extract_data(self):
        y_close_price = self.data[self.symbol]
        y_high = self.data['high']
        y_low = self.data['low']
        extracted_data = pd.DataFrame({
            'Close Price': y_close_price,
            'High': y_high,
            'Low': y_low
        })
        return extracted_data

    2 usages (2 dynamic)
    def backtest(self):
        self.data["returns"] = np.log(self.data[self.symbol].div(self.data[self.symbol].shift(1)))
        self.data.dropna(inplace=True)
        self.data['position'] = self.data['buy_signal'] - self.data['sell_signal']

    2 usages (2 dynamic)
    def test_results(self):
        self.data["strategy"] = self.data["returns"] * self.data["position"].shift(1)
        self.data.dropna(inplace=True)
```

```
        self.data["returnsbh"] = self.data["returns"].cumsum().apply(np.exp)
        self.data["returnsstrategy"] = self.data["strategy"].cumsum().apply(np.exp)
        perf = self.data["returnsstrategy"].iloc[-1]
        outperf = perf - self.data["returnsbh"].iloc[-1]
        self.results = self.data
        return round(perf, 6), round(outperf, 6)

    2 usages (2 dynamic)
    def calculate_statistics(self, minute_interval):
        if self.data is None or 'strategy' not in self.data.columns:
            print("Strategy results not found. Please run the backtest first.")
            return

        if minute_interval < 60:
            timeframe = 'Minute'
            multiplier = minute_interval
            scaling_unit = 60  # 60 minutes in an hour
        elif 60 <= minute_interval < 1440:
            timeframe = 'Hour'
            multiplier = minute_interval / 60
            scaling_unit = 24  # 24 hours in a day
        else:
            timeframe = 'Day'
            multiplier = minute_interval / 1440
            scaling_unit = 252  # Approximately 252 trading days in a year


        scaling_factor = np.sqrt(scaling_unit / multiplier)

        # Calculate the average return for the specific timeframe
        ret = np.exp(self.data["strategy"].mean() * scaling_factor)

        # Calculate the volatility, scaled to the next higher unit of time
        std = self.data["strategy"].std() * scaling_factor

        return ret, std

    2 usages (2 dynamic)
    def extract_results(self):
        if self.results is None:
            print("No results to extract. Run the test first.")
            return None
        y_returnsbh = self.results['returnsbh']
        y_returnsstrategy = self.results['returnsstrategy']
        extracted_data = pd.DataFrame({
            'ReturnsBH': y_returnsbh,
            'ReturnsStrategy': y_returnsstrategy
        })
        return extracted_data
```

hist_strategy4.py

```python
# IMPORTING LIBRARY MODULES
import numpy as np
import pandas as pd
import pandas_ta as ta
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVR
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
# IMPORTING USER-DEFINED MODULES
from historicalbot.hist_datagrab import request_all_data


1 usage
class SVRStrategy:
    def __init__(self, symbol, indicators=None):
        self.data = request_all_data(symbol)
        self.indicators = indicators if indicators is not None else {'RSI': 15, 'EMAF': 20, 'EMAM': 100, 'EMAS': 150}
        self.results = None

    1 usage (1 dynamic)
    def validation(self):
        if self.data is None:
            return False
        else:
            return True


    # Training + signal generation
    2 usages (2 dynamic)
    def generate_signals(self):
        # INDICATORS
        for indicator, length in self.indicators.items():
            if indicator == 'RSI':
                self.data['RSI'] = ta.rsi(self.data.close, length=length)
            elif indicator == 'EMAF':
                self.data['EMAF'] = ta.ema(self.data.close, length=length)
            elif indicator == 'EMAM':
                self.data['EMAM'] = ta.ema(self.data.close, length=length)
            elif indicator == 'EMAS':
                self.data['EMAS'] = ta.ema(self.data.close, length=length)

        self.data['targetnextclose'] = self.data['close'].shift(-1)
        self.data.drop(['volume', 'trade_count', 'vwap', 'symbol'], axis=1, inplace=True)
        self.data.dropna(inplace=True)

        # SPLITTING DATASET
        X = self.data.drop('targetnextclose', axis=1).values
        y = self.data['targetnextclose'].values
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

        # SCALING & MODEL TRAINING
        model = make_pipeline(MinMaxScaler(), SVR(C=1.0, epsilon=0.2))
```

```
        model.fit(X_train, y_train)

        y_pred = model.predict(X_test)

        df = pd.DataFrame({
            'Close': y_test,
            'Pred Close': y_pred
        })

        self.results = df
        self.results['buy_signal'] = np.where(df['Pred Close'].shift(-1) > df['Close'], 1, 0)
        self.results['sell_signal'] = np.where(df['Pred Close'].shift(-1) < df['Close'], -1, 0)
        self.results['signal'] = self.results['buy_signal'] + self.results['sell_signal']
        self.results.dropna(inplace=True)
        return self.results['signal'].iat[-1]

    2 usages (2 dynamic)
    def extract_data(self):
        if self.results is None:
            print("No results to extract. Run the test first.")
            return None
        y_actual_close = self.results["Close"]
        y_predicted_close = self.results["Pred Close"]
        extracted_data = pd.DataFrame({
            'Actual_Close': y_actual_close,

            'Predicted_Close': y_predicted_close
        })
        return extracted_data

    2 usages (2 dynamic)
    def backtest(self):
        self.results["returns"] = np.log(self.results['Close'].div(self.results['Close'].shift(1)))
        self.results.dropna(inplace=True)
        self.results['position'] = self.results['buy_signal'] - self.results['sell_signal']

    2 usages (2 dynamic)
    def test_results(self):
        self.results["strategy"] = self.results["returns"] * self.results["position"].shift(1)
        self.results.dropna(inplace=True)
        self.results["returnsbh"] = self.results["returns"].cumsum().apply(np.exp)
        self.results["returnsstrategy"] = self.results["strategy"].cumsum().apply(np.exp)
        perf = self.results["returnsstrategy"].iloc[-1]
        outperf = perf - self.results["returnsbh"].iloc[-1]
        return round(perf, 6), round(outperf, 6)

    2 usages (2 dynamic)
    def calculate_statistics(self, minute_interval):
        if self.data is None or 'strategy' not in self.data.columns:
            print("Strategy results not found. Please run the backtest first.")
            return

        if minute_interval < 60:
```

```
            timeframe = 'Minute'
            multiplier = minute_interval
            scaling_unit = 60  # 60 minutes in an hour
        elif 60 <= minute_interval < 1440:
            timeframe = 'Hour'
            multiplier = minute_interval / 60
            scaling_unit = 24  # 24 hours in a day
        else:
            timeframe = 'Day'
            multiplier = minute_interval / 1440
            scaling_unit = 252  # Approximately 252 trading days in a year

        scaling_factor = np.sqrt(scaling_unit / multiplier)

        rmse = np.sqrt(np.mean(((self.results["Pred Close"] - self.results["Close"]) ** 2)))

        std = self.results["strategy"].std() * scaling_factor  # Standard deviation scaled to the appropriate timeframe

        return rmse, std

    2 usages (2 dynamic)
    def extract_results(self):
        if self.results is None:
            print("No results to extract. Run the test first.")
            return None

        y_returnsbh = self.results['returnsbh']
        y_returnsstrategy = self.results['returnsstrategy']
        extracted_data = pd.DataFrame({
            'ReturnsBH': y_returnsbh,
            'ReturnsStrategy': y_returnsstrategy
        })
        return extracted_data
```

Live Bot

live_config.py

```python
import os
import sqlite3
import security


8 usages
def config():
    current_directory = os.path.dirname(os.path.abspath(__file__))
    config_path = os.path.join(current_directory, '..', 'bot_configuration.txt')
    db_path = os.path.join(current_directory, '..', 'database.db')
    configurations = []
    with open(config_path, 'r') as f:
        content = f.read()
        content_list = content.split('$')
        configurations.append(content_list[1])
        configurations.append(content_list[2])
        stocks_list = content_list[4].split(', ')
        configurations.append(stocks_list)
        strats_list = content_list[5].split(', ')
        configurations.append(strats_list)
        configurations.append(content_list[6])
    time_unit_string = configurations[0]
    amount = int(configurations[1])
    stocks = configurations[2]
    strategies = configurations[3]

    user_id = int(configurations[4])
    conn = sqlite3.connect(db_path)
    c = conn.cursor()
    c.execute("SELECT api_key, api_secret_key, av_key FROM details WHERE user_id=?", (user_id,))
    record = c.fetchone()
    c.close()
    conn.close()

    API_KEY = security.decrypt_data(record[0], str(user_id) + 'A')
    API_SECRET_KEY = security.decrypt_data(record[1], str(user_id) + 'B')

    limit = 500

    def convert_to_base(unit, amount):
        if unit == "Day":
            return amount * 24 * 60
        elif unit == "Hour":
            return amount * 60
        elif unit == "Minute":
            return amount
        else:
            raise ValueError(f"Unsupported unit: {unit}")

    in_minutes = convert_to_base(time_unit_string, amount)


    def calculate_days_back(bars, frequency):
        trading_minutes_per_day = 6.5 * 60
        total_minutes = bars * frequency
        days = total_minutes / trading_minutes_per_day
        return int(days) + 1

    for_the_past_x_days = calculate_days_back(limit, in_minutes)
    config = [time_unit_string, amount, stocks, strategies, API_KEY, API_SECRET_KEY, in_minutes, for_the_past_x_days]
    return config
```

live_datagrab.py

```
# IMPORTING API
from alpaca.data.historical import StockHistoricalDataClient
from alpaca.data.requests import StockBarsRequest
from alpaca.data.timeframe import TimeFrame, TimeFrameUnit
# IMPORTING OTHER MODULES
from datetime import datetime, timedelta
# IMPORTING USER-DEFINED MODULES
from livebot.live_config import config


1 usage
def get_timeframe_unit(unit_string):
    timeframe_unit = getattr(TimeFrameUnit, unit_string)
    return timeframe_unit


4 usages
def request_data(stock):
    configurations = config()
    time_unit_string = configurations[0]
    amount = configurations[1]
    API_KEY = configurations[4]
    API_SECRET_KEY = configurations[5]
    for_the_past_x_days = configurations[7]
    client = StockHistoricalDataClient(API_KEY, API_SECRET_KEY)
    current_date = datetime.now()

    date_x_days_ago = current_date - timedelta(days=for_the_past_x_days)
    formatted_date = date_x_days_ago.strftime('%Y-%m-%d')
    request_params = StockBarsRequest(
        symbol_or_symbols=stock,
        timeframe=TimeFrame(amount, get_timeframe_unit(time_unit_string)),
        start=datetime.strptime(f"{formatted_date}", '%Y-%m-%d'),
    )
    bars = client.get_stock_bars(request_params).df
    data = bars["close"].to_frame()
    data = data.reset_index().pivot(index='timestamp', columns='symbol', values='close')
    data = data.ffill()
    return data

```

live_main.py

```python
# IMPORTING API
from alpaca.trading.client import TradingClient
from alpaca.trading.requests import MarketOrderRequest
from alpaca.trading.enums import OrderSide, TimeInForce
# IMPORTING OTHER MODULES
import queue
import threading
import json
import websocket
import time
import traceback
# IMPORTING USER-DEFINED MODULES
from livebot.live_market import check_if_open, sleep
from livebot.live_strategy import Momentum
from livebot.live_strategy2 import Scalping
from livebot.live_strategy3 import MeanReversion
from livebot.live_datagrab import request_data
from livebot.live_risk import get_correlation_data, calculate_position_size
from livebot.live_config import config


2 usages
class LiveBot:
    def __init__(self, data_queue, stdout_queue):
        self.configurations = config()

        self.data_queue = data_queue
        self.stdout_queue = stdout_queue
        self.in_minutes = self.configurations[7]
        self.API_KEY = self.configurations[4]
        self.API_SECRET_KEY = self.configurations[5]
        self.stocks = self.configurations[2]
        self.strategies = [globals()[name] for name in self.configurations[3]]
        self.trading_client = TradingClient(self.API_KEY, self.API_SECRET_KEY, paper=True)
        self.risk_per_trade = 0.01
        self.min_risk = 0.005
        self.max_risk = 0.02
        self.reconnection_attempts = 0
        self.MAX_RECONNECTION_ATTEMPTS = 5
        self.correlation_list = get_correlation_data()
        self.data = request_data(self.stocks)
        self.message_count = 0
        self.x = self.configurations[6]
        self.is_running = False
        self.start_event = threading.Event()
        self.message_queue = queue.Queue()

    1 usage
    def action(self, stock, current_price):
        account_balance = float(self.trading_client.get_account().cash)
        self.stdout_queue.put(f"Account balance: {int(account_balance)}. ")
```

```
49
50          try:
51              position = self.trading_client.get_open_position(stock)
52              current_holding = position.qty
53          except:
54              current_holding = 0
55
56          signals = []
57          volatility_for_stock = []
58          correlation_for_stock = self.correlation_list[self.stocks.index(stock)]
59
60          for Strategy in self.strategies:
61              strategy_instance = Strategy(stock, self.data)
62              generated_signal = strategy_instance.generate_signals()
63              data = strategy_instance.extract_data()
64              self.data_queue.put(data)
65              strategy_instance.backtest()
66              if not strategy_instance.test_results()[1] > 0:
67                  self.stdout_queue.put(f'Skipping {Strategy.__name__} for {stock} due to negative performance. ')
68                  continue
69              signals.append(generated_signal)
70              _, annual_std = strategy_instance.calculate_statistics(self.in_minutes)
71              volatility_for_stock.append(annual_std)
72              results = strategy_instance.extract_results()
73              self.data_queue.put(results)
74
75          if signals:
76              average_volatility = sum(volatility_for_stock) / len(volatility_for_stock)
77
78              if all(signal == 1 for signal in signals) and int(current_holding) == 0:
79                  num_shares = calculate_position_size(account_balance, current_price, average_volatility,
80                                                       correlation_for_stock)
81                  market_order_data = MarketOrderRequest(
82                      symbol=stock,
83                      qty=num_shares,
84                      side=OrderSide.BUY,
85                      type='market',
86                      time_in_force=TimeInForce.IOC,
87                  )
88                  market_order = self.trading_client.submit_order(
89                      order_data=market_order_data
90                  )
91                  self.stdout_queue.put(f'Placed a buy order for {num_shares} shares of {stock}. ')
92
93              elif all(signal == -1 for signal in signals) and int(current_holding) > 0:
94                  market_order_data = MarketOrderRequest(
95                      symbol=stock,
96                      qty=current_holding,
```

```
                    side=OrderSide.SELL,
                    type='market',
                    time_in_force=TimeInForce.IOC,
                )
                market_order = self.trading_client.submit_order(
                    order_data=market_order_data
                )
                self.stdout_queue.put(f'Placed a sell order for {current_holding} shares of {stock}. ')
        else:
            self.stdout_queue.put("No action. ")
            pass

    1 usage
    def update_rolling(self, symbol, new_value):
        try:
            if self.data.index.dtype == 'int64':
                self.data.loc[self.data.index[-1] + 1] = self.data.iloc[-1]
            else:
                self.data.loc[self.data.index[-1]] = self.data.iloc[-1]
            # Update the new last row's specified column with the new value
            self.data.at[self.data.index[-1], symbol] = new_value
            # Drop the first row
            self.data.drop(self.data.index[0], inplace=True)
            # Reset index, so it remains sequential
            self.data.reset_index(drop=True, inplace=True)

        except Exception as e:
            print(f"Error in update_rolling: {e}")
            traceback.print_exc()

    1 usage
    def bot_message_processor(self):
        self.start_event.wait()
        try:
            print("Message processor started and waiting for messages... ")
            while self.is_running:
                message = self.message_queue.get()
                print("Processing a new message... ")
                msg = json.loads(message)
                # Ensure the message is a list and has at least one item
                if isinstance(msg, list) and len(msg) > 0 and "S" in msg[0]:
                    symbol = msg[0]["S"]
                    close = [msg[0]["c"]]
                    self.update_rolling(symbol, close[0])
                    self.action(symbol, close[0])
                else:
                    print(f"Ignoring message: {msg} ")

                self.message_queue.task_done()  # Mark the task as done after processing

        except Exception as e:
```

```
            print(f"Exception in message_processor: {e}")
            traceback.print_exc()

    1 usage
    def on_open(self, ws):
        self.reconnection_attempts = 0
        print("Opened connection")
        auth_data = {"action": "auth", "key": self.API_KEY, "secret": self.API_SECRET_KEY}
        ws.send(json.dumps(auth_data))
        listen_message = {"action": "subscribe", "bars": self.stocks}
        ws.send(json.dumps(listen_message))

    1 usage
    def on_message(self, ws, message):
        print("Received a message")
        print(message)
        self.message_count += 1
        if self.message_count % self.x == 0:
            self.message_queue.put(message)
            print(f"Queued message: {message[:100]}...")  # Print the first 100 characters of the message for brevity
            print("Is the processor thread alive?", self.processor_thread.is_alive())

    1 usage
    def on_close(self, ws):
        print("Closed connection")

    1 usage
    def on_error(self, ws, error):

        print("Error", error)
        # Check if max attempts reached
        if self.reconnection_attempts >= self.MAX_RECONNECTION_ATTEMPTS:
            print(f"Reached max reconnection attempts ({self.MAX_RECONNECTION_ATTEMPTS}). Not reconnecting.")
            self.stdout_queue.put("Connection error. Please restart the program. ")
            self.stop()
            return

        # If not, attempt to reconnect
        time.sleep(5)  # Wait for 5 seconds before attempting to reconnect
        self.reconnection_attempts += 1  # Increment the counter
        print(f"Reconnection attempt {self.reconnection_attempts} of {self.MAX_RECONNECTION_ATTEMPTS}")
        self.start_socket()  # Try to restart the connection

    2 usages
    def start_socket(self):
        socket = "wss://stream.data.alpaca.markets/v2/iex"
        self.ws = websocket.WebSocketApp(socket, on_open=self.on_open, on_message=self.on_message,
                                         on_close=self.on_close, on_error=self.on_error)
        self.ws.run_forever()

    2 usages (1 dynamic)
    def start(self):
        self.is_running = True
        while self.is_running:
            if check_if_open():
```

```
                self.processor_thread = threading.Thread(target=self.bot_message_processor)
                self.processor_thread.start()
                self.start_event.set()  # Signal the processor thread to start processing
                self.start_socket()  # Open the connection
            else:
                self.stdout_queue.put("Sleeping till market opens. ")
                sleep()

    3 usages (2 dynamic)
    def stop(self):
        self.is_running = False
        if hasattr(self, 'ws') and self.ws:
            self.ws.close()
        if hasattr(self, 'processor_thread') and self.processor_thread.is_alive():
            self.start_event.clear()  # Signal the processor thread to stop processing
            self.processor_thread.join()
```

live_market.py

```
# IMPORTING MODULES
import datetime
import pytz
import time
from alpaca.trading.client import TradingClient
from livebot.live_config import config

# DECLARING GLOBAL VARIABLES
market_open_time = datetime.time(9, 30)
market_close_time = datetime.time(16, 0)
market_timezone = pytz.timezone('America/New_York')


2 usages
def sleep():
    configurations = config()
    API_KEY = configurations[4]
    API_SECRET_KEY = configurations[5]
    trading_client = TradingClient(API_KEY, API_SECRET_KEY, paper=True)
    now = datetime.datetime.now(market_timezone)
    clock = trading_client.get_clock()
    next_open = clock.next_open.astimezone(market_timezone)
    time_until_open = next_open - now
    sleep_duration = time_until_open.total_seconds() + 5
    print(f"Sleeping for {time_until_open} until market opens...")
```

```
    time.sleep(sleep_duration)


2 usages
def check_if_open():
    current_datetime = datetime.datetime.now(market_timezone)
    current_day = current_datetime.weekday()

    if current_day > 4:
        return False

    start_datetime = current_datetime.replace(hour=market_open_time.hour, minute=market_open_time.minute, second=0,
                                              microsecond=0)
    end_datetime = current_datetime.replace(hour=market_close_time.hour, minute=market_close_time.minute, second=0,
                                            microsecond=0)

    if start_datetime <= current_datetime <= end_datetime:
        return True
    return False
```

live_risk.py

```
# IMPORTING USER-DEFINED MODULES
from livebot.live_datagrab import request_data
from livebot.live_config import config


2 usages
def get_correlation_data():
    configurations = config()
    stocks = configurations[2]
    correlation_data = []
    df = request_data(stocks)
    for stock in stocks:
        correlation_matrix = df.corr()
        stock_correlations = correlation_matrix[stock]
        average_correlation = stock_correlations.mean()
        correlation_data.append(average_correlation)
    return correlation_data


4 usages
class Node:
    def __init__(self, weight, value, next=None):
        self.weight = weight
        self.value = value
        self.next = next

```

```
25
    2 usages
26  def compute_score(node):
27      if not node:
28          return 0
29      return node.weight * node.value + compute_score(node.next)
30
31
    2 usages
32  def calculate_position_size(balance, price, volatility, correlation):
33      # Default value if volatility is None
34      if volatility is None:
35          volatility = 0.5  # Middle ground
36
37      # Maximum allocation percentage
38      MAX_ALLOCATION = 0.001  # Can be adjusted
39
40      # 1. Dynamic Weights
41      total_weight = 1
42      w_volatility = 0.25 + 0.1 * volatility
43      w_correlation = 0.25 - 0.1 * volatility  # The weight will decrease as correlation increases
44      w_balance = 0.25
45      w_price = total_weight - (w_volatility + w_correlation + w_balance)
46
47      # 2. Normalization and Linked List Creation
48      normalized_volatility = volatility
49      normalized_correlation = 1 - correlation  # Inverting correlation value so higher correlation reduces score
50      normalized_balance = balance / max(balance, 1)
51      normalized_price = price / max(price, 1)
52
53      # Creating linked list: Head -> Volatility -> Correlation -> Balance -> Price -> None
54      price_node = Node(w_price, normalized_price)
55      balance_node = Node(w_balance, normalized_balance, price_node)
56      correlation_node = Node(w_correlation, normalized_correlation, balance_node)
57      head = Node(w_volatility, normalized_volatility, correlation_node)
58
59      # 3. Compute combined score using recursion
60      score = compute_score(head)
61
62      # 4. Risk Management
63      risk_factor = 0.5 + 0.5 * volatility
64      position_size = score * balance * (1 - risk_factor) / price
65
66      # Scaling down by dividing with a constant
67      SCALING_FACTOR = 20  # Adjust based on user preference
68      position_size = position_size / SCALING_FACTOR
69
70      # 5. Adjust for Maximum Allocation
71      position_size = min(position_size, balance * MAX_ALLOCATION)
72      position_size = max(1, position_size)
73
74      return int(position_size)
75
```

live_strategy.py

```python
# IMPORTING LIBRARY MODULES
import numpy as np
import pandas as pd


1 usage
class Momentum:
    def __init__(self, symbol, data, lookback_period=10):
        self.symbol = symbol
        self.data = data[[self.symbol]].copy()
        self.lookback_period = lookback_period
        self.results = None

    2 usages (2 dynamic)
    def generate_signals(self):
        self.data['momentum'] = self.data[self.symbol] - self.data[self.symbol].shift(self.lookback_period)
        self.data['buy_signal'] = np.where((self.data['momentum'] > 0), 1, 0)
        self.data['sell_signal'] = np.where((self.data['momentum'] < 0), -1, 0)
        self.data['signal'] = self.data['buy_signal'] + self.data['sell_signal']
        self.data.dropna(inplace=True)
        return self.data['signal'].iat[-1]

    2 usages (2 dynamic)
    def extract_data(self):
        y_close_price = self.data[self.symbol]
        y_momentum = self.data['momentum']
        extracted_data = pd.DataFrame({

            'Close Price': y_close_price,
            'Momentum': y_momentum
        })
        return extracted_data

    2 usages (2 dynamic)
    def backtest(self):
        self.data["returns"] = np.log(self.data[self.symbol].div(self.data[self.symbol].shift(1)))
        self.data.dropna(inplace=True)
        self.data['position'] = self.data['buy_signal'] - self.data['sell_signal']

    2 usages (2 dynamic)
    def test_results(self):
        self.data["strategy"] = self.data["returns"] * self.data["position"].shift(1)
        self.data.dropna(inplace=True)
        self.data["returnsbh"] = self.data["returns"].cumsum().apply(np.exp)
        self.data["returnsstrategy"] = self.data["strategy"].cumsum().apply(np.exp)
        perf = self.data["returnsstrategy"].iloc[-1]
        outperf = perf - self.data["returnsbh"].iloc[-1]
        self.results = self.data
        return round(perf, 6), round(outperf, 6)

    2 usages (2 dynamic)
    def calculate_statistics(self, minute_interval):
        if self.data is None or 'strategy' not in self.data.columns:
            print("Strategy results not found. Please run the backtest first.")
            return
```

```
        scaling_unit = 60  # 60 minutes in an hour
        scaling_factor = np.sqrt(scaling_unit / minute_interval)

        # Calculate the average return for the specific timeframe
        ret = np.exp(self.data["strategy"].mean() * scaling_factor)

        # Calculate the volatility, scaled to hourly
        std = self.data["strategy"].std() * scaling_factor

        return ret, std

    2 usages (2 dynamic)
    def extract_results(self):
        if self.results is None:
            print("No results to extract. Run the test first.")
            return None
        y_returnsbh = self.results['returnsbh']
        y_returnsstrategy = self.results['returnsstrategy']
        extracted_data = pd.DataFrame({
            'ReturnsBH': y_returnsbh,
            'ReturnsStrategy': y_returnsstrategy
        })
        return extracted_data
```

live_strategy2.py

```
# IMPORTING LIBRARY MODULES
import numpy as np
import pandas as pd


1 usage
class Scalping:
    def __init__(self, symbol, data, threshold=0.001):
        self.symbol = symbol
        self.data = data[[self.symbol]].copy()
        self.threshold = threshold
        self.results = None

    2 usages (2 dynamic)
    def generate_signals(self):
        self.data['price_diff'] = self.data[self.symbol].pct_change()
        self.data['buy_signal'] = np.where((self.data['price_diff'] > self.threshold), 1, 0)
        self.data['sell_signal'] = np.where((self.data['price_diff'] < -self.threshold), -1, 0)
        self.data['signal'] = self.data['buy_signal'] + self.data['sell_signal']
        self.data.dropna(inplace=True)
        return self.data['signal'].iat[-1]

    2 usages (2 dynamic)
    def extract_data(self):
        y_close_price = self.data[self.symbol]
        y_price_diff = self.data['price_diff']
        extracted_data = pd.DataFrame({
```

```
            'Close Price': y_close_price,
            'Price Difference': y_price_diff
        })
        return extracted_data

    2 usages (2 dynamic)
    def backtest(self):
        self.data["returns"] = np.log(self.data[self.symbol].div(self.data[self.symbol].shift(1)))
        self.data.dropna(inplace=True)
        self.data['position'] = self.data['buy_signal'] - self.data['sell_signal']

    2 usages (2 dynamic)
    def test_results(self):
        self.data["strategy"] = self.data["returns"] * self.data["position"].shift(1)
        self.data.dropna(inplace=True)
        self.data["returnsbh"] = self.data["returns"].cumsum().apply(np.exp)
        self.data["returnsstrategy"] = self.data["strategy"].cumsum().apply(np.exp)
        perf = self.data["returnsstrategy"].iloc[-1]
        outperf = perf - self.data["returnsbh"].iloc[-1]
        self.results = self.data
        return round(perf, 6), round(outperf, 6)

    2 usages (2 dynamic)
    def calculate_statistics(self, minute_interval):
        if self.data is None or 'strategy' not in self.data.columns:
            print("Strategy results not found. Please run the backtest first.")
            return


        scaling_unit = 60  # 60 minutes in an hour
        scaling_factor = np.sqrt(scaling_unit / minute_interval)

        # Calculate the average return for the specific timeframe
        ret = np.exp(self.data["strategy"].mean() * scaling_factor)

        # Calculate the volatility, scaled to hourly
        std = self.data["strategy"].std() * scaling_factor

        return ret, std

    2 usages (2 dynamic)
    def extract_results(self):
        if self.results is None:
            print("No results to extract. Run the test first.")
            return None
        y_returnsbh = self.results['returnsbh']
        y_returnsstrategy = self.results['returnsstrategy']
        extracted_data = pd.DataFrame({
            'ReturnsBH': y_returnsbh,
            'ReturnsStrategy': y_returnsstrategy
        })
        return extracted_data
```

live_strategy3.py

```python
# IMPORTING LIBRARY MODULES
import numpy as np
import pandas as pd


1 usage
class MeanReversion:
    def __init__(self, symbol, data, lookback_period=10):
        self.symbol = symbol
        self.data = data[[self.symbol]].copy()
        self.lookback_period = lookback_period
        self.results = None

    2 usages (2 dynamic)
    def generate_signals(self):
        self.data['mean_price'] = self.data[self.symbol].rolling(window=self.lookback_period).mean()
        self.data['buy_signal'] = np.where((self.data[self.symbol] < self.data['mean_price']), 1, 0)
        self.data['sell_signal'] = np.where((self.data[self.symbol] > self.data['mean_price']), -1, 0)
        self.data['signal'] = self.data['buy_signal'] + self.data['sell_signal']
        self.data.dropna(inplace=True)
        return self.data['signal'].iat[-1]

    2 usages (2 dynamic)
    def extract_data(self):
        y_close_price = self.data[self.symbol]
        y_mean_price = self.data['mean_price']
        extracted_data = pd.DataFrame({

            'Close Price': y_close_price,
            'Mean Price': y_mean_price
        })
        return extracted_data

    2 usages (2 dynamic)
    def backtest(self):
        self.data["returns"] = np.log(self.data[self.symbol].div(self.data[self.symbol].shift(1)))
        self.data.dropna(inplace=True)
        self.data['position'] = self.data['buy_signal'] - self.data['sell_signal']

    2 usages (2 dynamic)
    def test_results(self):
        self.data["strategy"] = self.data["returns"] * self.data["position"].shift(1)
        self.data.dropna(inplace=True)
        self.data["returnsbh"] = self.data["returns"].cumsum().apply(np.exp)
        self.data["returnsstrategy"] = self.data["strategy"].cumsum().apply(np.exp)
        perf = self.data["returnsstrategy"].iloc[-1]
        outperf = perf - self.data["returnsbh"].iloc[-1]
        self.results = self.data
        return round(perf, 6), round(outperf, 6)

    2 usages (2 dynamic)
    def calculate_statistics(self, minute_interval):
        if self.data is None or 'strategy' not in self.data.columns:
            print("Strategy results not found. Please run the backtest first.")
            return
```

```
        scaling_unit = 60  # 60 minutes in an hour
        scaling_factor = np.sqrt(scaling_unit / minute_interval)

        # Calculate the average return for the specific timeframe
        ret = np.exp(self.data["strategy"].mean() * scaling_factor)

        # Calculate the volatility, scaled to hourly
        std = self.data["strategy"].std() * scaling_factor

        return ret, std

    2 usages (2 dynamic)
    def extract_results(self):
        if self.results is None:
            print("No results to extract. Run the test first.")
            return None
        y_returnsbh = self.results['returnsbh']
        y_returnsstrategy = self.results['returnsstrategy']
        extracted_data = pd.DataFrame({
            'ReturnsBH': y_returnsbh,
            'ReturnsStrategy': y_returnsstrategy
        })
        return extracted_data
```

Non-py files

turquoise.json

```
{
  "CTk": {
    "fg_color": ["gray95", "gray10"]
  },
  "CTkToplevel": {
    "fg_color": ["gray95", "gray10"]
  },
  "CTkFrame": {
    "corner_radius": 6,
    "border_width": 0,
    "fg_color": ["gray90", "gray13"],
    "top_fg_color": ["gray85", "gray16"],
    "border_color": ["gray65", "gray28"]
  },
  "CTkButton": {
    "corner_radius": 6,
    "border_width": 0,
    "fg_color": ["#A2C4C9", "#A2C4C9"],
    "hover_color": ["#8ca8ad", "#8ca8ad"],
    "border_color": ["#999999", "#999999"],
    "text_color": ["#FFFFFF", "#FFFFFF"],
    "text_color_disabled": ["#666666", "#666666"]
  },
  "CTkLabel": {
```

```
    "corner_radius": 0,
    "fg_color": "transparent",
    "text_color": ["#FFFFFF", "#FFFFFF"]
  },
  "CTkEntry": {
    "corner_radius": 6,
    "border_width": 2,
    "fg_color": ["#000000", "#000000"],
    "border_color": ["#434343", "#434343"],
    "text_color": ["#666666", "#FFFFFF"],
    "placeholder_text_color": ["gray52", "gray62"]
  },
  "CTkCheckBox": {
    "corner_radius": 6,
    "border_width": 3,
    "fg_color": ["#A2C4C9", "#A2C4C9"],
    "border_color": ["#3E454A", "#949A9F"],
    "hover_color": ["#8ca8ad", "#8ca8ad"],
    "checkmark_color": ["#DCE4EE", "gray90"],
    "text_color": ["gray14", "gray84"],
    "text_color_disabled": ["gray60", "gray45"]
  },
  "CTkSwitch": {
    "corner_radius": 1000,
    "border_width": 3,
    "button_length": 0,
    "fg_color": ["#939BA2", "#4A4D50"],
    "progress_color": ["#3a7ebf", "#1f538d"],
    "button_color": ["gray36", "#D5D9DE"],
    "button_hover_color": ["gray20", "gray100"],
    "text_color": ["gray14", "gray84"],
    "text_color_disabled": ["gray60", "gray45"]
  },
  "CTkRadioButton": {
    "corner_radius": 1000,
    "border_width_checked": 6,
    "border_width_unchecked": 3,
    "fg_color": ["#3a7ebf", "#1f538d"],
    "border_color": ["#3E454A", "#949A9F"],
    "hover_color": ["#325882", "#14375e"],
    "text_color": ["gray14", "gray84"],
    "text_color_disabled": ["gray60", "gray45"]
  },
  "CTkProgressBar": {
    "corner_radius": 1000,
    "border_width": 0,
    "fg_color": ["#939BA2", "#4A4D50"],
    "progress_color": ["#3a7ebf", "#1f538d"],
```

```
    "border_color": ["gray", "gray"]
  },
  "CTkSlider": {
    "corner_radius": 1000,
    "button_corner_radius": 1000,
    "border_width": 6,
    "button_length": 0,
    "fg_color": ["#939BA2", "#4A4D50"],
    "progress_color": ["gray40", "#AAB0B5"],
    "button_color": ["#3a7ebf", "#1f538d"],
    "button_hover_color": ["#325882", "#14375e"]
  },
  "CTkOptionMenu": {
    "corner_radius": 6,
    "fg_color": ["#3a7ebf", "#1f538d"],
    "button_color": ["#325882", "#14375e"],
    "button_hover_color": ["#234567", "#1e2c40"],
    "text_color": ["#DCE4EE", "#DCE4EE"],
    "text_color_disabled": ["gray74", "gray60"]
  },
  "CTkComboBox": {
    "corner_radius": 6,
    "border_width": 2,
    "fg_color": ["#F9F9FA", "#343638"],

    "border_color": ["#979DA2", "#565B5E"],
    "button_color": ["#979DA2", "#565B5E"],
    "button_hover_color": ["#6E7174", "#7A848D"],
    "text_color": ["gray14", "gray84"],
    "text_color_disabled": ["gray50", "gray45"]
  },
  "CTkScrollbar": {
    "corner_radius": 1000,
    "border_spacing": 4,
    "fg_color": "transparent",
    "button_color": ["gray55", "gray41"],
    "button_hover_color": ["gray40", "gray53"]
  },
  "CTkSegmentedButton": {
    "corner_radius": 6,
    "border_width": 2,
    "fg_color": ["#979DA2", "gray29"],
    "selected_color": ["#3a7ebf", "#1f538d"],
    "selected_hover_color": ["#325882", "#14375e"],
    "unselected_color": ["#979DA2", "gray29"],
    "unselected_hover_color": ["gray70", "gray41"],
    "text_color": ["#DCE4EE", "#DCE4EE"],
    "text_color_disabled": ["gray74", "gray60"]
  },
```

```
  "CTkTextbox": {
    "corner_radius": 6,
    "border_width": 0,
    "fg_color": ["gray100", "gray20"],
    "border_color": ["#979DA2", "#565B5E"],
    "text_color": ["#FFFFFF", "#FFFFFF"],
    "scrollbar_button_color": ["gray55", "gray41"],
    "scrollbar_button_hover_color": ["gray40", "gray53"]
  },
  "CTkScrollableFrame": {
    "label_fg_color": ["gray80", "gray21"]
  },
  "DropdownMenu": {
    "fg_color": ["gray90", "gray20"],
    "hover_color": ["gray75", "gray28"],
    "text_color": ["gray14", "gray84"]
  },
  "CTkFont": {
    "macOS": {
      "family": "SF Display",
      "size": 13,
      "weight": "normal"
    },
    "Windows": {
      "family": "Roboto",
      "size": 13,
      "weight": "normal"
    },
    "Linux": {
      "family": "Roboto",
      "size": 13,
      "weight": "normal"
    }
  }
}
```

Testing

Video evidence

Testing video can be accessed here: https://bit.ly/NEATST

If this link does not work, try: https://youtu.be/4_S8wLvdkSA?si=MNukob37Uciid91T

Alternately, scan this:

Testing table

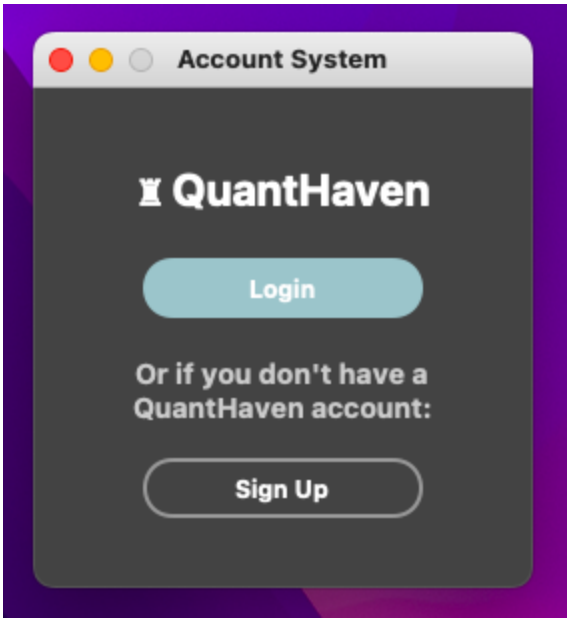
Section	Timestamp	Requirement(s)	Description	Pass/Fail
1: GUI & database testing	00:00 - 00:55	☑ 5.1 ☑ 5.1.2 ☐ 5.1.3 ☑ 6.1 & 6.1.1	A new account is made with ‘username1’ and ‘password1’. We see that valid Alpaca API keys are needed. Alpha Vantage validation proved to be redundant.	All pass except for 5.1.3
	00:55 - 01:05	☑ 6.4	This account is used to log in. We see the button for an input tab gets disabled once its respective window is open.	All pass
	01:05 - 01:30	☑ 6.1 & 6.1.1 ☑ 5.1.4	First glimpse of the home screen, we are greeted as ‘Name1’ (title case, after being entered in lowercase). The market is currently closed.	All pass
	01:30 - 02:20	☑ 4.1 (all sub objectives) ☑ 6.1 & 6.1.1 ☑ 6.5 & 6.5.1 ☑ 6.6	Here the ‘choose a stock’ feature is explored. There is a 60 second cool down due to free API key restrictions. We also see that capitalisation does not matter.	All pass
	02:20 - 04:20	☑ 4.2 ☑ 4.2.1	A valid time unit from ‘Minute’, ‘Hour’ or ‘Day’ must be entered	All pass

		☑ 4.2.1.1 ☑ 4.3.1 ☑ 6.1 & 6.1.1	along with an integer amount. The pop up windows are different depending on the time frames and which category of bot it falls under.	
	04:20 - 4:50	☑ 4.2.2	A few more records are added. Clicking on a record will select it and allow the user to either delete it or run it.	All pass
	04:50 - 5:40	☑ 2.6 ☑ 2.10 ☑ 2.11, 2.11.2 & 2.11.4 ☑ 4.3 (all sub objectives) ☑ 6.5 & 6.5.1	Two bots are run (one live and one historical) while the market is closed.	All pass
	5:40 - 5:45	☑ 6.3	Here, we can see that three new files have been added to the directory; 'database.db', 'bot_configuration.txt' and 'keys.key'. This will happen when the program is run for the first time.	All pass
	5:40 - 6:50	☑ 5.1.1 & 5.1.2 ☑ 6.1 & 6.1.1	Here a new account is made with 'username2' and 'password2'. The same Alpaca and Alpha Vantage account is used.	All pass
	6:50 - 08:45	☑ 2.11.3 ☑ 2.11.1 ☑ 3 (all sub objectives)	A live bot and a historical bot is run using all available strategies for two stocks each. One iteration is seen.	All pass
	08:45 - 09:25	☑ 5.2 (all sub objectives) ☑ 5.3, 5.3.1 & 5.3.2	Here information for the two accounts are seen in database.db. Passwords are shown to be hashed and API keys encrypted.	All pass
	09:25 - 09:50	☑ 5.3.3 ☑ 5.4.1.	The old account is logged back into and we see that only its	All pass

			own configurations are shown. 'Forget account' is clicked.	
	09:50 - 10:35	☑ 5.4	When we go back to database.db, we see that the account of user_id 1 and any related data is now gone. We also see that 'bot_configurations.txt' is used to store the configurations of the latest bot to be run. If we try login again with 'username1', we will be met with 'user not found'.	All pass
2: Proof of trade execution	10:35 - 12:55	☑ 1.2 ☑ 2.1 ☑ 2.2 ☑ 2.3 ☑ 2.4 ☑ 2.5 ☑ 3 (all sub objectives)	A live bot (3 minutes) is run using all available strategies for stocks AAPL, and MSFT. This is shown in 20x speed. We see both buy and sell orders being placed for both stocks.	All pass
	12:55 - 14:10	☑ 2.6.1 ☑ 3 (all sub objectives)	A historical bot (16 minutes) is run using all available strategies for stocks AAPL, and MSFT. This is shown in 200x speed. We see both buy and sell orders being placed for AAPL. We also see the point at which the market closes and the bot goes back to sleep.	All pass
3: Program integrity testing	14:10 - 17:05	☑ 6.7 ☑ 6.8 (all sub objectives)	Here, the program is terminated and interacted with in ways we haven't already seen (covering every possible way). All possibilities safely close the program (exit code 0).	All pass

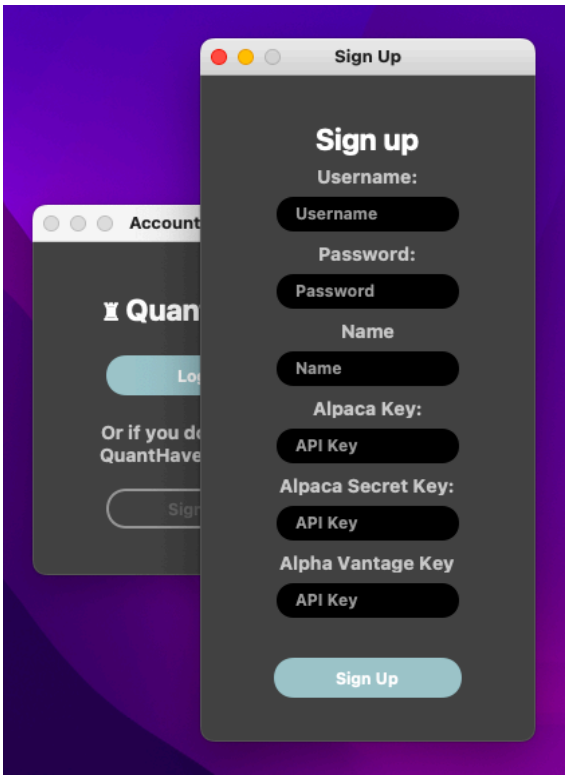
GUI Testing - Account system

Sign up

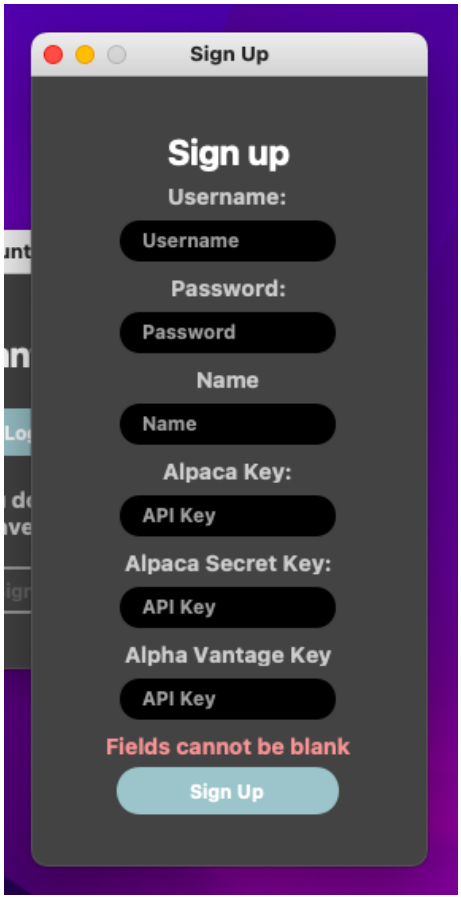


Test number	Description	Data type	Expected result	Pass/Fail	Cross reference
1	Sign up button pressed	Typical	The sign up window opened + sign up button disabled	Pass	Screenshot 1
2	Empty form + submit button pressed	Null	Error “Fields cannot be blank”	Pass	Screenshot 2
3	Existing user + submit button pressed	Erroneous	Error “Username already exists”	Pass	Screenshot 3
4	Invalid Alpaca key(s) + submit button pressed	Erroneous	Error “Alpaca keys invalid”	Pass	Screenshot 4
5	Invalid Alpha Vantage key + submit button pressed	Erroneous	Error “Alpha Vantage key invalid”	Fail	-
6	Sign up window	Typical	Window closed +	Pass	-

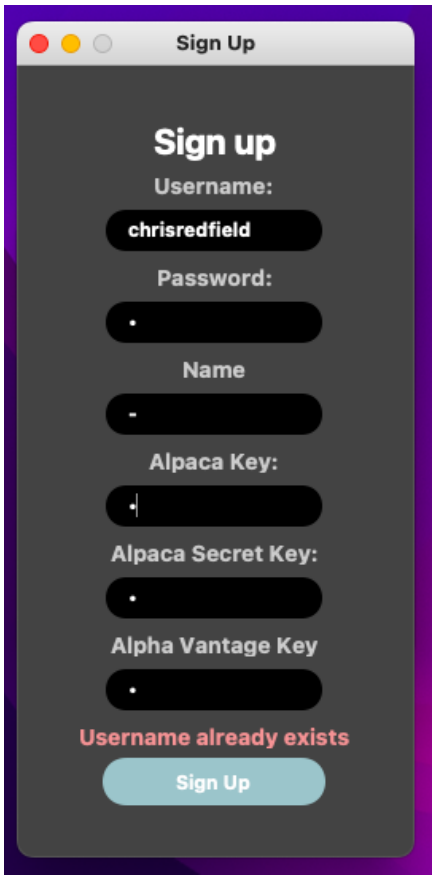
	closed using “x”		sign up button enabled		
7	Correct details + submit button pressed	Typical	Window closed + sign up button enabled	Pass	-



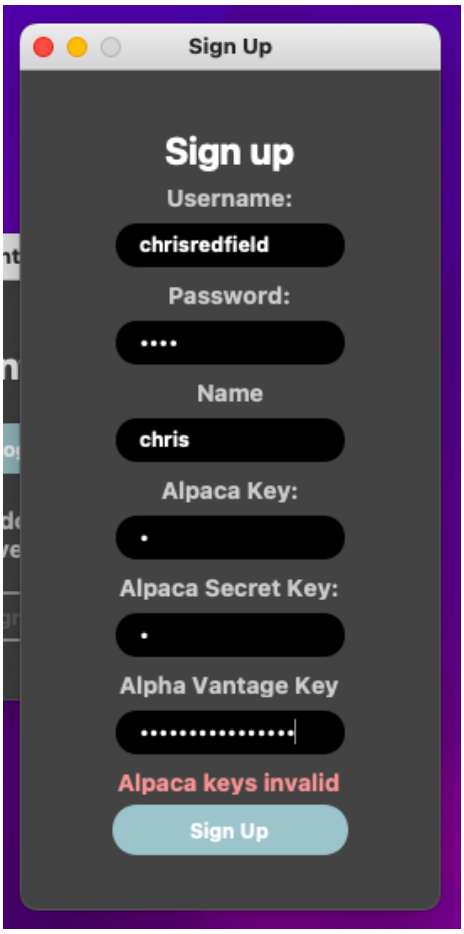
Screenshot 1



Screenshot 2



Screenshot 3

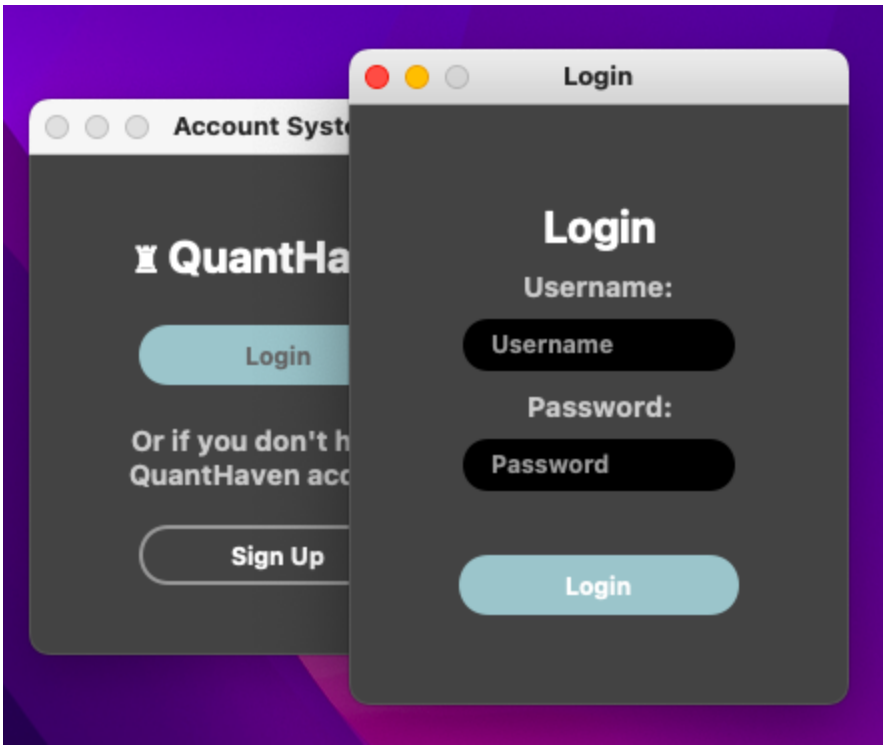


Screenshot 4

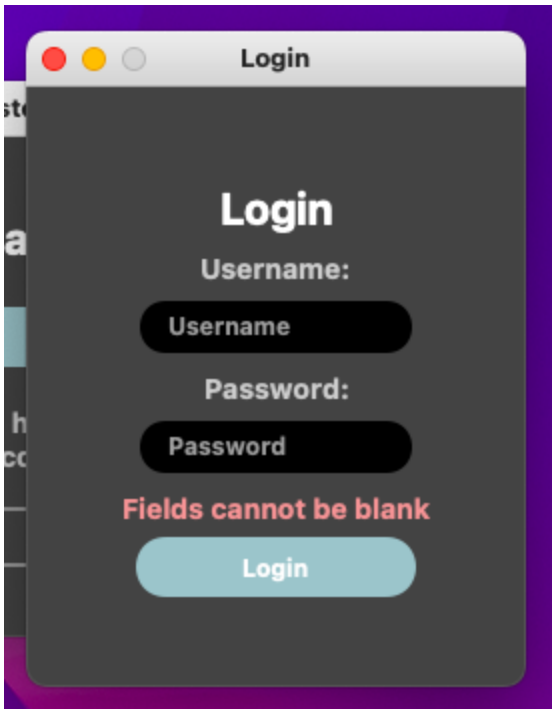
Login

Test number	Description	Data type	Expected result	Pass/Fail	Cross reference
1	Login button pressed	Typical	Login window opened + login button disabled	Pass	Screenshot 5
2	Empty form + submit	Null	Error “Fields cannot be blank”	Pass	Screenshot 6
3	Wrong credentials + submit button	Erroneous	Error “User not found”	Pass	Screenshot 7

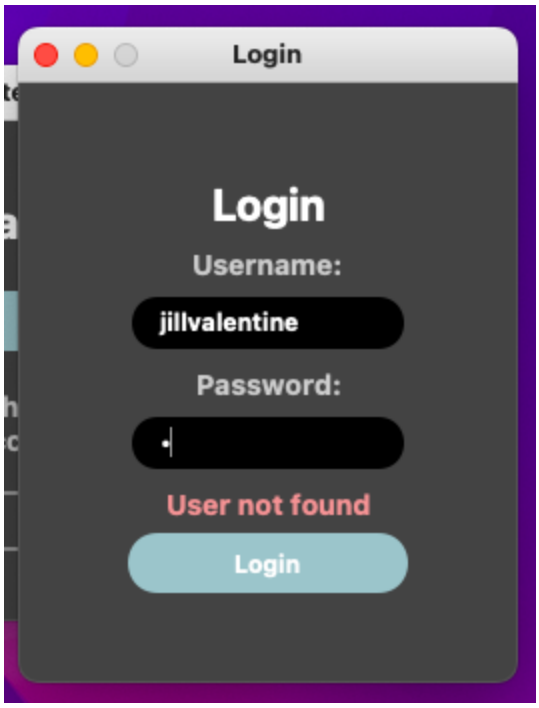
	pressed				
4	Correct username + wrong password + submit button pressed	Erroneous	Error “Password does not match user”	Pass	Screenshot 8
5	Login window closed using “x”	Typical	Login window closed + login button enabled	Pass	-
6	Correct credentials + submit button pressed	Typical	Login window closed + login button enabled + account system window closed	Pass	-



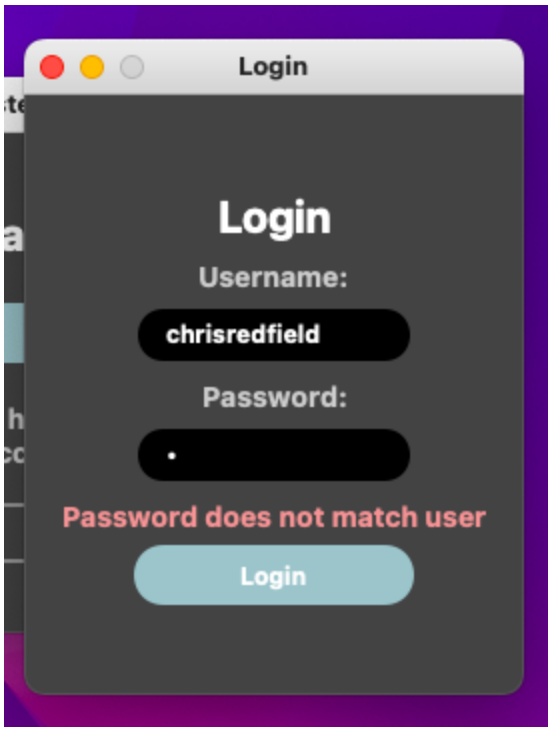
Screenshot 5



Screenshot 6

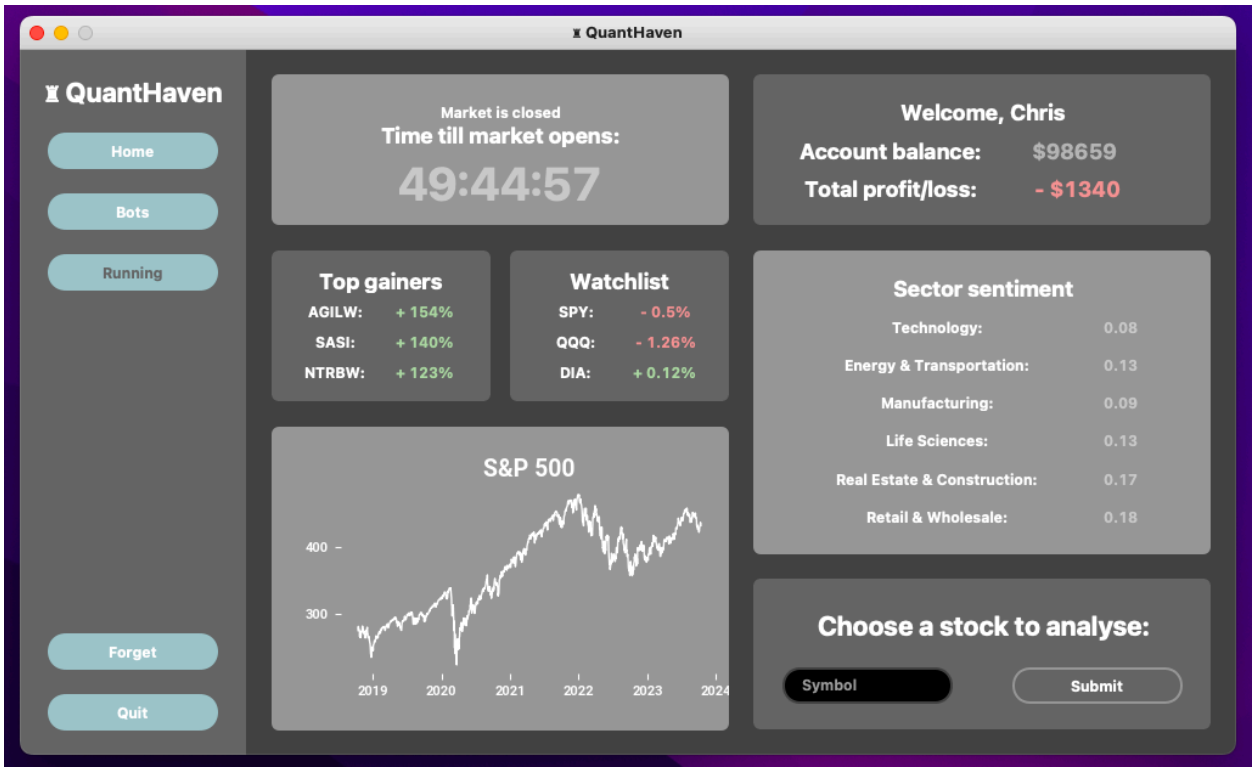


Screenshot 7



Screenshot 8

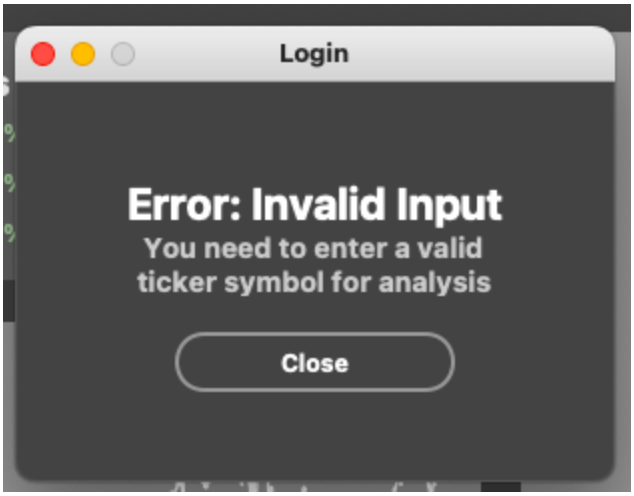
GUI Entry testing - Main App



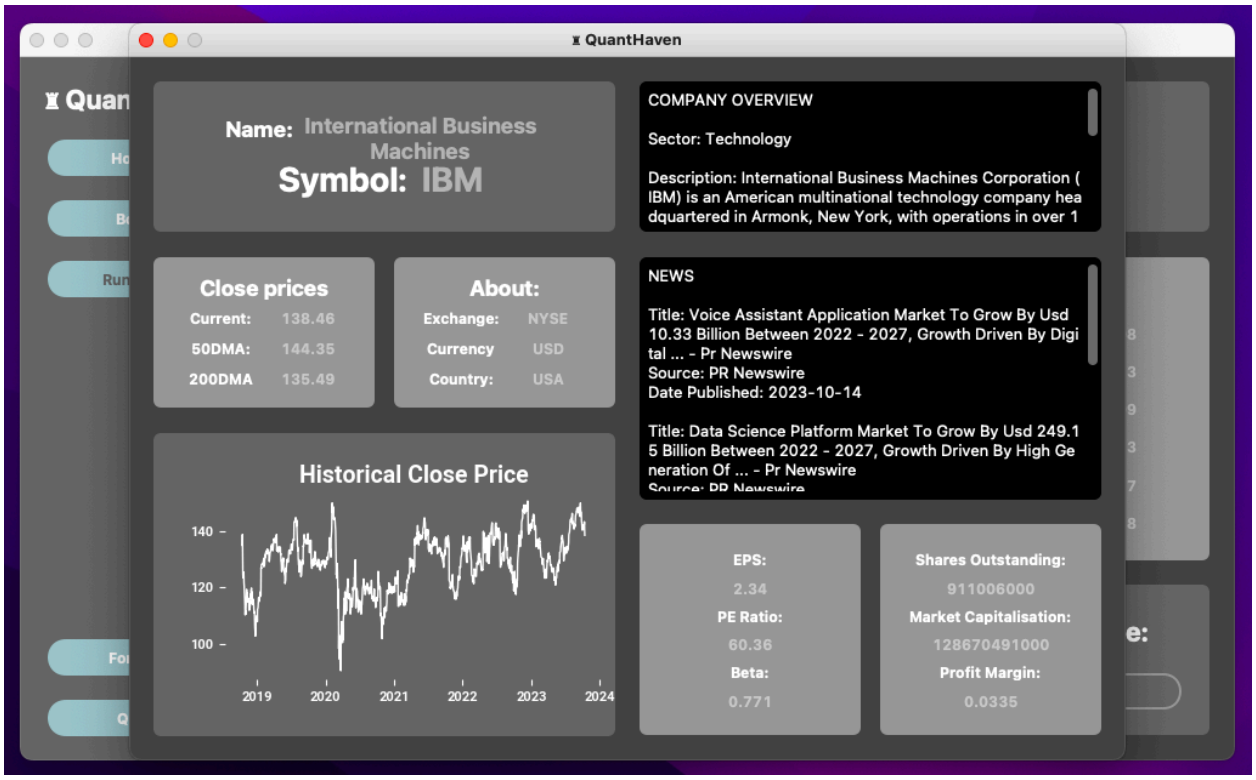
Stock entry

Test number	Description	Data type	Expected result	Pass/Fail	Cross reference
1	Empty entry + submit button pressed	Null	Error “Invalid input” + entry auto cleared	Pass	Screenshot 9
2	Invalid symbol + submit button pressed	Erroneous	Error “Invalid input” + entry auto cleared	Pass	Screenshot 9
3	Valid symbol + submit button pressed	Typical	Pop up window opened + entry auto cleared	Pass	Screenshot 10

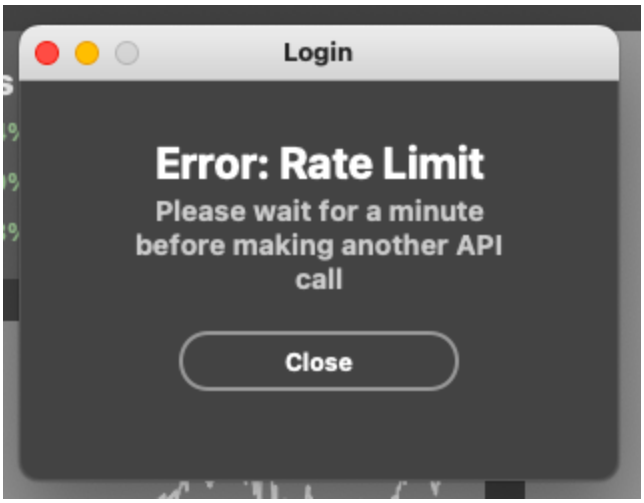
4	Valid symbol (incorrect capitalisation) + submit button pressed	Typical	Pop up window opened + entry auto cleared	Pass	Screenshot 10
5	Valid symbol entered under 60s of a pop up window being opened	Typical	Error “Rate limit”	Pass	Screenshot 11



Screenshot 9



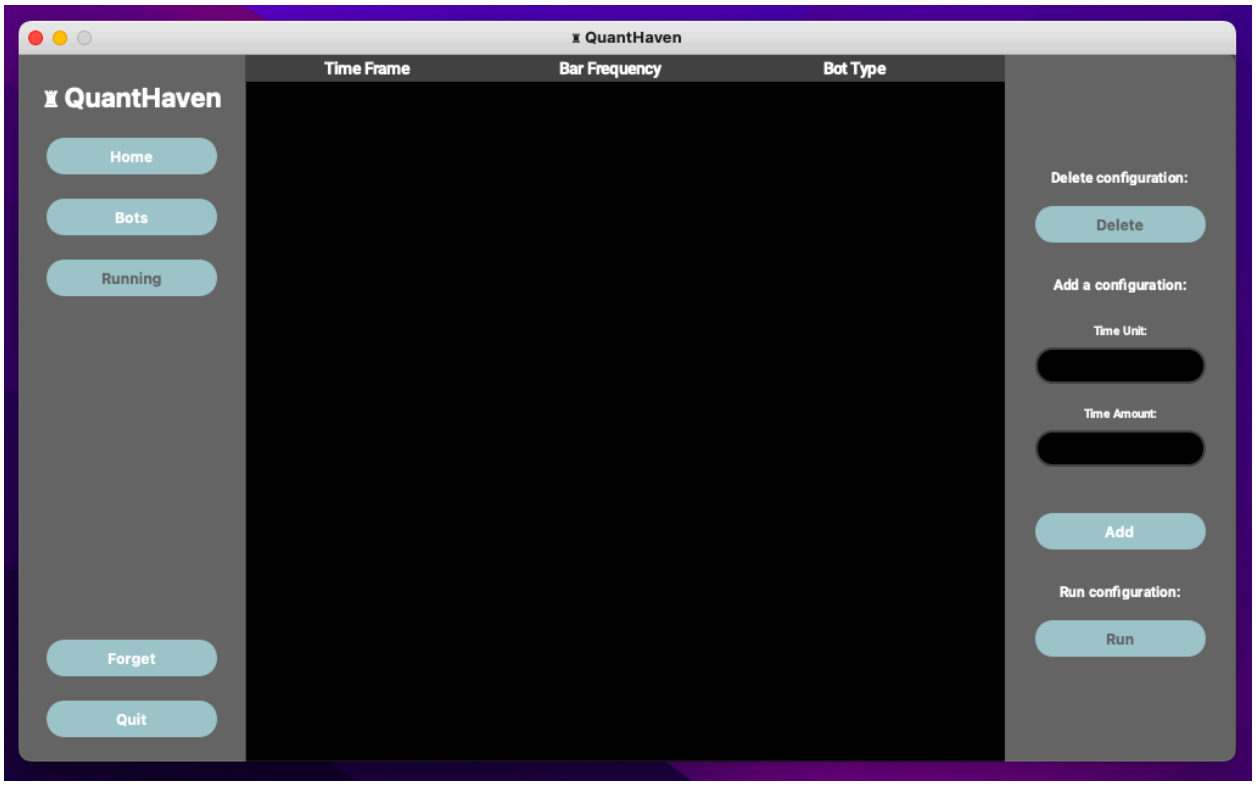
Screenshot 10



Screenshot 11

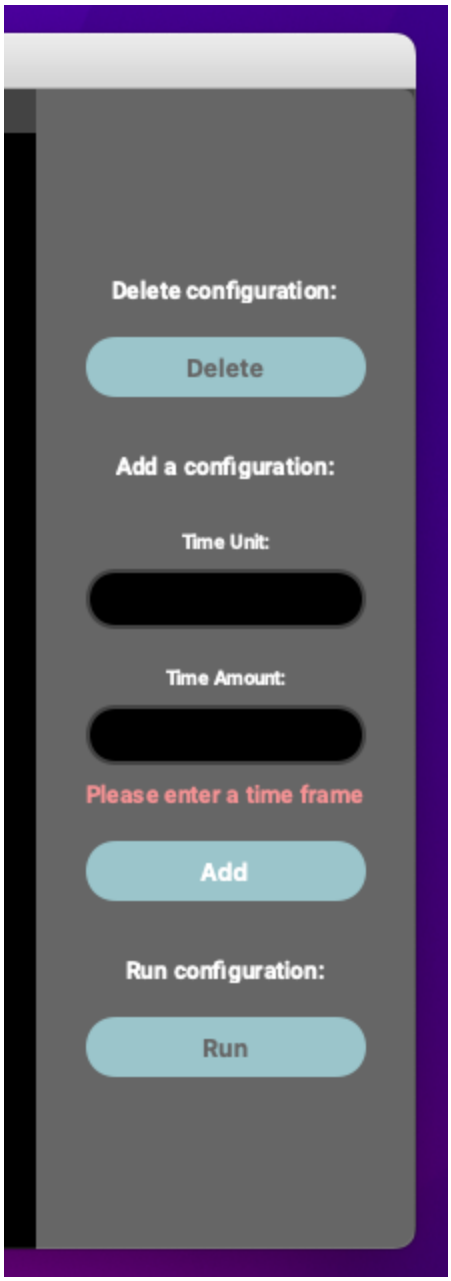
Bot configuration entry

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan.eet dolore magna aliquam erat volutpat.

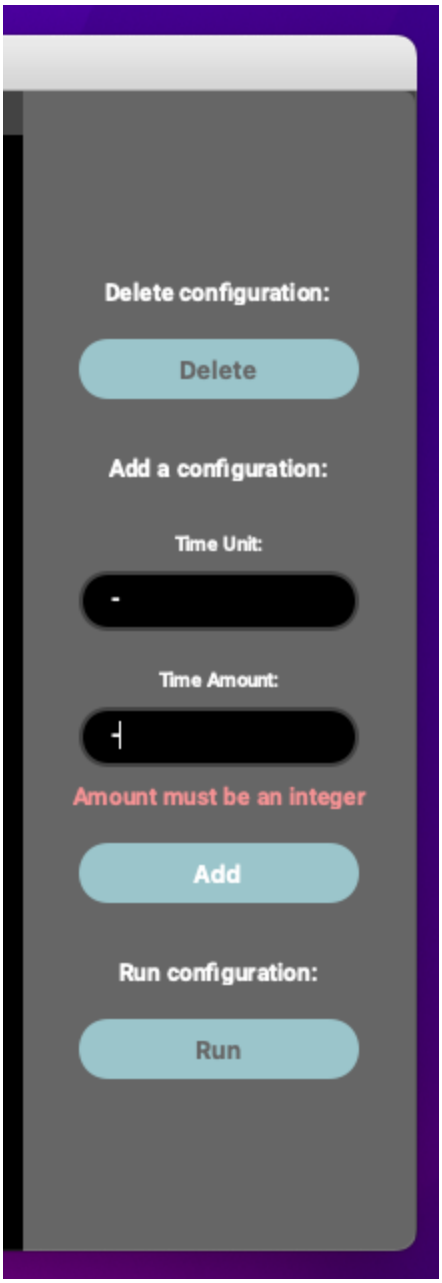


Test number	Description	Data type	Expected result	Pass/Fail	Cross Reference
1	Empty form + add button pressed	Null	Error “Please enter a time frame”	Pass	Screenshot 12
2	Wrong time unit and amount + add button pressed	Erroneous	Error “Amount must be an integer”	Pass	Screenshot 13
3	Correct amount + wrong time unit + add button pressed	Erroneous	Error “Unsupported unit”	Pass	Screenshot 14
4	Small time frame (<15 mins) + add button pressed	Typical	Live bot customisation window opens + add button disabled	Pass	Screenshot 15

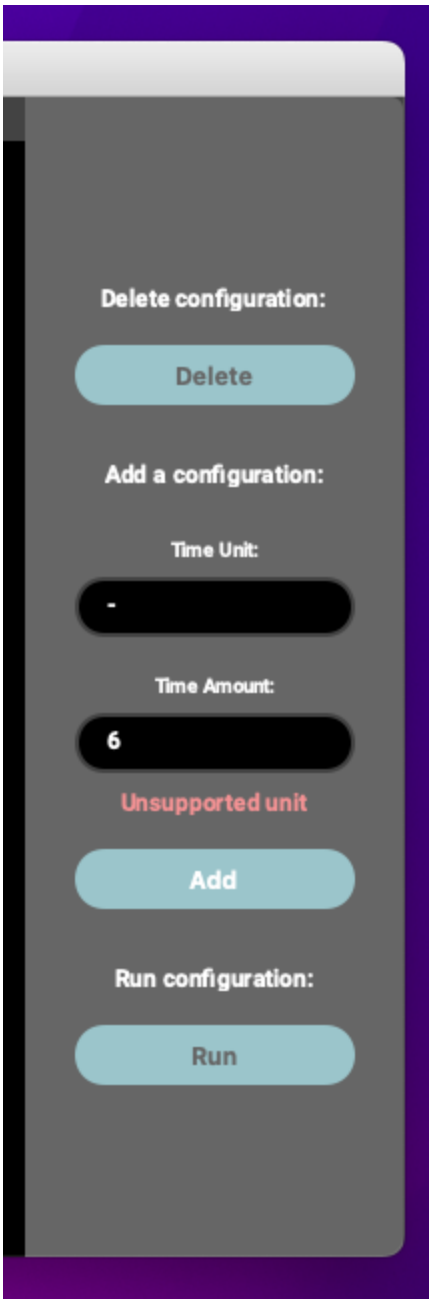
5	Large time frame (>15 mins) + wrong capitalization + add button pressed	Typical	Historical bot customisation window opens + add button disabled	Pass	Screenshot 16



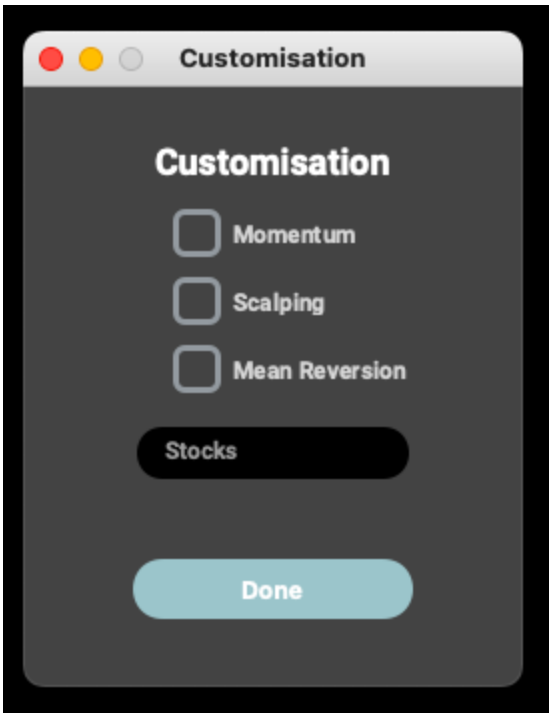
Screenshot 12



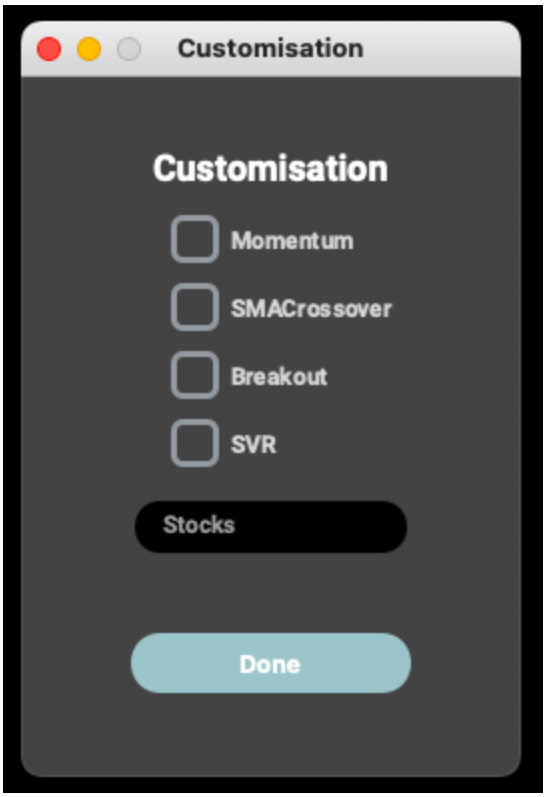
Screenshot 13



Screenshot 14



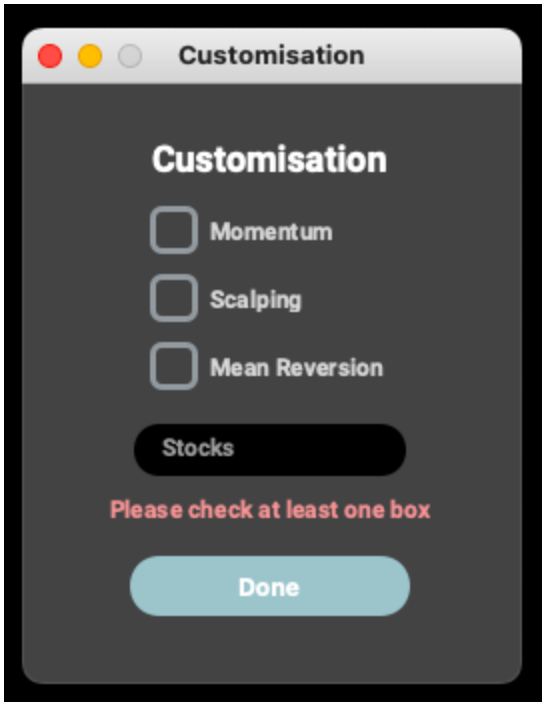
Screenshot 15



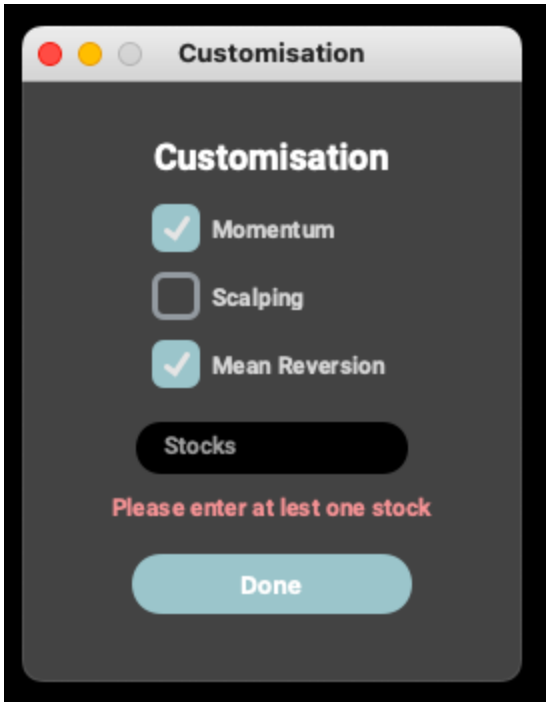
Screenshot 16

Customisation window(s)

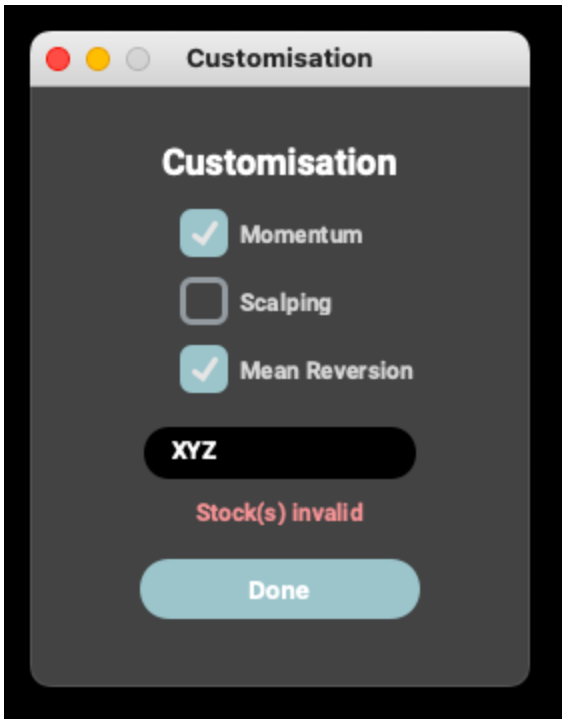
Test number	Description	Data type	Expected result	Pass/Fail	Cross Reference
1	Empty form + done button pressed	Null	Error "Please check atleast one box"	Pass	Screenshot 17 & 22
2	Checked boxes + done button pressed	Null	Error "Please enter atleast one stock"	Pass	Screenshot 18 & 23
3	Checked boxes + wrong stock + done button pressed	Erroneous	Error "Stock(s) invalid"	Pass	Screenshot 19 & 24
4	Checked boxes + correct stock + done button pressed	Typical	Window closes + add button enabled	Pass	-
5	Checked boxes + multiple correct stocks + one wrong stock + done button pressed	Erroneous	Error "Stock(s) invalid"	Pass	Screenshot 20 & 25
6	Checked boxes + too many (>3) correct stocks + done button pressed	Erroneous	Error "Too many stocks entered"	Pass	Screenshot 21 & 26
7	Multiple correct stocks (<3) + done button pressed	Typical	Window closes + add button enabled	Pass	-
8	Closing unfinished tab using "x"	Typical	Window closes + add button enabled	Pass	-



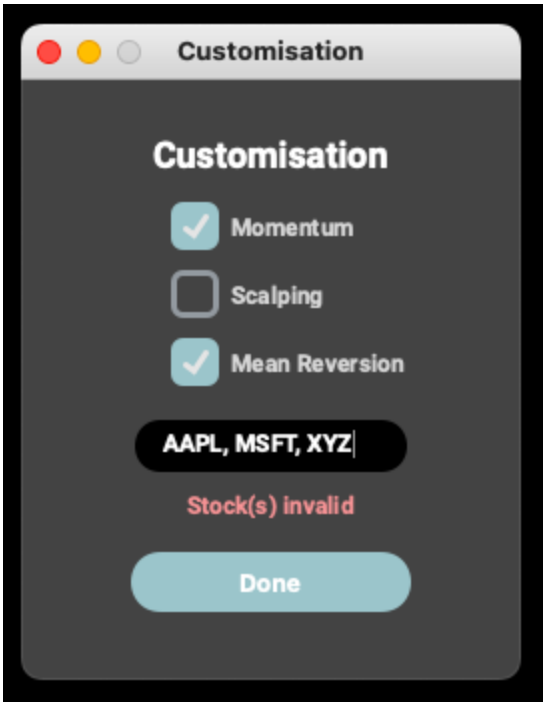
Screenshot 17



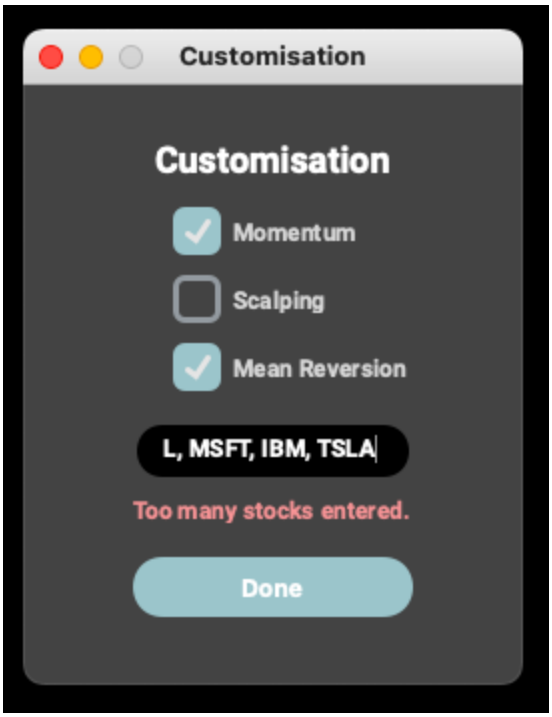
Screenshot 18



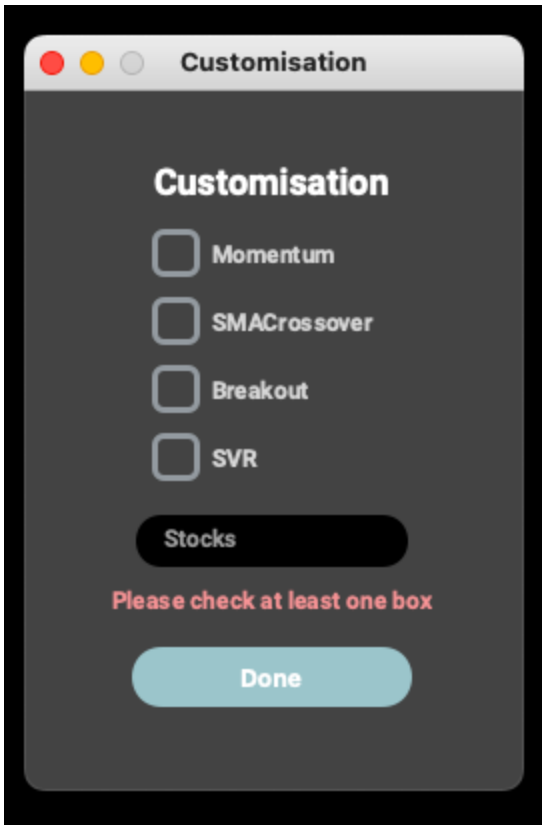
Screenshot 19



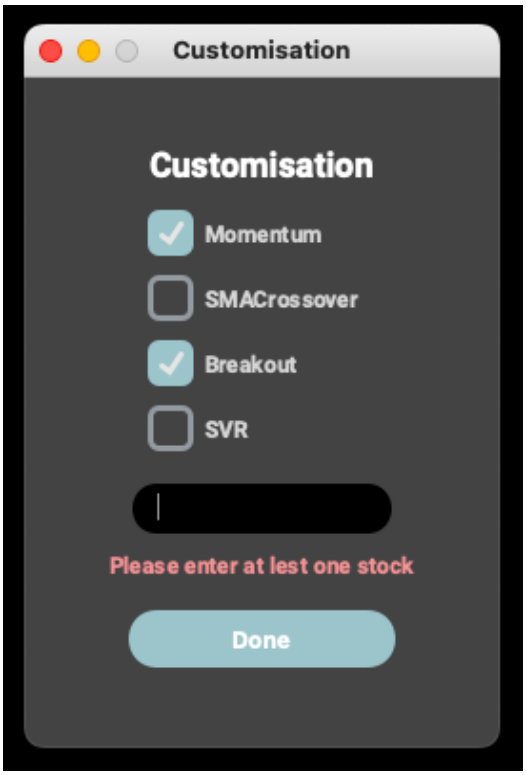
Screenshot 20



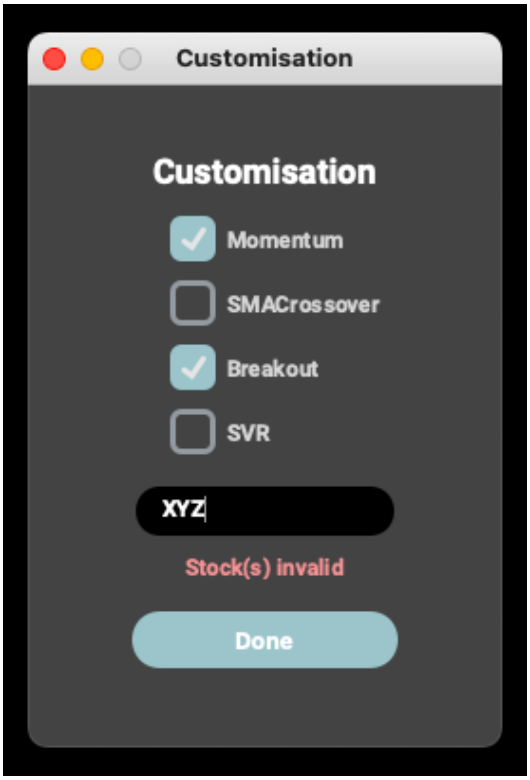
Screenshot 21



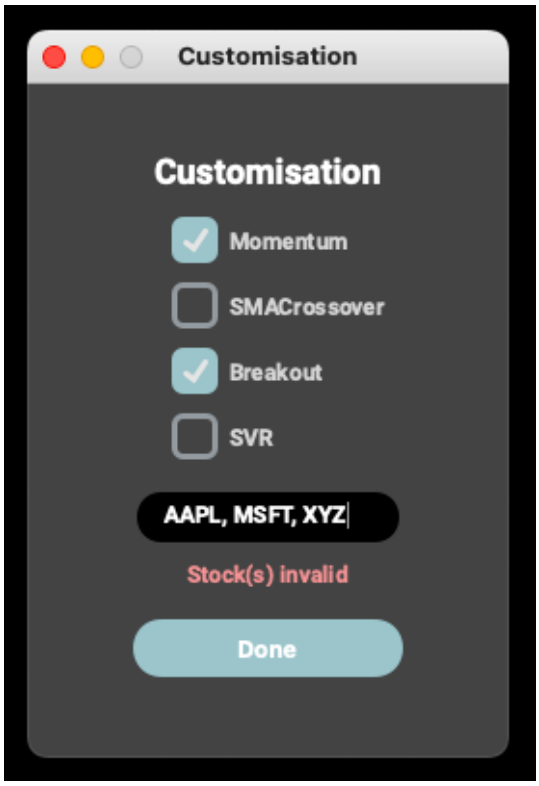
Screenshot 22



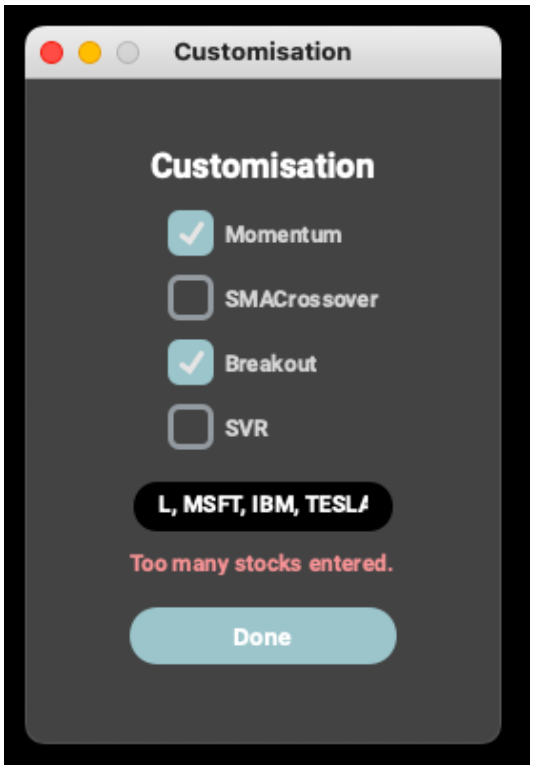
Screenshot 23



Screenshot 24



Screenshot 25



Screenshot 26

Evaluation

Overview

Key:

Objective met	Objective mostly met	Objective partially met	Objective not met

#	Objective	Comments
1	APIs	The program successfully uses the user's Alpaca account to both request data and execute trades. However, the user's Alpha Vantage account seemed to be redundant, as the API returns data even with invalid API keys.
2	Trading bots	Two types of trading bots (live and historical) with vastly different infrastructure have been implemented. Both types allow the user to customize time frames, iterate through multiple user selected stocks, and allow for multiple strategies (user adjustable). Each strategy is tested against buy and hold and only bought/sold if all/any valid strategies confirm for that stock. They can also handle market closing/opening, and risk will never be too high. For live: • Based on streamed data from a websocket connection. • Maintains a 'rolling window' of data. • Position size based on correlation and volatility. For historical: • Based on historical data • Uses machine learning (SVR) • Position size based on correlation, volatility and sentiment.

		Both bots log information for the user. They may be safely terminated through the press of a button.
3	Detailed performance reports	Two reports (graphs) for each analyzed stock-strategy combination are displayed: • Strategy vs buy and hold for specific stock-strategy combination • Graph of calculated statistics for specific stock-strategy combination
4	GUI/Interface	Home page: • Successfully displays any relevant financial data • Allows the user to retrieve financial data on a stock of their choice through the press of a button. Bots page: • Successfully lets the user manage and customize bots Running page: • Only available when a bot is running, and lets the user manage the running bot.
5	Account system	Upon signing up: • Makes sure username is not a duplicate • Validates all API keys • Hashes passwords • Encrypt API keys Upon logging in: • The user only needs to enter username and password. These are hashed and verified. A user's bots are only visible to them. They are also able to delete any data they wish, along with their account (which automatically clears all related data).
6	Validation and user experience	For all relevant places (where databases are involved), the user cannot have more than one input window open at a time. This is done through disabling buttons. The program can be closed through either the 'x' button or the respective 'quit' button. It can also be closed through

		'forget account', which will display a confirmation window before proceeding. For all relevant places, capitalisation does not matter. The program is able to handle a lack of or invalid inputs anywhere. These scenarios are met with a relevant error window. Program will not crash under connection errors (bot will stop).

End user evaluation

Interview 1

Interviewer: Good day! First off, thank you for taking the time to try out my trading bot application and agreeing to provide feedback on your experience. Can you start by sharing your overall thoughts on the app?

End User: Hey! Sure. To start with the positive, I was honestly blown away by the aesthetics of the app. It's very pleasing to look at, has a clean design and feels very modern. I appreciated that a lot.

Interviewer: That's great to hear! I did spend a lot of time on the design aspect to make it visually appealing. Now, can you let me know if there were any aspects you weren't quite satisfied with?

End User: Yes, certainly. While the design was visually pleasing, I felt limited in terms of the customisation options. For example, I noticed that for the SMA based strategy, I couldn't decide the length of the SMAs. Similarly, for the momentum-based strategy, there's no option for me to define the look-back period. As someone who has specific trading strategies in mind, this kind of limitation can be a bit restrictive.

Interviewer: I see, that's something I've noticed as well. Customisation is definitely an area that needs improvement. Were there any other issues or aspects you found challenging?

End User: Yes, another thing that stood out was the user interface. While it's aesthetically pleasing, it didn't feel very intuitive in some places. For instance, there were entries where only specific time units were accepted, which wasn't immediately clear. Another example is when entering stocks; I had to use a comma as a separator. This wasn't very user-friendly, and I had to figure it out through trial and error.

Interviewer: I apologize for that inconvenience. I'll definitely look into improving the intuitiveness and making the user experience smoother. Thank you again for your feedback.

Interview 2

Interviewer: Hello and thank you for taking the time today to discuss your experience with my trading bot app. Let's jump right in. Can you tell me what you particularly liked about the app?

End User: Absolutely. One thing I genuinely appreciated is that the app runs entirely on free APIs. That's a significant advantage, especially for someone like me who's always on the lookout for cost-effective solutions. Another thing I found really beneficial was the ability to see the latest news for any stock just by typing it in. This feature is super convenient and keeps me updated without having to switch apps or platforms.

Interviewer: I'm glad to hear that you value that aspect. I wanted to keep the app as accessible as possible. Now, on the flip side, were there areas of the app you felt could use improvement or that didn't meet your expectations?

End User: Yes, I did notice a few things. One of the primary concerns I had was the speed. There were times, particularly when transitioning from the login screen to the home screen, where it felt a bit sluggish. The same went for when I first ran a bot; the initial startup was a tad slow.

Interviewer: I see, thank you for pointing that out. Slow interactions and loading screens can certainly affect user experience pretty significantly.

End User: Yes, another aspect I'd like to mention is the trading options and markets covered. I would love to have more variety in terms of what I can trade. Currently, it seems limited to just stocks. Additionally, it would be great if the app could branch out to other markets, not just the American one. Diversifying the offering could appeal to a broader audience, like myself, who are interested in international trading.

Interviewer: Thank you for that feedback. Expanding the trading options and market reach is definitely worth considering, and it's great to have your perspective on this.

Analysis of feedback

I believe the most concerning issue mentioned was that of (somewhat) limited customizability- as this was one of the initial goals alongside detailed reports and sentiment. Although this was met to some extent, the goal of 'transparency' wasn't (use of hardcoded values for lookback periods, SMA lengths, etc.). This is definitely a point of improvement for the future as the code architecture already makes it easy for this change to be realized (each strategy accepts their respective customisation values as parameters).

Intuition was another valid issue, as entries that only accepted certain time units, and entries for which the stocks needed to be entered using ', ' as a separator. This is bad as the

original intention of the app was to make the interface as user-friendly as possible. This could however easily be addressed through the use of documentation, an extra popup window of instructions accessible through a button, or perhaps a README file.

An issue I won't give much weight to is that of slow/extensive loading times. This is due to having to make do with free API keys and python modules. And while expanding from just the American market and stocks is a possibility, I believe this would've been very repetitive with a lot of boilerplate code.

Conclusion

There are some shortcomings I would like to address, that are perhaps not obvious to users. The sell method was very limited compared to the buy method (in an attempt to reduce risk), as it simply sells all open positions. This could perhaps be altered to maximize profit rather than reduce risk. There are also too many hardcoded values than I would like- including the order type (IOC, DAY, etc.), and the size of the dataframes being dealt with.

The current storage of encryption keys is also not very secure, as the key.key file is simply stored in the same directory. If this program were to be rolled out publicly, this should be addressed (perhaps through the use of external software).

In conclusion, the analysis of feedback has highlighted important areas for improvement in the app. The issues of limited customizability and lack of transparency are key concerns that can be addressed in the future, given the flexible code architecture. Intuition-related problems can be mitigated through better documentation and user-friendly features. While slow loading times were noted, they are primarily due to resource constraints. Notable areas for improvement include enhancing the sell method, reducing hardcoded values, and addressing security concerns regarding encryption keys for potential public rollout.

Appendices

Normalisation

UNF	1NF	2NF	3NF
id name username	id username password	id username password	id username password

password api_key api_secret_key av_key time_unit time_amount bot_type stocks strategies	user_id name api_key api_secret_key av_key user_id id time_unit time_amount bot_type stocks strategies	user_id name api_key api_secret_key av_key user_id id time_unit time_amount bot_type stocks strategies	user_id name api_key api_secret_key av_key user_id id time_unit time_amount bot_type stocks strategies

Helpful sources

Section	Type	URL	Description
GUI	Course	https://youtu.be/YXPyB4XeYLA?si=P6QUHynBsm82qOjq	Introduced simple syntax regarding the use of tkinter.
GUI	Guide	https://youtu.be/iM3kjbbKHQU?si=pGkRdUtimgIyvL8n	Introduced the syntax for customtkinter
NN	Course	https://youtu.be/GDMkkmkJigw?si=KqPQgNC1YTl5NV8o	Introduced algorithm trading for a beginner
NN	Guide	https://youtu.be/m1rY2J8ZlsY?si=ZDJzT4xCVnJPW6B8	Explained neural networks in the context of day trading.
NN	Guide	https://youtu.be/hpfQE0bTeA4?si=-9nkvphLZ_NGBQiS	Explains LSTM as type of ML strategy
API	Guide	https://youtu.be/Mv6c_9FqNx4?si=kR-lUsTfTuVobqT0	A guide to Alpaca's API and websockets
API	Guide	https://youtu.be/8Vg8GKWrV5M?si=xCl3nYq6_1qG8wG1	A simple introduction to Alpaca's python client

Technical skills used

Group	Model (including data model/structure)	Algorithms	Where used | *file (identifier)*	Page number
A	Complex data model in database (e.g. several interlinked tables)	Cross-table parameterised SQL User/CASE-generated DDL script	main.py, home.py, home_datagrab.py, bot.py	56, 67, 83, 86
	Hash tables, lists, stacks, queues, graphs, trees or structures of equivalent standard	Graph/Tree Traversal List operations Queue Operations Hashing	main.py (generate_salt, hash_password, verify) run.py (poll_data_queue, poll_stdout_queue)	56, 97
	Complex scientific / mathematical / robotics / control / business model	Advanced matrix operations Recursive algorithms Complex user-defined algorithms (eg optimisation, minimisation, scheduling, pattern matching) or equivalent difficulty	home.py (update_time) run.py (update_plot) hist_risk.py (calculate_position_size) live_risk.py (calculate_position_size)	74, 100, 114, 134
	Complex user-defined use of object oriented programming (OOP) model (e.g. classes, inheritance, composition, polymorphism and interfaces)	Dynamic generation of objects based on complex user-defined use of OOP model	hist_main.py (HistoricalBot) live_main.py (LiveBot)	107, 128
	Complex client-server model	Server-side scripting using request and response objects and server-side extensions for a complex client-server model	popup.py (get_overview, get_news) hist_datagrab.py (request_data, request_all_data)	86, 105,

		Calling parameterised Web service APIs and parsing JSON/XML to service a complex client-server model	hist_risk.py (get_correlation_data, get_sentiment_data)	111
	Files(s) organized for direct access		Any config and datagrab files	-
B	Simple data model in database (eg two or three interlinked tables)	Simple user defined algorithms (eg a range of mathematical/statistical calculations)	home.py (get_name) bot.py (add_record) Any strategy file (calculate_statistics)	73, 96 -
	Text files File(s) organized for sequential access	Writing and reading from files	home_datagrab.py (home_data) bot.py (run_record)	84, 97
	Dictionaries Records		home_datagrab.py (home_data)	84
	Simple OOP model	Generation of objects based on simple OOP model	main.py (App, LoginApp) home.py (StartPage) bot.py (PageOne) run.py(PageTwo) task.py (Task)	57, 61, 87, 98, 103
	Simple client-server model	Server-side scripting using request and response objects and server-side extensions for a simple client-server model Calling Web service APIs and parsing JSON/ XML to service a simple client-server model	main.py (validate_alpha_vantage_key) home_datagrab.py (home_data)	61, 84
C	Single-dimensional arrays		hist_config.py (config) live_config.py (config)	104, 123
	Appropriate choice of simple data types	Simple mathematical calculations (e.g. average)	hist_config.py (config)	104

	Single table database	Non-SQL table access	Any strategy file	-