Centre for Professional and Continuing Education

MH6803: Graded Group Project Submission

Design a Stock Portfolio Tracker using Python

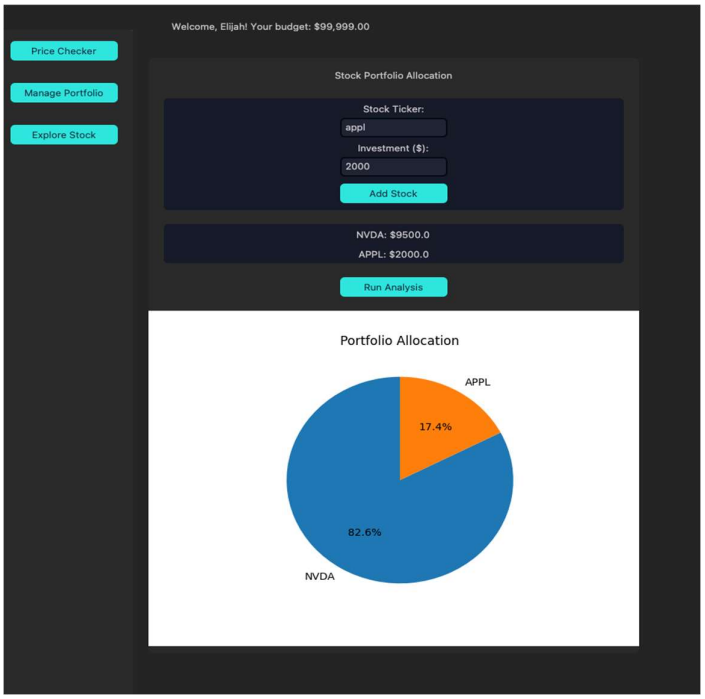
Submitted by: Team 4

Group Members: Daniel Lim (Leader), Kai Elijah Seah, Mark Joseph Fabre, Jes Bee Lian Lim

Table of Contents

Topic Description:

Our selected topic for this project is, “Design a Stock Portfolio Tracker using Python”. For this project, we used Tkinter to design and implement our basic graphical user interfaces. Tkinter helped us to incorporate widgets and event-driven programming principles to enhance user interaction and experience.



Sample of portfolio allocation output

We decided on this topic because the features that we wanted to implement in our program would challenge us to put into practice what we have learned over the course of the MH6803 module.

The main features of this program included:

1. User Interface & Input Handling
2. MarketStack API Integration:
3. Portfolio Management & Database Integration
4. Visualization & Historical Data Analysis

The program design, programming codes, and final implementation of the user interface allowed us to comprehensively experiment, explore, and troubleshoot the various python codes that were used in the delivery of the final product.

Functions to Implement for each Feature:

1. User Interface and Input Handling:
Design the main Tkinter window, handling user inputs (name, budget), and managing the overall layout.
Functions:
- def create_widgets(self): Create and organize all widgets (labels, buttons, entries).
- def submit_details(self): Handle user input validation and display a welcome message.
- def reset_inputs(self): Reset the input fields.

2. MarketStack API Integration:
Integrating the MarketStack API to search for stock prices and retrieve stock information.
Functions:
- def get_stock_price(self, symbol: str): Fetch the current price of a stock using MarketStack API.
- def search_stock(self, keyword: str): Search for stock details based on a keyword.
- def filter_assets(self, asset_type: str): Filter assets based on type (indices, options, funds).

3. Portfolio Management & Database Integration:
Adding, editing, and storing portfolio data in a database.
Functions:
- def add_asset_to_portfolio(self, symbol: str, category: str, qty: int, price: float): Add an asset to the portfolio.
- def save_portfolio_to_db(self): Save portfolio data to a SQL database.
- def view_portfolio(self): Display the portfolio list and allow for edits (add/subtract).

4. Visualization & Historical Data Analysis:
Generating visualizations (pie charts, line graphs, candlestick graphs) and handling historical data retrieval.
Functions:
- def generate_pie_chart(self): Display a pie chart of assets in the portfolio.
- def get_historical_data(self, symbol: str, start_date: str, end_date: str): Fetch historical stock data.
- def plot_line_graph(self, data): Plot a line graph of historical data

Algorithm Design used in the Program

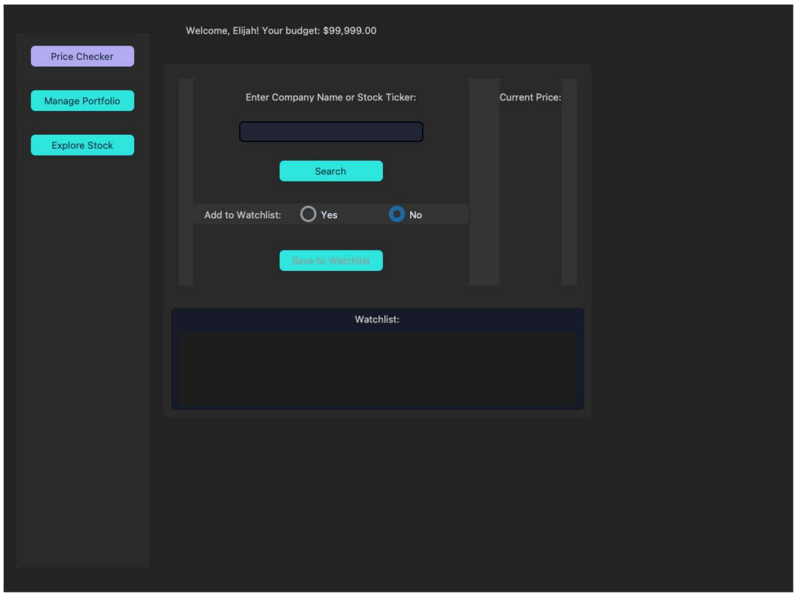
1. Key Features: Your Portfolio, Your Way!

"Add & Track": You can easily add your assets (stocks, crypto, whatever!) to a personal database. Imagine it as your digital filing cabinet for investments.

"See It All": A clean, separate window pops up showing your entire portfolio – asset names, quantities, and purchase prices, neatly organized.

"Make Changes": Need to update your holdings? No problem! You can modify your asset quantities and prices right within the portfolio view.

"Error-Proof": We've built in checks to handle incorrect inputs, so you won't accidentally break anything.



Screenshot showing User Interface to enter Company name or Stock Ticker

2. The Road to Portfolio Perfection

"Database Dilemmas": Getting Tkinter to play nice with SQLite took some tweaking. We had to ensure data flowed smoothly between the GUI and the database.
Resolution: Careful SQL queries and testing made it work!

"User-Friendly Frustrations": Making the GUI intuitive was tricky. We wanted it to be simple, not overwhelming.
Resolution: We focused on clear labels, logical layouts, and helpful messages.

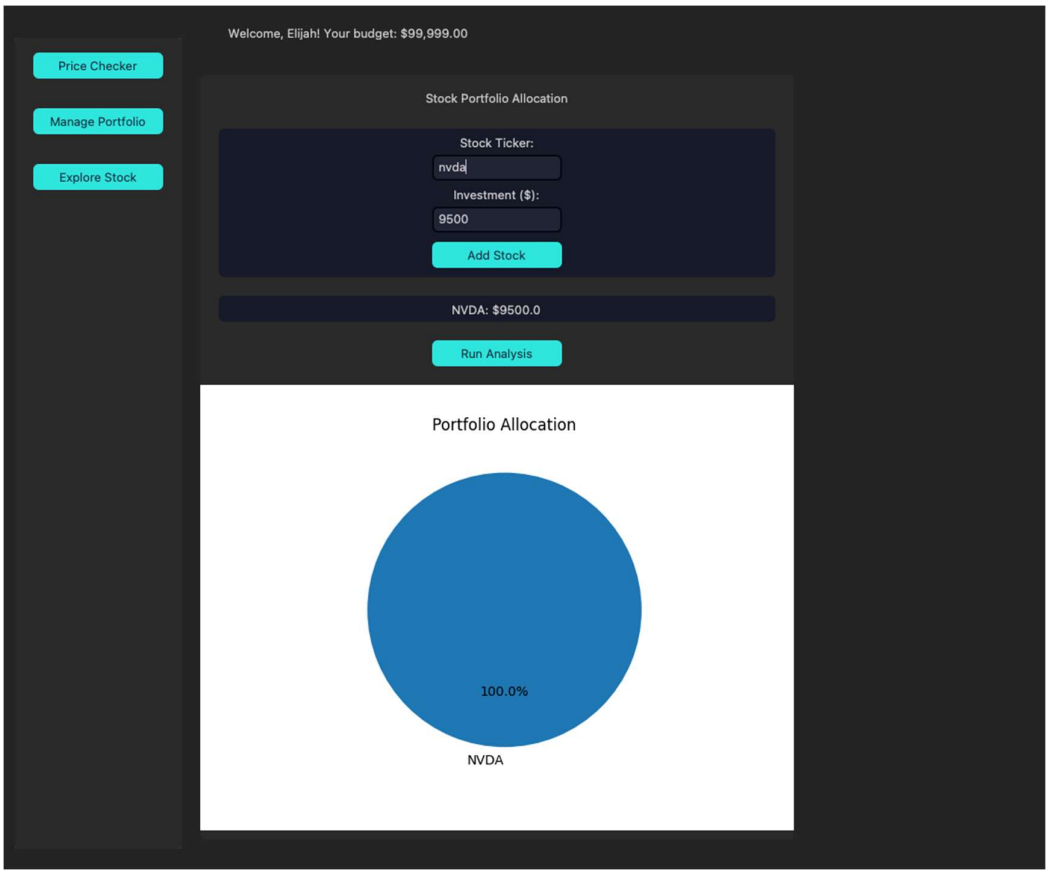
"Modification Madness": Allowing users to modify data within the treeview was a bit complex.
Resolution: We created a separate window to handle modifications, ensuring data integrity

3. Algorithm Design: The Logic Behind the Scenes

"Data Flow":
When you add an asset, the program grabs the input, validates it, and sends it to the SQLite database.
When you view your portfolio, the program fetches the data from the database and displays it in the treeview.
When you modify an asset, the program grabs the id of the selected asset, then allows for new data to be entered, then updates the database.

"Key Functions":
add_asset(): Handles adding new assets to the database.
view_portfolio(): Displays your portfolio in a new window.
modify_asset(): Handles the logic behind the modification window, and the update of the database.

"Error Handling":
try-except blocks are used to catch invalid input (e.g., non-numeric values) and display error messages.



Screenshot showing sample of how user can input stock ticker and investment amount, and pie chart of portfolio allocation is generated

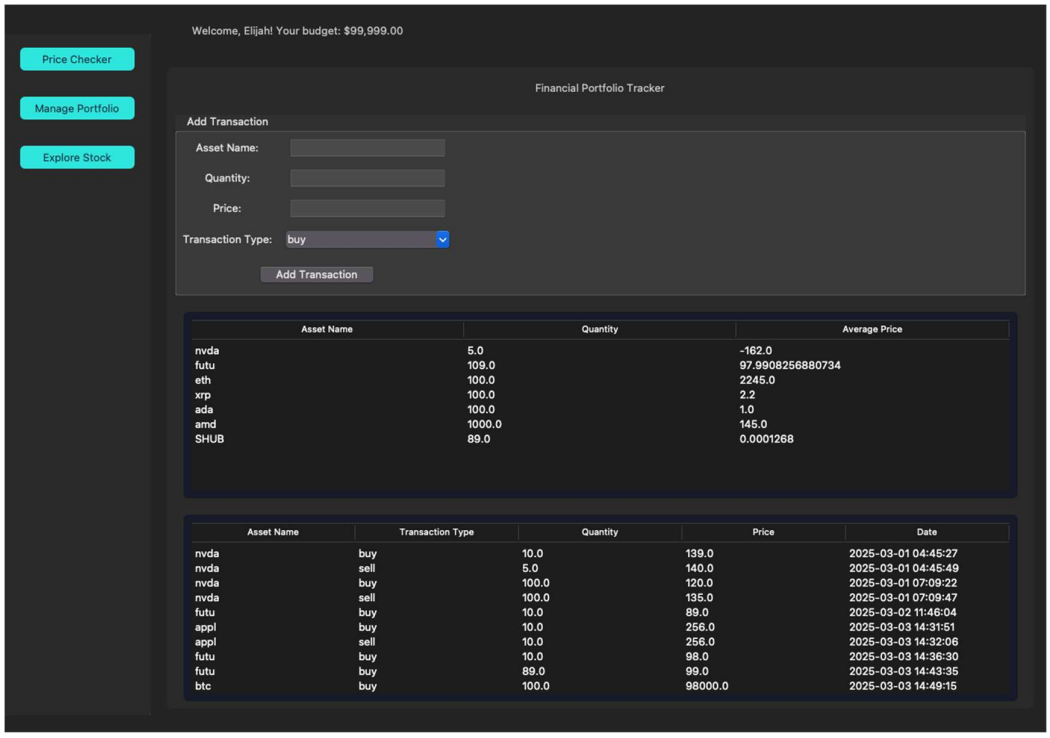
4. Screenshot Insights: A Visual Tour

"Main Window":
This screenshot shows the "Add Asset" section with input fields and the "View Portfolio" button.

"Portfolio Window":
This displays the treeview with your asset data. You'll see the columns for asset name, quantity, and purchase price. Also included is the modify asset button.

"Modification Window":
This shows the window that pops up when you select an asset to modify. It displays entry boxes to change the quantity, and price.

"Error Messages":
Screenshots of error messages (e.g., invalid input) demonstrate the program's robustness.



Screenshot showing how our program allows user to track portfolio and Calculate current holdings and preview transaction history below

Challenges Faced and Resolutions

During the course of submitting this project, our team faced several challenges pertaining to meeting coordination, juggling between the rigour of this project and our daily professional jobs, and also bridging the learning curve among team members.

Nevertheless, our team managed to resolve the majority of challenges faced, and was ultimately successful in compiling and aligning the respective codes to implement a neat and user friendly program that is able to meet our design objectives.

Challenges	Resolutions
Difficulty in collaborating using github	Team members will update each other on whatsapp on the codes progress and upload their codes on gdrive
Steep Learning curve for github terminal commands	Elijah compiled the codes and modify such that the main.py will be initialized on load. Additionally, create a centralized theme.py to standardized design across all files
Scheduling conflicts	Consensus on fortnightly updates via whatsapp chat and each member allocates time to work on the codes on weekends.
Segregated use of tkinter design	Peer to Peer discussion to align on the use of tkinter design output.
Steep Learning curve for members new to python	Elijah and Mark, being more experienced in python for programming, spent additional time for 1 on 1 sessions with the other members who were unfamiliar with python. They facilitated knowledge transfer and elevated the overall quality of work for the group
Errors/Bugs in code	Elijah and Mark spent additional time analyzing and cleaning up code to ensure all segments were aligned and program could run smoothly

Group Member Contribution Table:

Team Member	Percentage of contribution	Justification
Daniel Lim (Lead)	20	Visualization & Historical Data Analysis Generates a pie chart of portfolio asset distribution. Allow user to filter and categories assets Compiled and Prepared overall presentation & report
Mark Joseph Fabre	30	Portfolio Management & Database Integration Add assets to a SQLite database. Displays portfolio details in a new window Allow user to modify assets Work with Kai to test run codes and debug and rectify errors
Kai Elijah Seah	30	MarketStack API Integration with portfolio tracker Search for the name of the company and produce the ticker Fetches current stock prices using MarketStack API Ask the user if they want to add the searched results to the watchlist (create a watchlist.py) Compilation of team's code main.py of the program Create dashboard.py to fit the each of function within the dashboard using dashboard.py Create a theme.py file to ensure styling are standardised Worked with Mark to test run codes and debug and rectify errors
Jes Beelian	20	UI and Input Handling Collects user name and budget. Displays the main portfolio view Allows resetting input fields