Hong Kong Metropolitan University

COMP2090SEF

Data Structures, Algorithms And Problem Solving

Submittion Date : 18/4/2025

We declare that:

(i) all members of the group have read and checked that all parts of the project (including proposal, code programs, and reports), irrespective of whether they are contributed by individual member or all members as a group, here submitted is original except for source material explicitly acknowledged;

(ii) the project, in parts or in whole, has not been submitted for more than one purpose without declaration;

(iii) we are aware of the University's policy and regulations on honesty in academic work and understand the possible consequence when breaching such policy and regulations;

(iv) we confirm that we have declared in the report about the usage of AI and other generative models, including but not limited to ChatGPT, LLaMA, Gemini, Mistral, and Stable Diffusion, and complied with the instructions provided by HKMU; and

(v) we are aware that all members of the group should be held responsible and liable to disciplinary actions, irrespective of whether he/she has signed the declaration and whether he/she has contributed, directly or indirectly, to the problematic contents.

We confirm that we have read through and understood the project requirements. We understand that failure to comply with the project requirements will result in score deduction.

NAME	**SID**
LOK Kai Chun	13877806
MOHAMED IMRAN Mohamed Maheen	13766356
LIMBU Yunchho	13880269
GURUNG Albin	13872558
ROHAN Roka	13511636

Stock Monitoring & Investing Simulator

Log in page and main page

Core Functions

1 Login/Sign-Up Page

The login page allows users to access their portfolio by entering their username and password. For new users, there is a sign-up form where they can create an account by providing a username, password, confirming the password, and setting an initial balance.

Features:

- **Login:** Verifies the username and password against a database to grant access to the user's portfolio.
- **Sign-Up:** Registers a new user, validates the provided information, and creates a user account with a starting balance and empty stock portfolio.

The code employs SQLite for storing user credentials and pickle for serializing user portfolios. Form validation ensures data integrity (e.g., password confirmation, username length limits, and numeric balance check).

2 Main Menu Page

The main menu serves as the central hub, guiding the user through different functionalities:

- **Stock Simulation:** Takes users to the stock simulator page where they can simulate stock trades.
- **Stock Monitoring:** Opens a page that allows users to monitor real-time stock prices and trends.
- **View Portfolio:** Provides users access to view their portfolio with details on stock holdings and transactions.
- **Logout:** Allows users to safely log out of the application.

In addition, the main menu includes query buttons designed to guide users on how to use various features of the system. These buttons open pop-up windows with instructions for:

- How to Buy a Stock?
- How to Sell a Stock?
- How to Monitor Stocks?
- How to Check My Purchases and Transactions?

These interactive help windows are useful for users who are unfamiliar with the system's functionality, ensuring that they can easily navigate the application.

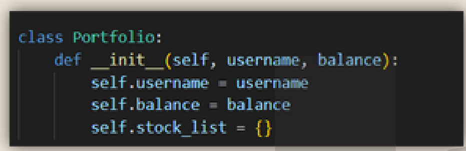
Example:
When the "How to Buy?" button is clicked, the user is provided with step-by-step instructions on purchasing stocks within the simulator. Similarly, the "How to Sell?" button outlines the procedure for selling stocks from their portfolio.

3. Use of Class and Data Structure

The core component of the login/signup system is the Portfolio class, which represents a user's stock portfolio. This class has the following attributes:

- **username:** A string that uniquely identifies the user.
- **balance:** An integer representing the user's initial balance.
- **stock list:** A dictionary used to store the stocks owned by the user. This is designed to track stock symbols and quantities.

Code Snippet

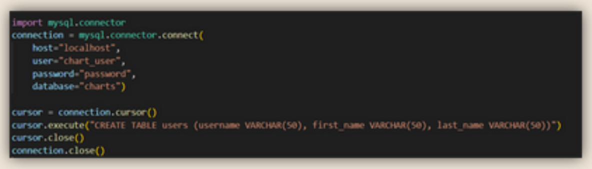


```python
class Portfolio:
    def __init__(self, username, balance):
        self.username = username
        self.balance = balance
        self.stock_list = {}
```

The use of SQLite allows efficient storage and retrieval of user data, including username, password, balance, and serialized stock portfolio. The pickle module is used to serialize the user's portfolio object, allowing us to store it in the local file system as a .pkl file.

A typical interaction with the database involves queries to check if a user exists (SELECT statement) and if a username already exists (SELECT to check uniqueness) to prevent duplicate entries. The database structure is simple yet effective for this project, comprising a table Stocks with columns for the username, password, balance, and a reference to the serialized portfolio file.

Code Snippet



```python
import mysql.connector
connection = mysql.connector.connect(
    host="localhost",
    user="chart_user",
    password="password",
    database="charts")

cursor = connection.cursor()
cursor.execute("CREATE TABLE users (username VARCHAR(50), first_name VARCHAR(50), last_name VARCHAR(50))")
cursor.close()
connection.close()
```

This code snippet demonstrates the connection to a MySQL database, where user information can be stored and managed efficiently, ensuring the scalability of the project.

4.External Resources Used

- **Tkinter:** Tkinter is the primary GUI toolkit used to create the user interface. It allows for the creation of entry fields, buttons, labels, and pop-up windows. Customization was achieved using the ttk (Themed Tkinter) module, enabling better-looking buttons, labels, and input fields.
- **SQLite:** SQLite is used for local storage of user data. It provides a lightweight, serverless database engine for storing user information securely on the user's local machine.
- **Pickle:** The pickle module is used to serialize the portfolio data (which includes stock information) into a binary format and store it as a file on the user's computer. This allows for easy persistence of user data between sessions.

Other Libraries:

- **sys and subprocess:** These modules were used for running external Python scripts when the user clicks on "Stock Simulator" and "Stock Monitoring."
- **messagebox:** Used for displaying alerts for user input validation and login feedback.
- **os:** Used to manage directories for storing serialized portfolio data.

5. Self-Reflection and Future Improvements

While the application functions as intended, there are always potential improvements that could be made:

1. **Enhanced User Interface (UI):** The current UI is functional but could be improved with more advanced design patterns or frameworks like PyQt for a more professional appearance.
2. **Additional Features:** Future iterations of the system could include advanced stock analysis tools, such as moving averages or predictive analytics, to further assist users in their investment decisions.
3. **Database Integration:** Moving from SQLite to MySQL would enhance performance and scalability, especially if the application grows to accommodate many users.

Additionally, implementing a feedback loop where users can rate their experience or provide input could help identify areas for future improvement.

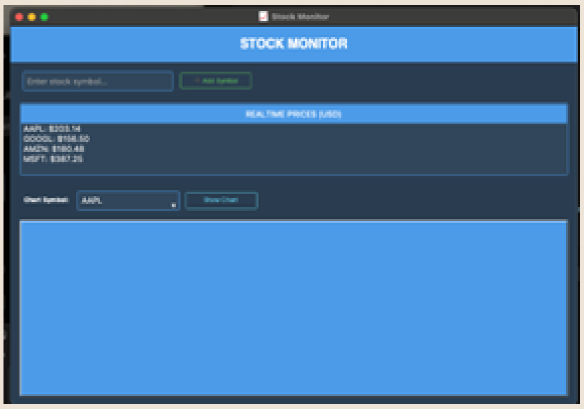
6. Conclusion

The Stock Market Monitoring and Simulation System provides a comprehensive platform for users to simulate stock purchases, monitor real-time stock data, and manage their virtual portfolios. By combining real-time data with an intuitive interface, users can interact with the stock market in a simulated environment. The system successfully implements core features such as login and sign-up, stock simulation, and portfolio management while leveraging external resources like Yahoo Finance and MySQL for data fetching and storage. This system offers a powerful learning tool for users interested in stock trading and investment simulation.

Stock Monitoring System

Project Overview:

The Stock Monitor App is a Python GUI application that allows users to track stock prices in real-time and visualize historical price data. Built with Tkinter and ttkbootstrap, it provides an intuitive interface for monitoring stock market information



```
AAPL: $203.14
GOOGL: $156.50
AMZN: $180.48
MSFT: $387.25
```

Technical Implementation:

- **GUI Framework:** ttkbootstrap (Themed Tkinter) for modern UI components
- **Data Fetching:** yfinance library for stock market data
- **Data Visualization:** Matplotlib for charting with Tkinter integration
- **UI Techniques:**
- Treeview widget for tabular price display
- Combobox for symbol selection
- Dynamic chart rendering with FigureCanvasTkAgg
- Placeholder text handling in Entry widgets
- Alternating row colors in Treeview

Usage of Classes and Data Structures

Object-Oriented Design:

Single StockMonitorApp class encapsulates all functionality

- _init__: Initializes UI and default symbols
- setup_ui: Creates all GUI components
- add_symbol: Handles new symbol input
- update_prices: Fetches and displays current prices
- plot_chart: Generates historical price charts

Data Structures:

- List: self.symbols stores tracked stock symbols
- Treeview: Displays prices in tabular format
- Matplotlib Figure: Stores and renders chart data
- yfinance Ticker: Interfaces with Yahoo Finance API

External Resources Declaration

Libraries Used:

- tkinter: Core GUI framework
- ttkbootstrap: Modern UI theming
- yfinance: Yahoo Finance API wrapper
- matplotlib: Data visualization
- FigureCanvasTkAgg: Matplotlib-Tkinter integration
-

Self-reflection: Weaknesses and Future Improvements

Current Limitations:

- Basic error handling for API failures
- Limited chart customization options
- No user authentication or personalization
- Single-threaded UI may freeze during data fetching

Potential Enhancements:

1)Data Features:
  a)Add more time periods (1 week, 3 months, 1 year)
  b)Include volume data in charts
  c)Technical indicators (SMA, RSI)

2)UI Improvements:
  a)Dark/light theme toggle
  b)Responsive layout for different window sizes
  c)Animated loading indicators
  d)Symbol removal functionality

3)Error Handling:
  a)Better validation for symbol input
  b)Retry mechanism for API failures
  c)Connection status indicator

Stock Simulator

Overview

The Stock Simulator system is a application developed with python GUI. It allows users to invest on the stock market with a virual capital and pratice their investing skill. It is built with Tkinter to provide an proper graphic interface for doing a simulation.

Description of Core Functions and Summary of Techniques

Following is a Python stock simulator application using Tkinter as the graphical user interface, and some data manipulation and plotting libraries. Some of the most featureful featured of the application are:

- **Stock Price Retrieval:** The program employs the yfinance library to fetch the current stock price, historical stock prices, and other instrument information.
- **Portfolio Management:** The user purchases and sells shares, and the program maintains a portfolio to record the purchased shares, purchase price, and present market price.
- **Graphical Representation:** Matplotlib is employed in the program to display volatility of stock price and change in balance with time.
- **Data Persistence:** The application stores the user's balance and portfolio in a JSON file so that when the application is executed again, it can load data.
- **User Interface:** The GUI is divided into frames to show stocks, show portfolios, and balance graphs so that the valuable information is shown in a well-structured way

Techniques Used

- **Object-Oriented Programming (OOP):** The program is encapsulated in a StockApp class, making it modular and reusable.
- **Event Handling:** The event-based programming paradigm of Tkinter is utilized, which supports dynamic user events (e.g., clicking on a button).
- **Data Structures:** The portfolio is maintained in a dictionary with stock tickers as keys and their corresponding data (shares, price bought) as values.
- Use of Class and Data Structure

Class Usage

Every application functionality is coded within the StockApp class. There are procedures for handling user input, updating the GUI, and updating stock information, all contained here. Easy to develop and keep up with just one class.

Data Structures

❖ Portfolio Dictionary:
  ➢ Key: Stock ticker (string)
  ➢ Value: Dictionary of:
    ■ shares: Number of shares (integer)
    ■ price bought: Buying price per share (float)

❖ History and Balance: There is balance in a float, and there are updates to plot in a list.

Declaration of External Resources

A few external libraries the program uses are:

- **Tkinter:** Python standard library for making a graphical user interface
- **yfinance:** Library to retrieve Yahoo Finance data, to get the current stock price.
- **Matplotlib:** Library for Python's static, animated, and interactive plots.
- Other than that, the application uses subprocess and json modules as they are default modules in Python used to deal with processes and process JSON information, respectively.

Self-Assessment of Possible Weaknesses and Future Development

Weaknesses

- **Error Handling:** The code can be enhanced with good error handling, particularly while retrieving data from Yahoo Finance or JSON parsing. It only displays a message box but never provides feedback in the form of information.
- **User Experience:** The user interface might be confusing for the novice. It would be better with the inclusion of tooltips or a help page to guide the users.
- **Performance:** Fetching the latest price of all the stocks in the portfolio on updates would be costly for the application, especially when handling large portfolios. It can be optimized by caching prices or by capping rates of updates.

Future Improvements

- **Improved Visualization:** Offering more visualization features, i.e., candlestick charts or performance vs. comparison, might assist users in getting better insights.
- **User Authentication:** Addition of user accounts can add the aspect of multiple users in such a way that each user will have their own portfolios.
- **Additional Functionality:** Addition of news feeds, price alert, or predictive analytics can add extra user functionality and activity.

Portfolio Monitoring & Analysing system

Core Functions

Viewing portfolio, transactions and profit/loss

The viewing portfolio page allows users to view how much profit or loss they are gaining in real time. The transactions allow users to see all their buys and sells. The view of profit or loss shows how much total profit or loss they gained after selling stocks.

Features:

- **View Portfolio:** shows what the user has bought, the quantity of the stocks bought and the price point which it was bought. Shows real-time updates on the profit or loss they are gaining from each individual stock.
- **View Transactions:** allow users to check if a stock they bought went through and saved on their account
- **View P/L:** displays profit and loss gained when selling each individual stock, displays total profit and losses.

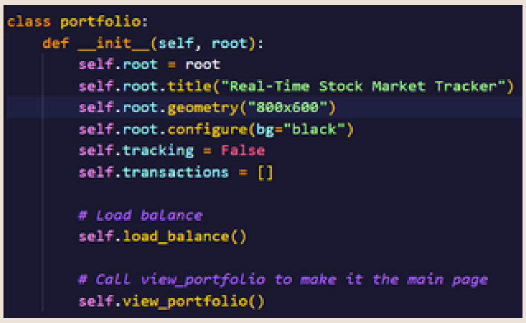
This part of the program makes usage of Json to store Balance, Transactions, Portfolio and Profit/Loss. The data stores multiple users' info and ensures no overlapping occurs.

stock simulation (buy and sell)

The portfolio page cannot work by itself as it needs data for viewing all the information. Collaboration was required so that when users buy or sell an item it would be stored into the Json files respectively.

- Buying Stock: stores the price point, quantity and ticker into the portfolio and transaction's part
- Selling Stock: deducts the stock from a ticker based on how many the of the stock the user sells and if its 0 removes the stock from portfolio. Writes the amount of stock sold and price point in transactions and deducts the selling price from the bought price to get the profit and loss from the stock.

Use of class, data structures and algorithms



```python
class portfolio:
    def __init__(self, root):
        self.root = root
        self.root.title("Real-Time Stock Market Tracker")
        self.root.geometry("800x600")
        self.root.configure(bg="black")
        self.tracking = False
        self.transactions = []

        # Load balance
        self.load_balance()

        # Call view_portfolio to make it the main page
        self.view_portfolio()
```

The core of the program which opens up the GUI window, loads the user's balance and calls the view portfolio function to show user's portfolio upon loading.

The class has the following attributes

- **Root:** Configure the main window to add GUI components like text widgets, labels and buttons
- **Self:** access and modify attributes, call attributes, maintain state across models

Data structures and algorithms used

The use of Json to store data allowing for easy storing, retrieving, removing and editing of data in the file. The program uses multiple data structures and algorithms across both the buy/sell of stock simulation and the viewing of personal portfolio details. Including:

Data structures

- Dictionaries to store key value pairs
- Lists to store numerous items such as portfolio entries, transactions, profit/Loss
- Text Widgets to display textual data in GUI
- Threads to handle real-time stock tracking

Algorithms

- Profit/loss calculations
- Real-time balance updates
- Portfolio update
- Data filtering and updating
- Error Handling

These data structures and algorithms are essential for the functionality of the stock simulator, enabling features like real- time updates, portfolio management and transaction tracking.

Potential improvements for the future

- **Code organization:**Separate GUI logic from business logic for better maintainability
- **Thread safety:**Ensure thread-safe operations when accessing shared resources like files.
- **Enhanced features:**Include more detailed analytics (eg. Historical performance)
- **Testing:**Add unit tests to ensure the correctness of methods like get_current_price

Conclusion:

The view portfolio, view transactions, view profit/Loss provides a comprehensive real time updates on stock prices and balance updates. Allowing users to track their performance and learn how to properly make trades and investments.
The portfolio is a stock market tracker and portfolio manager. It uses Tkinter for the GUI, yfinance for real-time stock data, and JSON files for data persistence. While it is functional and user-friendly, there is room for improvement in terms of code organization and additional features.

References

- ttkbootstrap - ttkbootstrap. (n.d.). https://ttkbootstrap.readthedocs.io/
- Using Matplotlib — Matplotlib 3.10.1 documentation. (n.d.). https://matplotlib.org/stable/users/index.html
- yfinance. (2025, March 21). PyPI. https://pypi.org/project/yfinance/
- tkinter — Python interface to Tcl/Tk. (n.d.). PythonDocumentation. https://docs.python.org/3/library/tkinter.html
- Amos, D. (2024, December 7). Python GUI programming withTkinter. https://realpython.com/python-gui-tkinter/

Work allocation

LOK Kai Chun-----20%
MOHAMED IMRAN Mohamed Maheen-----20%
LIMBU Yunchho-----20%
GURUNG Albin-----20%
ROHAN Roka-----20%

THE
END