VITyarthi Project

STOCKS CALCULATOR

Submitted By Indrajith Sen
25BET10015

Indrajith Sen
25BET10015

VITyarthi - Build Your Own Project

Stocks Calculator

Problem Statement

Creating a Stock Profit/Loss Calculator

Project Objective

The objective of this project is to address the practical and meaningful problem of helping individuals accurately determine the financial outcome of stock market transactions by developing a comprehensive and reliable stock profit/ loss calculator application. This project aims to design a technically sound solution that transforms real-world trading inputs—such as buy price, sell price, share quantity, and brokerage charges—into meaningful financial insights through systematic computation, percentage evaluation, and break-even analysis. The solution is implemented using the tools, programming methods, and concepts learned in the course, including modular function design, object-oriented programming, input validation techniques, exception handling, file storage through CSV integration, and logging for system tracking.

Functional Requirement

The scope of this project includes developing an advanced stock profit/loss calculator that allows users to perform multiple stock transaction evaluations, store historical results, and retrieve previously recorded data through a structured and interactive application flow.

The system includes three major functional modules.
The first module is the Input and Validation Module, which allows users to enter transaction details such as buy price, sell price, quantity of shares, and optional brokerage charges, while ensuring data accuracy through continuous input validation and error handling.
The second module is the Calculation and Processing Module, which performs core computations including total cost, total value, profit or loss amount,

percentage change, and break-even price using systematic arithmetic logic within an object-oriented structure.
The third module is the Output and Reporting Module, which displays a detailed summary of the transaction results to the user and stores each completed calculation in a CSV file, while also providing an option to view previously saved transaction history.

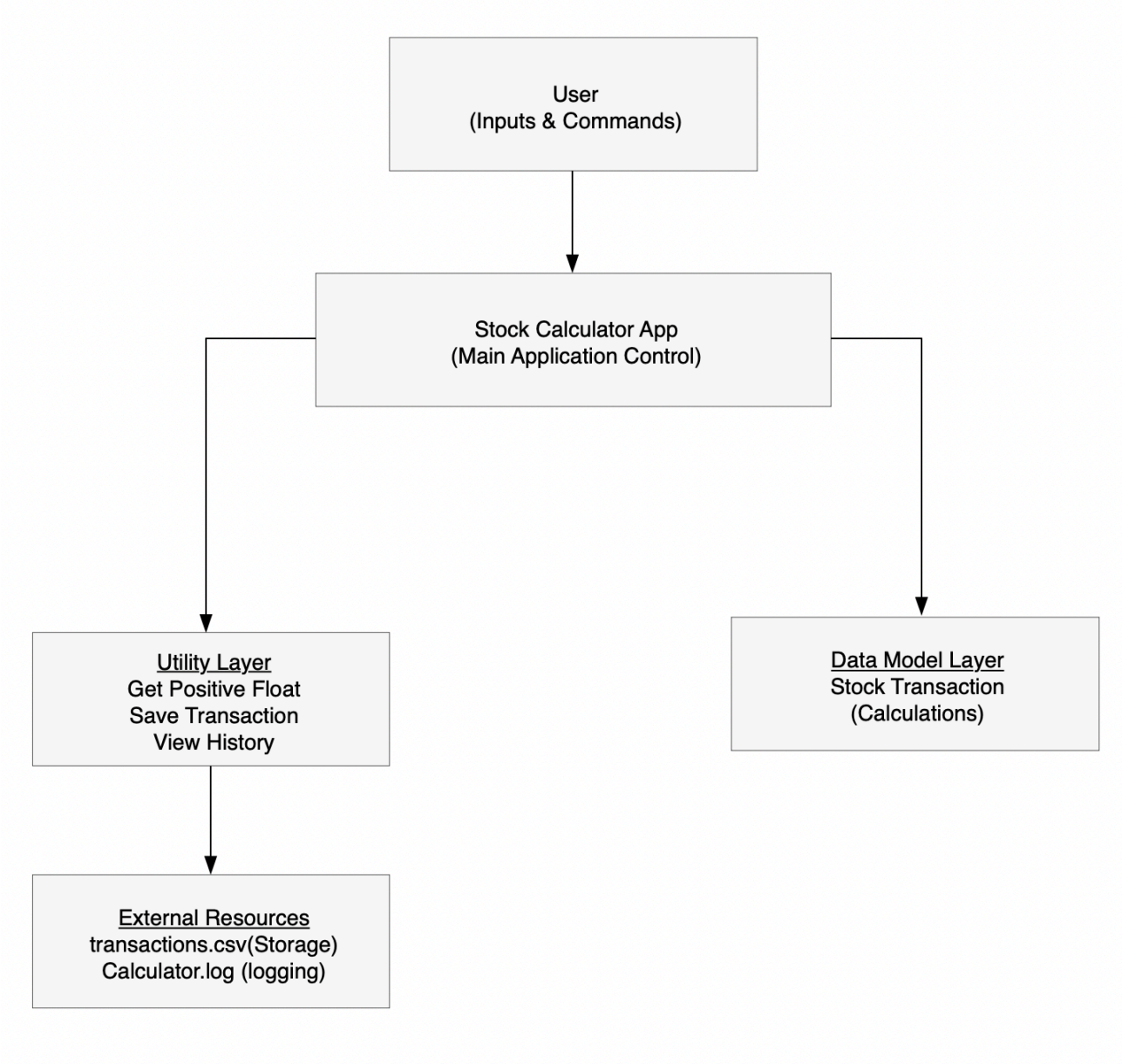
The system follows a clear input/output structure, where the user provides trading data as input, the application processes it internally through the calculation engine, and the output is presented in the form of readable financial results, percentage changes, and stored records for future reference.

The application maintains a logical workflow in which the user navigates through a menu-driven interface to select an action, enters transaction details, receives validated feedback and computation results, and optionally views past records before returning to the main menu or exiting the system.

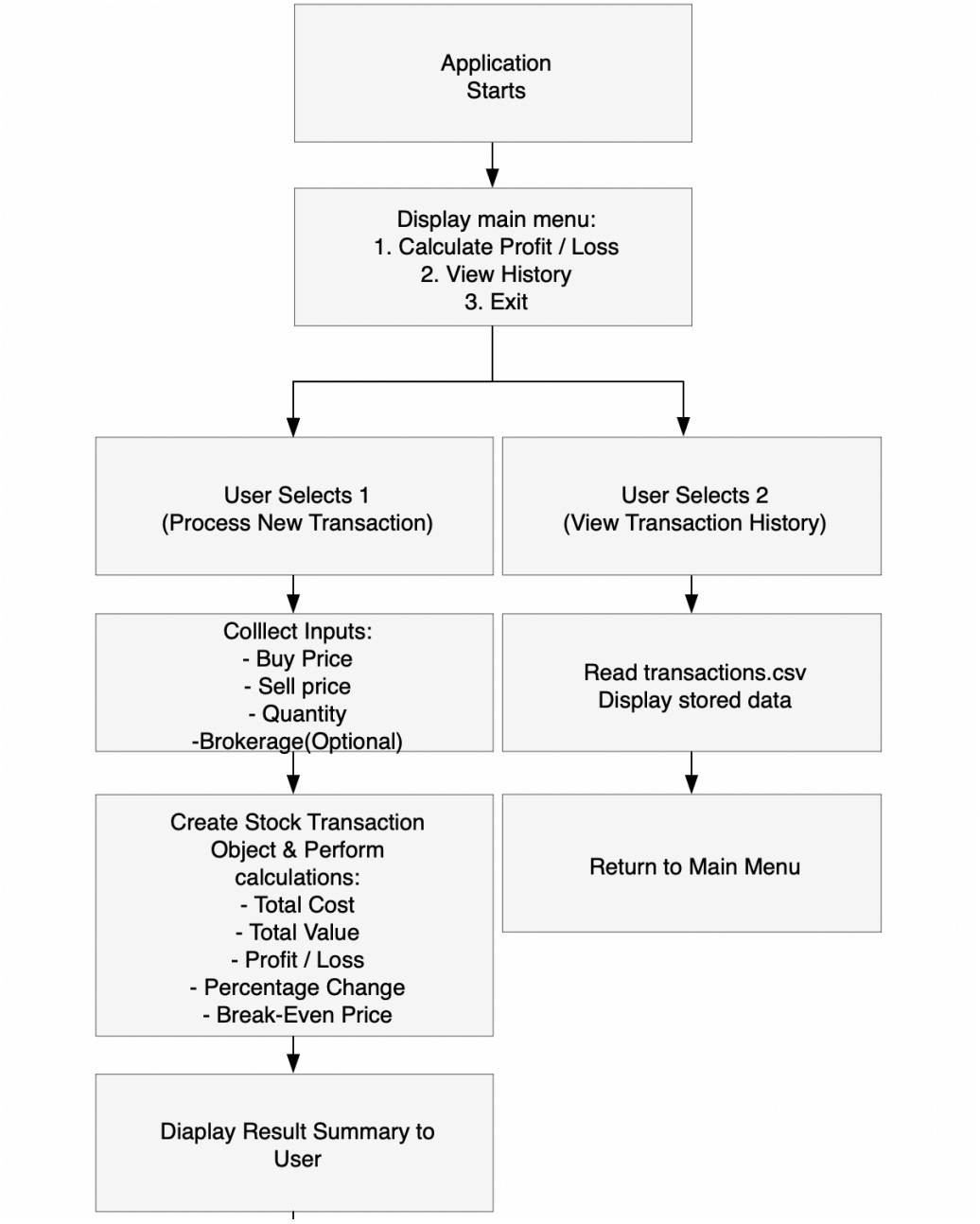
Non-Functional Requirement

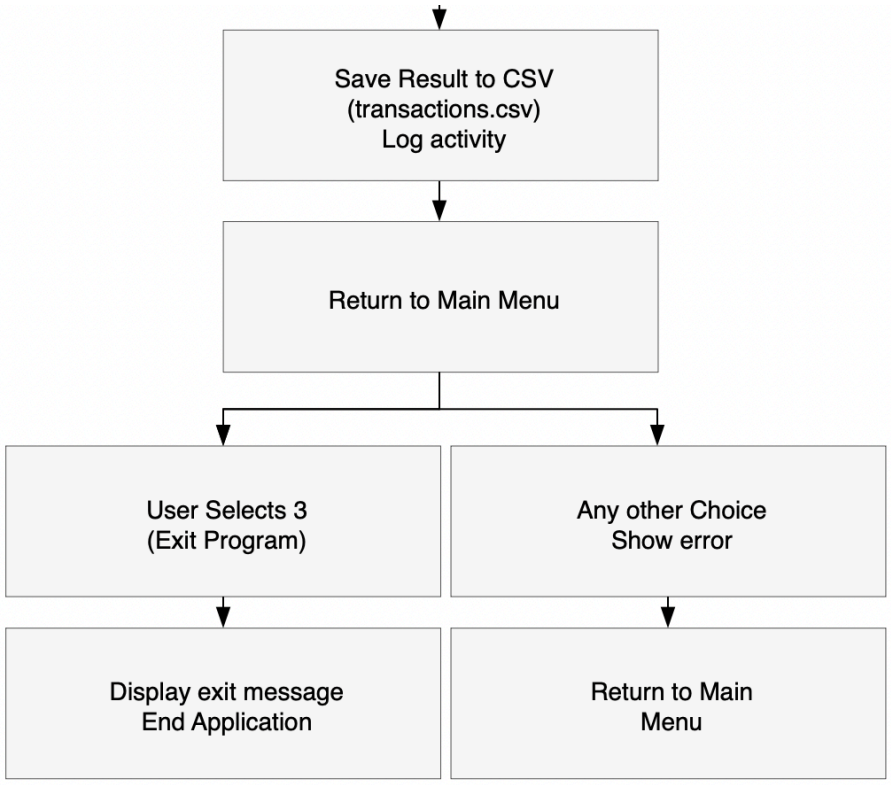
The non-functional requirements of this project specify that the Stock Profit/ Loss Calculator application is delivering optimal performance by processing user inputs and generating results without delay, ensuring a smooth and efficient experience. The system is maintaining high usability through a simple, intuitive, and user-friendly interface that enables users to input data easily and interpret outputs clearly. Reliability is ensured by producing consistent and accurate results while preventing system failures through proper handling of invalid or missing inputs. The application also support maintainability by using a modular and well-organized code structure, allowing future modifications, debugging, and enhancements to be carried out with minimal effort. Additionally, the error-handling strategy effectively identify incorrect inputs, prevent unexpected crashes, and provide meaningful feedback to guide the user. Finally, the system operates with resource efficiency, ensuring that it runs smoothly on standard computing environments without excessive memory or processing consumption.

System Architecture Diagram



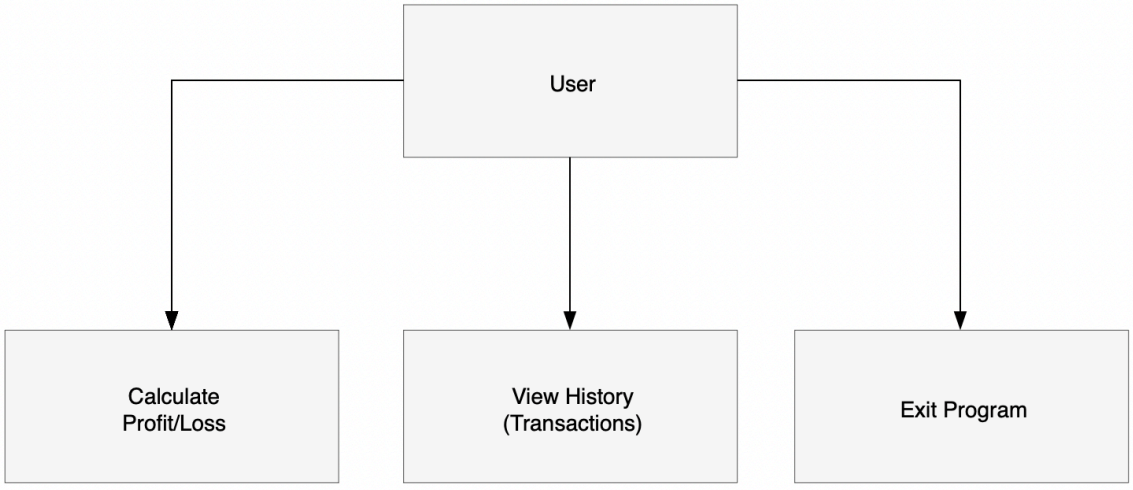
Process Flow or Workflow Diagram



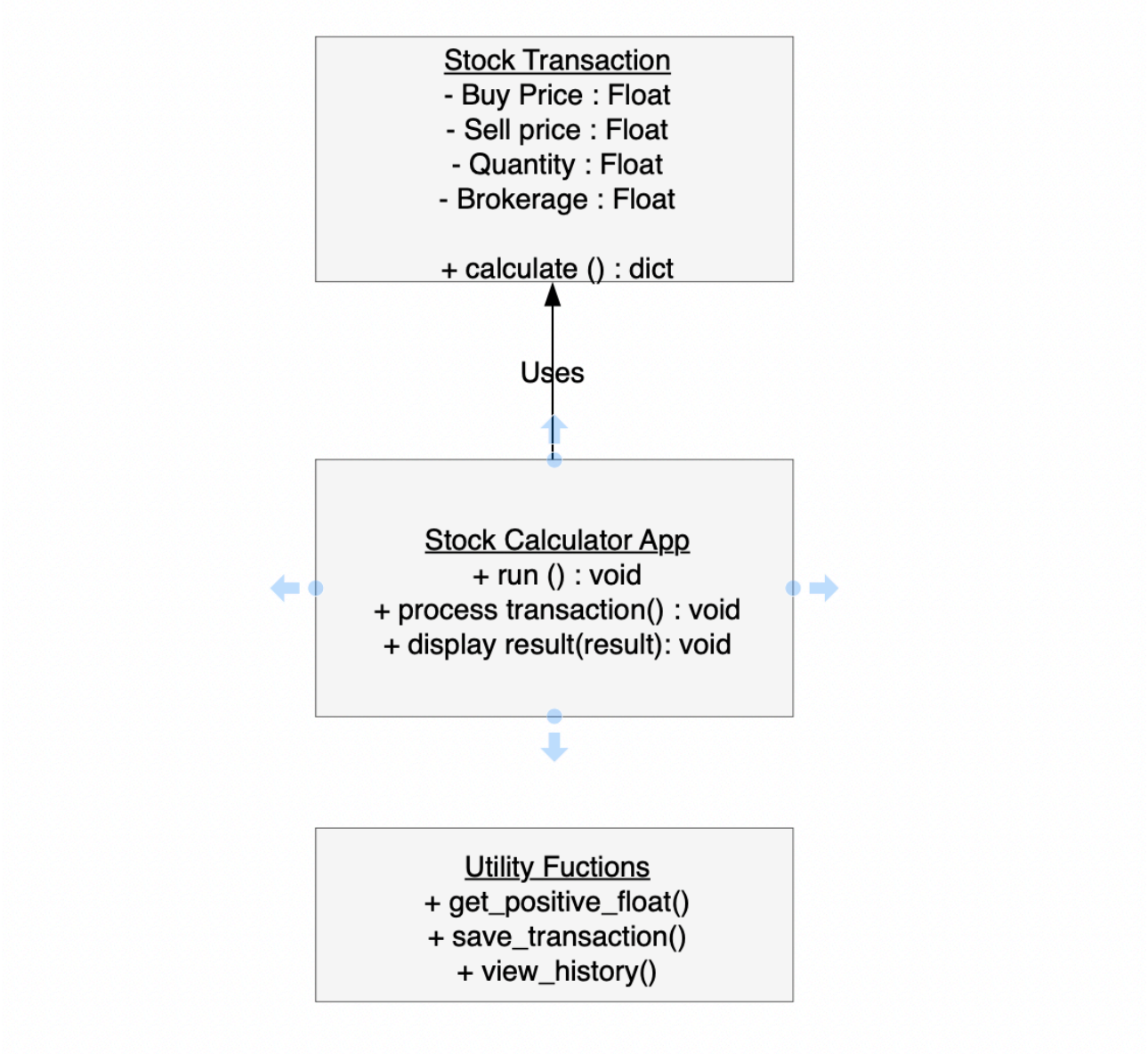


UML Diagrams

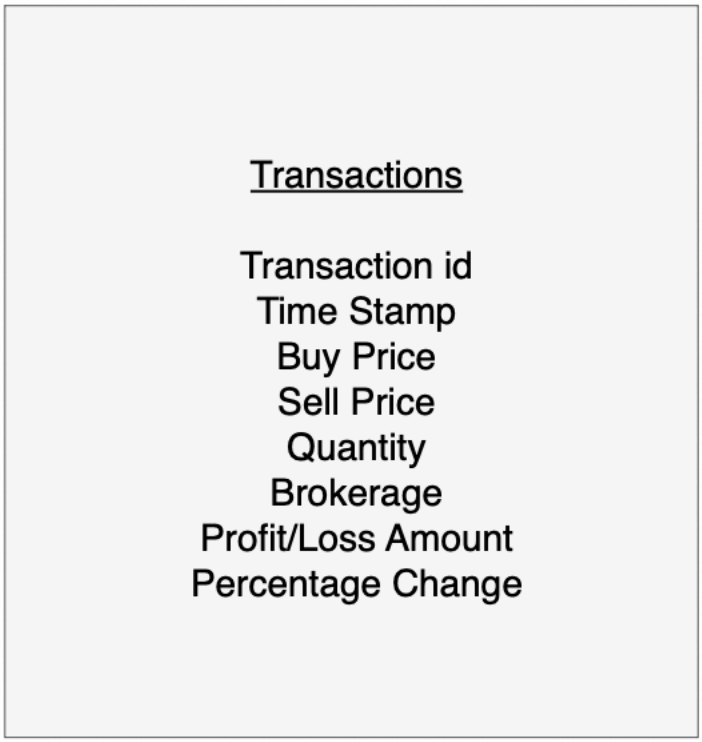
Case Diagram



Class Diagram



ER (Entity–Relationship) Diagram



Schema Design (for CSV-based storage)

Field Name	Data Type	Description
timestamp	STRING (YYYY-MM-DD)	Stores date and time of transaction
buy_price	FLOAT	Price per share at purchase
sell_price	FLOAT	Price per share at selling
quantity	INTEGER	Number of shares involved
brokerage	FLOAT	Total brokerage charges applied
profit_loss_amount	FLOAT	Net profit or loss calculated
percent_change	FLOAT	Gain or loss percentage

Design Decisions & Rationale

The development of the stock calculator application involved several key design decisions made to ensure clarity, maintainability, and practical usability. The program follows a combination of object-oriented and procedural programming, where the StockTransaction class is used to encapsulate the core calculation logic while supporting functions handle input collection, data storage, and viewing history. This approach avoids unnecessary complexity for a simple command-line application, while still allowing future scalability and reuse of the calculation model. The creation of the StockTransaction class with attributes such as buy price, sell price, quantity, and brokerage provides a clean separation of concerns, ensuring that all computation-related operations remain centralized, testable, and easy to maintain.

For data storage, the decision was made to use a CSV file instead of a database, as the application does not require relational data handling or complex queries. Using transactions.csv keeps the system lightweight, avoids additional dependencies, and makes the stored data easily accessible and editable. The logging mechanism was implemented using Python's logging module, configured to store logs in calculator.log, which enables tracking of system events and supports debugging without exposing internal details to the user. Input validation is handled through a dedicated function that continuously prompts the user until a positive numeric value is entered, thereby preventing incorrect calculations and improving overall user experience.

The user interface was designed as a simple menu-driven console interaction, allowing users to calculate profit or loss, view transaction history, or exit the application. This decision ensures ease of use, especially for beginners, and eliminates the need for graphical interface frameworks. The overall structure of the program is divided logically into sections such as logger setup, data model, utility functions, application controller, and the main execution block, which enhances readability and simplifies maintenance. Error handling is implemented using try-except blocks to manage invalid numerical input and missing history files, ensuring that the application continues running smoothly without crashing.

Finally, the system was designed with extensibility in mind, allowing future enhancements such as integration with a graphical interface, migration to a database l additional reporting features, or support for portfolio-level analysis. These decisions collectively ensure that the application remains simple yet robust, suitable for academic evaluation while being adaptable for real-world expansion.

Implementation Detail

The stock calculator application is implemented in Python and follows a modular structure to ensure clarity and ease of maintenance. The program begins with the setup of the logging configuration using the logging module, which records system events such as successful transactions and application actions into a file named calculator.log. This setup allows the application to maintain an execution trace without interfering with the user interface. The core logic of the system is implemented through the StockTransaction class, which acts as the data model responsible for handling all financial calculations. This class accepts input values including buy price, sell price, quantity, and brokerage, and generates computed results such as total cost, total value, profit or loss difference, percentage change, and break-even price using its calculate() method. The use of a class ensures that each transaction is processed as an independent object, leading to better organization and reusability.

The program also includes several utility functions that support secondary operations. The get_positive_float() function is used to collect and validate numeric input from the user, ensuring that only positive and valid values are accepted. This prevents runtime errors and maintains accurate calculations. The save_transaction() function writes processed transaction data into a CSV file named transactions.csv using Python's csv module, allowing persistent storage of transaction history in a structured format. Another function, view_history(), reads the stored data and displays it to the user, while handling missing files using exception handling to prevent application failure.

The StockCalculatorApp class manages the main workflow of the application through a menu-driven approach. Its run() method continuously displays options to the user such as performing a new calculation, viewing previous transactions, or exiting the program. When the user selects the calculation option, the process_transaction() method is executed, which collects the required input values, creates an instance of the StockTransaction class, and retrieves the computed results. The output is presented to the user through the display_results() method, which formats and prints values such as total cost, total value, profit or loss status, percentage change, and break-even point. Once the results are displayed, the program stores the transaction along with a timestamp using the datetime module, ensuring that each record is saved accurately.

Finally, the application is initiated through the main execution block using the condition if __name__ == "__main__":, which creates an instance of StockCalculatorApp and starts the program by calling the run() method. This structure allows the code to function both as a standalone script and be extendable for future imports or integration. Overall, the implementation combines modular programming practices, object-oriented structure, proper validation, persistent

storage, and logging to create a reliable and user-friendly stock profit and loss calculator.

Screenshots / Results

```
STOCK PROFIT/LOSS CALCULATOR
1. Calculate Profit/Loss
2. View Transaction History
3. Exit
Enter your choice: 1

ENTER TRANSACTION DETAILS
Buy price per share: ₹50000
Sell price per share: ₹76000
Number of shares: 100
Include brokerage? (yes/no): yes
Enter total brokerage: ₹3000

RESULT SUMMARY
Total Cost: ₹5003000.00
Total Value: ₹7597000.00
Profit: ₹2594000.00
Percentage Change: 51.85%
Break-Even Price: ₹50060.00

STOCK PROFIT/LOSS CALCULATOR
1. Calculate Profit/Loss
2. View Transaction History
3. Exit
Enter your choice: 3

Thank you for using the calculator!
```

Testing Approach

The testing approach for the stock profit and loss calculator focuses on ensuring that the system functions correctly, handles errors gracefully, and produces accurate results under different conditions. The application is tested using both functional and validation-based testing methods, beginning with unit-level verification of the StockTransaction class to confirm that calculations such as total cost, total value, percentage change, and break-even price are computed accurately for various input combinations including profit, loss, and no-gain scenarios. Input validation is tested by entering invalid values such as negative numbers, non-numeric characters, and empty inputs to verify that the get_positive_float() function consistently rejects incorrect data and prompts the user until valid input is provided. The workflow of the menu-driven interface is manually tested to ensure that all options—performing a calculation, viewing transaction history, and exiting the application—operate as expected without crashing or skipping steps.

The CSV storage functionality is tested by confirming that each completed transaction is correctly appended to the transactions.csv file with accurate formatting, and that the system can display previously stored data using the view_history() function. Additionally, error handling is evaluated by intentionally running the program without an existing CSV file to ensure that the system displays a user-friendly message rather than terminating unexpectedly. The logging mechanism is reviewed by checking entries written to calculator.log to verify that each transaction and important event is recorded appropriately. Boundary testing is performed using extreme input values such as very high quantities or zero brokerage to ensure that calculations remain stable. Overall, the testing approach confirms reliability, correctness, and robustness by combining manual testing, validation checks, edge-case execution, and storage and logging verification to ensure consistent system performance.

Challenges Faced

During the development of the stock profit and loss calculator, several challenges were encountered that required careful consideration and problem-solving. One of the primary difficulties was ensuring accurate handling of user inputs, especially when users entered invalid, negative, or non-numeric values, which demanded the implementation of repeated validation loops to prevent unexpected program termination. Designing a menu-driven structure that maintained a smooth user experience without causing logical flow interruptions also posed a challenge, as the application needed to continuously return to the main menu after each operation while ensuring clarity for the user. Another challenge was implementing data persistence using CSV storage, particularly ensuring that transaction records were appended correctly without overwriting existing data and could be retrieved and displayed in a readable format. Error handling for missing files required additional

attention to avoid crashes when the transaction history file was not present during the first run. Finally, integrating logging functionality introduced complexity in tracking system events while maintaining clean and understandable output for both the user and the developer. These challenges collectively contributed to refining the robustness, usability, and reliability of the final application.

Learnings & Key Takeaways

Through the development of the stock profit and loss calculator application, several important learnings and insights were gained. One of the key takeaways was the practical understanding of modular programming, where separating the application into classes and utility functions improved readability, maintainability, and scalability. Implementing the StockTransaction class reinforced the importance of using object-oriented concepts to encapsulate data and related behavior within a single structure. Additionally, the process highlighted the value of input validation and error handling, as ensuring safe and predictable user interactions is essential for building reliable applications. Working with CSV storage provided hands-on experience in basic data persistence and demonstrated how simple external files can be used to maintain historical records without a database. The integration of logging also emphasized how tracking system activities is useful for debugging and long-term maintenance, especially as applications grow in complexity. Lastly, building a menu-driven workflow improved understanding of user-centric design by focusing on intuitive navigation and clarity in output presentation. Overall, the project strengthened both programming skills and real-world problem-solving abilities, while demonstrating how theoretical concepts can be effectively applied in practical software development.

Future Enhancements

In the future, the stock profit and loss calculator application can be enhanced with several advanced features to improve functionality, usability, and scalability. One of the major enhancements would be the integration of a graphical user interface (GUI) , which would make the system more user-friendly compared to the current command-line interaction. The application can also be extended to support real-time market data retrieval through APIs, allowing users to fetch live stock prices instead of entering values manually. Another improvement would involve replacing the CSV-based storage with a structured database system which will enable more efficient data management, advanced querying, and secure storage. Additional analytical features like visual charts, trend analysis, and multi-stock portfolio tracking could provide deeper insights for users. Implementing user authentication and role-based access would increase security if the system evolves into a multi-user application. Furthermore, automated report generation in PDF or Excel format and cloud backup support would enhance accessibility and reliability. Incorporating unit testing and CI/CD pipelines would also ensure long-term maintainability and robustness. Overall,

these enhancements would transform the application into a more comprehensive, scalable, and professional stock management tool suitable for real-world use.

References

1. Github
2. Wikipedia
3. https://groww.in/calculators/brokerage-calculator
4. https://www.fidelitybanknc.com/calculators/stock-calculator/