UE22CS352B - Object Oriented Analysis & Design

Mini Project Report

Title - Stock Management System

Submitted by:

Name : SRN (Team Members)

NIDHI RAVI	PES1UG22CS381
NIDHI SHEKHAR	PES1UG22CS382
NASHITA AMAAN	PES1UG22CS376
NIKITHA M N	PES1UG22CS391

6th - G

Prof. Bhargavi

January - May 2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FACULTY OF ENGINEERING

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013) 100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Problem Statement:

Managing personal stock portfolios can be complex and time-consuming, especially for individual investors. There is a need for a centralized, user-friendly platform that enables users to efficiently manage their investments, track transactions, and analyze portfolio performance in real-time. Most existing systems are either too complex for casual investors or lack essential features like real-time valuation and performance insights.

This project aims to solve that by developing a web-based Stock Portfolio Management System that simplifies portfolio tracking, supports secure transactions (buy/sell), and provides insightful analysis to help users make informed investment decisions.

Key Features:

User Authentication

- Secure login and registration
- Password encryption using Spring Security

Portfolio Management

- Add, update, and delete stock holdings
- View complete portfolio with company-wise stock data

Transaction Processing

- Execute buy and sell operations for stocks
- Maintain transaction history for each user

Portfolio Analysis

- Analyze performance metrics like gain/loss, total value, and diversification
- Use of Strategy Pattern for flexible analysis approaches

Real-Time Portfolio Valuation

- Dynamically calculate and display portfolio worth
- (Can be extended to fetch live stock prices via API)

Design Pattern Implementations

- Singleton for configuration settings
- Factory for transaction creation
- Observer for stock price updates (simulated or real)

- Strategy for analysis logic

Database Integration

- Use MySQL for storing user data, stocks, and transactions
- Automatic schema generation with Spring Data JPA

MVC Architecture

- Follows the Model-View-Controller pattern
- Separation of concerns between data, UI, and logic

Models:

In this system, we follow an MVC (Model-View-Controller) architecture. The **model classes** represent the core business entities and their data, which are persisted to the database using JPA. Below is a description of each model class and its role in the system.

1. User

Represents a user of the system.

Attributes:

- id : int
- name : String
- email : String
- password : String
- balance : double

Methods:

- get/set methods
- addToBalance(amount: double)
- deductFromBalance(amount: double)

2. Transaction

Represents a buy/sell stock transaction.

Attributes:

- id : int
- stockSymbol : String
- quantity : int
- price : double
- type : String (e.g., “buy” or “sell”)
- timestamp : Date

3. Portfolio

Represents a user’s portfolio containing different stocks.

Attributes:

- id : int
- userId : int
- holdings : Map<String, Integer> (stock symbol and quantity)

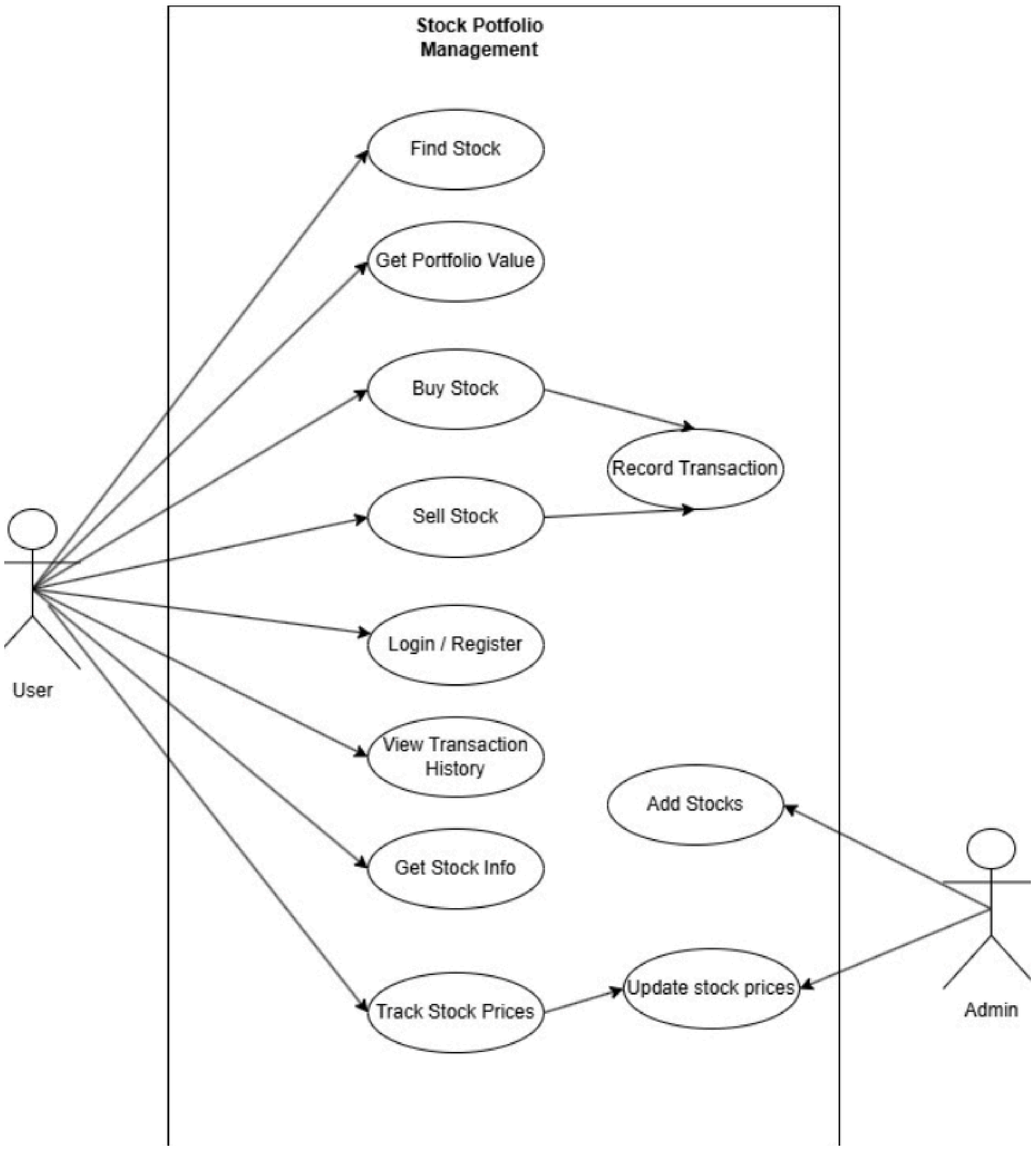
4. Stock

Represents stock information.

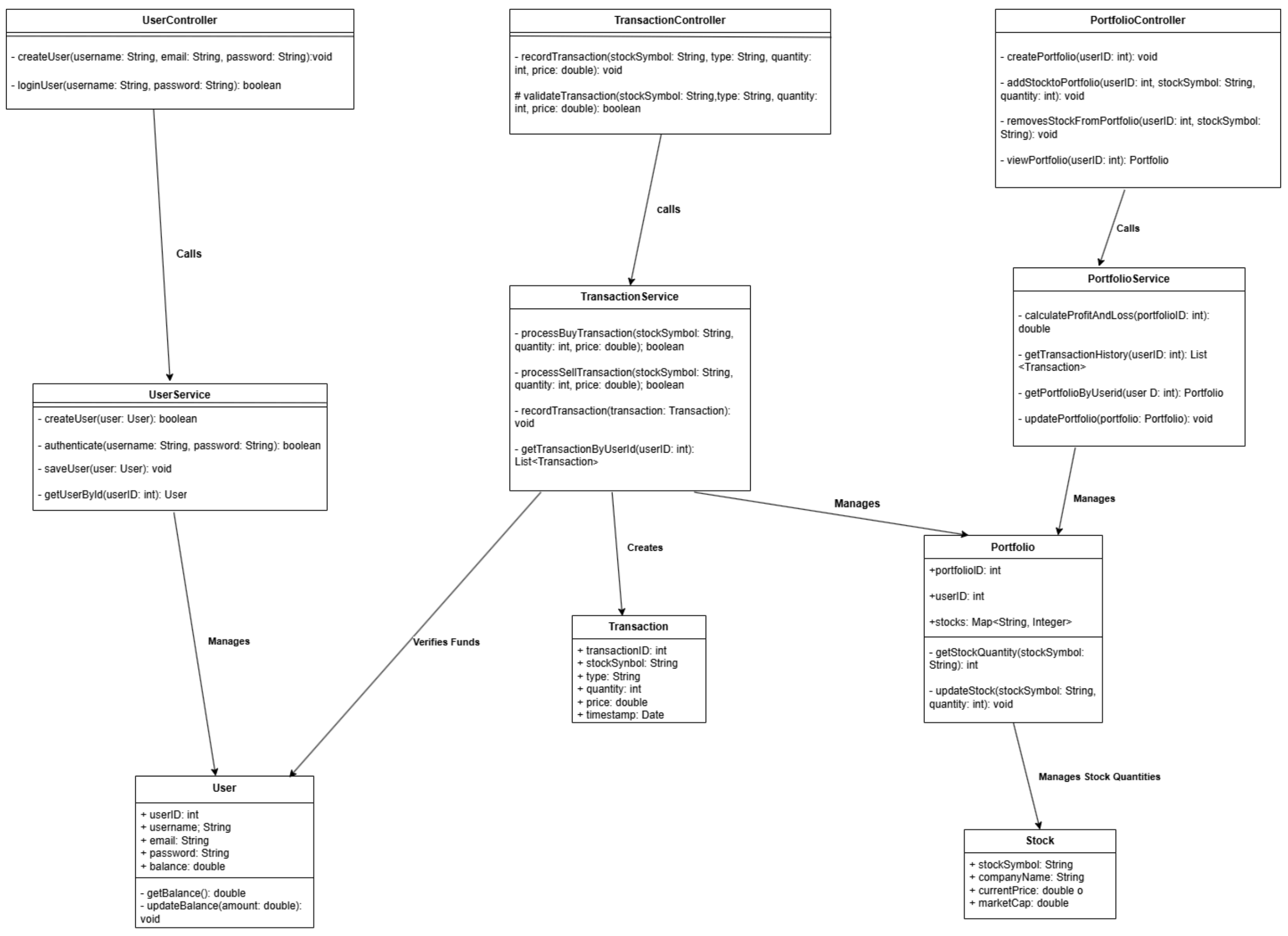
Attributes:

- symbol : String
- name : String
- price : double

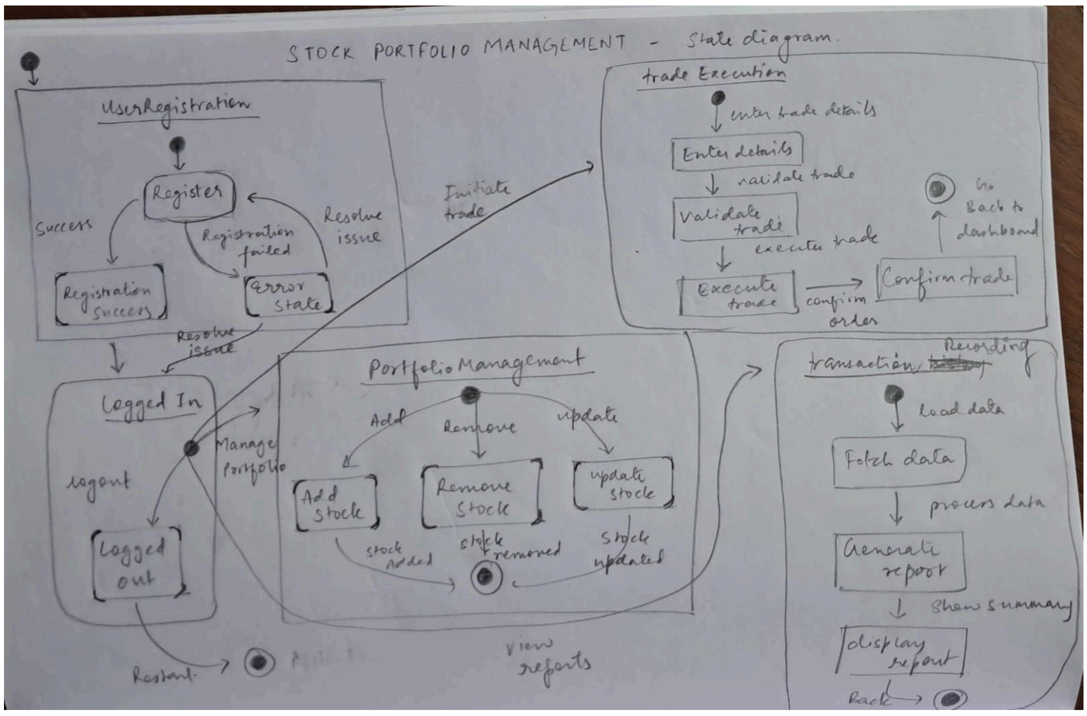
Use Case Diagram:



Class Diagram:

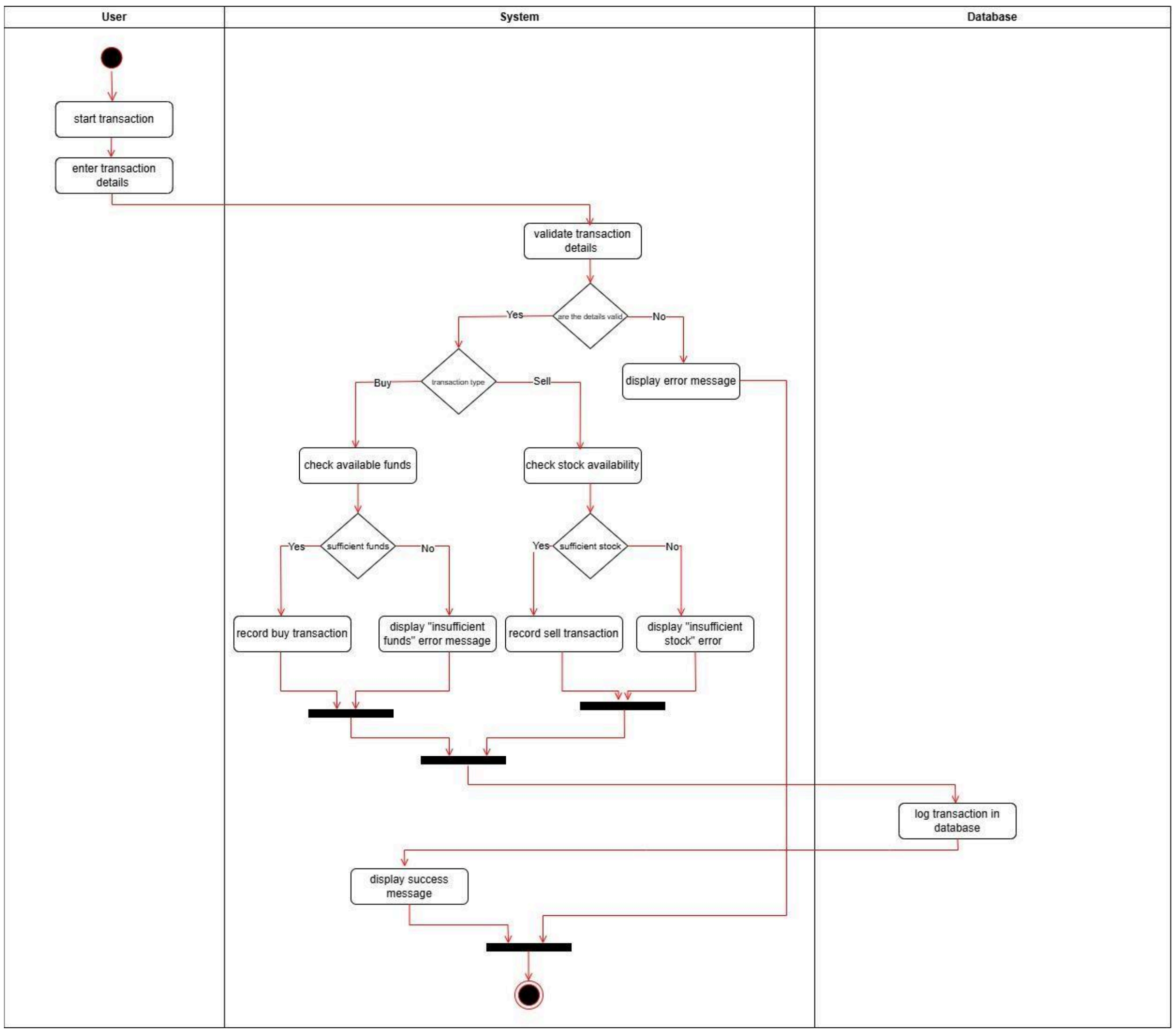

UserController
- createUser(username: String, email: String, password: String):void
- loginUser(username: String, password: String): boolean
Calls
UserService
- createUser(user: User): boolean
- authenticate(username: String, password: String): boolean
- saveUser(user: User): void
- getUserById(userID: int): User
Manages
User
+ userID: int
+ username; String
+ email: String
+ password: String
+ balance: double
- getBalance(): double
- updateBalance(amount: double): void
TransactionController
- recordTransaction(stockSymbol: String, type: String, quantity: int, price: double): void
# validateTransaction(stockSymbol: String,type: String, quantity: int, price: double): boolean
calls
TransactionService
- processBuyTransaction(stockSymbol: String, quantity: int, price: double); boolean
- processSellTransaction(stockSymbol: String, quantity: int, price: double); boolean
- recordTransaction(transaction: Transaction): void
- getTransactionByUserId(userID: int): List<Transaction>
Verifies Funds
Creates
Transaction
+ transactionID: int
+ stockSynbol: String
+ type: String
+ quantity: int
+ price: double
+ timestamp: Date
Manages
PortfolioController
- createPortfolio(userID: int): void
- addStocktoPortfolio(userID: int, stockSymbol: String, quantity: int): void
- removesStockFromPortfolio(userID: int, stockSymbol: String): void
- viewPortfolio(userID: int): Portfolio
Calls
PortfolioService
- calculateProfitAndLoss(portfolioID: int): double
- getTransactionHistory(userID: int): List <Transaction>
- getPortfolioByUserid(user D: int): Portfolio
- updatePortfolio(portfolio: Portfolio): void
Manages
Portfolio
+portfolioID: int
+userID: int
+stocks: Map<String, Integer>
- getStockQuantity(stockSymbol: String): int
- updateStock(stockSymbol: String, quantity: int): void
Manages Stock Quantities
Stock
+ stockSymbol: String
+ companyName: String
+ currentPrice: double o
+ marketCap: double

State Diagram:

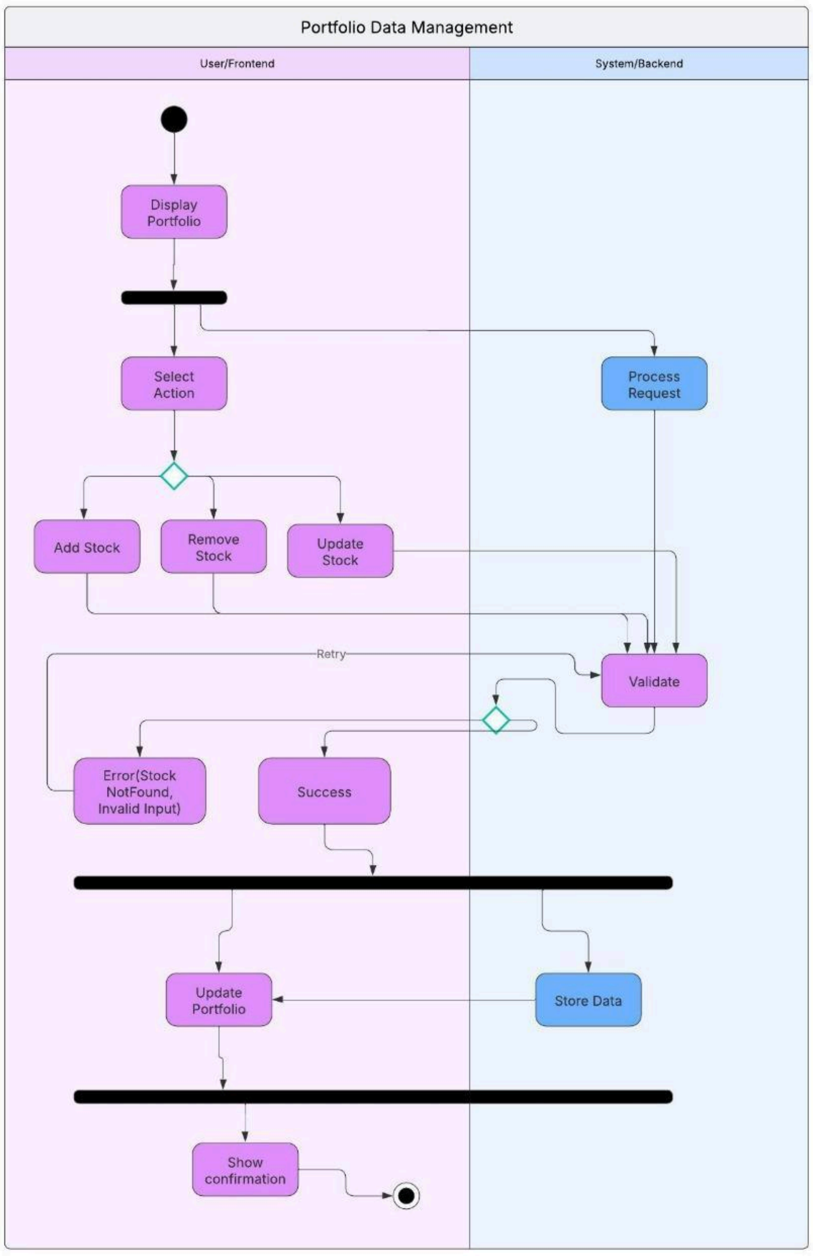


Activity Diagrams:

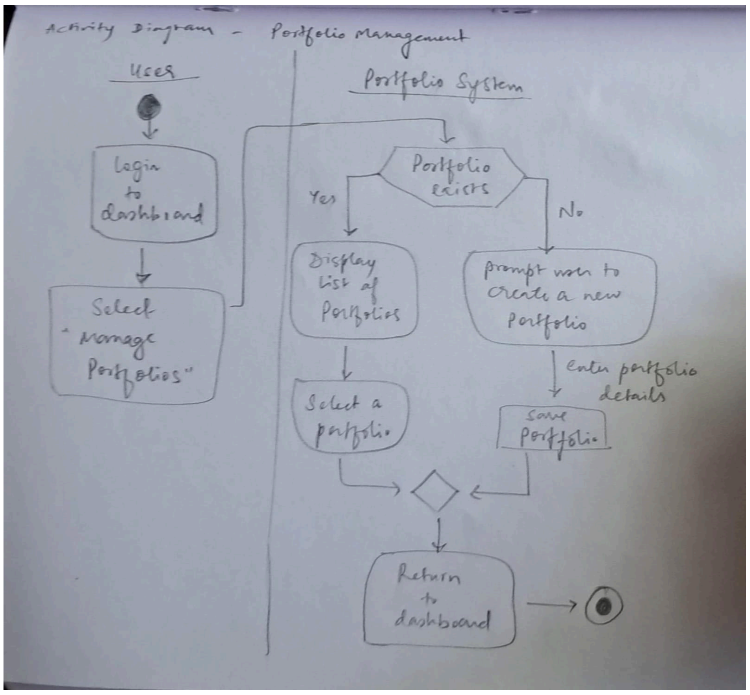
Transaction Controller activity diagram



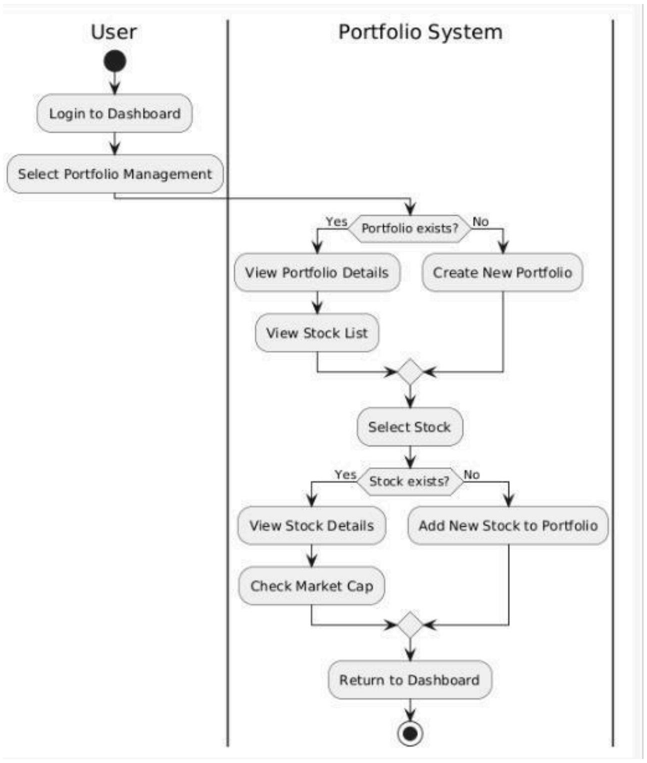
Portfolio Data Controller activity diagram



Portfolio Management Controller activity diagram



Authentication controller activity diagram



Architecture Patterns, Design Principles, and Design Patterns:

Architecture Patterns Followed :

1. Model-View-Controller (MVC) Pattern

Supports the UI and request flow

Applied in:

- **Spring MVC Framework**

💡 Characteristics:

- **Model:** Java entity classes (User, Stock, Transaction)
- **View:** Thymeleaf templates (.html files)
- **Controller:** Spring controllers that route requests and provide data to views.

Design Patterns followed :

1. Singleton Pattern

Used to restrict instantiation of certain components to one object only.

Applied in:

- **Configuration classes** (like SecurityConfig, AppConfig)
- **Database configuration and beans (e.g., DataSource)**

2. Factory Pattern

Used to encapsulate the creation logic of different transaction types (Buy/Sell).

Applied in:

- TransactionFactory class (if implemented)
- Helps in decoupling the creation logic from business logic.

3. Strategy Pattern

Used to implement different portfolio analysis strategies.

Applied in:

- Portfolio analysis functions:
    - Risk calculation
    - Performance evaluation
    - Custom strategies depending on user’s investment style

4. Observer Pattern

Useful for real-time stock price updates.

Applied in:

- If implemented: a stock market data provider class notifying portfolio objects to update value when stock prices change.

5. DAO Pattern (Data Access Object)

Used behind the scenes via Spring Data JPA Repositories.

Applied in:

- Repository interfaces like UserRepository, TransactionRepository, etc.
- Abstracts database interactions.

Design Principles

1. SOLID Principles

• Single Responsibility Principle (SRP)

Each class in the system has one clearly defined responsibility. For example, PortfolioService handles portfolio logic, while TransactionService handles transaction-related operations.

• Open/Closed Principle (OCP)

The system is designed to be open for extension but closed for modification. For instance, new types of portfolio analysis strategies can be added without changing the existing code, utilizing the Strategy Pattern.

• Liskov Substitution Principle (LSP)

Interfaces and abstract classes are used such that derived classes (e.g., different types of transactions) can be substituted without altering the correctness of the program.

• Interface Segregation Principle (ISP)

Interfaces are designed to be specific and minimal, ensuring classes do not implement methods they don't use.

• Dependency Inversion Principle (DIP)

High-level modules do not depend on low-level modules; both depend on abstractions. This is achieved using Spring's dependency injection and interface-based programming.

2. GRASP Principles

• Information Expert

Responsibilities are assigned to classes with the most relevant information. For example, the PortfolioService handles all logic related to stock performance and valuation.

• Controller

Controllers handle user input and system operations. In this system, Spring MVC controllers (e.g., PortfolioController) act as intermediaries between the view and the model.

• Creator

Classes that aggregate or closely use other classes are responsible for creating them. For instance, a Portfolio object may create and manage its own Transaction instances.

• Low Coupling

The system is designed with low interdependence between components. Services, controllers, and repositories communicate through interfaces and are loosely coupled.

• High Cohesion

Classes have a well-defined purpose and encapsulate closely related functionalities, increasing readability and maintainability.

• Polymorphism

Different types of transactions or strategies (e.g., buy/sell or risk/performance analysis) implement common interfaces, allowing them to be used interchangeably.

• Indirection

Controllers delegate work to services and services to repositories. This helps reduce direct dependencies between system layers.

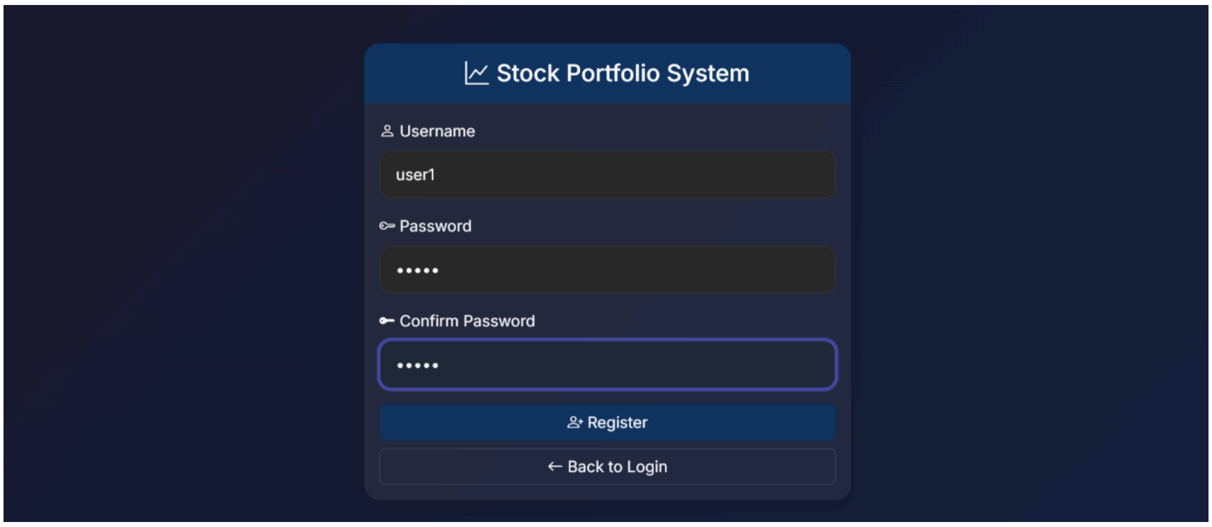
• Protected Variations

Interfaces and abstract classes are used to protect the system from the impact of changes. For instance, the Strategy Pattern protects the analysis logic from variations

Github link to the Codebase:

https://github.com/nidhiravi/Stock_Portfolio_Management

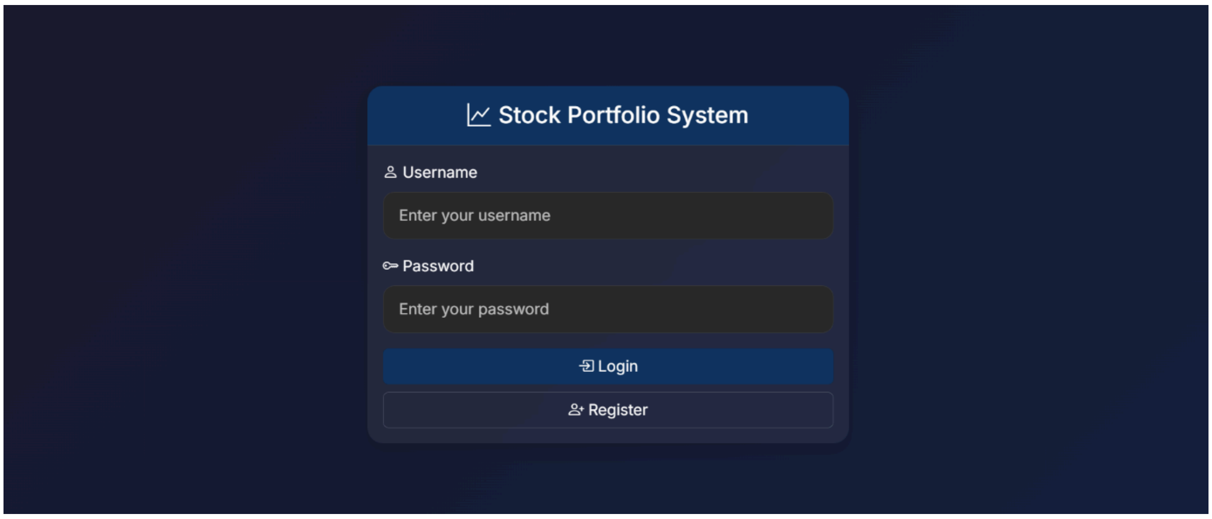
Screenshots of UI



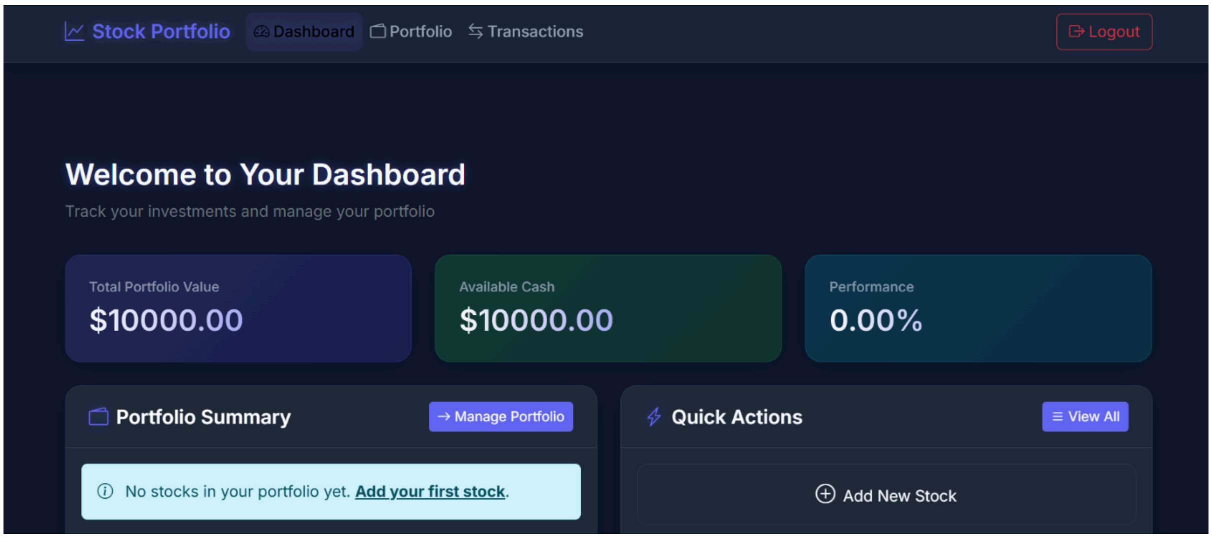
user1

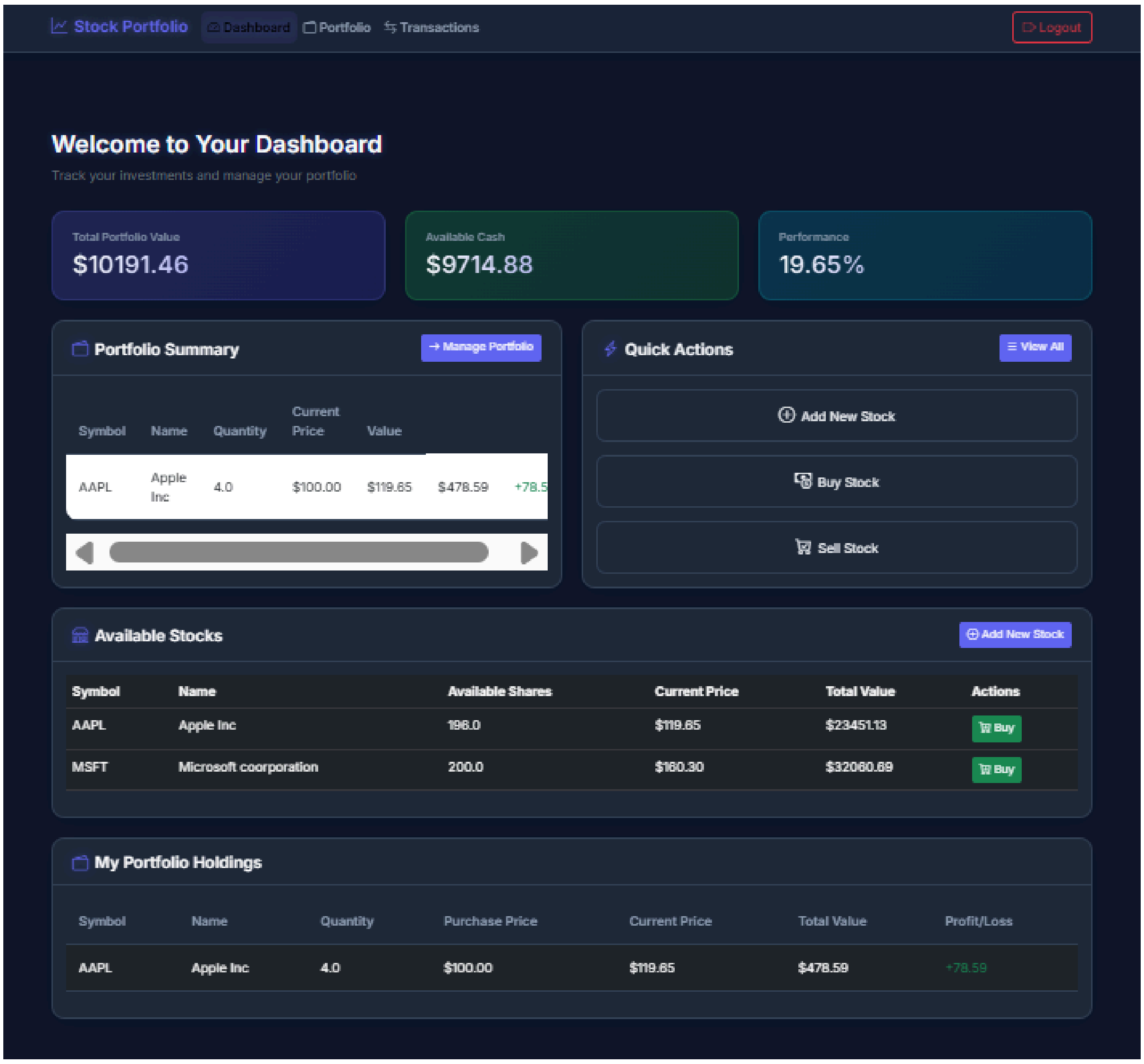
•••••

•••••


Stock Portfolio System
Username
Enter your username
Password
Enter your password
Login
Register

Dashboard:


Stock Portfolio
Dashboard
Portfolio
Transactions
Logout
Welcome to Your Dashboard
Track your investments and manage your portfolio
Total Portfolio Value
$10000.00
Available Cash
$10000.00
Performance
0.00%
Portfolio Summary
→ Manage Portfolio
No stocks in your portfolio yet. Add your first stock.
Quick Actions
View All
Add New Stock



Stock Portfolio | Dashboard | Portfolio | Transactions | Logout

Welcome to Your Dashboard

Track your investments and manage your portfolio

Total Portfolio Value
$10191.46

Available Cash
$9714.88

Performance
19.65%

Portfolio Summary → Manage Portfolio

Symbol	Name	Quantity	Current Price	Value		
AAPL	Apple Inc	4.0	$100.00	$119.65	$478.59	+78.5

Quick Actions ☰ View All

Add New Stock

Buy Stock

Sell Stock

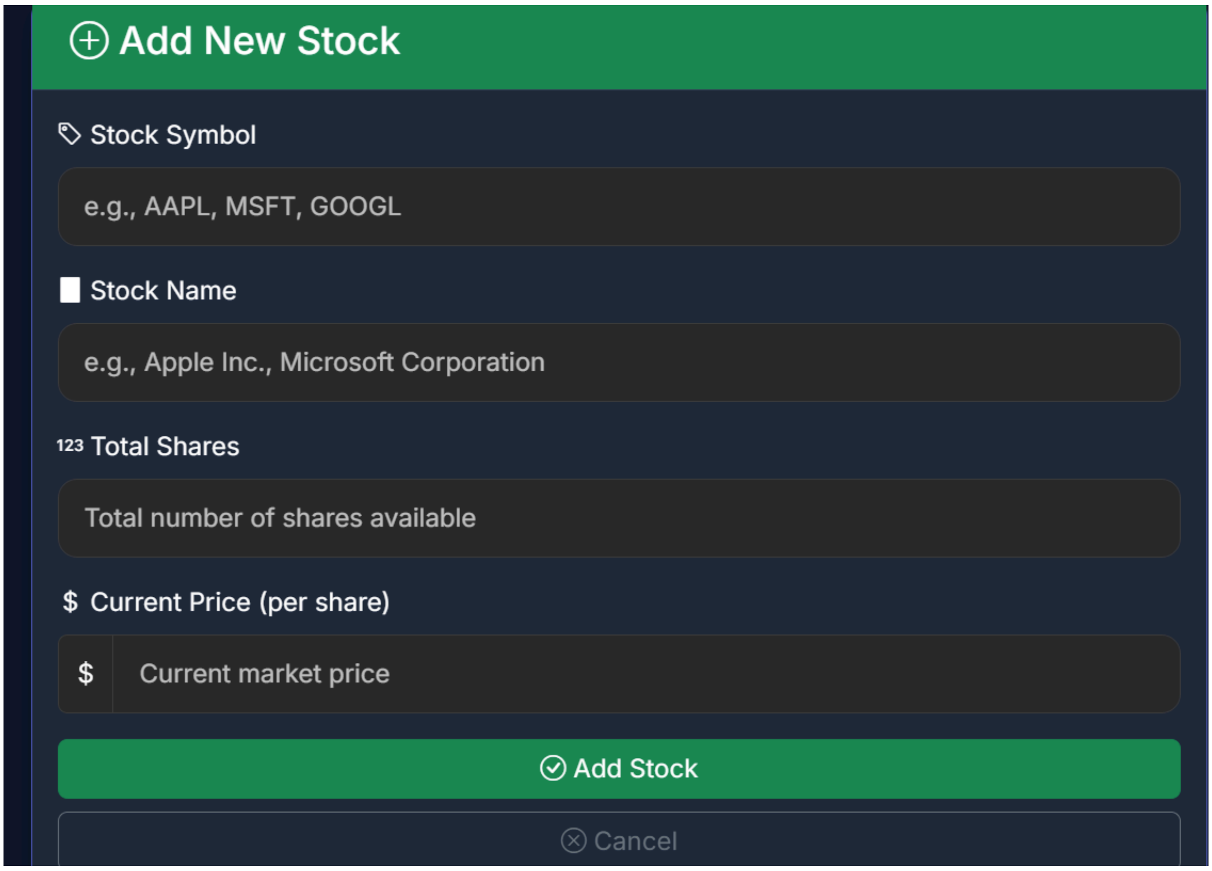
Available Stocks Add New Stock

Symbol	Name	Available Shares	Current Price	Total Value	Actions
AAPL	Apple Inc	196.0	$119.65	$23451.13	Buy
MSFT	Microsoft coorporation	200.0	$160.30	$32060.69	Buy

My Portfolio Holdings

Symbol	Name	Quantity	Purchase Price	Current Price	Total Value	Profit/Loss
AAPL	Apple Inc	4.0	$100.00	$119.65	$478.59	+78.59

Add New Stock



Add New Stock

Stock Symbol

e.g., AAPL, MSFT, GOOGL

Stock Name

e.g., Apple Inc., Microsoft Corporation

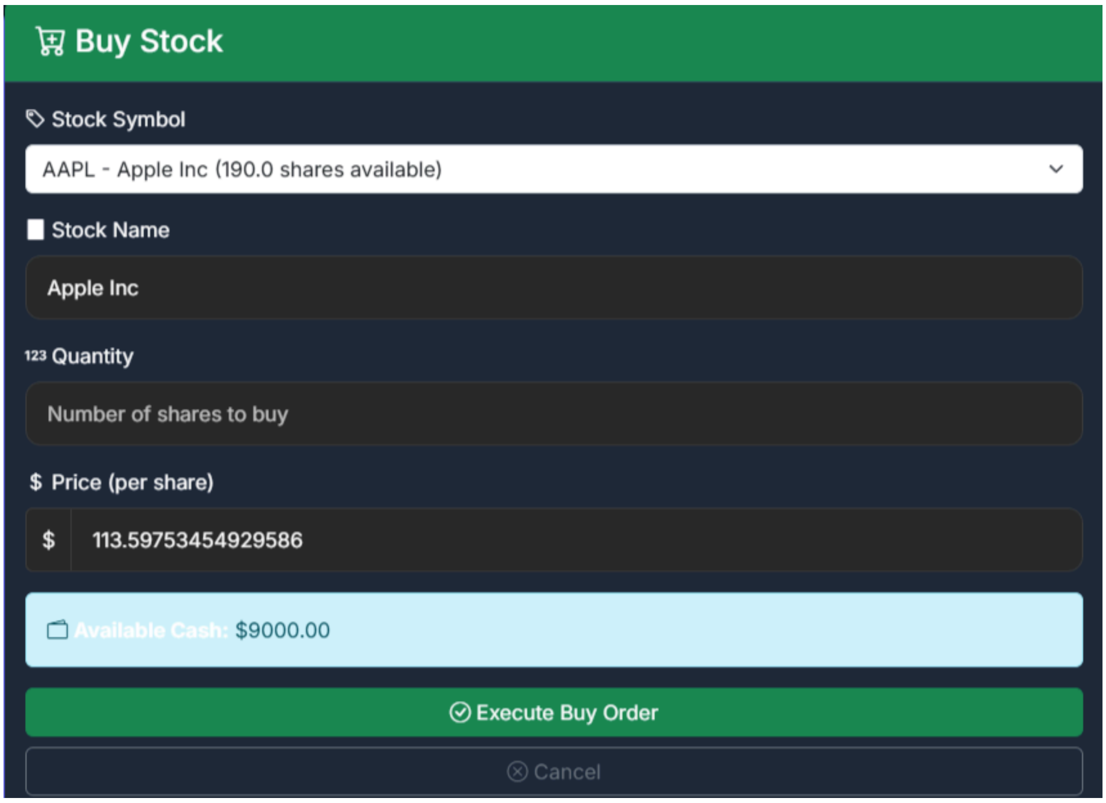
123 Total Shares

Total number of shares available

$ Current Price (per share)

$ Current market price

Buy Stock



Buy Stock

Stock Symbol

AAPL - Apple Inc (190.0 shares available)

Stock Name

Apple Inc

123 Quantity

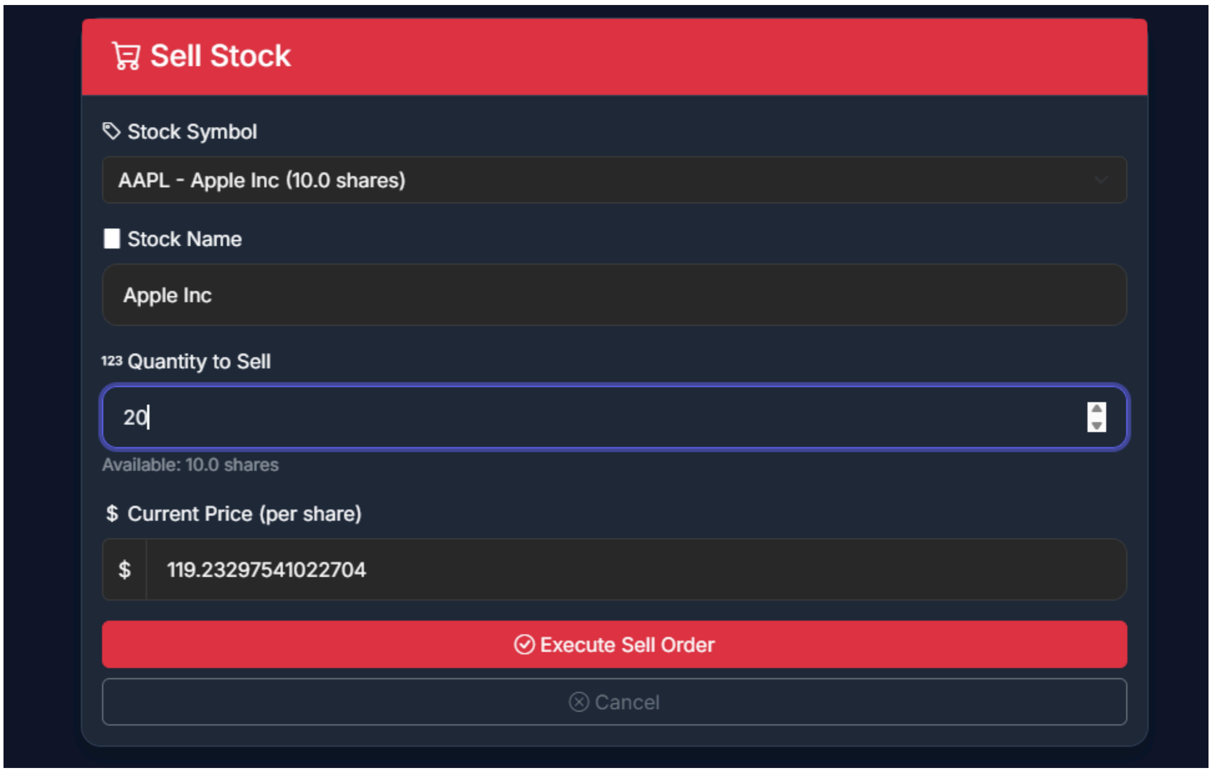
Number of shares to buy

$ Price (per share)

$ 113.59753454929586

Available Cash: $9000.00

Sell Stock



Sell Stock

Stock Symbol: AAPL - Apple Inc (10.0 shares)

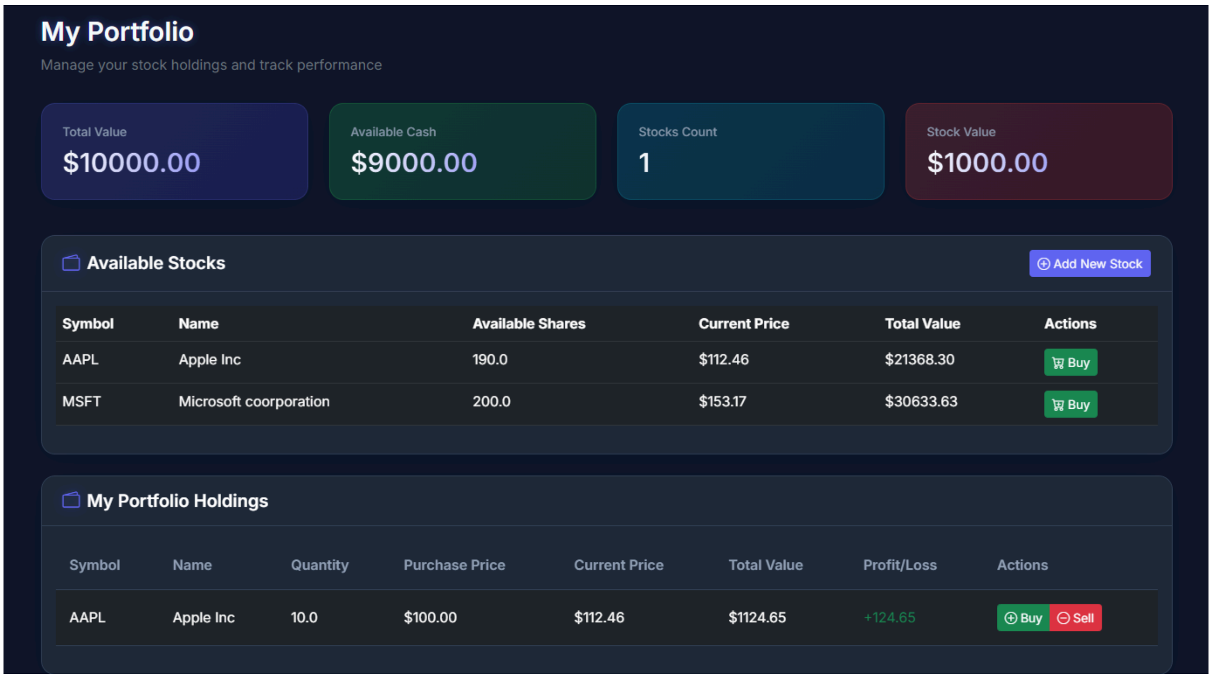
Stock Name: Apple Inc

Quantity to Sell: 20

Available: 10.0 shares

Current Price (per share): $ 119.23297541022704

Portfolio



My Portfolio

Manage your stock holdings and track performance

Total Value	Available Cash	Stocks Count	Stock Value
$10000.00	$9000.00	1	$1000.00

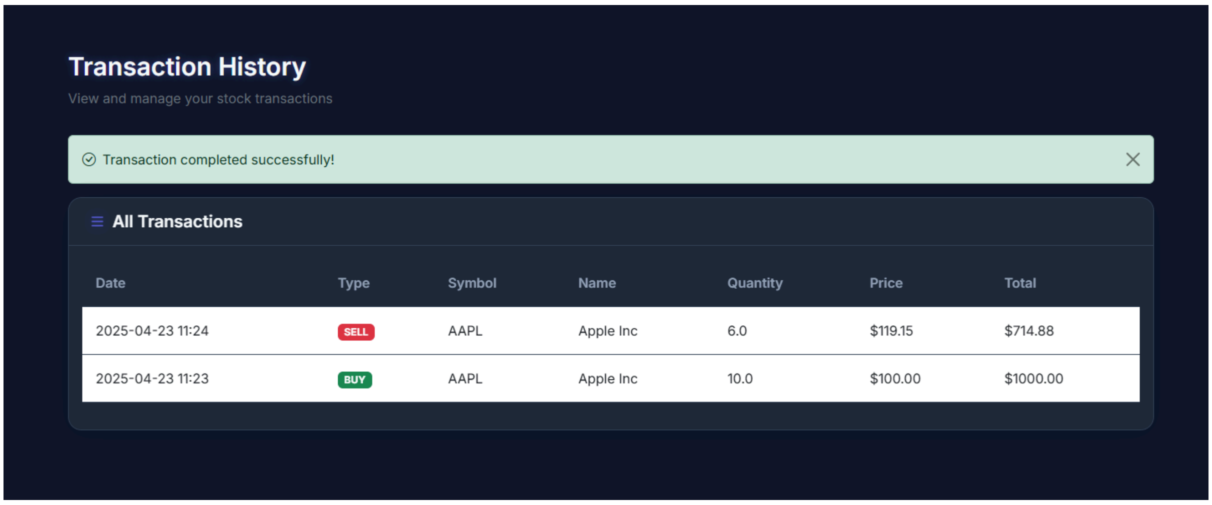
Available Stocks

Symbol	Name	Available Shares	Current Price	Total Value	Actions
AAPL	Apple Inc	190.0	$112.46	$21368.30	Buy
MSFT	Microsoft coorporation	200.0	$153.17	$30633.63	Buy

My Portfolio Holdings

Symbol	Name	Quantity	Purchase Price	Current Price	Total Value	Profit/Loss	Actions
AAPL	Apple Inc	10.0	$100.00	$112.46	$1124.65	+124.65	Buy Sell

Transactions



Transaction History

View and manage your stock transactions

Transaction completed successfully!

All Transactions

Date	Type	Symbol	Name	Quantity	Price	Total
2025-04-23 11:24	SELL	AAPL	Apple Inc	6.0	$119.15	$714.88
2025-04-23 11:23	BUY	AAPL	Apple Inc	10.0	$100.00	$1000.00

Individual contributions of the team members:

Name	**Module worked on**
Nikitha MN (PES1UG22CS391)	Stock.java (Observer pattern)
Nidhi Shekhar (PES1UG22CS382)	Portfolio.java (Strategy pattern)
Nidhi Ravi (PES1UG22CS381)	Transaction.java (Factory pattern)
Nashita Amaan (PES1UG22CS376)	User.java (Singleton pattern)