**ABSTRACT**

This project outlines the implementation of a stock trading application designed for educational purposes. The system automates stock trading activities, allowing users to buy and sell stocks, manage portfolios, and track their performance. The frontend is implemented using Java Swing, and the backend uses MySQL for data management. The project demonstrates essential features like user authentication, stock price tracking, portfolio management, transaction logging, and price history analysis. It ensures robust and efficient management of stock trading operations.

Language Used - Java Core with Swing for GUI

Concept Used – Java Swing

IDE Used – Virtual Studio Code

Database Used - MySQL

CONTENTS

|  |  |  |
| --- | --- | --- |
| **CHAPTERS** |  | **PAGE NO** |
| Chapter 1 | Introduction  1.1Problem Definition 1  1.2 Need 2 |  |
| Chapter 2 | Requirements  2.1 Software Requirement Specifications 3  2.2 Hardware Requirement Specifications 3 |  |
| Chapter 3 | Entity Relationship Diagram 4  3.1 Entity relationship diagram 5 |  |
| Chapter 4 | Schema Diagram 6  4.1 Schema diagram 7 |  |
| Chapter 5 | Implementation  5.1 Backend Implementation 8  5.2 Frontend implemenatation 9  5.3 Creating mainframe class 10 - 13 |  |
| Chapter 6 | Snapshots 14 - 19 |  |
|  | Conclusion |  |
|  | References |  |

**CHAPTER 1**

**INTRODUCTION**

Stock trading applications have transformed the way individuals and institutions engage in the financial markets. By leveraging modern technology, these applications provide users with the tools to monitor stocks, execute trades, and analyze market trends, all within an intuitive interface. In this project, we present the development of a stock trading application with robust backend functionality powered by MySQL and an interactive frontend interface implemented using Java Swing. This system not only caters to the needs of a simulated trading environment but also incorporates essential features like stock price monitoring, portfolio management, transaction tracking, and user authentication.

The backend system is designed to efficiently manage the complexities of stock trading, including maintaining user accounts, executing trades, and ensuring real-time updates of stock prices. A detailed relational database schema has been implemented with various tables, such as users, stocks, portfolio, transactions, and price\_history, to manage critical aspects of the application. Stored procedures for executing trades and updating stock prices ensure transactional consistency and high performance. The schema also supports the generation of insightful views, such as portfolio summaries and transaction histories, enabling users to analyze their investments effectively.

On the frontend, the use of Java Swing ensures an interactive user interface that simplifies user interactions, such as viewing stock information, executing trades, and customizing preferences like dark mode or light mode. Swing's graphical capabilities also allow the integration of dynamic charts and graphs to represent stock trends visually, enhancing the user experience.

The architecture follows a streamlined structure to facilitate understanding and development. The backend implementation includes robust database operations, and the frontend connects seamlessly to provide an all-encompassing experience. Below is the core backend implementation and schema design, along with the frontend structure initially outlined in the project.

**Backend**  
The backend design ensures scalability, reliability, and a secure trading simulation. The database schema comprises:

1. **Users Table :**
2. Manages user credentials, account balances, and statuses, allowing for secure authentication and fund management. Passwords are hashed using SHA-256 for enhanced security.
3. **Stocks\_**Table :
4. Maintains real-time data on stock prices, company names, market capitalization, and trading volumes, supporting comprehensive stock monitoring.
5. **Portfolio Table**  
   Tracks user-specific holdings, including quantities and average purchase prices. The unique key ensures users do not own duplicate stocks, making the data consistent and manageable.
6. **Transactions Table**  
   Logs all trading activities, such as buy/sell orders, quantities, prices, and timestamps, providing an immutable record of all trades.
7. **Price History Table**  
   Records stock price changes over time, enabling users to analyze historical trends.

Stored procedures like execute\_trade streamline trade execution while maintaining transactional integrity, and update\_stock\_price allows efficient updates to stock prices.

**Frontend System**  
The frontend, designed using Java Swing, provides an intuitive interface for interacting with the system. Users can:

* Log in and manage their accounts.
* View and monitor stock prices dynamically.
* Execute buy and sell trades with immediate portfolio updates.
* Analyze their portfolios with real-time data and graphical representations.
* Toggle between light and dark modes for better usability.

Below is the **source code and visual structure** of the frontend UI to represent a portion of the trading system's functionality. The accompanying image demonstrates the simplicity and organization of the initial interface, allowing seamless user navigation.

This project not only simulates the trading experience but also serves as an educational tool, providing insights into the underlying principles of stock trading and financial market operations. Whether for academic learning or practical use, this system encapsulates the essence of stock trading with an engaging and user-friendly application.

**1.1Problem Definition**

This project on Stock Management System is the automation of registration process of stock trading platform. The system is able to provide much information like user's portfolio, stock details and the booking details. The system allows us to add records when a passenger reserves a ticket. It also allows to delete and update the records based on passenger’s requirements. For data storage and retrieval we use the MySQL database. It enables us to add any number of records in our database from the frontend which is Java core. Any changes made in the frontend will be reflected at the backend.

**1.2 Need**

The stock trading application addresses the growing demand for a user-friendly platform to simulate trading in financial markets. It enables users to monitor stock prices, execute trades, and manage portfolios in real-time. Designed for educational and practical use, the system provides insights into trading strategies, portfolio analysis, and market trends. By integrating interactive features and secure backend functionality, it fosters financial literacy while offering a hands-on trading experience for students and enthusiasts.

**CHAPTER 2**

**REQUIREMENTS**

**2.1 Software Requirement Specifications**

Operating System Front End Back End Server Documentation Windows 10

Frontend Software: Virtual Studio Code

Backend Software: MySQL

**2.2 Hardware Requirement Specifications**

Computer Processor Core i3 Processor Speed 2.3 GHz Processor Hard Disk 400 GB or more RAM Min 2GB

**CHAPTER 3**

**ENTITY RELATIONSHIP DIAGRAM**

An entity-relationship (ER) diagram is a specialized graphic that illustrates the interrelationships between entities in a database. ER diagrams often use symbols to represent three different types of information. Boxes are commonly used to represent entities. Diamonds are normally used to represent relationships and ovals are used to represent attributes. If the application is primarily a database application, the entity-relationship approach can be used effectively for modeling some parts of the problem. The main focus in ER modeling is the Data Items in the system and the relationship between them. It aims to create conceptual scheme for the Data from the user’s perspective. The model thus created is independent of any database model. The ER models are frequently represented as ER diagram. Here we present the ER diagram of the above mentioned project.



**CHAPTER 4**

**SCHEMA DIAGRAM**

**4.1 SCHEMA DIAGRAM**

A database schema is the skeleton structure that represents the logical view of the entire database. A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It’s the database designers who design the schema to help programmers understand the database and make it useful.

A database schema can be divided broadly into two categories −

* Physical Database Schema − This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
* Logical Database Schema − This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

**CHAPTER 5**

**IMPLEMENTATION**

**5.1 Backend Implementation**

**MYSQL**

MySQL is an open-source relational database management system (RDBMS). A relational database organizes data into one or more data tables in which data types may be related to each other; these relations help structure the data. SQL is a language programmers use to create, modify and extract data from the relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.

**CREATE DATABASE strade;**

**USE strade;**

**-- Users table for authentication and balances**

**CREATE TABLE users (**

**user\_id INT PRIMARY KEY AUTO\_INCREMENT,**

**username VARCHAR(50) UNIQUE NOT NULL,**

**password VARCHAR(256) NOT NULL,**

**balance DECIMAL(15, 2) DEFAULT 100000.00,**

**email VARCHAR(100) UNIQUE NOT NULL,**

**account\_status ENUM('active', 'suspended', 'closed') DEFAULT 'active',**

**created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,**

**last\_login TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP**

**);**

**-- Stocks table to track stock details**

**CREATE TABLE stocks (**

**stock\_id INT PRIMARY KEY AUTO\_INCREMENT,**

**symbol VARCHAR(10) UNIQUE NOT NULL,**

**company\_name VARCHAR(100) NOT NULL,**

**current\_price DECIMAL(15, 2) NOT NULL,**

**daily\_high DECIMAL(15, 2),**

**daily\_low DECIMAL(15, 2),**

**volume BIGINT DEFAULT 0,**

**market\_cap DECIMAL(20, 2)**

**);**

**-- Portfolio table for user holdings**

**CREATE TABLE portfolio (**

**portfolio\_id INT PRIMARY KEY AUTO\_INCREMENT,**

**user\_id INT NOT NULL,**

**stock\_id INT NOT NULL,**

**quantity INT NOT NULL DEFAULT 0,**

**average\_purchase\_price DECIMAL(15, 2) NOT NULL,**

**last\_updated TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,**

**FOREIGN KEY (user\_id) REFERENCES users(user\_id),**

**FOREIGN KEY (stock\_id) REFERENCES stocks(stock\_id)**

**);**

**-- Transactions table to log all trades**

**CREATE TABLE transactions (**

**transaction\_id INT PRIMARY KEY AUTO\_INCREMENT,**

**user\_id INT NOT NULL,**

**stock\_id INT NOT NULL,**

**transaction\_type ENUM('buy', 'sell') NOT NULL,**

**quantity INT NOT NULL,**

**price\_per\_share DECIMAL(15, 2) NOT NULL,**

**total\_amount DECIMAL(15, 2) NOT NULL,**

**transaction\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,**

**FOREIGN KEY (user\_id) REFERENCES users(user\_id),**

**FOREIGN KEY (stock\_id) REFERENCES stocks(stock\_id)**

**);**

**-- Stored procedure for executing trades**

**DELIMITER //**

**CREATE PROCEDURE execute\_trade(**

**IN p\_username VARCHAR(50),**

**IN p\_symbol VARCHAR(10),**

**IN p\_transaction\_type ENUM('buy', 'sell'),**

**IN p\_quantity INT,**

**OUT p\_status VARCHAR(50)**

**)**

**BEGIN**

**DECLARE v\_user\_id, v\_stock\_id, v\_current\_shares INT;**

**DECLARE v\_current\_price, v\_total\_amount, v\_balance DECIMAL(15, 2);**

**START TRANSACTION;**

**SELECT user\_id, balance INTO v\_user\_id, v\_balance FROM users WHERE username = p\_username FOR UPDATE;**

**SELECT stock\_id, current\_price INTO v\_stock\_id, v\_current\_price FROM stocks WHERE symbol = p\_symbol;**

**SET v\_total\_amount = p\_quantity \* v\_current\_price;**

**IF p\_transaction\_type = 'buy' AND v\_balance >= v\_total\_amount THEN**

**UPDATE users SET balance = balance - v\_total\_amount WHERE user\_id = v\_user\_id;**

**INSERT INTO portfolio (user\_id, stock\_id, quantity, average\_purchase\_price)**

**VALUES (v\_user\_id, v\_stock\_id, p\_quantity, v\_current\_price)**

**ON DUPLICATE KEY UPDATE quantity = quantity + p\_quantity;**

**SET p\_status = 'success';**

**ELSEIF p\_transaction\_type = 'sell' THEN**

**SELECT quantity INTO v\_current\_shares FROM portfolio WHERE user\_id = v\_user\_id AND stock\_id = v\_stock\_id;**

**IF v\_current\_shares >= p\_quantity THEN**

**UPDATE users SET balance = balance + v\_total\_amount WHERE user\_id = v\_user\_id;**

**UPDATE portfolio SET quantity = quantity - p\_quantity WHERE user\_id = v\_user\_id;**

**SET p\_status = 'success';**

**END IF;**

**END IF;**

**IF p\_status = 'success' THEN COMMIT; ELSE ROLLBACK; END IF;**

**END //**

**DELIMITER ;**

**-- Example stock and user data**

**INSERT INTO stocks (symbol, company\_name, current\_price) VALUES ('AAPL', 'Apple', 150.00), ('TSLA', 'Tesla', 800.00);**

**INSERT INTO users (username, password, email) VALUES ('KING', SHA2('password', 256), 'king@example.com');**

**-- Views for analysis**

**CREATE VIEW portfolio\_summary AS**

**SELECT u.username, s.symbol, p.quantity, s.current\_price, (p.quantity \* s.current\_price) AS value**

**FROM users u JOIN portfolio p ON u.user\_id = p.user\_id JOIN stocks s ON p.stock\_id = s.stock\_id;**

**CREATE VIEW transaction\_history AS**

**SELECT u.username, s.symbol, t.transaction\_type, t.quantity, t.price\_per\_share, t.transaction\_date**

**FROM transactions t JOIN users u ON t.user\_id = u.user\_id JOIN stocks s ON t.stock\_id = s.stock\_id;**

**5.2 Frontend Implementation**

Java Core with Swing for GUI

Core Java is the part of Java programming language that is used for creating or developing a general-purpose application. It uses only one tier architecture that is why it is called as ‘stand alone’ application.Core java programming covers the swings, socket, awt, thread concept, collection object and classess.

**Swings**

**Swing** is a GUI widget toolkit for Java. It is part of Oracle's Java Foundation Classes (JFC) – an API for providing a graphical user interface (GUI) for Java programs.

Swing provides a look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform. It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and lists.

**5.3 Creating mainframe class**

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.awt.event.\*;

import java.sql.\*;

import java.util.\*;

import org.jfree.chart.ChartFactory;

import org.jfree.chart.ChartPanel;

import org.jfree.chart.JFreeChart;

import org.jfree.data.category.DefaultCategoryDataset;

public class EnhancedStockTradingApp extends JFrame {

private Connection connection;

private String currentUser;

private double balance = 100000.00;

private Map<String, Double> stockPrices = new HashMap<>();

private Map<String, Integer> portfolio = new HashMap<>();

private JLabel balanceLabel;

private DefaultTableModel stockTableModel, transactionTableModel;

public EnhancedStockTradingApp() { initializeLoginPage(); }

private void initializeLoginPage() {

JFrame loginFrame = new JFrame("Login");

loginFrame.setLayout(new FlowLayout());

loginFrame.setSize(400, 200);

JTextField usernameField = new JTextField(20);

JPasswordField passwordField = new JPasswordField(20);

JButton loginButton = new JButton("Login");

JLabel statusLabel = new JLabel();

loginButton.addActionListener(e -> {

if (validateLogin(usernameField.getText(), new String(passwordField.getPassword()))) {

currentUser = usernameField.getText();

loginFrame.setVisible(false);

initializeTradingDashboard();

} else { statusLabel.setText("Invalid credentials"); }

});

loginFrame.add(new JLabel("Username:")); loginFrame.add(usernameField);

loginFrame.add(new JLabel("Password:")); loginFrame.add(passwordField);

loginFrame.add(loginButton); loginFrame.add(statusLabel);

loginFrame.setVisible(true);

}

private boolean validateLogin(String username, String password) {

try {

connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/strade", "root", "alzic");

PreparedStatement stmt = connection.prepareStatement(

"SELECT \* FROM users WHERE username = ? AND password = SHA2(?, 256)");

stmt.setString(1, username); stmt.setString(2, password);

return stmt.executeQuery().next();

} catch (SQLException e) { e.printStackTrace(); return false; }

}

private void initializeTradingDashboard() {

stockPrices.put("AAPL", 150.00); stockPrices.put("GOOGL", 2800.00);

portfolio.put("AAPL", 10); portfolio.put("GOOGL", 5);

setTitle("Trading Dashboard"); setLayout(new BorderLayout());

setExtendedState(JFrame.MAXIMIZED\_BOTH); setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

createTopPanel(); createCenterPanel(); setVisible(true);

}

private void createTopPanel() {

JPanel panel = new JPanel(new FlowLayout());

balanceLabel = new JLabel("Balance: $" + balance);

JTextField stockField = new JTextField(10), quantityField = new JTextField(5);

JButton buyButton = new JButton("Buy"), sellButton = new JButton("Sell");

buyButton.addActionListener(e -> executeTrade(stockField.getText(), quantityField.getText(), "Buy"));

sellButton.addActionListener(e -> executeTrade(stockField.getText(), quantityField.getText(), "Sell"));

panel.add(balanceLabel); panel.add(new JLabel("Stock:")); panel.add(stockField);

panel.add(new JLabel("Quantity:")); panel.add(quantityField); panel.add(buyButton); panel.add(sellButton);

add(panel, BorderLayout.NORTH);

}

private void createCenterPanel() {

JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL\_SPLIT);

stockTableModel = new DefaultTableModel(new String[] {"Stock", "Price", "Quantity"}, 0);

JTable stockTable = new JTable(stockTableModel);

transactionTableModel = new DefaultTableModel(new String[] {"Date", "Action", "Stock", "Quantity", "Price"}, 0);

JTable transactionTable = new JTable(transactionTableModel);

splitPane.setLeftComponent(new JScrollPane(stockTable));

splitPane.setRightComponent(new JScrollPane(transactionTable));

add(splitPane, BorderLayout.CENTER); updateStockTable();

}

private void executeTrade(String stock, String quantityStr, String action) {

int quantity = Integer.parseInt(quantityStr);

double price = stockPrices.getOrDefault(stock, 0.0);

if ("Buy".equals(action) && balance >= price \* quantity) {

balance -= price \* quantity; portfolio.put(stock, portfolio.getOrDefault(stock, 0) + quantity);

} else if ("Sell".equals(action) && portfolio.getOrDefault(stock, 0) >= quantity) {

balance += price \* quantity; portfolio.put(stock, portfolio.get(stock) - quantity);

}

balanceLabel.setText("Balance: $" + balance); updateStockTable();

}

private void updateStockTable() {

stockTableModel.setRowCount(0);

stockPrices.forEach((stock, price) -> stockTableModel.addRow(new Object[]{stock, price, portfolio.getOrDefault(stock, 0)}));

}

public static void main(String[] args) {

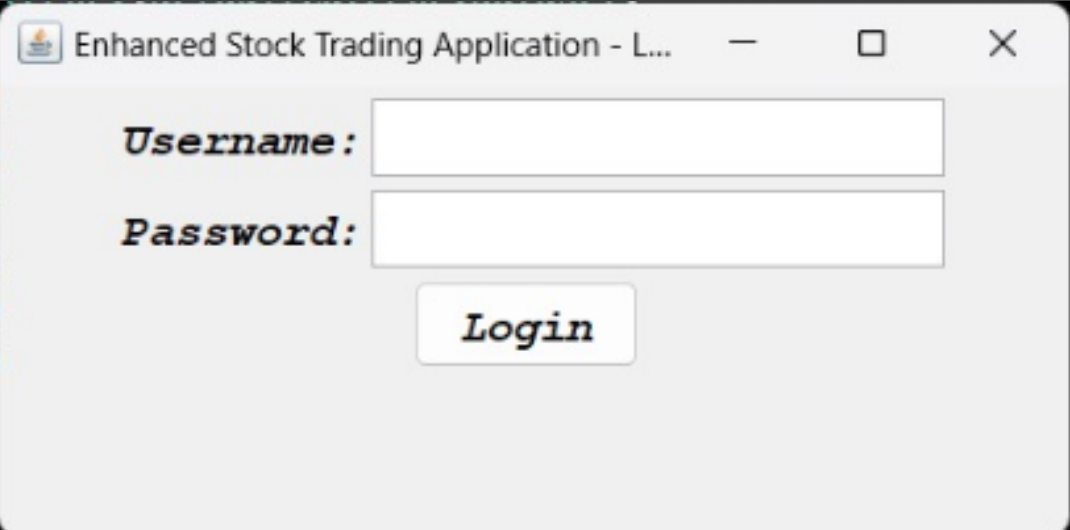
SwingUtilities.invokeLater(EnhancedStockTradingApp::new);

}

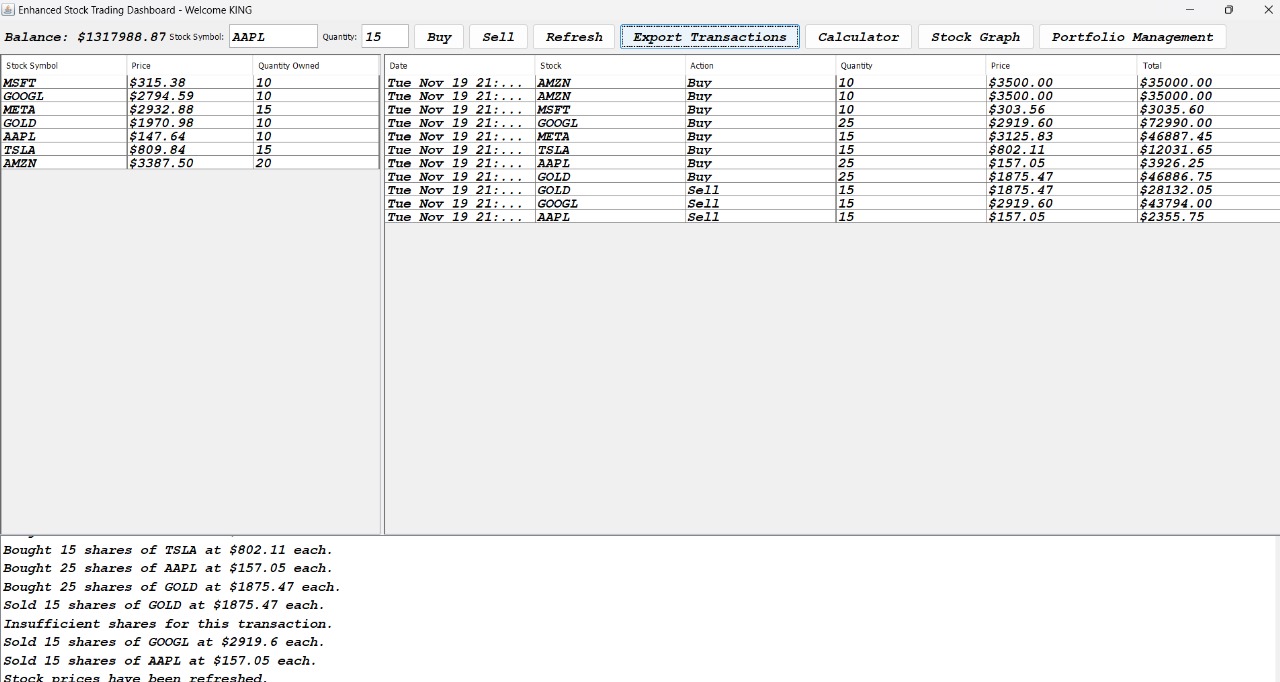
}

**CHAPTER 6**

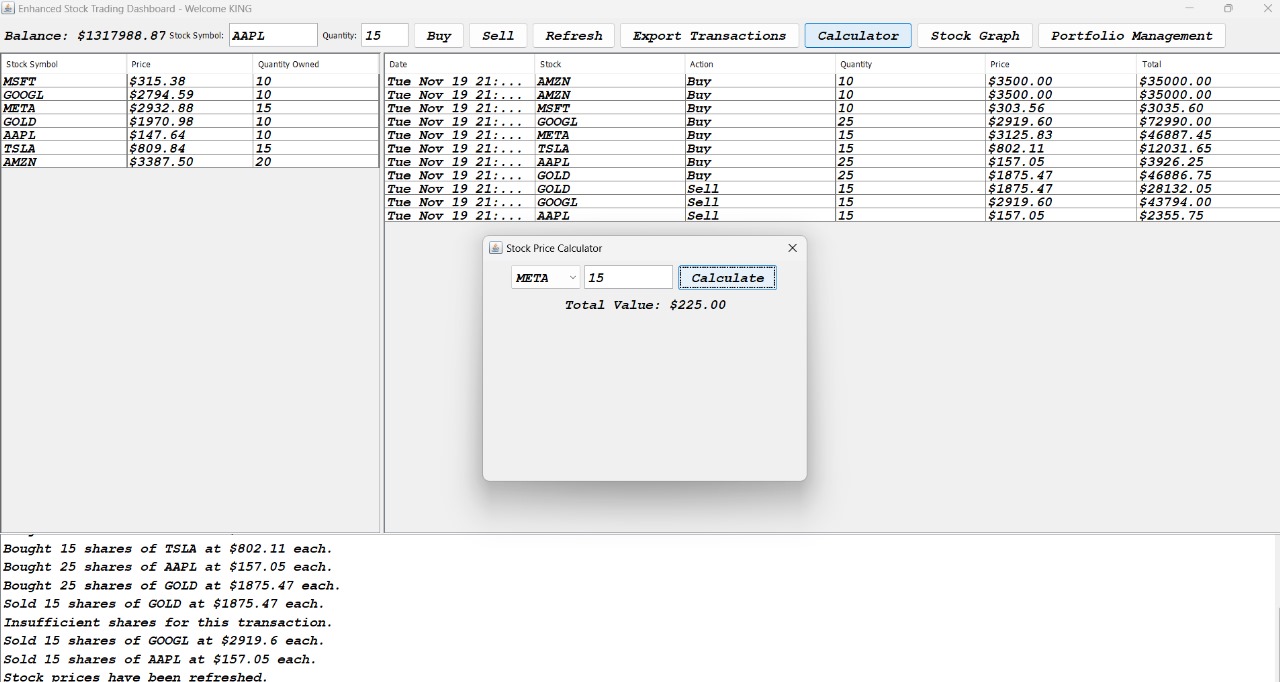
**SNAPSHOTS**

****

**Fig 6.1: Login Operation**



**Fig 6.2: Mainframe**



**Fig 6.3: Menu Button Functions**

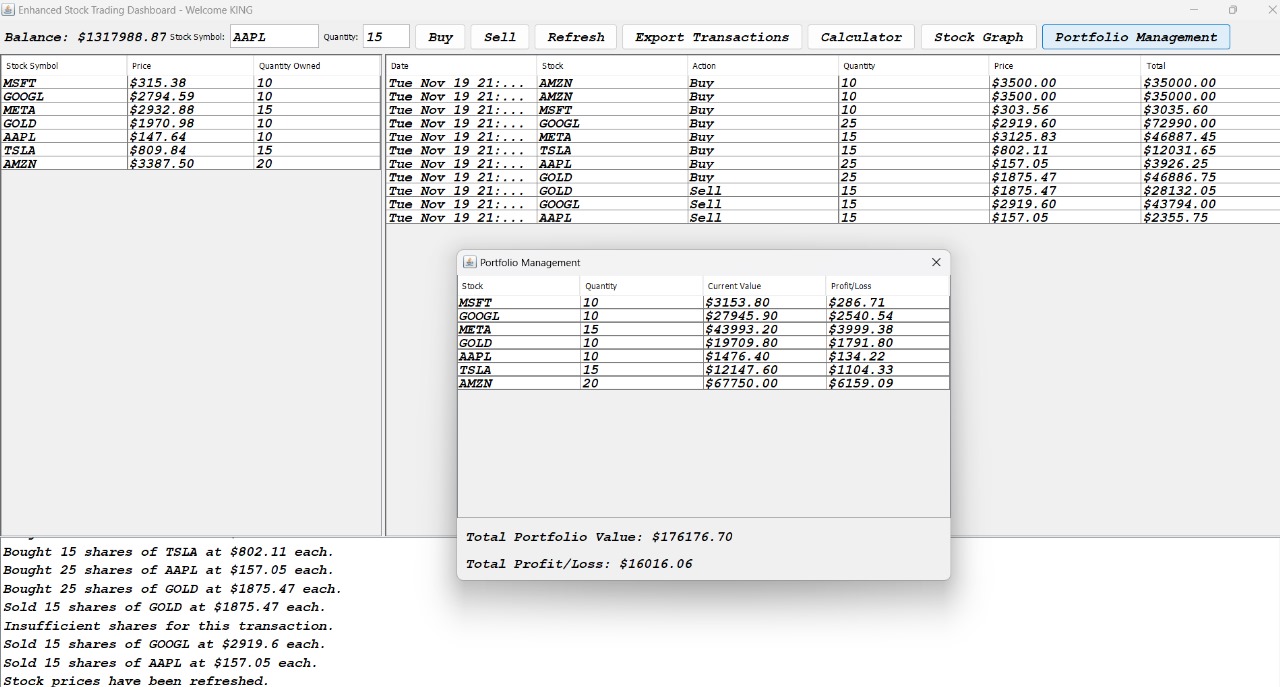
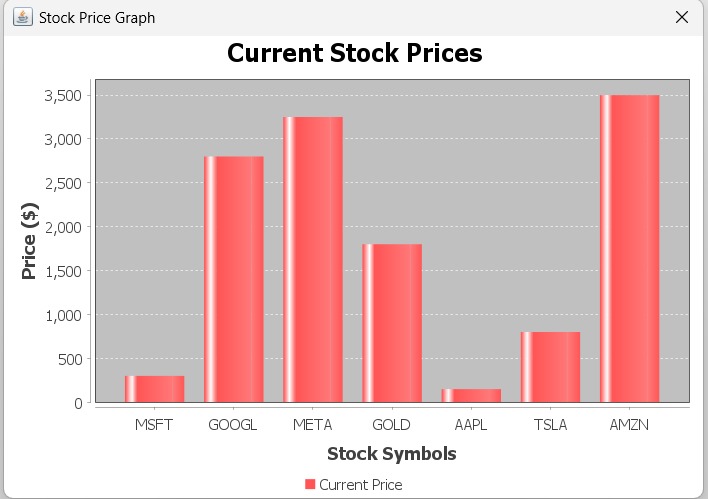
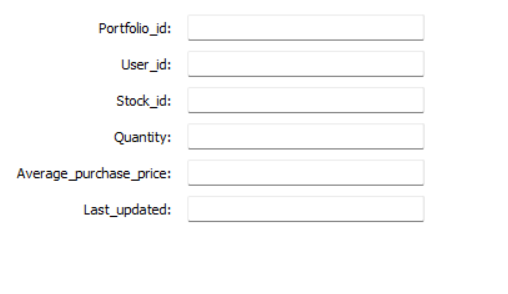


Fig 6.4: Stock Information



**Backend Implementation Tables:**

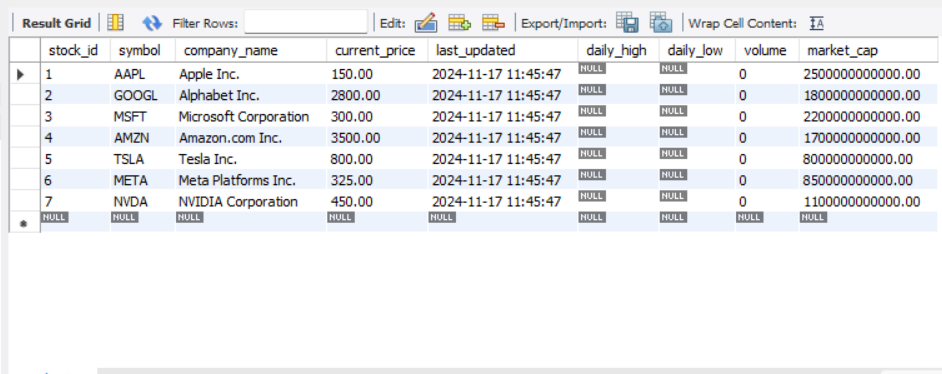
**Fig 6.5: Stock Price Graph**

****

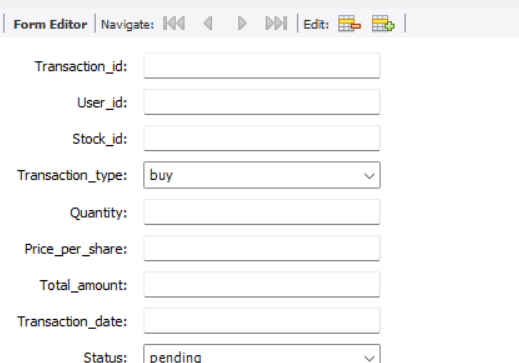
**Fig 6.6: Stock Portfolio**



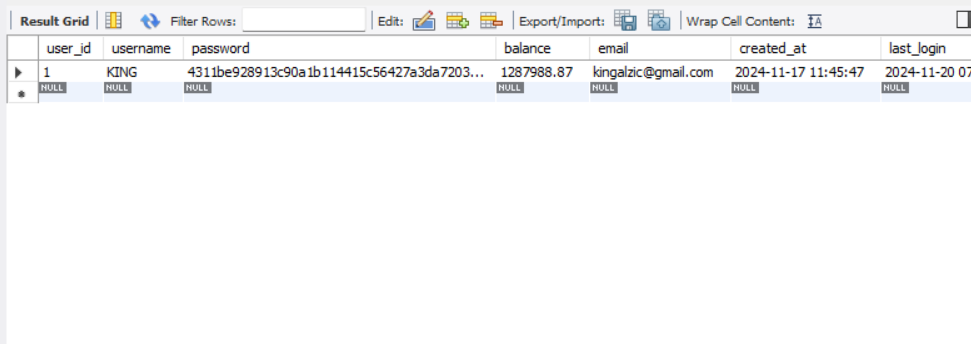
**Fig 6.7: Price History**

****

**Fig 6.8: Stocks**



**Fig 6.9 Transactions**



**Fig 6.10 Users**

**CONCLUSION**

The Stock Trading Project serves as a comprehensive simulation of real-world stock market activities, blending theoretical concepts with practical application. By implementing functionalities such as buying and selling stocks, tracking portfolio performance, and managing transaction logs, the project demonstrates the dynamics of stock trading in an interactive and user-friendly environment.

Through the use of \*Java\* as the core programming language, along with potential integrations like databases or data visualization tools, this project showcases how technology can effectively simulate complex financial systems. Features like real-time price updates and detailed transaction records make it an excellent learning tool for understanding market behavior, risk management, and decision-making in investments.

Overall, this project not only deepens knowledge of stock trading concepts but also hones programming skills, logical thinking, and an understanding of financial systems. It lays a solid foundation for developing more advanced trading applications or pursuing careers in fintech and software development.

**REFERENCES**

[1][**https://www.tdameritrade.com/education.page**](https://www.tdameritrade.com/education.page)

**[2]** [**https://finance.yahoo.com/**](https://finance.yahoo.com/)

**[3]** [**https://www.fool.com/investing/stock-trading-for-beginners/**](https://www.fool.com/investing/stock-trading-for-beginners/)

**[4]** [**https://www.nerdwallet.com/article/investing/how-to-invest-in-stocks**](https://www.nerdwallet.com/article/investing/how-to-invest-in-stocks)