STOCK MARKET MANAGEMENT SYSTEM

Submitted by

Anish chaudhary (RA2311003020135)

Abhijeet kumar (RA2311003020147)

Under the guidance of

Mr. R. Arunkumar, M. E., (Ph.D.).,

(Assistant Professor, Department of Computer Science and Engineering)

21CSC205P/DATABASE MANAGEMENT SYSTEMS

PROJECT REPORT

IV SEMESTER /II YEAR

FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTER SCIENCE ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
RAMAPURAM, CHENNAI

APRIL 2025

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Deemed to be University U/S3 of UGC Act,1956)

BONAFIDE CERTIFICATE

Certified that this project report titled **"STOCK MARKET MANAGEMENT SYSTEM"** is the Bonafide work of **Anish chaudhary [RA2311003020135]**, **Abhijeet Kumar [RA2311003020147]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an occasion on this or any other candidate. This project work confirms **21CSC205P/DATABASE MANAGEMENT SYSTEMS,** IV Semester, II year, 2025.

SIGNATURE

Mr. R. ARUNKUMAR, M. E., (Ph.D.).,
Assistant Professor
Computer Science and Engineering,
SRM Institute of Science and Technology, Ramapuram, Chennai.

SIGNATURE

Dr. J. Sutha,,M.E., Ph.D.,
Professor and Head
Computer Science and Engineering,
SRM Institute of Science and Technology, Ramapuram, Chennai.

Submitted for the project viva-voce held on _____ at SRM Institute of Science and Technology, Ramapuram, Chennai.

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
RAMAPURAM, CHENNAI

DECLARATION

We hereby declare that the entire work contained in this project report titled "**STOCK MARKET MANAGEMENT SYSTEM**" has been carried out by **Anish chaudhary [RA2311003020135]**, **Abhijeet Kumar [RA2311003020147]** at SRM Institute of Science and Technology, Ramapuram, Chennai, under the guidance of **Mr. R. ARUNKUMAR, M.E., (Ph.D.).,Assistant Professor**, Department of Computer Science and Engineering.

Place: Chennai

Date: **Anish chaudhary**

Abhijeet Kumar

ABSTRACT

The Stock Market Management System is a comprehensive, user-friendly software platform designed to transform and streamline the way users interact with stock market operations. This system serves as a centralized hub for investors, traders, and financial analysts, providing a range of essential features such as real-time trading, secure digital wallet management, portfolio tracking, and insightful data visualization. By integrating live stock market feeds, the platform allows users to make informed and timely decisions based on current market trends and stock performance.The system employs a robust MySQL backend to ensure the secure and efficient handling of large volumes of data, including user information, transaction logs, and market records. Its modular architecture allows for easy scalability and customization, ensuring that the system remains adaptable to the growing and changing demands of users and the financial market.With an interactive and intuitive interface developed using Custom Tkinter in Python, the platform offers a seamless user experience, regardless of technical expertise. Core functionalities include user registration and authentication, stock performance analysis, transaction processing, automated financial reporting, and historical data tracking. Additionally, real-time updates and graphical analytics empower users to visualize market trends, evaluate past investments, and refine their trading strategies accordingly.The system emphasizes security, transparency, and operational efficiency, making it suitable for both novice investors and seasoned professionals. By automating many aspects of stock trading and financial management, the Stock Market Management System reduces manual effort, enhances decision- making, and increases user satisfaction. Overall, this platform presents a reliable, scalable, and intelligent solution for navigating the complexities of modern stock market activities and managing investment portfolios with greater accuracy and control. Overall, the Stock Market Management System represents a reliable, scalable, and intelligent solution for navigating the complexities of modern stock market activities. It offers users greater control, deeper insights, and an elevated level of confidence in managing their investment portfolios in today's fast-paced financial world.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	iv
	LIST OF FIGURES	vi
	LIST OF ABBREVIATIONS	vii
1	**INTRODUCTION**	1
	1.1 INTRODUCTION TO DBMS, RDBMS, SQL	1
	1.2 PROBLEM STATEMENT	2
	1.3 OBJECTIVE	2
	1.4 SCOPE AND MOTIVATION	3
2	**EXISTING SYSTEM**	5
	2.1 HARDWARE REQUIREMENTS	6
	2.2 SOFTWARE REQUIREMENTS	6
3	**DESIGN (ER DESIGN AND FRONT END DESIGN)**	7
	3.1 ER DIAGRAM	7
	3.2 ARCHITECTURE DIAGRAM	8
	3.3 USE CASE DIAGRAM	9
	3.4 FRONT END DESIGN	10
4	**PROPOSED METHODOLOGY**	11
	4.1 MODULES DESCRIPTION	11
	4.2 DATABASE CONNECTIVITY	14
5	**IMPLEMENTATION**	15
6	**RESULT**	25
7	**CONCLUSION**	29
	REFERENCES	30

LIST OF FIGURES

Fig No.	Figure	Page No.
3.1	ER-DIAGRAM OF SMMS	7
3.2	ARCHITECTURE DIAGRAM OF SMMS	8
3.3	ORDER WORKFLOW OF SMMS	9
6.1	LANDING PAGE (DARK BG) OF SMMS	25
6.2	LANDING PAGE(WHITE BG) OF SMMS	25
6.3	SIGNUP PAGE OF SMMS	26
6.4	LOGIN PAGE OF SMMS	27
6.5	USER'S PAGE OF SMMS	27
6.6	ORDER INTERFACE OF SMMS	28
6.7	BUY AND SELL OF SMMS	28

LIST OF ABBREVIATIONS

ACID Atomicity, Consistency, Isolation, Durability.

DBMS Database Management System.

ER Entity- Relationship

SMMS Stock Market Management System

RAM Random Access Memory

RDBMS Relational Database Management Systems

SSD Solid State Drive

SQL Structured Query Language

UML Unified Modeling Language

LIST OF TABLES

S.No	Name	Page No.
1.	Hardware Requirements	13
2.	Software Requirements	13

1. INTRODUCTION

1.1 INTRODUCTION TO DBMS, RDBMS, SQL

A Database Management System (DBMS) is software designed to define, create, maintain, and control access to databases. It provides a structured and systematic way of managing data, ensuring data accuracy, integrity, and security. In the context of stock trading platforms, DBMS plays a vital role in managing vast amounts of user information, financial transactions, and real-time stock data. Unlike traditional file systems, a DBMS supports data abstraction, reduces redundancy, and enforces constraints to maintain consistency.

A Relational Database Management System (RDBMS) is a type of DBMS based on the relational model introduced by E. F. Codd. Data in an RDBMS is stored in tabular format—tables made up of rows and columns. These tables are linked using keys such as *primary keys* and *foreign keys*, ensuring referential integrity and supporting complex queries. RDBMSs are especially well-suited for stock market systems where relationships between users, companies, transactions, and shareholdings must be maintained in a consistent and normalized form. Widely-used RDBMS software includes MySQL, PostgreSQL, Oracle, and SQL Server.

Structured Query Language (SQL) is the standard language used to communicate with an RDBMS. SQL enables developers to perform various operations such as creating tables (CREATE), inserting data (INSERT), fetching data (SELECT), updating records (UPDATE), and deleting information (DELETE). For a stock market system, SQL is instrumental in retrieving up-to-date share values, updating wallet balances, recording trades, and managing user sessions. It also supports filters (WHERE), sorting (ORDER BY), grouping (GROUP BY), and joins (JOIN), allowing for dynamic querying of complex financial data.

In the Stock Market Management System, DBMS principles are used to model and manage entities such as users, digital wallets, stock transactions, and market data. RDBMS architecture ensures logical organization, scalability, and integrity across interconnected modules. SQL serves as the backbone for implementing all data-related operations, including user registration, login authentication, portfolio management, and transaction history retrieval. These components ensure that the platform is secure, responsive, and reliable for users performing critical stock trading tasks.

1.2 Problem Statement

Traditional methods of stock trading and portfolio management can be complex, time-consuming, and difficult for individuals to access and navigate. Users often face challenges such as lack of real-time stock data, unclear investment tracking, insecure transactions, and limited user-friendly tools for making informed decisions. These issues can lead to poor financial choices and reduced user engagement.

The StockUP project aims to create a secure, efficient, and user-friendly platform that simplifies stock trading and portfolio management. It provides real-time data, seamless transactions, and interactive tools to help users confidently invest and monitor their assets with ease.

1.3 Objective

The main objective of the Stock Market Management System (StockUP) is to design and implement a database-driven platform that simplifies and automates the stock trading and portfolio management process for users. The specific goals of this project are:

1. To develop a centralized system that allows users to register, log in, and interact with the platform through a secure and role-based interface.
2. To enable users to buy and sell stocks through a secure portal with real-time updates and wallet balance management.
3. To allow users to view their investment portfolios, track individual stock performance, and analyze portfolio value over time.
4. To create an admin module to manage stock listings, monitor user activity, and maintain data consistency across transactions.
5. To ensure data accuracy and integrity through the use of relational databases, proper table design, and normalization techniques.

6. To eliminate manual tracking of trades and wallet updates, reducing errors and improving efficiency.
7. To implement secure authentication using phone number and password, with encrypted storage of sensitive user information.
8. To demonstrate practical application of Database Management System (DBMS) concepts such as entity-relationship modeling, SQL queries, and data validation in a real-time financial system.

1.4 Scope and Motivation

The Stock Market Trading System (SMTS) is designed to cover all essential aspects of stock trading and portfolio management within a secure and user-friendly environment. It focuses on the core processes involving individual traders and system administrators, ensuring that each stakeholder has a dedicated interface to interact with the system effectively. The scope of this project includes secure login and authentication of users using their Email ID and Password, allowing traders to execute stock trades, manage their portfolios, and track real-time stock data.

This system is limited to handling operations directly related to stock trading, portfolio management, and user authentication. It does not currently cover additional financial functionalities such as tax calculations, loan management, or investment advisory services. However, the system is designed in a modular and scalable way, enabling future integration of these features if required.

The platform's primary goal is to make stock trading more accessible and efficient by leveraging modern technology and providing a reliable, secure environment for users to manage their investments. By focusing on key functionalities such as user authentication, real-time data updates, and secure financial transactions, the system addresses the core needs of traders and investors in today's fast-paced financial markets.

This project also emphasizes scalability and flexibility, allowing future integration of additional features such as tax optimization, investment recommendations, and advanced analytics. Through this approach, the Stock Market Trading System aims to create a comprehensive ecosystem that evolves with users' growing needs and the dynamic nature of the stock market.

Motivation

The primary motivation behind developing the Stock Market Trading System is to address the inefficiencies and challenges faced by investors and traders who rely on traditional or semi-digital platforms for stock trading and portfolio management. Many existing systems involve complex interfaces, limited real-time data, and manual transaction processes, which can result in delays, errors, and missed opportunities in fast-moving financial markets. Investors often struggle with managing their portfolios, tracking stock performance, and making informed decisions due to the lack of integrated, real-time tools.

The growing demand for more efficient, secure, and accessible stock trading platforms inspired the development of a system that not only streamlines these processes but also ensures security, accuracy, and real-time updates. This project leverages modern technologies to create a seamless trading experience, allowing users to easily buy, sell, and manage their investments.

From an academic perspective, this project serves as a practical application of concepts learned in database management, financial systems, and user interface design. It demonstrates how relational databases, security protocols, and real-time data handling can be used to solve real-world problems in the financial sector. The motivation also stems from the desire to create a platform that can be enhanced and adopted by individual traders and institutions, offering an efficient, scalable solution for stock market trading and portfolio management.Furthermore, the system is designed to grow alongside the evolving financial landscape, with the flexibility to incorporate advanced features like automated trading, predictive analytics, and more comprehensive financial tools. By building this platform, the project aims to empower both novice and experienced traders to make informed, confident investment decisions, contributing to the overall accessibility and efficiency of the stock market.

2. EXISTING SYSTEM

In many financial institutions and trading platforms, the stock trading and portfolio management process is still managed through traditional or semi-digital methods. Typically, traders are required to use basic trading platforms with limited functionality or rely on manual processes for portfolio tracking and transaction execution. These systems often lack integration, resulting in separate processes for stock trading, portfolio management, real-time data updates, and transaction history tracking. This fragmented workflow leads to delays, missed opportunities, and frequent errors in managing investments.

Most existing systems do not provide real-time market data updates or a centralized platform to manage stock transactions and portfolios efficiently. Traders often do not receive instant confirmation of their transaction status and may face difficulties in portfolio tracking due to the absence of automated tools that integrate stock data with portfolio performance. Additionally, investors rarely have access to up-to-date information on their holdings and stock prices, which makes planning and making informed decisions challenging.

For administrators and financial institutions, the workload is even more significant. They must manually verify user eligibility, handle transactions, and ensure the security of sensitive financial data. Since these tasks are often performed using spreadsheets or standalone software, the risk of data inconsistency and duplication is high. Additionally, the lack of a unified platform means there is no proper mechanism for tracking user actions or maintaining a history of transactions and trades.

Security is another major limitation in existing systems. Inadequate login mechanisms and insufficient role-based access control make the system vulnerable to unauthorized access, data breaches, and mismanagement. Without secure authentication (like email and password for each role), sensitive financial data can be exposed or manipulated.

Overall, while some platforms have attempted to digitize parts of the stock trading process, most existing systems are either too simplistic or inefficient for modern financial needs. They fail to provide a seamless, real-time, and centralized experience for all users—traders, investors, and administrators—necessitating the development of a more reliable and automated solution like the Stock Market Trading System (SMTS).

2.1 HARDWARE REQUIREMENTS

Table 1:-Hardware Requirement

Component	Minimum Requirement	Recommended Requirement
Processor (CPU)	Dual Core 2.0 GHz	Intel i5 or higher
RAM	4 GB	8 GB or higher
Hard Disk	250 GB (free space: 2 GB)	500 GB SSD
Display	1024 × 768 resolution	Full HD (1920 × 1080)
Input Devices	Keyboard and Mouse	Keyboard, Mouse, or Touchpad
Network	Basic Internet connectivity	Stable high-speed connection

2.2 SOFTWARE REQUIREMENTS

Table 2:-Software Requirement

Category	Software/Tool	Purpose
Operating System	Windows 10 / Linux / macOS	Platform for development and testing
Backend Runtime	python (v3.8 or above)	Server-side Script and logic handling
Web Framework	Custom Tkinter	GUI development for user interaction
Frontend Tools	CustomTkinter	User interface development
Database	MySQL (v8.0 or above)	RDBMS to store and manage data
Database Client	MySQL Workbench	Visual database management
Code Editor	Visual Studio Code / Sublime Text	Writing and editing code
Browser	Google Chrome / Firefox / Edge	Accessing the web application
Version Control	Git and Github	Source code management

3. DESIGN(ER DESIGN AND FRONT END DESIGN)

3.1 ER Diagram

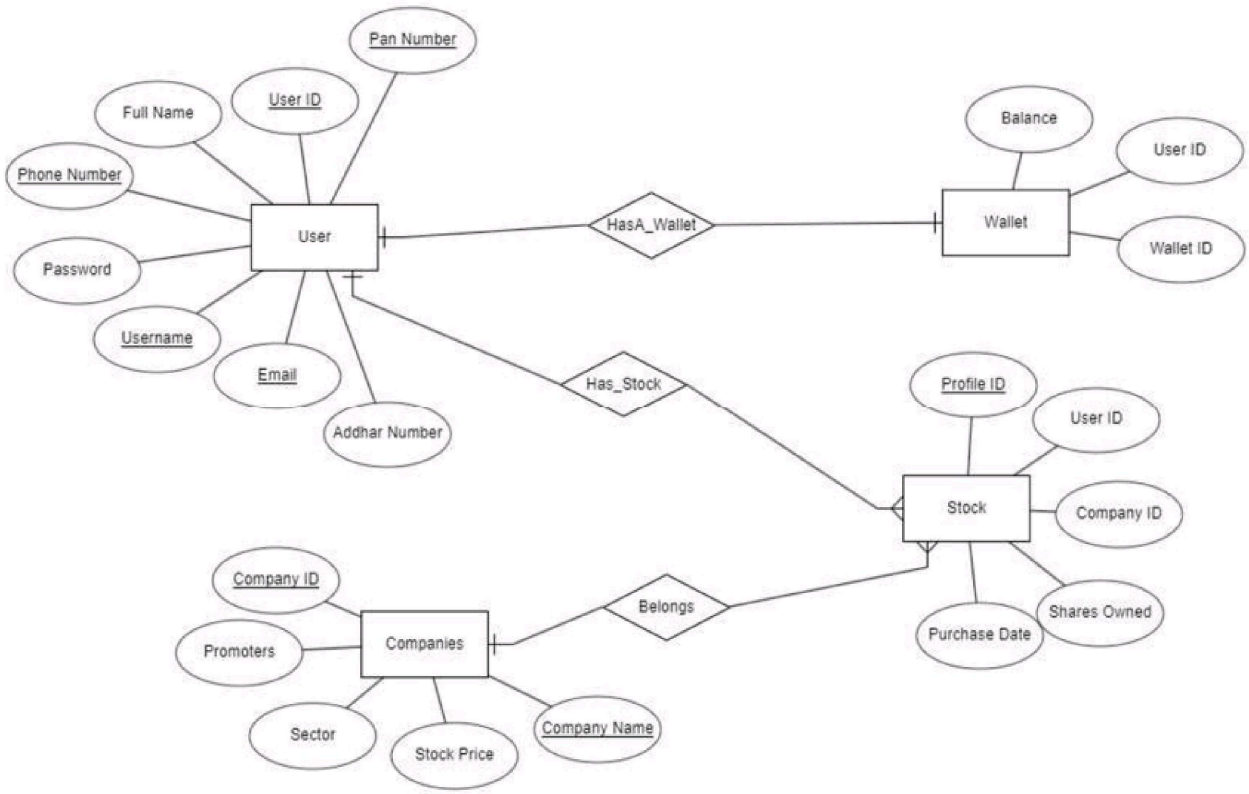


Fig 3.1 ER Model of stock market management System

Fig 3.1 illustrates the Entity-Relationship (ER) model for the Stock Market Management System. The model comprises core entities such as User, Wallet, Stock, Company, and Holdings, each defined by their attributes and relationships.

The User entity stores personal and authentication details, including *User ID, Full Name, PAN Number, Phone Number, Email*, and *Password*. Users are linked to a Wallet through the *HasA_Wallet* relationship, where each user has a unique wallet (one-to-one) containing *Wallet ID, Balance*, and a foreign key to the user (*User ID*).

The Stock entity tracks market data such as *Stock Price, Shares Owned*, and *Corporate Date*. Stocks are associated with a Company (via the *Belongs* relationship), where each company (*Company ID, Company Name*) can issue multiple stocks (one-to-many).

3.2 ARCHITECTURE DIAGRAM

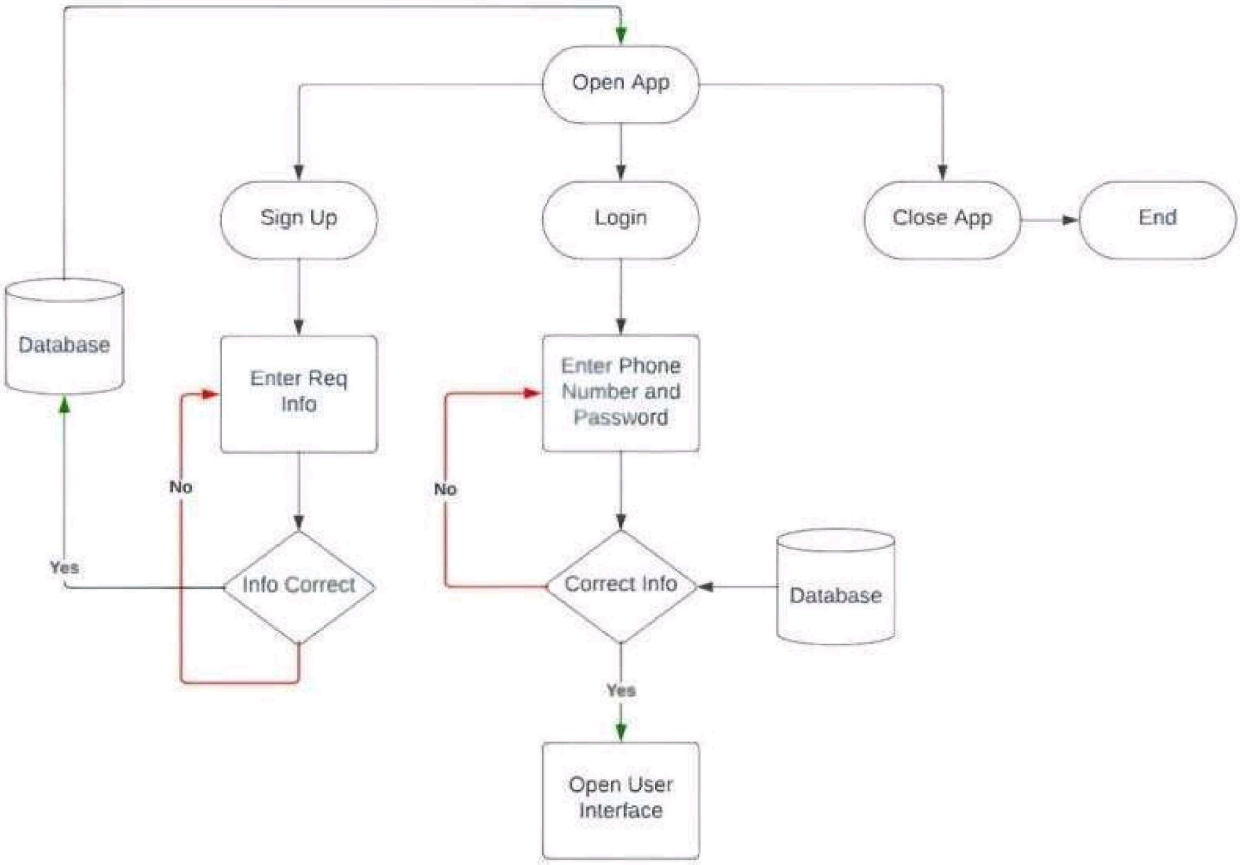


Fig 3.2 Architecture Diagram of stock market management system

Fig 3.2 outlines the layered architecture of the Stock Market Management System:

1. Client Layer: User interface built with Python's *Custom Tkinter* for desktop access.
2. Application Layer: Handles authentication, session management, and input validation.
3. Business Logic Layer: Executes trades, updates real-time stock data, and triggers alerts.
4. Database Layer: MySQL tables (*Users, Stocks, Transactions, Wallets*) ensure secure, ACID-compliant data storage.

3.3 Order workflow

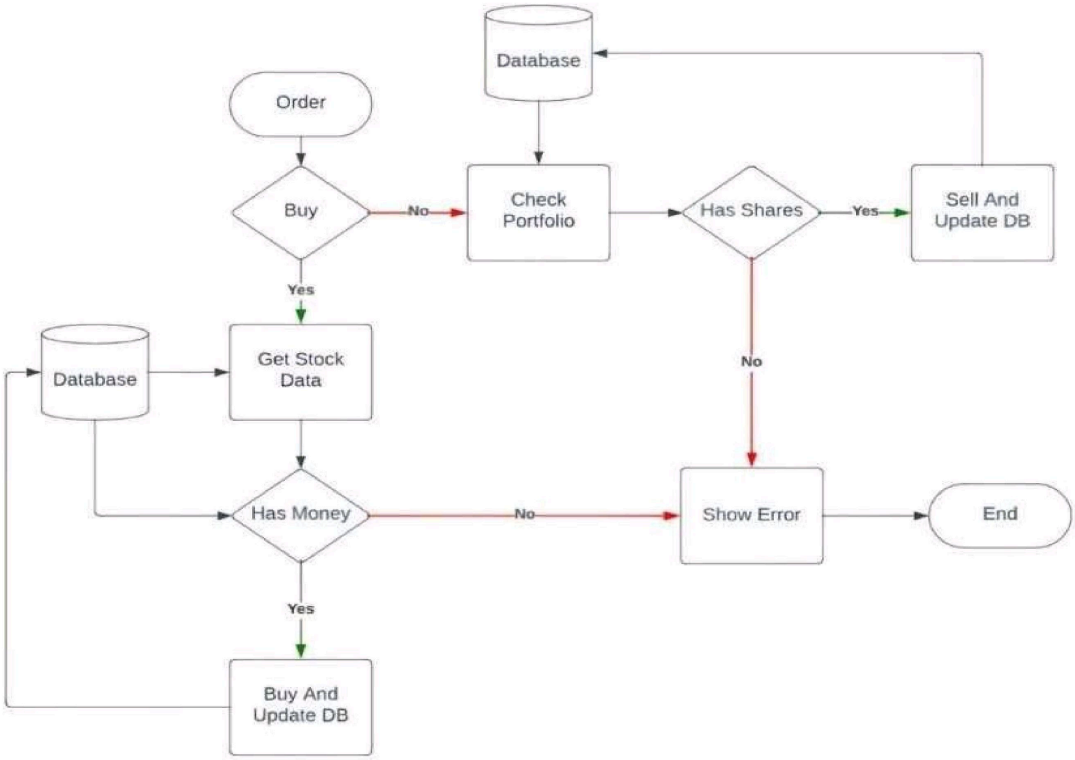
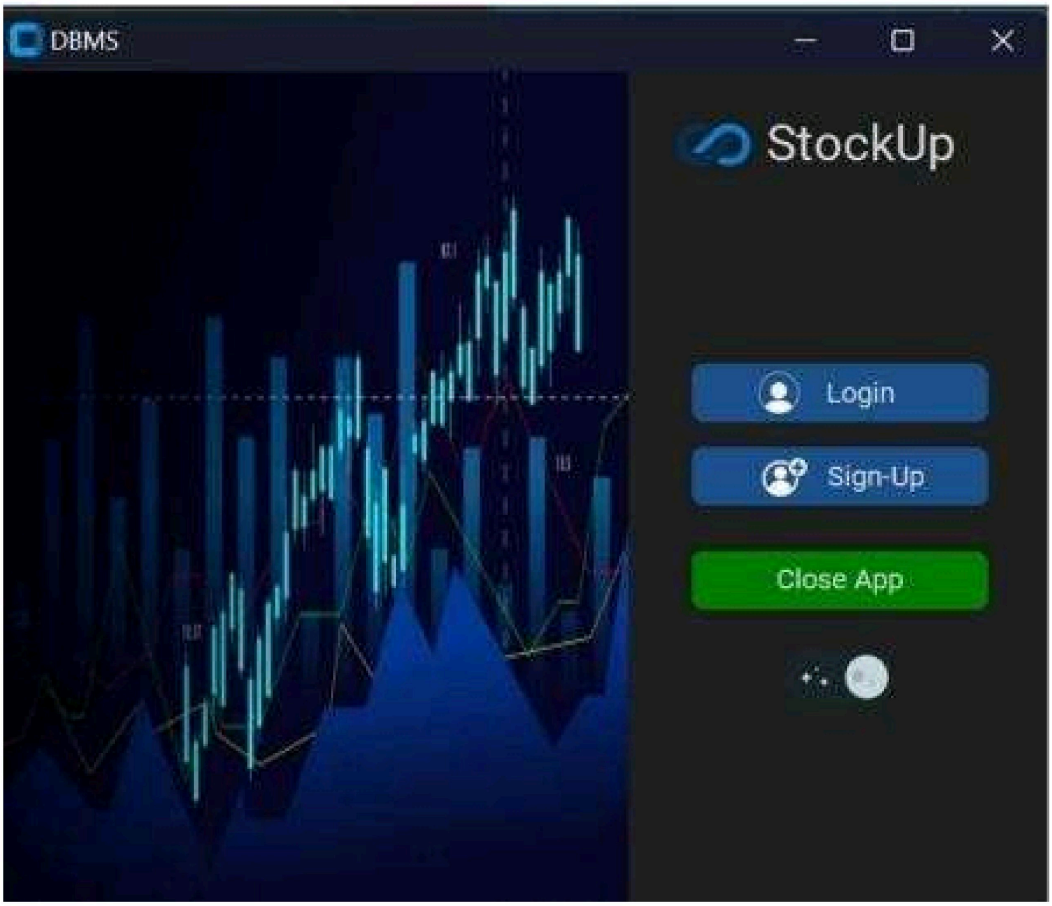


Fig 3.3 order workflow of stock market management system

Fig 3.3 outlines the Order Workflow for stock trading in the system:

1. A user initiates a Buy order, triggering a check for sufficient shares in their portfolio and available funds in their wallet.
2. If either check fails (*No shares* or *insufficient balance*), the system displays an error and terminates the request.
3. If validated (*Yes*), the system fetches real-time stock data, executes the trade, and updates the database (portfolio shares, wallet balance, transaction log).
4. The workflow ensures ACID compliance (atomicity, consistency) and real-time synchronization of user holdings and market data.

3.4 FRONT END DESIGN



Landing Page of smms

The landing page of the StockUp application presents a modern and sleek interface, featuring a financial graph visualization on the left and user options on the right, including Login, Sign-Up, and Close App buttons, providing users with a seamless entry point into the stock trading platform.

4 PROPOSED METHODOLOGY

The StockUp application is divided into multiple modules, each tailored to address the specific functionalities required by different user roles: User (Investor) and System Administrator. These modules are developed using Python as the core programming language, with CustomTkinter used for building the graphical user interface. The backend processes and data management are handled via a MySQL database, ensuring secure and efficient handling of financial and user-related data.

Each module interacts with the centralized database to facilitate real-time stock trading, secure wallet transactions, user portfolio tracking, and stock market data visualization. The modular design ensures smooth navigation, role-based access, and efficient data processing throughout the application.

4.1 MODULES DESCRIPTION

1. Authentication Module

This module manages secure user login and authentication for the StockUp system. Users must log in using their registered Phone Number and Password, which are verified against the records stored in the MySQL database. Only authenticated users can access their portfolio, wallet, and trading features.

- Frontend: The login interface is created using CustomTkinter in Python, offering a clean and interactive GUI experience.
- Backend: User credentials are validated through MySQL queries. Upon successful authentication, the user is redirected to their personalized dashboard.
- Security: Passwords are stored securely using hashing techniques to ensure data protection and user privacy.

2. user Module

This module enables registered users to explore stock information and manage their profiles. Features include:

- View Market Trends: Displays real-time or simulated stock trends using graphical representation.

- Account Management: Users can update personal information, reset passwords, and manage their wallet.
- Dashboard Access: Users are provided a personalized dashboard that summarizes their trading activity and holdings.

All features dynamically interact with the backend to fetch and update data from the MySQL database.

3. Trading Module

This module allows users to perform trading operations such as buying and selling stocks.

- Buy/Sell Stocks: Users can select available stocks and place buy or sell orders.
- Transaction Validation: Ensures that the user has sufficient balance or stock holdings before executing a transaction.
- Order History: Maintains a detailed log of all buy/sell transactions for each user.

Transactions update the portfolio and wallet records accordingly in the database.

4. Admin Module

The Admin Module serves as the backbone of the StockUp system, offering complete administrative control over all core functionalities. It allows administrators to manage users, stocks, transactions, and system settings to ensure smooth platform operations and maintain data integrity.

- User Management: Admins can create, update, or delete user accounts. This includes managing user roles, verifying KYC details, resetting passwords, and monitoring user activity to ensure compliance with platform policies.
- Stock Management: This section enables admins to manage the list of stocks available for trading. Admins can add new company listings, update existing stock prices, and deactivate or remove companies that are no longer active or have been delisted.
- Monitor Transactions: Admins have access to a comprehensive view of all financial transactions performed on the platform. They can track buy/sell activity, review transaction logs, and generate real-time or historical reports for auditing and regulatory purposes.

- Wallet Oversight: Admins can monitor user wallet balances, set minimum and maximum wallet limits, and flag suspicious or unusual financial activity. This helps maintain financial transparency and prevent misuse of the wallet system.
- System Configuration: Admins can configure key system settings, including trading hours, maintenance notifications, and platform-wide announcements. They also oversee database health, backup routines, and performance optimization tasks.

This module is highly secure and interacts with all major database tables, performing most of the critical write operations necessary for platform management and maintenance. It plays a pivotal role in ensuring the reliability, security, and efficiency of the StockUp system.

5. Wallet Module

The Wallet Module plays a vital role in the StockUp system by allowing users to manage the funds they use for trading. It ensures that financial transactions are smooth, secure, and well-documented.

- Add/Withdraw Funds: Users can easily add money to their virtual wallet using predefined methods. Similarly, they can withdraw unused or excess funds. The system ensures these operations are user-friendly and processed in real-time.
- Balance Validation: Before confirming any stock purchase, the system checks if the user has enough balance in their wallet. This prevents failed trades and promotes responsible usage of funds.
- Transaction Logs: Every wallet activity—like fund additions, deductions for trades, or withdrawals—is recorded and displayed in a user-friendly format. Users can view their wallet history at any time for transparency.
- Integration: The Wallet Module works closely with the Trading Module. Whenever a user places a buy or sell order, the wallet is automatically updated. It also connects with the Admin Module for wallet monitoring and user activity review.

This module ensures that users can manage their funds efficiently while maintaining the integrity and smooth functioning of the overall system.

4.1 Database Connectivity

The backend of the stock market project is built using Node.js with the Express.js framework. It connects to a MySQL database using the mysql2 npm package (or mysql npm package, depending on your preference). The MySQL database serves as the core of the application, storing critical records such as user accounts, stock market data, transactions, user portfolios, and more.

In Node.js, the database connectivity is established by creating a connection object using the mysql2 package. Below is an example of how the backend connects to the MySQL database:

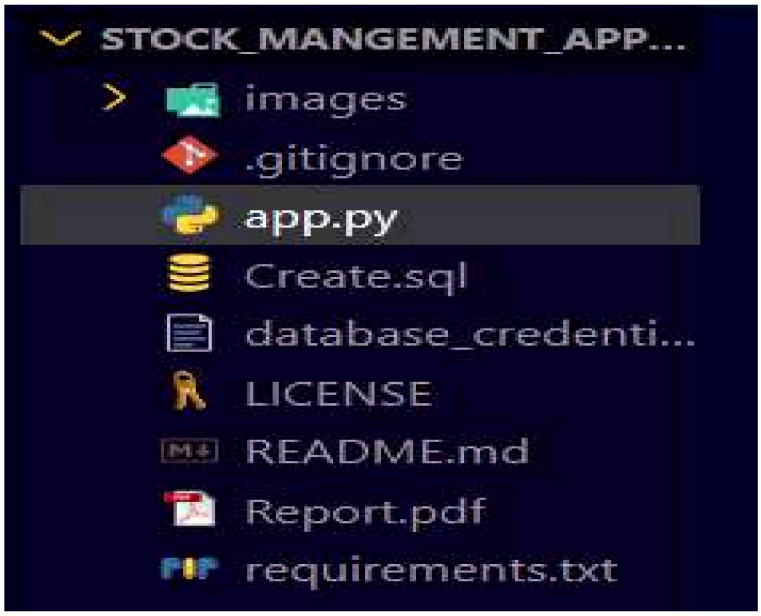
```
 // Database Connection
const db = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: '*******',
    database: 'Stock_Market',
    port: 3306
});

// Connect to the database
db.connect((err) => {
    if (err) {
        console.error('+ MySQL Connection Failed:', err);
        return;
    }
    console.log(' Connected to MySQL');
});


// Export the database connection object for use in other files
module.exports = db;
```

This connection is imported into route handler files where SQL queries are written to perform operations such as fetching available stocks, inserting new transactions, or updating stock prices.

5 IMPLEMENTATION



1. Create. SQL

```sql
CREATE DATABASE stock_market_db;

▷Run
USE stock_market_db;

▷Run | Select
CREATE TABLE users (
    user_id INT NOT NULL AUTO_INCREMENT,
    full_name VARCHAR(255) NOT NULL,
    pan_number VARCHAR(10) NOT NULL UNIQUE,
    aadhar_number VARCHAR(12) NOT NULL UNIQUE,
    phone_number VARCHAR(10) NOT NULL,
    password VARCHAR(255) NOT NULL,
    PRIMARY KEY (user_id)
);

▷Run | Select
CREATE TABLE wallets (
    wallet_id INT NOT NULL AUTO_INCREMENT,
    user_id INT NOT NULL,
    balance DECIMAL(10,2) NOT NULL,
    PRIMARY KEY (wallet_id),
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);

▷Run | Select
CREATE TABLE companies (
    company_id INT NOT NULL AUTO_INCREMENT,
    company_name VARCHAR(255) NOT NULL,
    sector VARCHAR(255) NOT NULL,
    stock_price DECIMAL(10,2) NOT NULL,
    PRIMARY KEY (company_id)
);
```

```sql
▷Run | Select
CREATE TABLE shares (
  share_id INT NOT NULL AUTO_INCREMENT,
  user_id INT NOT NULL,
  company_id INT NOT NULL,
  shares_owned INT NOT NULL,
  PRIMARY KEY (share_id),
  FOREIGN KEY (user_id) REFERENCES users(user_id),
  FOREIGN KEY (company_id) REFERENCES companies(company_id)
);

-- Insert data for 5 users
▷Run | Select
INSERT INTO users (full_name, pan_number, aadhar_number, phone_number, password)
VALUES
('Amit Kumar', 'ABCDEF12', '123456789012', '9876543210', 'password1'),
('Sneha Gupta', 'FGHIJ5678', '345678901234', '8765432109', 'password2'),
('Rajesh Verma', 'LMNOP1234', '5678901234', '7654321098', 'password3'),
('Priya Sharma', 'UVWXYZ5678', '7890123456', '6543210987', 'password4'),
('Ravi Singh', 'PQRSTU1234', '8900234567', '5432109876', 'password5');

-- Insert balance data for 5 users
▷Run | Select
INSERT INTO wallets (user_id, balance)
VALUES
(1, 10000.00), -- User 1 (Amit Kumar)
(2, 9500.00),  -- User 2 (Sneha Gupta)
(3, 11000.00), -- User 3 (Rajesh Verma)
(4, 8500.00),  -- User 4 (Priya Sharma)
(5, 10250.00); -- User 5 (Ravi Singh);
```

```sql
INSERT INTO companies (company_name, sector, stock_price)
VALUES
  ('Tata Motors', 'Automobile', 100.00),
  ('Infosys', 'Information Technology', 50.00),
  ('Reliance Industries', 'Conglomerate', 75.00),
  ('ICICI Bank', 'Banking', 45.00),
  ('HDFC Ltd', 'Finance', 60.00);

-- User 1 (Amit Kumar)
▷Run | Select
INSERT INTO shares (user_id, company_id, shares_owned)
VALUES (1, 1, 50),
       (1, 2, 75),
       (1, 4, 30);

-- User 2 (Sneha Gupta)
▷Run | Select
INSERT INTO shares (user_id, company_id, shares_owned)
VALUES (2, 2, 60),
       (2, 3, 90),
       (2, 5, 40);

-- User 3 (Rajesh Verma)
▷Run | Select
INSERT INTO shares (user_id, company_id, shares_owned)
VALUES (3, 1, 75),
       (3, 4, 20),
       (3, 5, 50);

-- User 4 (Priya Sharma)
▷Run | Select
INSERT INTO shares (user_id, company_id, shares_owned)
VALUES (4, 3, 100),
       (4, 5, 70);

-- User 5 (Ravi Singh)
▷Run | Select
INSERT INTO shares (user_id, company_id, shares_owned)
VALUES (5, 1, 40),
       (5, 2, 85),
       (5, 3, 60);
```

2. app.py

app.py > add_shares_to_portfolio

```python
import decimal
import re
from tkinter import messagebox

import mysql.connector
from PIL import Image
from customtkinter import *

print("""
Stock Market Management System

This Python program is a part of a stock market management system. It provides functionality for users to register,
log in, view their wallet balance, buy and sell shares of different companies, and manage their portfolio.
The program uses a graphical user interface (GUI) created using the 'customtkinter' library and displays images
using the Python Imaging Library (PIL). It also interacts with a MySQL database for user registration, login,
and managing user wallets and shares.

The main functionality of the program includes:
- User registration with details like full name, PAN card, Aadhar number, phone number, and initial balance.
- User login with phone number and password authentication.
- Display of the user's wallet balance.
- Buying and selling shares of different companies, where stock prices are fetched from a database.
- Managing the user's portfolio, displaying the number of shares owned for each company.

The code is organized into functions that handle various tasks, such as connecting to the database, checking user
credentials, adding money to the wallet, buying and selling shares, and updating the user's portfolio.
The GUI is built using the 'customtkinter' library, allowing for user-friendly interaction.\n\n""")

print("""
Author: Abhijeet kumar
Roll No.: 147
Date: 15 Oct 2023
Subject: DBMS
Database: MySQL
""")

print("""
```

```
  ____          _                 ____ _          _        _
 / ___|        | |               / ___| |        | |      | |
| (__ _  _ ___| |_ __ _ _ __   | (__ | |_ _ _ _ _| |_ __  _| |
 \__ \| | | / __| __/ _ \ '_ ` _ \   \__ \| __/ _` | '_| __/ _ \/ _` |
 ___) | |_| \__ \ || __/ | | | | |  ___) | || (_| | | | || __/ (_| |
|____/ \__, |___/\__\___|_| |_| |_| |____/ \__\__,_|_|  \__\___|\__,_|
        __/ |
       |___/
""")

app = CTk()
app.geometry("500x400")
set_appearance_mode("dark")
set_default_color_theme("blue")
app.title("DBMS")
frame = CTkFrame(master=app)
img_label = CTkLabel(master=app)
wallet_frame = CTkFrame(master=app)
port = CTkFrame(master=app)
stock_frame = CTkFrame(master=app)


def connect_to_database():
    """
    Connect to the Stock Market Database.

    This function establishes a connection to a MySQL database named 'stock_market_db' using the provided credentials
    The connection is essential for performing database operations, such as retrieving data, updating records, and
    managing user information.

    Returns:
        mysql.connector.MySQLConnection: A connection object to the 'stock_market_db' database.

    Raises:
        MySQLConnectionError: If the connection to the database cannot be established.

    """
```

```python
def connect_to_database():
    try:
        # Read credentials from the file
        with open("database_credentials.txt", "r") as file:
            lines = file.readlines()

        credentials = {}
        for line in lines:
            key, value = line.strip().split(": ")
            credentials[key] = value

        # Ensure required keys exist
        required_keys = ["host", "user", "password", "database", "port"]
        for key in required_keys:
            if key not in credentials:
                raise KeyError(f"Missing '{key}' in database_credentials.txt")

        # Connect to MySQL
        connection = mysql.connector.connect(
            host=credentials["host"],  # Should be "127.0.0.1" or "localhost"
            user=credentials["user"],  # Should be "root"
            password=credentials["password"],  # Should match your MySQL root passwor
            database=credentials["database"],  # Your DB name
            port=int(credentials["port"])  # Port should be an integer (default 3306)
        )

        print("✅ Successfully connected to the database!")
        return connection

    except FileNotFoundError:
        print("❌ Database credentials file not found.")
        sys.exit()
    except KeyError as e:
        print(f"❌ Missing key in credentials file: {e}")
        sys.exit()
    except mysql.connector.Error as err:
        print(f"❌ Failed to connect to the database: {err}")
```

```python
def buy_shares(user_id, company):
    global wallet_frame
    global port
    global stock_frame
    ids = {"TATA": "1", "INFO": "2", "RELIANCE": "3", "ICICI": "4", "HDFC": "5"}
    company_id = ids[company]
    shares_to_buy = CTkInputDialog(text="Enter Number of Shares To Buy", title=company).get_input()
    if shares_to_buy:
        shares_to_buy = int(shares_to_buy)
    else:
        return

    stock_price = get_stock_price(company_id)
    purchase_amount = shares_to_buy * float(stock_price)
    wallet_balance = float(get_wallet_balance(user_id))

    current_shares = get_user_shares(user_id, company_id)

    if wallet_balance >= purchase_amount:
        new_balance = wallet_balance - purchase_amount
        update_wallet_balance(user_id, new_balance)

        if current_shares > 0:
            # The user already owns shares in this company; update the share count.
            new_shares = current_shares + shares_to_buy
            update_user_shares(user_id, company_id, new_shares)
        else:
            # The user does not own shares in this company; insert a new entry.
            add_shares_to_portfolio(user_id, company_id, shares_to_buy)

        messagebox.showinfo("Done", f"{company} Shares Bought Successfully")
        frame_count = 0
        frames_to_destroy = []

        for widget in app.winfo_children():
            if "frame" in widget.winfo_name():
```

```python
def logged_in(userid):
    global frame
    global img_label
    global wallet_frame
    global port
    global stock_frame
    frame.destroy()
    img_label.destroy()
    balance = get_wallet_balance(userid)
    wallet_frame = CTkFrame(master=app, width=220, height=50)
    wallet_frame.grid(row=0, column=0, padx=10, pady=10)

    bal = CTkLabel(master=wallet_frame, text="Balance: ₹", font=("Roboto", 15))
    bal.place(x=20, y=10)
    if balance >= 1000000:
        display = str(round(balance / 1000000, 2)) + " M"
    elif balance >= 1000:
        display = str(round(balance / 1000, 2)) + " K"
    else:
        display = str(balance)
    amt = CTkLabel(master=wallet_frame, text=display, font=("Roboto", 15))
    amt.place(x=90, y=10)
    btn = CTkButton(master=wallet_frame, text="Add +", width=60, height=20, command=lambda: add(userid))
    btn.place(x=150, y=13)

    port = CTkFrame(master=app, width=220, height=320)
    port.grid(row=1, column=0, padx=10, pady=(0, 10))
    txt = CTkLabel(master=port, text="Portfolio", font=("Roboto", 24))
    txt.place(x=60, y=5)
    shares_data = check_shares(userid)
    for i, share in enumerate(shares_data):
        suitcase = CTkImage(dark_image=Image.open("./images/bagw.png"), light_image=Image.open("./images/bagb.png"),
                            size=(20, 15))
        bag = CTkLabel(master=port, text="", image=suitcase)
        bag.place(x=8, y=58 + 40 * i)
        stock_label = CTkLabel(master=port, text=f"{share} : {shares_data[share]} shares", font=("Roboto", 16))
```

```python
def login():
    """
    Create the login interface and handle login functionality.

    This function creates the login user interface, allowing users to enter their phone number and password. When the
    "Login" button is clicked, it calls the 'login_check' function to validate the credentials and perform the login.

    Returns:
        None
    """
    global frame
    global img_label
    frame.destroy()
    frame = CTkFrame(master=app, width=200, height=400)
    frame.grid(row=0, column=1)

    name = CTkLabel(master=frame, text="StockUp", font=("Roboto", 24))
    name.place(x=65, y=20)
    img_logo = CTkImage(Image.open("./images/logo.png"), size=(50, 50))
    logo = CTkLabel(master=frame, image=img_logo, text="")
    logo.place(x=15, y=10)
    # Phone Number Entry
    phone_entry = CTkEntry(master=frame, placeholder_text="Phone Number")
    phone_entry.place(x=30, y=150)

    password_entry = CTkEntry(master=frame, placeholder_text="Password", show="*")
    password_entry.place(x=30, y=190)

    login_button = CTkButton(master=frame, text="Login",
                             command=lambda: login_check(phone_entry.get(), password_entry.get()))
    login_button.place(x=30, y=230)

    back = CTkButton(master=frame, text="Back",
                     command=backfnc, hover_color="orange")
    back.place(x=30, y=270)
```

```python
def page1():
    """
    Create the initial page of the application.

    This function sets up the initial page of the application, which includes a background image, the application logo,
    and buttons to either log in or sign up.

    Returns:
        None
    """
    global frame
    global img_label
    frame.destroy()
    img_label.destroy()
    wallet_frame.destroy()
    port.destroy()
    stock_frame.destroy()
    img = CTkImage(Image.open("./images/login.jpg"), size=(300, 400))
    img_label = CTkLabel(master=app, image=img, text="")
    img_label.grid(row=0, column=0)

    frame = CTkFrame(master=app, width=200, height=400)
    frame.grid(row=0, column=1)

    img_logo = CTkImage(Image.open("./images/logo.png"), size=(50, 50))
    logo = CTkLabel(master=frame, image=img_logo, text="")
    logo.place(x=15, y=10)

    name = CTkLabel(master=frame, text="StockUp", font=("Roboto", 24))
    name.place(x=65, y=20)
    l_img = CTkImage(Image.open("./images/loginImg.png"), size=(21, 21))
    login_button = CTkButton(master=frame, text=" Login    ", command=login, image=l_img)
    login_button.place(x=30, y=140)
    login_button.place(x=30, y=140)
    s_img = CTkImage(Image.open("./images/signup.png"), size=(25, 21))
    sign_btn = CTkButton(master=frame, text="Sign-Up", command=signup, image=s_img)
    sign_btn.place(x=30, y=180)

    close_btn = CTkButton(master=frame, text="Close App", command=close, fg_color="green", hover_color="red")
    close_btn.place(x=30, y=230)

    t = CTkImage(dark_image=Image.open("./images/dark.png"), light_image=Image.open("./images/light.png"),
                 size=(66, 34))
    theme = CTkButton(master=frame, text="", image=t, fg_color="transparent", hover=False, width=66, height=34,
                      command=change_theme)
    theme.place(x=60, y=270)


page1()
# view()
app.mainloop()
print("""
App Closed...
""")
""" Code Ends """
```

6. RESULT

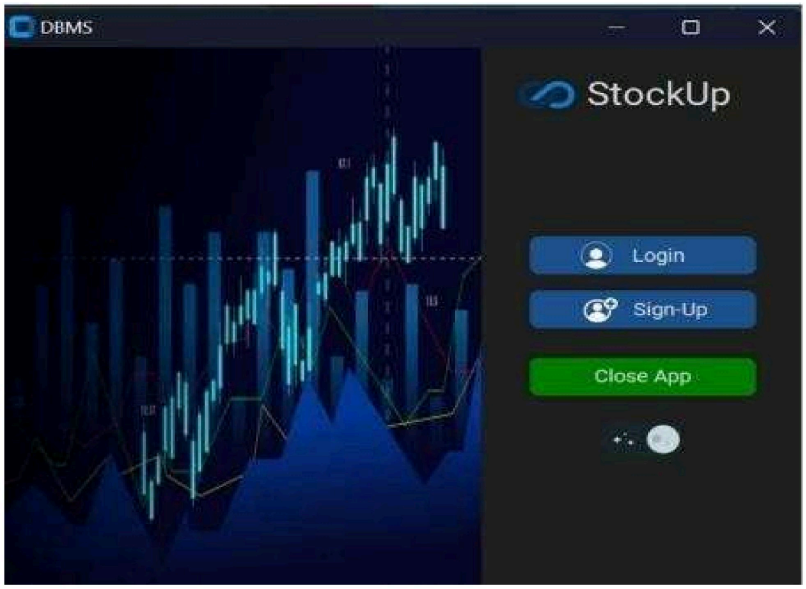


Fig 6.1 Landing PAGE (Dark Background) of smms

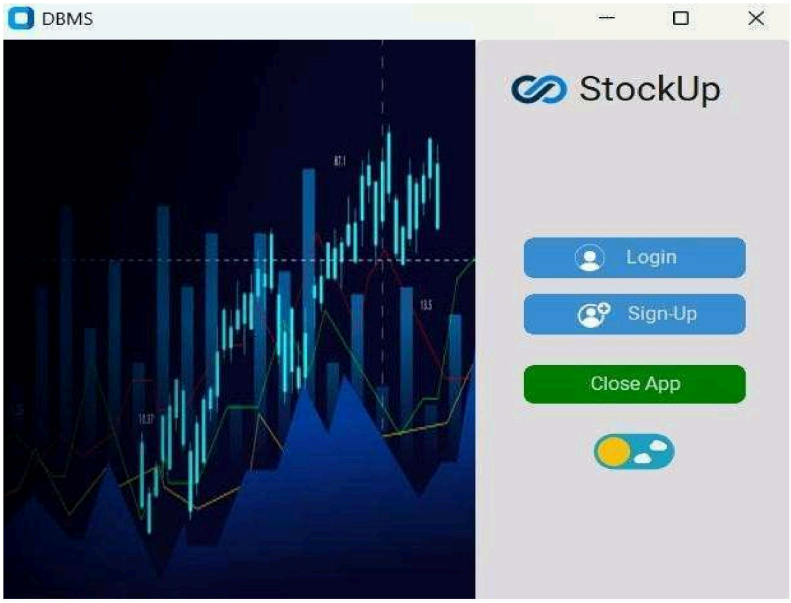


Fig 6.2 Landing PAGE (White Backgound) of smms

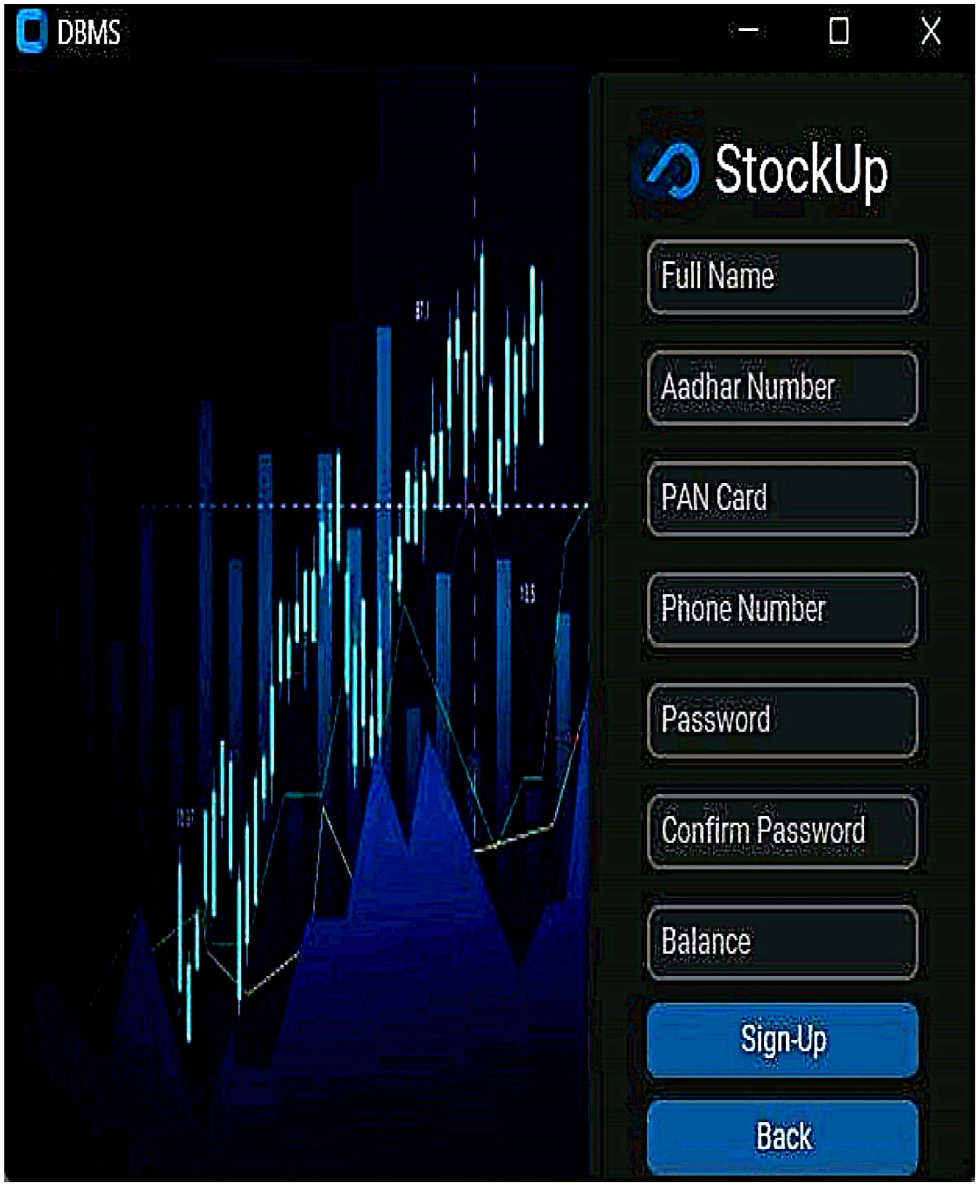


Fig 6.3 SIGNUP PAGE OF SMMS

Fig 6.4 LOGIN PAGE OF SMMS

Fig 6.5 USER'S PAGE OF SMMS

Fig 6.6 ORDER INTERFACE OF SMMS

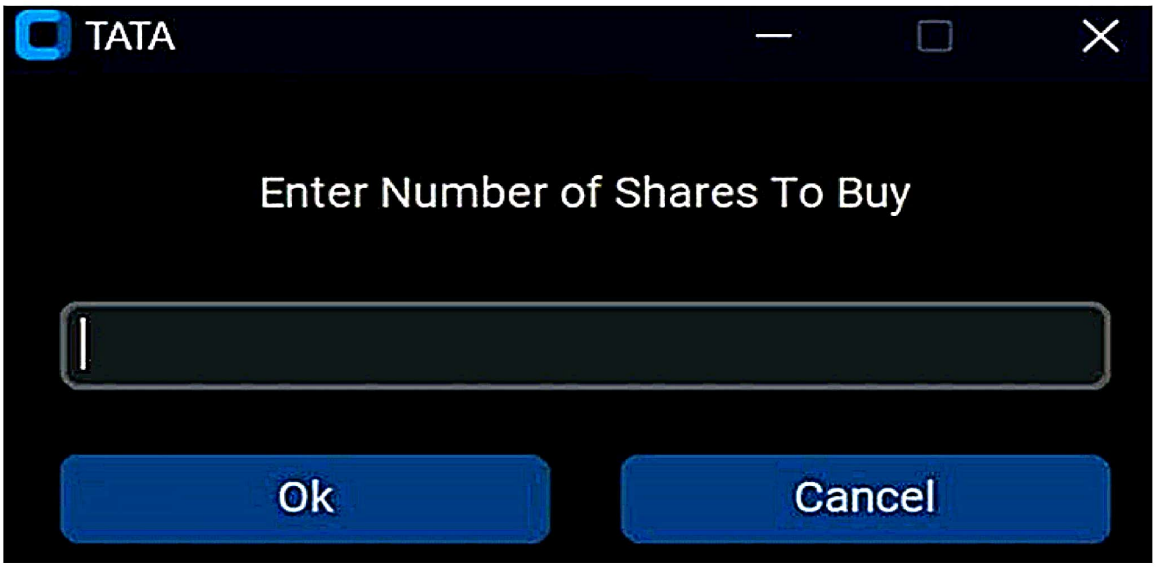


Fig 6.7 BUY AND SELL OF SMMS

6. CONCLUSION

The Stock Market Management System revolutionizes stock trading by unifying investors, traders, and administrators into a single, secure platform. It eliminates manual inefficiencies, enhances accuracy, and delivers a frictionless experience for all users.For investors, the system provides an intuitive interface to track real-time market trends, execute trades instantly, and manage portfolios with ease. It simplifies complex financial decisions, empowering users to invest confidently without the stress of fragmented tools.Administrators gain full oversight to monitor transactions, manage stock listings, and ensure compliance. Automated workflows reduce paperwork, minimize errors, and free up time for strategic decision-making.Built with Python's Custom Tkinter for a dynamic, user-friendly GUI and MySQL for robust backend operations, the system ensures secure data handling, real-time updates, and seamless scalability. Advanced encryption safeguards sensitive financial data, while relational databases maintain transactional integrity.This project not only modernizes stock market interactions but also lays the groundwork for future innovations—AI-driven predictions, mobile integration, or advanced analytics. It's a leap toward a smarter, more accessible financial ecosystem where users trade with precision, transparency, and control.

7.REFERENCES

- https://developer.mozilla.org/en-US/docs/Glossary/SQL
- https://code.visualstudio.com/
- Constantine A. Balanis, Antenna Theory: Analysis and Design, 2nd Edition; John Wiley Sons Inc. New York, 2001,pp12-130
- Inder J Gupta, Aharon A Ksienski, "Effect of Mutual Coupling on the Performance of Adaptive Arrays" IEEE Trans. Antennas Propagat., vol. AP-31, no.5, pp.785-791, Sept. 1993
- M. Duncan. "Engineering Concepts on Ice. Internet: www.iceengg.edu/staff.html, Oct.25, 2000 [Nov. 29, 2003].
- G. Brandli and M. Dick, "Alternating current fed power supply," U.S.Patent 4 084 217,Nov. 4, 1978.