Financial Asset Management System

DBMS – SE Project 2024

Aditya Hriday Rath	PES1UG22AM013
Priyansh Surana	PES1UG22AM013

Software Requirements Specification
for
Investment Portfolio Optimizer

Prepared by:
Aditya Hriday Rath (PES1UG22AM013)
Priaynsh Surana (PES1UG22AM905)

November 20, 2024

Contents

1 Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) is to define the requirements for the Investment Portfolio Optimizer (IPO). The system will manage user portfolios, assess risk, integrate with market data sources, and provide tools for goal-based financial planning. This document is intended to be used by developers, project managers, and stakeholders to ensure that the system meets all specified requirements.

1.2 Scope

The Investment Portfolio Optimizer will support the following functionalities:

- Portfolio Management
- Risk Assessment and Optimization
- Goal-Based Planning
- Real-Time Market Data Integration
- Reporting and Analysis

1.3 Definitions, Acronyms, and Abbreviations

- **IPO**: Investment Portfolio Optimizer
- **API**: Application Programming Interface
- **DBMS**: Database Management System

1.4 References

- Wealthfront Website: `https://www.wealthfront.com`
- IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications

2 Overall Description

2.1 Product Perspective

The IPO is a web-based application designed to help users manage and optimize their investment portfolios efficiently. The system integrates with external financial data sources and provides real-time data for analysis and decision-making.

2.2 Product Functions

- **Portfolio Management**: Users can create, view, and manage investment portfolios.
- **Risk Assessment and Optimization**: Tools to evaluate portfolio risk and suggest optimizations.
- **Goal-Based Planning**: Users can set financial goals and track progress.
- **Real-Time Market Data Integration**: Integration with external APIs for up-to-date market information.
- **Reporting and Analysis**: Generate detailed reports and analysis of portfolio performance.

2.3 User Classes and Characteristics

- **Admin**: Manages system settings and user accounts.
- **Investors**: Manage personal investment portfolios and receive optimization advice.
- **Advisors**: Manage multiple client portfolios and provide financial planning services.
- **Institutional Users**: Manage large-scale portfolios and require advanced analytics.

2.4 Operating Environment

- **Platform**: Web-based application, accessible via modern web browsers.
- **Database**: MySQL relational DBMS.

2.5 Design and Implementation Constraints

- The system must comply with financial regulations regarding data privacy and security.
- High availability is required to ensure continuous operation during market hours.

2.6 Assumptions and Dependencies

- Reliable internet access is available to all users.
- The system will integrate with existing financial APIs for data retrieval.

3 System Architecture

3.1 Class Diagram

The following class diagram represents the core components of the Investment Portfolio Optimizer system, including the relationships between users, portfolios, assets, transactions, market data, and financial goals.

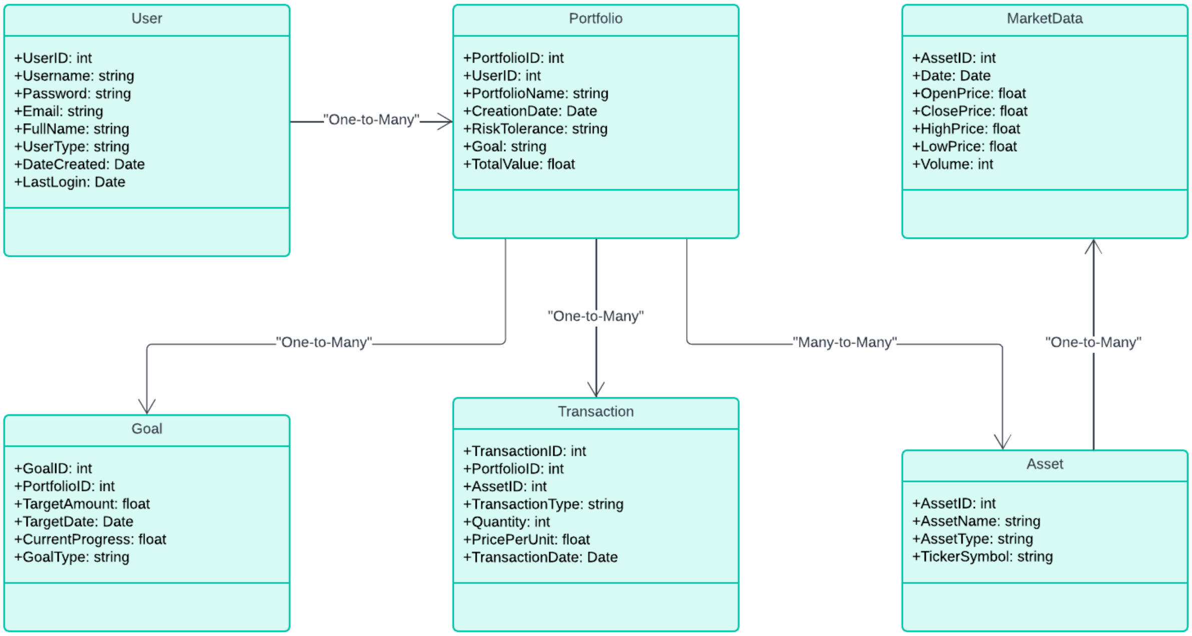


Figure 1: Class Diagram for Investment Portfolio Optimizer

4 Functional Requirements

4.1 Portfolio Management

- **FR1.1**: The system shall allow users to create, update, and view portfolios.
- **FR1.2**: The system shall calculate and display the current value of each portfolio.

4.2 Risk Assessment and Optimization

- **FR2.1**: The system shall assess the risk level of portfolios based on asset allocation.
- **FR2.2**: The system shall provide optimization recommendations to minimize risk and maximize returns.

4.3 Goal-Based Planning

- **FR3.1**: The system shall allow users to set financial goals linked to specific portfolios.
- **FR3.2**: The system shall track and report progress towards achieving financial goals.

4.4 Real-Time Market Data Integration

- **FR4.1**: The system shall integrate with external APIs to fetch real-time market data.
- **FR4.2**: The system shall update asset values and notify users of significant market changes.

4.5 Reporting and Analysis

- **FR5.1**: The system shall generate detailed reports on portfolio performance.
- **FR5.2**: The system shall provide analysis tools for users to evaluate their investment strategies.

5 Non-Functional Requirements

5.1 Performance

- **NFR1.1**: The system shall support up to 1,000 concurrent users without performance degradation.

5.2 Security

- **NFR2.1**: The system shall implement secure authentication and authorization mechanisms.
- **NFR2.2**: The system shall encrypt sensitive data, including user credentials and financial information.

5.3 Usability

- **NFR3.1**: The system shall provide an intuitive user interface that is easy to navigate.

5.4 Reliability

- **NFR4.1**: The system shall maintain 99.9

5.5 Maintainability

- **NFR5.1**: The system shall be designed for easy updates and maintenance, with comprehensive documentation.

6 Additional Considerations

6.1 Scalability Considerations

The system architecture shall be designed to accommodate future growth, ensuring that additional users, portfolios, and data can be managed without significant changes to the existing structure.

6.2 Future Enhancements

The system may be expanded in future versions to include advanced features such as tax-loss harvesting, socially responsible investing (SRI), and integration with other financial tools and platforms. These enhancements will be considered based on user feedback and market demands.

Project Plan: Financial Asset Management System

Aditya HR
PES1UG22AM013

Priyansh Surana
PES1UG22AM905

1 Lifecycle Model

Chosen Lifecycle: Agile Scrum

We have chosen the Agile Scrum model for the execution of our project for the following reasons:

- **Short Iterations (Sprints)**: Agile allows us to break down our project into manageable 1-week sprints. Approximating a 45-day timeline, this ensures regular deliverables.
- **Flexibility for Changes**: Agile is flexible and allows us to adapt to changing requirements or address issues during the project.
- **Team Size**: With a small team of 2, Agile's focus on collaboration, daily updates, and iterative progress suits our workflow.
- **Time-boxing and Continuous Delivery**: Agile ensures we focus on prioritized features and deliver functional increments regularly.

2 Tools Used Throughout the Lifecycle

The following tools will be used during the project lifecycle:

- **Planning Tool: Trello**
  Trello is chosen due to its simplicity and ease of use, making it appropriate for a two-member team with a 45-day deadline.
- **Design Tool: GoodNotes 5**
  GoodNotes 5 will be used for designing diagrams and notes during brainstorming sessions.
- **Version Control: Git and GitHub**
  We will use Git and GitHub for version control and collaborative development.
- **Development Tool: VSCode**
  Visual Studio Code (VSCode) is chosen as the integrated development environment (IDE) for frontend and backend development.

- **Bug Tracking: GitHub Issues**
  GitHub Issues will be used to track bugs and manage the development tasks.
- **Testing Tool: Postman**
  We will use Postman for testing the queries to our api service.

3 Deliverables: Reuse vs Build Components

- **Database**: We will build the database from scratch using a relational schema derived from the ER diagram.
- **Frontend**: We will use React to build the frontend from scratch.
- **Backend**: Python will be used to develop the backend logic, with APIs implemented using a lightweight framework like Flask.

4 Work Breakdown Structure (WBS)

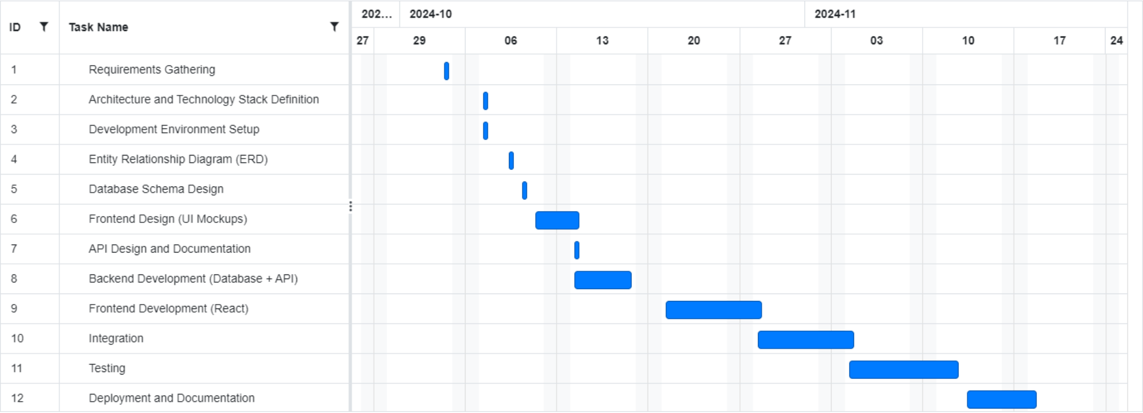
Stage/Task	**Start Date**	**End Date**	**Duration (Days)**
Requirements Gathering	Day 1	Day 1	1
Architecture and Technology Stack Definition	Day 2	Day 2	1
Development Environment Setup	Day 3	Day 3	1
Entity Relationship Diagram (ERD)	Day 6	Day 6	1
Database Schema Design	Day 7	Day 7	1
Frontend Design (UI Mockups)	Day 8	Day 9	2
API Design and Documentation	Day 9	Day 10	2
Backend Development (Database + API)	Day 11	Day 15	5
Frontend Development (React)	Day 16	Day 24	9
Integration	Day 25	Day 30	6
Testing	Day 31	Day 40	10
Deployment and Documentation	Day 41	Day 45	5

5 Effort Estimation and Gantt Chart

- **Stage 1 (Requirements and Setup)**: 3 days × 6 hours = 18 hours per person
- **Stage 2 (Design)**: 5 days × 6 hours = 30 hours per person
- **Stage 3 (Development)**: 20 days × 6 hours = 120 hours per person
- **Stage 4 (Testing)**: 10 days × 6 hours = 60 hours per person

- **Stage 5 (Deployment and Documentation)**: 5 days × 6 hours = 30 hours per person

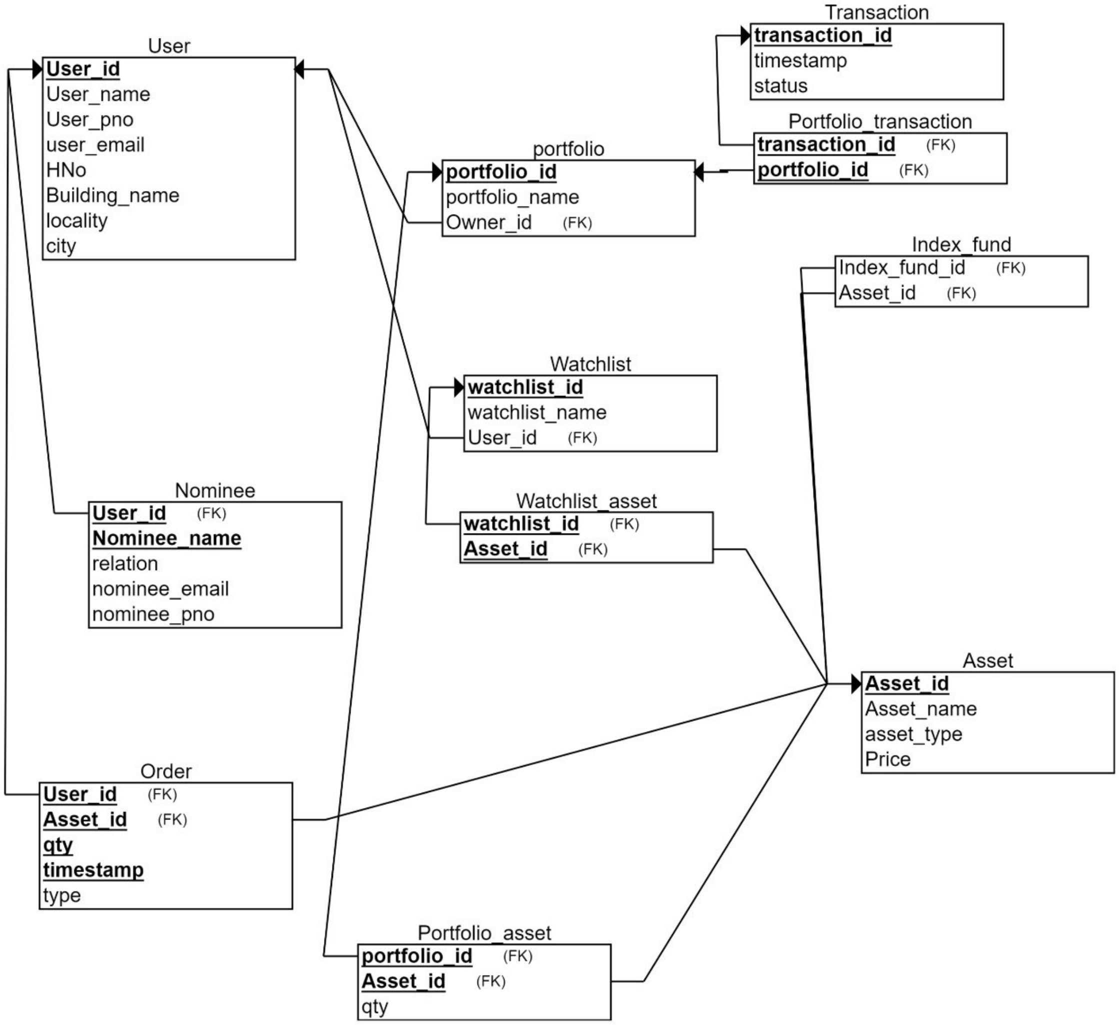
Total Effort: 1.58 person-months per team member



ID	Task Name
1	Requirements Gathering
2	Architecture and Technology Stack Definition
3	Development Environment Setup
4	Entity Relationship Diagram (ERD)
5	Database Schema Design
6	Frontend Design (UI Mockups)
7	API Design and Documentation
8	Backend Development (Database + API)
9	Frontend Development (React)
10	Integration
11	Testing
12	Deployment and Documentation

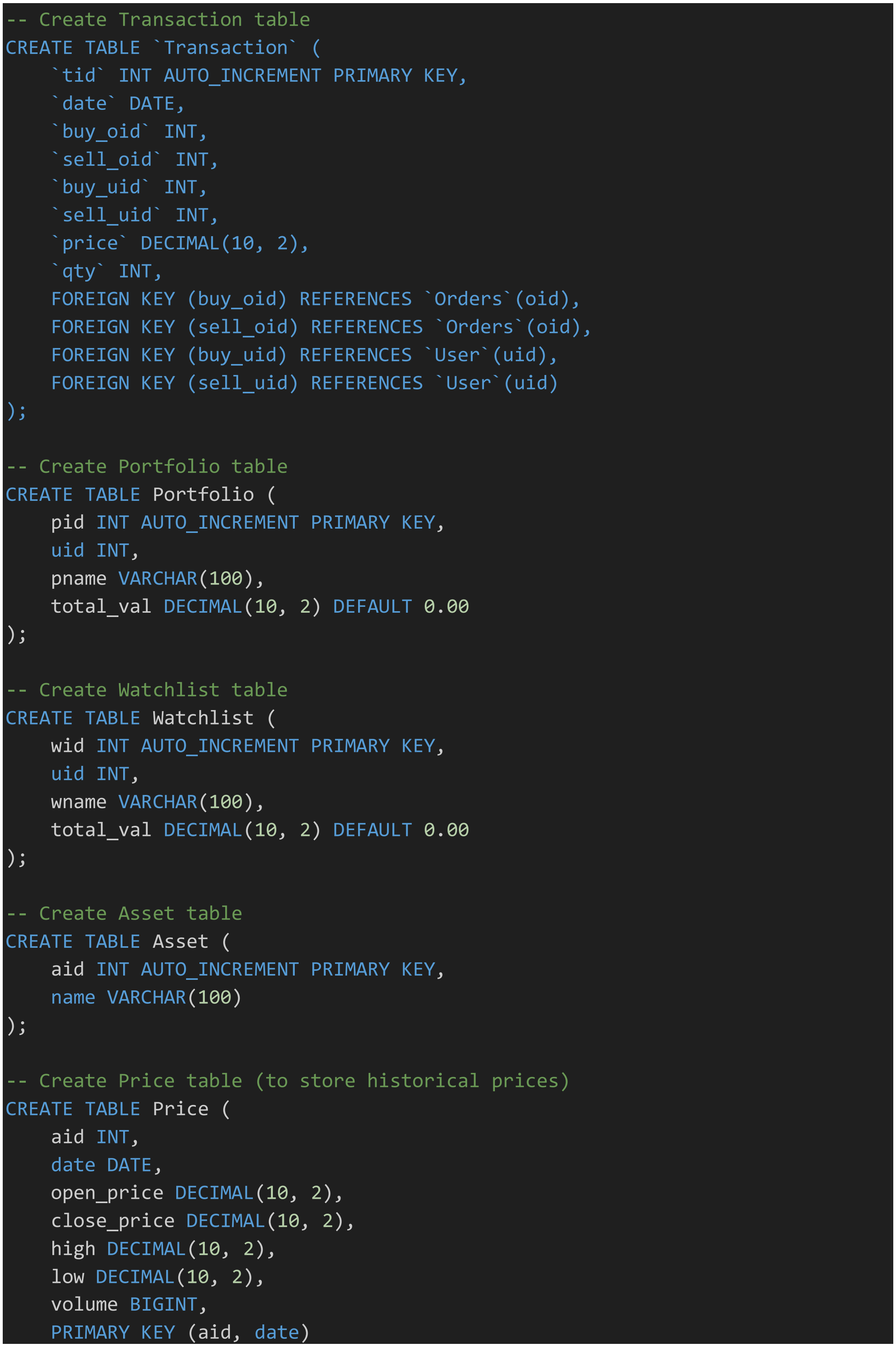
Figure 1: Gantt Chart for the Project

Relational Schema:

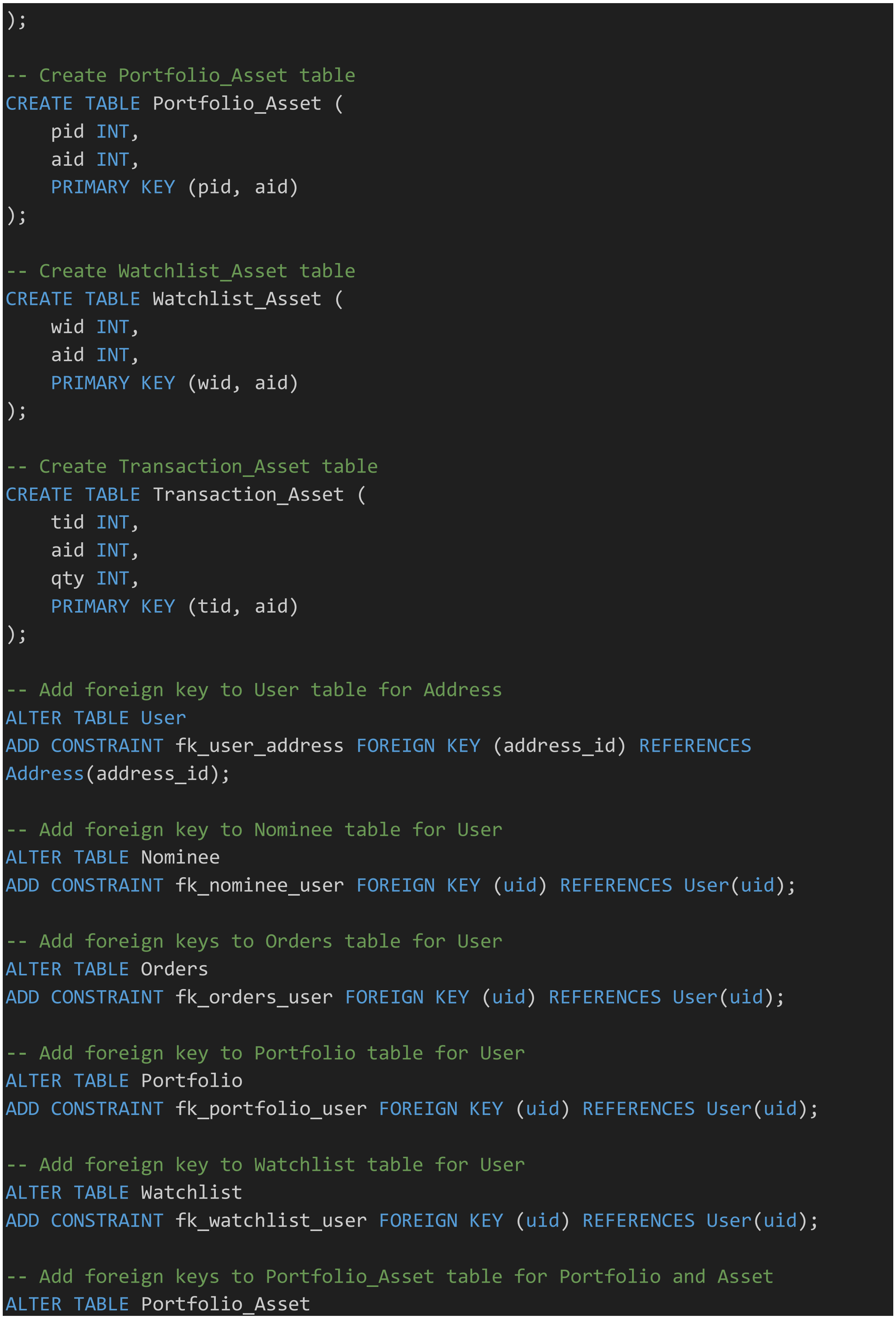


DDL Commands

```sql
CREATE DATABASE IF NOT EXISTS stock_app;
USE stock_app;

-- Create User table
CREATE TABLE User (
    uid INT AUTO_INCREMENT PRIMARY KEY,
    uname VARCHAR(100) NOT NULL,
    uemail VARCHAR(100) UNIQUE NOT NULL,
    upno VARCHAR(20),
    equity_funds DECIMAL(10, 2) DEFAULT 0.00,
    commodity_funds DECIMAL(10, 2) DEFAULT 0.00,
    address_id INT
);

-- Create Address table
CREATE TABLE Address (
    address_id INT AUTO_INCREMENT PRIMARY KEY,
    locality VARCHAR(100),
    city VARCHAR(100),
    building VARCHAR(100),
    hno VARCHAR(20)
);

-- Create Nominee table
CREATE TABLE Nominee (
    uid INT,
    nid INT,
    nemail VARCHAR(100),
    ntype VARCHAR(50),
    relation VARCHAR(50),
    nname VARCHAR(100),
    PRIMARY KEY (uid, nid)
);

-- Create Orders table
CREATE TABLE Orders (
    oid INT AUTO_INCREMENT PRIMARY KEY,
    uid INT,
    price DECIMAL(10, 2),
    qty INT,
    date DATE,
    otype VARCHAR(50),
    status VARCHAR(20) DEFAULT 'Pending',
    time TIME
);
```



```sql
-- Create Transaction table
CREATE TABLE `Transaction` (
    `tid` INT AUTO_INCREMENT PRIMARY KEY,
    `date` DATE,
    `buy_oid` INT,
    `sell_oid` INT,
    `buy_uid` INT,
    `sell_uid` INT,
    `price` DECIMAL(10, 2),
    `qty` INT,
    FOREIGN KEY (buy_oid) REFERENCES `Orders`(oid),
    FOREIGN KEY (sell_oid) REFERENCES `Orders`(oid),
    FOREIGN KEY (buy_uid) REFERENCES `User`(uid),
    FOREIGN KEY (sell_uid) REFERENCES `User`(uid)
);

-- Create Portfolio table
CREATE TABLE Portfolio (
    pid INT AUTO_INCREMENT PRIMARY KEY,
    uid INT,
    pname VARCHAR(100),
    total_val DECIMAL(10, 2) DEFAULT 0.00
);

-- Create Watchlist table
CREATE TABLE Watchlist (
    wid INT AUTO_INCREMENT PRIMARY KEY,
    uid INT,
    wname VARCHAR(100),
    total_val DECIMAL(10, 2) DEFAULT 0.00
);

-- Create Asset table
CREATE TABLE Asset (
    aid INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100)
);

-- Create Price table (to store historical prices)
CREATE TABLE Price (
    aid INT,
    date DATE,
    open_price DECIMAL(10, 2),
    close_price DECIMAL(10, 2),
    high DECIMAL(10, 2),
    low DECIMAL(10, 2),
    volume BIGINT,
    PRIMARY KEY (aid, date)
```



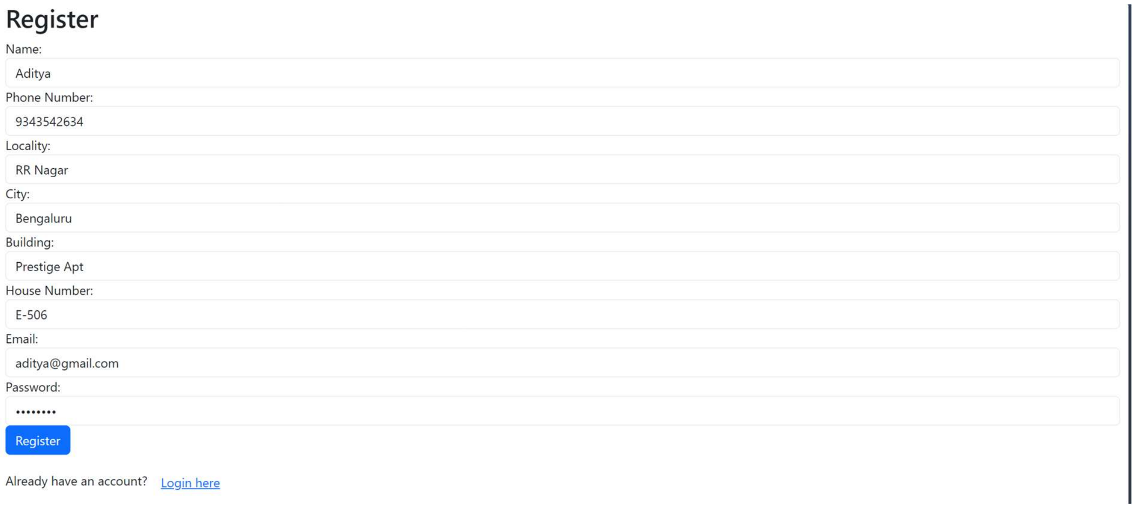
```sql
);

-- Create Portfolio_Asset table
CREATE TABLE Portfolio_Asset (
    pid INT,
    aid INT,
    PRIMARY KEY (pid, aid)
);

-- Create Watchlist_Asset table
CREATE TABLE Watchlist_Asset (
    wid INT,
    aid INT,
    PRIMARY KEY (wid, aid)
);

-- Create Transaction_Asset table
CREATE TABLE Transaction_Asset (
    tid INT,
    aid INT,
    qty INT,
    PRIMARY KEY (tid, aid)
);

-- Add foreign key to User table for Address
ALTER TABLE User
ADD CONSTRAINT fk_user_address FOREIGN KEY (address_id) REFERENCES
Address(address_id);

-- Add foreign key to Nominee table for User
ALTER TABLE Nominee
ADD CONSTRAINT fk_nominee_user FOREIGN KEY (uid) REFERENCES User(uid);

-- Add foreign keys to Orders table for User
ALTER TABLE Orders
ADD CONSTRAINT fk_orders_user FOREIGN KEY (uid) REFERENCES User(uid);

-- Add foreign key to Portfolio table for User
ALTER TABLE Portfolio
ADD CONSTRAINT fk_portfolio_user FOREIGN KEY (uid) REFERENCES User(uid);

-- Add foreign key to Watchlist table for User
ALTER TABLE Watchlist
ADD CONSTRAINT fk_watchlist_user FOREIGN KEY (uid) REFERENCES User(uid);

-- Add foreign keys to Portfolio_Asset table for Portfolio and Asset
ALTER TABLE Portfolio_Asset
```

```sql
ADD CONSTRAINT fk_portfolio_asset_portfolio FOREIGN KEY (pid) REFERENCES
Portfolio(pid),
ADD CONSTRAINT fk_portfolio_asset_asset FOREIGN KEY (aid) REFERENCES
Asset(aid);

-- Add foreign keys to Watchlist_Asset table for Watchlist and Asset
ALTER TABLE Watchlist_Asset
ADD CONSTRAINT fk_watchlist_asset_watchlist FOREIGN KEY (wid) REFERENCES
Watchlist(wid),
ADD CONSTRAINT fk_watchlist_asset_asset FOREIGN KEY (aid) REFERENCES
Asset(aid);

-- Add foreign keys to Transaction_Asset table for Transaction and Asset
ALTER TABLE Transaction_Asset
ADD CONSTRAINT fk_transaction_asset_transaction FOREIGN KEY (tid) REFERENCES
Transaction(tid),
ADD CONSTRAINT fk_transaction_asset_asset FOREIGN KEY (aid) REFERENCES
Asset(aid);

ALTER TABLE Portfolio DROP COLUMN total_val;

CREATE OR REPLACE VIEW AssetPriceView AS
SELECT
    a.aid,
    a.name,
    a.asset_type,        -- Include asset_type from Asset table
    p.close_price AS current_price
FROM
    Asset a
JOIN
    Price p ON a.aid = p.aid
JOIN
    (SELECT aid, MAX(date) AS latest_date FROM Price GROUP BY aid) AS
latest_prices
    ON p.aid = latest_prices.aid AND p.date = latest_prices.latest_date;



ALTER TABLE Portfolio_Asset
ADD COLUMN qty INT NOT NULL DEFAULT 0;

CREATE VIEW PortfolioTotalValueView AS
SELECT
    p.pid,
    p.uid,
    p.pname,
    SUM(pa.qty * apv.current_price) AS total_value
FROM
    Portfolio p
```

```
JOIN
    Portfolio_Asset pa ON p.pid = pa.pid
JOIN
    AssetPriceView apv ON pa.aid = apv.aid
GROUP BY
    p.pid, p.uid, p.pname;

ALTER TABLE Asset
ADD COLUMN asset_type ENUM('Equity', 'Commodity') NOT NULL DEFAULT 'Equity';
```

CRUD with GUI

Create Varied roles with user registration



Register

Name:
Aditya

Phone Number:
9343542634

Locality:
RR Nagar

City:
Bengaluru

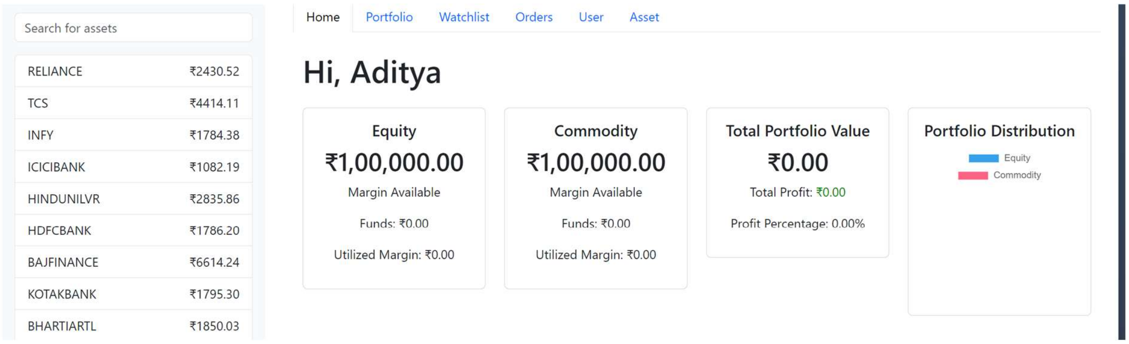
Building:
Prestige Apt

House Number:
E-506

Email:
aditya@gmail.com

Password:
••••••••

Already have an account? Login here



RELIANCE	₹2430.52
TCS	₹4414.11
INFY	₹1784.38
ICICIBANK	₹1082.19
HINDUNILVR	₹2835.86
HDFCBANK	₹1786.20
BAJFINANCE	₹6614.24
KOTAKBANK	₹1795.30
BHARTIARTL	₹1850.03

Hi, Aditya

Equity
₹1,00,000.00
Margin Available
Funds: ₹0.00
Utilized Margin: ₹0.00

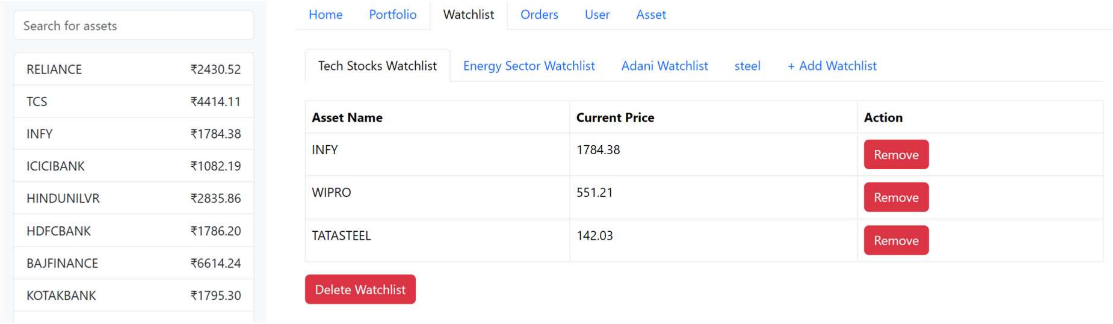
Commodity
₹1,00,000.00
Margin Available
Funds: ₹0.00
Utilized Margin: ₹0.00

Total Portfolio Value
₹0.00
Total Profit: ₹0.00
Profit Percentage: 0.00%

Portfolio Distribution
🟦 Equity
🟥 Commodity

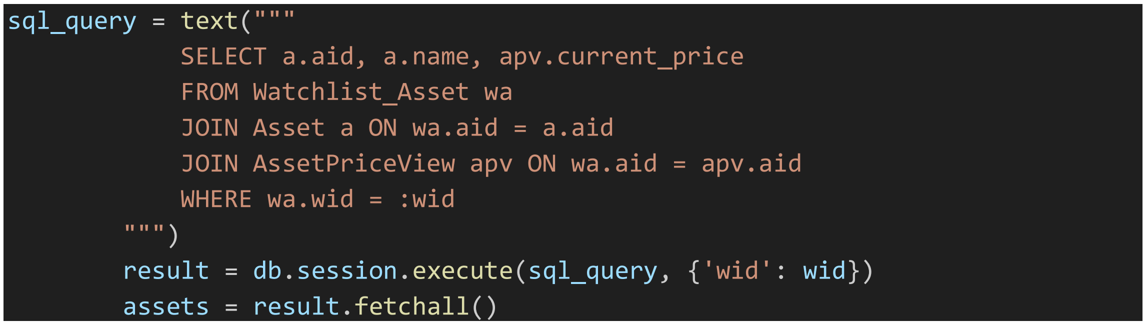
```
sql_insert_user = text("""
            INSERT INTO `User` (uname, uemail, upno, equity_funds,
commodity_funds, address_id, password)
            VALUES (:uname, :uemail, :upno, 0.00, 0.00, :address_id,
:password)
        """)
```

Read Watchlist:



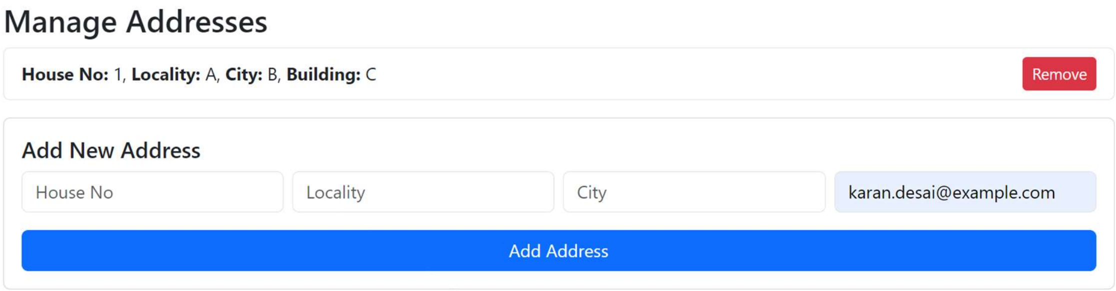
RELIANCE	₹2430.52
TCS	₹4414.11
INFY	₹1784.38
ICICIBANK	₹1082.19
HINDUNILVR	₹2835.86
HDFCBANK	₹1786.20
BAJFINANCE	₹6614.24
KOTAKBANK	₹1795.30

Asset Name	Current Price	Action
INFY	1784.38	Remove
WIPRO	551.21	Remove
TATASTEEL	142.03	Remove



```
sql_query = text("""
            SELECT a.aid, a.name, apv.current_price
            FROM Watchlist_Asset wa
            JOIN Asset a ON wa.aid = a.aid
            JOIN AssetPriceView apv ON wa.aid = apv.aid
            WHERE wa.wid = :wid
        """)
        result = db.session.execute(sql_query, {'wid': wid})
        assets = result.fetchall()
```

Delete Address:

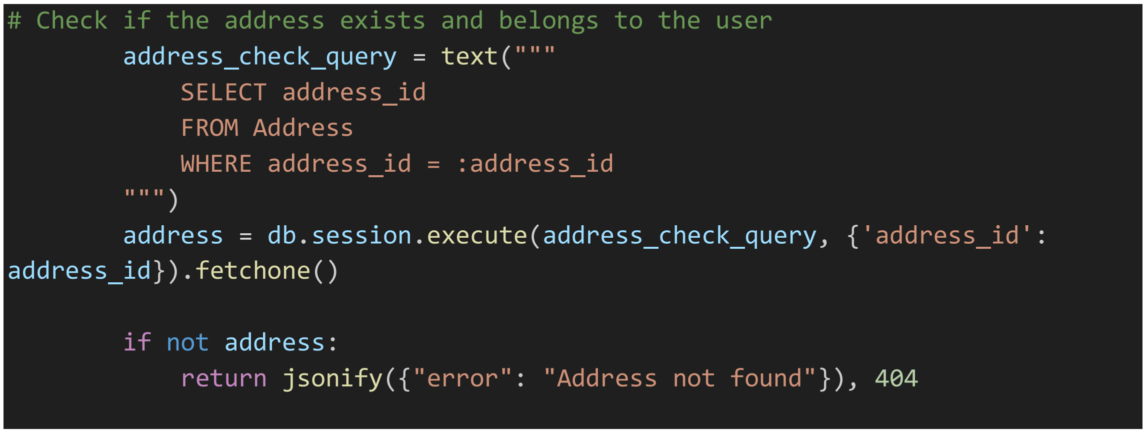


Manage Addresses

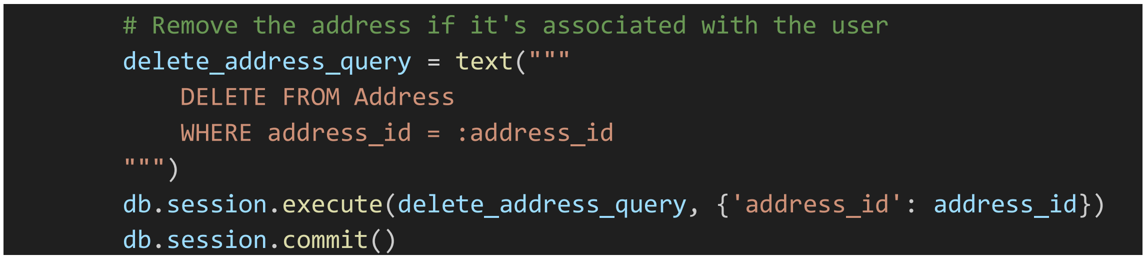
House No: 1, **Locality:** A, **City:** B, **Building:** C

Add New Address

karan.desai@example.com



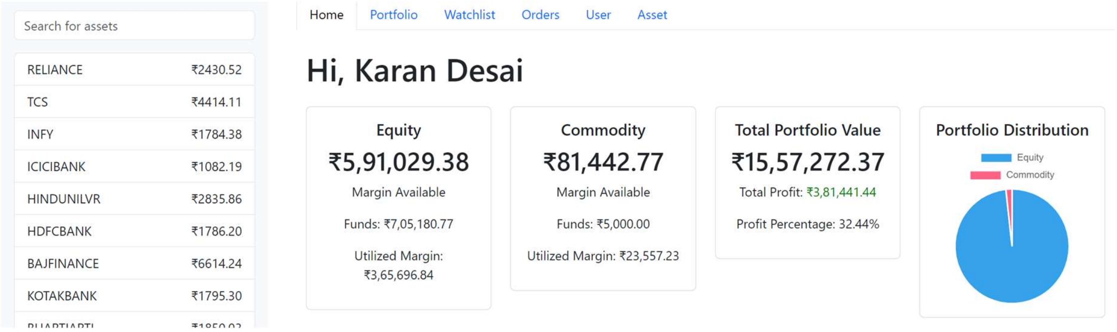
```
# Check if the address exists and belongs to the user
        address_check_query = text("""
            SELECT address_id
            FROM Address
            WHERE address_id = :address_id
        """)
        address = db.session.execute(address_check_query, {'address_id':
address_id}).fetchone()

        if not address:
            return jsonify({"error": "Address not found"}), 404
```



```
        # Remove the address if it's associated with the user
        delete_address_query = text("""
            DELETE FROM Address
            WHERE address_id = :address_id
        """)
        db.session.execute(delete_address_query, {'address_id': address_id})
        db.session.commit()
```

Update funds:

Before:



RELIANCE	₹2430.52
TCS	₹4414.11
INFY	₹1784.38
ICICIBANK	₹1082.19
HINDUNILVR	₹2835.86
HDFCBANK	₹1786.20
BAJFINANCE	₹6614.24
KOTAKBANK	₹1795.30
BHARTIARTL	₹1850.03

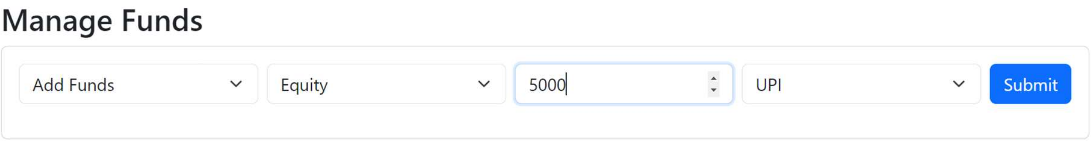
Hi, Karan Desai

Equity
₹5,91,029.38
Margin Available
Funds: ₹7,05,180.77
Utilized Margin: ₹3,65,696.84

Commodity
₹81,442.77
Margin Available
Funds: ₹5,000.00
Utilized Margin: ₹23,557.23

Total Portfolio Value
₹15,57,272.37
Total Profit: ₹3,81,441.44
Profit Percentage: 32.44%

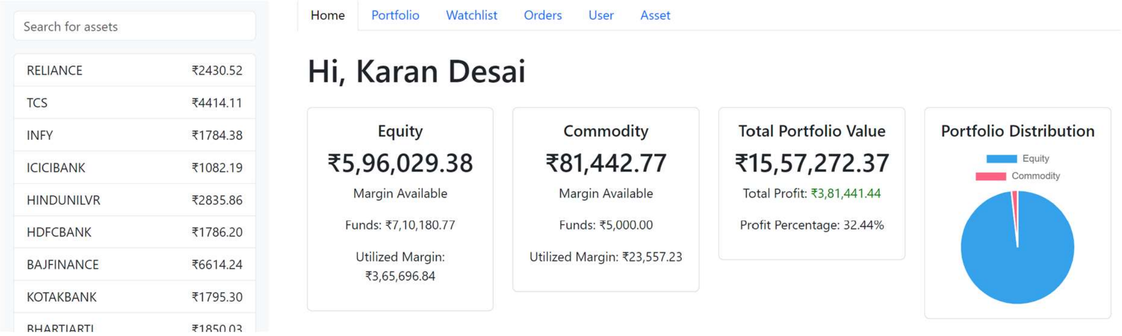
Portfolio Distribution



Manage Funds

Add Funds | Equity | 5000 | UPI

After:



RELIANCE	₹2430.52
TCS	₹4414.11
INFY	₹1784.38
ICICIBANK	₹1082.19
HINDUNILVR	₹2835.86
HDFCBANK	₹1786.20
BAJFINANCE	₹6614.24
KOTAKBANK	₹1795.30
BHARTIARTL	₹1850.03

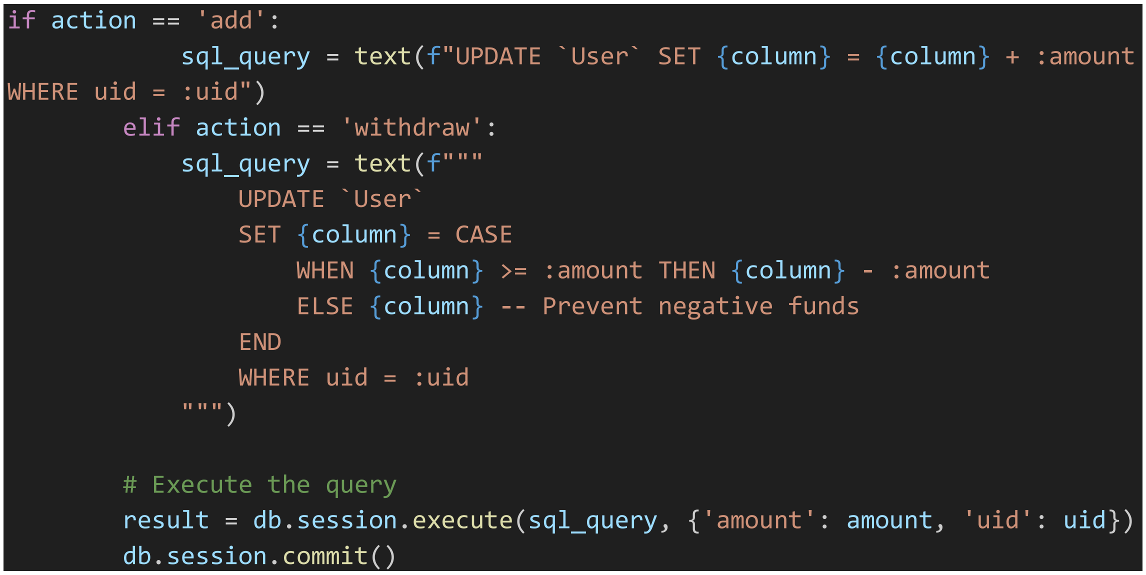
Hi, Karan Desai

Equity
₹5,96,029.38
Margin Available
Funds: ₹7,10,180.77
Utilized Margin: ₹3,65,696.84

Commodity
₹81,442.77
Margin Available
Funds: ₹5,000.00
Utilized Margin: ₹23,557.23

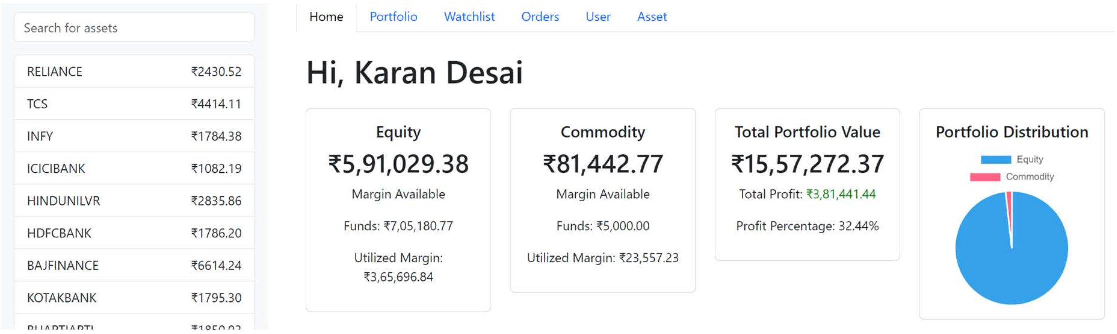
Total Portfolio Value
₹15,57,272.37
Total Profit: ₹3,81,441.44
Profit Percentage: 32.44%

Portfolio Distribution



```
if action == 'add':
            sql_query = text(f"UPDATE `User` SET {column} = {column} + :amount 
WHERE uid = :uid")
        elif action == 'withdraw':
            sql_query = text(f"""
                UPDATE `User`
                SET {column} = CASE
                    WHEN {column} >= :amount THEN {column} - :amount
                    ELSE {column} -- Prevent negative funds
                END
                WHERE uid = :uid
            """)

        # Execute the query
        result = db.session.execute(sql_query, {'amount': amount, 'uid': uid})
        db.session.commit()
```

List of All Pages with Functionalities:



Search for assets

RELIANCE	₹2430.52
TCS	₹4414.11
INFY	₹1784.38
ICICIBANK	₹1082.19
HINDUNILVR	₹2835.86
HDFCBANK	₹1786.20
BAJFINANCE	₹6614.24
KOTAKBANK	₹1795.30
BHARTIARTL	₹1850.02

Home Portfolio Watchlist Orders User Asset

Hi, Karan Desai

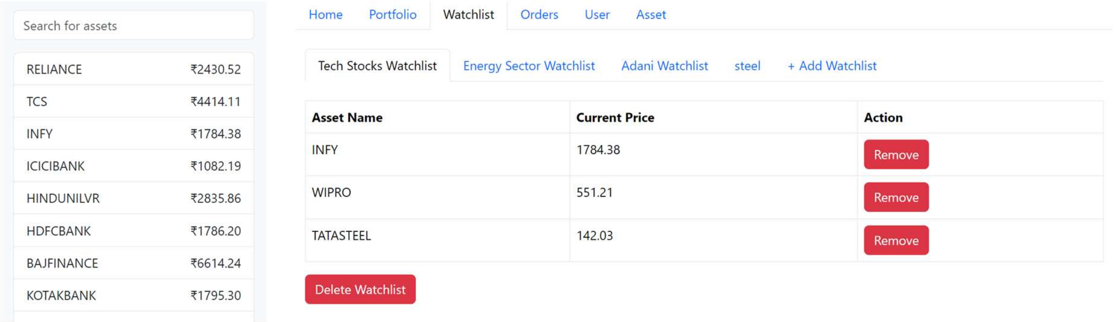
Equity
₹5,91,029.38
Margin Available
Funds: ₹7,05,180.77
Utilized Margin: ₹3,65,696.84

Commodity
₹81,442.77
Margin Available
Funds: ₹5,000.00
Utilized Margin: ₹23,557.23

Total Portfolio Value
₹15,57,272.37
Total Profit: ₹3,81,441.44
Profit Percentage: 32.44%

Portfolio Distribution

Home: Overview of portfolio and funds status, dashboard with widgets, sidebar: present in all screens can be used to search any asset or commodity in the system.



Search for assets

RELIANCE	₹2430.52
TCS	₹4414.11
INFY	₹1784.38
ICICIBANK	₹1082.19
HINDUNILVR	₹2835.86
HDFCBANK	₹1786.20
BAJFINANCE	₹6614.24
KOTAKBANK	₹1795.30

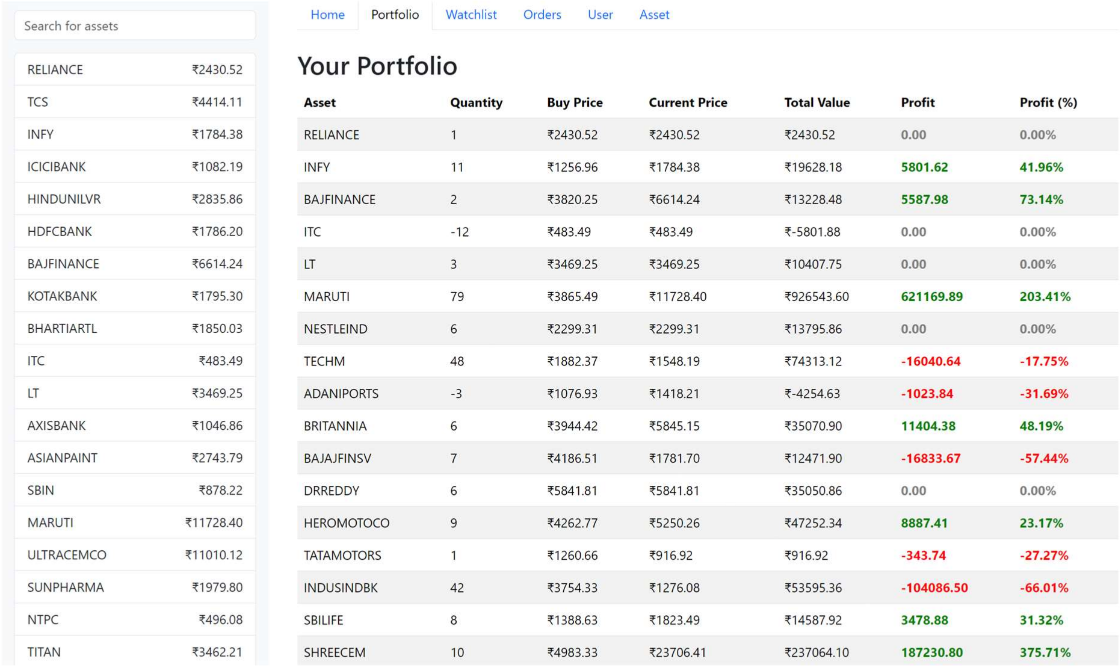
Home Portfolio Watchlist Orders User Asset

Tech Stocks Watchlist Energy Sector Watchlist Adani Watchlist steel + Add Watchlist

Asset Name	Current Price	Action
INFY	1784.38	Remove
WIPRO	551.21	Remove
TATASTEEL	142.03	Remove

Delete Watchlist

Watchlist: Create multiple custom watchlists with custom names, add and remove stocks to any watchlist.

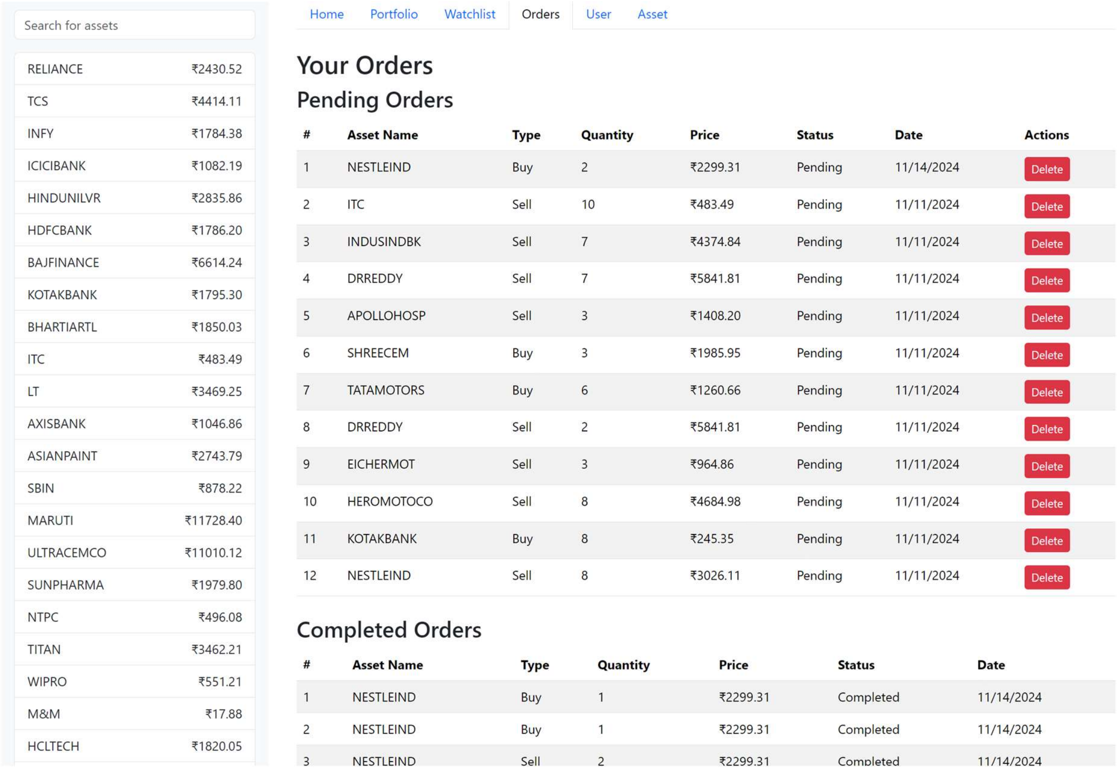


RELIANCE	₹2430.52
TCS	₹4414.11
INFY	₹1784.38
ICICIBANK	₹1082.19
HINDUNILVR	₹2835.86
HDFCBANK	₹1786.20
BAJFINANCE	₹6614.24
KOTAKBANK	₹1795.30
BHARTIARTL	₹1850.03
ITC	₹483.49
LT	₹3469.25
AXISBANK	₹1046.86
ASIANPAINT	₹2743.79
SBIN	₹878.22
MARUTI	₹11728.40
ULTRACEMCO	₹11010.12
SUNPHARMA	₹1979.80
NTPC	₹496.08
TITAN	₹3462.21

Your Portfolio

Asset	Quantity	Buy Price	Current Price	Total Value	Profit	Profit (%)
RELIANCE	1	₹2430.52	₹2430.52	₹2430.52	0.00	0.00%
INFY	11	₹1256.96	₹1784.38	₹19628.18	5801.62	41.96%
BAJFINANCE	2	₹3820.25	₹6614.24	₹13228.48	5587.98	73.14%
ITC	-12	₹483.49	₹483.49	₹-5801.88	0.00	0.00%
LT	3	₹3469.25	₹3469.25	₹10407.75	0.00	0.00%
MARUTI	79	₹3865.49	₹11728.40	₹926543.60	621169.89	203.41%
NESTLEIND	6	₹2299.31	₹2299.31	₹13795.86	0.00	0.00%
TECHM	48	₹1882.37	₹1548.19	₹74313.12	-16040.64	-17.75%
ADANIPORTS	-3	₹1076.93	₹1418.21	₹-4254.63	-1023.84	-31.69%
BRITANNIA	6	₹3944.42	₹5845.15	₹35070.90	11404.38	48.19%
BAJAJFINSV	7	₹4186.51	₹1781.70	₹12471.90	-16833.67	-57.44%
DRREDDY	6	₹5841.81	₹5841.81	₹35050.86	0.00	0.00%
HEROMOTOCO	9	₹4262.77	₹5250.26	₹47252.34	8887.41	23.17%
TATAMOTORS	1	₹1260.66	₹916.92	₹916.92	-343.74	-27.27%
INDUSINDBK	42	₹3754.33	₹1276.08	₹53595.36	-104086.50	-66.01%
SBILIFE	8	₹1388.63	₹1823.49	₹14587.92	3478.88	31.32%
SHREECEM	10	₹4983.33	₹23706.41	₹237064.10	187230.80	375.71%

Portfolio: View of portfolio with details of each asset and calculated profit and profit percent



RELIANCE	₹2430.52
TCS	₹4414.11
INFY	₹1784.38
ICICIBANK	₹1082.19
HINDUNILVR	₹2835.86
HDFCBANK	₹1786.20
BAJFINANCE	₹6614.24
KOTAKBANK	₹1795.30
BHARTIARTL	₹1850.03
ITC	₹483.49
LT	₹3469.25
AXISBANK	₹1046.86
ASIANPAINT	₹2743.79
SBIN	₹878.22
MARUTI	₹11728.40
ULTRACEMCO	₹11010.12
SUNPHARMA	₹1979.80
NTPC	₹496.08
TITAN	₹3462.21
WIPRO	₹551.21
M&M	₹17.88
HCLTECH	₹1820.05

Your Orders

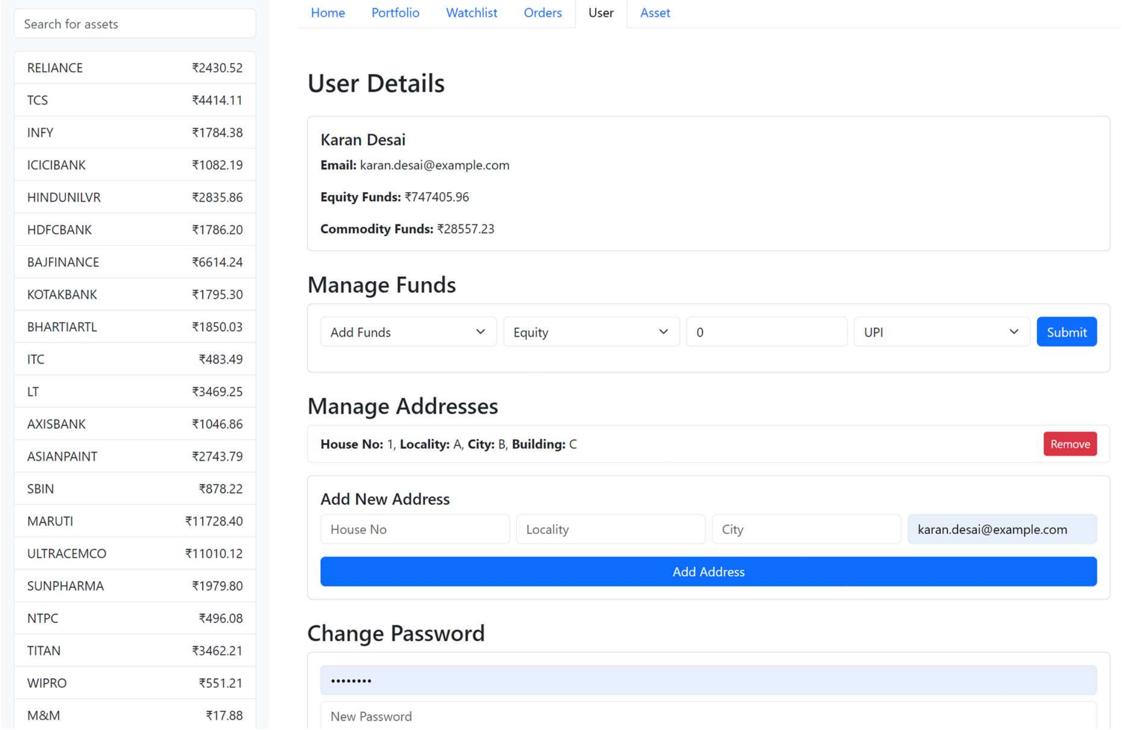
Pending Orders

#	Asset Name	Type	Quantity	Price	Status	Date	Actions
1	NESTLEIND	Buy	2	₹2299.31	Pending	11/14/2024	Delete
2	ITC	Sell	10	₹483.49	Pending	11/11/2024	Delete
3	INDUSINDBK	Sell	7	₹4374.84	Pending	11/11/2024	Delete
4	DRREDDY	Sell	7	₹5841.81	Pending	11/11/2024	Delete
5	APOLLOHOSP	Sell	3	₹1408.20	Pending	11/11/2024	Delete
6	SHREECEM	Buy	3	₹1985.95	Pending	11/11/2024	Delete
7	TATAMOTORS	Buy	6	₹1260.66	Pending	11/11/2024	Delete
8	DRREDDY	Sell	2	₹5841.81	Pending	11/11/2024	Delete
9	EICHERMOT	Sell	3	₹964.86	Pending	11/11/2024	Delete
10	HEROMOTOCO	Sell	8	₹4684.98	Pending	11/11/2024	Delete
11	KOTAKBANK	Buy	8	₹245.35	Pending	11/11/2024	Delete
12	NESTLEIND	Sell	8	₹3026.11	Pending	11/11/2024	Delete

Completed Orders

#	Asset Name	Type	Quantity	Price	Status	Date
1	NESTLEIND	Buy	1	₹2299.31	Completed	11/14/2024
2	NESTLEIND	Buy	1	₹2299.31	Completed	11/14/2024
3	NESTLEIND	Sell	2	₹2299.31	Completed	11/14/2024

Orders: See all currently pending and completed orders. Orders are dynamically matched with availiable orders of same or better pricing across the system. Partial quantities are also fulfilled with one order being split into 2 in such cases.



RELIANCE	₹2430.52
TCS	₹4414.11
INFY	₹1784.38
ICICIBANK	₹1082.19
HINDUNILVR	₹2835.86
HDFCBANK	₹1786.20
BAJFINANCE	₹6614.24
KOTAKBANK	₹1795.30
BHARTIARTL	₹1850.03
ITC	₹483.49
LT	₹3469.25
AXISBANK	₹1046.86
ASIANPAINT	₹2743.79
SBIN	₹878.22
MARUTI	₹11728.40
ULTRACEMCO	₹11010.12
SUNPHARMA	₹1979.80
NTPC	₹496.08
TITAN	₹3462.21
WIPRO	₹551.21
M&M	₹17.88

User Details

Karan Desai

Email: karan.desai@example.com

Equity Funds: ₹747405.96

Commodity Funds: ₹28557.23

Manage Funds

Add Funds | Equity | 0 | UPI | Submit

Manage Addresses

House No: 1, **Locality:** A, **City:** B, **Building:** C Remove

Add New Address

House No | Locality | City | karan.desai@example.com

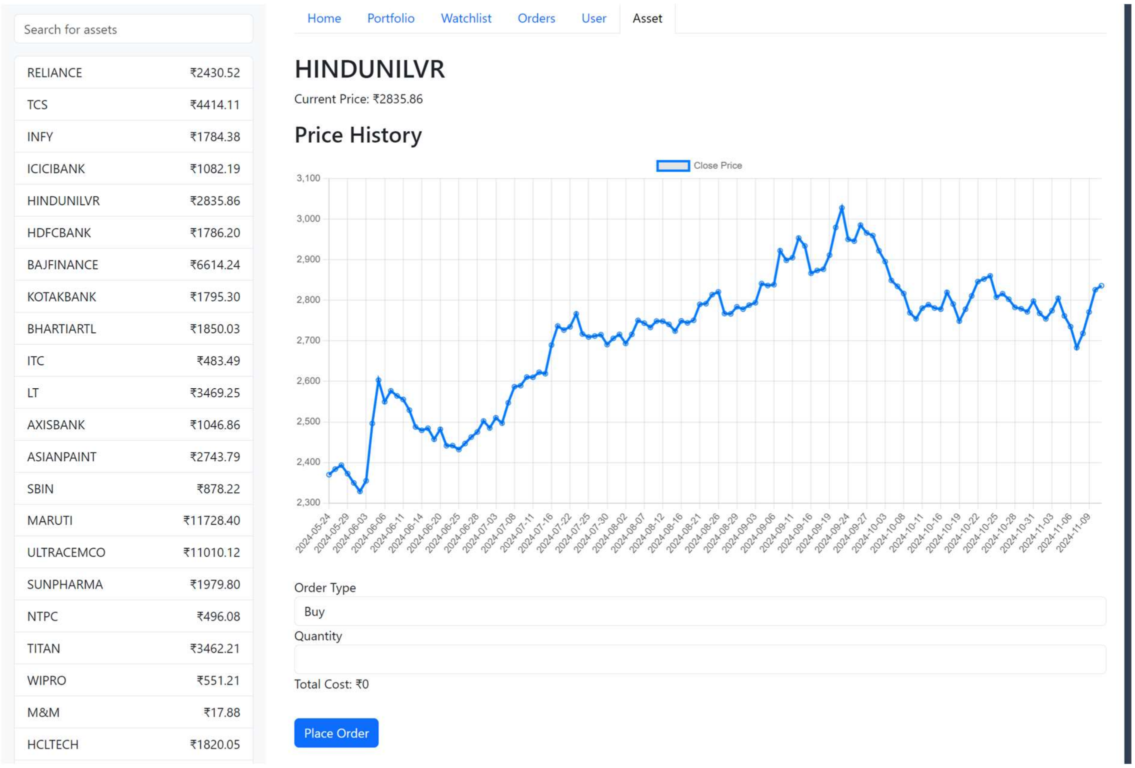
Add Address

Change Password

••••••••

New Password

User: All user management like funds, addresses, passwords, logout, etc.



RELIANCE	₹2430.52
TCS	₹4414.11
INFY	₹1784.38
ICICIBANK	₹1082.19
HINDUNILVR	₹2835.86
HDFCBANK	₹1786.20
BAJFINANCE	₹6614.24
KOTAKBANK	₹1795.30
BHARTIARTL	₹1850.03
ITC	₹483.49
LT	₹3469.25
AXISBANK	₹1046.86
ASIANPAINT	₹2743.79
SBIN	₹878.22
MARUTI	₹11728.40
ULTRACEMCO	₹11010.12
SUNPHARMA	₹1979.80
NTPC	₹496.08
TITAN	₹3462.21
WIPRO	₹551.21
M&M	₹17.88
HCLTECH	₹1820.05

HINDUNILVR

Current Price: ₹2835.86

Price History

Order Type

Buy

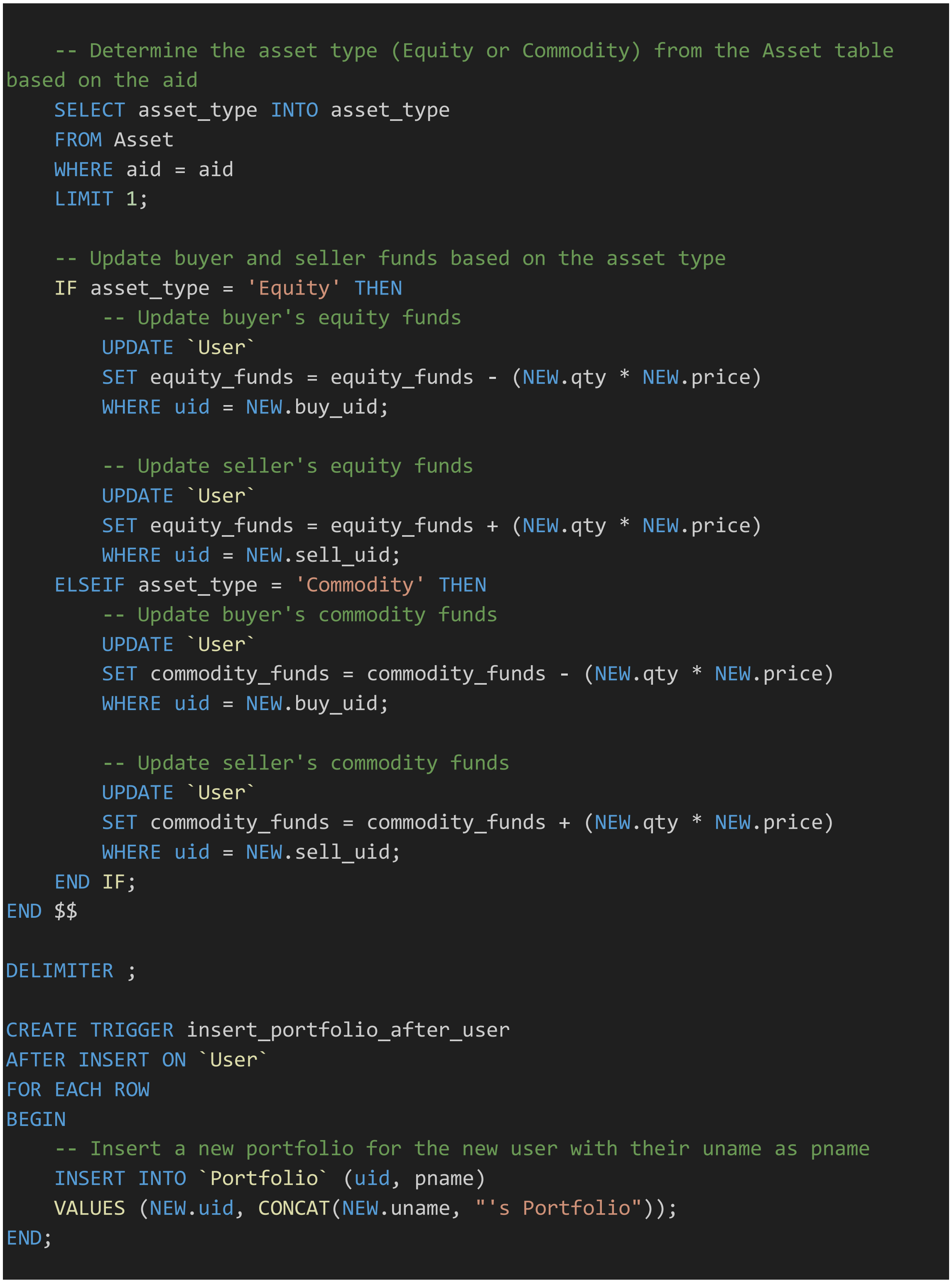
Quantity

Total Cost: ₹0

Asset: View pricing of assets, buy or sell an asset or add a particular asset to a particular watchlist

Triggers (work with GUI)

```
DELIMITER $$

DROP TRIGGER IF EXISTS update_funds_after_transaction;

CREATE TRIGGER update_funds_after_transaction
AFTER INSERT ON Transaction
FOR EACH ROW
BEGIN
    DECLARE asset_type VARCHAR(20);
    DECLARE aid INT;

    -- Fetch the asset ID (aid) from the Orders table using buy_oid from the new transaction
    SELECT o.aid INTO aid
    FROM Orders o
    WHERE o.oid = NEW.buy_oid
    LIMIT 1;
```

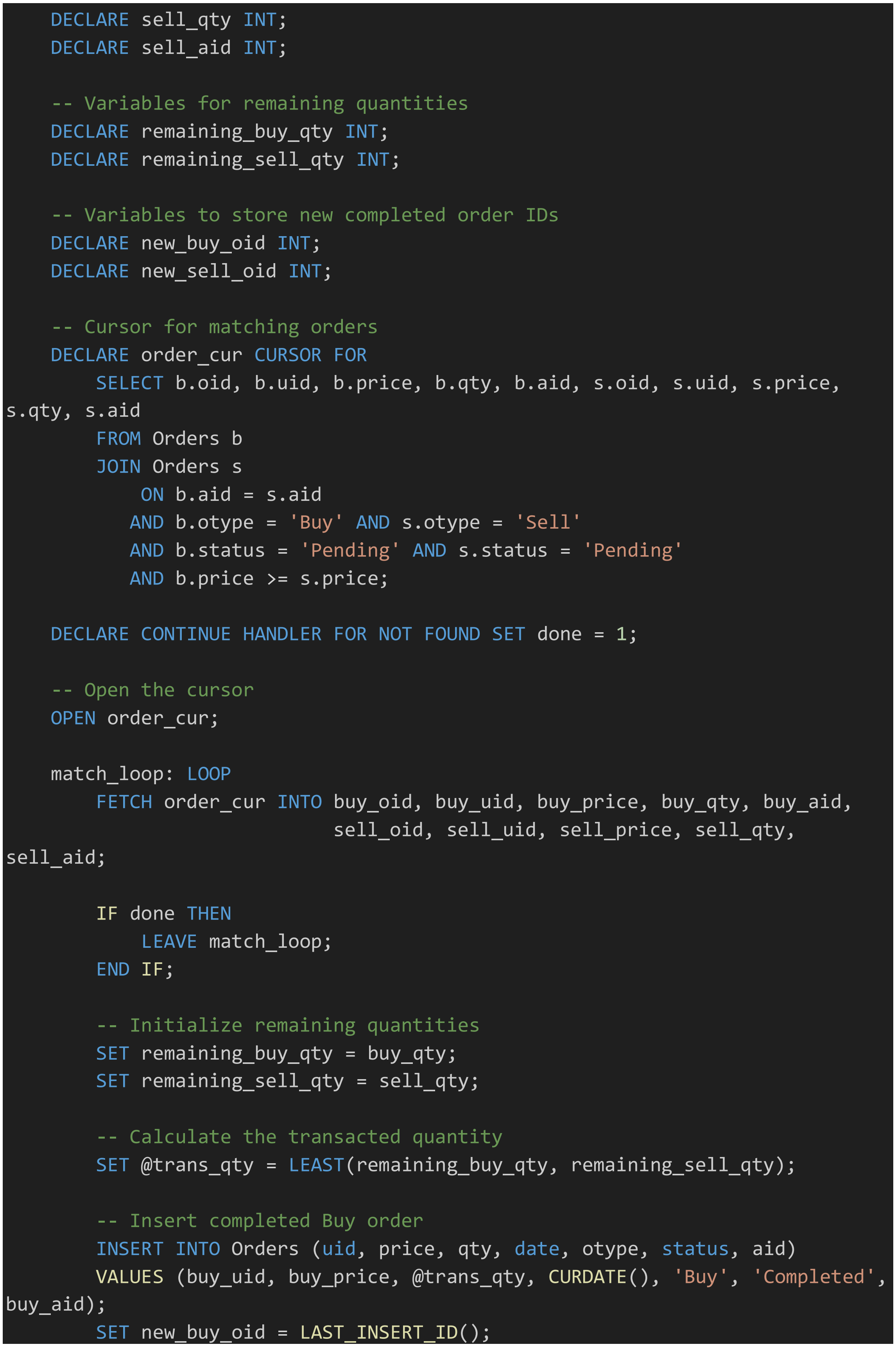


```sql
    -- Determine the asset type (Equity or Commodity) from the Asset table based on the aid
    SELECT asset_type INTO asset_type
    FROM Asset
    WHERE aid = aid
    LIMIT 1;

    -- Update buyer and seller funds based on the asset type
    IF asset_type = 'Equity' THEN
        -- Update buyer's equity funds
        UPDATE `User`
        SET equity_funds = equity_funds - (NEW.qty * NEW.price)
        WHERE uid = NEW.buy_uid;

        -- Update seller's equity funds
        UPDATE `User`
        SET equity_funds = equity_funds + (NEW.qty * NEW.price)
        WHERE uid = NEW.sell_uid;
    ELSEIF asset_type = 'Commodity' THEN
        -- Update buyer's commodity funds
        UPDATE `User`
        SET commodity_funds = commodity_funds - (NEW.qty * NEW.price)
        WHERE uid = NEW.buy_uid;

        -- Update seller's commodity funds
        UPDATE `User`
        SET commodity_funds = commodity_funds + (NEW.qty * NEW.price)
        WHERE uid = NEW.sell_uid;
    END IF;
END $$

DELIMITER ;

CREATE TRIGGER insert_portfolio_after_user
AFTER INSERT ON `User`
FOR EACH ROW
BEGIN
    -- Insert a new portfolio for the new user with their uname as pname
    INSERT INTO `Portfolio` (uid, pname)
    VALUES (NEW.uid, CONCAT(NEW.uname, "'s Portfolio"));
END;
```

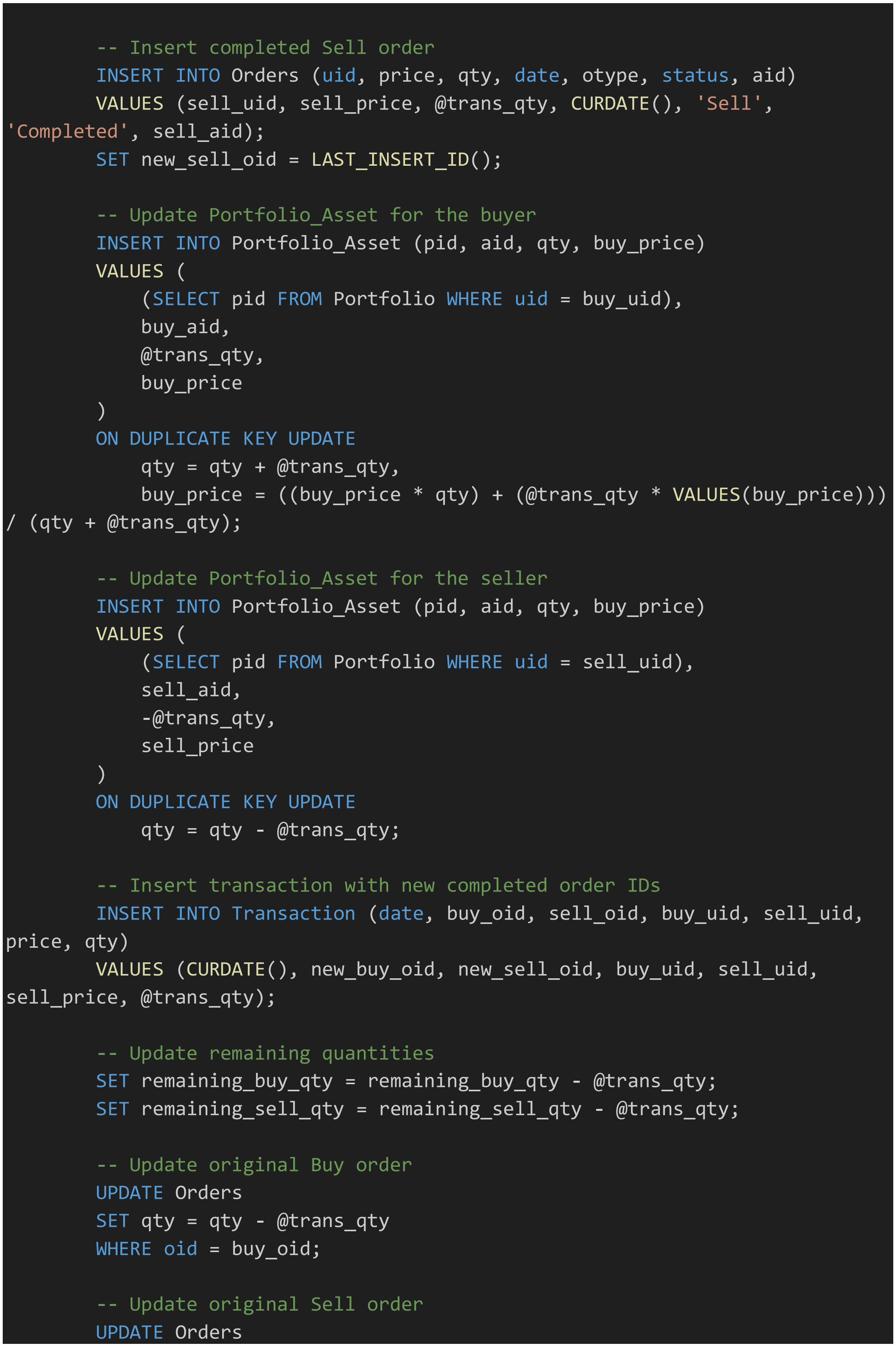
One to update funds of user after a transaction has occurred, one to create a new portfolio for each new user.

Procedures (Work with GUI):

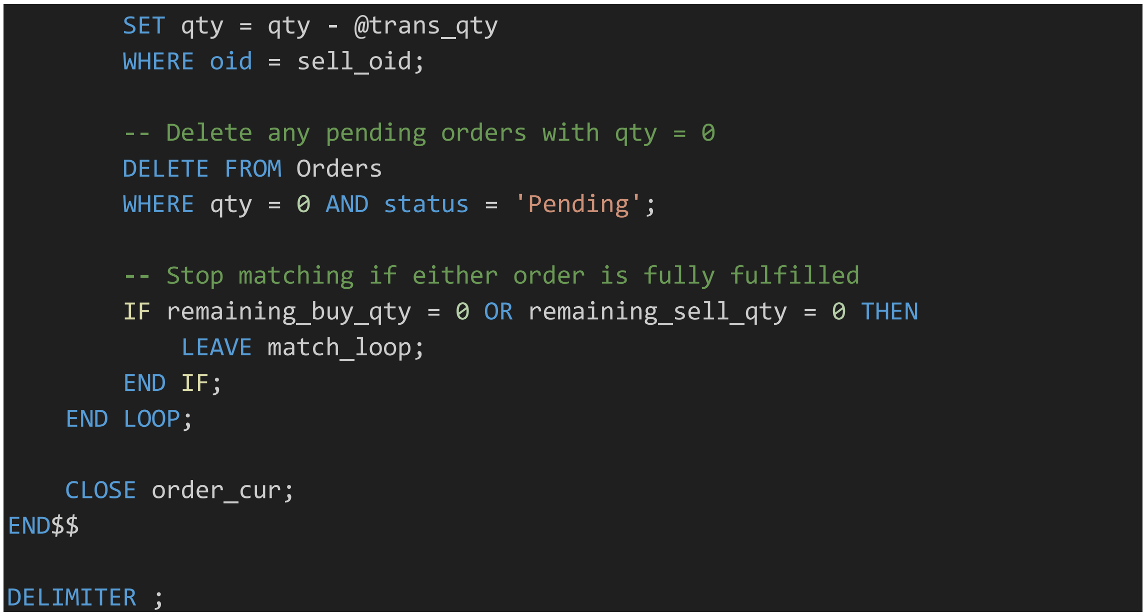
```
USE stock_app;

DROP PROCEDURE IF EXISTS place_order;

DELIMITER $$

CREATE PROCEDURE place_order(
    IN uid INT,
    IN asset_id INT,
    IN qty INT,
    IN otype VARCHAR(50)
)
BEGIN
    DECLARE asset_price DECIMAL(10,2);

    -- Get the current price from AssetPriceView
    SELECT current_price INTO asset_price
    FROM AssetPriceView
    WHERE aid = asset_id
    LIMIT 1;

    -- Insert the new order
    INSERT INTO `Orders` (uid, aid, qty, price, otype, status, date, time)
    VALUES (uid, asset_id, qty, asset_price, otype, 'Pending', CURDATE(),
CURTIME());
END$$

DELIMITER ;

DROP PROCEDURE IF EXISTS MatchOrders;

DELIMITER $$

CREATE PROCEDURE MatchOrders()
BEGIN
    DECLARE done INT DEFAULT 0;

    -- Variables for Buy and Sell orders
    DECLARE buy_oid INT;
    DECLARE buy_uid INT;
    DECLARE buy_price DECIMAL(10,2);
    DECLARE buy_qty INT;
    DECLARE buy_aid INT;
    DECLARE sell_oid INT;
    DECLARE sell_uid INT;
    DECLARE sell_price DECIMAL(10,2);
```



```
    DECLARE sell_qty INT;
    DECLARE sell_aid INT;

    -- Variables for remaining quantities
    DECLARE remaining_buy_qty INT;
    DECLARE remaining_sell_qty INT;

    -- Variables to store new completed order IDs
    DECLARE new_buy_oid INT;
    DECLARE new_sell_oid INT;

    -- Cursor for matching orders
    DECLARE order_cur CURSOR FOR
        SELECT b.oid, b.uid, b.price, b.qty, b.aid, s.oid, s.uid, s.price,
s.qty, s.aid
        FROM Orders b
        JOIN Orders s
            ON b.aid = s.aid
           AND b.otype = 'Buy' AND s.otype = 'Sell'
           AND b.status = 'Pending' AND s.status = 'Pending'
           AND b.price >= s.price;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    -- Open the cursor
    OPEN order_cur;

    match_loop: LOOP
        FETCH order_cur INTO buy_oid, buy_uid, buy_price, buy_qty, buy_aid,
                             sell_oid, sell_uid, sell_price, sell_qty,
sell_aid;

        IF done THEN
            LEAVE match_loop;
        END IF;

        -- Initialize remaining quantities
        SET remaining_buy_qty = buy_qty;
        SET remaining_sell_qty = sell_qty;

        -- Calculate the transacted quantity
        SET @trans_qty = LEAST(remaining_buy_qty, remaining_sell_qty);

        -- Insert completed Buy order
        INSERT INTO Orders (uid, price, qty, date, otype, status, aid)
        VALUES (buy_uid, buy_price, @trans_qty, CURDATE(), 'Buy', 'Completed',
buy_aid);
        SET new_buy_oid = LAST_INSERT_ID();
```

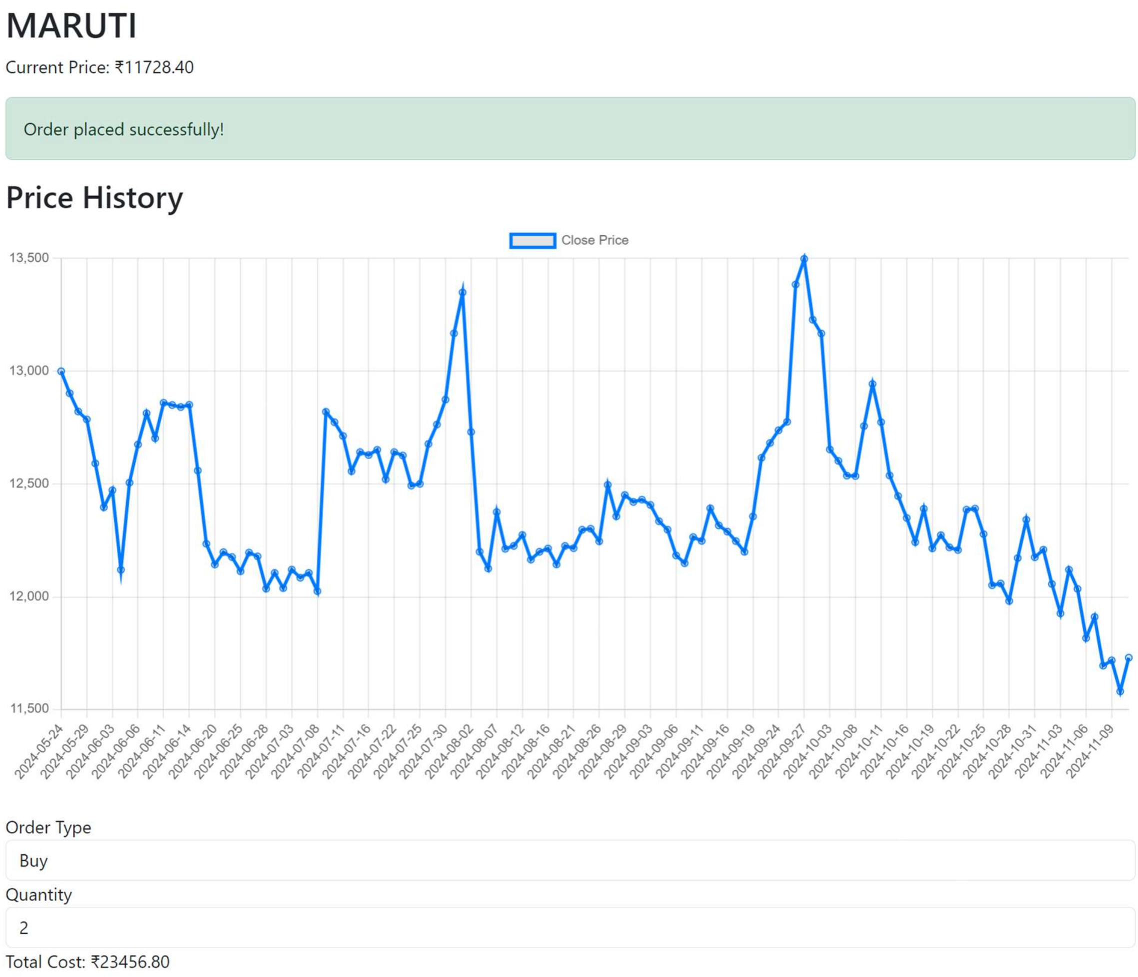


```
        -- Insert completed Sell order
        INSERT INTO Orders (uid, price, qty, date, otype, status, aid)
        VALUES (sell_uid, sell_price, @trans_qty, CURDATE(), 'Sell',
'Completed', sell_aid);
        SET new_sell_oid = LAST_INSERT_ID();

        -- Update Portfolio_Asset for the buyer
        INSERT INTO Portfolio_Asset (pid, aid, qty, buy_price)
        VALUES (
            (SELECT pid FROM Portfolio WHERE uid = buy_uid),
            buy_aid,
            @trans_qty,
            buy_price
        )
        ON DUPLICATE KEY UPDATE
            qty = qty + @trans_qty,
            buy_price = ((buy_price * qty) + (@trans_qty * VALUES(buy_price)))
/ (qty + @trans_qty);

        -- Update Portfolio_Asset for the seller
        INSERT INTO Portfolio_Asset (pid, aid, qty, buy_price)
        VALUES (
            (SELECT pid FROM Portfolio WHERE uid = sell_uid),
            sell_aid,
            -@trans_qty,
            sell_price
        )
        ON DUPLICATE KEY UPDATE
            qty = qty - @trans_qty;

        -- Insert transaction with new completed order IDs
        INSERT INTO Transaction (date, buy_oid, sell_oid, buy_uid, sell_uid,
price, qty)
        VALUES (CURDATE(), new_buy_oid, new_sell_oid, buy_uid, sell_uid,
sell_price, @trans_qty);

        -- Update remaining quantities
        SET remaining_buy_qty = remaining_buy_qty - @trans_qty;
        SET remaining_sell_qty = remaining_sell_qty - @trans_qty;

        -- Update original Buy order
        UPDATE Orders
        SET qty = qty - @trans_qty
        WHERE oid = buy_oid;

        -- Update original Sell order
        UPDATE Orders
```



```
        SET qty = qty - @trans_qty
        WHERE oid = sell_oid;

        -- Delete any pending orders with qty = 0
        DELETE FROM Orders
        WHERE qty = 0 AND status = 'Pending';

        -- Stop matching if either order is fully fulfilled
        IF remaining_buy_qty = 0 OR remaining_sell_qty = 0 THEN
            LEAVE match_loop;
        END IF;
    END LOOP;

    CLOSE order_cur;
END$$

DELIMITER ;
```

Used to place an order and find matching orders.



MARUTI

Current Price: ₹11728.40

Order placed successfully!

Price History

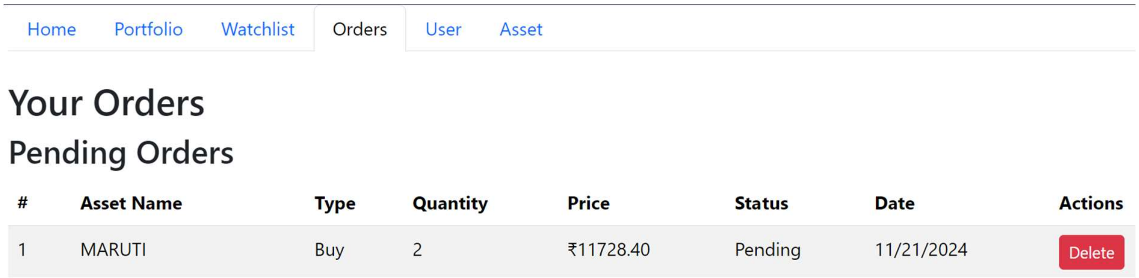
Order Type

Buy

Quantity

2

Total Cost: ₹23456.80



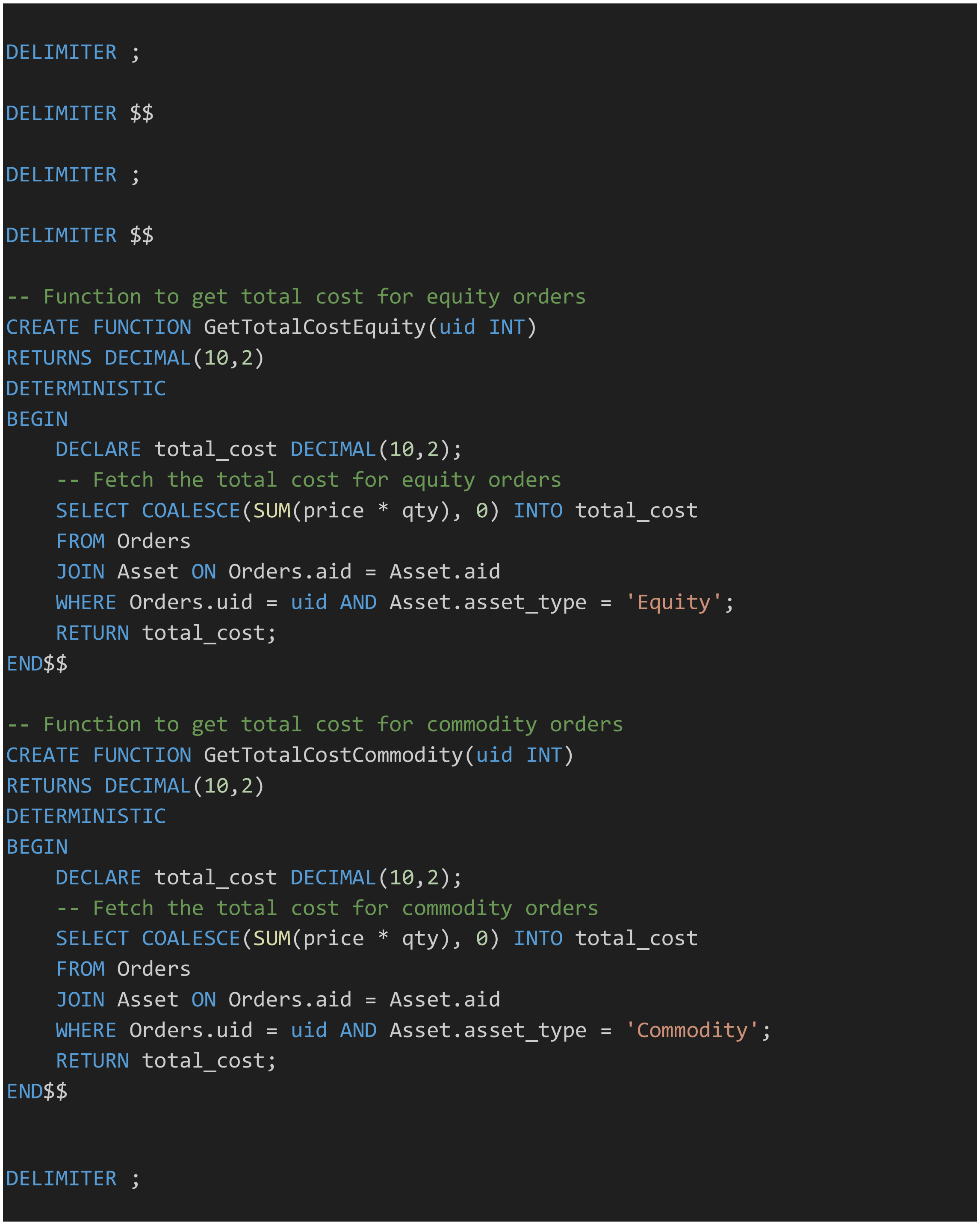
Your Orders

Pending Orders

#	Asset Name	Type	Quantity	Price	Status	Date	Actions
1	MARUTI	Buy	2	₹11728.40	Pending	11/21/2024	Delete

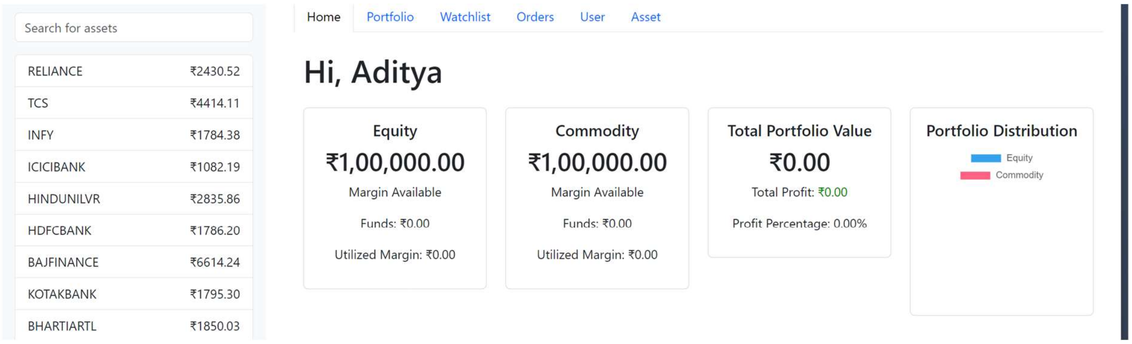
Functions (work with GUI):

```
DELIMITER $$

CREATE FUNCTION calculate_portfolio_value(uid INT)
RETURNS DECIMAL(15,2)
DETERMINISTIC
BEGIN
    DECLARE total_value DECIMAL(15,2) DEFAULT 0;
    DECLARE asset_value DECIMAL(15,2);

    -- Cursor to loop through all assets in the user's portfolio
    DECLARE done INT DEFAULT 0;
    DECLARE cur_aid INT;
    DECLARE cur_qty INT;
    DECLARE cur_price DECIMAL(10,2);

    -- Cursor to get all assets in the user's portfolio with their quantity
    DECLARE asset_cursor CURSOR FOR
    SELECT p.aid, p.qty, a.price
    FROM Portfolio_Asset p
    JOIN Asset a ON p.aid = a.aid
    WHERE p.uid = uid;

    -- Continue loop handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    -- Open the cursor
    OPEN asset_cursor;

    read_loop: LOOP
        -- Fetch each asset and its quantity and price
        FETCH asset_cursor INTO cur_aid, cur_qty, cur_price;

        -- Exit the loop if no more rows
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Calculate the asset value and add it to the total portfolio value
        SET asset_value = cur_qty * cur_price;
        SET total_value = total_value + asset_value;
    END LOOP;

    -- Close the cursor
    CLOSE asset_cursor;

    -- Return the total portfolio value
    RETURN total_value;
END$$

DELIMITER ;

DELIMITER $$

-- Function to calculate total pending cost for equity orders
CREATE FUNCTION GetTotalPendingCostEquity(uid INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE total_cost DECIMAL(10,2);
    -- Fetch the total pending cost for equity (use COALESCE to handle NULL)
    SELECT COALESCE(SUM(price * qty), 0) INTO total_cost
    FROM Orders
    JOIN Asset ON Orders.aid = Asset.aid
    WHERE Orders.uid = uid AND Orders.status = 'Pending' AND Asset.asset_type
= 'Equity';
    RETURN total_cost;
END$$

-- Function to calculate total pending cost for commodity orders
CREATE FUNCTION GetTotalPendingCostCommodity(uid INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE total_cost DECIMAL(10,2);
    -- Fetch the total pending cost for commodity (use COALESCE to handle
NULL)
    SELECT COALESCE(SUM(price * qty), 0) INTO total_cost
    FROM Orders
    JOIN Asset ON Orders.aid = Asset.aid
    WHERE Orders.uid = uid AND Orders.status = 'Pending' AND Asset.asset_type
= 'Commodity';
    RETURN total_cost;
END$$
```



```sql
DELIMITER ;

DELIMITER $$

DELIMITER ;

DELIMITER $$

-- Function to get total cost for equity orders
CREATE FUNCTION GetTotalCostEquity(uid INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE total_cost DECIMAL(10,2);
    -- Fetch the total cost for equity orders
    SELECT COALESCE(SUM(price * qty), 0) INTO total_cost
    FROM Orders
    JOIN Asset ON Orders.aid = Asset.aid
    WHERE Orders.uid = uid AND Asset.asset_type = 'Equity';
    RETURN total_cost;
END$$

-- Function to get total cost for commodity orders
CREATE FUNCTION GetTotalCostCommodity(uid INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE total_cost DECIMAL(10,2);
    -- Fetch the total cost for commodity orders
    SELECT COALESCE(SUM(price * qty), 0) INTO total_cost
    FROM Orders
    JOIN Asset ON Orders.aid = Asset.aid
    WHERE Orders.uid = uid AND Asset.asset_type = 'Commodity';
    RETURN total_cost;
END$$


DELIMITER ;
```

For calculating dashboard values



RELIANCE	₹2430.52
TCS	₹4414.11
INFY	₹1784.38
ICICIBANK	₹1082.19
HINDUNILVR	₹2835.86
HDFCBANK	₹1786.20
BAJFINANCE	₹6614.24
KOTAKBANK	₹1795.30
BHARTIARTL	₹1850.03

Hi, Aditya

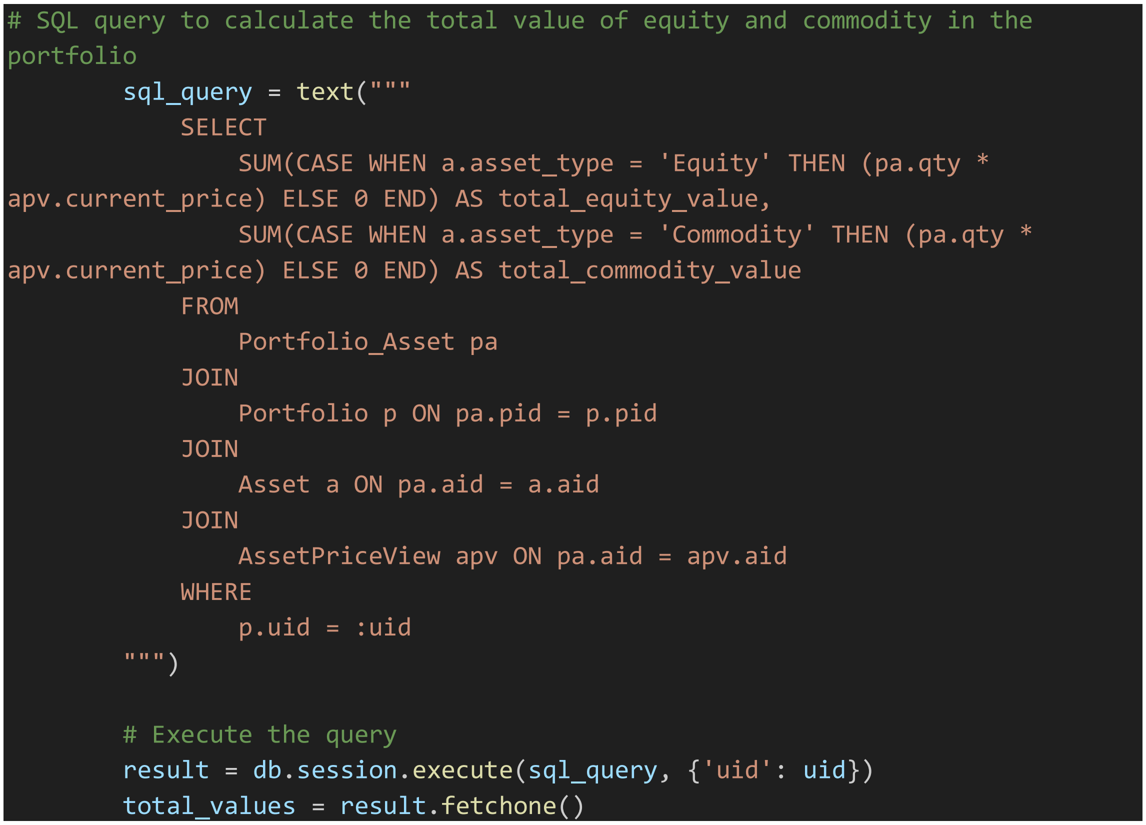
Equity
₹1,00,000.00
Margin Available
Funds: ₹0.00
Utilized Margin: ₹0.00

Commodity
₹1,00,000.00
Margin Available
Funds: ₹0.00
Utilized Margin: ₹0.00

Total Portfolio Value
₹0.00
Total Profit: ₹0.00
Profit Percentage: 0.00%

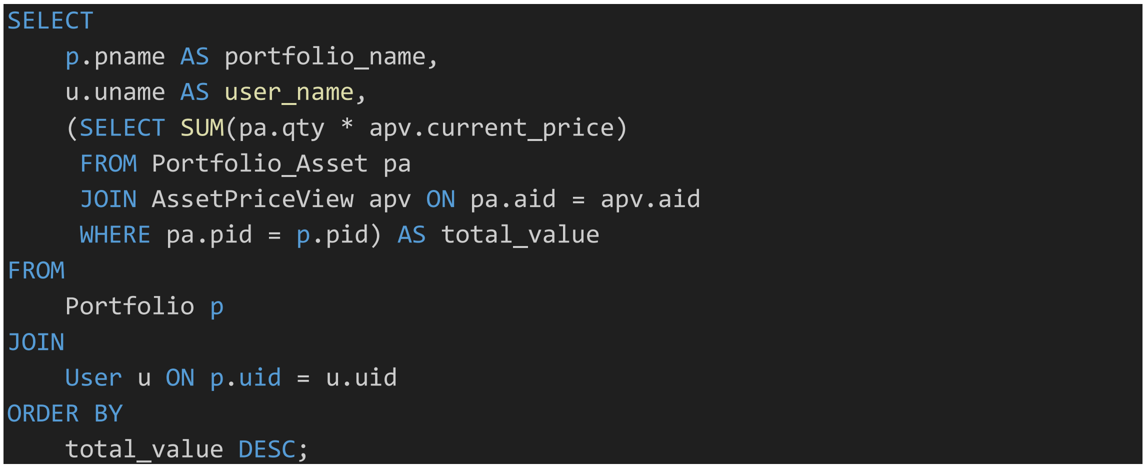
Portfolio Distribution

Join query (GUI):



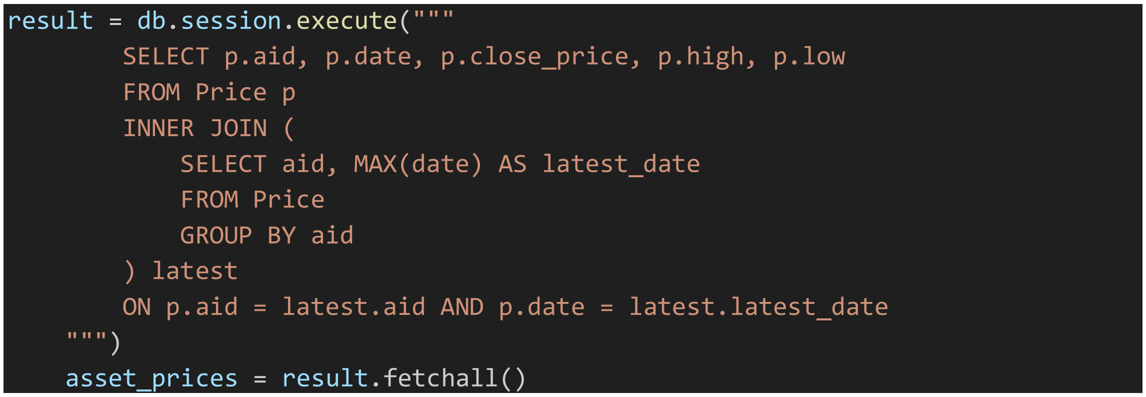
```
# SQL query to calculate the total value of equity and commodity in the portfolio
        sql_query = text("""
            SELECT
                SUM(CASE WHEN a.asset_type = 'Equity' THEN (pa.qty * apv.current_price) ELSE 0 END) AS total_equity_value,
                SUM(CASE WHEN a.asset_type = 'Commodity' THEN (pa.qty * apv.current_price) ELSE 0 END) AS total_commodity_value
            FROM
                Portfolio_Asset pa
            JOIN
                Portfolio p ON pa.pid = p.pid
            JOIN
                Asset a ON pa.aid = a.aid
            JOIN
                AssetPriceView apv ON pa.aid = apv.aid
            WHERE
                p.uid = :uid
        """)

        # Execute the query
        result = db.session.execute(sql_query, {'uid': uid})
        total_values = result.fetchone()
```

Nested query (GUI):



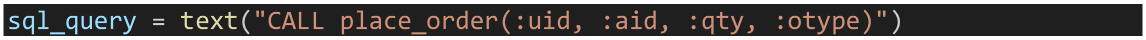
```
SELECT
    p.pname AS portfolio_name,
    u.uname AS user_name,
    (SELECT SUM(pa.qty * apv.current_price)
     FROM Portfolio_Asset pa
     JOIN AssetPriceView apv ON pa.aid = apv.aid
     WHERE pa.pid = p.pid) AS total_value
FROM
    Portfolio p
JOIN
    User u ON p.uid = u.uid
ORDER BY
    total_value DESC;
```

Aggregate query (GUI)

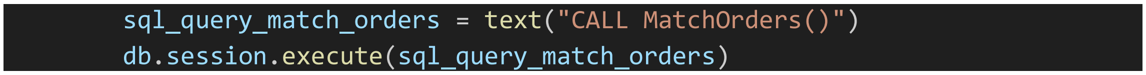


```
result = db.session.execute("""
        SELECT p.aid, p.date, p.close_price, p.high, p.low
        FROM Price p
        INNER JOIN (
            SELECT aid, MAX(date) AS latest_date
            FROM Price
            GROUP BY aid
        ) latest
        ON p.aid = latest.aid AND p.date = latest.latest_date
    """)
    asset_prices = result.fetchall()
```

Calling:



```
sql_query = text("CALL place_order(:uid, :aid, :qty, :otype)")
```



```
        sql_query_match_orders = text("CALL MatchOrders()")
        db.session.execute(sql_query_match_orders)
```

Github repo link:

https://github.com/TheAditya700/stockapp