PERSONAL FINANCIAL PORTFOLIO MANAGER

A MINI-PROJECT REPORT-

CS23333 OOPS Using Java

Submitted by

AFRIL BIVISHA B 240701021

RUPASHRI S K 240701445

In partial fulfillment of the award of the degree

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE,

CHENNAI

An Autonomous Institute

BONAFIDE CERTIFICATE

Certified that this project "**PERSONAL FINANCIAL PORTFOLIO MANAGER"** is the Bonafide work of **"Afril Bivisha B, Rupashri S K"** who carried out the project work under my supervision.

SIGNATURE
Mrs.Deepa B
PROFESSOR,
Dept. of Computer Science and Engineering,
Rajalakshmi Engineering College,
Chennai.

This mini project report is submitted for the viva voce examination to be held on _____

INTERNAL EXAMINER _____ EXTERNAL EXAMINER _____

ABSTRACT

The Personal Financial Portfolio Manager is a Java and SQL-based web application designed to provide users with a secure and centralized platform for managing their personal finances. The system allows users to create accounts, log in securely, check their balance, deposit and withdraw funds, view complete transaction history, and manage their investments in an organized manner. By automating these operations, the system eliminates manual record- keeping, reduces errors, and ensures faster and more reliable financial tracking. Java is used to build the user interface and business logic, while MySQL handles data storage, ensuring consistency, accuracy, and secure database transactions. The use of stored procedures for critical operations like deposits and withdrawals enhances performance, prevents data manipulation, and maintains transactional integrity. The system follows a multi-layer architecture for better scalability and maintainability, making it suitable for individual users as well as future expansion into professional finance management. Overall, this project demonstrates an efficient and user-friendly approach to maintaining personal financial records while ensuring data privacy, security, and long-term reliability.

ACKNOWLEDGEMENT

We express our sincere thanks to our beloved and honorable chairman **MR. S. MEGANATHAN** and the chairperson **DR. M. THANGAM MEGANATHAN** for their timely support and encouragement. We are greatly indebted to our respected and honorable principal **Dr. S. N. MURUGESAN** for his able support and guidance. Our heartfelt thanks to our Head of Department **Dr. E. M. MALATHY** and Deputy Head **Dr. J. MANORANJINI** for their constant encouragement throughout our project. We also extend our sincere gratitude to our internal guide **Mrs. Deepa B** for her valuable guidance and motivation during the completion of this project. Finally, we thank our family members, friends, and all staff members of the Computer Science and Engineering department for their continuous support.

240701021 AFRIL BIVISHA B

240701054 ASHIKA R

TABLE OF CONTENT

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In today's digital era, financial management has become an essential part of every individual's daily life. Whether it is maintaining savings, tracking expenses, monitoring investments, or managing account balances, people constantly deal with financial data that must be recorded, secured, and managed efficiently. Traditionally, most individuals rely on passbooks, notebooks, Excel sheets, or multiple mobile apps that offer fragmented solutions. These methods, however, are prone to errors, lack security, and require manual intervention, leading to inaccurate records and poor financial planning. As personal finance becomes increasingly complex with the rise of digital banking, online transactions, and multiple investment avenues, there is a growing demand for a unified platform that can automate, track, and present financial information in a structured and user-friendly manner.

To address this need, the Personal Financial Portfolio Manager has been developed as a web-based solution using Java and SQL. This system provides a centralized platform where users can securely record and monitor their financial activities, including deposits, withdrawals, account balance, and investment details. By integrating essential features into a single interface, the application eliminates the need for multiple applications or manual bookkeeping. The system also ensures transparency and reliability by maintaining transaction logs and enabling users to view a complete history of their financial operations at any time.

The main focus of the system is to combine usability and security. Java is used as the front-end and business logic layer because of its platform independence, object-oriented structure, vast API support, and robustness in handling web and enterprise applications. SQL is used in the back-end to manage and store user and transaction data securely with accuracy and reliability. Using MySQL as the database enables fast querying, indexing, and structured data organization which are crucial for financial applications. The implementation of stored procedures for deposits, withdrawals, and transaction history ensures that operations are handled atomically, preventing data corruption and race conditions. These procedures also help secure the system against SQL injection and unauthorized access while improving performance.

The Personal Financial Portfolio Manager follows a **three-tier architecture**, consisting of the presentation layer, business logic layer, and data layer. The presentation layer interacts directly with the user and handles input/output operations through web pages or GUI components. The business logic layer acts as the core of the system, processing user input, validating transactions, recording updates, and enforcing business rules. The data layer handles database connectivity, storage, retrieval, and management of financial records, ensuring that each transaction is stored accurately and can be retrieved efficiently. This layered architecture makes the system modular, scalable, and easier to maintain or upgrade in the future.

The application includes multiple core functionalities designed to simulate real-world banking and investment operations. Users can log into their accounts, perform fund transfers, check their balance, and track their investment performance. Every transaction is automatically recorded and can be reviewed in the transaction history module. The system also includes basic investment management features where users can record investment categories, amounts, returns, and duration. Over time, this module can be expanded to support portfolio analysis, profit/loss calculation, and integration with real-time market data.

Security is a major consideration in financial applications. The system incorporates authentication mechanisms, hashed passwords, validation checks, and controlled access to ensure that sensitive user data is protected. Logical constraints are applied at both application and database levels to prevent invalid operations, overdrafts, or unauthorized withdrawals. Additionally, the use of stored procedures ensures that financial rules are executed consistently, regardless of user input.

From an academic perspective, this project demonstrates the real-world application of Java programming, SQL database management, web-based user interaction, system architecture design, module integration, and secure coding practices. It reflects important software engineering principles such as modular development, reusability, data consistency, validation, and error handling. The project also emphasizes the importance of database-driven systems in modern applications and shows how business logic and persistence layers must work together to build reliable financial software.

In conclusion, the Personal Financial Portfolio Manager provides a practical, secure, and efficient solution for tracking personal financial data and managing day-to-day transactions. It simplifies financial record-keeping, reduces human errors, enhances data security, and offers users complete visibility over their financial status. With further improvements such as budget analytics, mobile app integration, multi-currency support, and AI-driven profiling, this system has the potential to evolve into a fully-fledged personal finance assistant for users across different domains. The current implementation lays a strong foundation for such future enhancements while fulfilling the core requirements of a centralized personal finance management system using Java and SQL.

CHAPTER 2

SYSTEM

SPECIFICATIONS

2.1 Hardware Requirements

- Processor: intel core i5
- Memory Size: 8GB RAM
- HardDiskDrive: 256GB

2.2 Software Requirements

- Operating System: Windows 11
- Front-end: Java
- Back-end: MYSQL

CHAPTER 3

MODULE DESCRIPTION

The Personal Financial Portfolio Manager system is divided into several functional modules to improve clarity, maintainability, and scalability. Each module handles a specific set of operations and interacts with the database through secure SQL queries and stored procedures. The primary modules of the system are described below:

3.1 User Management Module

This module is responsible for handling all user-related operations such as registration, login, session management, and authentication. Passwords are stored in hashed form for security, and login credentials are validated before granting access.

Functionality:

- User registration
- Login authentication
- Logout and session handling

Key Data: user_id, username, password_hash, email, created_at
Associated Components: UserController.java, UserService.java, users table, login.jsp

3.2 Account Management Module

This module manages the user's account balance and ensures that every financial activity directly affects the stored balance in the database. It interacts with stored procedures to maintain data accuracy and consistency.

Functionality:

- View account balance
- Update balance automatically after transactions

Key Data: account_id, user_id, balance
Associated Components: AccountController.java, AccountService.java, accounts table

3.3 Deposit Module

The deposit module allows users to add money to their portfolio account. Every deposit is recorded with timestamp and updated balance for accurate tracking. Stored procedures ensure secure and atomic updates.

Functionality:

- Add funds
- Update balance
- Generate transaction log

Key API / DB Operation: CALL sp_deposit(accountId, amount)
Associated Components: DepositController.java, sp_deposit stored procedure

3.4 Withdrawal Module

This module handles fund withdrawals while preventing overdrafts and invalid transactions. It includes balance validation and transaction safety using database-level locking.

Functionality:

- Withdraw money
- Validate balance
- Log withdrawal history

Key API / DB Operation: CALL sp_withdraw(accountId, amount)
Associated Components: WithdrawController.java, sp_withdraw stored procedure

3.5 Transaction History Module

This module allows users to view their entire transaction history including deposits, withdrawals, and investment records. Data is displayed in chronological order with date, type, and updated balance.

Functionality:

- Retrieve and display transaction list
- Filter by date or type
- View transaction summary

Key Data: transaction_id, account_id, type, amount, balance_after, created_at
Associated Components: TransactionController.java, transactions table

3.6 Investment Management Module

This module allows users to record investments and track their portfolio. Users can add investment entries and review them in the dashboard for reference and financial planning.

Functionality:

- Add investment details
- View and manage existing investments
- Track investment performance (basic level)

Key Data: investment_id, user_id, type, amount, date, status
Associated Components: InvestmentController.java, investments table

3.7 Core System Module

This module contains all the system configuration, application setup, database connection logic, and resource initialization.

Functionality:

- Application startup
- Database connection management
- Stored procedure execution

Associated Files: application.properties, DBConnection.java, main application class

CHAPTER – 4

ARCHITECTURE DIAGRAM & ER DIAGRAM

4.1 ARCHITECTURE:

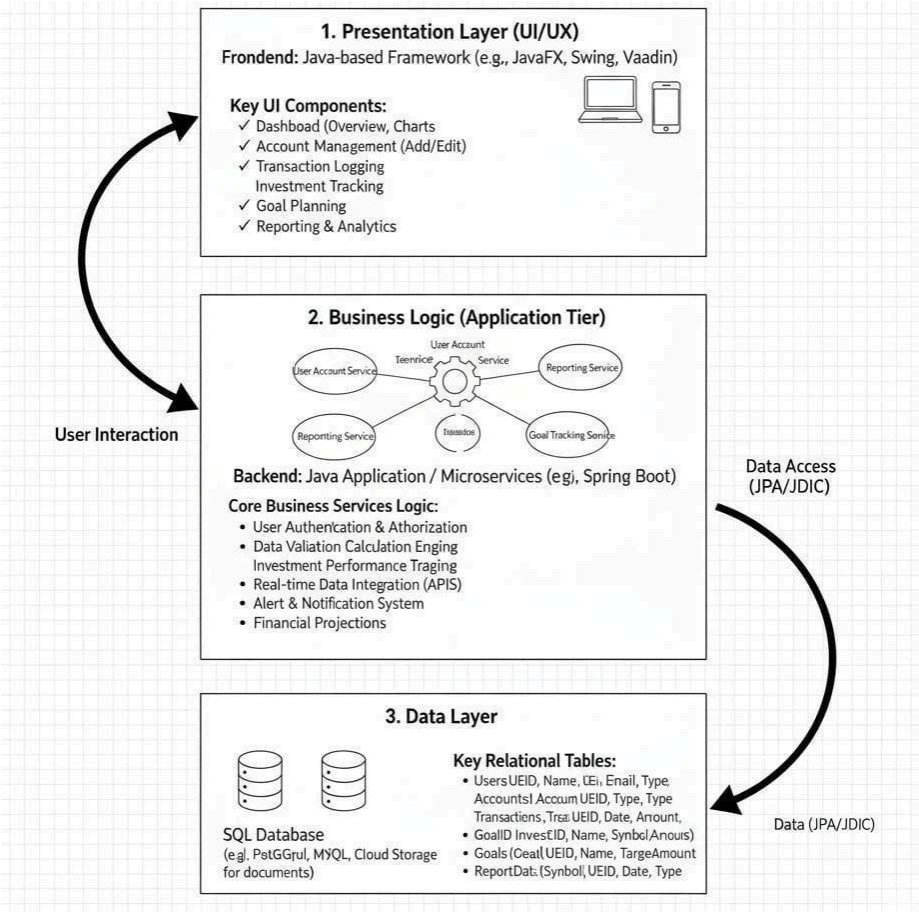


FIG NO: 4.1 ARCHITECTURE DIAGRAM

Architecture Overview

The Personal Financial Portfolio Manager is designed using a **Three-Tier Architecture**, which divides the system into three independent layers

1. **Presentation Layer (Front-End)**

   This layer is responsible for interacting with the user. It includes all graphical elements, input forms, dashboards, and pages where users perform operations such as login, depositing funds, viewing balance, and viewing transaction history. The layer is developed using Java- based technologies such as JSP/Servlets or Spring Boot with HTML, CSS, and JavaScript. It performs validation on user input and communicates with the Business Logic Layer through HTTP requests.

2. Business Logic Layer (Application Layer / Service Layer)

The Business Logic Layer acts as the central processing unit of the system. It handles requests from the UI, applies business rules, checks balance limits, executes transactions, and connects to the database. This layer executes stored procedures for secure operations such as deposits and withdrawals. It ensures the consistency of financial operations through centralized validation and exception handling.

3. Data Layer (Database Layer / Persistence Layer)

This layer is built using MySQL and stores all the information regarding users, accounts, transactions, and investments. It manages data using SQL queries and stored procedures to ensure accurate and secure financial processing. All critical operations such as fund transfers, balance updates, and transaction logging are handled at this layer to maintain integrity and prevent data loss.

4.2 E-R DIAGRAM :

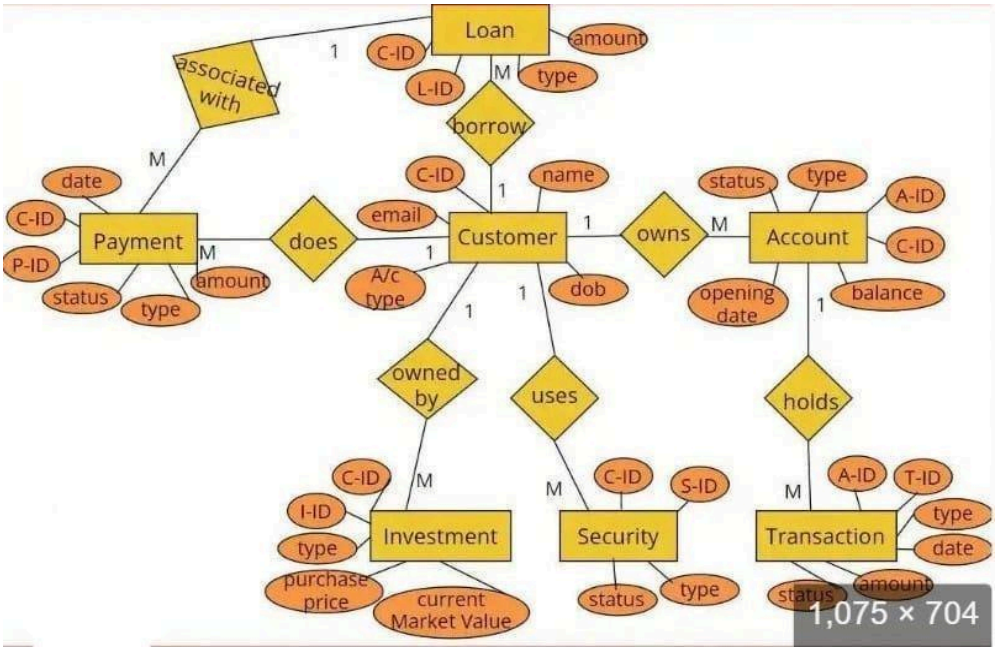


FIG NO: 4.2 ENTITY RELATIONSHIP DIAGRAM.

Entities

1. **User** – Represents registered users who manage their personal financial accounts.
    - **Attributes:** User_ID, Username, Password, Email, Created_At
2. **Account** – Represents the financial account maintained by each user.
    - **Attributes:** Account_ID, User_ID, Balance, Created_At
3. **Transaction** – Stores detailed logs of all deposits, withdrawals, or investment-related transactions.
    - **Attributes:** Transaction_ID, Account_ID, Type, Amount, Balance_After, Note, Date_Time
4. **Investment** – Represents investment entries added by users to track their portfolio.
    - **Attributes:** Investment_ID, User_ID, Investment_Type, Amount, Date_Invested, Status

Relationships

1. **User – Account → One-to-One**

   Each user has one account, but every account belongs to a single user.

2. **Account – Transaction → One-to-Many**

   A single account can have multiple transactions recorded in the system.

3. **User – Investment → One-to-Many**

   A user can create and manage multiple investment entries.

4. **User – Transaction → Indirect One-to-Many**

   A user indirectly owns multiple transactions through their account.

Connections

User ↔ Account ↔ Transaction

- Represents how a user owns an account and performs multiple transactions through that account.

User ↔ Investment

- Shows that investment entries are directly linked to the user who created them.

Administrator ↔ System Data

- Represents administrative control used for monitoring and maintaining system integrity.

Account ↔ Transaction

- Establishes the link between each account and its financial integrity.

CHAPTER 5

IMPLEMENTATION

```
import java.sql.*;
import
java.util.*;
import java.time.LocalDateTime;
class DatabaseConnection {
   private static final String URL =
"jdbc:mysql://localhost:3306/finance_db";
   private static final String USER = "root"; // replace with your
DB username
   private static final String PASSWORD = "password"; //
replace with your DB password

   public static Connection getConnection()
throws SQLException {
      return DriverManager.getConnection(URL, USER,
PASSWORD);
   }
}
class User {
   int
   userId;
   String username;
   double balance;
   public User(int userId, String username, double balance) {
      this.userId = userId;
      this.username = username;
      this.balance = balance;
```

```
    }
}
```

```
class Transaction {
   int transactionId;
   String type;
   double amount;
   LocalDateTime dateTime;
   public Transaction(int transactionId, String type, double
amount, LocalDateTime dateTime) {
      this.transactionId = transactionId;
      this.type = type;
      this.amount = amount;
      this.dateTime = dateTime;
   }
}
class Investment {
   int
   investmentId;
   String investmentType;
   double amount;
   LocalDateTime dateTime;
   public Investment(int investmentId, String investmentType,
double amount, LocalDateTime dateTime) {
      this.investmentId = investmentId;
      this.investmentType = investmentType;
      this.amount = amount;
      this.dateTime = dateTime;
   }
}
class UserService {
   public static User login(String username, String password) {
      String sql = "SELECT * FROM users WHERE username = ?
AND password = ?";
```

```
try (Connection conn =
```

```
DatabaseConnection.getConnection();
            PreparedStatement pst = conn.prepareStatement(sql))
           { pst.setString(1, username);
           pst.setString(2, password);
           ResultSet rs = pst.executeQuery();
           if (rs.next()) {
              return new User(rs.getInt("user_id"),
rs.getString("username"), rs.getDouble("balance"));
           }
       } catch (SQLException e)
           { e.printStackTrace();
       }
       return null;
   }
   public static void depositFunds(User user, double amount) {
       if (amount <= 0) {
           System.out.println("Amount must be greater than 0!");
           return;
       }
       String updateBalance = "UPDATE users SET balance =
balance + ? WHERE user_id = ?";
       String addTransaction = "INSERT INTO
transactions(user_id, type, amount) VALUES (?, 'Deposit', ?)";
       try (Connection conn =
DatabaseConnection.getConnection()) {
           conn.setAutoCommit(false);
           try (PreparedStatement pst1
           =
conn.prepareStatement(updateBalance);
                PreparedStatement pst2 =
conn.prepareStatement(addTransaction)) {
```

```
            pst1.setDouble(1, amount);
            pst1.setInt(2, user.userId);
            pst1.executeUpdate();
            pst2.setInt(1, user.userId);
            pst2.setDouble(2, amount);
            pst2.executeUpdate();
            conn.commit();
            user.balance += amount;
            System.out.println("Deposit successful! Current
balance: " + user.balance);
        } catch (SQLException e)
            { conn.rollback();
            e.printStackTrace();
        }
    } catch (SQLException e)
        { e.printStackTrace();
    }
  }
  public static void withdrawFunds(User user, double amount)
{
    if (amount <= 0 || amount > user.balance) {
        System.out.println("Invalid withdrawal amount!");
        return;
    }
    String updateBalance = "UPDATE users SET balance =
balance - ? WHERE user_id = ?";
    String addTransaction = "INSERT INTO
transactions(user_id, type, amount) VALUES (?, 'Withdrawal',
?)";
    try (Connection conn =
DatabaseConnection.getConnection()) {
```

```
            conn.setAutoCommit(false);
            try (PreparedStatement pst1
            =
conn.prepareStatement(updateBalance);
                 PreparedStatement pst2 =
conn.prepareStatement(addTransaction))
                { pst1.setDouble(1, amount);
                pst1.setInt(2, user.userId);
                pst1.executeUpdate();
                pst2.setInt(1, user.userId);
                pst2.setDouble(2, amount);
                pst2.executeUpdate();
                conn.commit();
                user.balance -= amount;
                System.out.println("Withdrawal successful! Current
balance: " + user.balance);
            } catch (SQLException e)
                { conn.rollback();
                e.printStackTrace();
            }
        } catch (SQLException e)
            { e.printStackTrace();
        }
    }
    public static void viewTransactionHistory(User user) {
        String sql = "SELECT * FROM transactions WHERE
        user_id
= ?";
        try (Connection conn =
DatabaseConnection.getConnection();
             PreparedStatement pst = conn.prepareStatement(sql))
```

```
{ pst.setInt(1, user.userId);
ResultSet rs = pst.executeQuery();
```

```
        System.out.println("----- Transaction History        ");
        while (rs.next()) {
          System.out.printf("%d | %s | %.2f | %s\n",
                rs.getInt("transaction_id"),
                rs.getString("type"),
                rs.getDouble("amount"),
               rs.getTimestamp("date_time").toLocalDateTime());
        }
      } catch (SQLException e)
        { e.printStackTrace();
      }
   }
   public static void manageInvestment(User user, String type,
double amount) {
      if (amount <= 0 || amount > user.balance) {
        System.out.println("Invalid investment amount!");
        return;
      }
      String updateBalance = "UPDATE users SET balance =
balance - ? WHERE user_id = ?";
      String addInvestment = "INSERT INTO
investments(user_id, investment_type, amount) VALUES (?, ?,
?)";

      try (Connection conn =
DatabaseConnection.getConnection()) {
        conn.setAutoCommit(false);
        try (PreparedStatement pst1
        =
conn.prepareStatement(updateBalance);
            PreparedStatement pst2 =
```

conn.prepareStatement(addInvestment)) {

```
            pst1.setDouble(1, amount);
            pst1.setInt(2, user.userId);
            pst1.executeUpdate();
            pst2.setInt(1, user.userId);
            pst2.setString(2, type);
            pst2.setDouble(3, amount);
            pst2.executeUpdate();
            conn.commit();
            user.balance -= amount;
            System.out.println("Investment successful! Current
balance: " + user.balance);
        } catch (SQLException e)
            { conn.rollback();
            e.printStackTrace();
        }
    } catch (SQLException e)
        { e.printStackTrace();
    }
  }

public static void viewInvestments(User user) {
    String sql = "SELECT * FROM investments WHERE user_id
= ?";
    try (Connection conn =
DatabaseConnection.getConnection();
         PreparedStatement pst = conn.prepareStatement(sql))
        { pst.setInt(1, user.userId);
        ResultSet rs = pst.executeQuery();
        System.out.println("----- Investment Portfolio
        ");
```

```
while (rs.next()) {
```

```
            System.out.printf("%d | %s | %.2f | %s\n",
                    rs.getInt("investment_id"),
                    rs.getString("investment_type"),
                    rs.getDouble("amount"),
  rs.getTimestamp("date_time").toLocalDateTime());
          }
       } catch (SQLException e)
          { e.printStackTrace();
       }
    }
}

public class PersonalFinanceManager {
    public static void main(String[] args) {
       Scanner sc = new
       Scanner(System.in); User
       currentUser = null;
        while (true) {
          System.out.println("=== Personal Financial Portfolio
Manager ===");
          System.out.println("1. Login");
          System.out.println("2. Exit");
          System.out.print("Choose option:
          "); int choice = sc.nextInt();
          sc.nextLine(); // consume newline
          if (choice == 1) {
             System.out.print("Username: ");
             String          username          =
             sc.nextLine();
             System.out.print("Password:  ");
             String          password          =
```

```
            sc.nextLine();
            currentUser                                    =
UserService.login(username, password);
```

```
            if (currentUser != null) {
                System.out.println("Login successful! Welcome, " +
currentUser.username);
                userDashboard(sc, currentUser);
            } else {
                System.out.println("Invalid credentials!");
            }
        } else if (choice == 2) {
            System.out.println("Goodbye!");
            break;
        } else {
            System.out.println("Invalid option!");
        }
    }
    sc.close();
  }

  public static void userDashboard(Scanner sc, User user) {
    while (true) {
        System.out.println("\n--- Dashboard ---");
        System.out.println("1. Check Balance");
        System.out.println("2. Deposit Funds");
        System.out.println("3. Withdraw Funds");
        System.out.println("4. View Transaction History");
        System.out.println("5. Manage Investments");
        System.out.println("6. View Investments");
        System.out.println("7. Logout");
        System.out.print("Choose option: ");
        int choice = sc.nextInt();
        sc.nextLine(); // consume newline
```

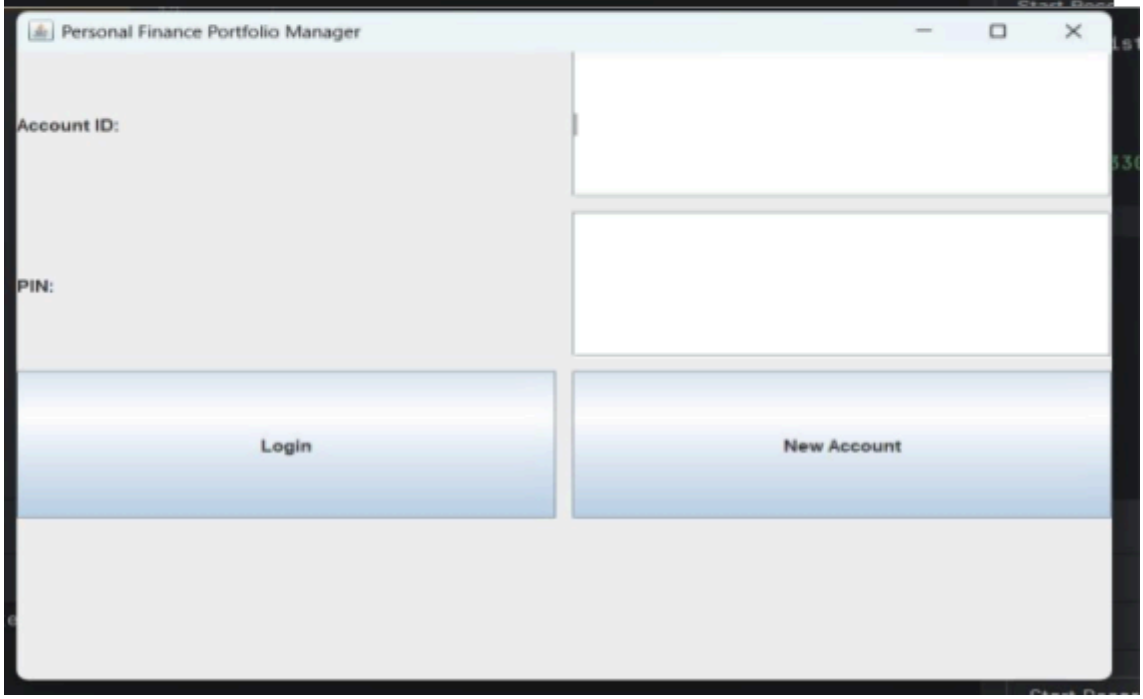
```
        switch (choice) {
            case 1:
                System.out.println("Current Balance: " +
user.balance);
                break;
            case 2:
                System.out.print("Enter deposit amount: ");
                double deposit = sc.nextDouble();
                UserService.depositFunds(user, deposit);
                break;
            case 3:
                System.out.print("Enter withdrawal amount: ");
                double withdraw = sc.nextDouble();
                UserService.withdrawFunds(user, withdraw);
                break;
            case 4:
                UserService.viewTransactionHistory(user);
                break;
            case 5:
                System.out.print("Enter investment type (e.g.,
Stocks, Bonds): ");
                String type = sc.nextLine();
                System.out.print("Enter investment amount: ");
                double amount = sc.nextDouble();
                UserService.manageInvestment(user, type, amount);
                break;
            case 6:
                UserService.viewInvestments(user);
                break;
            case 7:
                System.out.println("Logged out successfully!");
```

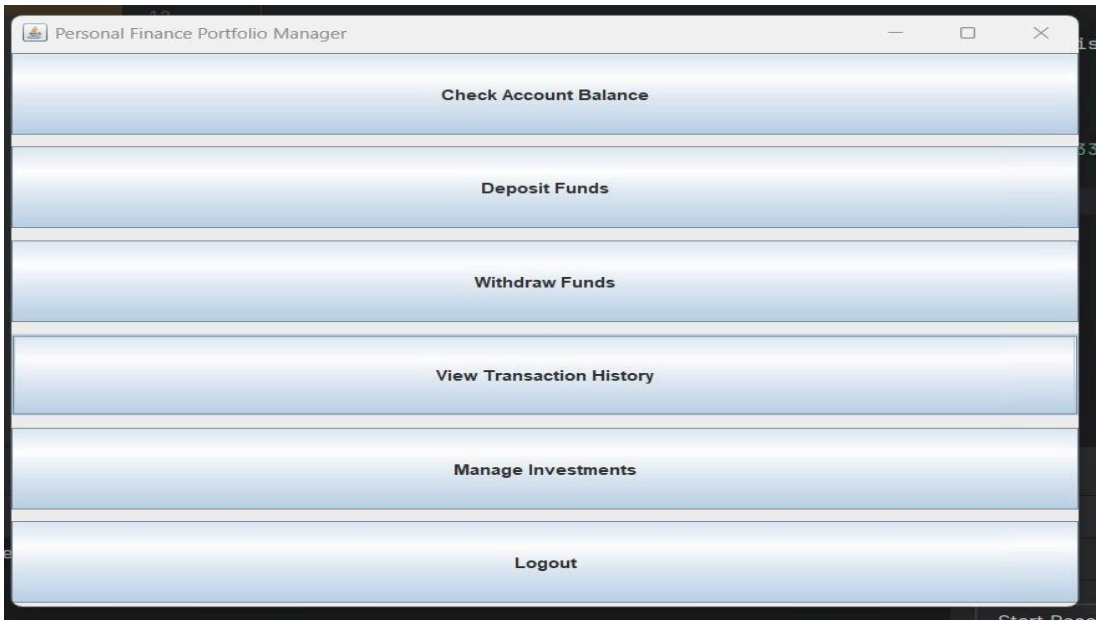
```
                return
            ; default:
                System.out.println("Invalid option!");
        }
    }
  }
}
```

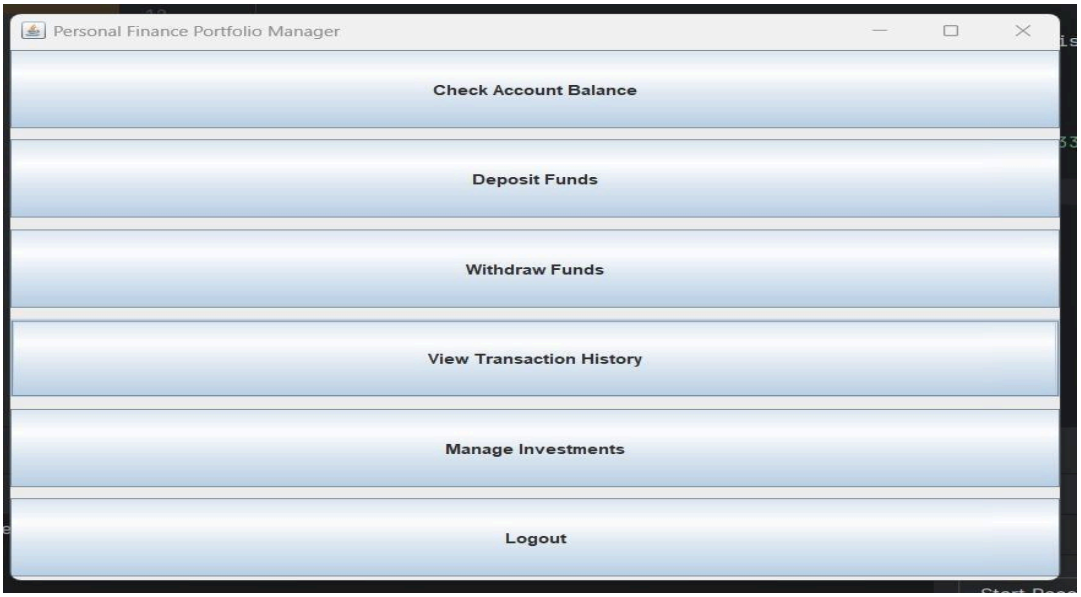
CHAPTER 6

SCREENSHOT

S




Personal Finance Portfolio Manager
Check Account Balance
Deposit Funds
Withdraw Funds
View Transaction History
Manage Investments
Logout


Personal Finance Portfolio Manager
Check Account Balance
Deposit Funds
Withdraw Funds
View Transaction History
Manage Investments
Logout

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

Conclusion

The Personal Financial Portfolio Manager project showcases how Java and SQL can be combined to build a functional system for managing personal finances. By implementing features such as login, balance checking, deposit and withdrawal, transaction history, and investment management, the system provides users with a secure and efficient way to monitor and control their financial activities. The use of a MySQL database ensures that all user and transaction data is stored reliably and can be retrieved accurately when needed.

Through this project, important programming concepts were applied, including object-oriented design, database connectivity using JDBC, and proper handling of financial transactions with commit and rollback operations. The modular structure of the code, with separate classes for users, transactions, and investments, not only makes the system organized and maintainable but also allows for easy future enhancements, such as adding reports, notifications, or more advanced investment options.

Overall, this project demonstrates practical skills in both software development and financial management logic. It serves as a strong foundation for building more complex financial applications while providing valuable experience in designing secure, user-friendly, and reliable systems. The project effectively bridges theoretical programming knowledge with real-world application.

Future Enhancement

1. **Graphical User Interface(GUI):**

   Currently, the project uses a command-line interface. Implementing a GUI with Java Swing or JavaFX would make the system more user-friendly and visually appealing.

2. **Advanced Investment Options:**

   Introduce features like tracking stocks, mutual funds, and recurring investments. Users could receive investment suggestions or alerts based on market trends.

3. **Reports and Analytics:**

   Add monthly or yearly financial reports, charts, and summaries to help users better understand their spending, savings, and investment patterns.

4. **Security improvements:**

   Enhance security by encrypting passwords, adding multi-factor authentication, and protecting sensitive data during transactions.

5. **Integration with Online Banking APIs:**

   Allow users to link their bank accounts for automatic balance updates, fund transfers, and real-time transaction tracking.

6. **Mobile or Web Version:**

   Extend the application to a mobile app or web platform for easier access and convenience from anywhere.

REFERENCES

1. https://www.w3schools.com/java/

2. https://www.tutorialspoint.com/java/

3. https://www.geeksforgeeks.org/java/

4. https://www.oracle.com/java/technologies/

5. https://www.github.com/