

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi - 590018



TECHNICAL TRAINING

Mini project On

“CUSTOMER’S VIRTUAL WALLET ”

By

ADITI KAMATH

4MT21CS010

ARAVIND

4MT21CS026

ARJUN

4MT21CS027

ASHRITHA S

4MT21CS032

BHAGYALAKSHMEE

4MT21CS035



DEPARTMENT OF COMPUTER SCIENCE& ENGINEERING

(Accredited by NBA)

**MANGALORE INSTITUTE OF TECHNOLOGY &
ENGINEERING**

Accredited by NAAC with A+ Grade, An ISO 9001: 2015 Certified Institution

(A Unit of Rajalaxmi Education Trust®, Mangalore - 575001)

Affiliated to VTU, Belagavi, Approved by AICTE, New Delhi

Badaga Mijar, Moodabidri-574225, Karnataka 2022-23

Table of Contents

1. Abstract 2. Introduction

2.1 Background

2.2 Objectives

3. Technologies Used

4. System Architecture

4.1 Front-End

4.2 Back-End

4.3 Database

5. Project Modules

5.1 Module 1: User Registration and Authentication

5.2 Module 2: Fund Management

5.3 Module 3: Transaction Recording

5.4 Module 4: Data Security

6. Design and Implementation

6.1 Front-End Design

6.2 Back-End Design

6.3 Database Design

7. Features and Functionality

7.1 Feature 1: Transaction History

7.2 Feature 2: Balance Checking

7.3 Feature 3: Fund Management

8. Testing

8.1 Unit Testing

8.2 Integration Testing

8.3 User Acceptance Testing

9. Challenges Faced

10. Future Enhancements

11. Conclusion

12. References

13. Appendices

13.1 Screenshots

1. Abstract

The Customer's Virtual Wallet project is a comprehensive software application designed to facilitate secure and convenient financial transactions for users in a digital environment. This project focuses on implementing basic CRUD (Create, Read, Update, Delete) operations to manage user funds. Users can leverage this application to record and monitor their financial transactions, specifically when making purchases at various establishments such as stores, shops, and cafes. The project employs file handling concepts to ensure robust data storage and retrieval.

This report provides an in-depth exploration of the Customer's Virtual Wallet project, including its objectives, technological foundations, system architecture, major modules, design and implementation details, features and functionality, testing methodologies, challenges faced, future enhancements, and a concluding summary of the project's outcomes. As the financial landscape continues to evolve towards digitalization, this application stands as a valuable tool for individuals seeking a secure and user-friendly platform to manage their funds, make informed financial decisions, and track their spending habits. Through this report, readers will gain a comprehensive understanding of the project's significance and its potential to address the evolving needs of modern consumers.

2. Introduction

2.1 Background

In an era marked by the digital transformation of financial services, the development of the Customer's Virtual Wallet project becomes essential to address the evolving needs of modern consumers. The project stems from the recognition of several key factors in today's financial landscape:

2.1.1 Increasing Digitization

The rapid transition from physical cash transactions to digital payments has reshaped the way individuals manage their finances. With the proliferation of smartphones and online shopping, consumers expect convenient, secure, and accessible ways to handle their funds.

2.1.2 Need for Fund Management

People require efficient tools to manage their funds, monitor transactions, and gain insights into their financial habits. Traditional methods of tracking expenses on paper or through spreadsheets are becoming obsolete and cumbersome.

2.1.3 Security Concerns

With the increasing prevalence of digital financial transactions, ensuring the security of sensitive financial data has become paramount. Users seek trustworthy platforms that protect their financial information from unauthorized access and fraud.

2.1.4 Record-Keeping

Effective financial management often involves keeping detailed records of income, expenses, and transactions. A digital solution can streamline this process, making it easier for users to maintain a comprehensive record of their financial activities.

2.2 Objectives

2.2.1 Fund Management

The primary aim of this project is to provide users with a robust platform for managing their funds. Users will be able to perform basic CRUD (Create, Read, Update, Delete) operations to add, view, update, and delete their financial transactions.

2.2.2 Secure Transactions

Security is a top priority. The project seeks to ensure that all financial transactions conducted within the application are secure, employing encryption and authentication measures to protect user data.

2.2.3 User Convenience

The application is designed with user-friendliness in mind, offering an intuitive interface that allows users to easily record and monitor their transactions. This convenience extends to making purchases at stores, shops, cafes, and similar establishments.

2.2.4 Data Integrity

To maintain accurate financial records, the project will leverage file handling concepts to store and retrieve user data. This approach ensures data integrity and persistence even in the face of system failures.

3. Technologies Used

The Customer's Virtual Wallet project leverages a combination of technologies, programming languages, and tools to provide users with a seamless and secure financial management

experience. The following is a list of the key technologies and components employed in this project:

3.1 Programming Language

C Programming Language: The core of the project is built using the C programming language. C is chosen for its efficiency, low-level control, and ability to interact directly with the file system.

3.2 File Handling

File Handling Concepts: The project extensively utilizes file handling concepts to store and manage user data. This includes creating, reading, updating, and deleting records of financial transactions.

3.3 Security

Encryption: To ensure the security of user data, encryption techniques are employed to protect sensitive financial information during storage and transmission.

3.4 User Interface

Console-Based User Interface: The user interface is primarily text-based, providing a command-line interface (CLI) for users to interact with the application. This minimalist approach is chosen for its simplicity and ease of use.

3.5 Development Tools

Code Editor : Online Debuggers, VS Code

3.6 Version Control

Version Control System: Git

3.7 Documentation Tools:

Documentation tools : Microsoft Word

4. System Architecture

4.1 Front-End

The front-end of the application is responsible for user interaction, presenting information, and collecting user inputs. Given the minimalist and command-line nature of the application, the front-end is relatively straightforward.

Key Front-End Components and Features:

- **Command-Line Interface (CLI):** Users interact with the application through a text-based CLI, which displays menus and prompts for various operations.
- **Menu Navigation:** Menus allow users to select options for viewing transactions, adding funds, making purchases, and more.
- **User Input Handling:** The front-end processes user inputs, validates them, and triggers appropriate back-end actions.

4.2 Back-End

The back-end of the application contains the logic responsible for executing user requests, managing transactions, and ensuring data security.

Key Back-End Components and Features:

- **CRUD Operations:** The back-end handles basic CRUD operations, including creating, reading, updating, and deleting user transactions and wallet balances.
- **File Handling:** Data is stored and managed using file handling concepts. Each user has a dedicated data file to store their transaction history and wallet balance.
- **Security:** The back-end implements encryption and access control mechanisms to protect user data and ensure that only authorized users can access their records.
- **Business Logic:** The back-end enforces business rules, such as checking available funds before allowing a purchase and updating wallet balances accordingly.

4.3 Database

In this application, file handling is used as a database mechanism. Each user's data is stored in dedicated text files, providing a simple yet effective way to store and retrieve information.

Key Database Features:

- **User Data Files:** Each registered user has a separate data file for storing their transaction history and current wallet balance.
- **Data Persistence:** Data is persisted between sessions, ensuring that users can access their transaction history and current balance even after closing the application.

The separation of front-end, back-end, and database components allows for modularity and ease of maintenance. It also ensures that user data remains secure and that the application functions smoothly.

5. Project Modules

5.1 Module 1: User Registration and Authentication

Function: This module allows users to create accounts, providing their basic information and setting up their virtual wallets. Users must log in to access the application's features.

Interactions: Users register by providing personal details and creating a username and password. Once registered, they can log in with their credentials.

5.2 Module 2: Fund Management

Function: This core module handles the management of user funds. It enables users to perform CRUD operations on their wallet balances, including adding funds, viewing balances, making purchases, and deleting transactions.

Interactions: Users can add funds by specifying the amount. They can also view their current balance, make purchases by deducting funds, and delete specific transactions if necessary.

5.3 Module 3: Transaction Recording

Function: This module records all financial transactions, including purchases, additions of funds, and any other wallet activity. Each transaction is timestamped and associated with the user's account.

Interactions: Transactions are automatically recorded whenever users add funds or make purchases. Users can also view their transaction history.

5.4 Module 4: Data Security

Function: Ensuring the security of user data is a paramount concern. This module is responsible for encrypting sensitive user information and securing the data files containing user transactions and wallet balances.

Interactions: Data security functions transparently in the background, protecting user data from unauthorized access.

6. Design and Implementation

6.1 Front-End Design

Design Principles:

- **Simplicity:** The front-end is designed to be user-friendly and straightforward, with clear menus and prompts.
- **Clarity:** Menus and messages provide clear instructions and feedback to users.
- **Efficiency:** User input is validated and processed efficiently, ensuring a smooth user experience.

User Experience Considerations:

- **Intuitiveness:** The CLI interface is intuitive, allowing users to navigate the application without complexity.
- **Feedback:** The front-end provides immediate feedback to users about the success or failure of their actions.

6.2 Back-End Design

Design Principles:

- **Modularity:** The back-end is modular, making it easy to maintain and extend.
- **Security:** Security measures are integrated to protect user data, including encryption of sensitive information.
- **File Handling:** File handling functions manage user data, ensuring data integrity.

Algorithms and Considerations:

- **CRUD Operations:** Algorithms handle CRUD operations efficiently, including reading, writing, updating, and deleting records.
- **Transaction Processing:** Transaction processing includes checking available funds, updating wallet balances, and recording transactions.

6.3 Database Design

Database Schema:

- **User Data Files:** Each user has a dedicated data file containing transaction records. These records include date, description, transaction type (debit or credit), and amount.
- **File Organization:** Data files are organized and named based on user identifiers, ensuring data separation.

7. Features and Functionality

7.1 Feature 1: Transaction History

Functionality: Users can view their transaction history, including dates, descriptions, and transaction types (debit or credit).

Importance: Transaction history provides a comprehensive overview of a user's financial activities, aiding in budgeting, expense tracking, and financial planning.

7.2 Feature 2: Balance Checking

Functionality: Users can check their current wallet balances at any time to know how much funds are available for spending.

Importance: Balance checking is crucial for informed financial decisions. It prevents users from making purchases without sufficient funds and helps them budget effectively.

7.3 Feature 3: Fund Management

Functionality: Users can perform basic CRUD (Create, Read, Update, Delete) operations on their virtual wallet balances. This includes adding funds, viewing balances, making purchases, and deleting specific transactions.

Importance: Fund management is at the core of the application, enabling users to track their finances, make transactions, and maintain an accurate record of their wallet balances.

8. Testing

8.1 Unit Testing

Methodology: Unit testing was performed to test individual functions and components of the application in isolation. Each function was tested to ensure that it behaves as expected and handles various input scenarios.

Tools: Standard C testing frameworks and manual testing were used to conduct unit tests.

8.2 Integration Testing

Methodology: Integration testing focused on verifying the interaction between different modules and components. This ensured that the modules work seamlessly together as a cohesive system.

Tools: Manual testing was primarily used for integration testing to simulate user interactions and verify data flow between modules.

8.3 User Acceptance Testing

Methodology: User acceptance testing involved real users interacting with the application. Users were provided with test scenarios to perform various tasks, such as adding funds, making purchases, and viewing transaction history.

Feedback: User feedback was collected and analyzed to identify any usability issues, bugs, or areas for improvement.

9. Challenges Faced

The development of the Customer's Virtual Wallet project encountered several challenges:

- **Data Security:** Implementing robust data security measures to protect user information required careful consideration and extensive testing.

- User Experience: Designing an intuitive and user-friendly CLI interface presented challenges in ensuring a smooth user experience.
- Error Handling: Ensuring error-free operations and providing meaningful error messages for users was a complex task.

10. Future Enhancements

While the project currently meets its primary objectives, several future enhancements can be considered:

- Multi-Platform Support: Expanding the application to support multiple platforms, including mobile devices and web browsers, would increase accessibility.
- Enhanced Security: Implementing advanced security features, such as two-factor authentication, would further protect user data.
- Data Analytics: Adding data analytics capabilities to help users gain insights into their spending habits and financial trends.
- Additional Features: Incorporating features like budget planning, bill reminders, and financial goal setting to make the application more comprehensive.

11. Conclusion

In conclusion, the Customer's Virtual Wallet project provides users with a secure and efficient tool for managing their finances in a digital age. By implementing basic CRUD operations, file handling, and data security measures, the application empowers users to track transactions, add funds, make purchases, and view their financial history with ease.

The project highlights the significance of secure and user-friendly financial management tools in today's digital landscape. It serves as a foundation for future enhancements that can further improve user experiences and support financial well-being.

12. References

1. GitHub: GitHub is a popular platform for hosting and sharing code. You can use the GitHub search feature to find open-source projects and code related to virtual wallets in C.
<https://github.com/>

2. SourceForge: SourceForge hosts a variety of open-source software projects, including financial and wallet-related applications in C.

<https://sourceforge.net/>

3. CodeProject: CodeProject is a community of developers that shares code samples and projects. You may find C-based virtual wallet projects and code examples there.

<https://www.codeproject.com/>

4. Stack Overflow: Stack Overflow is a programming Q&A community where developers often share code snippets and solutions to specific problems. You can search for questions related to virtual wallets in C to find relevant code examples.

<https://stackoverflow.com/>

5. Online Coding Forums: Online programming forums like Reddit's r/C_Programming or similar communities can also be a good place to find discussions and code snippets related to virtual wallet implementations in C.

https://www.reddit.com/r/C_Programming/

13. Appendices

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
// Structure to store user information
```

```
struct User {
    int id;
    char username[50];
    float balance;
};
```

```
// Function to create a new user
```

```
struct User createUser(int id, char username[50], float balance) {
    struct User newUser;
    newUser.id = id;
    strcpy(newUser.username, username);
    newUser.balance = balance;
    return newUser;
}
```

```

// Function to display user details
void displayUser(struct User user) {
    printf("User ID: %d\n", user.id);
    printf("Username: %s\n", user.username);
    printf("Balance: $%.2f\n", user.balance);
}

// Function to add money to a user's wallet
void addMoney(struct User *user, float amount) {
    user->balance += amount;
    printf("Added $%.2f to %s's wallet. New balance: $%.2f\n", amount, user->username, user->balance);
}

// Function to make a purchase
void makePurchase(struct User *user, float amount) {
    if (user->balance >= amount) {
        user->balance -= amount;
        printf("Made a purchase of $%.2f. New balance: $%.2f\n", amount, user->balance);
    } else {
        printf("Insufficient balance for the purchase.\n");
    }
}

// Function to find a user by ID
struct User *findUserById(struct User users[], int userCount, int userId) {
    for (int i = 0; i < userCount; i++) {
        if (users[i].id == userId) {
            return &users[i];
        }
    }
    return NULL; // User not found
}

// Function to display the main menu
int showMainMenu() {
    int choice;
    printf("Main Menu:\n");
    printf("1. User\n");
    printf("2. Admin\n");
    printf("3. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    return choice;
}

// Function to handle the user menu

```

```

void handleUserMenu(struct User users[], int *userCount) {
    int choice;
    int userId;
    char username[50];
    float amount;

    // User authentication
    printf("User Login:\n");
    printf("Enter your user ID: ");
    scanf("%d", &userId);
    printf("Enter your username: ");
    scanf("%s", username);

    struct User *user = findUserById(users, *userCount, userId);

    if (user != NULL && strcmp(user->username, username) == 0) {
        printf("User access granted.\n");
    } else {
        printf("User not found or authentication failed. Access denied.\n");
        return;
    }

    while (1) {
        printf("User Menu:\n");
        printf("1. View Wallet\n");
        printf("2. Add Money\n");
        printf("3. Make Purchase\n");
        printf("4. Return to Main Menu\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: // View Wallet
                displayUser(*user);
                break;

            case 2: // Add Money
                printf("Enter the amount to add: $");
                if (scanf("%f", &amount) != 1 || amount <= 0) {
                    printf("Invalid amount.\n");
                    continue; // Continue the loop to re-enter the amount
                }
                addMoney(user, amount);
                break;

            case 3: // Make Purchase
                printf("Enter the purchase amount: $");

```

```

        if (scanf("%f", &amount) != 1 || amount <= 0) {
            printf("Invalid amount.\n");
            continue; // Continue the loop to re-enter the amount
        }
        makePurchase(user, amount);
        break;

    case 4: // Return to Main Menu
        return;
    default:
        printf("Invalid choice.\n");
        break;
    }
}
}

// Define admin username and password
#define ADMIN_USERNAME "admin"
#define ADMIN_PASSWORD "password"

// Function to handle the admin menu
void handleAdminMenu(struct User users[], int *userCount) {
    char username[50];
    char password[50];
    int choice;
    int userId;
    char newUsername[50];
    float newBalance;

    // Admin login step
    printf("Admin Login:\n");
    printf("Username: ");
    scanf("%s", username);
    printf("Password: ");
    scanf("%s", password);

    if (strcmp(username, ADMIN_USERNAME) != 0 || strcmp(password, ADMIN_PASSWORD) != 0) {
        printf("Invalid admin credentials. Access denied.\n");
        return;
    }
    printf("Admin access granted.\n");

    while (1) {
        printf("Admin Menu:\n");
        printf("1. Add User\n");
        printf("2. Edit User\n");
        printf("3. Delete User\n");
    }
}

```

```

printf("4. Return to Main Menu\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1: // Add User
        printf("Enter the new user ID: ");
        scanf("%d", &userId);
        // Check if the user ID already exists
        if (findUserId(users, *userCount, userId) != NULL) {
            printf("User with the same ID already exists.\n");
            continue; // Continue the loop to re-enter the ID
        }
        printf("Enter the new username: ");
        scanf("%s", newUsername);
        printf("Enter the initial balance: $");
        if (scanf("%f", &newBalance) != 1 || newBalance < 0) {
            printf("Invalid balance.\n");
            continue; // Continue the loop to re-enter the balance
        }
        users[*userCount] = createUser(userId, newUsername, newBalance);
        (*userCount)++;
        printf("User added successfully.\n");
        break;
    case 2: // Edit User
        printf("Enter the user ID to edit: ");
        scanf("%d", &userId);
        struct User *userToEdit = findUserId(users, *userCount, userId);
        if (userToEdit != NULL) {
            printf("Enter the new username: ");
            scanf("%s", newUsername);
            printf("Enter the new balance: $");
            if (scanf("%f", &newBalance) != 1 || newBalance < 0) {
                printf("Invalid balance.\n");
                continue; // Continue the loop to re-enter the balance
            }
            strcpy(userToEdit->username, newUsername);
            userToEdit->balance = newBalance;
            printf("User updated successfully.\n");
        } else {
            printf("User not found.\n");
        }
        break;
    case 3: // Delete User
        printf("Enter the user ID to delete: ");
        scanf("%d", &userId);

```



```

        int deleteUserIndex = -1;
        for (int i = 0; i < *userCount; i++) {
            if (users[i].id == userId) {
                deleteUserIndex = i;
                break;
            }
        }
        if (deleteUserIndex != -1) {
            // Shift users to fill the gap
            for (int i = deleteUserIndex; i < *userCount - 1; i++) {
                users[i] = users[i + 1];
            }
            (*userCount)--;
            printf("User deleted successfully.\n");
        } else {
            printf("User not found.\n");
        }
        break;

    case 4: // Return to Main Menu
        return;
    default:
        printf("Invalid choice.\n");
        break;
    }
}

int main() {
    struct User users[100]; // Assuming a maximum of 100 users
    int userCount = 0;

    // Load user data from a file (assuming a simple text file with user data)
    FILE *file = fopen("users.txt", "r");
    if (file != NULL) {
        while (fscanf(file, "%d %s %f", &users[userCount].id, users[userCount].username,
            &users[userCount].balance) != EOF) {
            userCount++;
        }
        fclose(file);
    }

    int choice;
    while (1) {
        choice = showMainMenu();

        switch (choice) {
            case 1: // User Menu

```

```

        handleUserMenu(users, &userCount);
        break;

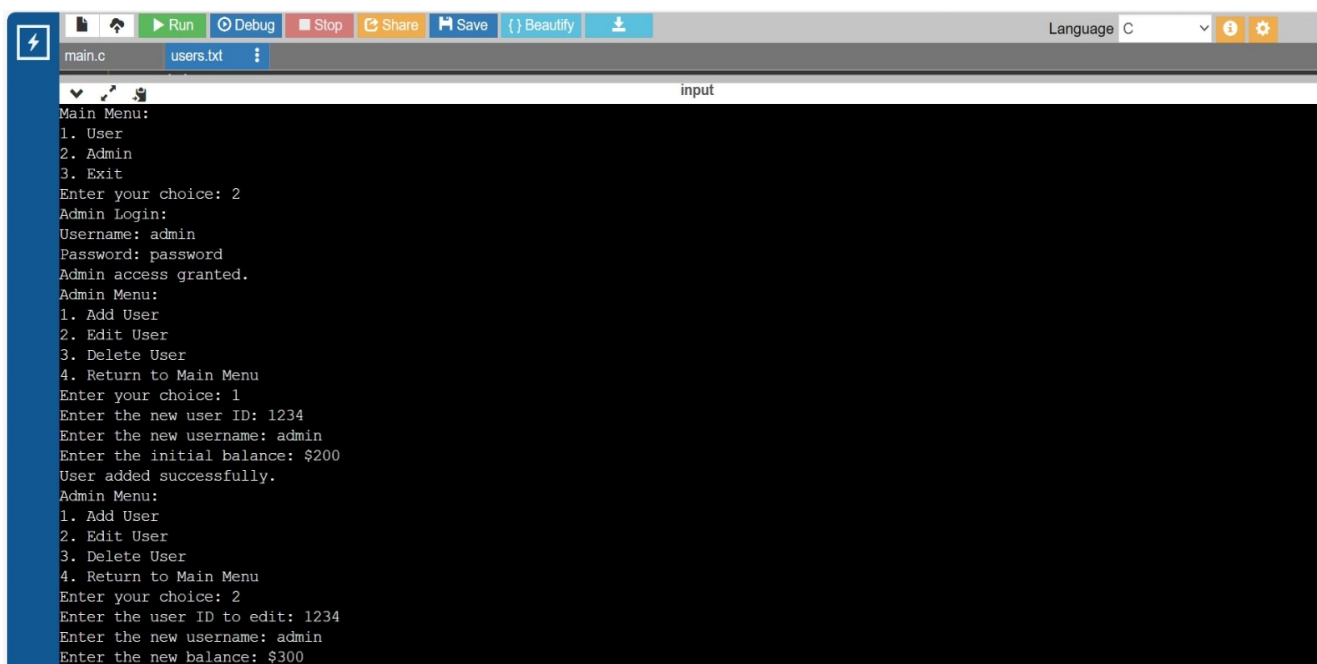
case 2: // Admin Menu
    handleAdminMenu(users, &userCount);
    break;

case 3: // Exit
    // Save user data to a file before exiting
    file = fopen("users.txt", "w");
    if (file != NULL) {
        for (int i = 0; i < userCount; i++) {
            fprintf(file, "%d %s %.2f\n", users[i].id, users[i].username, users[i].balance);
        }
        fclose(file);
    }
    exit(0);

default:
    printf("Invalid choice.\n");
    break;
    }
}
return 0;
}

```

13.1 Screenshots



```

main.c  users.txt  :
input
Main Menu:
1. User
2. Admin
3. Exit
Enter your choice: 2
Admin Login:
Username: admin
Password: password
Admin access granted.
Admin Menu:
1. Add User
2. Edit User
3. Delete User
4. Return to Main Menu
Enter your choice: 1
Enter the new user ID: 1234
Enter the new username: admin
Enter the initial balance: $200
User added successfully.
Admin Menu:
1. Add User
2. Edit User
3. Delete User
4. Return to Main Menu
Enter your choice: 2
Enter the user ID to edit: 1234
Enter the new username: admin
Enter the new balance: $300

```

```
main.c users.txt input
User updated successfully.
Admin Menu:
1. Add User
2. Edit User
3. Delete User
4. Return to Main Menu
Enter your choice: 4
Main Menu:
1. User
2. Admin
3. Exit
Enter your choice: 1
User Login:
Enter your user ID: 1234
Enter your username: admin
User access granted.
User Menu:
1. View Wallet
2. Add Money
3. Make Purchase
4. Return to Main Menu
Enter your choice: 1
User ID: 1234
Username: admin
Balance: $300.00
User Menu:
1. View Wallet
2. Add Money
```

```
main.c users.txt input
3. Make Purchase
4. Return to Main Menu
Enter your choice: 2
Enter the amount to add: $450
Added $450.00 to admin's wallet. New balance: $750.00
User Menu:
1. View Wallet
2. Add Money
3. Make Purchase
4. Return to Main Menu
Enter your choice: 3
Enter the purchase amount: $250
Made a purchase of $250.00. New balance: $500.00
User Menu:
1. View Wallet
2. Add Money
3. Make Purchase
4. Return to Main Menu
Enter your choice: 4
Main Menu:
1. User
2. Admin
3. Exit
Enter your choice: 3
...Program finished with exit code 0
Press ENTER to exit console.
```