**Command-Line Payment Application Documentation**

1. **Overview:**

This Document provide a comprehensive description of the Payment Application project.it details the project’s purpose, scope, functional requirements, System architecture, code structure, Deployment and Environment Setup.

1. **Purpose and scope**
   1. **Purpose**

The Command-Line Payment Application is a Java-based application that facilitates financial transactions through a command-line interface. It allows users to register, log in, update their profiles, send money to other users or bank accounts, make payments, refund transactions, and view their transaction history. The application uses MySQL as the backend database and JDBC for database connectivity

* 1. **Scope**
* User Management: Registration, login, profile updates.
* Wallet Management: Balance checking and adding funds from multiple sources.
* Transaction Management: Logging and processing different transaction types.
* Payment Processing: Integrating various payment methods and handling refunds.
* Database Integration: Use of MySQL for storing users, transactions, payments, and bank details.

1. **System Overview:**

The system is divided into several layers:

* Presentation Layer: Command-line interface (CLI) provided via the Main class.
* Business Logic Layer: Implemented in service classes (e.g., User Service, Transaction Service, Payment Service, Wallet Service).
* Data Access Layer: Managed by the Database Manager class for MySQL connectivity.
* Domain Model: Represented by model classes (User, Bank, Transaction, Payment).

1. **Functional Requirements**

**1.User Management:**

Registration: New users can register with a unique phone number and password.

Login/Logout: Secure login and logout functionalities.

Profile Update: Users can update their profile information (name, email, phone number).

**2.Wallet Operations:**

Balance Inquiry: Users can check their current wallet balance.

Add Money: Users can add money to their wallet via bank account, card, or UPI.

**3.Transaction Handling:**

Wallet-to-Wallet: Transfer funds between users’ wallets.

Wallet-to-Bank: Transfer funds from the wallet to a bank account.

Other Payment Methods: Handle transactions via card, UPI, netbanking.

Bank-to-Bank: Facilitate direct bank-to-bank transfers.

Refund Processing: Ability to refund transactions to either the original source or wallet.

**4.Payment Processing:**

Payment Logging: Record payment details for each transaction.

Status Updates: Update transaction statuses (PENDING, COMPLETED, REFUNDED).

1. **Non-Functional requirements**

* Performance: Ensure quick processing of transactions and minimal response times.
* Security: Secure handling of sensitive data (e.g., passwords, payment details) and database connections.
* Scalability: Architecture must support future feature expansion and increased load.
* Reliability: Ensure high availability and consistency of transaction data.
* Maintainability: Code and documentation are maintained to support future enhancements and troubleshooting.

1. **System Architecture and Design**
   1. **Component Overview**

**Database Manager**: Manages MySQL connection and disconnection.

**Model Classes**:

**User:** Stores user-related data.

**Bank:** Contains bank account details.

**Transaction:** Records transaction details.

**Payment:** Captures payment-specific information.

**Service Classes**:

**User Service:** Handles user registration, login, and profile updates.

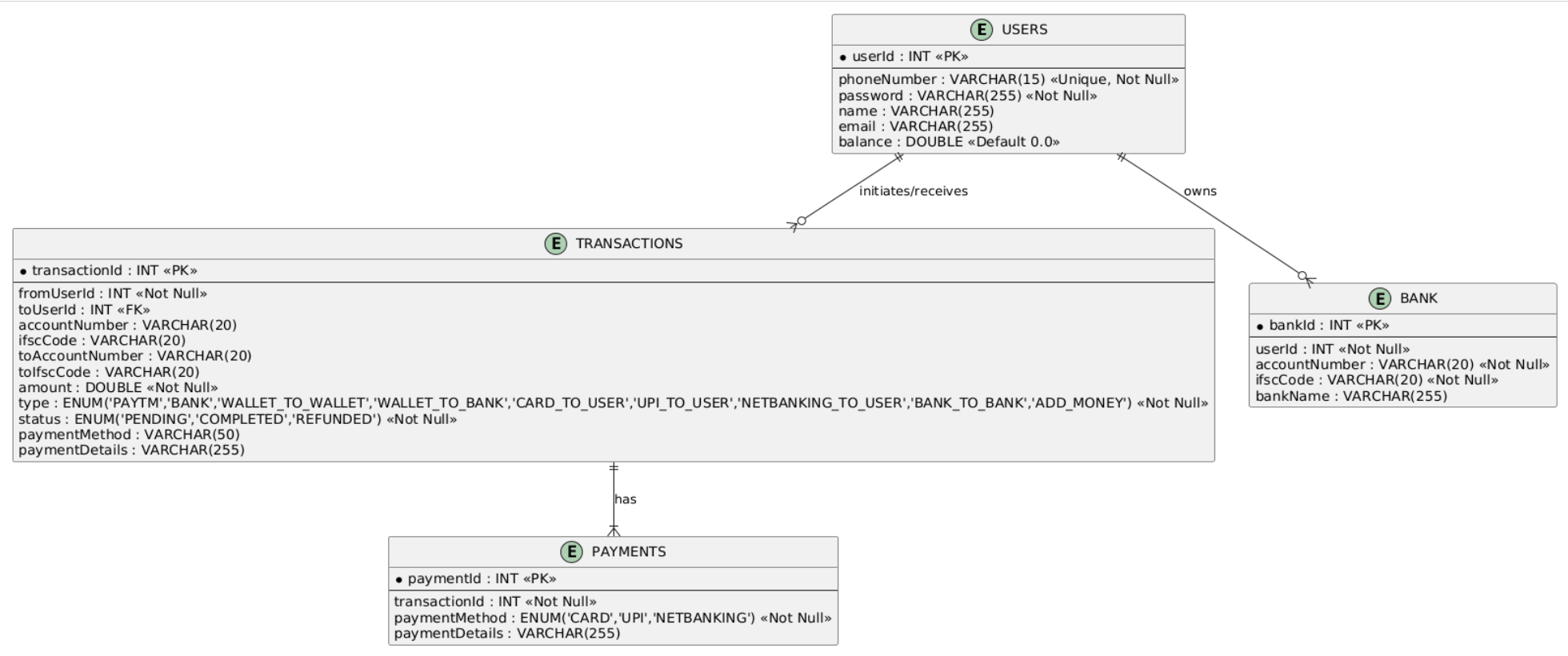
**Wallet Service:** Manages wallet balance and fund additions.

**Transaction Service:** Processes various types of transactions and refunds.

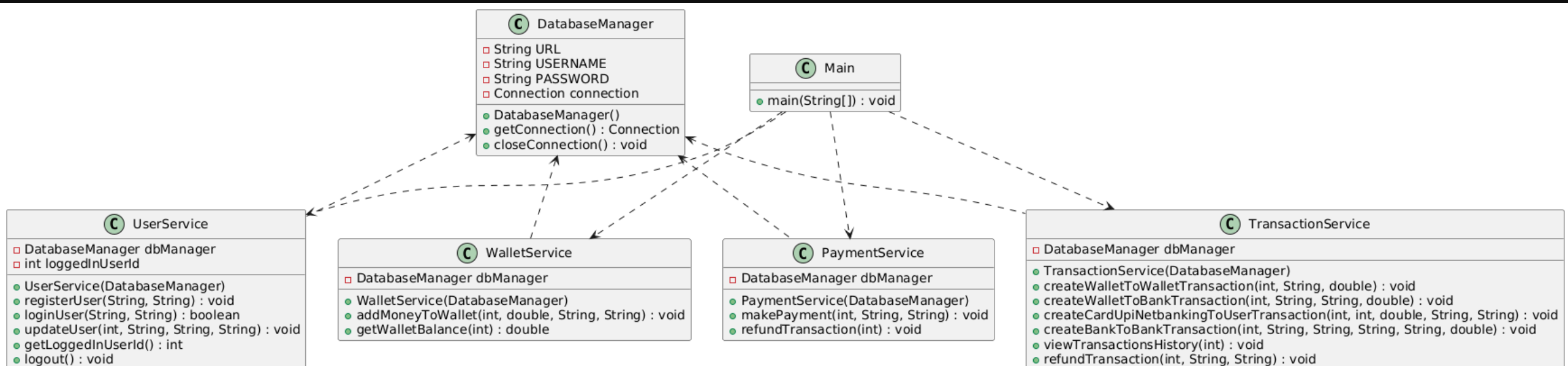
**Payment Service:** Manages direct payment operations and refunds.

**Main:** Entry point for the application, handling the CLI and user interactions.

* 1. **Class Diagram**



* 1. **Schema Diagram**



1. **Code Structure**
   1. **Package Organization**

database: Contains the DatabaseManager class.

models: Includes model classes (User, Bank, Transaction, Payment).

services: Contains business logic in service classes (UserService, WalletService, TransactionService, PaymentService).

Main Application: The Main class orchestrates user interaction via CLI.

* 1. **Class Description**

Database Manager: Handles connection lifecycle with the MySQL database.

User: Encapsulates user attributes and provides getter/setter methods.

Bank: Stores bank details for a user.

Transaction: Logs detailed transaction information and supports status updates.

Payment: Captures payment method and details related to a transaction.

User Service: Provides functionalities for user registering, logging in, and profile updates.

Wallet Service: Manages wallet balance operations and logging of fund additions.

Transaction Service: Processes various transactions (wallet-to-wallet, wallet-to-bank, etc.) and handles refunds.

PaymentService:Manages payment creation and refunds linked to transactions.

1. **Deployment and Environment Setup**
   1. **Technologies Used**

**Programming Language:** Java

**Database:** MySQL

**Database Connectivity:** JDBC (MySQL Connector/J 9.2.0)

**IDE:** VS Code

* 1. **Setup Instructions**

**Prerequisites:**

Before setting up the project, ensure the following are installed:

Java Development Kit (JDK): Version 8 or later.

MySQL Server: installed and running.

MySQL Connector/J: Download the MySQL JDBC driver (mysql-connector-j-9.2.0.jar).

* 1. **Database Setup**

**Create the database:** Open MySQL Command Line or MySQL Workbench and run the following commands:

**SQL Commands:**

CREATE DATABASE PaymentApplication;

USE PaymentApplication;

**Create Tables:**

Run the following SQL commands to create the required tables:

CREATE TABLE Users (

userId INT AUTO\_INCREMENT PRIMARY KEY,

phoneNumber VARCHAR(15) UNIQUE NOT NULL,

password VARCHAR(255) NOT NULL,

name VARCHAR(255),

email VARCHAR(255),

balance DOUBLE DEFAULT 0.0

);

CREATE TABLE Transactions (

transactionId INT AUTO\_INCREMENT PRIMARY KEY,

fromUserId INT NOT NULL,

toUserId INT,

amount DOUBLE NOT NULL,

type ENUM('WALLET\_TO\_WALLET', 'WALLET\_TO\_BANK', 'CARD\_TO\_USER', 'UPI\_TO\_USER', 'NETBANKING\_TO\_USER', 'BANK\_TO\_BANK', 'ADD\_MONEY') NOT NULL,

status ENUM('PENDING', 'COMPLETED', 'REFUNDED') NOT NULL,

paymentMethod VARCHAR(50),

paymentDetails VARCHAR(255),

refundReason VARCHAR(255),

accountNumber VARCHAR(20),

ifscCode VARCHAR(20),

toAccountNumber VARCHAR(20),

toIfscCode VARCHAR(20),

createdAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (fromUserId) REFERENCES Users(userId),

FOREIGN KEY (toUserId) REFERENCES Users(userId)

);

CREATE TABLE Payments (

paymentId INT AUTO\_INCREMENT PRIMARY KEY,

transactionId INT NOT NULL,

paymentMethod ENUM('CARD', 'UPI', 'NETBANKING') NOT NULL,

paymentDetails VARCHAR(255),

FOREIGN KEY (transactionId) REFERENCES Transactions(transactionId)

);

CREATE TABLE Bank (

bankId INT AUTO\_INCREMENT PRIMARY KEY,

userId INT NOT NULL,

accountNumber VARCHAR(20) NOT NULL,

ifscCode VARCHAR(20) NOT NULL,

bankName VARCHAR(255),

FOREIGN KEY (userId) REFERENCES Users(userId)

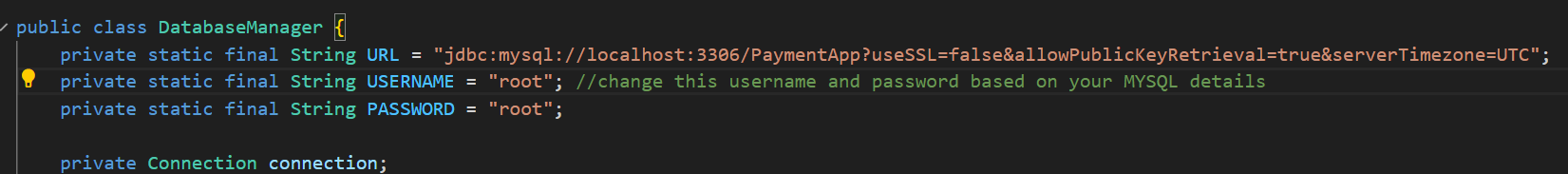
);

* 1. **Project Setup**

**Download the Project:** Clone or download the project to your local machine using the Github url(<https://github.com/kamalhemanth/projects>)

**Add MySQL Connector/J:** Place the mysql-connector-j-9.2.0.jar file in the lib folder of the project.

**Update Database Credentials:** Open the DatabaseManager.java file and update the URL, USERNAME and PASSWORD constants with your MySQL credentials as show in below Example



**Compile the Project:**

Open a terminal or command prompt in the project's root directory and run:

javac -cp lib/mysql-connector-j-9.2.0.jar -d bin src/models/User.java src/models/Transaction.java src/models/Payment.java src/database/DatabaseManager.java src/services/UserService.java src/services/TransactionService.java src/services/PaymentService.java src/services/WalletService.java src/Main.java

**Run the Application:**

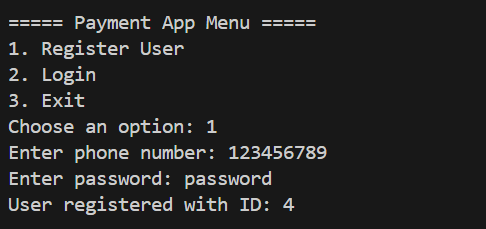
Run the following command to start the application:

java -cp bin;lib/mysql-connector-j-9.2.0.jar Main (in case the command didn't work mention the complete path for the jar file as mentioned below)

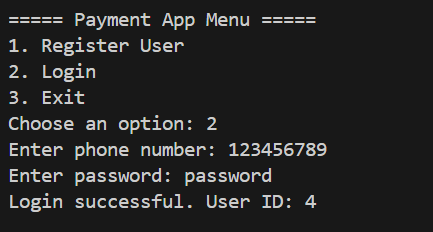
java -cp "bin;C:\Users\kamal\OneDrive\Desktop\PaymentApplication\lib\mysql-connector-j-9.2.0.jar" Main

* 1. **Usage Instructions**

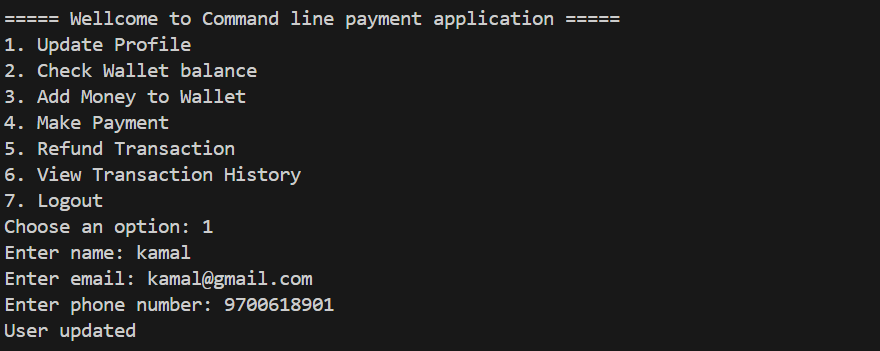
**1. Register a User:** Choose option 1 from the main menu and enter a phone number and password for user registration



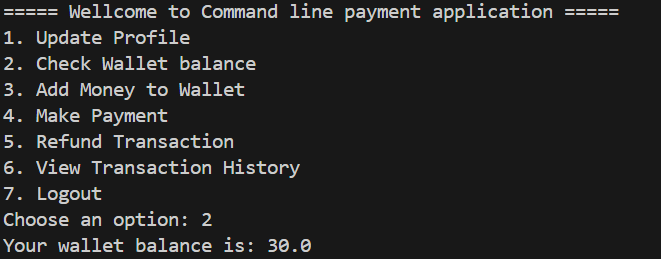
**2. Log In:** Choose option 2 from the main menu and enter the registered phone number and password.



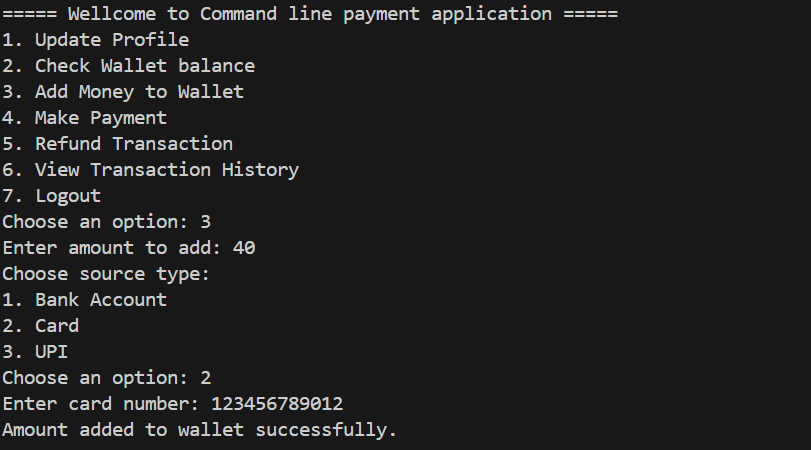
**3. Update Profile:** After logging in, choose option 1 from the payment app Login menu.



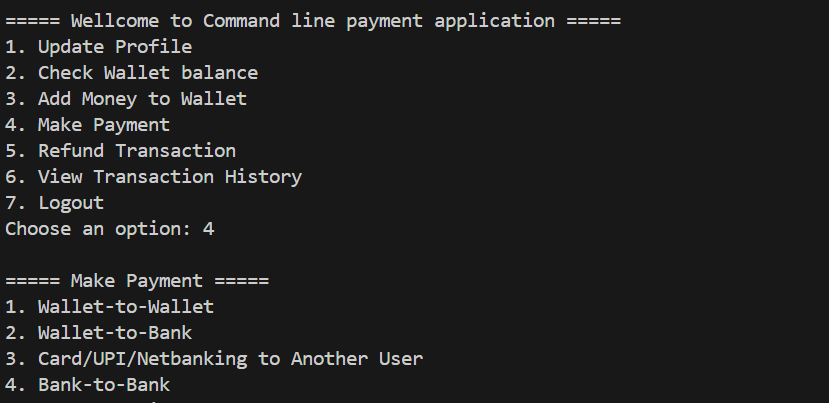
**4.Check Wallet Balance:** Choose option 2 to check your wallet balance



**5.Add Money to Wallet:** you can add money to your wallet by choosing option 3

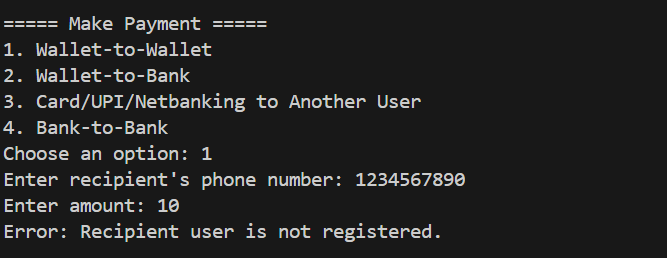


**6**. **Make a Payment:** Choose options 4 to make a payment as your wish



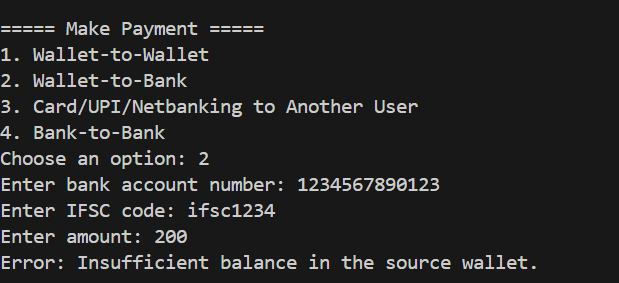
1)Send money other user wallet

* Enter the recipient's phone number and the amount if the user is registered the transfer will be successful



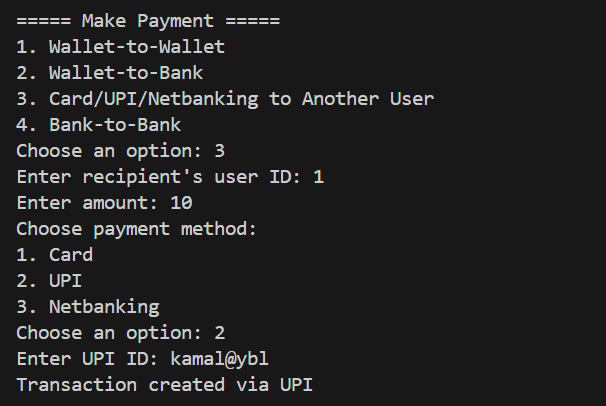
2)Send Money to a Bank Account using wallet if you have enough fund in your wallet transaction will be successful

* Enter the account number, IFSC code, and the amount.



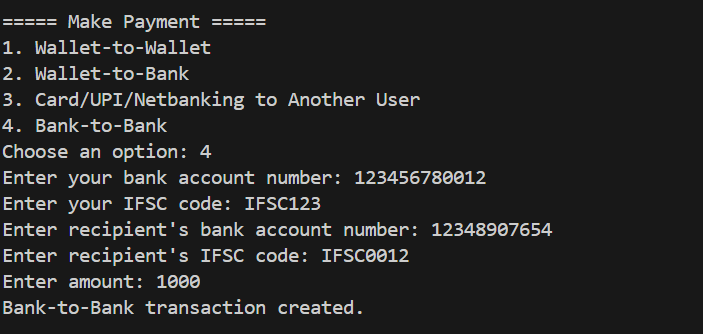
3)Make Payment via Card/UPI/Netbanking

* Enter the amount, payment method (CARD/UPI/NETBANKING), and payment details.



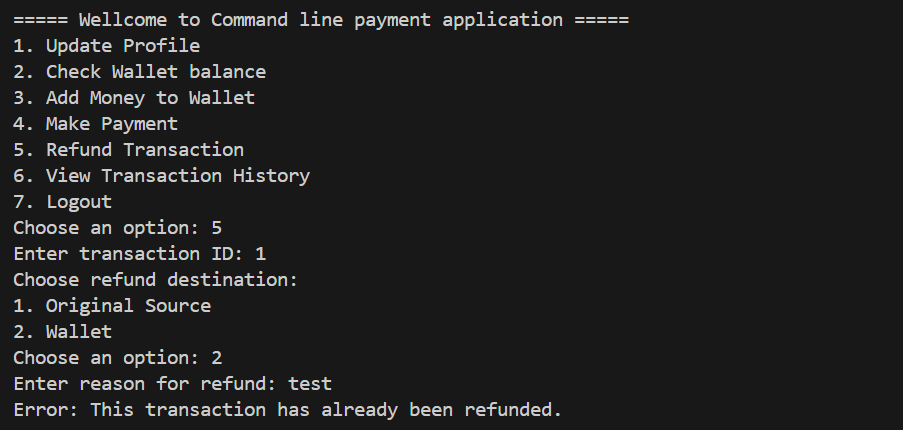
4) Bank to bank payment

* You can send money from your bank to other bank using the bank account number and IFSC code



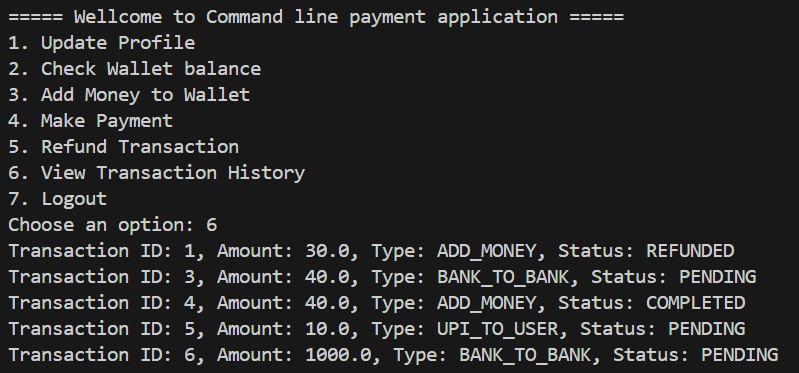
5. Refund a Transaction

Choose option 5 from the payment app Login menu and Enter the transaction ID to refund.



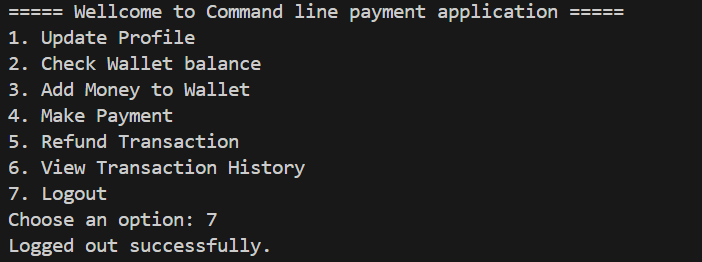
6. View Transaction History

Choose option 6 from the payment app Login menu to view all transactions for the logged-in user.



7. Logout

Choose option 7 from the payment app Login menu to log out and return to the main menu.



1. **Future Enhancements:**

**Password Encryption:** Encrypt user passwords before storing them in the database.

**Transaction Notifications:** Send notifications to users for successful transactions.

**GUI:** Develop a graphical user interface (GUI) for better user experience.

**10.Conclusion:**

The Command-Line Payment Application is a robust and scalable solution for managing financial transactions. It demonstrates the use of Java, MySQL, and JDBC to build a functional command-line application. With proper setup and usage, it can be extended to support more advanced features.