Payment Gateway Project Documentation

Overview

The Payment Gateway project is a system designed to manage payments, view sales, and handle inventory. It includes user authentication and CRUD operations for managing users. The project is built using Java and integrates with a MySQL database for data storage.

Features

User Authentication: Secure login functionality for users.

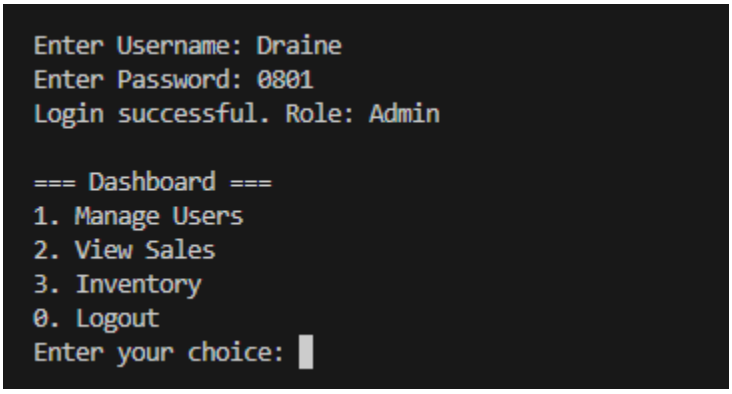
CRUD Operations for Users: Create, Read, Update, and Delete user information.

View Sales: Functionality to view sales data.

Manage Inventory: Functionality to manage inventory items.

Screenshots of CRUD Functionalities

Log In



Create user

```
=== Dashboard ===
1. Manage Users
2. View Sales
3. Inventory
0. Logout
Enter your choice: 1

=== User Operations ===
1. Create User
2. Read Users
3. Update User
4. Delete User
5. Exit
Enter your choice: 1
Enter Username: Allen
Enter Email: allen@gmail.com
Enter Role: cashier
Enter Password: 234
User created successfully.
```

Read

```
=== User Operations ===
1. Create User
2. Read Users
3. Update User
4. Delete User
5. Exit
Enter your choice: 2
User ID Username        Email   Role    Created At      Updated At
101     Draine  draine@gmail.com        Admin   2025-03-04 10:31:19.0   2025-03-09 22:21:28.0
102     micos   micos@gmail.com Cashier 2025-03-04 10:31:19.0   2025-03-09 22:21:28.0
103     admin   admin@example.com       Admin   2025-03-04 10:38:42.0   2025-03-09 22:21:28.0
104     user1   user1@example.com       Cashier 2025-03-04 10:38:42.0   2025-03-09 22:21:28.0
105     hezekiah        hezekiah@gmail.com      Cashier 2025-03-10 00:25:04.0   2025-03-10 00:25:04.0
106     Allen   allen@gmail.com Cashier 2025-03-10 22:41:04.0   2025-03-10 22:41:04.0
```

Update

```
=== User Operations ===
1. Create User
2. Read Users
3. Update User
4. Delete User
5. Exit
Enter your choice: 3
Enter User ID to Update: 106
Enter New Username: Ramis
Enter New Email: ramis@gmail.com
Enter New Role: cashier
User updated successfully.
```

Delete

```
=== User Operations ===
1. Create User
2. Read Users
3. Update User
4. Delete User
5. Exit
Enter your choice: 4
Enter User ID to Delete: 106
Are you sure you want to delete this user? (yes/no): yes
User deleted successfully.
```

CODE SNIPPET

Create

```
public static void createUser(String username, String email, String role, String password) {
    String sql = "INSERT INTO user (username, email, role, password_hash) VALUES (?, ?, ?, ?)";

    try (Connection conn = DatabaseUtil.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {

        String passwordHash = hashPassword(password);

        pstmt.setString(1, username);
        pstmt.setString(2, email);
        pstmt.setString(3, role);
        pstmt.setString(4, passwordHash);
        pstmt.executeUpdate();

        System.out.println("User created successfully.");

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Read Users

```
public static void readUsers() {
    String sql = "SELECT * FROM user";

    try (Connection conn = DatabaseUtil.getConnection();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {

        System.out.println("User ID\tUsername\tEmail\tRole\tCreated At\tUpdated At");
        while (rs.next()) {
            System.out.println(rs.getInt("user_id") + "\t" +
                               rs.getString("username") + "\t" +
                               rs.getString("email") + "\t" +
                               rs.getString("role") + "\t" +
                               rs.getTimestamp("created_at") + "\t" +
                               rs.getTimestamp("updated_at"));
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Update User

```
public static void updateUser(int id, String username, String email, String role) {
    String sql = "UPDATE user SET username = ?, email = ?, role = ?, updated_at = CURRENT_TIMESTAMP WHER

    try (Connection conn = DatabaseUtil.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(1, username);
        pstmt.setString(2, email);
        pstmt.setString(3, role);
        pstmt.setInt(4, id);
        pstmt.executeUpdate();

        System.out.println("User updated successfully.");

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Delete User

```
public static void deleteUser(int id) {
    String sql = "DELETE FROM user WHERE user_id = ?";

    try (Connection conn = DatabaseUtil.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setInt(1, id);
        pstmt.executeUpdate();

        System.out.println("User deleted successfully.");

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Challenges Encountered and Solutions Applied

Challenge 1: Handling SQL Exceptions

Problem: SQL exceptions were frequently encountered during database operations. Solution: Implemented try-with-resources statements to ensure that database resources are properly closed and added detailed exception handling to provide more informative error messages.

Challenge 2: Configuration on Running Java on Visual Studio Code

Problem: Configuring the environment to run Java applications in Visual Studio Code. Solution: Created a launch.json file to configure the Java runtime environment and ensure smooth execution of the application.

Challenge 3: Input Validation

Problem: Users could enter invalid or duplicate data, causing errors in the application. Solution: Added input validation checks to ensure that all required fields are filled and that duplicate entries are not allowed.

Prerequisites

Java Development Kit (JDK) 8 or higher

MySQL Database

JDBC Driver for MySQL

Database Schema Diagram