Source Code (JAVA)

Food.java

```java
package org.example;

import java.sql.*;
import java.util.Scanner;

public class Food {
    static Scanner scanner = new Scanner(System.in);
    static String dbUrl = "jdbc:postgresql://localhost:5433/fodo";
    static String dbUser = "postgres";
    static String dbPassword = "6789";

    public static void main(String[] args) {
        while (true) {
            System.out.println("Enter Username: ");
            String username = scanner.next();
            System.out.println("Enter Password: ");
            String password = scanner.next();

            User user = authenticate(username, password);
            if (user != null) {
                switch (user.getRole()) {
                    case "customer":
                        handleCustomer(user);
                        break;
                    case "courier":
                        handleCourier();
                        break;
                    case "admin":
                        handleAdmin();
                        break;
                }
            } else {
                System.out.println("Invalid login, please try again.");
            }
        }
    }

    public static User authenticate(String username, String password) {
        try (Connection connection = DriverManager.getConnection(dbUrl, dbUser,
dbPassword)) {
            String query = "SELECT username, password, role FROM users WHERE username = ?
AND password = ?";
            PreparedStatement stmt = connection.prepareStatement(query);
            stmt.setString(1, username);
            stmt.setString(2, password);

            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return new User(rs.getString("username"), rs.getString("password"),
rs.getString("role"));
            }
        } catch (SQLException e) {
            System.out.println("Error during authentication: " + e.getMessage());
        }
        return null;
    }

    public static void handleCustomer(User user) {
        String customerName = user.getUsername();
        try (Connection connection = DriverManager.getConnection(dbUrl, dbUser,
dbPassword)) {
            String[] orderedItems = new String[100];
            int[] quantities = new int[100];
            int totalCost = 0;
            int itemCount = 0;

            while (true) {
                System.out.println("Menu:");
                String query = "SELECT food_id, food_name, price, stock FROM fooditems";
                Statement stmt = connection.createStatement();
                ResultSet rs = stmt.executeQuery(query);

                while (rs.next()) {
                    int foodId = rs.getInt("food_id");
                    String foodName = rs.getString("food_name");
                    int price = rs.getInt("price");
                    int stock = rs.getInt("stock");

                    if (stock > 0) {
                        System.out.println(foodId + ". " + foodName + " = $" + price + "
(" + stock + " available)");
                    }
                }

                System.out.println("Enter the number of the food item you want to
order:");
                int choice = scanner.nextInt();
                System.out.println("How many?");
                int quantity = scanner.nextInt();

                String selectQuery = "SELECT price, stock FROM fooditems WHERE food_id
= ?";
                PreparedStatement selectStmt = connection.prepareStatement(selectQuery);
                selectStmt.setInt(1, choice);
                ResultSet selectedItem = selectStmt.executeQuery();

                if (selectedItem.next()) {
                    int price = selectedItem.getInt("price");
                    int stock = selectedItem.getInt("stock");

                    if (quantity > stock) {
                        System.out.println("Oh  sorry, wedon't have that much.");
                        continue;
                    }

                    String updateQuery = "UPDATE fooditems SET stock = stock - ? WHERE
food_id = ?";
                    PreparedStatement updateStmt =
connection.prepareStatement(updateQuery);
                    updateStmt.setInt(1, quantity);
                    updateStmt.setInt(2, choice);
                    updateStmt.executeUpdate();

                    orderedItems[itemCount] = "Food ID: " + choice;
                    quantities[itemCount] = quantity;
                    totalCost += quantity * price;
                    itemCount++;
                }

                System.out.println("Anything else? (Y/N)");
                String more = scanner.next();
                if (more.equalsIgnoreCase("N")) {
                    break;
                }
            }

            System.out.println("Total Cost: $" + totalCost);
            System.out.println("Enter your address:");
            scanner.nextLine(); // consume the leftover newline
            String address = scanner.nextLine();

            String insertOrderQuery = "INSERT INTO orders (user_id, total_price,
delivery_address) VALUES ((SELECT user_id FROM users WHERE username = ?), ?, ?)";
```

```java
            PreparedStatement insertOrderStmt =
connection.prepareStatement(insertOrderQuery);
            insertOrderStmt.setString(1, customerName);
            insertOrderStmt.setInt(2, totalCost);
            insertOrderStmt.setString(3, address);
            insertOrderStmt.executeUpdate();

            System.out.println("Thank you for ordering! Your food will arrive soon.");

        } catch (SQLException e) {
            System.out.println("Error while processing the order: " + e.getMessage());
        }
    }



    public static void handleCourier() {
        try (Connection connection = DriverManager.getConnection(dbUrl, dbUser,
dbPassword)) {
            String query = "SELECT order_id, delivery_address, total_price FROM orders";
            Statement stmt = connection.createStatement();
            ResultSet rs = stmt.executeQuery(query);

            System.out.println("Orders:");
            while (rs.next()) {
                System.out.println("Order ID: " + rs.getInt("order_id") + ", Address: " +
rs.getString("delivery_address") + ", Total: $" + rs.getInt("total_price"));
            }
        } catch (SQLException e) {
            System.out.println("Error while retrieving orders: " + e.getMessage());
        }
    }



    public static void handleAdmin() {
        try (Connection connection = DriverManager.getConnection(dbUrl, dbUser,
dbPassword)) {
            while (true) {
                System.out.println("Admin Menu:");
                System.out.println("1. Show information of orders placed by the
customers");
                System.out.println("2. Show and manage food stock");
                System.out.println("3. Exit to login page");

                int choice = scanner.nextInt();
                if (choice == 1) {
                    String query = "SELECT * FROM orders";
                    Statement stmt = connection.createStatement();
                    ResultSet rs = stmt.executeQuery(query);

                    System.out.println("Orders:");
                    while (rs.next()) {
                        System.out.println("Order ID: " + rs.getInt("order_id") + ", User
ID: " + rs.getInt("user_id") + ", Total Price: $" + rs.getInt("total_price"));;
                    }
                } else if (choice == 2) {
                    while (true) {
                        System.out.println("Food Management:");

                        String foodQuery = "SELECT food_id, food_name, stock FROM
fooditems";
                        Statement stmt = connection.createStatement();
                        ResultSet rs = stmt.executeQuery(foodQuery);

                        while (rs.next()) {
                            int foodId = rs.getInt("food_id");
                            String foodName = rs.getString("food_name");
                            int stock = rs.getInt("stock");
                            System.out.println(foodId + ". " + foodName + " = " + stock);
                        }

                        System.out.println("Enter the ID of the food item to increase
stock followed by the quantity (e.g., '1 5' to add 5 items):");
                        int foodId = scanner.nextInt();
                        int quantity = scanner.nextInt();

                        String updateQuery = "UPDATE fooditems SET stock = stock + ?
WHERE food_id = ?";
                        PreparedStatement updateStmt =
connection.prepareStatement(updateQuery);
                        updateStmt.setInt(1, quantity);
                        updateStmt.setInt(2, foodId);

                        int rowsUpdated = updateStmt.executeUpdate();
                        if (rowsUpdated > 0) {
                            System.out.println("Stock updated successfully!");
                        } else {
                            System.out.println("Invalid food ID.");
                        }

                        System.out.println("Enter Y to leave to Admin page");
                        String leave = scanner.next();
                        if (leave.equalsIgnoreCase("Y")) {
                            break;
                        }
                    }
                } else if (choice == 3) {
                    break;
                }
            }
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```
X

Main.java

```
package org.example;



import java.util.Scanner;

class User {
    String username;
    String password;
    String role;

    public User(String username, String password, String role) {
        this.username = username;
        this.password = password;
        this.role = role;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    public String getRole() {
        return role;
    }
}
```

```
    class Order {
    String customerName;
    String[] foodItems;
    int[] quantities;
    int totalCost;
    String address;

    public Order(String customerName, String[] foodItems, int[] quantities, int
totalCost, String address) {
        this.customerName = customerName;
        this.foodItems = foodItems;
        this.quantities = quantities;
        this.totalCost = totalCost;
        this.address = address;
    }

    public String toString() {
        StringBuilder orderDetails = new StringBuilder();
        orderDetails.append("Customer Name: ").append(customerName).append("\n");
        for (int i = 0; i < foodItems.length; i++) {
            orderDetails.append(foodItems[i]).append(":
").append(quantities[i]).append("\n");
        }
        orderDetails.append("Total Cost: ").append(totalCost).append("\n");
        orderDetails.append("Address: ").append(address).append("\n");
        return orderDetails.toString();
    }
}
```

Source Code Explanation

1. Food Class

This is the main class containing the program's logic for login and role-based functionality.

a) Class Variables

- `scanner`: A `Scanner` object used to read user input.
- **Database Connection Information**:
  - **`dbUrl`**: Database URL (`localhost` with port 5433, database named `fodo`).
  - **`dbUser`**: Database username (`postgres`).
  - **`dbPassword`**: Database password (6789).

b) Main Method

This is the entry point for the program:

1. **Infinite Loop for Login**:
   a. Prompts the user to enter a username and password.
   b. Calls `authenticate()` to validate credentials.
2. **Role-Based Redirection**:
   a. Depending on the returned user's role:
      i. **`customer`**: Calls `handleCustomer()` for ordering food.
      ii. **`courier`**: Calls `handleCourier()` for viewing and handling orders.
      iii. **`admin`**: Calls `handleAdmin()` for managing the system.

c) `authenticate()` Method

Handles user authentication:

1. Connects to the PostgreSQL database.
2. Executes a query to validate credentials:

```
sql
Copy code
SELECT username, password, role FROM users WHERE username = ? AND
password = ?
```

   a. Uses `PreparedStatement` to prevent SQL injection.
3. If credentials are valid, returns a `User` object.
4. If credentials are invalid or an error occurs, returns `null`.

d) `handleCustomer()` Method

Handles the **Customer Role**:

1. Connects to the database to retrieve available food items (`fooditems` table).
2. **Menu Display**:
    a. Fetches all items with stock > 0 and displays their details (ID, name, price, and stock).
3. **Order Processing**:
    a. Allows the customer to select items by ID and specify the quantity.
    b. Verifies stock availability for the selected quantity.
    c. Updates the stock in the database:

```
UPDATE fooditems SET stock = stock - ? WHERE food_id = ?
```

    d. Tracks ordered items, quantities, and calculates the total cost.
4. **Finalize Order**:
    a. Asks for the delivery address.
    b. Inserts the order into the `orders` table:

```
INSERT INTO orders (user_id, total_price, delivery_address) VALUES
((SELECT user_id FROM users WHERE username = ?), ?, ?)
```

5. **Summary**:
    a. Displays the total cost and confirms the order.

e) `handleCourier()` Method

Handles the **Courier Role**:

1. Connects to the database and retrieves all orders (`orders` table).
2. Displays order details (ID, delivery address, and total price).

f) `handleAdmin()` Method

Handles the **Admin Role**:

1. Provides three options:
    a. **View Orders**:
        i. Displays all orders from the `orders` table.
    b. **Manage Food Stock**:
        i. Retrieves current stock of food items from the `fooditems` table.
        ii. Allows the admin to increase the stock of specific items by ID.
        iii. Updates the stock in the database:

```
UPDATE fooditems SET stock = stock + ? WHERE food_id = ?
```

    c. **Exit**: Returns to the login screen.
2. Loops until the admin chooses to exit.

2. User Class

Represents a user in the system. Contains:

- **Attributes**:
    - `username`, `password`, and `role` (e.g., customer, courier, admin).
- **Constructor**:
    - Initializes the user object with these attributes.
- **Getter Methods**:
    - `getUsername()`, `getPassword()`, and `getRole()`.

3. Nested Order Class

Represents a customer order. Contains:

- **Attributes**:
    - `customerName`: The name of the customer who placed the order.
    - `foodItems[]`: Array of ordered food item names.
    - `quantities[]`: Array of corresponding quantities.
    - `totalCost`: Total cost of the order.
    - `address`: Delivery address.

- **Constructor**:
    - Initializes the order object.
- **`toString()` Method**:
    - Formats the order details into a readable string.

4. Database Tables (Assumed)

- **`users`**:
    - `user_id`, `username`, `password`, `role`.
- **`fooditems`**:
    - `food_id`, `food_name`, `price`, `stock`.
- **`orders`**:
    - `order_id`, `user_id`, `total_price`, `delivery_address`.

How the Code Works Together

1. **Login**:
    a. User enters credentials.
    b. `authenticate()` verifies them and returns a `User` object.
2. **Role-Specific Behavior**:
    a. **Customer**: Orders food, manages the cart, and places an order.
    b. **Courier**: Views all orders for delivery.
    c. **Admin**: Manages food stock and reviews customer orders.
3. **Database Interaction**:
    a. Uses `JDBC` for communication with the PostgreSQL database.
    b. Proper use of `PreparedStatement` ensures safe and efficient queries.

Potential Improvements

1. **Password Hashing**:
    a. Store and verify hashed passwords instead of plaintext.
2. **Error Handling**:
    a. Enhance error messages and use more granular exception handling.
3. **Input Validation**:
    a. Validate user inputs to avoid runtime errors.

4. **Encapsulation**:
    a. Move SQL queries and database logic to a separate class (e.g., `DatabaseHelper`).
5. **Use Object Collections**:
    a. Replace arrays with `ArrayList` or `HashMap` for flexibility.
6. **Exit Option**:
    a. Add a way for users to exit the application cleanly.