



**UNIVERSITY OF
PLYMOUTH**

NSBM Green University

Faculty of Computing

PUSL2024 Software Engineering 02

Java Group Project

Green Supermarket Web-Report

	Group Number	B2
	Student ID (Plymouth)	Name
1	10899343	Priyantha Ranasinghe
2	10899228	Rankira Kosgollage
3	10899233	Dodampe Nimna
4	10903090	Ranasinghe
5	10899368	Andara Siddhanth
6	10899313	Agalakotuwe Jayawardhana

Table of Contents

Introduction to Scenario	3
Processors of Green Supermarket	3
Use case Diagram	4
Class Diagram	4
Sequence Diagram	5
ER Diagram	5
Samples of the developed system with an explanation of the functionality	6
Front-end	6
Back-end	14
PayPal Sandbox Configurations	22
SMS or Email notification of purchase and cancellation options	24
GitHub Project Link	27
References	27
Contribution	27

Introduction to Scenario

Green Supermarket is one of the leading supermarkets in the business industry hence it has many loyal customers as it offers the best products and services but one of the stumbling blocks for this company is its website which is known for being old-fashioned and having a weak foundation.

The green supermarket evolved at a fast pace after the origin of their business, the organization's incapability to offer an online shopping facility has brought many regrets to them and their customers, due to that reason, the supermarket decided to restore its website by adding new features such as online shopping, online payment handling, ceasing shopping collisions, visualization of the customer feedback and email notification of purchase and cancellation options.

By making these changes, Green Supermarket hopes for an increase in their sales from their website because it will be easier and more convenient. This would help them sell more and make their customers happier by giving them better ways to shop.

Processors of Green Supermarket

Online Shopping: This feature allows customers to browse through the products available at Green Supermarket and make purchases directly through the company's website. Customers can view product details, add items to their virtual shopping cart, and complete the purchase process entirely online without physically visiting the store.

Online Payment Handling: Once customers have selected the items, they want to purchase through the online shopping feature, online payment handling enables them to securely pay for their purchases over the internet.

Avoid Shopping Collisions: Ensuring that when multiple customers are browsing the online store simultaneously, they do not encounter issues like products becoming unavailable (e.g., sold out) after being added to their shopping cart. This processor aims to manage the inventory in real-time, ensuring that items are available for purchase until the transaction is completed.

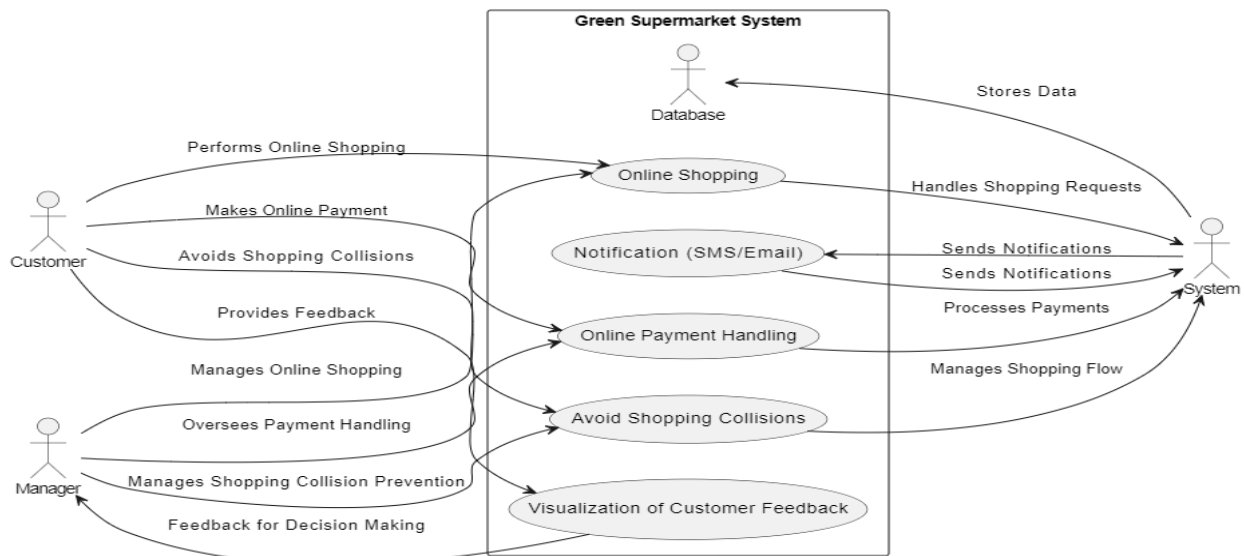
Visualization of Customer Feedback for Managerial Decision-Making: This involves collecting and presenting customer feedback in an easily understandable format for the supermarket's managers or decision-makers. By using graphs, charts, or reports, the supermarket can analyze and understand customer opinions, preferences, complaints, and suggestions. This information can be used to make informed decisions about improving services, products, or the overall customer experience.

Email Notification of Purchase and Cancellation Options:

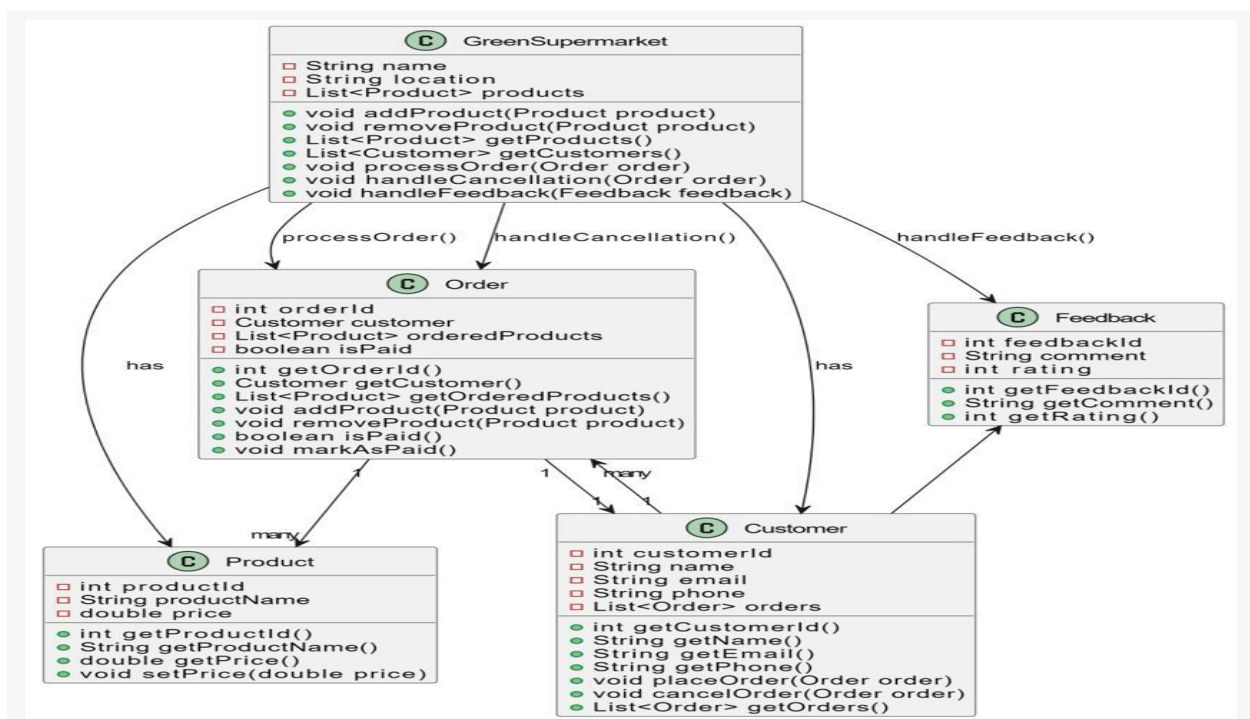
Purchase Notification - When a customer completes a purchase online, they receive an email confirming their order. It acts as transaction confirmation and gives the customer the necessary details about their purchase.

Cancellation Notification - When a customer cancels an order or if there are changes to their order status, then the system send an email to notify the customer which contains details about the order cancellation.

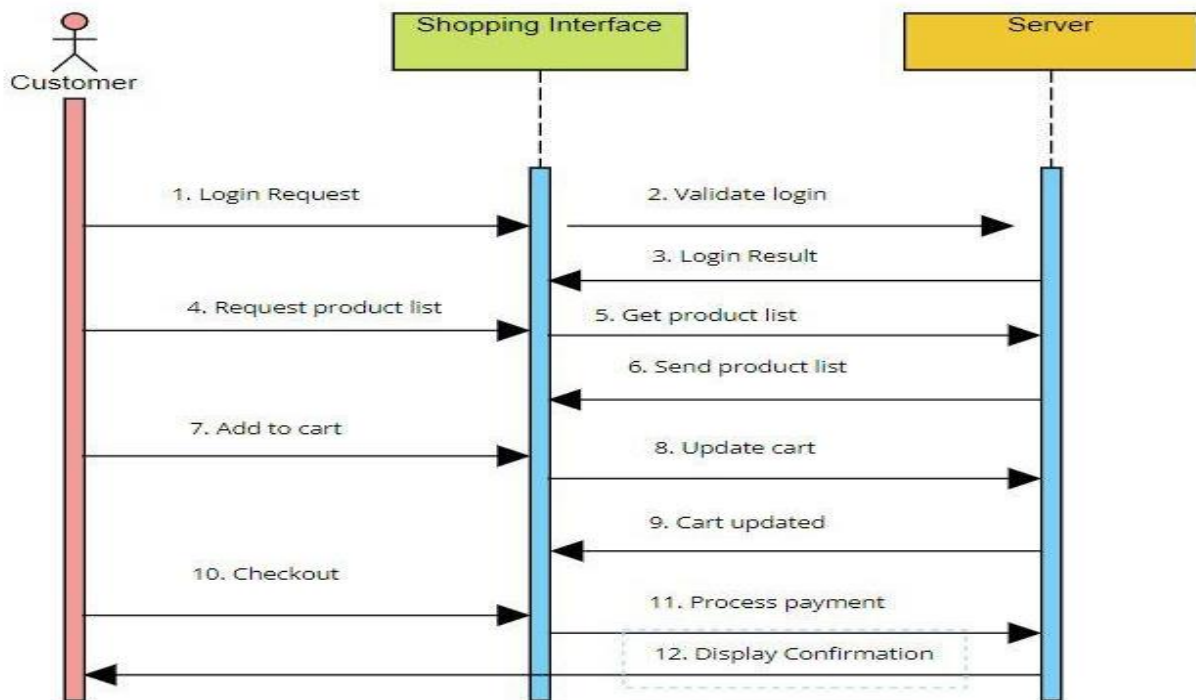
Use case Diagram



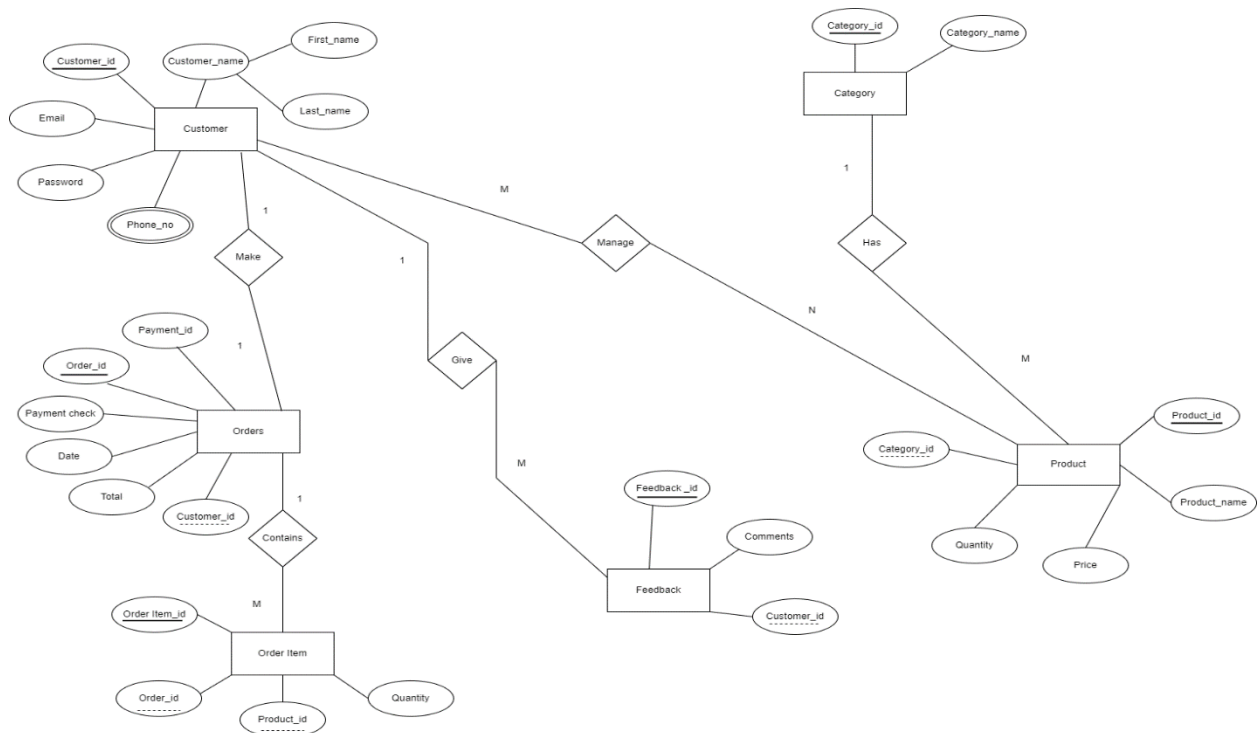
Class Diagram



Sequence Diagram



ER Diagram



Samples of the developed system with an explanation of the functionality

Front-end

Index



Cach Controlling: Using meta tags to control caching.

Image: Image tag is used to display the company logo.

Redirecting: After simulating image loading delay and redirecting to the welcome page.

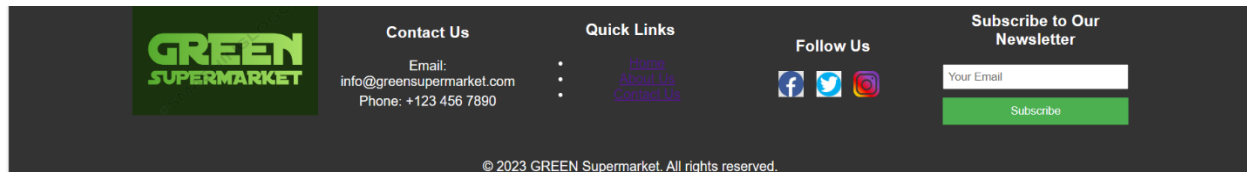
Navbar



Tabs: The condition checks if the session attribute "customer_id" is null. If it is, it assumes the customer is not logged in and displays the registration, login, and cart links. It assumes the user is logged in and displays the profile and cart links.

Company Logo: Logo is displayed by an img tag.

Footer

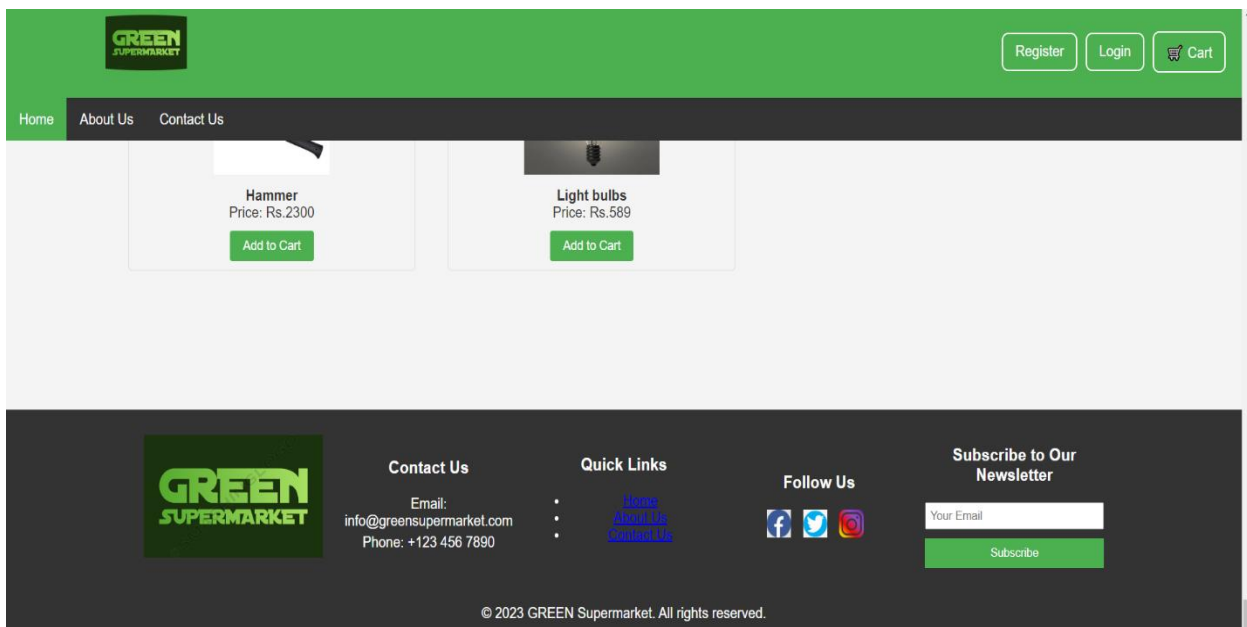


Contact us Section: In the Contact us section Email and Phone information are displayed.

Quick Link Section: In the quick links section when the customer clicks on either one of the three links displayed there are redirected to that specific page.

Follow us Section: In the follow us section the social media pages are displayed. When subscribed to the newsletter the customers get new updates from the website.

Welcome

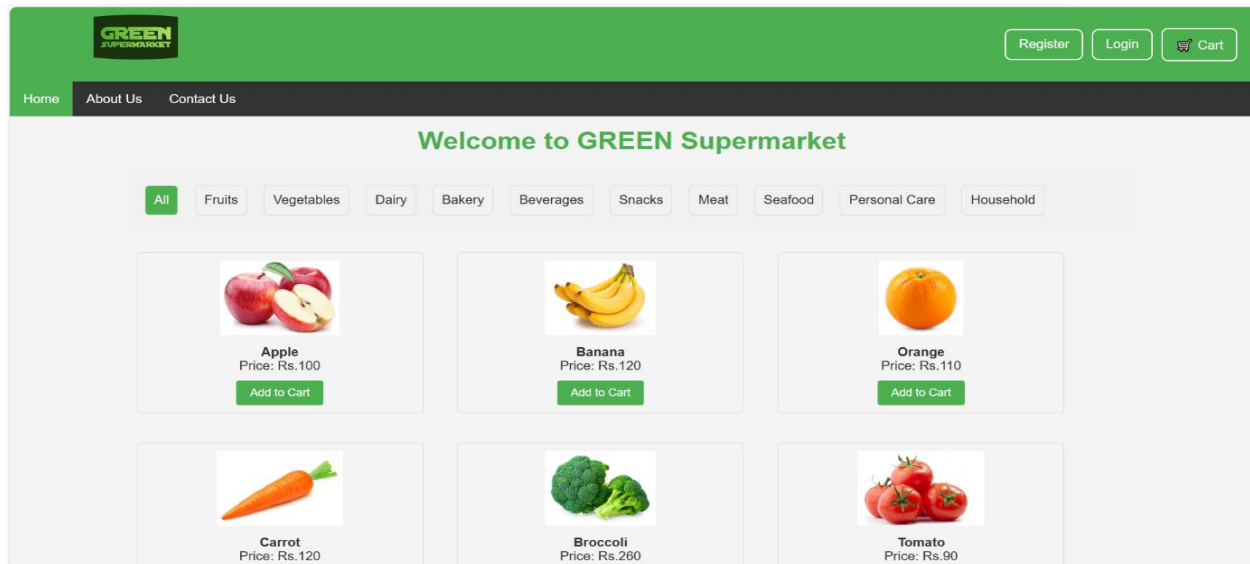


Page linking: Inside the navbar Home, About Us and Contact Us pages are inserted as links so when the customer clicks on these pages, they are redirected to that specific page in the website.

Ajax: If the readyState property of the XMLHttpRequest object is 4 and the status property is 200 then set the innerHTML of the contentDiv to the response text received from the server.

Active Tabs: Finds the currently active tab in the navigation (active Tab) and removes the active class from it if it exists. Adds the active class to the tab that was clicked (clicked Tab).

Home



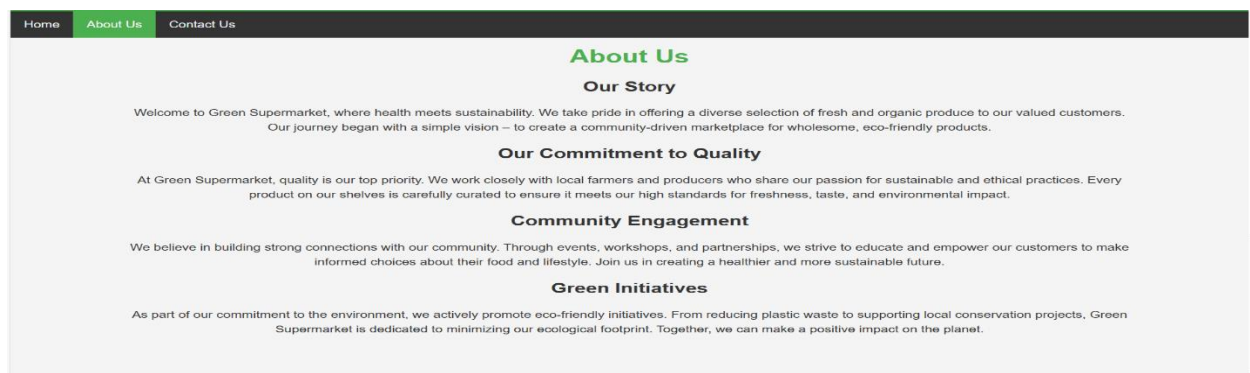
Dynamic Content: The page dynamically retrieves and displays product categories from the database. It connects to the database, executes a query to retrieve category data, and generates HTML to display these categories.

JavaScript Functions: JavaScript functions are used to handle user interactions on the client side. `showAllProducts()` and `showProductsByCategory(category)` are responsible for showing products based on the selected category. They also update the visual style of the selected category.

AJAX Requests: The `showProducts(categoryId)` function uses AJAX to asynchronously fetch product data from the server. The retrieved data is then used to dynamically generate and display product information on the page without a full page reload.

Adding to Cart: The `addToCart(productId)` function checks if a customer is logged in. If not, it redirects to the login page. If logged in, it makes an AJAX call to the server to add the selected product to the customer's shopping cart.

About Us



This contain contains all the details of the supermarket.

Contact Us

GREEN SUPERMARKET

Register Login Cart

Home About Us **Contact Us**

Contact Us

Welcome to our Contact Us page. You can reach us through the following methods:

Email: contact@example.com
Phone: [+1 123-456-7890](tel:+11234567890)
Address: [1234 Elm Street, Springfield, SL](#)
Social Media:
[Facebook](#)
[Twitter](#)
[Instagram](#)
Online Chat: Use our live chat option on the website

Feedback

Email:

Your Feedback:

Content: The page displays a heading "Contact Us" and provides information on how users can reach out, including email, phone, address, social media links, and an online chat option.

Feedback Form: A form is provided for users to submit feedback. It uses the POST method to submit data to the "FeedbackServlet." The form includes input fields for the customer's email and a textarea for entering feedback. The form has a "Submit Feedback" button.

Registration

First Name:

Last Name:

Email:

Phone Number:

Address:

Postal Code:

Password:

Confirm Password:

Register

Modal JavaScript Functions: The display Modal(message) function is used to display a modal with a specified message. The hideModal () function is used to hide the modal.

Form Validation JavaScript Function: The validateForm () function checks whether the entered passwords match. If not, it displays a modal with an error message.

Form: The form uses the POST method and is submitted to the "RegistrationServlet."

It includes input fields for the user's first name, last name, email, phone number, address, postal code, password, and password confirmation.

A custom alert is displayed if there is an error message from the server (e.g., duplicate email during registration).

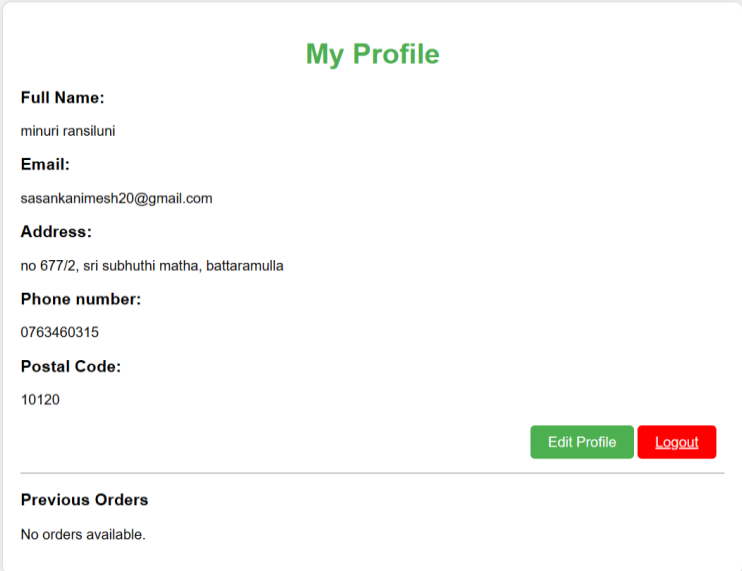
The form includes JavaScript validation using the onsubmit attribute to ensure that the passwords match.

Dynamic Content: The page checks for an error message attribute in the request and displays it in a custom alert if present.

Modal and Overlay: The modal and overlay are used for displaying messages to the user.

Styling: The CSS file (forms.css) is linked to style the appearance of the registration form. Specific styling details are not provided in the code snippet.

Customer Profile



The screenshot displays a web form titled "My Profile" in green text. The form contains several labeled input fields with their respective values: "Full Name:" with "minuri ransiluni", "Email:" with "sasankanimesh20@gmail.com", "Address:" with "no 677/2, sri subhuthi matha, battaramulla", "Phone number:" with "0763460315", and "Postal Code:" with "10120". At the bottom right of the form are two buttons: a green "Edit Profile" button and a red "Logout" button. Below the form, there is a section titled "Previous Orders" which states "No orders available."

Customer Information Section: Displays customer information such as full name, email, address, phone number, and postal code.

Allows the user to edit their information by toggling between text and input fields.

Provides buttons to edit and save the user information.

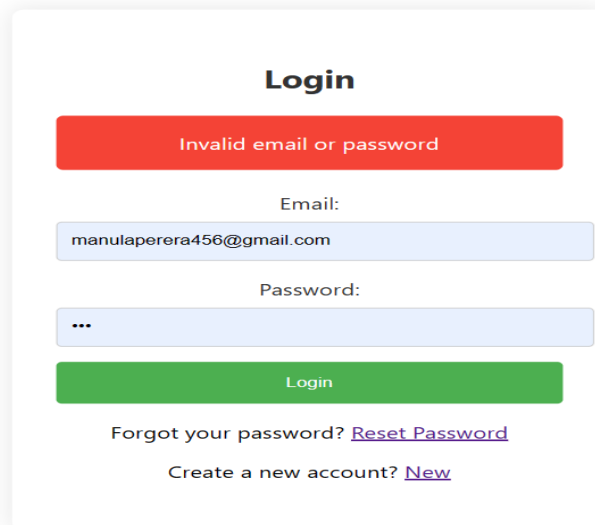
Previous Orders Section: Displays a list of previous orders, including product details and order quantities.

Offers a "Cancel Order" button to cancel an order (via an AJAX call to OrderCancellationServlet).

JavaScript: editInfo () function toggles between displaying text and input fields for user information.

cancelOrder () function prompts the user to confirm the cancellation and initiates an AJAX call to OrderCancellationServlet.

Login



Login

Invalid email or password

Email:

manulaperera456@gmail.com

Password:

...

Login

Forgot your password? [Reset Password](#)

Create a new account? [New](#)

Basic Setup: We've set the page language to Java, declared the character encoding, and structured our HTML document.

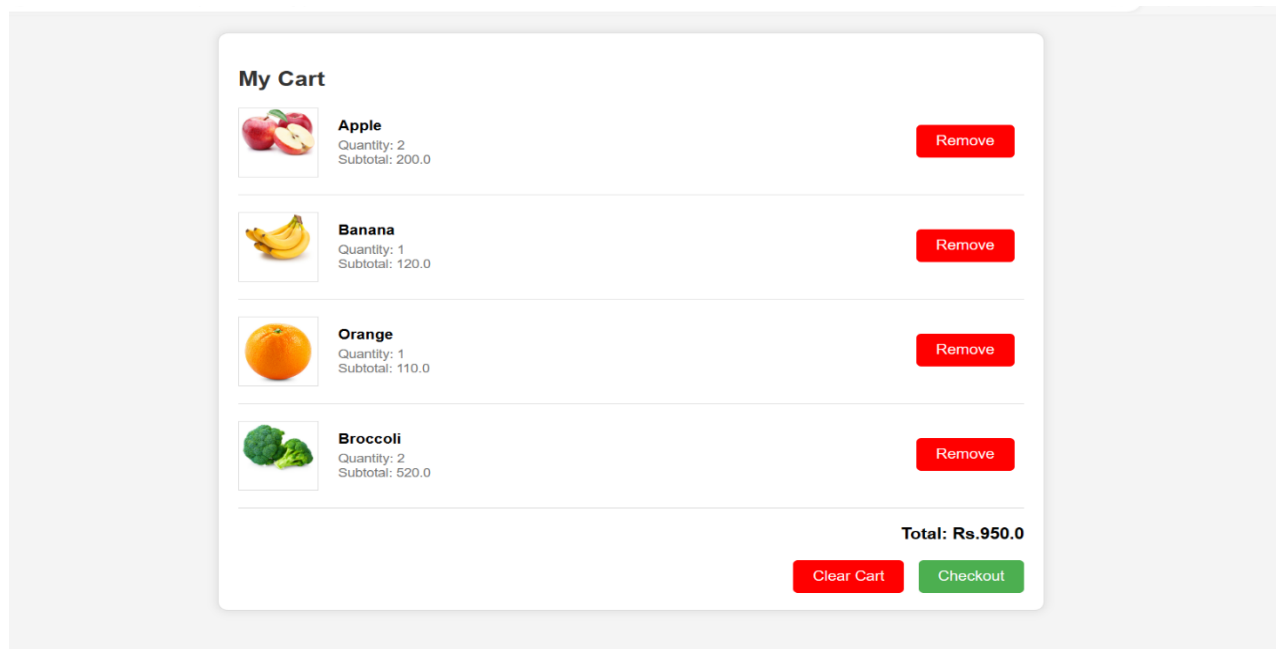
Main Container: Created a container with a heading "Login" to hold our login form.

Error Message Handling: Dynamically checks if there's an error message from the server. If yes, it displays a styled alert div to inform users about any issues during login.

Login Form: Designed a simple form with fields for email and password. It's configured to send a POST request to our "LoginServlet" upon submission.

Forgot Password and Registration Links: Provided links for users who might have forgotten their password or want to create a new account. These links point to related pages.

Cart



Dynamic Content: The page dynamically displays the customer's cart items, including product details (name, quantity, subtotal, image).

The total amount of the order is displayed.

JavaScript Functions: The `redirectToCheckout ()` function navigates to the Checkout page.

The `removeItem(productId)` function removes a specific item from the cart by making an AJAX call to the "RemoveItemServlet" and reloading the page upon success.

The `clearCart ()` function clears the entire cart by making an AJAX call to the "ClearCartServlet" and reloading the page upon success.

Checkout

The screenshot shows a 'Checkout' form with a white background and rounded corners. It is divided into two main sections: 'Order Details' and 'Shipping Details'. The 'Order Details' section shows a banana image, the name 'Banana', quantity '2', and subtotal '240.0'. Below this is a line for 'Total Amount: 240.0'. The 'Shipping Details' section contains three input fields: 'Full Name:', 'Last Name:', and 'Address:'. At the bottom, there is a label for 'Postal Code:' followed by an input field.

Checkout

Order Details

Banana
Quantity: 2
Subtotal: 240.0

Total Amount: 240.0

Shipping Details

Full Name:

Last Name:

Address:

Postal Code:

Address:

Postal Code:

Phone:

07X-XXX-XXXX

Payment Method

Cash on Delivery ☐

PayPal ☒

PayPal

Debit or Credit Card

Powered by **PayPal**

Confirm Order

Order Details Section: Displays a list of ordered items with product details (name, quantity, subtotal).

Shows the total amount of the order.

Shipping Details Section: Form for users to enter shipping details, including full name, last name, address, postal code, and phone number.

Payment Method Section: Allows customers to choose between Cash on Delivery and PayPal.

Includes a PayPal button, which is initially hidden for Cash on Delivery but displayed when PayPal is selected.

JavaScript: Controls the visibility of the PayPal button based on the selected payment method.

Integrates the PayPal JavaScript SDK for creating and capturing orders.

PayPal Integration: Uses the PayPal JavaScript SDK to create and capture orders.

Updates the display upon successful payment.

Back-end

Java Database Connectivity (JDBC)

```
10 public class DatabaseUtil {
11
12     private static final String JDBC_URL = "jdbc:mysql://localhost:3306/greensupermarket";
13     private static final String JDBC_USER = "root";
14     private static final String JDBC_PASSWORD = "*****";
15
16     static {
17         try {
18             Class.forName("com.mysql.cj.jdbc.Driver");
19         } catch (ClassNotFoundException e) {
20             e.printStackTrace();
21         }
22     }
23
24     public static Connection getConnection() throws SQLException {
25         return DriverManager.getConnection(JDBC_URL, JDBC_USER, JDBC_PASSWORD);
26     }
27
28     public static void close(ResultSet resultSet, Statement statement, Connection connection) {
29         if (resultSet != null) {
30             try {
31                 resultSet.close();
32             } catch (SQLException e) {
33                 e.printStackTrace();
34             }
35         }
36         if (statement != null) {
37             try {
38                 statement.close();
39             } catch (SQLException e) {
40                 e.printStackTrace();
41             }
42         }
43         if (connection != null) {
44             try {
45                 connection.close();
46             } catch (SQLException e) {
47                 e.printStackTrace();
48             }
49         }
50     }
51 }
```

```
47
48
49     <dependency>
50         <groupId>com.mysql</groupId>
51         <artifactId>mysql-connector-j</artifactId>
52         <version>8.2.0</version>
53     </dependency>
```

Integrated JDBC into Java application to connect with a MySQL database.

Database Setup: Created a MySQL database named greensupermarket in MySQL Workbench.

JDBC Components: Configured the JDBC URL, loaded the MySQL JDBC driver, and stored database credentials (username: *****, password: *****).

Connection Establishment: Implemented the getConnection method to establish a connection to the database, returning a Connection object.

Maven Dependency: Managed the MySQL JDBC driver dependency (MySQL-connector-java) using Maven for seamless integration. (pom.xml)

Class Loading: Loaded the MySQL JDBC driver class using Class.forName to enable communication with the MySQL database.

ProductFetchServlet

```
@WebServlet("/ProductsFetchServlet")
public class ProductsFetchServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("application/json");
        PrintWriter out = response.getWriter();

        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;

        try {
            connection = DatabaseUtil.getConnection();
            statement = connection.createStatement();

            int categoryIdParam = Integer.parseInt(request.getParameter("categoryId"));
            String query = (categoryIdParam == 0) ? "SELECT * FROM products" : "SELECT * FROM products WHERE category_id = " + categoryIdParam;
            resultSet = statement.executeQuery(query);

            List<Product> products = new ArrayList<>();
            while (resultSet.next()) {
                int productId = resultSet.getInt("product_id");
                String productName = resultSet.getString("product_name");
                double price = resultSet.getDouble("price");
                String imageUrl = resultSet.getString("image_url");
                int categoryId = resultSet.getInt("category_id");

                Product product = new Product(productId, productName, price, imageUrl, categoryId);
                products.add(product);
            }

            Gson gson = new Gson();
            String jsonProducts = gson.toJson(products);
            out.println(jsonProducts);
        }
    }
}
```

doGet () Method: This method overrides the doGet () method of HttpServlet, which gets executed when the servlet receives a GET request.

Executing Database Query: It creates a SQL query to fetch products based on the provided category ID and If the category ID is 0, it retrieves all products or else it fetches products for the specific category.

Processing Query Results: It iterates through the result set obtained from the database query and creates Product objects based on the retrieved data.

JSON Conversion & Sending Response: Converts retrieved data to JSON format using GSON and sends it as the response to the client.

AddtoCartServlet

```
}

private int getOpenOrderForCustomer(Connection connection, String customerId) throws SQLException {
    int orderId = -1;

    try (PreparedStatement stmt = connection.prepareStatement(
        "SELECT order_id FROM orders WHERE customer_id = ? AND status = 'PENDING'")) {
        stmt.setString(1, customerId);
        try (ResultSet resultSet = stmt.executeQuery()) {
            if (resultSet.next()) {
                orderId = resultSet.getInt("order_id");
            }
        }
    }

    return orderId;
}

private int createNewOrder(Connection connection, String customerId) throws SQLException {
    int orderId = -1;

    try (PreparedStatement stmt = connection.prepareStatement(
        "INSERT INTO orders (customer_id, total, status) VALUES (?, 0, 'PENDING')",
        PreparedStatement.RETURN_GENERATED_KEYS)) {
        stmt.setString(1, customerId);
        int affectedRows = stmt.executeUpdate();

        // Check if the order was created successfully
        if (affectedRows > 0) {
            ResultSet generatedKeys = stmt.getGeneratedKeys();
            if (generatedKeys.next()) {
                orderId = generatedKeys.getInt(1);
            }
        }
    }
}

private void insertOrderItem(Connection connection, int orderId, int productId, double subtotal)
    throws SQLException {
    try (PreparedStatement stmt = connection.prepareStatement(
        "INSERT INTO order_items (order_id, product_id, quantity, subtotal) VALUES (?, ?, 1, ?)")) {
        stmt.setInt(1, orderId);
        stmt.setInt(2, productId);
        stmt.setDouble(3, subtotal);
        stmt.executeUpdate();
    }
}

private void updateOrderTotal(Connection connection, int orderId) throws SQLException {
    try (PreparedStatement stmt = connection.prepareStatement(
        "UPDATE orders SET total = (SELECT SUM(subtotal) FROM order_items WHERE order_id = ?) WHERE order_id = ?")) {
        stmt.setInt(1, orderId);
        stmt.setInt(2, orderId);
        stmt.executeUpdate();
    }
}

private List<OrderItem> getOrderedItems(Connection connection, int orderId) throws SQLException {
    List<OrderItem> orderedItems = new ArrayList<>();

    try (PreparedStatement stmt = connection.prepareStatement(
        "SELECT oi.quantity, oi.subtotal, p.product_name, p.image_url, p.product_id FROM order_items oi " +
        "JOIN products p ON oi.product_id = p.product_id " +
        "WHERE oi.order_id = ?")) {
        stmt.setInt(1, orderId);
        try (ResultSet resultSet = stmt.executeQuery()) {
            while (resultSet.next()) {
                OrderItem item = new OrderItem();
                item.setQuantity(resultSet.getInt("quantity"));
                item.setSubtotal(resultSet.getDouble("subtotal"));
                item.setProductName(resultSet.getString("product_name"));
                item.setImageUrl(resultSet.getString("image_url"));
                item.setProductId(resultSet.getInt("product_id"));
            }
        }
    }
}
```

- Establish database connection.
- Check if an open order exists for the customer and if no open order exists, create a new order, Insert or update the selected product into the order_items table.
- If the product already exists, update the quantity and subtotal or if the product doesn't exist, insert a new order item.
- The orders table is updated with the total and the ordered items are fetched.

LoginServlet

```
public User getUserByEmail(String email) throws SQLException {
    try (Connection connection = DatabaseUtil.getConnection(); PreparedStatement preparedStatement = connection.prepareStatement(
        "SELECT * FROM customers WHERE email = ?")) {

        preparedStatement.setString(1, email);

        try (ResultSet resultSet = preparedStatement.executeQuery()) {
            if (resultSet.next()) {
                User user = new User();

                user.setCustomerId(resultSet.getInt("customer_id"));
                user.setFirstName(resultSet.getString("first_name"));
                user.setLastName(resultSet.getString("last_name"));
                user.setEmail(resultSet.getString("email"));
                user.setPassword(resultSet.getString("password"));
                user.setPhone(resultSet.getString("phone_number"));
                return user;
            }
        }
    }
}
```

This JSP file represents our web application's login page.

Basic Setup: We've set the page language to Java, declared the character encoding, and structured our HTML document.

Styling and Title: Linked an external CSS file for styling and set the page title as "Login Page."

Main Container: Created a container with a heading "Login" to hold our login form.

Error Message Handling: Dynamically checks if there's an error message from the server. If yes, it displays a styled alert div to inform users about any issues during login.

Login Form: Designed a simple form with fields for email and password. It's configured to send a POST request to our "LoginServlet" upon submission.

Forgot Password and Registration Links: Provided links for users who might have forgotten their password or want to create a new account. These links point to related pages.

LogoutServlet

```
@WebServlet("/LogoutServlet")
public class LogoutServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // Invalidate the session and redirect to the home page
        request.getSession().invalidate();
        response.sendRedirect("Welcome.jsp");
    }
}
```

request. getSession(). invalidate (): Invalidates (closes) the current user session.

When a user accesses this servlet via a GET request, it ends their current session and redirects them to the "Welcome.jsp" page.

Password-Recovery Servlet

```
@WebServlet("/PasswordRecoveryServlet")
public class PasswordRecoveryServlet extends HttpServlet {

    private UserDao userDao = new UserDao();

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String email = request.getParameter("email");

        if (userDao.isUserExists(email)) {
            // Get the new password entered by the user
            String newPassword = request.getParameter("newPassword");

            // Update the user's password in the database
            userDao.updatePassword(email, newPassword);

            // Redirect to a Login page
            response.sendRedirect("Login.jsp");
        } else {
            request.setAttribute("errorMessage", "Entered email does not existing!");
            request.getRequestDispatcher("Forgot-Password.jsp").forward(request, response);
        }
    }
}
```

- The servlet handles a password update functionality.
- If the customer exists, it updates the password through the UserDao class and redirects to the login page.
- If the user does not exist, it sets an error message attribute, presumably to be displayed to the customer in the user interface (Login.jsp).

FeedbackServlet

```
try {
    connection = DatabaseUtil.getConnection();

    // Retrieve customer_id based on the entered email
    String email = request.getParameter("email");
    Integer customerId = null;

    // Query the database to get customer_id
    String query = "SELECT customer_id FROM customers WHERE email=?";
    PreparedStatement preparedStatement = connection.prepareStatement(query);
    preparedStatement.setString(1, email);
    ResultSet resultSet = preparedStatement.executeQuery();

    if (resultSet.next()) {
        // Check if the result set contains data before retrieving the customer ID
        customerId = resultSet.getInt("customer_id");
    }

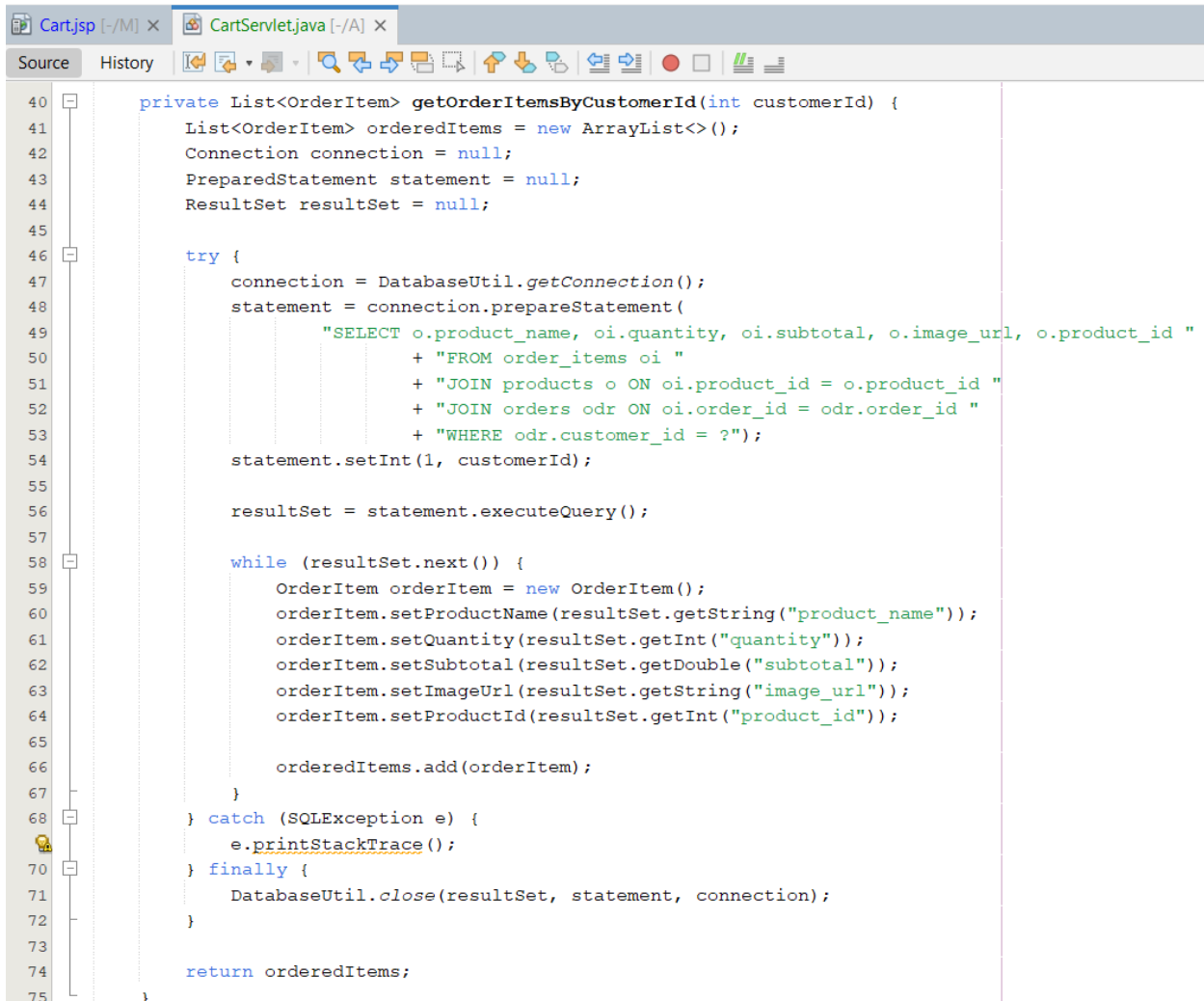
    // Insert feedback into the feedback table
    if (customerId != null) {
        String feedback = request.getParameter("feedback");

        query = "INSERT INTO feedback (customer_id, comments) VALUES (?, ?)";
        preparedStatement = connection.prepareStatement(query);
        preparedStatement.setInt(1, customerId);
        preparedStatement.setString(2, feedback);
        preparedStatement.executeUpdate();

        // Set success message as request attribute
        request.setAttribute("successMessage", "Feedback submitted successfully!");
    } else {
        request.setAttribute("errorMessage", "Error: Customer not found with the provided email address.");
    }
} catch (Exception e) {
    e.printStackTrace();
} finally {
```

- Created an API or backend functionality to collect feedback from users. This involved designing a form/mechanism to gather ratings, comments, or suggestions.
- Implemented a database to save and manage the feedback data securely, including details such as user ID (if logged in), timestamp, feedback content, and any associated metadata.
- Developed a reporting interface within the backend to generate summarized or detailed reports based on feedback data. This could include statistical analysis, trends, or visual representations of feedback.
- Only authenticated users can submit feedback. Implement security measures to prevent unauthorized access or manipulation of feedback data.
- The system allows administrators or moderators to respond to user feedback, creating a channel for communication and addressing user concerns or suggestions.

CartServlet



```
40 private List<OrderItem> getOrderItemsByCustomerId(int customerId) {
41     List<OrderItem> orderedItems = new ArrayList<>();
42     Connection connection = null;
43     PreparedStatement statement = null;
44     ResultSet resultSet = null;
45
46     try {
47         connection = DatabaseUtil.getConnection();
48         statement = connection.prepareStatement(
49             "SELECT o.product_name, oi.quantity, oi.subtotal, o.image_url, o.product_id "
50             + "FROM order_items oi "
51             + "JOIN products o ON oi.product_id = o.product_id "
52             + "JOIN orders odr ON oi.order_id = odr.order_id "
53             + "WHERE odr.customer_id = ?");
54         statement.setInt(1, customerId);
55
56         resultSet = statement.executeQuery();
57
58         while (resultSet.next()) {
59             OrderItem orderItem = new OrderItem();
60             orderItem.setProductName(resultSet.getString("product_name"));
61             orderItem.setQuantity(resultSet.getInt("quantity"));
62             orderItem.setSubtotal(resultSet.getDouble("subtotal"));
63             orderItem.setImageUrl(resultSet.getString("image_url"));
64             orderItem.setProductId(resultSet.getInt("product_id"));
65
66             orderedItems.add(orderItem);
67         }
68     } catch (SQLException e) {
69         e.printStackTrace();
70     } finally {
71         DatabaseUtil.close(resultSet, statement, connection);
72     }
73
74     return orderedItems;
75 }
```

The Cart Servlet is responsible for:

Session Handling: It might use the HttpSession to associate the cart information with a specific customer. The customer_id may be stored in the session to identify the customer and retrieve their specific cart details (pending orders).

Fetching Cart Items: When a customer visits their shopping cart page, the servlet retrieves the items they have added to the cart. This information is often stored in a database.

Calculating Total: It calculates the total price of all items in the cart. This information is usually obtained from the database as well.

Displaying Cart Contents: The servlet then sends this information to the Cart JSP (JavaServer Page) file, which is responsible for rendering the HTML content. The JSP uses this information to dynamically generate the cart page, showing the customer what items they have in their cart and the total cost.

ClearCartServlet

```
private void clearCart(int customerId) {
    Connection connection = null;
    PreparedStatement deleteOrderItemsStatement = null;
    PreparedStatement updateTotalStatement = null;

    try {
        // Get a database connection
        connection = DatabaseUtil.getConnection();

        // Delete order items for the given customer from the order_items table
        String deleteOrderItemsQuery = "DELETE FROM order_items WHERE order_id IN (SELECT order_id FROM orders WHERE customer_id = ?)";
        deleteOrderItemsStatement = connection.prepareStatement(deleteOrderItemsQuery);
        deleteOrderItemsStatement.setInt(1, customerId);
        int deletedRows = deleteOrderItemsStatement.executeUpdate();

        // If there were order items deleted, update the total in the orders table
        if (deletedRows > 0) {
            String updateTotalQuery = "UPDATE orders SET total = 0 WHERE customer_id = ?";
            updateTotalStatement = connection.prepareStatement(updateTotalQuery);
            updateTotalStatement.setInt(1, customerId);
            updateTotalStatement.executeUpdate();
        }
    }
}
```

- The servlet handles HTTP POST requests, retrieves customer ID, and performs clear cart operations.
- It opens a database connection and defines two Prepared Statement objects for SQL queries:
 - deleteOrderItemsStatement and updateTotalStatement.
- The servlet also executes a query to delete order items and update total amount if deleted.
- A transaction commits the transaction, ensuring atomicity and rolling back if necessary.

RemoveItemsServlet

```
46 String deleteOrderItemQuery = "DELETE FROM order_items WHERE order_id IN "
47     + "(SELECT order_id FROM orders WHERE customer_id = ?) AND product_id = ?";
48 deleteOrderItemStatement = connection.prepareStatement(deleteOrderItemQuery);
49 deleteOrderItemStatement.setInt(1, customerId);
50 deleteOrderItemStatement.setInt(2, productId);
51 int deletedRows = deleteOrderItemStatement.executeUpdate();
52
53 // If the order item was deleted, update the total in the orders table
54 if (deletedRows > 0) {
55     String updateTotalQuery = "UPDATE orders SET total = (SELECT SUM(subtotal) FROM order_items WHERE order_id IN "
56         + "(SELECT order_id FROM orders WHERE customer_id = ?)) WHERE customer_id = ?";
57     updateTotalStatement = connection.prepareStatement(updateTotalQuery);
58     updateTotalStatement.setInt(1, customerId);
59     updateTotalStatement.setInt(2, customerId);
60     updateTotalStatement.executeUpdate();
}
```

- The servlet seems to handle a password update functionality.
- If the customer exists, it updates the password through the UserDao class and redirects to the login page.
- If the user does not exist, it sets an error message attribute, presumably to be displayed to the customer in the user interface (Login.jsp).

PayPal Sandbox Configurations

The first screenshot shows a basic checkout form with fields for Address, Postal Code, and Phone. It includes radio buttons for 'Cash on Delivery' and 'PayPal', a 'PayPal' button, and a 'Debit or Credit Card' button. The second screenshot shows a detailed billing address form with fields for First name, Last name, Street address, Apt., ste., bldg., City, State, ZIP code, Mobile +1, and Email. It also includes a 'Ship to billing address' checkbox and a 'Pay Now' button. The third screenshot shows a mobile payment page with the PayPal logo, a total amount of \$ 9.44, a 'Log In' button, and a 'Pay with debit or credit card' section. It includes fields for Country/Region (Sri Lanka), Email (minuri@gmail.com), Phone type (Mobile), and Phone number (+94). The JavaScript code on the right side of the third screenshot shows the integration of the PayPal SDK, including the creation of an order and the handling of the payment process.

- The integration of PayPal Checkout into our web page has been successful, providing a secure payment experience.
- The process involved creating a developer account on the PayPal Developer website, using the sandbox environment for development and testing, obtaining a client ID, integrating the PayPal JavaScript SDK script, handling payments using the PayPal.
- Buttons API, implementing server-side logic, conducting thorough testing in the PayPal sandbox environment, and moving the integration to the live environment.

PaypalCheckoutServlet

```
13 @WebServlet("/PayPalCheckoutServlet")
14 public class PayPalCheckoutServlet extends HttpServlet {
15
16     @Override
17     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException {
18         // Retrieve data sent by the customer
19         String orderID = request.getParameter("orderID");
20         String paymentID = request.getParameter("paymentID");
21
22         // Process the payment (validate with PayPal API, update database, etc.)
23         boolean paymentSuccessful = processPayment(orderID, paymentID);
24
25         // Send a response to the client
26         PrintWriter out = response.getWriter();
27         response.setContentType("application/json");
28
29         // Send a JSON response indicating the payment status
30         out.print("{\"success\": " + paymentSuccessful + "}");
31         out.flush();
32     }
33
34     private boolean processPayment(String orderID, String paymentID) {
35         boolean updateSuccessful = updateOrderStatusAndPaymentCheck(orderID, paymentID);
36
37         // Return the result of the database update
38         return updateSuccessful;
39     }
40
41     //method for updating order status and payment check column
42     private boolean updateOrderStatusAndPaymentCheck(String orderID, String paymentID) {
```

- The `@WebServlet` annotation maps the servlet to the specified URL.
- The `doPost` method processes HTTP POST requests, retrieves customer data, and handles payment processing.
- The `update order status And Payment Check` method updates the order status and payment check in the database.
- A JSON response is sent to the client, and database connections are managed within a try-catch finally block.

SMS or Email notification of purchase and cancellation options

Order Confirmation Inbox x

sasankanimesh100@gmail.com
to me ▾
Dear Customer,

Thank you for your order! We are pleased to confirm your recent purchase.

Order Details:
Product: Cheese
Quantity: 1
Subtotal: Rs.490.0

Product: Cakes
Quantity: 1
Subtotal: Rs.580.0

Product: Milk
Quantity: 1
Subtotal: Rs.160.0

Total: Rs.1230.0

We appreciate your business and look forward to serving you again.
Best regards,
Green Supermarket.

Order Cancelled Inbox x

sasankanimesh100@gmail.com
to me ▾
Dear Customer,

We wanted to inform you that your order has been canceled successfully.
If you have any questions or concerns, please do not hesitate to contact our support team.
Thank you for considering us. We appreciate your business.

Best regards,
GREEN Supermarket.

Reply Forward

```
History
import java.util.Properties;

public class EmailSender {
    public static void sendEmail(String toEmail, String subject, String body) {
        final String username = "sasankanimesh100@gmail.com";
        final String password = "cgll yqrx skfe wjit";

        Properties props = new Properties();
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.host", "smtp.gmail.com");
        props.put("mail.smtp.port", "587");

        Session session = Session.getInstance(props,
            new Authenticator() {
                @Override
                protected PasswordAuthentication getPasswordAuthentication() {
                    return new PasswordAuthentication(username, password);
                }
            });

        try {
            Message message = new MimeMessage(session);
            message.setFrom(new InternetAddress(username));
            message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(toEmail));
            message.setSubject(subject);
            message.setText(body);

            Transport.send(message);

            System.out.println("Email sent successfully.");
        } catch (MessagingException e) {
            e.printStackTrace();
        }
    }
}
```

Credentials: Set up Gmail username and app password for authentication.

SMTP Configuration: Configured the necessary properties for sending emails using the SMTP protocol, including authentication, host (smtp.gmail.com), and port (587).

Email Composition: A method (sendEmail) to compose and send emails. It takes the recipient's email address, subject, and body.

Message Configuration: Configured the sender, recipient, subject, and body of the email using a MimeMessage object.

Email Sending: Initiated the sending process using Transport. Send(message).

CheckoutServlet

```
33 } catch (NumberFormatException e) {
34     // Handle the case where the postal code is not a valid integer
35     response.getWriter().println("Postal code must be a valid number.");
36     return; // Exit the method to avoid further processing
37 }
38
39 HttpSession session = request.getSession();
40 int customerId = (int) session.getAttribute("customer_id");
41
42 // Update customer details in the database
43 boolean updateSuccess = updateCustomerDetails(customerId, firstName, lastName, address, postalCode, phone);
44
45 if (updateSuccess) {
46     // Fetch ordered items
47     List<OrderItem> orderedItems = getOrderedItems(customerId);
48
49     // Send the order confirmation email
50     sendOrderConfirmationEmail(getEmailByCustomerId(customerId), orderedItems);
51
52     // Invalidate the session
53     //session.invalidate();
54
55     // Forward to the order confirmation page
56     updateOrderStatus(customerId);
57
58     request.setAttribute("orderedItems", orderedItems);
59     request.getRequestDispatcher("OrderConfirmation.jsp").forward(request, response);
60 } else {
61     // Handle the update failure
62     response.getWriter().println("Failed to update customer details.");
63 }
64
65 private void updateOrderStatus(int customerId) {
66     try (Connection connection = DatabaseUtil.getConnection();
67         PreparedStatement updateStmt = connection.prepareStatement(
68             "UPDATE orders SET status = 'CONFIRMED' WHERE customer_id = ? AND status = 'PENDING'")) {
```

Form Data Handling: Retrieves customer updated details (first name, last name, address, postal code, phone) from the checkout form.

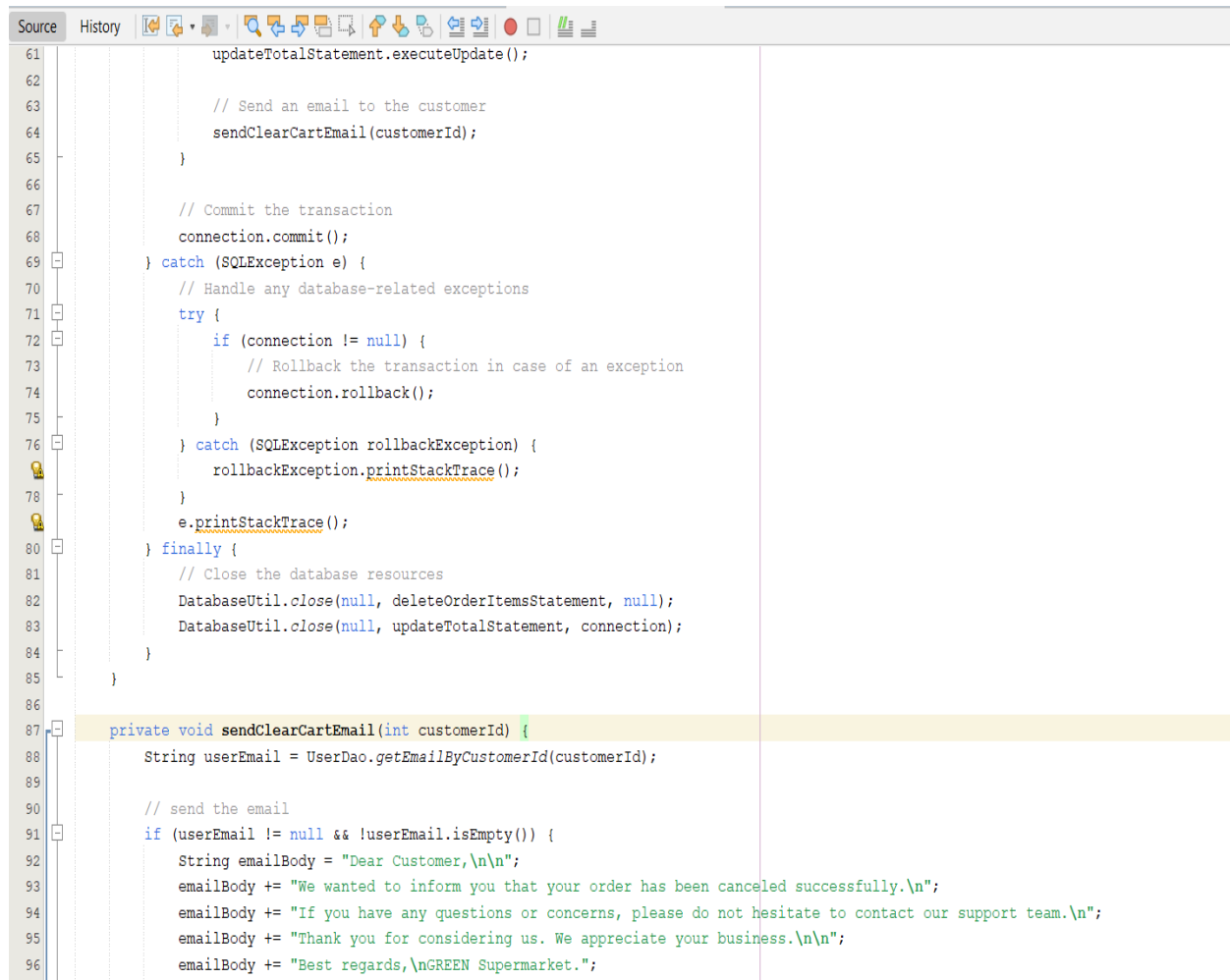
Database Update: Updates customer details in the database using a prepared statement.

Order Confirmation Email: Fetches ordered items and sends a confirmation email to the customer using the EmailSender class.

Order Status update: Updates the order status to 'CONFIRMED' in the database.

Forwarding: Forwards the request to the OrderConfirmation.jsp page.

OrderCancellationServlet



```
61         updateTotalStatement.executeUpdate();
62
63         // Send an email to the customer
64         sendClearCartEmail(customerId);
65     }
66
67     // Commit the transaction
68     connection.commit();
69 } catch (SQLException e) {
70     // Handle any database-related exceptions
71     try {
72         if (connection != null) {
73             // Rollback the transaction in case of an exception
74             connection.rollback();
75         }
76     } catch (SQLException rollbackException) {
77         rollbackException.printStackTrace();
78     }
79     e.printStackTrace();
80 } finally {
81     // Close the database resources
82     DatabaseUtil.close(null, deleteOrderItemsStatement, null);
83     DatabaseUtil.close(null, updateTotalStatement, connection);
84 }
85
86
87 private void sendClearCartEmail(int customerId) {
88     String userEmail = UserDao.getEmailByCustomerId(customerId);
89
90     // send the email
91     if (userEmail != null && !userEmail.isEmpty()) {
92         String emailBody = "Dear Customer,\n\n";
93         emailBody += "We wanted to inform you that your order has been canceled successfully.\n";
94         emailBody += "If you have any questions or concerns, please do not hesitate to contact our support team.\n";
95         emailBody += "Thank you for considering us. We appreciate your business.\n\n";
96         emailBody += "Best regards,\nGREEN Supermarket.";
```

Order Status Update: Ensures that only confirmed orders are canceled by updating the order status to 'CANCELLED' in the database.

Clearing Order Items: Deletes order items associated with the customer from the order items table. This ensures that the user's cart is cleared of items.

Updating Order Total: Updates the total in the orders table to 0 if order items are deleted. This reflects the correct total when the cart is cleared.

Email Notification: Sends a personalized email to the customer notifying them of the successful order cancellation. The email contains a thank-you message and contact information for any questions or concerns.

Redirect After Cancellation: Redirects the user to the "Welcome.jsp" page after the cart is cleared and the order is canceled. This provides a smooth user experience.

GitHub Project Link

<https://github.com/sasankanimesh/Java-web-application.git>

References

Green's Supermarket. (2022). Homepage. Available at: <https://www.greens.com.mt/> (Accessed: 27/12/2023).

Contribution

	Group Number	B2	
	Student ID (Plymouth)	Name	Contribution
1	10899343	Priyantha Ranasinghe	Back-end developing, PayPal configurations, Email Configurations
2	10899228	Rankira Kosgollage	Front-end, Back-end developing
3	10899233	Dodampe Nimna	Front-end Developing, ER-Diagram
4	10903090	Ranasinghe Ranasinghe	Back-end Developing, Use case-Diagram
5	10899368	Andara Siddhanth	Front-end Developing, Class-Diagram
6	10899313	Agalakotuwe Jayawardhana	Front-end Developing, Sequence-Diagram