VIETNAM NATIONAL UNIVERSITY OF HO CHI MINH CITY

INTERNATIONAL UNIVERSITY

SCHOOL OF INFORMATION TECHNOLOGY

Coffee Management System

Group 11

Students' name: Huynh Tuan Anh ITITIU23003

Tran Thi Truc Mai ITCSIU23024

Course: IT093IU – Web Application Development

Instructors: Assoc. Prof. Nguyen Van Sinh

Msc. Nguyen Trung Nghia

Date: 27th of December, 2025

Ho Chi Minh city

December, 2025

Abstract

This project demonstrates how we can apply technology, techniques, and theoretical knowledge in the Web Application Development course to design and implement a coffee management system web-based application. By featuring a user-friendly interface, this website helps the customers to browse menus, customize drinks, place orders, and purchase beverages and cakes conveniently for home delivery.

The project integrates core concepts such as system analysis and design, database management, server-side programming, and client-side interaction to deliver a complete end-to-end solution. In addition, essential algorithms for order processing, cart management, and price calculation are implemented to ensure accuracy and efficiency. The overall development process, including system architecture, functional modules, and implementation techniques, is explained in detail to demonstrate how theoretical knowledge can be transformed into a practical, real-world web application.

Table of Contents

1 Introduction

1.1 Project Overview

The Coffee Management System is a web-based application developed to support online coffee ordering and basic administrative management for a coffee shop. The system allows customers to browse the menu, add items to a shopping cart, place orders, track orders, and make orders through different payment methods. For the admin site, the administrators are able to manage products, view orders, view users, and update order statuses.

This project was developed as an end-of-course assignment to demonstrate the application of web development concepts using Java-based technologies, including Java Servlets, JavaServer Pages (JSP), and a relational database. The system follows a structured development approach MVC to ensure clarity, maintainability, and usability.

1.2 Objectives of the System

The main objectives of this project are as follows:

- Applying the techniques, tools, and knowledge learned in the Web Application Development course to a real-world application scenario.
- Designing and developing a functional coffee management system that supports online ordering and product customization.
- Building a user-friendly web interface that enhances user experience and simplifies the ordering process.
- Implementing core system functionalities such as user authentication, cart management, order processing, and payment handling.

1.3. Scope of the Project

The scope of the Coffee Management System includes two main user roles: customers and administrators. Customers can register an account, log in, browse the menu, manage their shopping cart, place orders, and view their order history. Administrators can log in to the admin dashboard to manage products, users, and orders.

The system focuses on essential functionalities required for a small-scale coffee ordering platform. Advanced features such as real-time order tracking, mobile application support, and third-party delivery services are outside the scope of this project and may be considered for future enhancements.

2. System Overview

2.1.Description of the Coffee Management System

The Coffee Management System is a web-based application designed to facilitate online coffee ordering and basic shop management operations. The system provides an interface for customers to view available coffee and food items, place orders, and complete payments online. In addition, it offers an administrative interface that allows staff to manage products, users, and customer orders efficiently.

The application is developed using Java Servlet and JSP technologies, following a clear separation between presentation, business logic, and data access layers. This approach improves system maintainability and supports structured development.

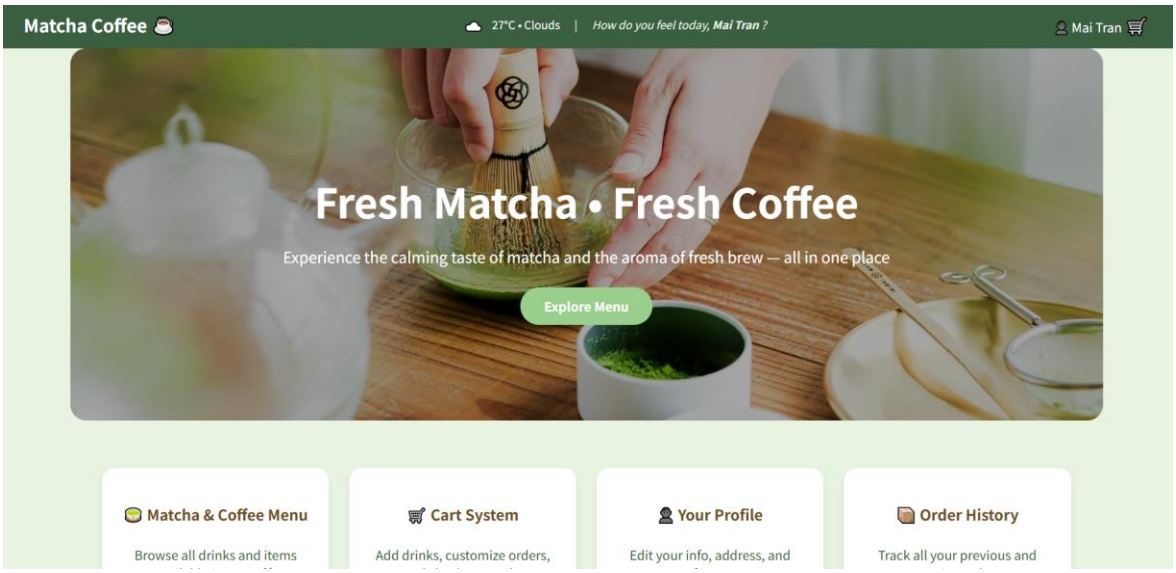


Figure 2.1 Homepage of the Matcha Coffee web application.

2.2. Users role

The system supports two main user roles:

- Customer: Customers can register an account, log in, browse the coffee menu, add items to cart, proceed to checkout, choose a payment method, and view their other history.
- Administrator: Administrators can access the admin dashboard to view the total users, revenue, and products. They can also manage the menu items, view and process customer orders, view the users, and monitor overall system activity.

There is one more role, which is guest. The guest can only access to the homepage of the system. If they want to make an order, they have to log in or register an account.

Each role is controlled through authentication and authorization mechanisms to ensure that users can only access functions relevant to their permissions.

2.3.Main features of the System

The main features of the Coffee Management System include user authentication and account management, menu display and product management, shopping cart functionality, order placement and order tracking, multiple payment options, and an administrative dashboard for managing system data.

These features cover the essential requirements of a small-scale coffee ordering platform and demonstrate the integration of frontend interfaces with backend processing and database operations.

3. System Overview

Diagrams are used in this report to provide a clear visual representation of the system's functionality, structure, and data relationships. They help illustrate how users interact with the system, how different components are connected, and how data is stored and managed within the database.

3.1. Use Case Diagram

The Use Case Diagram illustrates the main interactions between users and the Coffee Management System. The system supports two primary actors: Guest/Customer and Administrator. Each actor is associated with a set of actions that represent the core functionalities available to them.

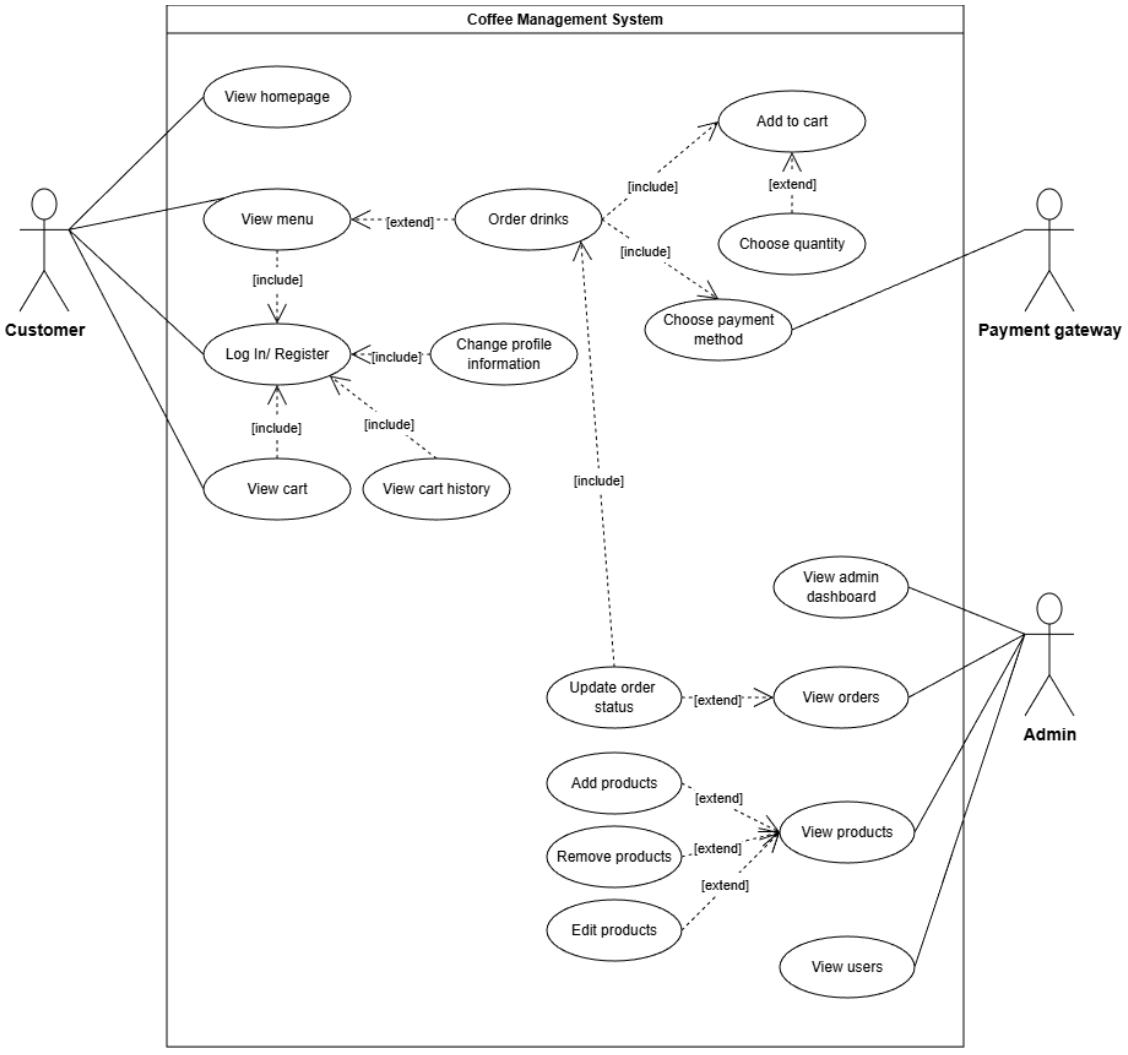


Figure 3.1. Use Case Diagram for Coffee Management System

The customers or guests can view the homepage without logging in/registering. However, if the guests want to view menu, view cart, view cart history or change their personal information, they will need to log

in or register first. Therefore, the arrows represent as “include” relationship. When viewing the menu, customers can order drinks, they have to add the items to the cart, choose quantity if wanted, and choose payment method. For the administrator side, they can view the admin dashboard to see the total revenue, total users, total orders, and total products. The administrator can view the orders and change the order status or payment status. Moreover, they can modify the products, which are adding, disabling, or editing the products information. Lastly, the admin can view the basic information of users inside the system.

This diagram provides a high-level overview of system functionality from the user’s perspective and helps clarify the responsibilities of each user role.

3.2.Database Diagram (ERD)

The Entity Relationship Diagram (ERD) illustrates the structure of the database used in the Coffee Management System and describes how data entities are related to one another. Two levels of representation are used to provide a complete understanding of the system. The conceptual ERD presents a high-level view of the system by identifying the main entities such as Users, Orders, Menu, Cart Items, Shipping, and Shipping Zone, as well as the relationships between them. This diagram focuses on business logic and helps explain how different components of the system interact conceptually.

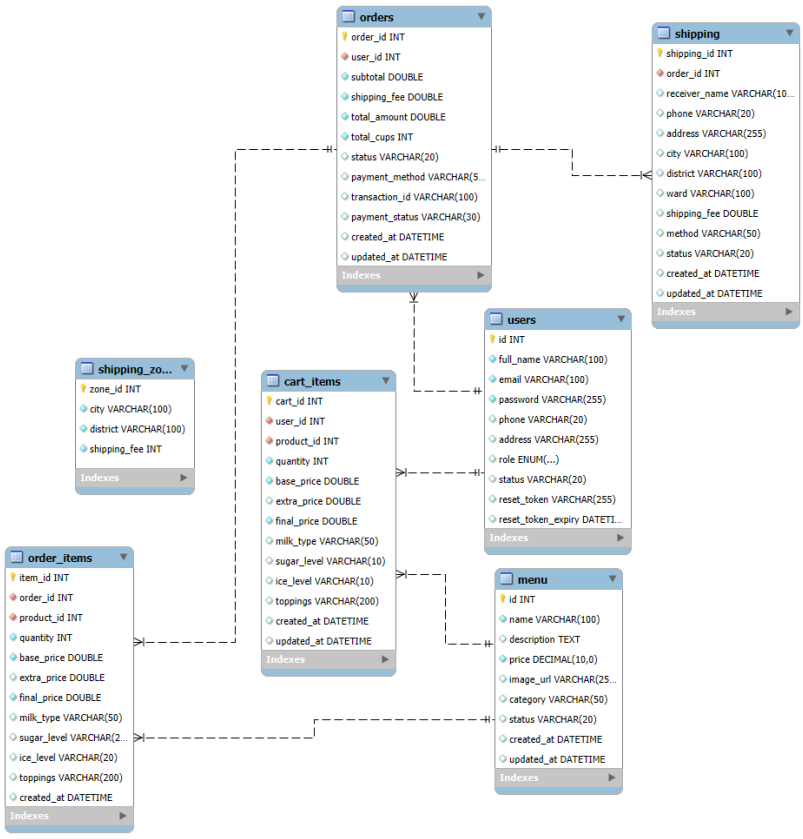


Figure 3.2. Logical Entity Relationship Diagram (ERD) of the Coffee Management System

The logical ERD *(figure 3.3.)* further expands this design by detailing the attributes of each entity and defining primary and foreign key relationships. It shows how data is physically organized in the database, including table structures, data fields, and relational links. For example, each user can place multiple orders, each order can contain multiple order items, and each order is associated with a shipping record. These relationships ensure data consistency and support core system functions such as order processing, payment handling, and delivery tracking.

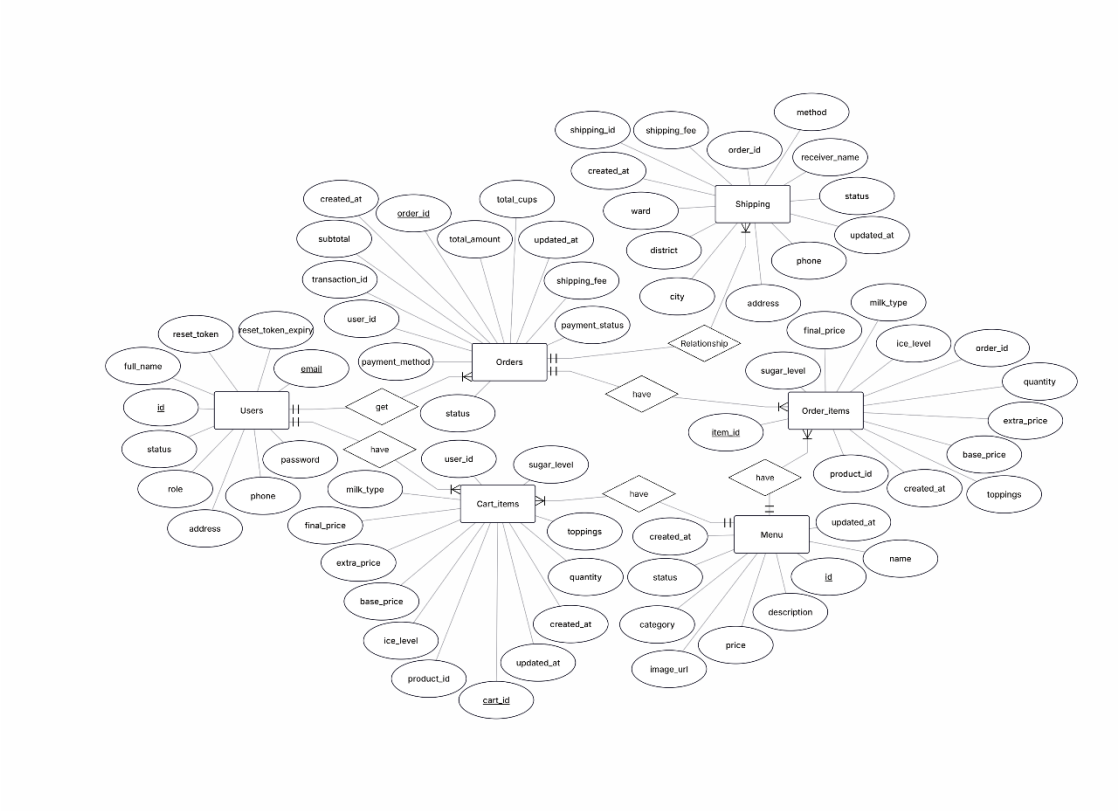


Figure 3.3.Conceptual Entity Relationship Diagram (ERD) of the Coffee Management System

3.3.Sequence Diagrams

Sequence diagrams are used to describe the detailed execution flow of important features. They show the interaction between the browser (user), JSP pages, servlet controllers, DAO layer, and the database. This helps explain how requests are handled and how data moves through the system for each major workflow.

3.3.1. [User] User Registratrion Sequence

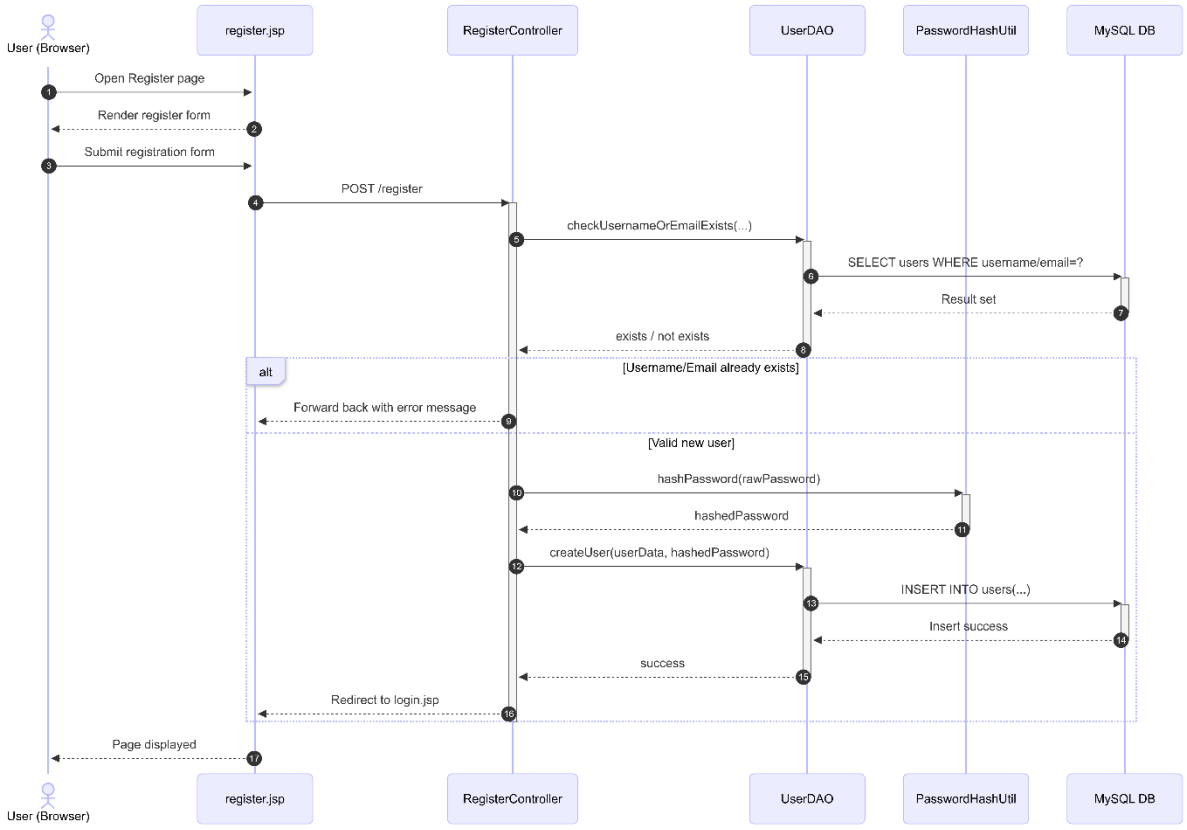


Figure 3.4 User Registration Sequence Diagram

This sequence diagram illustrates the workflow of the user registration and authentication process in the Coffee Management System. It shows how requests are handled from the browser through the controller, data access layer, and database, as well as how validation and security checks are performed.

Step 1: Accessing the Registration Page

The process begins when the user accesses the registration page by clicking the register option or navigating directly to the registration URL. The browser sends a **GET request** to the server, which is handled by the `RegisterController`. The controller forwards the request to `register.jsp`, allowing the user to view and fill in the registration form.

Step 2: Submitting the Registration Form

After entering the required information, the user submits the registration form. This action sends a **POST request** to the `RegisterController`, containing user input such as username, email, and password.

Step 3: Checking for Existing Users

The controller calls the `UserDAO` to verify whether the submitted username or email already exists in the database. This is done by executing a database query through the DAO layer. If a matching record is found, the system determines that the user already exists.

If the username or email already exists, the controller forwards the request back to the registration page and displays an error message to inform the user.

Step 4: Password Hashing and User Creation

If the user does not already exist, the system proceeds with registration. The raw password entered by the user is passed to the `PasswordHashUtil`, where it is securely hashed before being stored.

The controller then calls the `createUser()` method in `UserDAO`, which inserts the new user record into the database with the hashed password.

Step 5: Registration Success and Redirection

Once the user data is successfully stored in the database, the controller redirects the user to the login page. At this stage, the user can log in using the newly created credentials.

Step 6: Authentication and Access Control (Filter Handling)

When accessing protected resources, a filter intercepts incoming requests. The filter checks whether the requested resource is a static file or a public page. If so, the request is allowed to proceed without authentication.

For protected resources, the filter checks whether a valid user session exists. If the user is not authenticated, the system redirects them to the login page. If authentication is successful, the request continues to the appropriate controller.

3.3.2. [User] User Login Sequence

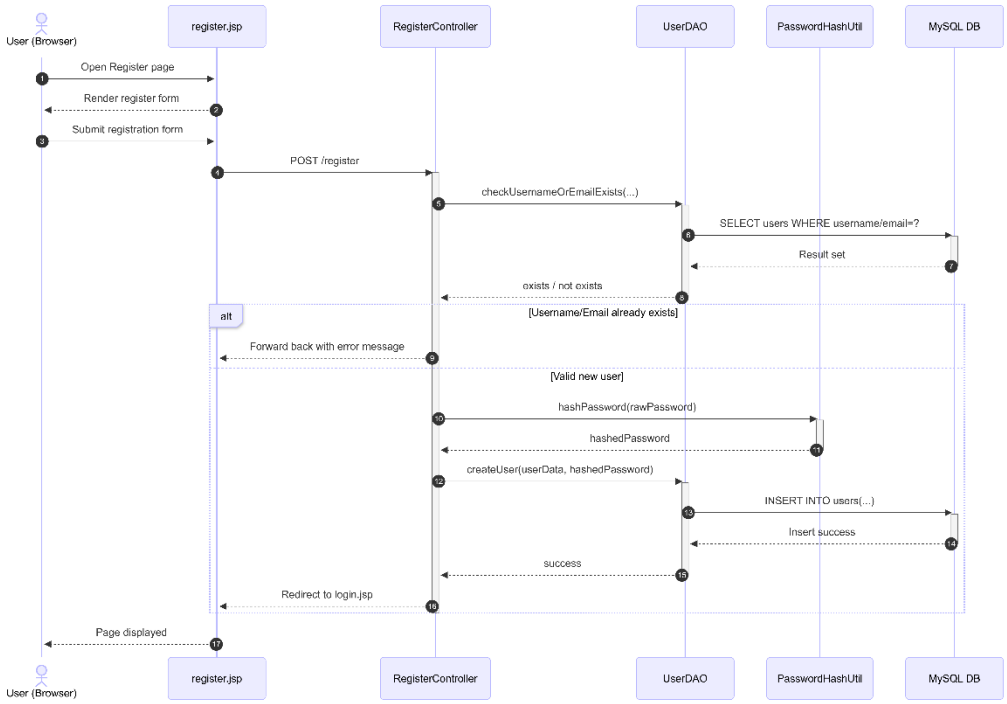


Figure 3.5 User Log In Sequence diagram

Step 1: Accessing the Login Page

The process begins when the user accesses the login page by clicking the login option or navigating directly to the login URL. The browser sends a **GET request** to the server, which is handled by the `LoginController`. The controller forwards the request to `login.jsp`, where the login form is displayed for the user to enter their credentials.

Step 2: Submitting Login Credentials

After entering the username or email and password, the user submits the login form. This action sends a **POST request** to the `LoginController`, containing the login credentials entered by the user.

Step 3: Retrieving User Information

The `LoginController` calls the `UserDAO` to retrieve user information based on the provided username or email. The DAO executes a database query to check whether a matching user record exists in the system.

If no matching record is found, the system identifies the login attempt as invalid.

Step 4: Handling Invalid User Credentials

If the user does not exist or the provided credentials are incorrect, the controller forwards the request back to `login.jsp`. An error message is displayed to inform the user that the login credentials are invalid.

Step 5: Password Verification

If a matching user record is found, the controller passes the entered password to the `PasswordHashUtil`. The utility compares the raw password with the stored hashed password in the database to verify its validity.

Step 6: Successful Authentication

If the password verification is successful, the system considers the user authenticated. The controller creates or retrieves the current HTTP session and stores the user information in the session using `setAttribute()`.

Step 7: Redirecting After Login

After successful authentication, the user is redirected to the main page (such as `menu.jsp`). The system then allows the user to access protected resources based on their authenticated session.

Step 8: Access Control via Filter

For subsequent requests, a filter checks whether the requested resource is protected. If the user is authenticated, the request is allowed to proceed to the target controller or page. If not, the user is redirected back to the login page.

3.3.3. [User] Add to Cart and Update Quantity sequence

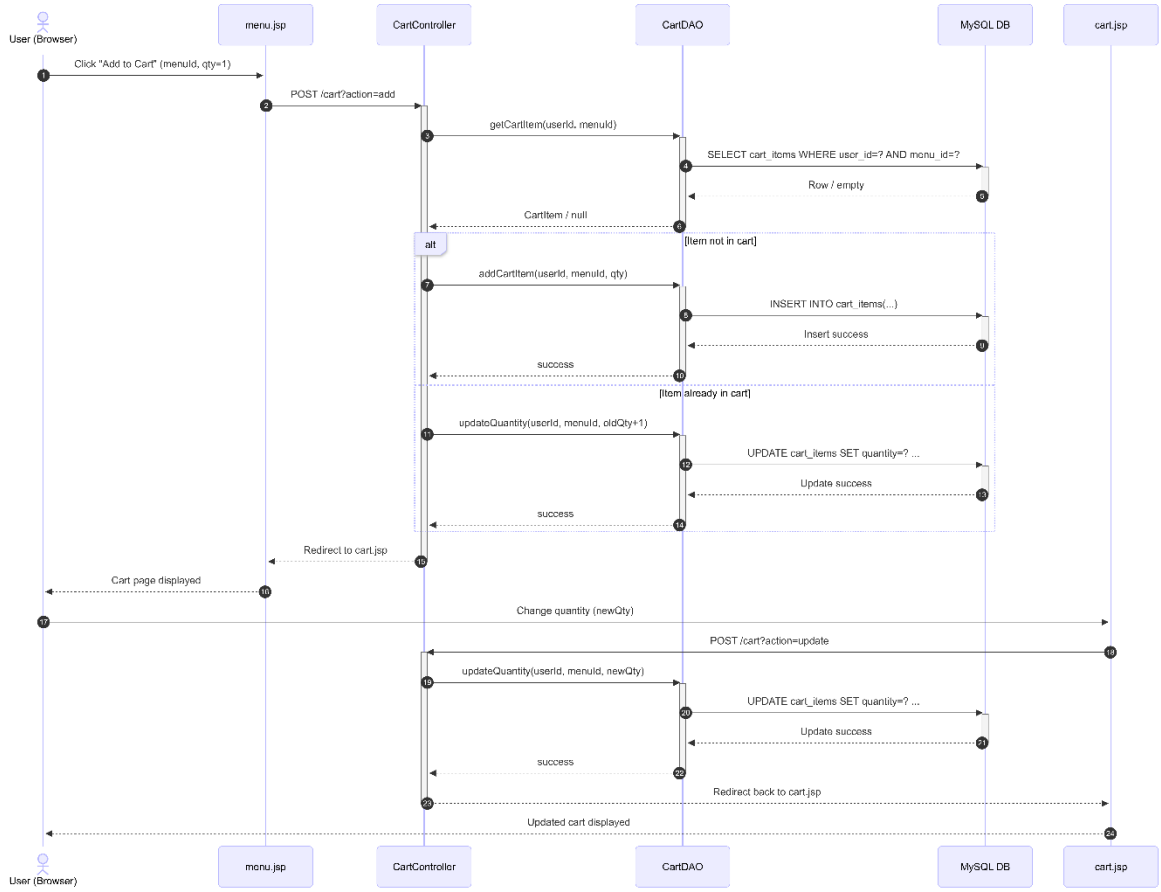


Figure 3.6 Adding Items and Update the quanity Sequence Diagram

The sequence diagram illustrates how the Coffee Management System handles the shopping cart process, focusing on two key functions: adding a product to the cart from the menu page and updating the quantity of items in the cart. It shows the interaction between the user's browser, JSP pages, the `CartController`, the `CartDAO`, and the MySQL database to ensure cart data is correctly stored and updated.

Step 1: Adding an Item to Cart from the Menu Page

The workflow begins when the user clicks **"Add to Cart"** on `menu.jsp`. This action sends a **POST request** to the server (`/cart?action=add`), which is handled by the `CartController`. The request includes the selected `menuId` and an initial quantity (typically `qty = 1`).

Step 2: Checking Whether the Item Already Exists in the Cart

After receiving the request, the `CartController` calls `CartDAO.getCartItem(userId, menuId)` to check if the same product already exists in the user's cart. The `CartDAO` queries the database using the `userId` and `menuId`.

If a matching record is found, it means the item is already in the cart.

If no record is found, it means the item is not yet in the cart.

Step 3: Adding a New Item (Item Not in Cart)

If the item does not exist in the cart, the controller proceeds to add it as a new cart entry. The `CartController` calls `CartDAO.addCartItem(userId, menuId, qty)`, and the DAO performs an **INSERT** operation into the `cart_items` table. If the insert is successful, the cart is updated with the new item and quantity.

Step 4: Updating Quantity for Existing Item (Item Already in Cart)

If the item already exists in the cart, the system updates its quantity instead of inserting a new row. The controller calculates the updated quantity (for example, `oldQty + 1`) and calls `CartDAO.updateQuantity(userId, menuId, newQty)`. The DAO then executes an **UPDATE** query to modify the existing cart record.

Step 5: Redirecting to Cart Page

After either inserting a new cart item or updating an existing one, the `CartController` redirects the user to `cart.jsp`. The updated cart content is then displayed to the user.

Step 6: Updating Item Quantity in Cart

The second part of the workflow happens when the user changes the quantity of an item directly on `cart.jsp`. When the user submits the new quantity, the page sends a **POST request** to `/cart?action=update`, which is again handled by the `CartController`.

Step 7: Saving the Updated Quantity to the Database

The `CartController` calls `CartDAO.updateQuantity(userId, menuId, newQty)` to save the new quantity. The DAO executes an **UPDATE** query on the `cart_items` table. Once the update succeeds, the controller redirects the user back to `cart.jsp`.

Step 8: Displaying the Updated Cart

After the redirection, the updated cart is displayed on `cart.jsp`, reflecting the new quantity selected by the user. This ensures the cart data remains consistent between the user interface and the database.

3.3.4. [User] Check out and Payment Sequence

This section explains the checkout and payment process of the system. The workflow is divided into three main parts:

- Order Creation and Checkout Process
- Payment Handling (COD, VietQR, Card) and Order Completion Flow

Each part represents a logical step in the overall order flow.

Order Creation and Checkout Process

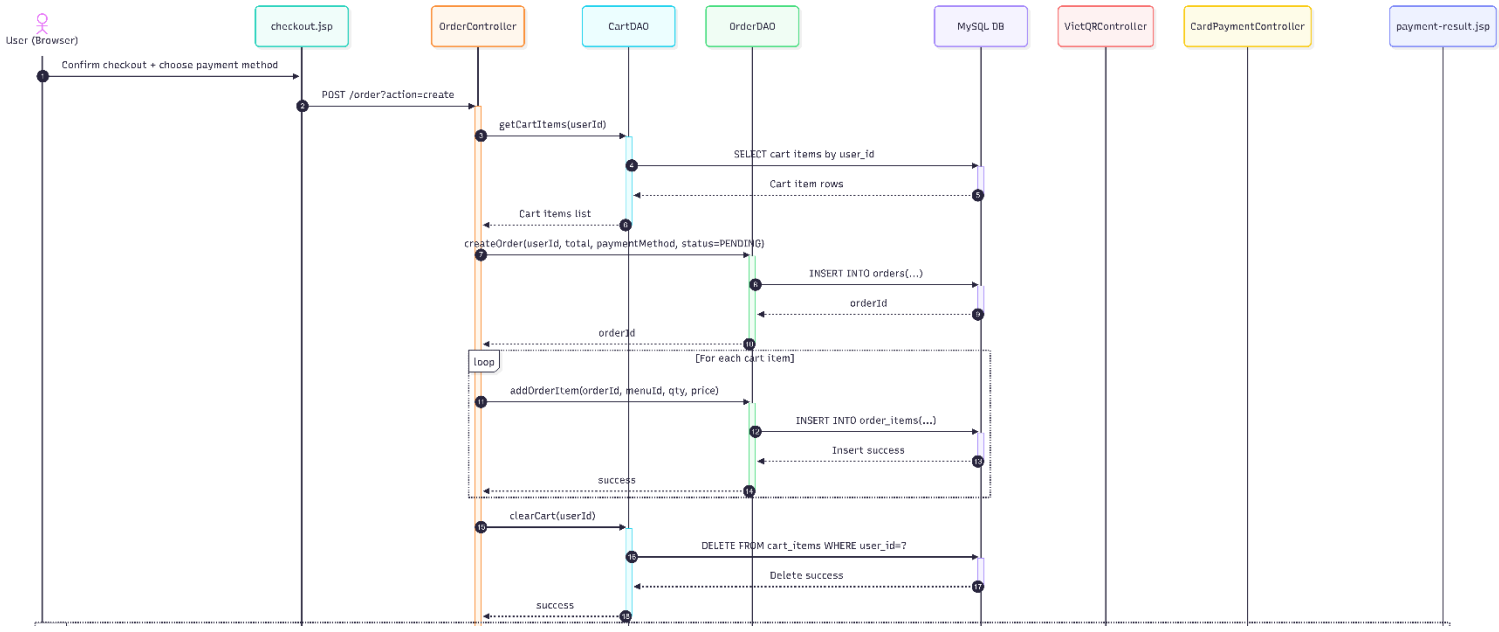


Figure 3.7. Order Creation and Checkout Sequence Diagram

Step 1: Accessing the Checkout Page

The process begins when the user clicks the **"Checkout"** button on `cart.jsp`. The browser redirects the user to `checkout.jsp`, where order confirmation and payment options are displayed.

Step 2: Submitting the Checkout Request

After reviewing the order and selecting a payment method, the user submits the checkout form. A **POST request** is sent to `OrderController` with the action `create`.

Step 3: Retrieving Cart Items

The `OrderController` retrieves all cart items associated with the current user by calling `CartDAO.getCartItems(userId)`.
The DAO queries the database and returns the list of cart items belonging to the user.

Step 4: Creating the Order Record

Once the cart data is retrieved, the system creates a new order record. The controller calls `OrderDAO.createOrder(...)`, which inserts a new row into the `orders` table with:

- user ID
- total amount
- selected payment method
- initial payment status (e.g. `PENDING`)

The generated `orderId` is returned for further processing.

Step 5: Creating Order Items

For each item in the cart, the system creates a corresponding record in the `order_items` table. This is done by calling `OrderDAO.addOrderItem(...)` in a loop, ensuring all products are linked to the newly created order.

Step 6: Clearing the Cart

After successfully creating the order and its items, the system clears the user's cart by calling `CartDAO.clearCart(userId)`. This ensures that the cart is empty once the checkout process is completed.

Payment Handling Flow

This section describes how the system processes payment and finalizes an order after the checkout process has been completed.

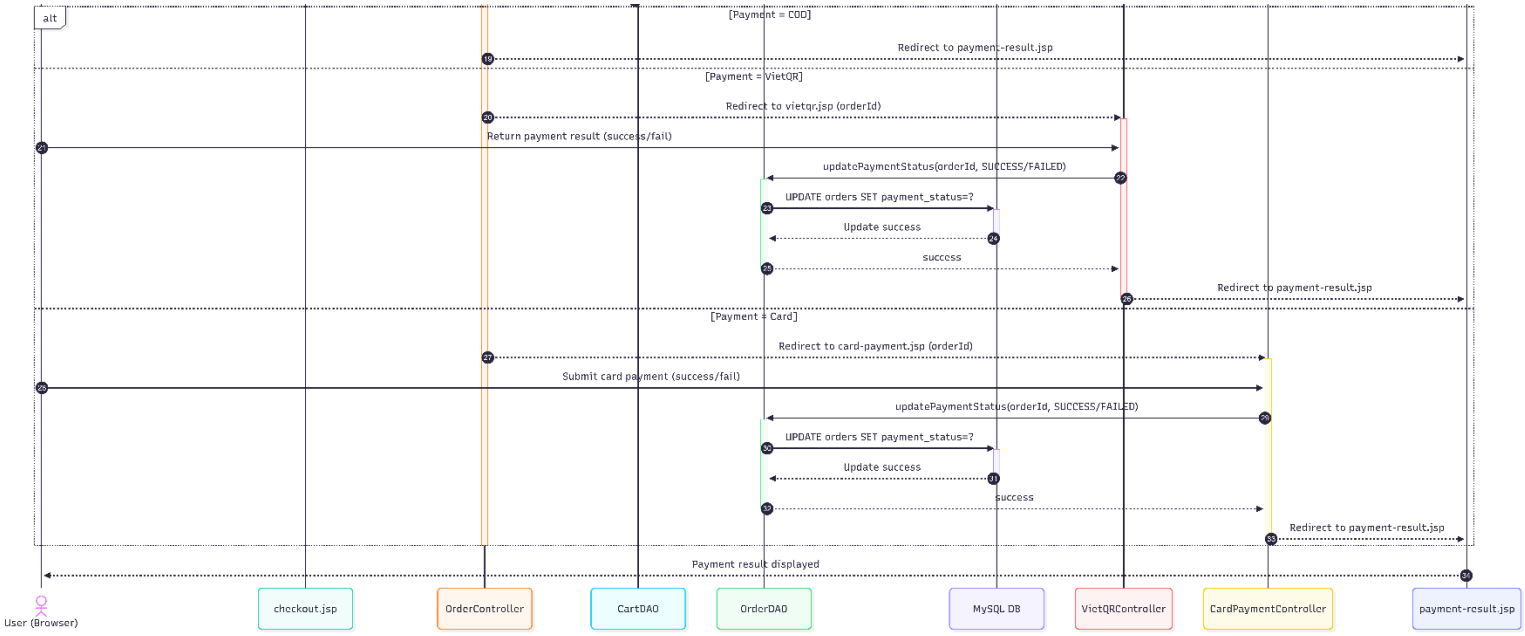


Figure 3.8. Payment Handling and Order Completion

Step 1: Payment Method Selection

After the order is created and stored in the system, the user selects a payment method on the checkout page. The available options include:

- Cash on Delivery (COD)
- VietQR payment
- Credit/Debit card payment

Based on the selected method, the system redirects the user to the corresponding payment handling process.

Step 2: Cash on Delivery (COD) Processing

When the user selects **Cash on Delivery**, the system does not require external payment verification.

The order is immediately marked with an appropriate payment status (e.g., *Pending* or *Confirmed*), and the user is redirected to the payment result page.

The order remains stored in the database and will be processed by the administrator.

Step 3: VietQR Payment Processing

If the user selects **VietQR**, the system redirects the user to the VietQR payment page.

After the payment attempt, the `VietQRController` receives the payment result and updates the order's payment status in the database as either **Success** or **Failed**.

Once the update is completed, the user is redirected to the payment result page to view the transaction outcome.

Step 4: Card Payment Processing

For card payments, the user is redirected to the card payment page. After the payment is processed, the `CardPaymentController` updates the payment status in the database accordingly. The system then redirects the user to the payment result page.

Step 5: Displaying the Payment Result

Finally, the system displays the payment result on `payment-result.jsp`. This page shows whether the payment was successful or failed and reflects the final order status stored in the database. This completes the checkout and payment workflow.

3.3.5. [Admin] Admin Product Management

This sequence diagram describes the **Admin Product Management workflow** in the Coffee Management System. It shows how the admin loads the product list on `admin-products.jsp`, and how product management actions (add, disable, remove) are processed through `AdminProductsController`, `AdminDAO`, and the MySQL database, before the updated product list is displayed again.

Viewing the Product List

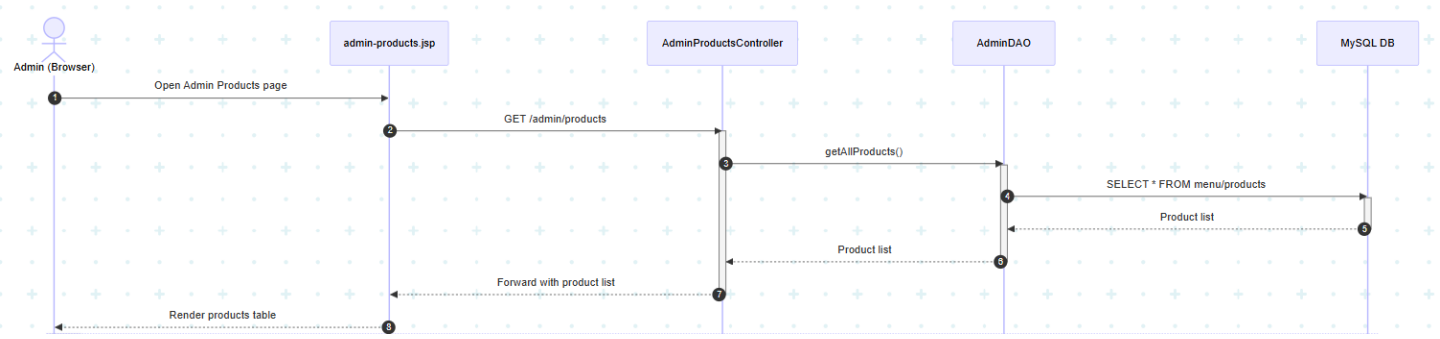


Figure 3.9 Viewing the Product List Sequence Diagram

Step 1: Opening the Admin Products Page

The workflow begins when the admin opens the Admin Products page in the browser. The page sends a **GET request** to `/admin/products`, which is handled by `AdminProductsController`.

Step 2: Retrieving All Products

The controller calls `AdminDAO.getAllProducts()` to retrieve the full product list. The DAO queries the database using a `SELECT` statement to fetch all products from the menu/products table.

The database returns the product list to the DAO, which then returns it to the controller.

Step 3: Displaying the Products Table

After receiving the product list, the controller forwards the request to `admin-products.jsp` along with the product data.

The JSP renders a table displaying all products, including their names, prices, images, and current status.

Add/Disable/Remove products

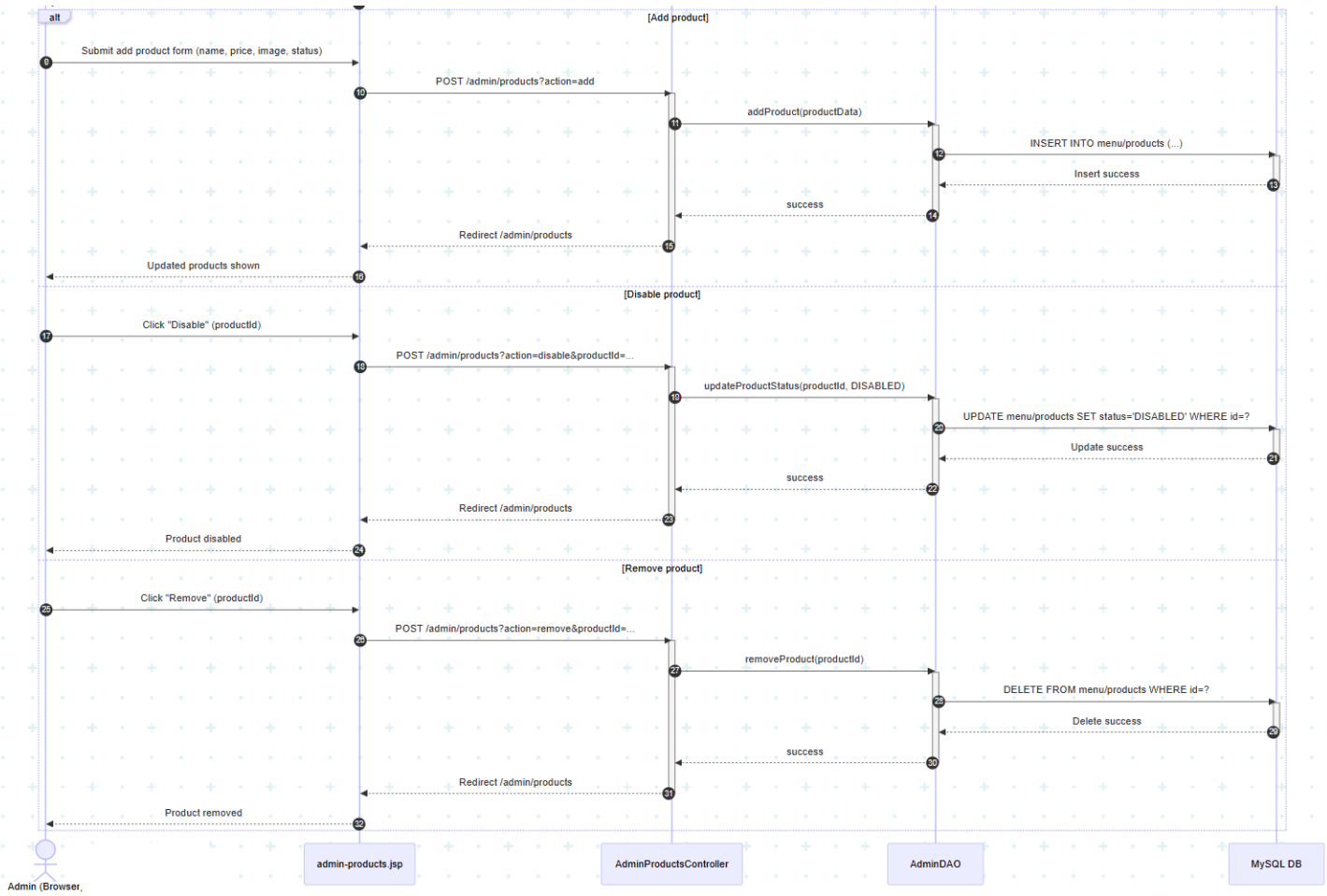


Figure 3.10 Add/Disable/Remove products Sequence Diagram

Adding a New Product

Step 1: Submitting the Add Product Form

When the admin submits the add product form on `admin-products.jsp`, the browser sends a **POST** to: `/admin/products?action=add`

This request contains the product information such as name, price, image, and status.

Step 2: Inserting the Product into the Database

`AdminProductsController` calls `AdminDAO.addProduct(productData).`

The DAO executes an `INSERT` query to add the new product record into the database.

Step 3: Refreshing the Product List

After the insert succeeds, the controller redirects back to `/admin/products`.

The updated product list is retrieved again and displayed on `admin-products.jsp`.

Disabling an Existing Product

Step 1: Admin Clicks "Disable"

When the admin clicks the **Disable** action for a specific product, the browser sends a **POST request** to: `/admin/products?action=disable&productId=...`

Step 2: Updating Product Status

The controller calls `AdminDAO.updateProductStatus(productId, DISABLED)`.

The DAO updates the product's status in the database using an `UPDATE` query, changing the status field to `DISABLED`.

Step 3: Displaying Updated Status

After the update succeeds, the controller redirects back to `/admin/products`. The product list is reloaded and the disabled product is shown with its updated status.

Removing a Product

Step 1: Admin Clicks "Remove"

When the admin clicks the **Remove** action for a product, the browser sends a **POST request** to: `/admin/products?action=remove&productId=...`

Step 2: Deleting the Product Record

The controller calls `AdminDAO.removeProduct(productId)`.

The DAO performs a `DELETE` query to remove the product from the database.

Step 3: Reloading the Product List

After deletion, the controller redirects back to `/admin/products`.

The product list is loaded again, and the removed product no longer appears in the table.

Edit products flow

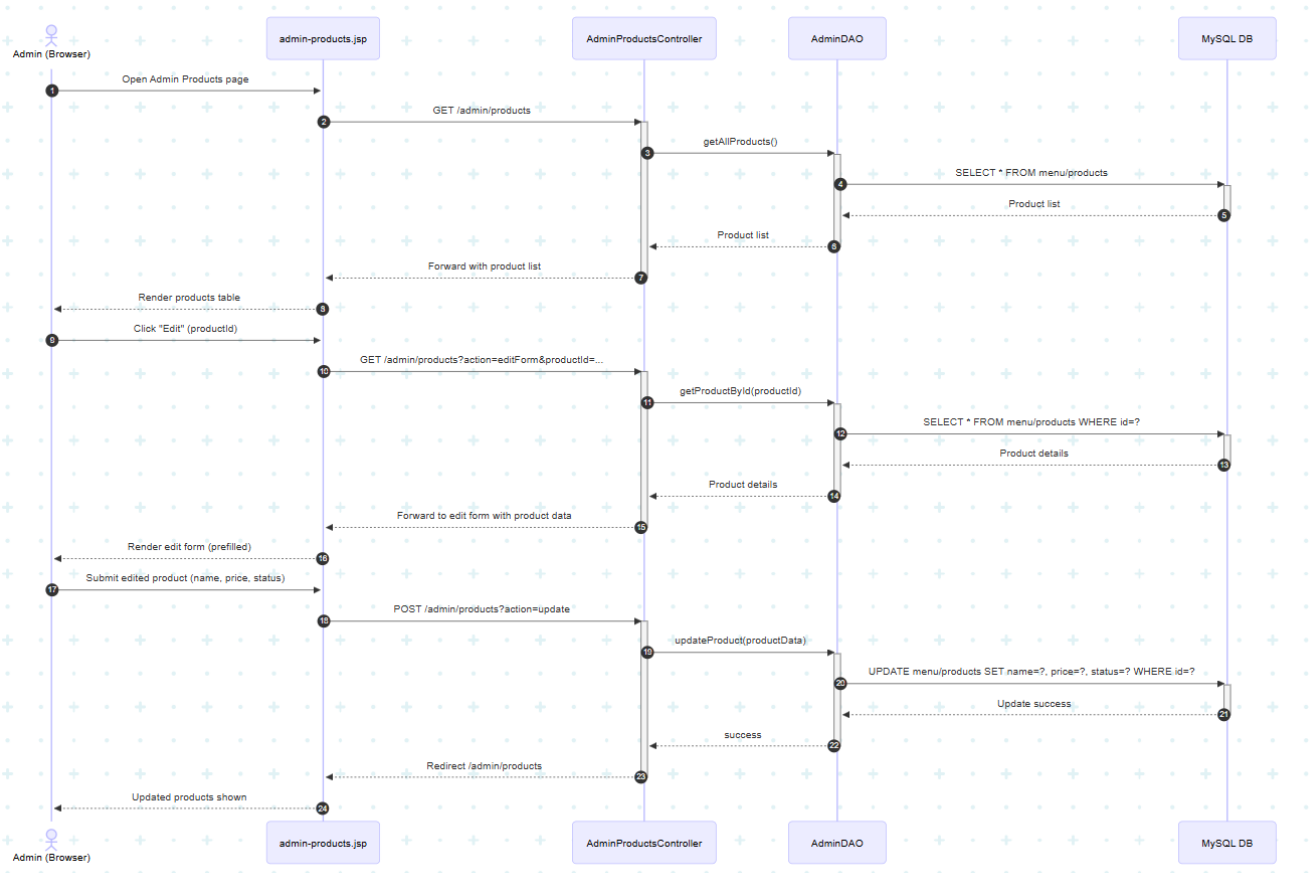


Figure 3.11 Edit products Sequence Diagram

4. System Implemenation

This section describes how the Coffee Management System is implemented based on the system design and database structure discussed earlier. It focuses on the technologies used in development and explains how different system components are implemented for both users and administrators.

The implementation follows a structured approach using the Model–View–Controller (MVC) architecture, which helps separate business logic, data access, and user interface components. This design improves maintainability, scalability, and code readability.

4.1. Technologies and Tools Used

The Coffee Management System is developed using a combination of backend, frontend, and database technologies. Each technology plays a specific role in ensuring the system operates efficiently and reliably.

Backend Technologies

- **Java (Servlets)**
  Used to handle HTTP requests, process business logic, manage sessions, and control application flow.
- **JavaServer Pages (JSP)**
  Used to generate dynamic web content and present data to users through HTML pages.
- **JDBC (Java Database Connectivity)**
  Used to connect the application with the MySQL database and execute SQL queries.

Frontend Technologies

- **HTML & CSS**
  Used to structure and style the web pages, ensuring a clean and user-friendly interface.
- **JavaScript**
  Used for basic client-side interactions such as form validation and dynamic updates.

Database

- **MySQL**
  Used to store all persistent data including users, products, orders, cart items, and shipping information.

Architecture and Design Pattern

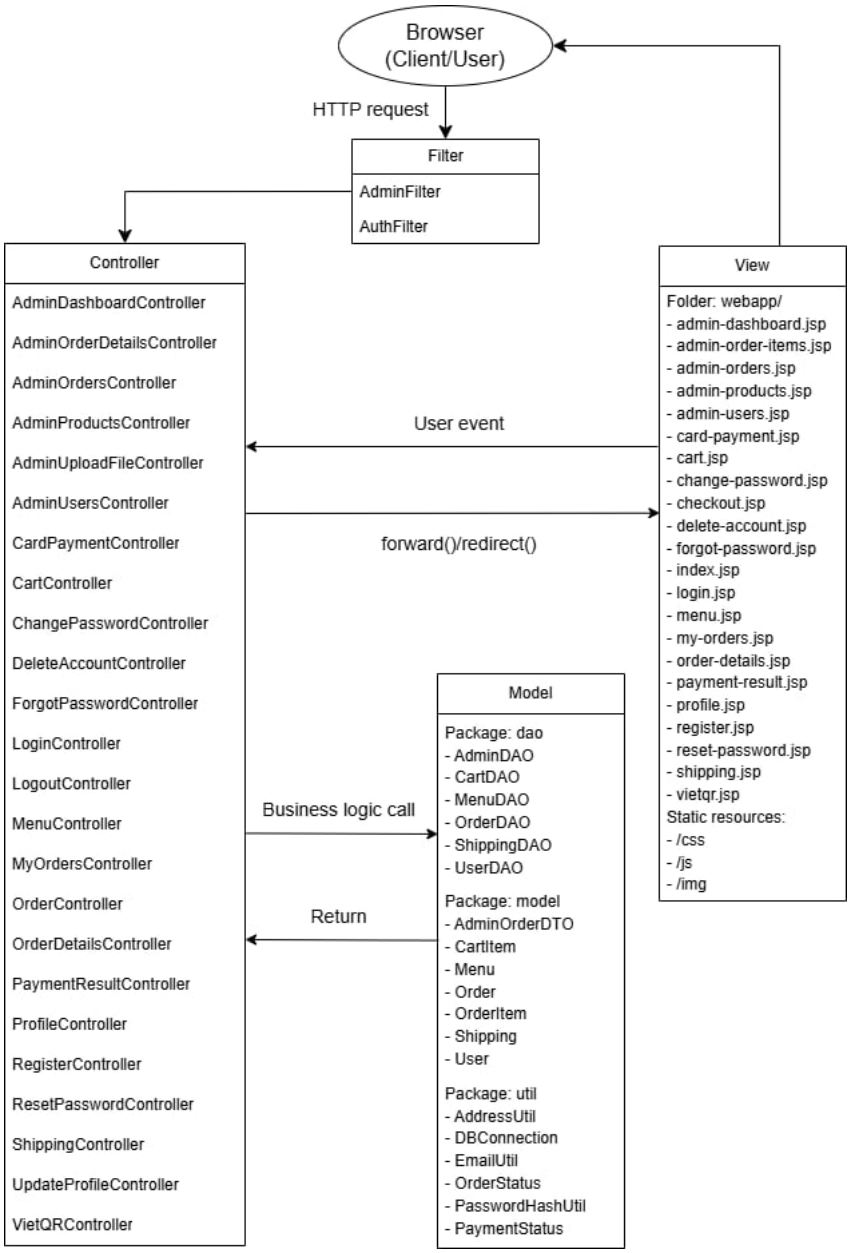


Figure 4.1 System Architecture overview using MVC model

This system follows the **Model–View–Controller (MVC)** architecture to separate responsibilities and improve maintainability.

- **View (JSP pages)**

  The View layer handles the **user interface**. It includes JSP files such as `login.jsp`, `admin-dashboard.jsp`, `cart.jsp`, `profile.jsp`, etc. These pages display data to users and send user actions (requests) to the Controller.

- **Controller (Servlets)**

  The Controller layer receives **HTTP requests** from the browser. It processes user actions such as login, ordering, payment, or profile updates. Controllers (e.g., `LoginController`, `OrderController`, `AdminProductsController`) interact with the Model to execute business logic, then forward or redirect results back to the appropriate View.
- **Model (DAO, DTO, Utility)**

  The Model handles **business logic and data access**.
  - **DAO** classes communicate with the database.
  - **DTO/Entity** classes represent data objects (User, Order, Cart, etc.).
  - **Utils** provide supporting functions (e.g., password hashing, email handling, database connection).
  - **Filters (AuthFilter, AdminFilter)**
  - Filters intercept requests before reaching controllers to handle authentication and authorization.

Flow Summary

User sends request via browser -> Filter checks permissions -> Controller processes logic -> Model handles data operations -> Result is returned to View -> View renders response to user

This structure ensures **clear separation of concerns**, better scalability, and easier maintenance.

Development Tools

- **IDE**: Eclipse
- **Server**: Apache Tomcat
- **Version Control**: Git
- **Database Management**: phpMyAdmin / MySQL Workbench

4.2. Implementation Overview

This section explains how the Coffee Management System is implemented in terms of **functional features**, **request–response handling**, and **code execution flow**. The implementation is divided into three parts: **(1) User-side features**, **(2) Admin-side features**, and **(3) External API features**. Each feature is presented with the corresponding interface screenshot and a step-by-step explanation of how the browser, controllers, DAO layer, utilities, and database work together.

4.3. Guets/Customer Implementation

4.3.1. View the Hompage

This feature allows users to access the homepage both **without logging in** and **after logging in**. The homepage adapts based on session state, where logged-in users can see personalized information (e.g., greeting message or account menu), while guest users see default content.

Figure 4.2. Homepage without Log In (Guest)

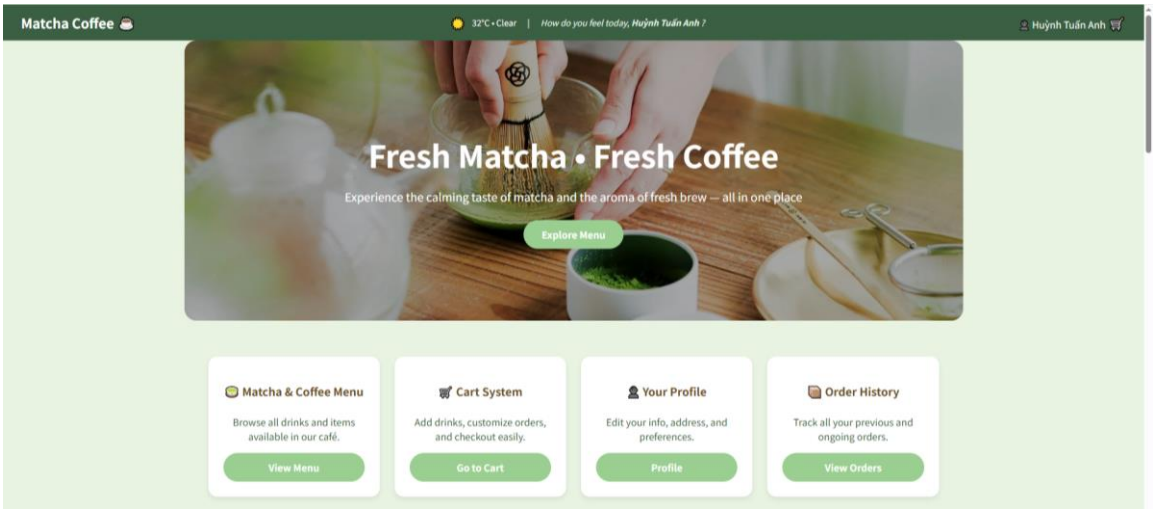


Figure 4.3 Homepage with Log In (customer)

4.3.2. Registration

The registration feature enables new users to create an account by providing personal and contact details. The registration process includes validation, duplicate checking, secure password handling, and database insertion.

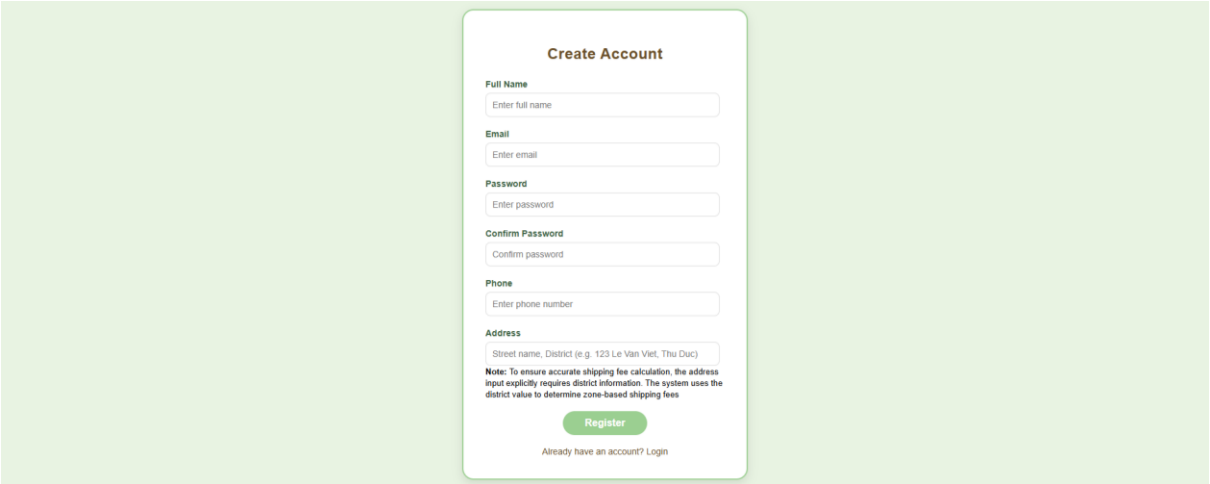


Figure 4.4 Registration Page

Explanation of the Code Flow of Registration

- **Browser → GET /register**
  - The user clicks Create Account or navigates to the register page.
  - A GET request is sent to /register.
- **RegisterController receives GET request**
  - The doGet() method is executed.

- **Controller forwards request to register.jsp**
    - The registration form is displayed.
    - Any previous messages are shown.
- **User fills in registration information**
    - Full name, email, password, confirm password, phone number, address (including district).
- **Browser → POST /register**
    - The user submits the form.
    - A POST request is sent to the server.
- **RegisterController receives POST request**
    - The doPost() method retrieves all parameters.
- **Controller performs basic validation**
    - Required fields are checked.
    - If missing: error message set and forwarded back to register.jsp.
- **Controller validates email format**
    - Email is checked against a regex pattern.
    - If invalid: error message displayed.
- **Controller checks password confirmation**
    - Password and confirm password must match.
    - If not: error message displayed.
- **Controller checks email duplication**
    - Calls UserDAO.getUserByEmail(email).
    - If email exists: error message displayed.
- **Controller hashes the password**
    - Password is encrypted using PasswordHashUtil (bcrypt).
- **Controller creates User object**
    - Includes full name, email, hashed password, phone, address, role, status.
- **Controller calls UserDAO.registerUser(user)**
    - Database insertion is handled in DAO layer.
- **UserDAO executes INSERT**
    - Uses PreparedStatement to prevent SQL injection.
- **DAO returns result**

- True if success, false if failure.

- **Controller handles success**
  - Success message attached.
  - Forward to login.jsp.
- **Controller handles failure**
  - Error message shown, user stays on register.jsp.
- **HTML response returned**
  - Final page rendered and returned to browser.

4.3.3. LogIn (with Authentication Filter)

Login allows users to authenticate using their email/username and password. After successful login, user session is created. Access to protected pages is controlled through authentication filters.

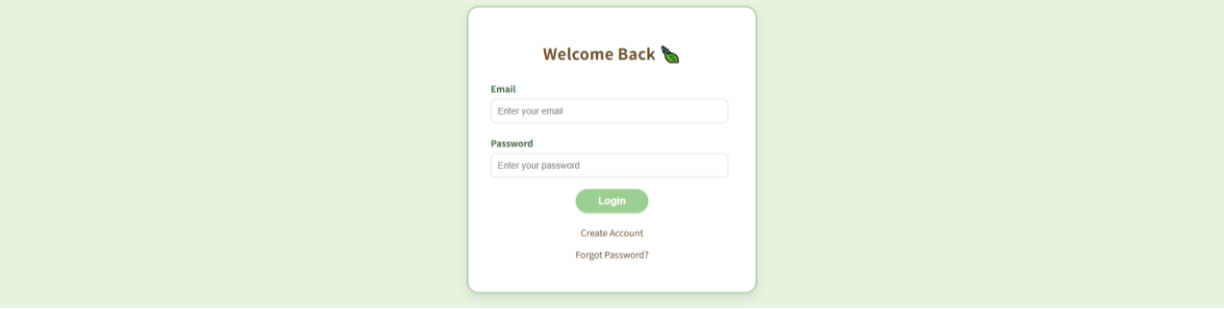


Figure 4.5. Login Page

Explanation of the Flow of Logging In

- **Browser → GET /login**
  - User navigates to login page.
  - Request handled by `LoginController`.
- **LoginController handles GET**
  - `doGet()` executes and forwards to `login.jsp`.
  - JSP displays form and messages using JSTL.
- **Filter checks static resources**
  - CSS/JS/images/fonts bypass authentication via `chain.doFilter()`.
- **Filter checks public URLs**
  - Login/register/forgot-password/homepage are allowed.
- **Filter checks authentication status**

- For protected pages, session is retrieved (without creating new).
- Checks `userId` in session.

- **User not authenticated**
  - Redirect to login page.
- **User authenticated**
  - Request proceeds to the requested servlet/JSP.

4.3.4. View the Menu

This feature displays all available menu products with filtering and sorting capabilities. The system loads product data from the database and renders it dynamically using JSP and JSTL.

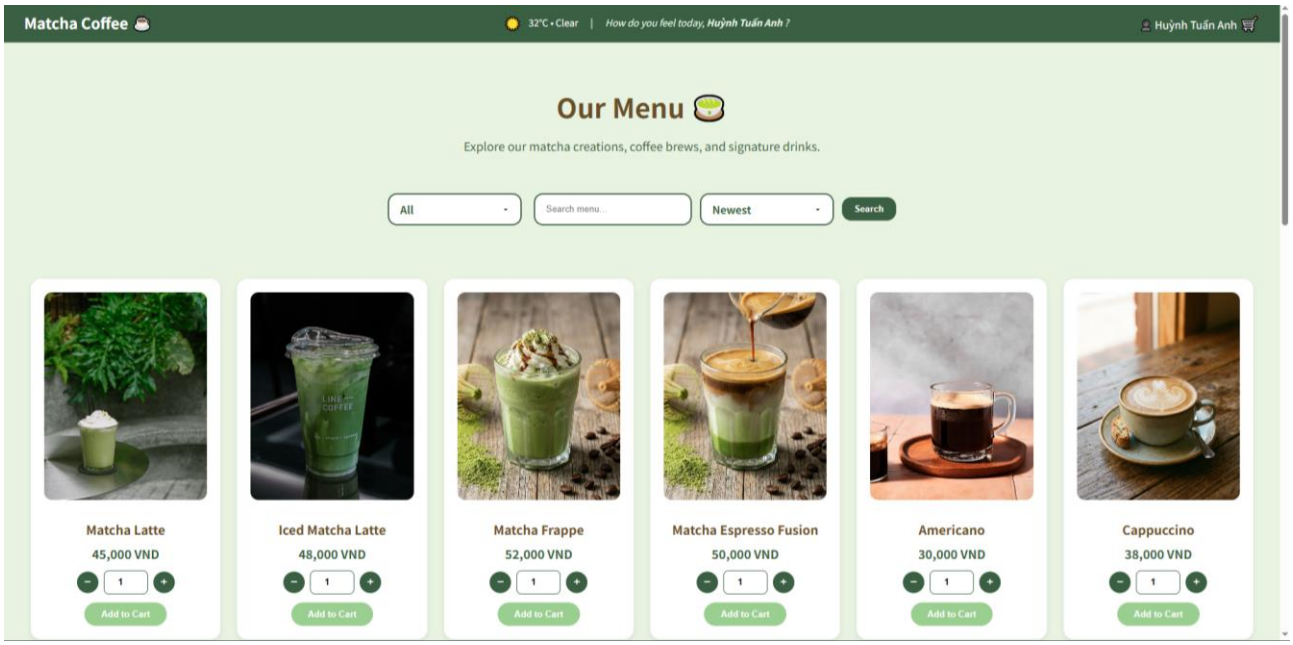


Figure 4.6. Menu Page

Explanation of the Code Flow of Viewing Menu

- **Browser → GET /menu**
  - Optional parameters: category, q, sort.
- **MenuController receives request**
  - Reads category, keyword, sort parameters.
- **Controller calls MenuDAO.getMenu(category, keyword, sort)**
  - Data retrieval is delegated to DAO.
- **MenuDAO builds dynamic SQL**
  - Base query filters only available products.

- Adds category filter, search keyword, sorting rules.

- **DAO executes query**
  - Uses PreparedStatement, binds parameters safely.
- **DAO processes ResultSet**
  - Builds List<Menu> objects.
- **DAO returns List<Menu>**
  - Empty list if no results.
- **Controller sets request attributes**
  - menuList, selectedCategory, keyword, sort.
- **Controller forwards to menu.jsp**
  - Preserves request attributes.
- **JSP renders menu list**
  - Uses `<c:forEach>` to display products.
- **HTML response returned**
  - Final menu page displayed.

4.3.5. Add to Cart from Menu

Users can select product quantity and add items to the cart. Quantity changes on the menu page are handled using JavaScript. The cart update occurs only when the user submits Add to Cart.

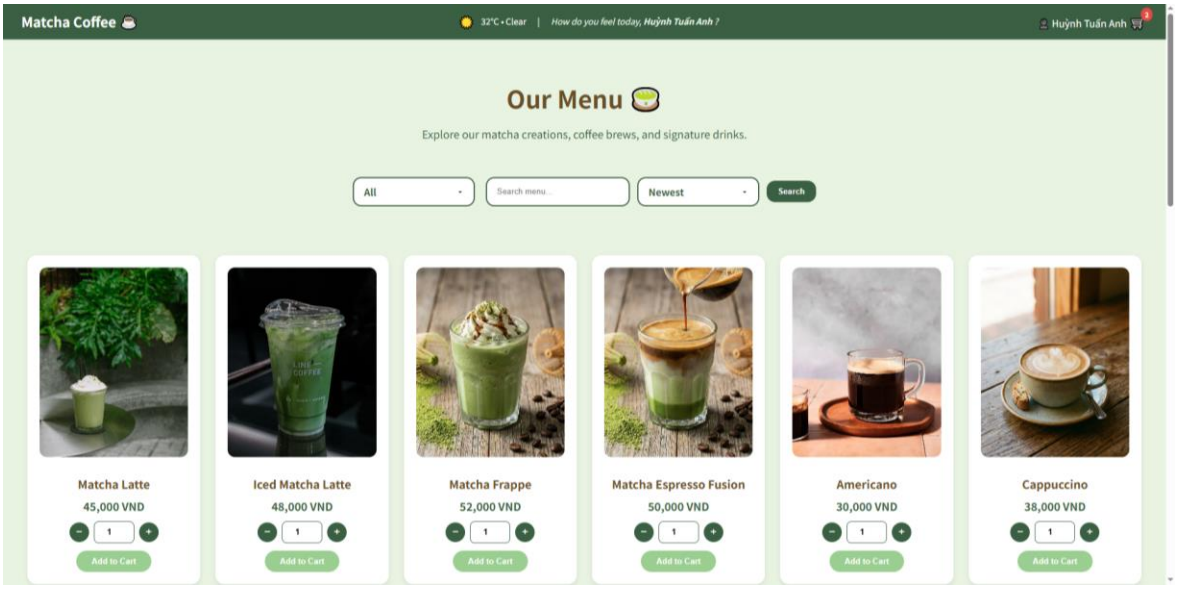


Figure 4.7 The quanity updated when add items to cart

Flow A: Quantity Increase/Decrease on Menu Page (Client-side)

- User clicks "+" or "–"
- JavaScript updates quantity input immediately
- Validates min = 1 and max = 20
- No server request is made until Add to Cart is clicked

Flow B: Add to Cart (Server-side)

- **Browser → GET /cart?op=add&pid=...&qty=...**
- **CartController checks authentication**
  - If not logged in: redirect to login.jsp
- **Controller retrieves product info**
  - `MenuDAO.getMenuById(pid)`
- **Controller calculates final price**
  - Base price + extra price based on milk selection
- **Controller calls CartDAO.addToCart()**
  - DAO checks for identical cart item (including options)
  - Inserts or updates quantity
- **Controller updates cart badge**
  - `CartDAO.getTotalQuantity(userId)` saved in session
- **Redirect back to menu**
  - Cart count badge updated

4.3.6. Modify Quanity and Options in Cart

This feature allows users to modify milk/sugar/ice selections and update cart contents. Options are updated using JavaScript toggle groups and automatically submitted to the server.

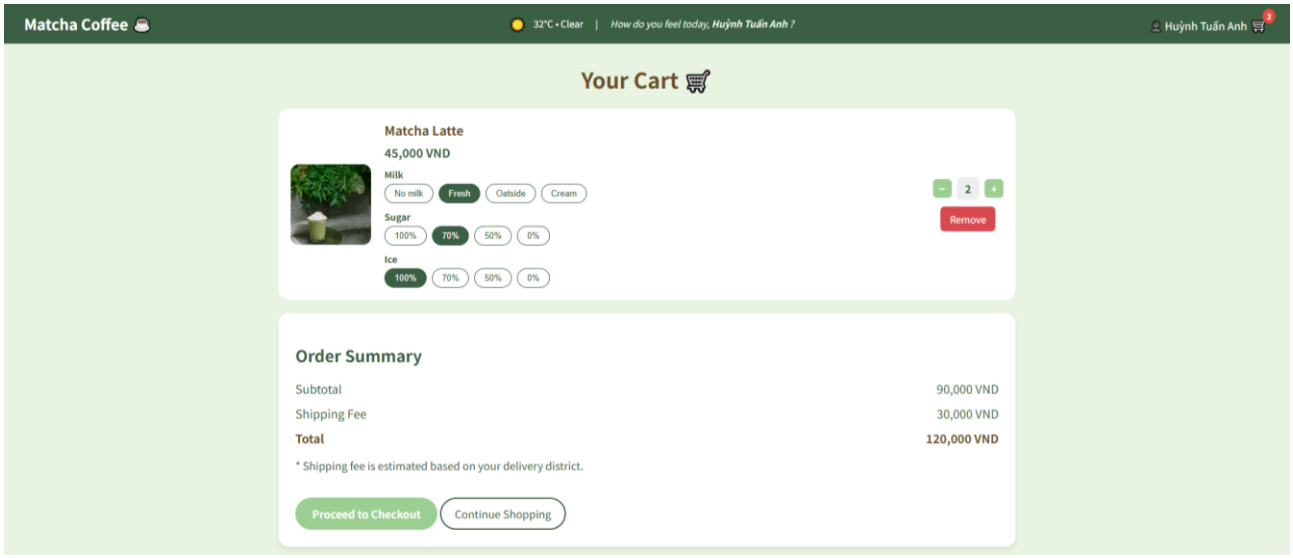


Figure 4.8 Cart View with Options

Explanation of Modifying Options in Cart

- JavaScript detects click on option buttons
- Updates UI active state and hidden inputs
- Automatically submits form after short delay
- **Browser → POST /cart (op=updateOptions)**
- `CartController` receives request
- Calls `CartDAO.updateOptions()`
- DAO recalculates extra price and final price
- DAO updates database via SQL UPDATE
- Controller redirects back to cart
- Cart reloads with updated values

4.3.7. Checkout and Payment Method

The checkout page displays order summary, shipping fee, and available payment options. The controller loads cart items, validates cart state, calculates pricing, and forwards to checkout.jsp.

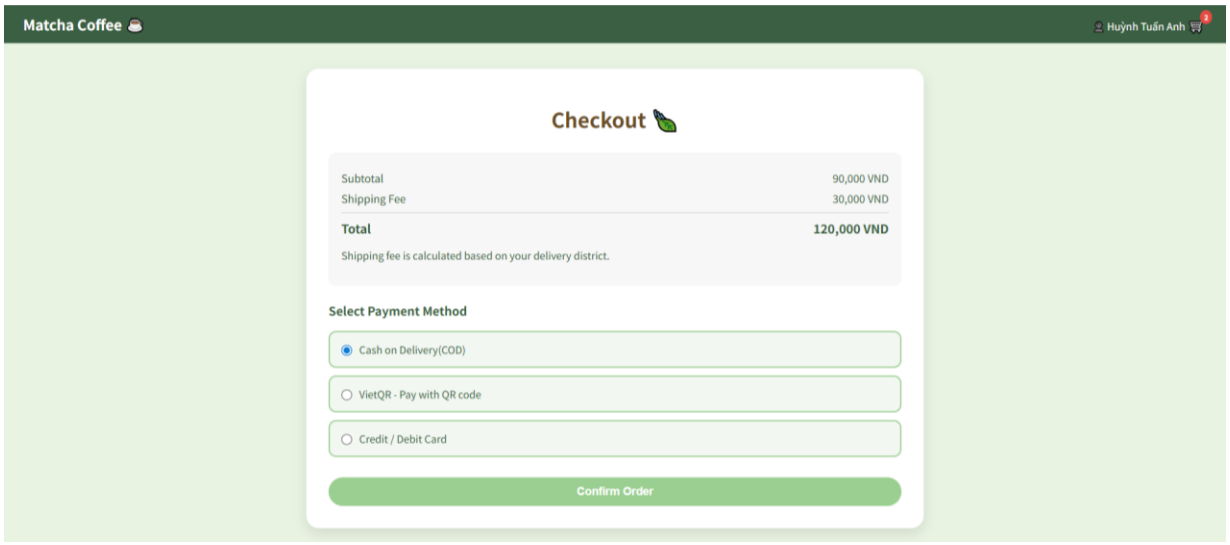


Figure 4.9 Checkout Page

Explanation of Showing Checkout Page

- **Browser → GET /checkout**
- `OrderController.doGet()` retrieves userId from session
- Loads cart via `CartDAO.getCartByUser(userId)`
- If cart empty → redirect back to cart
- Calculates subtotal, total cups
- Calculates shipping via `AddressUtil` + `ShippingService`
- Sets attributes: cart, subtotal, shipping, total, cups
- Forwards to `checkout.jsp`
- JSP renders summary and payment methods

4.3.7.1. Cash On Delivery (COD)

COD creates an order and marks payment as unpaid while confirming the order immediately.

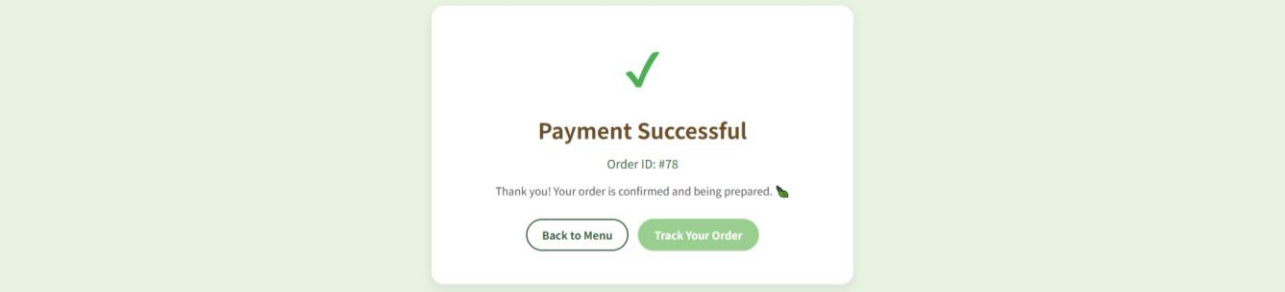


Figure 4.10 Payment-result for COD

4.3.7.2.VietQR Payment

VietQR generates a QR image link, uses countdown timer, and updates payment status based on user action.

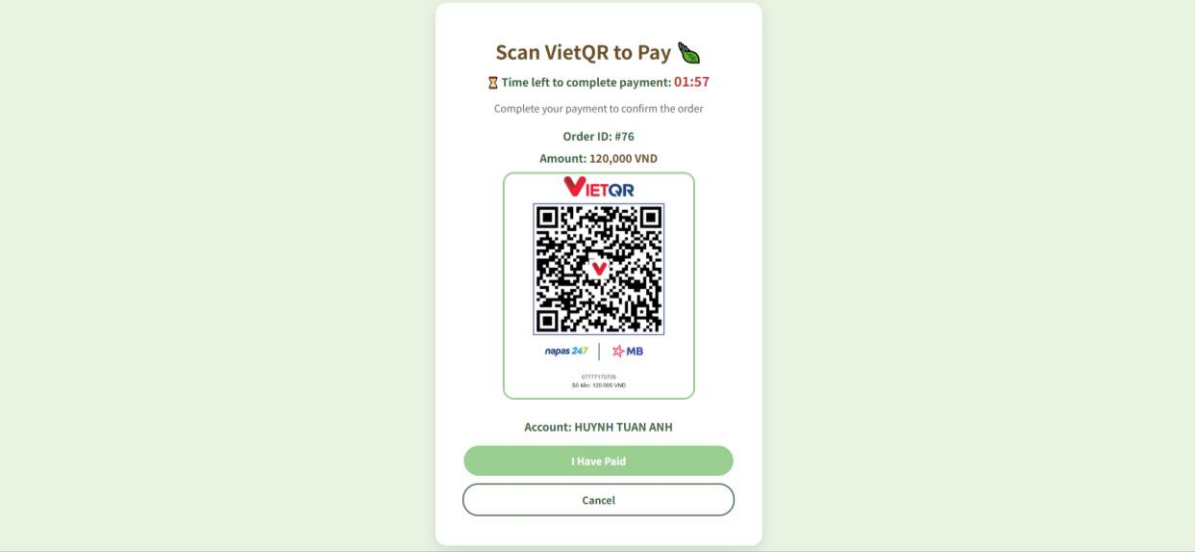


Figure 4.11 VietQR page with countdown

4.3.7.3. Card Payment

Card payment redirects user to card-payment page and updates payment status based on result.

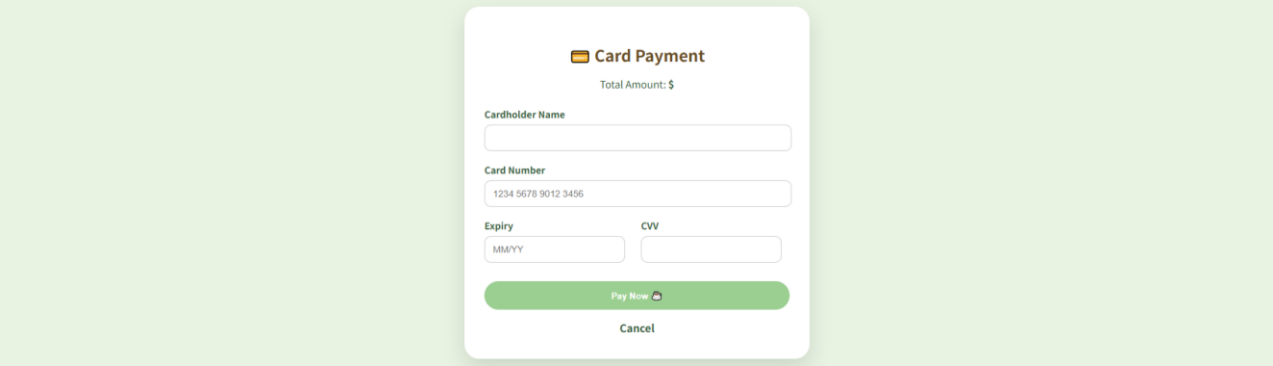
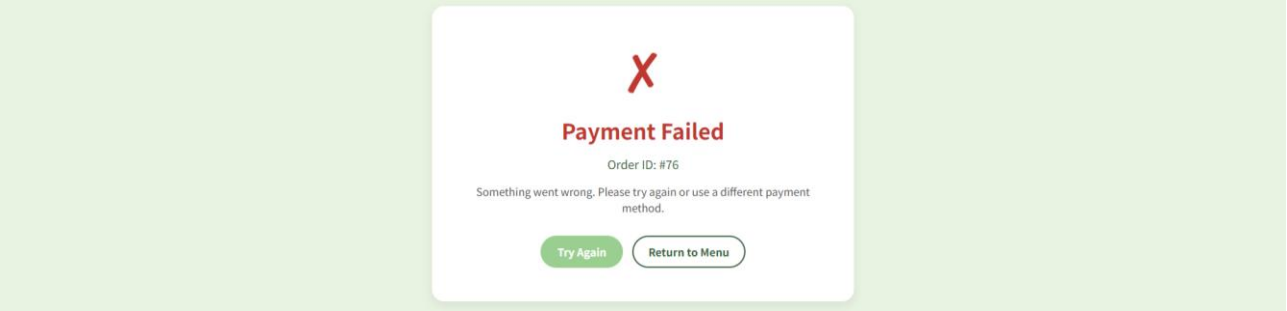


Figure 4.12 Card Payment form

4.3.7.4. Rety Checkout After Failed Payment

Card payment redirects user to card-payment page and updates payment status based on result.



4.3.8. View Orders

Users can view order history and order details.

4.3.8.1.View All History Orders

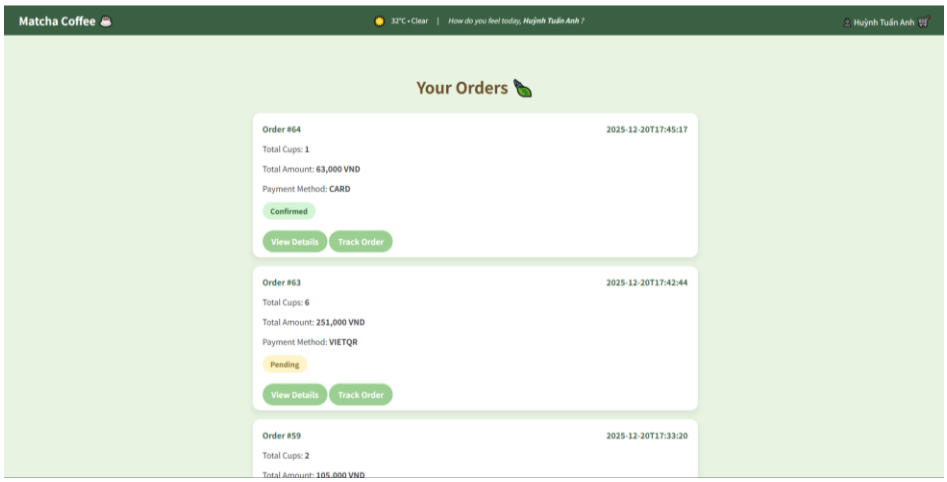


Figure 4.13 View Order History

- **Browser → GET /my-orders**
- `MyOrdersController` reads userId
- Calls `OrderDAO.getOrdersByUser(userId)`
- Controller forwards to `my-orders.jsp`
- JSP renders list with status labels

4.3.8.2. View Order Details

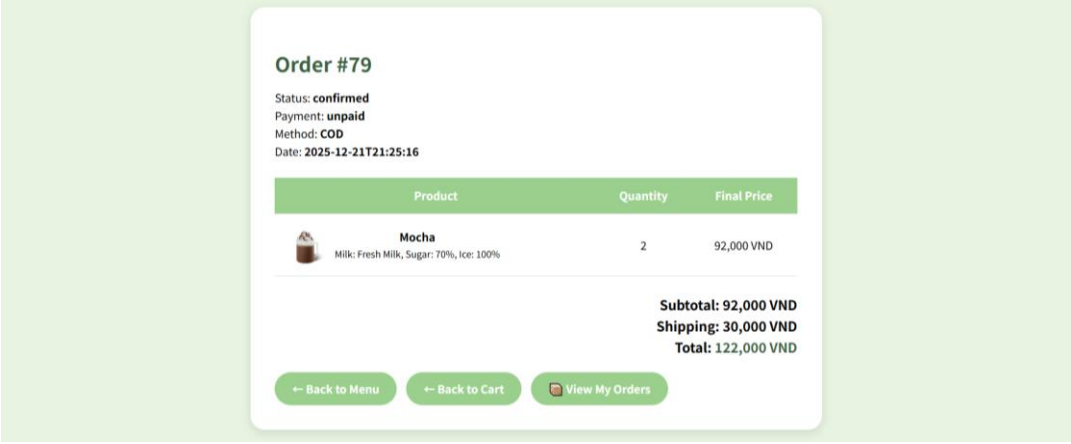


Figure 4.14 View Order Details

- **Browser → GET /order-details?id=...**

- `OrderDetailsController` validates ID
- Loads order summary + order items via DAO
- Forwards to `order-details.jsp`

4.3.9. Track Order for Shipping

Users can track shipping status using `/shipping?orderId=...`.

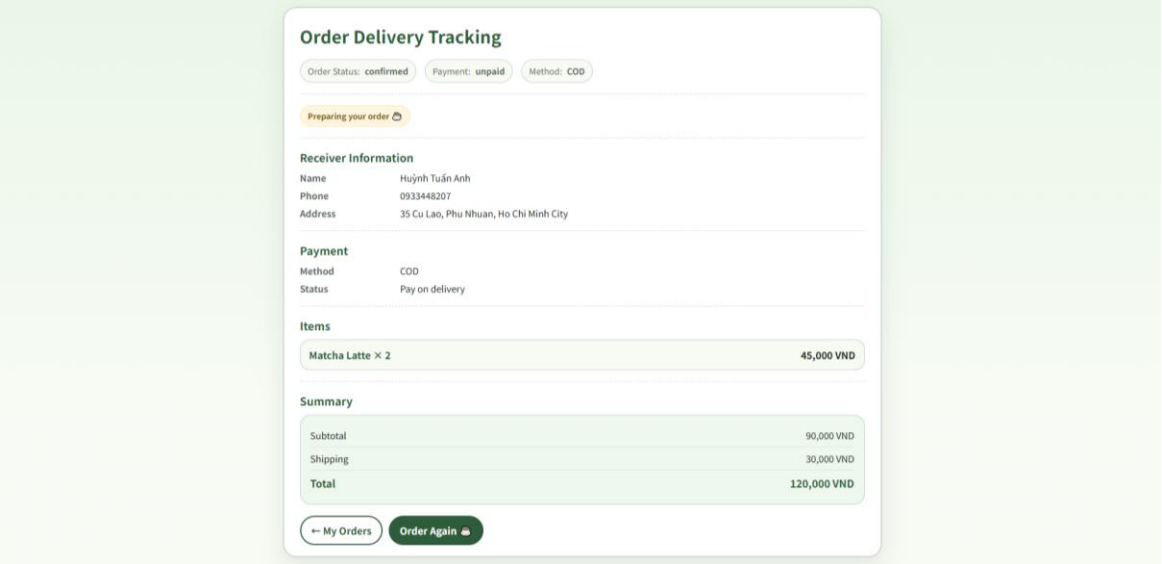


Figure 4.15 Track Orders Page

4.4. Admin Implementation

4.4.1. View Homepage

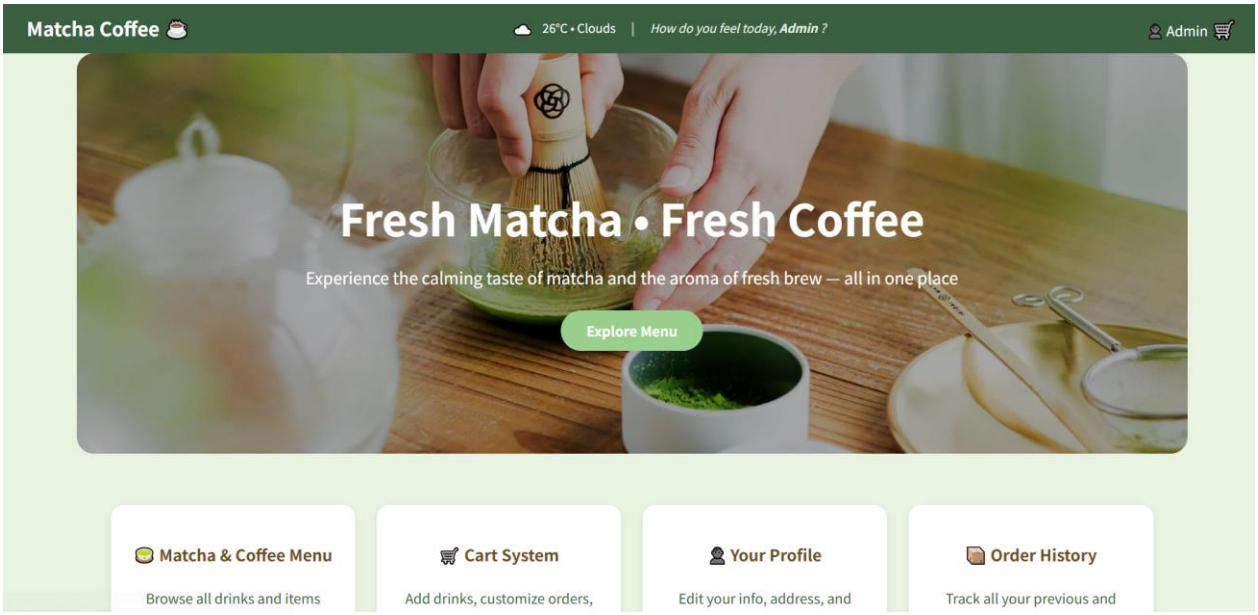
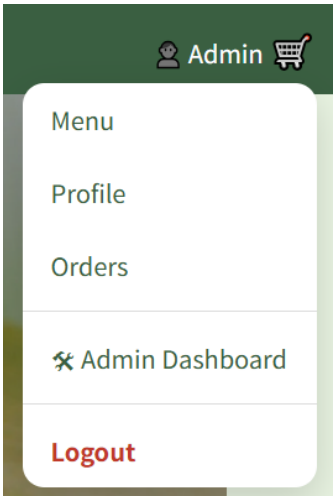


Figure 4.16 View Homepage as Admin



Explanation of the code flow when admin clicks on 'admin dashboard':

- Browser → GET /admin/dashboard
- The admin is currently on the homepage of the website
- The options will drop down when the cursor is moved to the admin icon
- The browser sends a GET request to the dashboard route mapped in web.xml
- Filters run before the controller
- AdminFilter / AdminAuthFilter intercepts the request
- The filter checks:
  - User is logged in (session exists).
  - User role is admin (not customer).
- If not logged in / not admin:
  - Redirect to login.jsp or show access denied.
- If valid admin:
  - The request continues to the controller.

- AdminDashboardController receives request
  - AdminDashboardController.doGet() is executed.
- Controller retrieves dashboard summary data
  - The controller calls DAO methods to load the numbers shown on the dashboard.
  - These DAO methods query tables such as orders, order_items, users, and return totals / lists.
- DAO returns data to controller
  - Controller receives:

    - Summary metrics (counts, totals)
    - Optional lists
- Controller sets request attributes
    - The controller attaches dashboard data to the request.
- Controller forwards to admin-dashboard.jsp
    - Uses RequestDispatcher.forward() to keep request attributes available.
    - Example target: /admin-dashboard.jsp.
- JSP renders the dashboard UI
    - admin-dashboard.jsp reads the request attributes.
    - It displays:
        - KPI cards (totals)
        - Tables/lists (recent orders/users)
        - Any status summaries (if present)
    - If the JSP uses JSTL:
        - <c:if> / <c:choose> for conditional blocks
        - <c:forEach> for lists like recent orders
- HTML response sent to browser
    - The fully rendered Admin Dashboard page is returned.
    - The admin now sees the dashboard data and can navigate to admin-orders.jsp, admin-users.jsp, etc.

4.4.2. View Dashboard

Admin dashboard is protected by admin filters and loads summary information such as totals and recent activities.

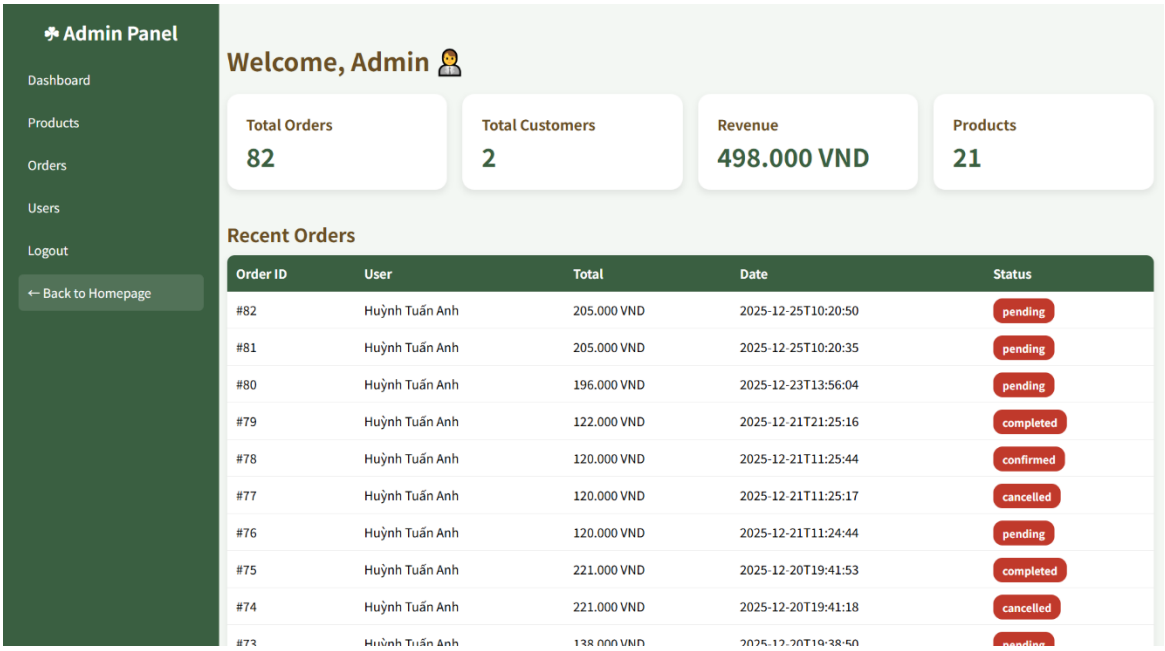


Figure 4.17 Admin Dashboard Page

4.4.3. Modifying products to menu

Admin can manage menu products through `AdminProductsController` and `AdminDAO`.

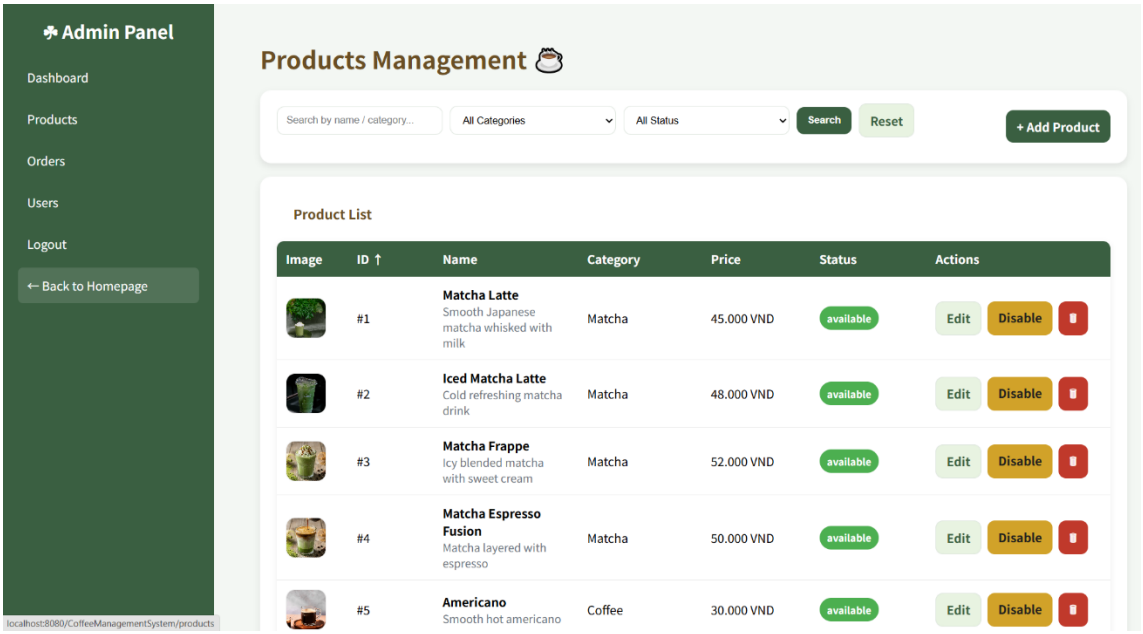


Figure 4.18 Products page

4.3.3.1. Add products to menu

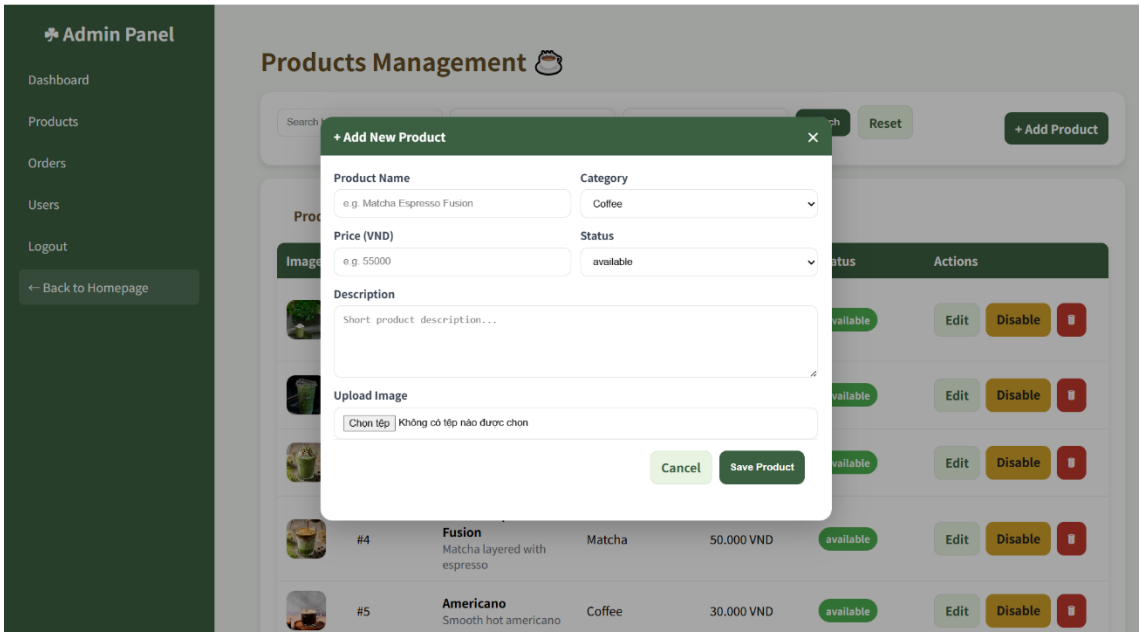


Figure 4.19 Add Products

4.3.3.2. Edit a product

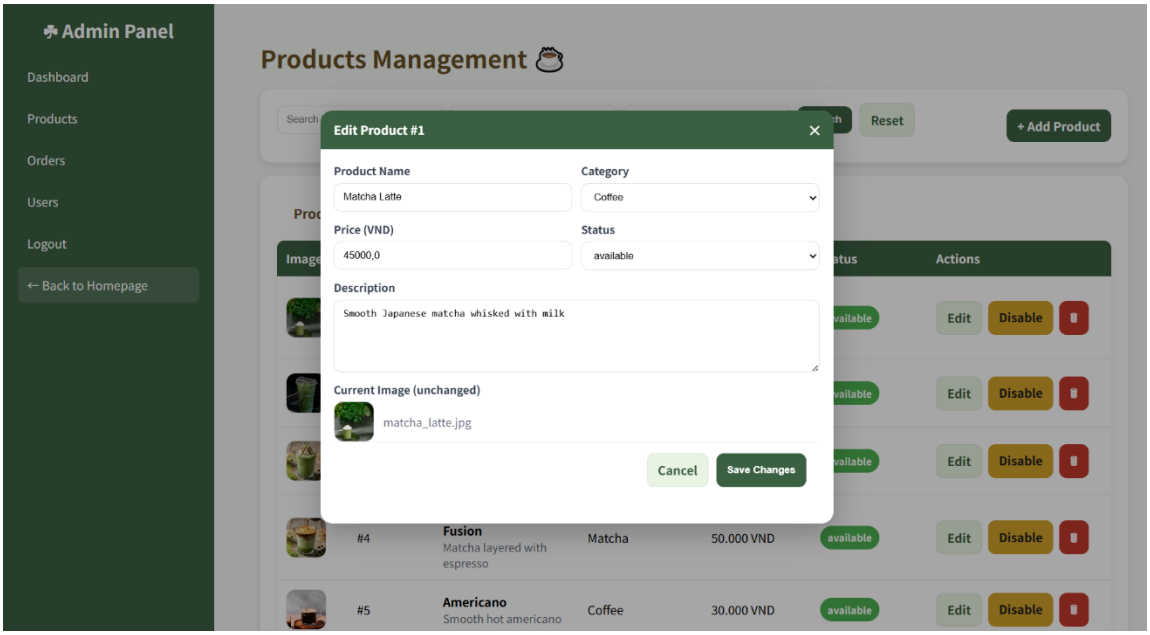


Figure 4.20 Edit a Product

4.4.4. View all Orders

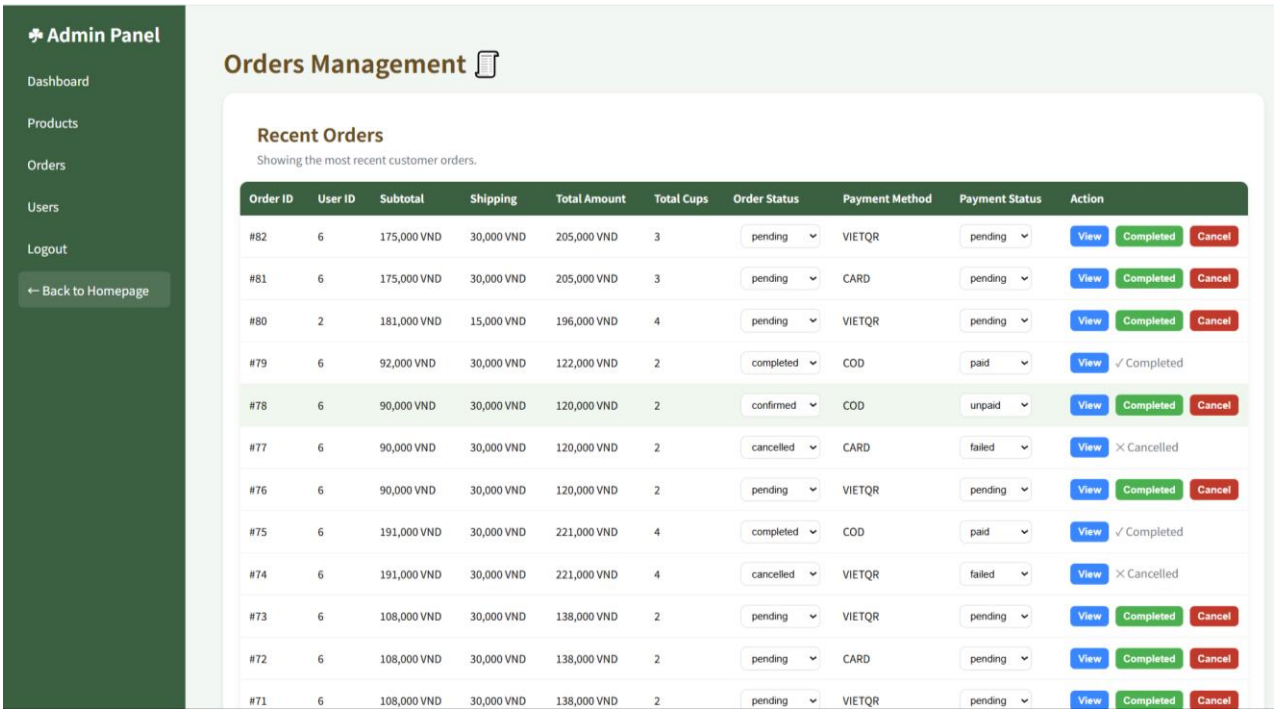


Figure 4.21 View all Orders

4.4.5. *View Users information*

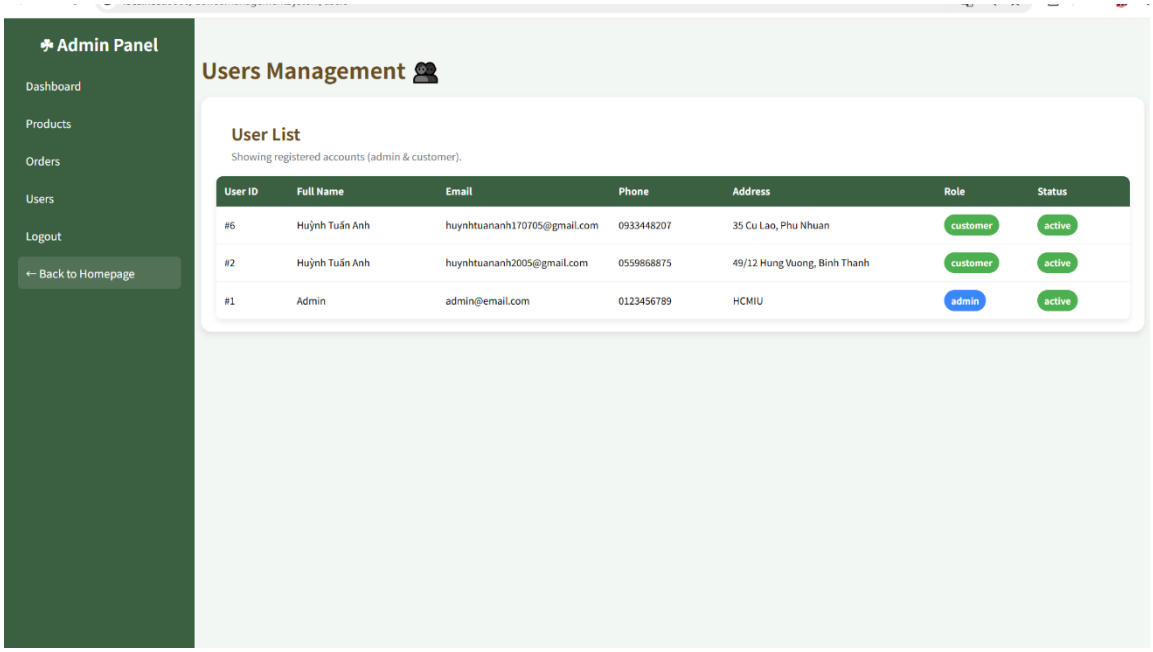


Figure 4.22 View all Users information

4.5. Delete Account

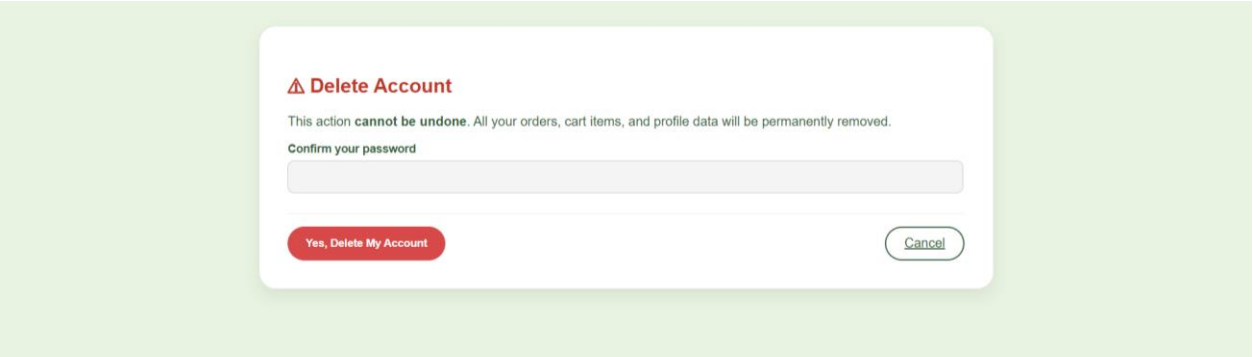


Figure 4.23 Delete Warning Page

4.6. Change password

Whenever the user wants to change their password, or maybe forget their passwords. They can reset the password

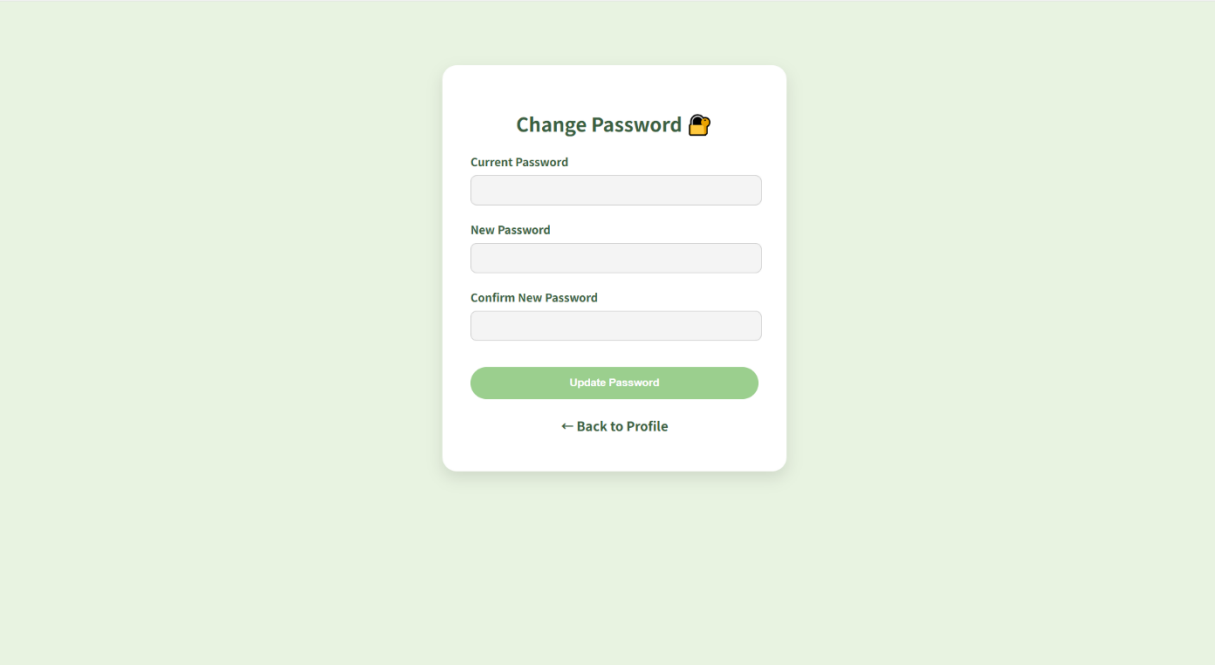


Figure 4.24 Change password page

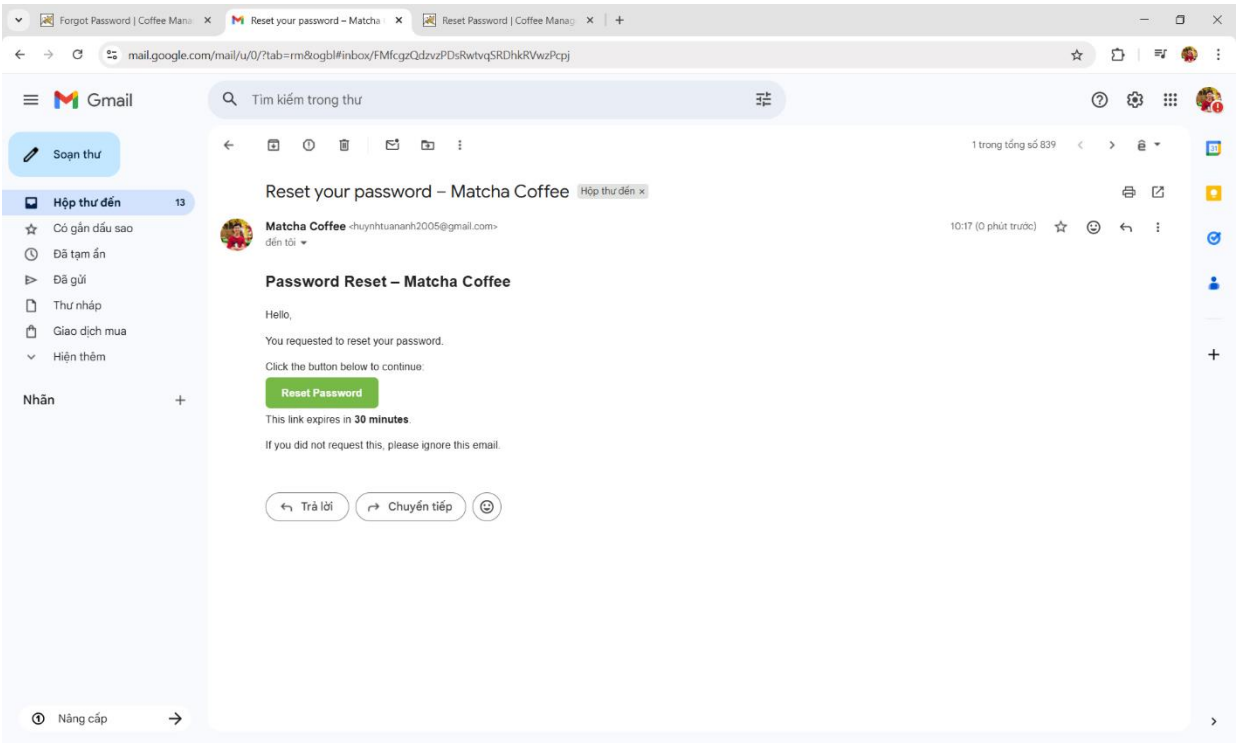


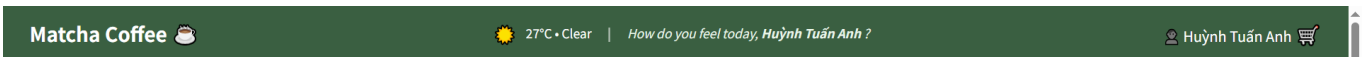
Figure 4.25 Email Sent to reset Password

An email will be sent to their registered email, giving the request to reset their password.

4.7. External API Feature

4.7.1. Display Weather and Quote via OpenWeatherAPI

The navigation bar displays weather data and a greeting message. Weather data is fetched through OpenWeatherAPI and cached in localStorage to reduce API calls.



Explanation of Displaying Weather and Greeting

- Navbar renders default “Loading…” weather box
- JSP checks login status using JSTL to display username or “Guest”
- JavaScript runs after DOMContentLoaded
- Checks localStorage cache (valid for 30 minutes)
- If no cache: calls OpenWeather API for HCMC
- Parses response temperature + condition
- Updates icon + weather text
- If API fails: display “Weather unavailable”

5. Conclusion

Conclusion

This project successfully developed a web-based coffee management system that supports both customers and administrators. The system allows users to browse products, manage their shopping cart, place orders, and manage their accounts, while administrators can manage products, orders, and users through the admin panel. All main features work as intended and meet the project requirements.

The system follows the Model–View–Controller (MVC) structure, which helps separate application logic, database access, and user interface components. Controllers handle requests, DAOs manage database operations, and JSP pages display dynamic content. Security is also considered through the use of filters that restrict access to admin functions based on user roles. Features such as pagination, sorting, and filtering improve usability and system performance.

Overall, this project provides a solid foundation for an online coffee store system. It demonstrates the application of core web development concepts, database interaction, and role-based access control in a practical and structured way.

Future Improvements

Although the system meets the basic requirements, there are several areas that can be improved in the future.

First, system security can be enhanced by adding stronger authentication methods, such as multi-factor authentication, and improving password recovery features. Input validation and error handling can also be improved to make the system more robust.

Second, the user interface can be made more user-friendly by improving the layout and design, adding client-side validation, and making the website more responsive on mobile devices. Using a modern front-end framework could also improve the overall user experience.

Third, more features can be added to increase system functionality, such as sales reports, order statistics, inventory tracking, and email notifications for order updates. Integration with online payment gateways and delivery tracking services would make the system more practical for real-world use.

Forth, the order status and payment status can automatically confirm by payment gateway or third-party application.

Finally, the system can be further developed by using modern frameworks such as Spring Boot and building RESTful APIs. This would improve scalability, make the system easier to maintain, and allow it to integrate with other applications in the future.