

---

**Newbie Coder**

---

**Coffee Base  
Software Architecture Document**

**Version <1.1>**

|                                |                   |
|--------------------------------|-------------------|
| Coffee Base                    | Version: <1.1>    |
| Software Architecture Document | Date: <20/Dec/25> |
| <document identifier>          |                   |

## Revision History

| Date       | Version | Description                   | Author              |
|------------|---------|-------------------------------|---------------------|
| 20/Dec/25  | <1.1>   | Deployment and Implement View | Quang               |
| 05/Dec/25> | <1.0>   | Initial Document              | Cường, Quang, Thanh |

|                                |                   |
|--------------------------------|-------------------|
| Coffee Base                    | Version: <1.1>    |
| Software Architecture Document | Date: <20/Dec/25> |
| <document identifier>          |                   |

## Table of Contents

|             |                         |      |
|-------------|-------------------------|------|
| 1.          | Introduction            | 42.  |
| Constraints | Architectural Goals and | 43.  |
| Model       | Use-Case                | 54.  |
| View        | Logical                 | 64.1 |
| abc         | Component:              | 65.  |
| View        | Deployment              | 166. |
|             | Implementation          | 17   |

|                                |                   |
|--------------------------------|-------------------|
| Coffee Base                    | Version: <1.1>    |
| Software Architecture Document | Date: <20/Dec/25> |
| <document identifier>          |                   |

# Software Architecture Document

## 1. Introduction

### 1.1 Purpose

- This Software Architecture Document (SAD) provides a concise and comprehensive overview of Coffee Base system's architecture, serving as a reference for stakeholders. In this document, important information regarding architectural decisions, structure and behaviour will be described. The document will also guide future communication, implementation and maintenance.

### 1.2 Scope

- The scope of this document covers the high-level system overview, architectural views, interactions between components, selected technologies, and major non-functional requirements.

### 1.3 Overview

- This SAD follow this structure and is divided into sections:
  - Section 1 - Introduction
  - Section 2 - Architecture Goals and Constraints
  - Section 3 - Use-case Model
  - Section 4 - Logical View
  - Section 5 - Deployment
  - Section 6 - Implementation View

## 2. Architectural Goals and Constraints

### 2.1 Architectural goals

- Performance goals
  - Responding time under 530ms under medium load.
  - No performance degradation for 100 concurrent users
  - New database updates take under 1 second
- Security goals
  - Enforce password rules: At least 8 digits, must contain number, lowercase letter, uppercase letter, and one special character.
  - Implement secure authorization using OAuth2, JWT or equivalent securing measures.
  - Must implement token refresh and token expiration mechanisms.

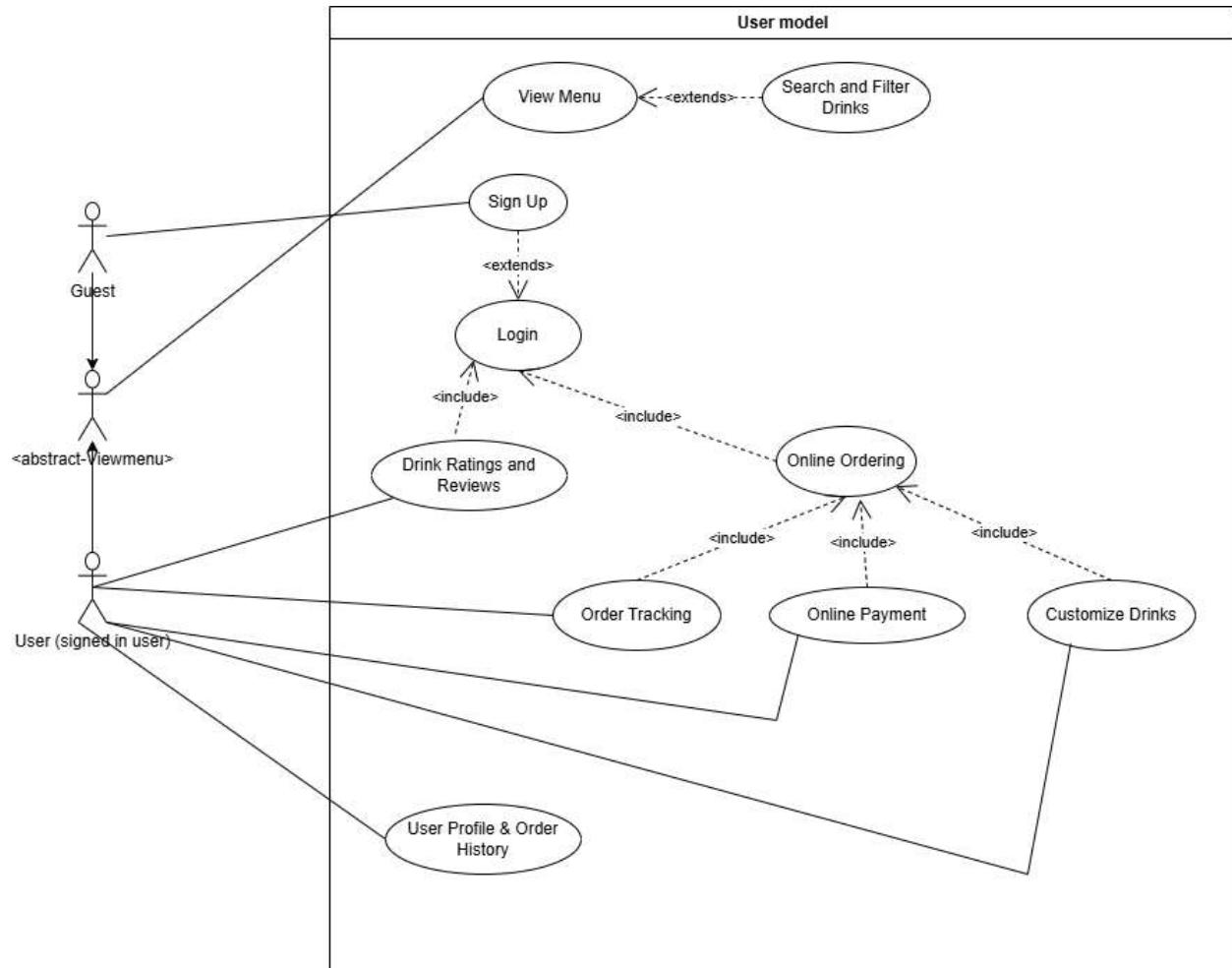
### 2.2 Constraints

- Technical constraints
  - Compatibility with modern web browsers
  - Support screen size for both small phone and large desktop.
  - Allow dynamic update for menu items and advertising banner
- Development tools constraints
  - Use free tools for communication and managing (Messenger, Discord, Jira, etc)
  - Documents be stored and edited collectively on Google Drive.
  - Must adopt a version control system (Git, Github)
  - Source code can be deployed and be tested on a local machine before pushing to web.
- Security constraints
  - All authentication must use secure protocols (HTTPS, JWT, OAuth2, etc.) and no storage of plain-text passwords.
  - No development tool should expose sensitive data .

|                                |                   |
|--------------------------------|-------------------|
| Coffee Base                    | Version: <1.1>    |
| Software Architecture Document | Date: <20/Dec/25> |
| <document identifier>          |                   |

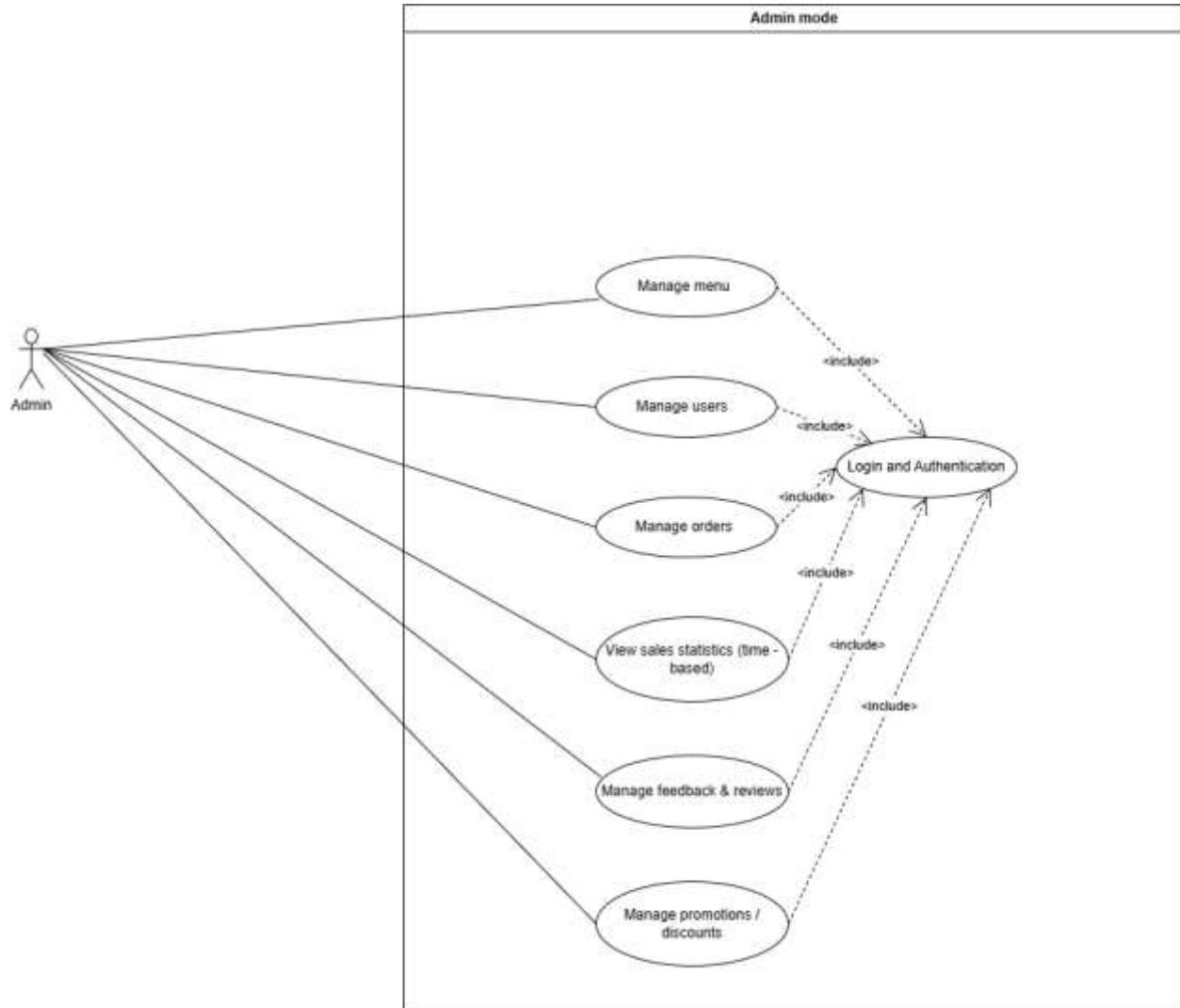
### 3. Use-Case Model

#### 3.1 User Model



|                                |                   |
|--------------------------------|-------------------|
| Coffee Base                    | Version: <1.1>    |
| Software Architecture Document | Date: <20/Dec/25> |
| <document identifier>          |                   |

### 3.2 Admin Model



## 4. Logical View

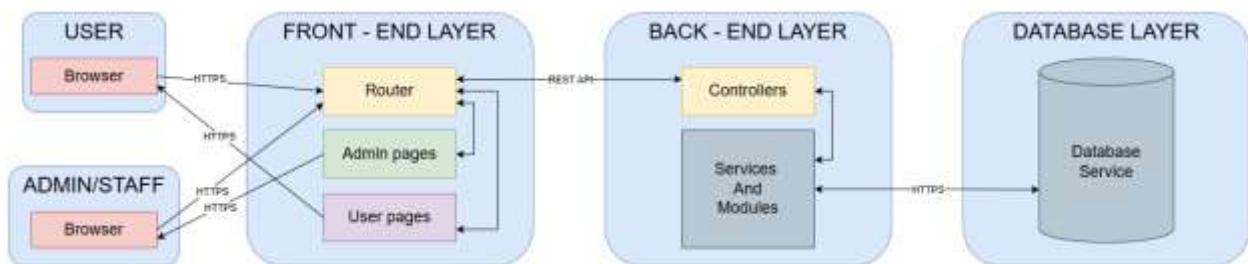
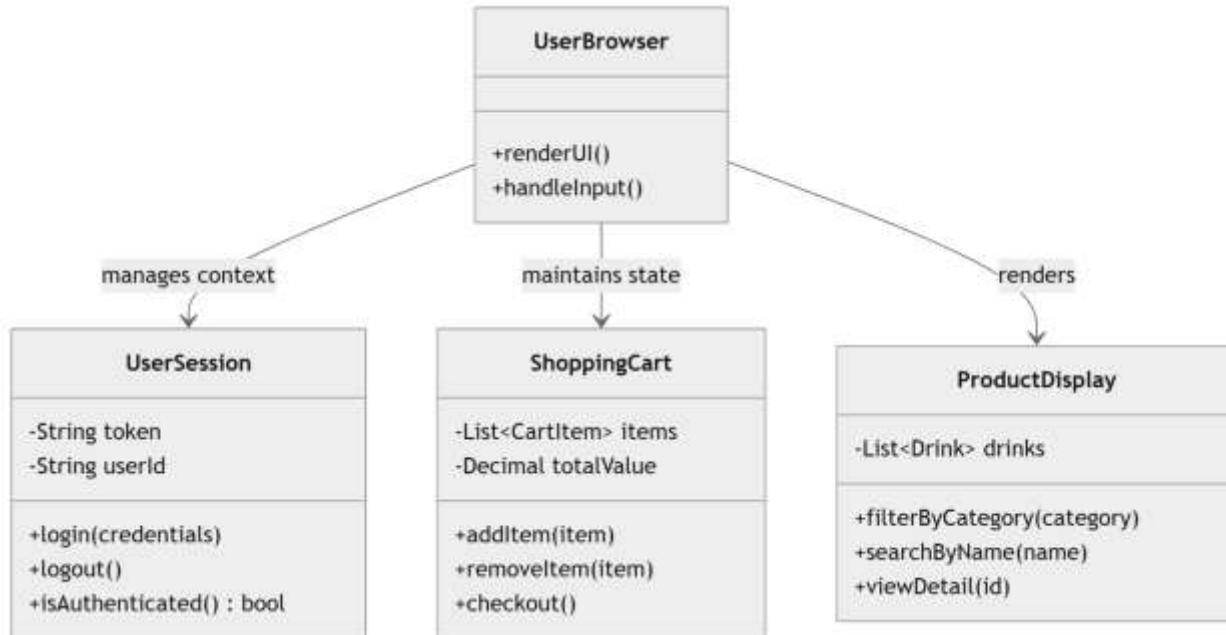


Figure 1: Layer model

### 4.1 Component: User Browser

|                                |                   |
|--------------------------------|-------------------|
| Coffee Base                    | Version: <1.1>    |
| Software Architecture Document | Date: <20/Dec/25> |
| <document identifier>          |                   |

This component represents the client-side runtime environment (web browser) used by customers (Guests and Signed-in Users) to interact with the Coffee Base application.



### Description of Classes:

- The **UserBrowser** acts as the main container for the customer's interaction, handling the rendering of HTML/CSS and capturing user events.
- The **ShoppingCart** class manages the local state of selected items before the order is finalized, ensuring data persists during the session.
- **UserSession** handles authentication states, storing the JWT or session tokens required for securing personal data like order history.
- **ProductDisplay** is responsible for presenting the menu and handling client-side filtering and searching logic.

### Services and Responsibilities:

- **UI Rendering:** Displays the menu, product details, and shopping cart to the user.
- **State Management:** Maintains temporary data such as cart contents and filter preferences locally.
- **Input Validation:** Validates forms (e.g., Sign Up, Login) before submission to reduce server load.
- **Session Handling:** Manages user login states and secure token storage.

### Communication Interfaces:

- **Protocol:** HTTPS.
- **Communication:** Synchronous request/response.
- **Target:** Interactions are directed to the **Router** component in the Front-End Layer.
- **Payload:**
  - + **Requests:** JSON objects containing user actions (e.g., `{ "action": "addToCart", "itemId": 101, "qty": 2 }`).
  - + **Responses:** HTML content or JSON data streams representing UI updates.

|                                |                   |
|--------------------------------|-------------------|
| Coffee Base                    | Version: <1.1>    |
| Software Architecture Document | Date: <20/Dec/25> |
| <document identifier>          |                   |

#### Non-functional Concerns:

- The User Browser component focuses heavily on **Usability** and **Compatibility**, ensuring the interface is responsive for both mobile phones and desktops as per project constraints.
- **Security** is critical; the component must ensure that session tokens are stored securely (e.g., HttpOnly cookies or secure local storage) to prevent XSS attacks.
- **Performance** is addressed by lazy-loading images to ensure main pages load within the 2–4 second target

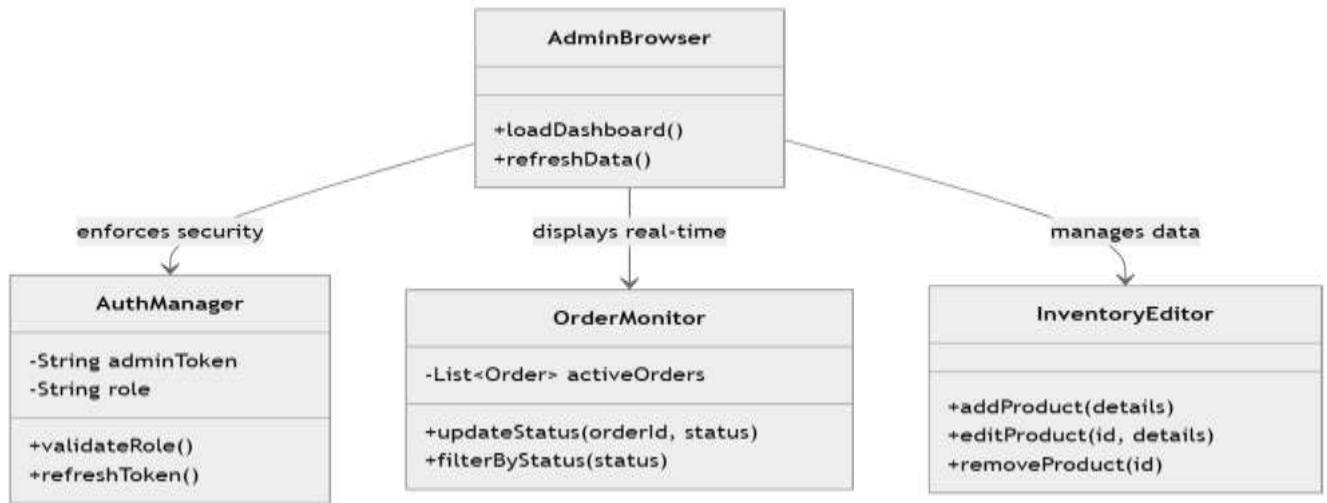
#### Trace to Use-cases:

- **UCU01 View Drink Menu:** The component renders the list of drinks and handles category filtering.
- **UCU06 Online Ordering:** The component manages the **ShoppingCart** state and captures user delivery inputs.

## 4.2 Component: Admin Browser

This component represents the client-side runtime environment used by Administrators and Staff to access the management dashboard and perform operational tasks.

|                                |                   |
|--------------------------------|-------------------|
| Coffee Base                    | Version: <1.1>    |
| Software Architecture Document | Date: <20/Dec/25> |
| <document identifier>          |                   |



### Description of Classes:

- The **AdminBrowser** serves as the shell for the Single Page Application (SPA) used by staff.
- **AuthManager** is strictly implemented to enforce role-based access control, ensuring only authorized personnel can access sensitive modules.
- **OrderMonitor** provides a view of incoming orders, allowing admins to change status from "Processing" to "Done".
- **InventoryEditor** provides the forms and validation logic required to create or modify menu items.

### Services and Responsibilities:

- **Dashboard Visualization:** Renders charts and tables for sales statistics and order tracking.
- **CRUD Operations:** Provides interfaces for Creating, Reading, Updating, and Deleting menu items and user accounts.
- **Real-time Updates:** Polls or listens for status changes to keep the order list current.
- **Access Control:** Restricts UI elements based on the specific admin role (e.g., Super Admin vs. Staff).

|                                |                   |
|--------------------------------|-------------------|
| Coffee Base                    | Version: <1.1>    |
| Software Architecture Document | Date: <20/Dec/25> |
| <document identifier>          |                   |

### Communication Interfaces:

- **Protocol:** HTTPS.
- **Communication:** Synchronous (REST calls) and potentially Asynchronous (for background status updates).
- **Target:** Communicates primarily with the **Router** component.
- **Payload:**
  - + **Data Format:** JSON (e.g., { "orderId": 1054, "status": "Done" }).
  - + **Security Header:** Authorization Bearer Token (JWT).

### Non-functional Concerns:

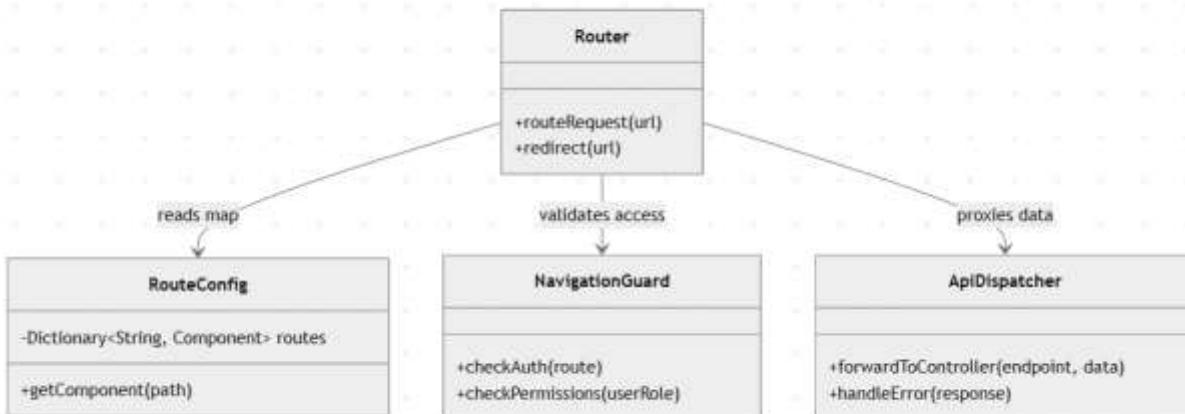
- **Security** is the primary concern; the component must auto-expire sessions after inactivity to prevent unauthorized access.
- **Data Integrity** is ensured by validating all inputs (e.g., price must be numeric) before sending data to the backend.
- **Maintainability** is supported by using modular UI components that can be updated independently.

### Trace to Use-cases:

- **UCA4 Manage Orders:** The component provides the interface for viewing order lists and updating their status manually.
- **UCA2 Manage Menu:** The component renders the forms for adding or editing product details like price and description.

### 4.3 Component: Router

The Router acts as the central navigation hub within the Front-End Layer, intercepting browser requests and dispatching them to the appropriate Page components or API controllers.



### Description of Classes:

- The **Router** is the entry point for the front-end application logic.
- It uses **RouteConfig** to map URLs (e.g., `/menu`, `/admin/orders`) to specific view components.
- The **NavigationGuard** implements the logic to check if a user is logged in or has the correct admin privileges before loading a protected route.
- The **ApiDispatcher** manages the transmission of data between the UI components and the Back-End Controllers via REST API.

### Services and Responsibilities:

|                                |                   |
|--------------------------------|-------------------|
| Coffee Base                    | Version: <1.1>    |
| Software Architecture Document | Date: <20/Dec/25> |
| <document identifier>          |                   |

- **Navigation Management:** Handles client-side routing, loading the correct page without a full browser refresh.
- **Request Dispatching:** Acts as the intermediary between the Browser components and the Back-End Controllers.
- **Authorization Enforcement:** Intercepts navigation to protected pages (e.g., Admin Dashboard) and redirects unauthorized users to the login page.
- **Error Handling:** Manages 404 (Not Found) or 500 (Server Error) states globally.

#### Communication Interfaces:

- **Protocol:** HTTPS.
- **Inbound:** Receives requests from **User Browser** and **Admin Browser**.
- **Outbound:** Sends **REST API** requests to **Controllers** in the Back-End Layer.
- **Routing Logic:**
  - o `/admin/*` → Routes to **Admin Pages**.
  - o `/user/*` or `/` → Routes to **User Pages**.
  - o `/api/*` → Proxies to **Back-End Controllers**.

#### Non-functional Concerns:

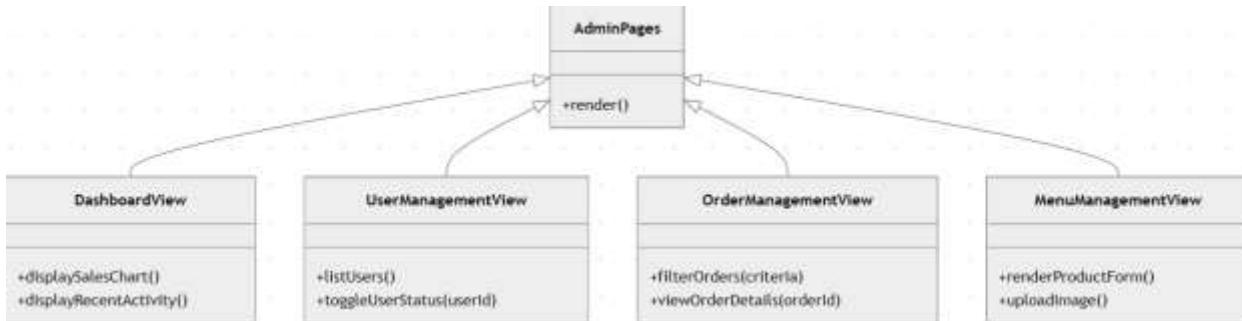
- **Performance** is optimized by supporting code-splitting, ensuring that the Admin Page logic is not loaded for standard users, reducing the initial bundle size.
- **Scalability** is handled by the stateless nature of the router, allowing it to serve multiple concurrent browser instances efficiently.
- **Logging** is implemented to track navigation errors or failed API calls for debugging.

#### Trace to Use-cases:

- **UCU03 Login:** The Router redirects the user to the "Home" page or "Dashboard" upon successful authentication.
- **UCA1 Admin Login:** The Router prevents access to admin routes until the **NavigationGuard** verifies the admin's credentials.

#### 4.4 Component: Admin Pages

This component represents the collection of specific UI modules and views designed for administrative tasks, which are rendered dynamically by the Router.



|                                |                   |
|--------------------------------|-------------------|
| Coffee Base                    | Version: <1.1>    |
| Software Architecture Document | Date: <20/Dec/25> |
| <document identifier>          |                   |

### Description of Classes:

- **AdminPages** is a generalized container for specific views.
- **DashboardView** integrates with charting libraries to display sales statistics visually.
- **UserManagementView** allows admins to view user lists and perform actions like banning accounts.
- **OrderManagementView** presents the operational data required to fulfill customer orders.
- **MenuManagementView** handles the complex UI for product creation, including image uploading and categorization.

### Services and Responsibilities:

- **Data Presentation:** Formats raw JSON data from the backend into human-readable tables and graphs.
- **Form Management:** Provides structured forms for data entry (e.g., adding a new coffee drink with toppings).
- **Interactive Controls:** Offers buttons and toggles for operational actions (e.g., "Ban User", "Approve Review").
- **Feedback Display:** Shows success or error messages to the admin after an action is performed.

### Communication Interfaces:

- **Protocol:** Internal Component Call / Event Binding.
- **Interaction:** These pages are loaded and managed by the **Router** component.
- **Data Flow:** They emit events or invoke methods on the Router/Dispatcher to fetch or save data.
- **Format:** Accepts Props/State objects containing lists of **Users**, **Orders**, or **Products**.

### Non-functional Concerns:

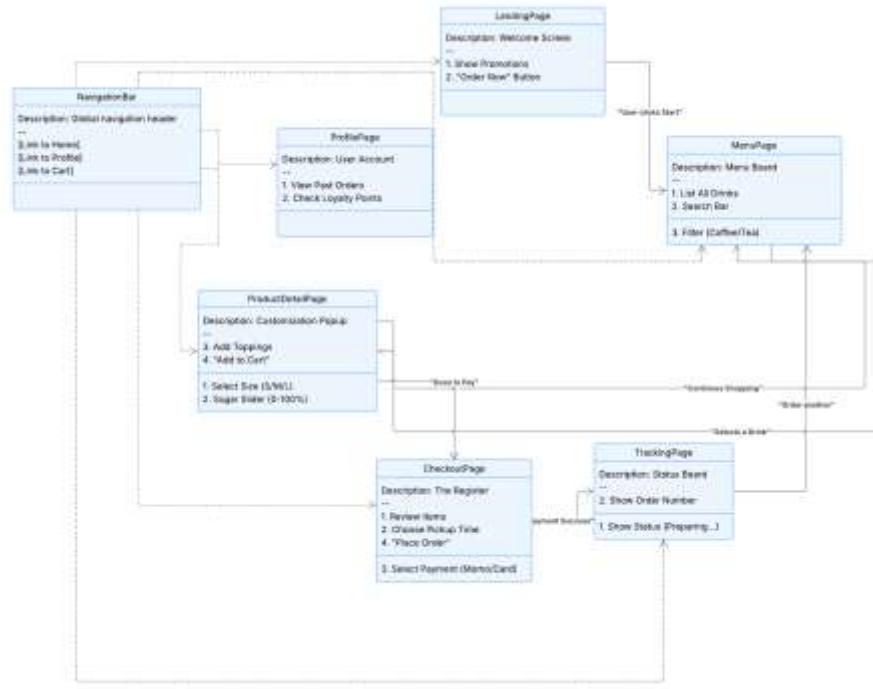
- **Usability** is key for this component to ensure efficient workflows for staff; forms must be intuitive to minimize errors.
- **Data Integrity** is supported by client-side validation logic (e.g., ensuring end-date is after start-date for promotions).
- **Privacy** is respected by masking sensitive user data (like passwords) within the User Management views.

### Trace to Use-cases:

- **UCA3 Manage Users:** The **UserManagementView** displays the list of accounts and their credit points.
- **UCA5 View Sales Statistics:** The **DashboardView** renders the visual charts for income over time.

|                                |                   |
|--------------------------------|-------------------|
| Coffee Base                    | Version: <1.1>    |
| Software Architecture Document | Date: <20/Dec/25> |
| <document identifier>          |                   |

## 4.5 Component: User Pages



**Description:** This component represents the **Customer Interface**—the actual website screens that your customers interact with. Unlike the Admin pages (which are dense with data tables), these pages are designed for **Speed, Simplicity, and Visual Appeal**. They focus on helping a "distracted customer" (as described in your User Environment) navigate from "Hungry" to "Ordered" in under 3 minutes.

### Key Responsibilities:

1. **Visual Presentation:** Displays the drink menu, prices, and images attractively to entice customers.
2. **Input Collection:** Provides easy-to-use buttons and forms for customizing drinks (e.g., "50% Sugar", "Less Ice") without typing.
3. **Responsive Design:** Automatically adjusts the layout to fit perfectly on a small mobile phone (for commuters) or a large laptop screen (for office workers).
4. **Feedback:** Instantly shows the user what is happening (e.g., a "spinner" when loading, or a green checkmark when an order is placed).

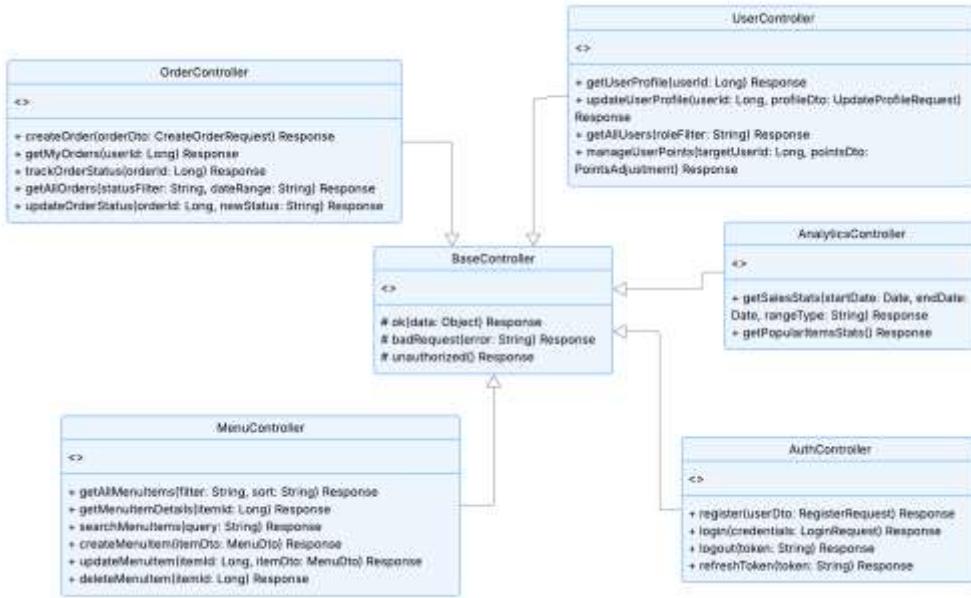
### Specific Pages (The "Screens"):

- **LandingPage**
  - **Role:** The "Storefront Window."
  - **Function:** Shows current **Promotions**, "Best Sellers," and a big "Order Now" button to get users started quickly.
- **MenuPage:**
  - **Role:** The Digital Menu Board.
  - **Function:** Displays the list of drinks (Coffee, Tea, etc.). Includes the **Search and Filter** bar (Feature 4.1.4) so users can find "Latte" or "Cheap drinks" instantly.
- **ProductDetailPage:**
  - **Role:** The Customization Station.

|                                |                   |
|--------------------------------|-------------------|
| Coffee Base                    | Version: <1.1>    |
| Software Architecture Document | Date: <20/Dec/25> |
| <document identifier>          |                   |

- **Function:** Pops up when a user clicks a drink. Allows **Customizing drinks** (Size, Toppings, Sugar Level) (Feature 4.1.5) and adding them to the cart.
- **CheckoutPage:**
- **Role:** The Cashier Counter.
- **Function:** Summarizes the order, allows selecting **Payment Methods** (Momo, ZaloPay), and confirms the pickup time.
- **TrackingPage:**
- **Role:** The "Waiting Area."
- **Function:** Shows real-time status updates (**Preparing -> Ready**) so the customer knows exactly when to walk to the counter (Feature 4.1.7).
- **ProfilePage:**
- **Role:** The Customer's Personal Space.
- **Function:** Shows **Order History**, saved settings, and Loyalty Points balance.

## 4.6 Component: Controllers



**Description:** The **Controllers** serve as the entry point for the application's backend logic. They act as the "traffic directors" for the Coffee Base system, receiving HTTP requests (GET, POST, PUT, DELETE) from the Front-End Router and determining how to process them. They do not contain complex business logic; instead, they parse the request, validate the input, and delegate the work to the appropriate **Services and Modules**.

### Key responsibilities:

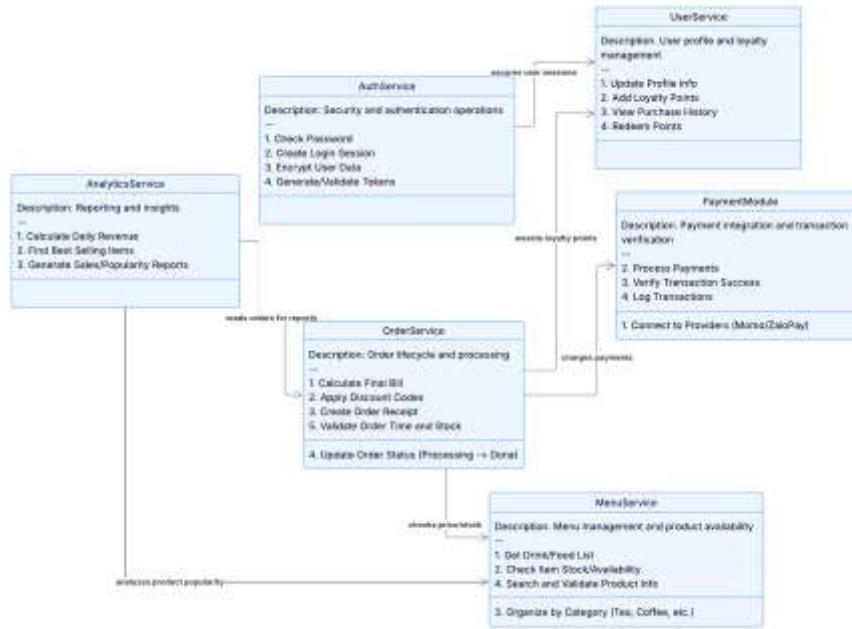
1. **Request Handling:** Receives REST API calls originating from both Customer browsers (e.g., placing an order) and Admin/Staff browsers (e.g., updating order status).
2. **Input Validation:** Performs initial checks on incoming data (e.g., ensuring a password is not empty during login, or that a drink ID exists when ordering) before passing it deeper into the system.
3. **Delegation:** Calls the specific business logic functions located in the **Services and Modules** layer.
4. **Response Formatting:** Returns the final result (data or error messages) to the Front-End in a standard JSON format.

|                                |                   |
|--------------------------------|-------------------|
| Coffee Base                    | Version: <1.1>    |
| Software Architecture Document | Date: <20/Dec/25> |
| <document identifier>          |                   |

**Specific controllers for Coffee Base:** Based on your Vision Document features, the system will include the following specific controllers:

- **AuthController:**
  - Handles **Sign Up** and **Log In** for both Customers and Admins (Feature 4.1.1, 4.1.2).
  - Manages authentication tokens (verifying who is a "Guest", "User", or "Admin").
- **MenuController:**
  - **Customer Side:** Handles requests to **View drink menu**, **Search/Filter drinks**, and fetch details for **Customize drinks** (Feature 4.1.3, 4.1.4, 4.1.5).
  - **Admin Side:** Handles requests to **Create, Edit, or Remove products** from the menu (Feature 4.2.2).
- **OrderController:**
  - **Customer Side:** Processes **Online ordering** requests and handles **Order tracking** status checks (Feature 4.1.6, 4.1.7).
  - **Admin Side:** Receives updates to change order status (e.g., from "Processing" to "Done") (Feature 4.2.4).
- **UserController:**
  - Manages **User Profile** data updates and **Order history** retrieval (Feature 4.1.10).
  - Allows Admins to **Manage users** and view customer credit points (Feature 4.2.3).
- **AnalyticsController:**
  - Dedicated to the Admin requirement for **Sales statistics**, handling requests to generate time-based income reports (Feature 4.2.5).

## 4.7 Services and Modules



**Description:** This component acts as the "**Brain**" or the "**Chef**" of the Coffee Base system. While the Controllers just take orders (like a waiter), the Services layer does the actual work. It holds all the business rules—the specific logical steps that define how the coffee shop operates. It ensures that no data enters the database unless it makes sense (e.g., you can't buy a coffee if the price is negative).

### Key responsibilities:

|                                |                   |
|--------------------------------|-------------------|
| Coffee Base                    | Version: <1.1>    |
| Software Architecture Document | Date: <20/Dec/25> |
| <document identifier>          |                   |

1. **Applying the Rules:** It enforces the logic of the business. (e.g., "If a user buys 10 drinks, give them a free one" or "Don't allow an order if the shop is closed").
2. **Validating Data:** It checks "ingredients" before cooking. It ensures passwords are strong, email addresses are real, and ordered items actually exist on the menu.
3. **Connecting the Dots:** It coordinates between different parts of the system. For example, when an order is placed, it talks to the *Payment Module* to charge the card, then tells the *Inventory* to remove the beans, and finally tells the *User Profile* to add loyalty points.

#### Specific services:

- **AuthService**
- **Job:** Protects the system.
- **Tasks:** Checks if your login password matches the one stored (encrypted) in the database. Creates the secure "Session Key" (Token) that lets you stay logged in while browsing.
- **MenuService**
- **Job:** Manages the products.
- **Tasks:** Fetches the drink list for the customer. Checks if an item is "In Stock" or "Sold Out." Organizes drinks into categories like "Coffee," "Tea," or "Smoothies" so the menu looks neat.
- **OrderService**
- **Job:** Handles the core buying process.
- **Tasks:** Calculates the final bill (Base Price + Toppings - Discounts). Ensures the pickup time chosen by the user is valid. Updates the status of the order as it moves from "Preparing" to "Ready."
- **UserService (Customer Service)**
- **Job:** Manages customer relationships.
- **Tasks:** Updates your profile (Name, Phone Number). Most importantly, it handles the **Loyalty Program**—adding points when you spend money and deducting them when you redeem rewards.
- **PaymentModule**
- **Job:** Handles money safely.
- **Tasks:** Connects to outside services like **Momo, ZaloPay, or Banking Apps**. It tells the system if the transaction was "Success" or "Failed" so the order can proceed.
- **AnalyticsService**
- **Job:** Helps the Admin understand business performance.
- **Tasks:** Reads through past order history to create reports. It answers questions like "How much money did we make today?" and "What is our most popular drink this month?"

## 5. Deployment

The Coffee Base system is deployed using the layered architecture to ensure security and scalability. The front-end layer supports client and admin browsers, while the logic and data layers are located in secured server nodes.

### 5.1 Components Mapping

| Layer | Physical Machines | How components are hosted |
|-------|-------------------|---------------------------|
|       |                   |                           |

|                                |                   |
|--------------------------------|-------------------|
| Coffee Base                    | Version: <1.1>    |
| Software Architecture Document | Date: <20/Dec/25> |
| <document identifier>          |                   |

|                       |                                     |  |
|-----------------------|-------------------------------------|--|
| Front-end (Client)    | User/Admin's Smartphone or PC       | User Browser, Admin Browser, User Pages, Admin Pages.                        |
| Back-end (Web Server) | Local Host/ Server Host (Cloud)     | Local, Internet through Routers, Controllers, Services & Modules.            |
| Database              | <b>Supabase</b> Server (postgresql) | Supabase help store data like menu, users, user authentication, invoices,... |

## 6. Implementation View

The code follows a strict layered pattern. This ensures that the UI (Presentation) can change without affecting the core business rules (Services).

### 6.1 Deployment Mapping

Tách layer thành từng git riêng

```
/coffee-base-root
  ├── /presentation-layer      # Client-side components & UI logic
  │   ├── /public              # Static assets & entry HTML
  │   └── /src
  │       ├── /views            # User Pages & Admin Pages components
  │       ├── /router            # The Router & Navigation Guards
  │       └── /state              # ShoppingCart & UserSession management
  │       package.json
  └── /application-layer      # The "Server" containing logic
      ├── /controllers          # Entry points: Auth, Menu, Order, Analytics
      ├── /services              # Business Rules: Loyalty calculations, Order logic
      ├── /security              # JWT handling & AuthManager logic
      └── server.js             # Server entry point
  └── /domain-layer           # Core entities and data definitions
      ├── /models                # Database Schemas (User, Product, Order)
      └── /dtos                  # Data Transfer Objects for API payloads
  └── /infrastructure-layer   # External system integrations
      ├── /database              # Connection logic & Migrations
      ├── /gateways              # PaymentModule (Momo/ZaloPay integration)
      └── /logger                 # System logging & Error handling
```