PHENIKAA UNIVERSITY

PHENIKAA SCHOOL OF COMPUTING

COURSE: SOFTWARE ARCHITECTURE

Lab 1: Elicitation and Modeling Requirements for Hotel Management System

INSTRUCTOR:	Vũ Quang Dũng
GROUP 6:	Nguyễn Duy Khánh (23010335 - K17 KTPM)
	Nguyễn Hải Đăng (23010099 - K17 CNTT)
	Trần Trọng Minh (23010563 - K17)
	Nguyễn Thị Lan Anh (23010823 - K17)

CREDIT CLASS:CSE703110-1-2-25(N02)

Ha Noi, December 6, 2025

Contents

1. Abstract/Summary

The Coffee Shop Management System project aims to develop a comprehensive platform to manage: customers, staff, order placement, payments, inventory management, and revenue reporting. This first lab focuses on requirement elicitation and modeling.

We have documented Functional Requirements, Non-Functional Requirements, and Architectural Significant Requirements (ASRs) in a structured format. The core behavior of the system is formally modeled using UML Use Case Diagrams, clearly defining system boundaries, external actors, and key behavioral relationships. This platform establishes the scope and the important architectural guidance elements that will be addressed in subsequent labs, specifically the Layered Architecture design in the Lab.

2. Lab Specific Section: I. Requirements Elicitation & Modeling

The primary entities that interact with the Coffee Shop Management System are:
1. User
2. Administrator
3. Payment gateway
4. Receptionist

2.1. Software Requirements Specifications (SRS)

The following tables document the elicited requirements, which collectively form the basis for the Coffee Shop Management System design.

2.1.1. Functional Requirements (FRs)

These requirements define the specific behaviors and functions the system must provide.

ID	**Description**	**Priority**
FR-01	The system must allow a Staff/Barista to create and manage new orders (add/remove/modify items, add notes, apply discounts).	High
FR-02	The system must allow Staff to process payments for an order. The system must integrate with an External Payment System to securely handle card/e-wallet transactions.	Critical
FR-03	The system must allow Staff to view the current status of all pending orders	Critical
FR-04	The system must allow an Administrator/Manager to perform management operations: Manage Menu, Manage Staff, and Manage Suppliers.	High
FR-05	The system must automatically generate and store an invoice upon successful payment and provide the option to print or send a digital receipt to the Customer (if requested).	Medium

2.1.2. Non-Functional Requirements (NFRs)

These requirements specify criteria that can be used to judge the operation of the system, not specific functions.

ID	**Attribute**	**Description**	**Impact**
NFR-01	Performance (Latency)	The system must respond to order processing and invoice printing queries within 2.0-3.0 seconds under peak load to maintain fast service flow.	High
NFR-02	Security (Integrity)	All transaction data and employee personal information must be encrypted at rest (in the database). Credit card details (if accepted) must be processed directly via the external payment system and not stored internally.	Critical
NFR-03	Reliability (Availability)	The POS system (order creation, payment) must maintain a minimum uptime of 99.9% during operating hours (e.g., 7 AM – 10 PM) to ensure uninterrupted sales.	Critical

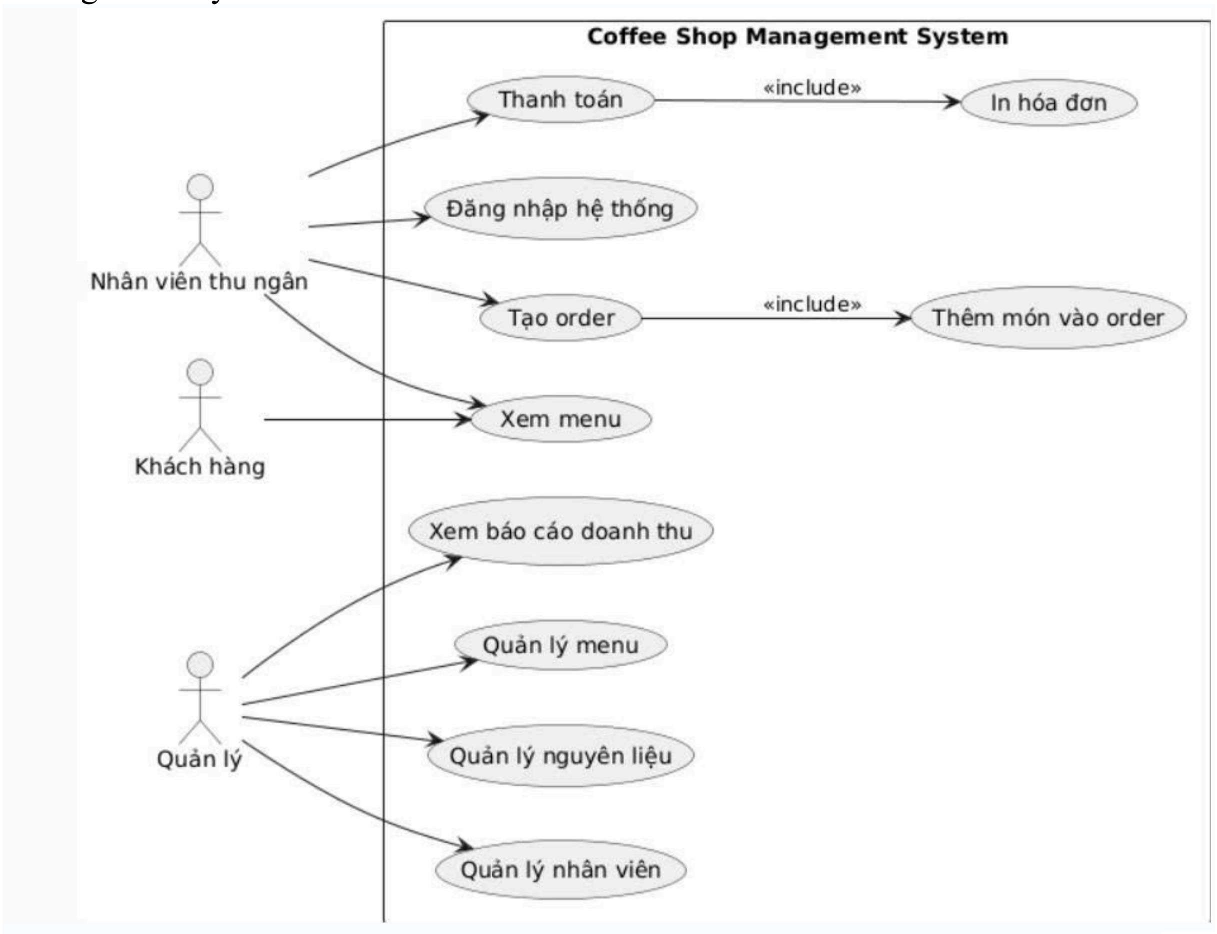
2.1.3. Architecturally Significant Requirements (ASRs)

These are the non-functional requirements that have the most profound influence on the system's architecture, driving the selection of patterns.

ASR ID	**Quality Attribute**	**Requirement Statement**	**Architectural Rationale**
ASR-1	Scalability	The architecture must support a sudden increase in traffic from 100 to 1,000 concurrent users during peak holiday seasons.	This drives the Python backend towards a stateless design, allowing for horizontal scaling and effective load balancing in future deployment stages.
ASR-2	Security	All access to administration functionalities (FR-03, FR-04) must be restricted to authenticated Admins via secure session/token validation.	Necessitates a dedicated Security Middleware/Component located in the Business Logic layer to enforce access control before requests reach the data.
ASR-3	Modifiability	Updates to the Frontend interface (HTML/CSS) must not require changes to the Backend logic (Python).	Enforces the principle of Separation of Concerns. The architecture must strictly separate the Presentation Layer from the Business Layer, ensuring UI changes do not break core functionality.

2.2. Modeling Artifact: UML Use Case Diagram

The UML Use Case Diagram below models the functional scope of the Coffee Shop Management System.



Actors: The system interacts with three primary actors: Cashier Staff, Manager, and Customer.

System Boundary: The box delineates the scope of the Coffee Shop Management System.

Cashier Staff Interactions: Interacts with the system to Log into the system, Create Order (which obligatorily Includes Add item to order), and Payment (which obligatorily Includes Print Invoice).

Manager Interactions: Responsible for managing the system by performing actions such as View revenue report, Manage menu, Manage ingredients, and Manage staff.

Customer Interactions: The sole interaction for this actor is to View menu.

3. Architectural Design

3.1. The Problem Statement

The problem is to design an architecture that satisfies the core requirements while enforcing separation of concerns. The **Layered Architecture** pattern is chosen as the first structural approach to address the Modifiability and Security ASRs. This fits the chosen technology stack (Python Web Frameworks) well.

3.2. Impact of ASRs on Layered Architecture

- ASR-3 (Modifiability) Layered Structure: ASR-3 demands that the Presentation (HTML/CSS) be isolated from the Logic. The Layered Architecture supports this by defining distinct layers. This means the Business Logic Layer (Python services) can be modified without impacting the Presentation Layer (HTML templates), and vice versa. Interaction with the Payment Gateway should be encapsulated within a specific module in the Business Logic Layer. This ensures that switching payment providers (e.g., from Stripe to PayPal) does not affect the Frontend or other backend services.
- ASR-2 (Security) Business Logic Layer: ASR-2 requires rigorous authorization for admin tasks. In the Layered Architecture, this logic is enforced in the Business Logic Layer. This ensures that security checks are applied to core services (e.g., RoomService), protecting them regardless of whether the request comes from the web UI or an API call

4. Conclusion & Reflection

The requirements elicitation phase for the Coffe Shop Management System successfully defined the project scope through detailed Functional and Non-Functional Requirements. The UML Use Case Diagram provides a clear visual model of user interactions. The identified ASRs—especially those relating to Modifiability (Separation of HTML/CSS from Python) and Security—directly necessitate the adoption of a structured pattern like the Layered Architecture for the subsequent design phase (Lab 2) behaviors.