

Template - Sample Lab Report: Lab 1 - Requirements Elicitation & Modeling (ShopSphere)

Contents

1. Cover Page	1
2. Abstract/Summary	1
3. Lab Specific Section: I. Requirements Elicitation & Modeling	2
3.1 Software Requirements Specifications (SRS)	2
3.2 Modeling Artifact: UML Use Case Diagram	4
4. Architectural Design (<i>note: Problem analysis of next lab</i>)	4
5. Conclusion & Reflection	5

1. Cover Page

- **Title:** Elicitation and Modeling Requirements for the ShopSphere E-commerce Platform
- **Course Info:** Software Architecture
- **Student Details:** [Your Name], [Your Student ID]
- **Date:** December 5, 2025

2. Abstract/Summary

The ShopSphere project aims to develop a robust e-commerce platform. This initial lab focused on **requirements elicitation and modeling**. We documented essential **Functional, Non-Functional, and Architecturally Significant Requirements (ASRs)** in a structured format. The core system behavior was formally modeled using a **UML Use Case Diagram**, clearly defining system boundaries, external actors, and key behavioral relationships.

This foundation establishes the scope and the critical architectural drivers that will be addressed in subsequent labs, particularly the design of the **Layered Architecture** in next Lab 2.

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

This section documents the deliverables for Lab 1, as specified in the report requirements.

3.1 Software Requirements Specifications (SRS)

The following tables document the elicited requirements, which collectively form the basis for the ShopSphere design.

3.1.1 Functional Requirements (FRs)

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

ID	Description	Priority
FR-01	The system must allow a Web Customer to browse and search for products using keywords and categories.	High
FR-02	The system must allow a Web Customer to add products to a shopping cart and manage quantities.	High
FR-03	The system must support the Checkout process, including shipping information and payment processing (e.g., Make Purchase).	Critical
FR-04	The system must allow an Admin to perform CRUD (Create, Read, Update, Delete) operations on product inventory and categories.	High
FR-05	The system must generate an order confirmation and send it to the customer upon successful purchase.	Medium

3.1.2 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 90% of product search queries within 2.0 seconds under normal load.	High
NFR-02	Security (Integrity)	All customer data, especially payment information, must be transmitted and stored using encryption (e.g., HTTPS, AES-256).	Critical

ID	Attribute	Description	Impact
NFR-03	Reliability (Availability)	The system must maintain a minimum of 99.9% uptime during operational hours (8:00 AM - 10:00 PM UTC).	Critical
NFR-04	Usability	The user interface must be accessible and optimized for mobile devices.	Medium

3.1.3 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 rapid horizontal scaling to accommodate a sudden increase from 500 to 5,000 concurrent users during seasonal peak sales (e.g., Black Friday).	This high-demand requirement challenges a simple monolithic structure and will drive the consideration of component separation and horizontal deployment in later labs.
ASR-2	Security	All access to administration functionalities (FR-04) must be restricted to authenticated and authorized Admin users via secure token validation.	Directly necessitates a dedicated Security Component to enforce access control, which must be integrated at an appropriate layer (e.g., Business Logic).
ASR-3	Modifiability	Integrating a new third-party Payment Gateway must not require changes to the Product Inventory or User Management components.	Enforces the principle of Separation of Concerns , driving the design toward a layered or componentized structure where business logic is well-isolated.

3.2 Modeling Artifact: UML Use Case Diagram

The UML Use Case Diagram below models the functional scope of the ShopSphere system, showing the system boundary, the external **Actors (Web Customer, Admin)**, and the core Use Cases.

- **Actors:** The diagram clearly identifies the two required actors: **Web Customer** and **Admin**.
 - **System Boundary:** The box delineates the scope of the system under consideration.
 - **Core Use Cases:** Key high-level functions, such as **Make Purchase** and **Manage Inventory**, are shown.
 - **Critical Path Flow:** The diagram details the **Checkout** process, which is a critical path for the system.
 - **Relationships Flow:**
 - **Include Relationship:** The "Select Payment Method" Use Case is a mandatory part of the **Checkout** process, modeled using the "include" relationship.
 - **Extend Relationship:** The "Apply Coupon Code" Use Case is an optional behavior that extends the **Checkout** process, modeled using the "extend" relationship.
-

4. Architectural Design *(note: Problem analysis of next lab)*

The requirements elicited in Lab 1 have a direct impact on the design challenge of **Lab 2: Layered Architecture Design**.

4.1 The Problem Statement

The problem is to design an architecture that satisfies the core requirements while enforcing **separation of concerns** and **strict dependency rules**. The **Layered Architecture** pattern is chosen as the first structural approach to address the Modifiability and Security ASRs identified in Lab 1.

4.2 Impact of ASRs on Layered Architecture

- **ASR-3 (Modifiability) → Layered Structure:** ASR-3 demands that component changes remain isolated (e.g., a new Payment Gateway should not affect the Product component). The **Layered Architecture** inherently supports this by strictly defining four layers (**Presentation, Business Logic, Persistence, Data**) with **downward dependencies**. This means:

- The **Business Logic** (where payment integration code resides) can be modified without impacting the **Presentation** layer.
- The **Persistence** layer can be changed (e.g., switching databases) without affecting the **Business Logic** layer, provided the component interfaces (Lollipop/Socket) remain consistent.
- **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 the core services (e.g., ProductService), protecting them from unauthorized access regardless of whether the request originates from a web UI (Presentation Layer) or an internal script.

The next step (Lab 2) will use these requirements to formally define the four architectural layers and model the logical components (Controller, Service, Repository) and their interfaces within those layers.

5. Conclusion & Reflection

The requirements elicitation phase for ShopSphere successfully defined the project scope through detailed Functional, Non-Functional, and Architecturally Significant Requirements. The UML Use Case Diagram provides a clear visual model of user interactions and the critical path. The identified ASRs—especially those relating to **Modifiability** and **Security**—directly necessitate the adoption of a structured pattern like the **Layered Architecture** for the subsequent design phase (next Lab 2), ensuring the architecture can sustainably support the required system behaviors.