Lab Report Structure and Content Requirements

Each lab report should follow a consistent structure. For the final project submission, a comprehensive report covering all 8 labs is typically required.

I. General Report Structure

Section	Purpose	Typical Length
1. Cover Page	Title, Course Info, Student Details, Date.	1 Page
2. Abstract/Summary	Brief overview of the project (ShopSphere) and the key architectural patterns implemented and compared.	1 Paragraph
3. Lab Specific Section (I-VIII)	Detailed documentation of the activities, diagrams, code snippets, and results for each lab.	Variable
4. Conclusion & Reflection	Summary of architectural learning and key trade-offs observed.	1 Page

---

II. Required Content Breakdown by Lab

The following outlines the specific deliverables required in the content section for each of the eight labs.

Lab 1: Requirements Elicitation & Modeling

- **Requirements Tables:** Documented Functional, Non-Functional, and the three **Architecturally Significant Requirements (ASRs)**.
- **Modeling Artifact: UML Use Case Diagram** showing the system boundary, Actors (Web Customer, Admin), core Use Cases (e.g., Make Purchase), and the use of $\ll\text{include}\gg$ and $\ll\text{extend}\gg$ relationships for a critical path (e.g., Checkout).

Lab 2: Layered Architecture Design

- **Documentation:** Table defining the four layers (Presentation, Business Logic, Persistence, Data) and their strict downward dependencies.

- **Modeling Artifact: UML Component Diagram** showing the logical components (Controller, Service, Repository) within their respective layers, explicitly illustrating the interfaces (lollipop/socket notation) and dependencies.

Lab 3: Layered Architecture Implementation

- **Code Structure:** Screenshot or listing of the final project directory structure (presentation/, business_logic/, persistence/).
- **Code Snippets:** Key Python code snippets demonstrating the flow: the ProductController calling the ProductService, and the ProductRepository handling data access.
- **Verification:** Screenshots or output logs showing successful **CRUD** operation tests (e.g., POST request success, GET request retrieving data).

Lab 4: Microservices Decomposition & Communication

- **Decomposition Table:** Table listing the five core **Microservices** and their associated business capabilities and data ownership.
- **Documentation:** Defined **Service Contract (API Endpoints)** for the Product Service (GET /api/products, GET /api/products/{id}).
- **Modeling Artifact: C4 Model (Level 1: System Context Diagram)** showing the ShopSphere system boundary, external Actors, External Systems (Payment Gateway, Email Service), and the high-level communication types.

Lab 5: Implementing the Product Microservice

- **Setup:** Listing of the pip install dependencies (Flask, Flask-SQLAlchemy).
- **Code Snippets:** Listing of the **Product Model** (SQLAlchemy schema) and the Flask routes for the core GET endpoints (/api/products and /api/products/<id>).
- **Verification:** Output logs/screenshots showing the service running independently on its dedicated port (**5001**) and successful testing of the search/read endpoints.

Lab 6: Introducing the API Gateway Pattern

- **Code Snippet:** Key Python code for the **Security Stub** (validate_token function).
- **Code Snippet:** The Flask route in gateway.py demonstrating the **request forwarding logic** using the requests library.
- **Verification:** Test output showing the following three critical results:
    1. **401 Unauthorized** response (Security check passed).

2. **200 OK** response (Successful routing to Product Service).
3. **503 Service Unavailable** response (Graceful handling when the backend service is stopped).

Lab 7: Event-Driven Architecture (EDA) & Integration

- **Setup:** Confirmation that the **RabbitMQ broker** was successfully started (e.g., Docker status).
- **Code Snippets:** Key code for the **Producer** (order_service_producer.py) showing the use of channel.basic_publish.
- **Code Snippets:** Key code for the **Consumer** (notification_service_consumer.py) showing the callback function and channel.start_consuming.
- **Verification:** Terminal output from the two separate processes demonstrating that the Producer finishes publishing quickly, while the Consumer processes the events with a deliberate delay, proving **asynchronous decoupling**.

Lab 8: Deployment View & Quality Attribute Analysis (ATAM)

- **Modeling Artifact: UML Deployment Diagram** showing nodes (Load Balancer, Application Cluster), artifacts (Product Service, API Gateway, Message Broker), and their network associations.
- **Analysis:** Completed **ATAM Analysis Table** evaluating the Monolithic vs. Microservices architecture against the two defined scenarios (**Scalability - SS1** and **Availability - AS1**).
- **Reflection:** A summary paragraph identifying the main architectural **Trade-offs** learned throughout the project.