Software Engineering- System Architecture/Component Diagram

# Project Topic (e.g. Online Grocery Store Management System)

Group Members

|  |  |  |
| --- | --- | --- |
| Member Name | Member Surname | Student ID |
| Alican | Kaya | 49429 |
| Ege Bora | Güzel | 58351 |
| İrfan | Bag | 49279 |
| Cenker | Sule | 58566 |
| Efe Berk | Tancı | 59115 |
| Berke | Turk | 58169 |

# Introduction

The purpose of this System Architecture and Component Diagram document is to define the high-level structure and design of the Hotel Booking and Management Platform. It offers a comprehensive overview of the system’s architecture, its components, and their interactions. This document guides developers, architects, project managers, and stakeholders in understanding how the platform fulfills its functional and non-functional requirements, ensures maintainability, and supports future growth.

A well-defined architecture:

* Clarifies the system’s structure and component responsibilities.
* Ensures performance, security, and usability requirements are met.
* Facilitates teamwork through clear module boundaries.
* Promotes scalability and adaptability via modular design.

## Architectural Style

Chosen Style: Layered Architecture

Justification:

1. Separation of Concerns: Each layer (Presentation, Business Logic, Data Access) isolates responsibilities, improving maintainability and easing future updates.
2. Controlled Dependencies: Layers communicate only through defined interfaces, reducing coupling and simplifying testing.

Alignment with Requirements:

* Functional Requirements: The Presentation layer handles all user interactions (registration, login, booking flow), the Business Logic layer enforces reservation and payment rules, and the Data Access layer manages CRUD operations on the database.
* Non-Functional Requirements: Layered separation enhances security by isolating data operations, improves performance through targeted optimizations per layer, and increases reliability via clear error-handling boundaries.

Scalability and Flexibility:

* Additional services (e.g., external payment gateways, analytics) can be integrated into the Business Logic layer without impacting the Presentation layer.
* Data Access implementations can be swapped (e.g., migrating from MySQL to PostgreSQL) by modifying only the Data Access layer.

# Identification of components

Identify at least six major components of your system (e.g., user interface, business logic, data storage). Each component must have a clear name and a distinct role in the system. Provide a brief description of what each component does. For example: "The data storage component handles all database interactions, including create, read, update, and delete (CRUD) operations." This ensures your components are purposeful and well-defined. Show how your architecture and components connect to specific requirements. For example: "The security component meets the non-functional requirement for data encryption."

## Component 1 – Presentation Layer

Role: Renders web pages (index.php, rooms.php, contact.php), processes user input, manages sessions.

Addressed requirements: FR: User registration, login, booking; NFR: Usability

## Component 2 – Business Logic Layer

Role: Implements core workflows: reservation validation, cancellation rules, payment orchestration.

Addressed requirements: FR: Reservation lifecycle, cancellation; NFR: Correctness

## Component 3 – Data Access Layer

Role: Provides database abstraction using PDO, executes CRUD operations on domain tables (users, rooms, bookings)

Addressed requirements: FR: Data persistence; NFR: Data integrity, error handling

## Component 4 – Database Server

Role: MySQL/MariaDB instance storing tables: users, rooms, bookings, carousel, settings, queries, features.

Addressed requirements: FR: Reliable data storage; NFR: Availability, Backup

## Component 5 – Payment Processor

Role: Simulates payment gateway, validates payment details, updates booking status upon successful transaction.

Addressed requirements: FR: Payment processing; NFR: Security, Transactional safety

## Component 6 – Admin Panel

Role: Web-based dashboard (admin/index.php, dashboard.php) for managing carousel, rooms, features, user queries.

Addressed requirements: FR: Admin CRUD operations; NFR: Authorization, Security

# Component Diagram

Below is a textual overview of the UML component diagram. Each component communicates via well-defined interfaces:

* Presentation Layer → (uses) → Business Logic Layer
* Business Logic Layer → (uses) → Data Access Layer
* Data Access Layer → (uses) → Database Server
* Business Logic Layer → (calls) → Payment Processor
* Admin Panel → (uses) → Business Logic Layer & Data Access Layer

+---------------------+ +-----------------------+ +-------------------+  
| Presentation Layer | ---> | Business Logic Layer | ---> | Data Access Layer |  
+---------------------+ +-----------------------+ +-------------------+  
 | |  
 v v  
 +----------------+ +---------------------+  
 | Payment | | Database Server |  
 | Processor | | (MySQL/MariaDB) |  
 +----------------+ +---------------------+  
 ^ ^  
 | |  
 +-------------+ +--------------+  
 | Admin Panel | ------------> | User Queries |  
 +-------------+ +--------------+

* Interfaces:
* IUserService between Presentation and Business Logic for user operations.
* IBookingService for reservation workflows.
* IPaymentGateway for payment interactions.
* IDataRepository<T> generic interface for Data Access.

# Evaluation Criteria

**1. Architectural Style Selection (2 points)**

* **Choice of Architecture (1 point)**  
  You must select an architectural style (e.g., monolithic, microservices, layered, event-driven) that suits your project’s needs and constraints. To earn this point, justify your choice with at least two specific reasons. For example: "A monolithic architecture is chosen for its simplicity and ease of deployment, given the small team size and tight timeline."
* **Alignment with Requirements (1 point)**  
  Explain how your chosen architecture supports the functional and non-functional requirements from your project’s Requirements Specification. For example: "The layered architecture separates concerns, improving maintainability, which aligns with the requirement for easy updates."

**2. Component Identification and Description (2 points)**

* **Identification of Components (1 point)**  
  Identify at least three major components of your system (e.g., user interface, business logic, data storage). Each component must have a clear name and a distinct role in the system.
* **Component Responsibilities (1 point)**  
  Provide a brief description of what each component does. For example: "The data storage component handles all database interactions, including create, read, update, and delete (CRUD) operations." This ensures your components are purposeful and well-defined.

**3. Component Diagram Quality (2 points)**

* **Diagram Completeness (1 point)**  
  Your UML component diagram must include all identified components, their interfaces (if applicable), and the relationships between them (e.g., dependencies or communication paths). Missing elements will cost you this point.
* **Clarity and Standards (1 point)**  
  The diagram must follow UML standards for component diagrams, with clear labels, consistent notation, and a logical layout. It should be easy to understand without clutter or confusion.

**4. Integration with Project Requirements (1 point)**

* **Traceability to Requirements (0.5 points)**  
  Show how your architecture and components connect to specific requirements. For example: "The security component meets the non-functional requirement for data encryption."
* **Scalability and Flexibility (0.5 points)**  
  Briefly explain how your design supports future growth or changes. For example: "The microservices architecture allows high-traffic components to scale independently."

**5. Timely Submission (1 point)**

* Submit the task by the deadline to earn this point. Late submissions, even if high-quality, will not receive this point.