1 Introduction

1.1 Identifying Information

- **Architecture Name**: **Layered Client-Server Architecture**
    - The system is divided into distinct layers: API Layer (handles user requests), Service Layer (implements business logic), Data Access Layer (manages databases and external APIs), and Event-Driven Messaging (supports asynchronous communication).
- **System of Interest: Travel Agency Booking System**
    - A software system for managing hotel bookings, event ticketing, and user accounts, integrated with external service providers. It is designed for scalability and efficiency, ensuring responsiveness during peak booking times.

1.2 Supplementary Information

- **Date of Issue:**
    - Initial Issue Date: October 8, 2022
    - Last Updated: November 29, 2024
- **Document Status:**
    - Version: 1.0
    - Status: Draft
- **Authors:**
    - Rahaf Yasser: Software Architect
    - Halla Ayman: Senior Developer
    - Mariam Sherif: Database Administrator

- **Reviewers:**
  - Mohamed Mohsen: Lead Developer
  - Alhassan: Project Manager
- **Issuing Organization:**
  - Cairo University, Software Development Team
- **Change History:**
  - Version 1.0: Initial version created for the Travel Agency Booking System architecture.
- **Version Control Information:**
  - Version 1.0: First draft of the architecture document.
- **Summary:** This document outlines the architecture for the Travel Agency Booking System, a platform for managing hotel bookings, event ticketing, and user account management. It is designed with a layered client-server structure to ensure maintainability, scalability, and integration with third-party APIs.
- **Scope:** This architecture applies to the online travel agency system that supports web and mobile platforms for users to search, book, and manage hotel and event reservations. It is built for scalability and future enhancements like personalized recommendations.
- **Context:** The system will use external APIs for real-time hotel and event data. It is designed to handle high traffic during peak times, ensuring fast performance and secure user transactions. The architecture is scalable, supporting future growth and new features like service integrations and personalized recommendations.

- **Glossary:**
    - **API (Application Programming Interface):** A set of protocols that allow different software systems to communicate. It enables the system to fetch data from external services, like hotel availability or event details.
    - **DAO (Data Access Object):** A design pattern that simplifies access to a database. It separates how data is retrieved from the rest of the system, making it easier to manage and update.
    - **Layered Architecture:** A design approach where the system is divided into layers, each with a specific role, like handling user interface, business logic, or database management. This separation helps in building, maintaining, and scaling the system.

1.3 Other information

System Overview

The system follows a **Layered Client-Server Architecture**, which organizes the system into distinct layers to ensure clarity, maintainability, and scalability. The architecture includes the following layers and components:

Layers:

1. **API Layer**:
    - Handles user requests and connects the frontend to the backend. It exposes endpoints for tasks like searching hotels or booking events.
2. **Service Layer**:
    - **User Service:** Manages user accounts and dashboards.
    - **Hotel Service:** Manages hotel searches and bookings.
    - **Event Service:** Handles event details and bookings.
    - **Notification Service:** Sends emails and SMS for bookings and recommendations.

3. **Data Access Layer:**
    - Manages data storage and retrieval by interacting with the database and external APIs.

Components:

1. **API Endpoints:**
    - /**register:** For user sign-up.
    - **/search-hotels**: For searching hotels.
    - **/book-event:** For event bookings.
2. **Database:**
    - Stores all the system's data, such as user information, bookings, and notifications.
3. **Message Queue:**
    - Allows services to send and receive updates (e.g., when a booking is made, a notification is triggered).
4. **External APIs:**
    - The backend connects to third-party systems for hotel and event information.

Connectors:

1. **HTTP/HTTPS:**
    - Connects the frontend to the backend securely.
2. **Message Queue (e.g., RabbitMQ):**
    - Allows services to send updates to each other without direct connections.
3. **Database Connection:**
    - Links the backend to the database.
4. **External API Connectors:**
    - Connects to third-party hotel and event systems.

The system uses **Event-Driven Messaging** for fast, seamless communication between layers. This keeps the backend organized, scalable, and easy to maintain while supporting future features like personalized recommendations, new service integrations, and mobile optimization.

Stakeholders and Concerns

1. **Users:**
   - **Concerns**: Usability, performance, availability, and security. Administrators need tools to monitor, fix, and update the system. These tools should be able to handle more traffic during peak times and quickly detect and solve security issues.
2. **System Administrators:**
   - **Concerns:** Maintainability, scalability, and security. Administrators need tools for system monitoring, quick scaling, and security management.
3. **Acquirers (Business Owners/Investors):**
   - **Concerns:** Cost-effectiveness, and maintainability. The system should be efficient, delivered on time, and provide long-term business value.
4. **Owners (Travel Agency Executives):**
   - **Concerns:** Availability, and cost-effectiveness. The system should support uninterrupted operations and adhere to regulations.
5. **Suppliers (Third-Party Providers):**
   - **Concerns:** Compatibility, performance, and security. Suppliers need smooth integration and secure data exchange.
6. **Developers:**
   - **Concerns:** Maintainability, testing, and scalability. Developers need a modular system that is easy to update, test, and extend for future features.
7. **Builders (Infrastructure Managers):**
   - **Concerns:** Availability, scalability, and cost-effectiveness. Infrastructure must be adaptable, scalable, and resilient to failures.

8. **Maintainers:**
    - **Concerns:** Maintainability, availability, and. The system must remain stable, easy to support, and upgrade over time.

Reader’s Guide to this Architecture Description (AD)

This document explains the system's architecture and is divided into the following sections:

- **Identifying Information**: Includes details like the architecture name, system, date, authors, and version.
- **Architecture Overview**: Provides a summary of the system’s architectural style and structure.
- **Architecture Views**: Describes key system views, such as the Conceptual View and Execution View.
- **Design Rationale**: Explains why specific architecture choices, like the Layered Client-Server model, were made.
- **Scenarios and Evaluation**: Shows real-world examples to test how the architecture handles key use cases.

This guide helps readers understand the structure and reasoning behind the architecture and how it meets both technical and business needs.

Results of Architecture Evaluations

The architecture will be tested through different scenarios to check its strength, scalability, and response time. The scenarios include:

- **High Traffic Load**: Tests how the system handles more users during busy booking times.
- **API Failure**: Tests how the system deals with failures from external services (like hotel or event APIs).

These evaluations will ensure the architecture can manage the expected load, stay available, and recover smoothly from errors or failures.

1.3.1 Requirements

Functional Requirements:

1) **Hotel Search and Booking:**
    - System allows users to search for hotel rooms by type (single, double, family) and availability.
    - Enable users to book hotel rooms directly through the system.
2) **Event Search and Booking:**
    - System allows users to search for events using filters like date or location.
    - Enable users to book tickets for events.
3) **User Account Management:**
    - System requires users to create accounts to book hotels or events.
    - Allow users to log in to access their bookings.
4) **Access Dashboard:**
    - System provides users with a dashboard to view their bookings.
    - Allow users to print booking details from the dashboard.
5) **Event Recommendations:**
    - System recommends events happening in the destination location during the user's stay.
6) **Manage Notification Templates:**
    - System supports templates for different types of notifications (e.g., booking confirmation, password reset).
7) **Queue Notifications:**
    - Notification Service System implements a queue to store notifications that need to be sent.
    - Avoid sending notifications directly during the invocation, reducing wait times.
8) **Handle Notification Queue:**
    - Notification Service System dequeue notifications and send them through the appropriate channel (email or SMS).
    - Handle failures for notifications that are not sent successfully.
    - Retry sending notifications if possible.
9) **Provide Notification Statistics:**

- Notification System Statistics calculates total successfully sent notifications by type (email, SMS).
- Total failed notifications with reasons of failure.
- Most notified email address or phone number.

10) **Payment**
    - System enables users to securely complete payments for hotel and event bookings.

Non-Functional Requirements

- **Usability**: The system should have an intuitive and easy to follow GUI and have users complete key functions within 5 steps.
- **Availability**: The system should be available to use 99% of the time outside of the maintenance times
- **Scalability**: The system should be able to handle up to 1000 concurrent order requests without degradation in performance.
- **Security**: The system should use up-to-date network protocols (HTTPS) and use hashing to encrypt sensitive user data to ensure privacy.
- **Compatibility**: The system should support multiple platforms like IOS, Android, Linux and Windows
- **Performance**: The system should load pages and perform actions within a timeframe of 3 seconds.

1.3.2 Rationale for Key Decisions

Several important decisions were made to ensure the system meets both functional and non-functional needs:

- **Layered Architecture**: We chose a Layered Client-Server model to separate different parts of the system, making it easier to scale and maintain. This also reduces dependencies between layers.
- **Modular Components**: Each layer (Presentation, Business Logic, Data Access, and Integration) is modular, so changes can be made in one area without affecting the whole system. This makes the system easier to maintain and test.

- **API Integration**: External hotel and event APIs were included to provide real-time data, keeping the system up-to-date and competitive in the travel industry.
- **Security**: Security measures like encryption and secure API calls were added, especially in the Business Logic and Data Access layers, to protect user data and transactions.

View and Model Correspondences

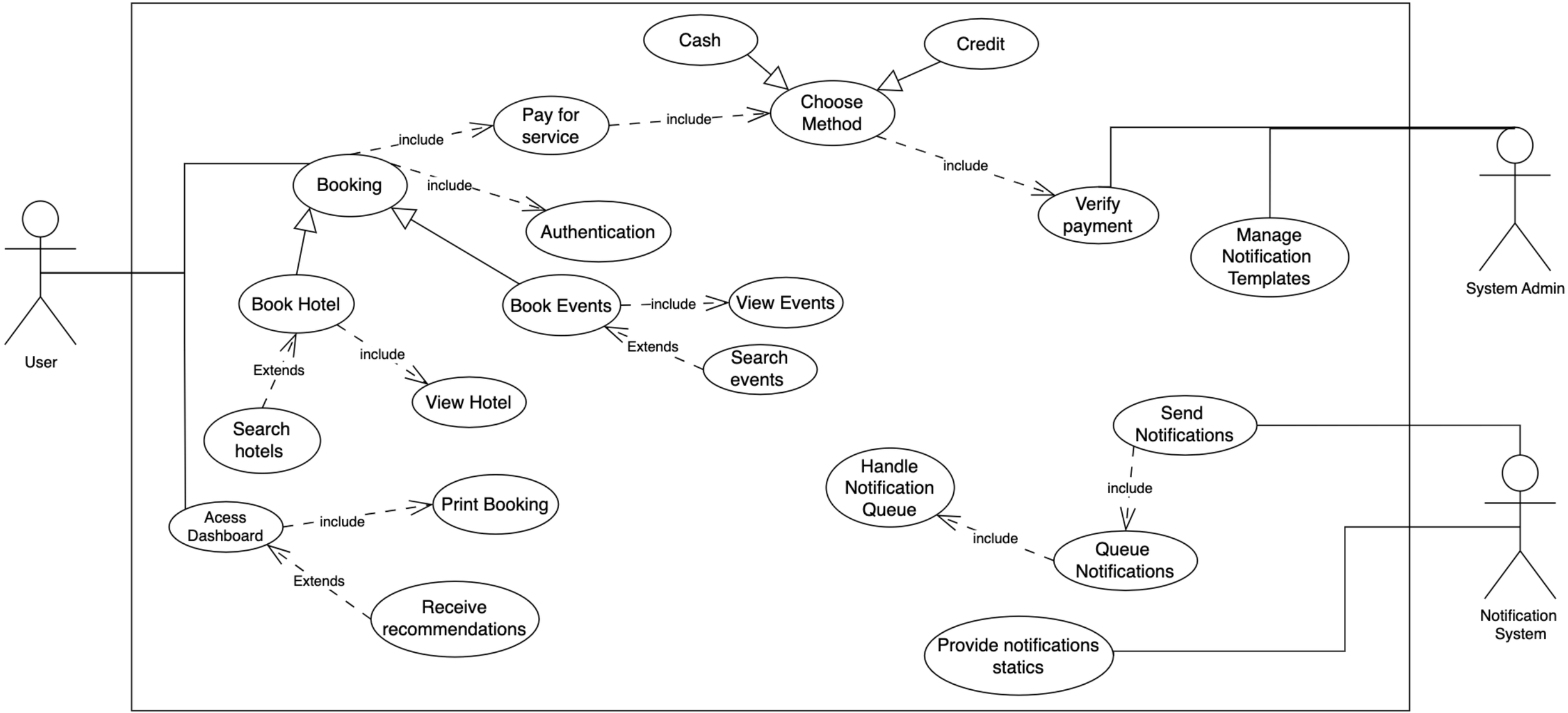
The **Conceptual View** and **Execution View** are aligned with the layered structure of the system.

- **Conceptual View**: This view represents the system's static structure, focusing on the components and their relationships within each layer (Presentation, Business Logic, Data Access, and Integration).
- **Execution View**: This view describes the dynamic interactions and data flows between components, especially during user interactions such as booking a hotel or event. The view also highlights how concurrency is managed across layers to ensure smooth operations.
- **Allocation View:** This view will be used to allocate specific resources (e.g., servers, databases, external services) to each component of the system. It ensures that the system is designed for optimal performance by considering factors like resource usage, load balancing, and scalability. The Allocation View will help identify how physical resources will support the system's architecture, ensuring that the system can handle expected traffic and scale effectively during peak loads.

Evaluation Scenarios

Usability	A user attempts to book a hotel room but cannot find availability due to confusing search filters.	The system provides real-time suggestions, such as alternative dates or nearby hotels, with a tooltip explaining how to refine search filters.
Availability	The database server becomes temporarily unavailable during a user's hotel booking.	The system stores the booking request in a local queue and retries the operation when the database connection is restored. The user is notified about the delay, and no duplicate bookings are created.
Availability	The network connection to the events API becomes unavailable.	The system switches to a cached version of recent event data. Notifications on the user dashboard are delayed until the connection is restored. The system logs this incident for review.
Modifiability	A new type of notification (e.g., promotional discount notification) must be supported.	The notification module allows developers to add a new notification template via the admin interface. The template includes placeholders for dynamic fields such as user name, discount amount, and expiration date. The addition is reflected immediately across all supported languages and channels (email and SMS).
Modifiability	The supplier of the external events API discontinues their service, requiring integration with a new events API provider.	The backend' s abstract service layer for event data is reimplemented to integrate with the new events API. This ensures seamless operation for client components, as they continue to interact with the unified service layer without any changes.
Security	An unauthorized user attempts to modify a booking through an API call.	The system rejects the request and logs the activity. An alert is sent to the admin dashboard with details of the malicious request.
Scalability	The number of active users surpasses the expected threshold, causing slow dashboard performance.	A caching layer is added for user dashboards, storing pre-fetched booking data and reducing the load on the database. The cluster scales automatically to distribute user requests effectively.

Compatibility	The system must integrate with a new payment gateway for hotel bookings.	The backend uses a payment adapter pattern to connect the new gateway, ensuring uniform behavior across all supported gateways. The frontend is updated to list the new payment option.
Performance	The average response time for API requests exceeds 2 seconds due to increased traffic.	The system prioritizes critical operations (e.g., bookings) by queueing non-essential requests. Caching mechanisms for frequently accessed data (e.g., hotel details) are optimized, and real-time monitoring alerts administrators to performance bottlenecks.


Cash
Credit
Choose Method
Pay for service
include
include
Booking
include
Authentication
include
Verify payment
Manage Notification Templates
System Admin
User
Book Hotel
Book Events
include
View Events
Extends
Search events
include
Extends
View Hotel
Search hotels
Send Notifications
Handle Notification Queue
include
include
Queue Notifications
Acess Dashboard
include
Print Booking
Extends
Receive recommendations
Provide notifications statics
Notification System

2. Stakeholders and Concerns

2.1 Stakeholders

The following stakeholders are identified for the Travel Agency Booking System, each with unique interest and concerns that directly influence architectural decisions.

1. **Users**:
    - **Description**: Customers using the system to search for and book hotels, events, and travel services.
    - **Concerns**:
        i. Usability: Users expect the system to be simple and easy to use, with a smooth experience.
        ii. Performance: Users expect the system to be fast and responsive even during peak usage times.
        iii. Security: Users expect their personal data and payment information are highly secured.
        iv. Availability: Users require the system to be available for browsing and booking throughout the day.
2. **System Administrators**:
    - **Description**: People who manage the system to keep it running well every day.
    - **Concerns**:
        i. Maintainability: They need tools to monitor, fix, and update the system easily.
        ii. Scalability: The system must handle more users during busy times, like holidays.
        iii. Security: System Administrators need ways to detect and handle threats or data breaches quickly.

3. **Acquirers**:
    - **Description**: Business managers, investors, and project leaders funding and supervising the system.
    - **Concerns**:
        i. Cost-Effectiveness: Acquirers want the system to deliver value without overspending.
        ii. Maintainability: They expect the system to support future updates and business growth.
4. **Owners**:
    - **Description**: Executives or managers of the travel agency operating the system.
    - **Concerns**:
        i. Availability: They need the system to always support bookings and operations.
        ii. Cost-Effectiveness: The system should increase revenue through smooth bookings or loyalty features.
        iii. Security: The system must follow data protection laws and regulations.
5. **Suppliers** (Third-Party Service Providers):
    - **Description**: Hotels, event organizers, and payment processors integrated with the system.
    - **Concerns**:
        i. Compatibility: Their systems must integrate easily with the platform for updates and bookings.
        ii. Performance: They need accurate and real-time data exchange without delays.
        iii. Security: They expect their sensitive information to be securely handled.

6. **Developers**:
    - **Description**: Software engineers building and coding the system.
    - **Concerns**:
        i. Maintainability: They need the system to be modular and easy to fix or update.
        ii. Testing: The system should allow automated testing to ensure reliability.
        iii. Scalability: It should be easy to add new features without affecting existing ones.
7. **Builders**:
    - **Description**: Teams managing the servers, databases, and networks powering the system.
    - **Concerns**:
        i. Scalability: The infrastructure must handle increased traffic during busy periods.
        ii. Availability: The system should quickly recover from failures or disasters.
        iii. Cost-Effectiveness: They want efficient infrastructure to reduce unnecessary costs.
8. **Maintainers**:
    - **Description**: Teams responsible for long-term support and updates.
    - **Concerns**:
        i. Maintainability: The system should be stable and easy to fix or update over time.
        ii. Availability: Tools and documentation should help resolve issues quickly.

2.2 Concerns

This section identifies the key concerns that are critical for ensuring that the system aligns with both its intended purposes and the needs of its stakeholders throughout its life cycle.

- **Purpose(s) of System-of-Interest**

The main purpose of the Travel Agency Booking System is to offer a reliable platform for users to search, book, and manage hotel and event reservations. It aims to make booking easier for travel agency customers in Cairo, providing real-time hotel availability and event ticket access.

- **Suitability of the Architecture for Achieving the System's Purpose(s)**

The **Layered Client-Server Architecture** is well-suited to achieve the purposes of the system:

  - Scalability: The layered approach allows for the independent scaling of each layer. For instance, as user traffic increases, more resources can be added to the Business Logic Layer without affecting other parts of the system. This ensures that the system can handle peak loads, such as during travel seasons.
  - Flexibility: The modular design of the system ensures that the platform can be easily extended to accommodate new services (e.g., adding flight bookings or new third-party integrations) without major rework.
  - Maintainability: The clear separation between layers (Presentation, Business Logic, Data Access, and Integration) ensures that updates or changes can be made to one layer without affecting others, making maintenance easier and reducing the risk of introducing bugs in the system.

- Feasibility of Constructing and Deploying the System

- Technology Stack: The system uses well-known technologies (ex. relational databases, web services, cloud infrastructure) that are widely supported, reducing the risk of encountering unforeseen technological challenges.
- Deployment Options: The architecture supports flexible deployment options, such as cloud services or on-premise infrastructure, allowing the system to be deployed based on the agency's infrastructure preferences and budget.
- Development Timeframe: Given the system's modular design, construction can be phased, with critical components (like user authentication and hotel bookings) prioritized for earlier releases. This phased approach helps manage development risks and ensures that the system can be deployed in stages, facilitating gradual rollouts.

- Potential Risks and Impacts to Stakeholders Throughout the Life Cycle

Several potential risks and impacts have been identified for stakeholders:

1. Users:
    - Risk: System Downtime or Slow Performance During Peak Periods
        - Impact: Users may experience frustration if the system can't handle high traffic volumes, leading to potential loss of business.
        - Mitigation: System's scalability, supported by cloud infrastructure and modular layers, mitigates this risk by allowing the addition of resources as needed.
1. System Administrators:
    - Risk: Security Vulnerabilities or Data Breaches
        - Impact: Sensitive customers data (payment details), harming agency's reputation, and violating regulations.
        - Mitigation: Use encryption, secure communication protocols, and follow industry standards (ex. PCI – DSS) to protect user data.
2. Suppliers:
    - Risk: Integration Failures or Discrepancies in Real-Time Data
        - Impact: The system might fail to show accurate availability or pricing information, frustrating users and harming suppliers relationships.
        - Mitigation: use integration layer with failover mechanisms and caching to ensure smooth third-party service integration.

3. Acquirers:
    - Risk:
        - Impact: Project may miss deadlines or exceed budget, affecting the return on the investment.
        - Mitigation: Phased development and modular architecture allow for manageable deliverables and early validation of critical functionality, reducing development risks.

- Maintenance and Evolution of the System-of-Interest

1. Software Updates: the layered design ensures that updates to one layer (ex. the Business Logic Layer or Presentation Layer) do not disrupt others. This makes it easier to fix bugs, add new features, or adapt to changing business needs without overhauling the entire system.

2. Scalability: As the agency grows, additional features or services can be added to the system (e.g., integrating flight booking or adding new hotel partners). The modularity of the architecture allows for such expansions with minimal impact on the overall system.

3. Infrastructure Upgrades: The system is built with cloud-based infrastructure in mind, which makes it easier to scale and upgrade hardware resources as the user base grows. This ensures that the system can continue to support high demand without requiring a complete redesign.

4. Adapting to Changing Business Needs: As the travel industry evolves, the system's architecture allows for the introduction of new business requirements, such as personalized recommendations, loyalty programs, or international expansion. The modular and extensible nature of the architecture supports these future changes.

2.3 Concern-Stakeholder Traceability

The following table illustrates the association of each identified concern from Section 2.2 (Concerns) with the relevant stakeholders identified in Section 2.1 (Stakeholders). This table ensures that each concern is addressed by the appropriate stakeholders, allowing for a clear traceability of interests and responsibilities throughout the system's architecture.

Concerns	Users	System Administrator	Acquires	Owners	Suppliers	Developers	Builders	Maintainers
Usability	✓							
Performance	✓				✓			
Security	✓	✓		✓	✓			
Availability	✓			✓			✓	✓
Scalability		✓				✓	✓	
Compatibility					✓			
Cost-Effectiveness			✓	✓			✓	
Maintainability		✓	✓			✓		✓

3 Viewpoints +

3.1 Conceptual Viewpoint

3.2 Overview

The Conceptual Viewpoint gives a high-level overview of the Travel Agency Booking System, focusing on its main parts, their roles, and how they work together. It helps both technical and non-technical stakeholders understand the system's structure without getting into too many technical details. The key points are:

- **Modular Structure:** The system is divided into layers like the Presentation, Business Logic, Data Access, and Integration Layers. This separation helps manage, maintain, and scale the system easily.
- **Component Interactions:** It shows how the components within each layer interact to perform tasks like booking a hotel room or processing payments.
- **System Boundaries**: It defines what's part of the system (e.g., the user interface and what's external (e.g., third-party hotel), helping clarify what's included in the system.

3.3 Concerns and Stakeholders

The Conceptual Viewpoint highlights key concerns about the Travel Agency Booking System's design and structure, focusing on:

- **Usability**: The system must be easy to use for a positive experience.
- **Scalability**: It should be able to grow with more users and new features (e.g., flight bookings).
- Maintainability: Its modular design makes it easier to update and fix issues.
- **Cost-effectiveness**: Organizing the system into layers helps use resources efficiently.
- **Revenue Generation:** The system supports important features (like hotel booking) that impact the agency's revenue.

The stakeholders for this viewpoint include:

- **Users**: They gain an understanding of the system's services and how to interact with it.
- **System Administrators:** It helps them manage and scale the system.
- **Acquirers (Project Managers/Investors):** It shows the system's architecture and growth potential, helping assess its feasibility.
- **Owners (Travel Agency Management):** It helps them see how the system aligns with business goals like scalability and integration with external services.
- **Developers:** It gives them an overview of the system's architecture to plan their work on specific components.

3.3.1 Concerns

This section identifies the key architecture concerns for the Travel Agency Booking System, framed by the Conceptual Viewpoint. These concerns guide architectural decisions to meet stakeholder expectations and align with business and technical goals. Each concern includes guiding questions to help stakeholders assess how the architecture addresses key issues.

- **System Purpose**: Defines the main objectives of the system, like booking management and revenue generation.
    - Question: What core services does the system provide?
- **Architecture Suitability**: Assesses if the chosen architecture supports the system's goals.
    - Question: How does the architecture support core functions like booking and managing reservations?
- **Feasibility**: Focuses on whether the system's design is realistic in terms of technology, budget, and deployment timeline.
    - Question: Is the design achievable within the given resources and timeline?
- **Risks and Impacts**: Identifies potential risks to stakeholders (e.g., security issues, integration failures).

    - Question: What risks could affect the system, and how does the design address them?
- **Maintainability and Evolution**: Ensures the system is easy to update and expand over time.
    - Question: How does the architecture support future updates and new features?
- **Scalability and Flexibility**: Focuses on the system's ability to grow with increasing users and new services.
    - Question: How can the system scale to handle more users and services?
- **Usability and User Experience**: Ensures the system is easy and enjoyable for users to interact with.
    - Question: How does the system ensure a user-friendly interface?
- **Interoperability**: Examines how the system integrates with external services like hotel APIs and payment systems.
    - Question: How does the system communicate with third-party services?

These concerns help ensure that the system meets business needs, is feasible to build, and provides a positive user experience while being scalable and easy to maintain.

3.3.2 Typical Stakeholders

This section identifies the key stakeholders for the Conceptual Viewpoint of the Travel Agency Booking System. These stakeholders use the high-level overview to make decisions and carry out tasks. The viewpoint focuses on the system's components, structure, and interactions to meet its objectives. Stakeholders want to understand the system's design, goals, and approach.

This section lists the main people who need to understand the system's design and structure:

1. **Users:**
    a. **Who**: People booking services (hotels, events).
    b. **Why**: They need to know how to use the system for booking and managing reservations, Conceptual Viewpoint helps ensure that the system design meets their needs for a user-friendly interface and efficient operation
    c. **Focus**: Usability, performance, and functionality.
2. **Operators:**
    a. **Who**: System admins who ensure the system runs smoothly.
    b. **Why**: They need to understand how to manage and fix the system, The Conceptual Viewpoint provides a clear overview of the system's structure, which helps operators monitor the system effectively.
    c. **Focus**: Easy maintenance, reliability, and scalability.
3. **Acquirers (Project Managers/Investors):**
    a. **Who**: People funding or overseeing the project.
    b. **Why**: They want to see how the system meets business goals like growth and profitability, The Conceptual Viewpoint provides high-level insights into how the system is structured to meet these needs.
    c. **Focus**: Cost, scalability, and return on investment.
4. **Owners (Travel Agency Management):**
    a. **Who**: The agency's management team.
    b. **Why**: They need to know how the system supports business goals, The Conceptual Viewpoint helps them assess the overall structure and business value of the system.
    c. **Focus**: Business growth, customer satisfaction, and flexibility.

5. **Suppliers:**
    a. **Who**: Companies providing services (e.g., hotels, payment processors).
    b. **Why**: They want to ensure smooth integration with the system, The Conceptual Viewpoint outlines the integration points and the role of external services.
    c. **Focus**: Integration, reliability, and security

6. **Developers:**
    a. **Who**: Programmers building the system.
    b. **Why**: They need an overview to plan and build the system's components,. The Conceptual Viewpoint provides them with an overview of the system's components, relationships, and dependencies, which is critical for the development process.
    c. **Focus**: Easy to maintain and extend, Maintainability, modularity.
7. **Builders (Infrastructure Team):**
    a. **Who**: Team setting up the system's infrastructure.
    b. **Why**: They need to know how to deploy and scale the system, The Conceptual Viewpoint informs them of the system's structure and deployment needs, including how the layers of the system interact and what hardware or software resources are required.
    c. **Focus**: Easy deployment and scalability.

8. **Maintainers:**
    a. **Who**: Team managing the system after launch.
    b. **Why**: They need to know how to update and fix the system, The Conceptual Viewpoint helps them by showing how the system is structured and where components can be modified or extended without disrupting the entire system.
    c. **Focus**: System stability and easy maintenance.

Each group is interested in different parts of the system, depending on their role and needs.

3.5 Conceptual Viewpoint Model Kind

3.5.1 Conventions for Conceptual Models

The Conceptual Viewpoint uses simple models to describe the system's main components, their roles, and how they interact.

- **Notations and Languages:**
  - **Block Diagrams**: To show layers (Presentation, Business Logic, Data Access, and Integration).
  - **UML Component Diagrams:** For visualizing components and their relationships.
- **Process:**
  - Define the layers and their roles.
  - Show how components communicate to perform tasks like booking and payment.
  - Highlight connections to external services like hotel booking APIs.
- **Key Elements:**
  - **Entities**: Components like the Booking System or Payment Gateway.
  - **Attributes**: Features such as usability and scalability.
  - **Relationships**: Links between layers and external services.
  - **Constraints**: Rules to maintain modularity, scalability, and cost-effectiveness.

3.5.2 Operations on Models

- **Building Models:** Use diagrams and templates to describe system layers and interactions.
- **Interpreting Models:** Help stakeholders understand the structure and how components work together.
- **Analyzing Models:** Ensure scalability, usability, and alignment with system goals.
- **Applying Models:** Guide development and deployment of the system.

3.5.3 Correspondence Rules

- Ensure all layers (Presentation, Business Logic, Data Access, and Integration) align with each other.
- Verify external services (like APIs) are represented clearly and integrate smoothly.
- Confirm that the design supports growth, maintainability, and usability.

3.6 Operations on Views

Key Operations

1. **Construction**
    a. Use diagrams and templates to create high-level models of the system.
    b. Focus on making the design modular and easy to understand.
2. **Interpretation**
    a. Provide insights to stakeholders, especially non-technical ones, about how the system works.
3. **Analysis**
    a. Check if the system can handle more users, new features, and external integrations.
4. **Implementation**
    a. Use the conceptual models as a guide for building and deploying the system.

3.7 Correspondence Rules

Rules Across Models and Views

1. **Consistency Across Layers**: Ensure the Presentation, Business Logic, Data Access, and Integration layers fit together logically.
2. **External Integration:** Confirm external services like hotel booking APIs are properly represented and connected.
3. **Scalability and Modularity:** Make sure the design can grow and be updated easily.
4. **Business Alignment:** Ensure the design supports usability, revenue goals, and cost-efficiency.

3.1 Execution Viewpoint

3.2 Overview

The Execution Viewpoint explains how the Travel Agency Booking System operates in real-time, processing actions like hotel bookings and payments. It shows how data flows, how components interact, and how the system handles issues like errors or high traffic.
Key points of the Execution Viewpoint include:

- **Real-Time Data Flow**: How data moves through the system when a user books a service or makes a payment.
- **Process Interaction**: How different parts of the system talk to each other and work together during use.
- **Error Handling**: How the system deals with problems like network failures or service errors.

3.3 Concerns and Stakeholders

This viewpoint focuses on several main concerns about the system's performance during use:

- **Performance**: Can the system handle a lot of users at once?
- Reliability: Will the system keep working even if something goes wrong?
- **Security**: Is the user's personal data safe during transactions?
- **Scalability**: Can the system grow as the business grows?

Stakeholders:

- **End Users:** People using the system for bookings and payments. They care about how fast and easy the system is to use.
- **System Administrators**: People who maintain the system. They need to make sure it works properly and fix issues.
- **Project Managers/Investors**: People funding the system. They want to know if the system will perform well and meet expectations.

- **Owners (Travel Agency Management):** The people who own and run the business. They care about keeping the system running smoothly and making sure it supports their business goals.
- **Developers:** People who build and improve the system. They need to understand how the system runs in order to fix problems and make improvements.

3.3.1 Concerns

The Execution Viewpoint addresses key concerns about the system's behavior and performance during operation. It focuses on system performance, runtime behavior, and component interactions, especially when processing real-world tasks. These concerns ensure the system meets business and technical goals and handles real-time operations effectively. Each concern includes guiding questions to help stakeholders evaluate how the architecture supports the system's performance.

- **System Purpose**: Defines the main objectives of the system, such as booking management, revenue generation, and user engagement.
    - Question: What are the primary functions of the system that must be supported to meet business goals?
- **User Experience (UX):** Focuses on creating an intuitive, easy-to-navigate platform for end users.
    - Question: How can the system's interface and interactions be designed to ensure a smooth and enjoyable user experience?
- **Performance & Scalability**: Ensures the system can handle increasing numbers of users and data over time without sacrificing performance.
    - Question: How can the system be designed to support future growth in terms of traffic, bookings, and data storage?
- **Reliability & Availability**: Addresses the need for the system to be dependable and accessible at all times.
    - Question: What measures should be implemented to ensure the system remains operational and responsive even during peak usage?

- **Security:** Protects the system and user data from unauthorized access and cyber threats.
    - Question: What security protocols must be in place to safeguard user information, transactions, and system integrity?
- **Integration with External Services**: Ensures the system can communicate effectively with third-party services like payment processors or travel service providers.
    - Question: How can the system be integrated with external services while ensuring data consistency and reliability?
- **Maintainability**: Focuses on making the system easy to update and maintain over time.
    - Question: How can the system be built so that future changes, bug fixes, and enhancements can be implemented with minimal disruption?

3.3.2 Typical Stakeholders

This section identifies the key stakeholders for the Execution Viewpoint of the Travel Agency Booking System. They need this view to understand how the system will run and be maintained in real-world environments. It focuses on runtime behavior, component distribution, performance, and the execution environment. Stakeholders use this viewpoint to ensure the system meets performance, scalability, and operational goals during deployment.

1. **End Users:**
    a. **Who**: Customers booking travel services.
    b. **Why**: They need a user-friendly system that allows for quick, secure bookings. The Conceptual Viewpoint helps define the user interfaces and services that make the system intuitive and responsive.
    c. **How**: The CV informs the design of a seamless customer journey, ensuring smooth navigation, easy payment methods, and access to services.

2. **System Operators**
    a. **Who**: Administrators responsible for maintaining system availability and performance.
    b. **Why**: They need the system to be stable and reliable. The Conceptual Viewpoint provides a clear overview of the system's architecture, allowing operators to understand key components and how to monitor them.
    c. **How**: The CV helps in identifying critical system layers, enabling efficient troubleshooting, monitoring, and performance optimization.
3. **Project Managers/Investors**
    a. **Who**: Those overseeing the project and ensuring it meets business objectives.
    b. **Why**: They are focused on cost-effectiveness and scalability. The Conceptual Viewpoint outlines the system's modular design, helping stakeholders understand how the system can scale and adapt to future demands.
    c. **How**: The CV clarifies how various components interact and how resources are allocated, allowing managers to gauge system growth potential and investment needs.
4. **Owners (Travel Agency Management)**
    a. **Who**: The management team overseeing the business's growth.
    b. **Why**: They need the system to support business goals, such as customer satisfaction and growth. The Conceptual Viewpoint helps ensure the system is designed to scale with the business and provide the services customers expect.
    c. **How**: The CV provides insight into how the system will handle increased bookings, how it integrates with other systems (like payment processors), and how it supports marketing efforts.
5. **Developers**
    a. **Who**: The team building and maintaining the system.
    b. **Why**: Developers need to understand the system's structure to write efficient, maintainable code. The Conceptual Viewpoint offers a high-level view of the architecture and helps developers plan and integrate components.

c. **How**: The CV informs decisions about system design, ensuring that development aligns with business goals and that new features can be added without disrupting the architecture.

3.5 Execution Viewpoint Model Kind

3.5.1 Conventions for Execution Viewpoint Models

The Execution Viewpoint uses specific conventions to create and interpret models, ensuring clarity and consistency.

- **Languages and Notations**
  - **UML (Unified Modeling Language):** Activity diagrams for data flow and process interactions.
  - **Sequence Diagrams:** To visualize how system components interact in real-time.
- **Process**
  - **Mapping Data Flow:** Show how data moves through the system in real-time.
  - **Error Handling Models:** Create diagrams for managing failures, such as retries and fallbacks.
  - **Performance Modeling:** Use simple techniques like queuing theory to predict system behavior under heavy loads.
- **Metamodel Elements**
  - **Entities:** Users, system services (e.g., booking, payment).
  - **Attributes:** Key performance metrics like response time or error rate.
  - **Relationships:** Interactions between system components and users.
  - **Constraints:** Limits on response time or service availability.

3.5.2 Operations on Models

- **Creating Models:** Use templates to quickly build diagrams for booking or payment workflows.
- **Understanding Models:** Train stakeholders to read diagrams for clear insights into system behavior.
- **Analyzing Models**: Simulate scenarios like high traffic to test performance and reliability.
- **Implementing Models**: Translate workflows into system designs and code.

3.5.3 Correspondence Rules

- **Consistency**: Ensure all diagrams follow the same data flow and process patterns.
- **Alignment**: Models must match other system views, like structural or behavioral diagrams.
- **Performance Validation:** Models should meet performance goals, such as fast response times.

3.6 Operations on Views

The Execution View includes methods for building, analyzing, and applying models:

- **Building Views**
  - Use templates to create workflows for key processes like bookings and payments.
  - Highlight critical paths to guide development.
- **Understanding Views**
  - Simplify diagrams to help stakeholders interpret workflows.
  - Use tools to simulate processes and identify weak points.
- **Analyzing Views**
  - Test diagrams under heavy traffic to check system reliability.
  - Evaluate scalability for future growth.

- **Applying Views**
  - Convert workflows into design tasks.
  - Define how external services like payment gateways integrate into the system.

3.7 Correspondence Rules

- **Structural and Execution Alignment**
  - Components in the execution view (e.g., services) should match those in the structural view.
- **Behavioral and Execution Alignment**
  - Real-time workflows must match use cases and expected system behavior.
- **Performance Validation**
  - Ensure all processes meet system performance goals, like fast response times or low error rates.
  - By following these simplified conventions and rules, the execution viewpoint provides a clear and actionable way to model and understand system behavior during runtime.

3.1 Allocation Viewpoint

3.2 Overview

The Allocation Viewpoint describes how the Travel Agency Booking System's software components are deployed onto physical and virtual resources. It highlights the infrastructure needed to support backend operations, such as user booking management, notification services, and integrations with third-party APIs. This view focuses on the distribution of components for performance, scalability, and reliability.

Key Points:

- **Deployment Topology**: Illustrates the mapping of modules to physical or virtual machines.
- **Integration Points**: Specifies connections to external systems like event APIs and notification gateways.
- **Resource Distribution**: Shows the allocation of tasks to different servers, ensuring optimal load handling and fault tolerance.

3.3 Concerns and Stakeholders

Concerns:

- **Performance**: Ensuring the infrastructure can handle peak traffic demands.
- **Reliability**: Minimizing downtime with redundant systems and failover mechanisms.
- **Security**: Protecting sensitive user data during processing and transmission.
- **Scalability**: Supporting growing user bases and increasing booking volume.
- **Cost-Effectiveness**: Balancing resource allocation with operational costs.

Stakeholders:

- **System Administrators**: Require detailed deployment knowledge for monitoring and troubleshooting.
- **Developers**: Need to understand the infrastructure to ensure compatibility and optimal module performance.
- **Owners (Travel Agency Management)**: Concerned with system reliability and uptime to maintain customer trust.
- **Project Managers/Investors**: Focused on budget adherence and scalability to align with business goals.

3.3.1 Concerns

The Allocation Viewpoint addresses critical operational concerns, ensuring the system runs smoothly in real-world environments. These concerns guide infrastructure design to support both business and technical goals.

Key Concerns:

1. **Performance**:
    - **Question**: How does the system ensure efficient processing of bookings and notifications during peak traffic?
    - **Solution**: Use of load balancers and caching mechanisms for critical modules.
2. **Scalability**:
    - **Question**: Can the infrastructure accommodate future increases in user activity?
    - **Solution**: Deploy microservices on a cloud platform with auto-scaling capabilities.
3. **Reliability**:
    - **Question**: How does the system handle failures in individual components or APIs?
    - **Solution**: Implement redundancy and failover for critical services.

4. **Integration Feasibility**:
    - **Question**: How effectively can the system interact with external APIs and gateways?
    - **Solution**: Use middleware for seamless communication with third-party services.
5. **Cost-Effectiveness**:
    - **Question**: Are resources utilized efficiently to minimize operational expenses?
    - **Solution**: Modular deployment to allow cost-effective scaling of individual components.

3.3.2 Typical Stakeholders

1. **System Administrators**:
    - **Who**: Maintain and monitor system operations.
    - **Why**: Need clarity on infrastructure for effective troubleshooting.
    - **Focus**: Reliability, monitoring tools, failover mechanisms.
2. **Developers**:
    - **Who**: Build and optimize backend modules.
    - **Why**: Require details on hosting and dependencies.
    - **Focus**: Efficient module integration and resource utilization.
3. **Owners (Travel Agency Management)**:
    - **Who**: Oversee the agency's operations.
    - **Why**: Require system uptime to maintain customer satisfaction.
    - **Focus**: Reliability, scalability, and alignment with business needs.
4. **Project Managers/Investors**:
    - **Who**: Oversee project funding and delivery.
    - **Why**: Concerned with cost control and long-term scalability.
    - **Focus**: Cost-effective deployment and resource allocation.

3.5 Allocation Viewpoint Model Kind

3.5.1 Conventions for Allocation Viewpoint Models

The Allocation Viewpoint employs specific conventions for modeling resource deployment and distribution, ensuring stakeholders understand how components are allocated across physical and virtual environments.

- **Languages and Notations:**
  - **Package Diagrams (UML):** Use package diagrams to show how software components are organized into modules and deployed across various physical or virtual resources.
  - **Deployment Diagrams (UML):** Visualize how software components are mapped to hardware resources and demonstrate system deployment.
  - **Resource Allocation Charts:** Depict the distribution of workloads across servers or cloud instances.
- **Process:**
  - Component Mapping: Create deployment diagrams to represent where each component runs.
- **Metamodel Elements:**
  - **Entities**: Hardware resources (e.g., servers, cloud instances), software modules.
  - **Attributes**: Deployment settings, resource capacity, and scalability limits.
  - **Relationships**: Connections between modules and resources or external systems.
  - **Constraints**: Ensure sufficient redundancy and fault tolerance.

3.5.2 Operations on Models

- **Creating Models:**
  - Use predefined templates to quickly build deployment and topology diagrams.

    - Incorporate performance metrics into resource charts for accurate allocation modeling.
- **Understanding Models:**
    - Train stakeholders on interpreting deployment diagrams and resource charts.
    - Annotate diagrams to clarify roles of specific resources in supporting components.
- **Analyzing Models**:
    - Simulate load distribution to evaluate system scalability and fault tolerance.
    - Validate network topology diagrams for secure and reliable integration points.
- **Implementing Models**:
    - Translate diagrams into cloud or on-premise infrastructure configurations.
    - Deploy monitoring tools to track resource utilization based on allocation models.

3.5.3 Correspondence Rules

- **Consistency:** Ensure that deployment diagrams align with component dependencies and functional requirements.
- **Alignment:** Resource allocation must match the performance and scalability goals outlined in other views.
- **Validation:** Models should be stress-tested to confirm load handling and fault tolerance.

3.6 Operations on Views

- **Building Views**
    - Create deployment diagrams to show how modules are allocated across resources.
    - Highlight integration points with third-party services in network topology diagrams.

    - Use layered views to distinguish between physical and virtual resources.

- **Understanding Views**
    - Simplify diagrams for stakeholders unfamiliar with technical deployment details.
    - Use annotations and legends to explain resource roles and their significance.

- **Analyzing Views**
    - Evaluate resource allocation for bottlenecks under peak traffic conditions.
    - Analyze failover mechanisms and redundancy to ensure reliability.

- **Applying Views**
    - Implement resource allocation strategies in cloud or on-premise infrastructure.
    - Use deployment views as references for system monitoring and optimization.

3.7 Correspondence Rules

- **Structural and Allocation Alignment:**
    - Ensure deployment diagrams reflect the modular structure of the system.
- **Behavioral and Allocation Alignment:**
    - Validate that allocated resources support real-time workflows and system behavior.
- **Performance Validation:**
    - Test that resource distribution meets performance metrics, such as response times and throughput under load.

4 Views +

4.1 View: Conceptual View

The Conceptual View provides a high-level understanding of the system's structure, focusing on its main components and their interactions.

4.1.1Key Components

1. Booking Subsystem

- **BookingManagerComponent**: Manages event and hotel reservations.
  - Provided: Reserve or view or search booking,
  - Required: Fetch available events and hotels (from external services).
- **BookingRecommendationComponent**: Suggests bookings based on history.
  - Provided: Generate recommendations.
  - Required: Fetch booking details.

2. Notification Subsystem

- **NotificationQueueManager**: Handles notification queuing.
  - Provided: Queue a notification.
  - Required: Retrieve templates (from Notification Template Manager).
- **NotificationSender**: Sends queued notifications.
  - Provided: Deliver a notification.
  - Required: Fetch queued notifications.
- **NotificationTemplateManager**: Manages notification templates.
  - Provided: Create and retrieve templates.

3. User Management Subsystem

- **AuthenticationComponent**: Handles user login/logout.
  - Provided: User session authentication.

- **UserProfileComponent**: Manages user data.
  - Provided: Retrieve and update user details.
  - Required: Fetch user session form auth

4. Dashboard Subsystem

- **BookingSummaryComponent**: Summarizes current and future bookings.
    - Provided: View booking summary.
    - Required: Booking details.
- **NotificationSummaryComponent**: Displays user notifications.
    - Provided: Fetch unread and all notifications.
    - Required: Notification history and user details.

Simplified Subsystem-Level Interactions

- **Dashboard** integrates with Booking, Notification, present an aggregated view.
- **Booking Subsystem** offers booking data to the Dashboard and takes inputs from external services.
- **Notification Subsystem** manages and sends notifications while interacting with templates and preferences.
- **User Management Subsystem** authenticates users and provides profile and preference data.

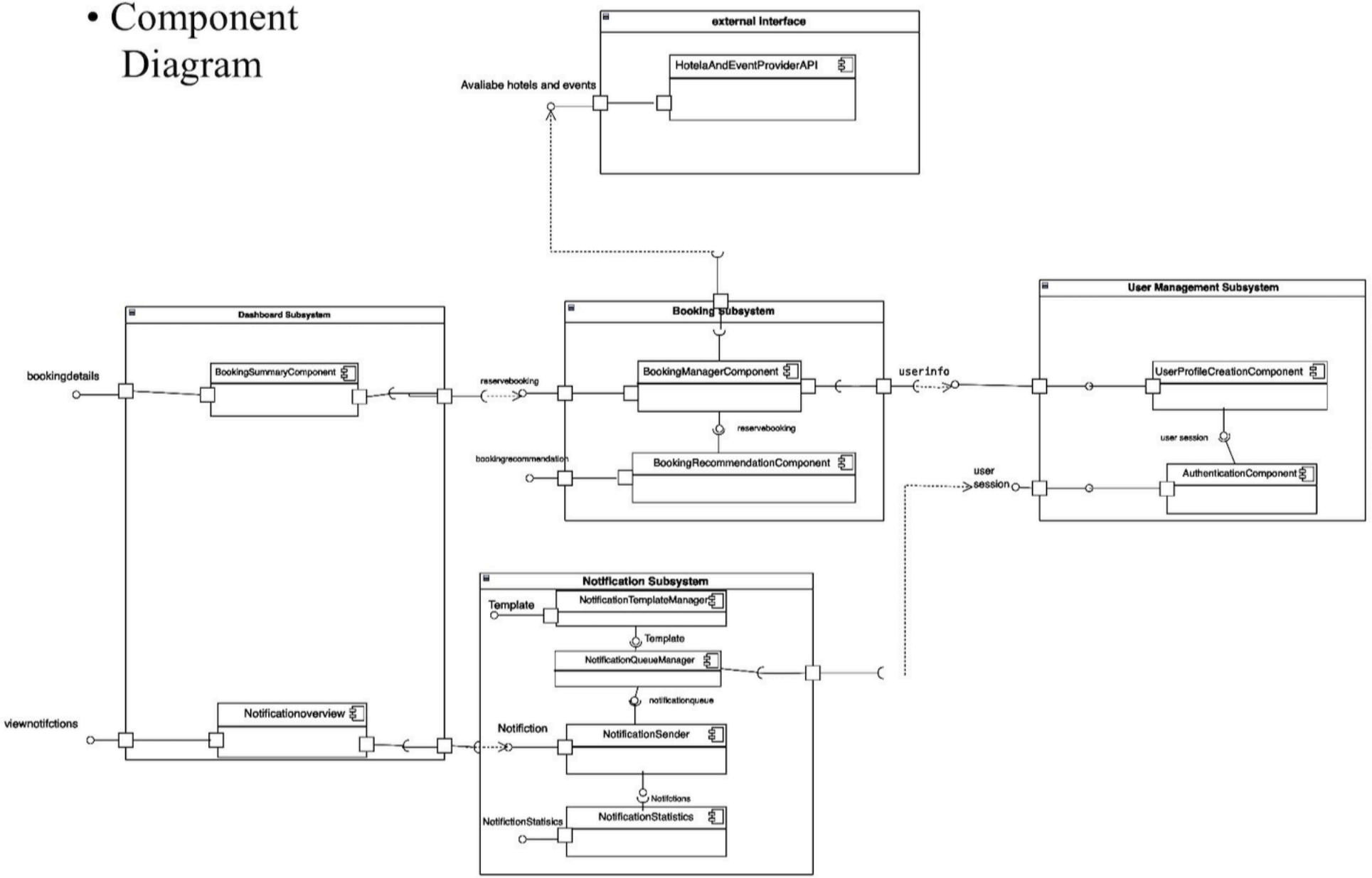
4.1.2 Model Name

- **Model Name:** Conceptual Backend Architecture Model
- **Version Identification:**
    - Model Version: 1.0
    - Version Date: December 2024
- **Governing Model Kind:**
    - Model Kind: Component Diagram
- **Description**: Adheres to conventions for identifying high-level components and their interactions. This model outlines the logical structure without specific implementation details.

4.1.3 Known Issues with View

- **Simplification**: The diagram omits technical implementation details to make it accessible for non-technical stakeholders.
- **Future Updates:** As new features (e.g., flight bookings) are added, the diagram will require updates to include the new components.
- **Integration Details:** Connections to external APIs are represented at a high level without specifying protocols or technical specifics.

- Component Diagram



4.1 View: Execution View

4.1.1 Models

The Execution View uses an Activity Diagram to show how the system works in real-time. This diagram helps explain:

1. **How Data Flows:**
   - Shows how information, like a booking request or payment, moves through the system.
2. **How Components Work Together:**
   - Illustrates how parts of the system, such as the User Interface, Payment Gateway, and Booking Engine, interact to complete tasks.
3. **How Errors Are Handled:**
   - Includes paths for handling problems, like failed payments or network issues, and shows backup plans (e.g., retries).

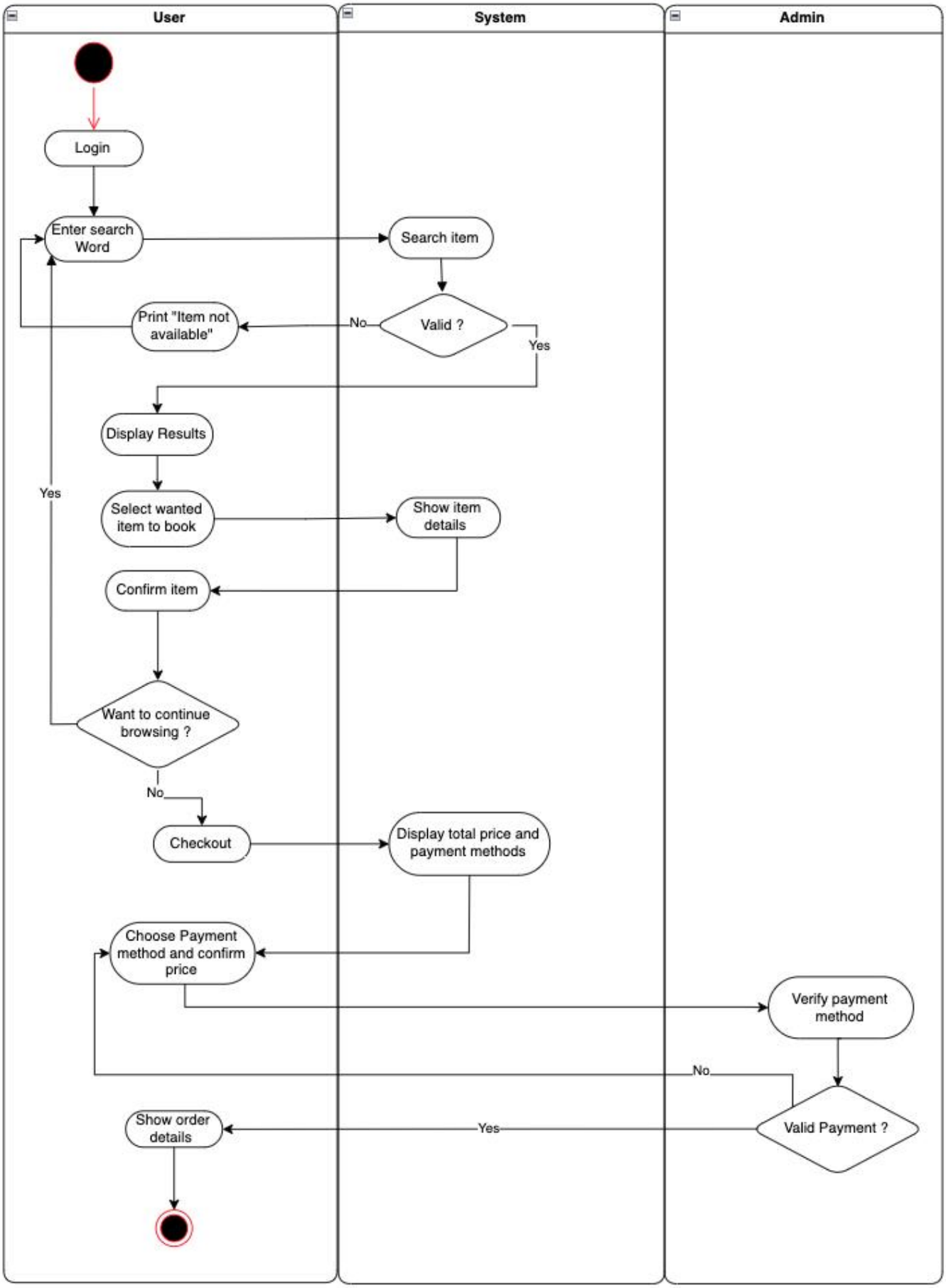
4.1.2 Model Name

- **Model Name:** Execution Activity Diagram for Hotel Room Booking Process
- **Version:** 1.0
- **Governing Model Kind:** Activity Diagram (UML)
- **Description:** This model outlines the flow of activities in the hotel room booking process. It shows the user journey from login, through room search, availability check, and payment processing, to booking confirmation. The diagram highlights system responses to user actions and addresses possible exceptions, ensuring an efficient and seamless booking experience.

4.1.3 Known Issues with View

Some areas in the model need improvement:

- **Error Handling:** Not all error scenarios, like external service failures, are fully shown.
- **Scalability**: The diagram doesn’t yet cover how the system handles heavy traffic.
- **External Services:** More detail is needed on how the system connects to outside services like payment processors.


User
System
Admin
Login
Enter search Word
Search item
Valid ?
No
Print "Item not available"
Yes
Display Results
Yes
Select wanted item to book
Show item details
Confirm item
Want to continue browsing ?
No
Checkout
Display total price and payment methods
Choose Payment method and confirm price
Verify payment method
No
Valid Payment ?
Yes
Show order details

4.1 View: Allocation View

The Allocation View uses Package Diagram for deployment.

Key Elements:

- **Modules/Packages:** Each software module is represented as a package, indicating its role within the system.
- **Dependencies:** Dependencies between packages and modules are displayed to show interactions between components.
- **Deployment Topology:** Illustrates the distribution of software components onto physical or virtual machines.

4.1.2 Model Name

- **View Name:** Allocation Viewpoint
  - **Model 1:** Package Diagram for Deployment
- **Version:** 1.0
- **Governing Model Kind:** Package Diagram (UML)
- **Description**: This view outlines how the Travel Agency Booking System's software components are allocated to physical or virtual resources, focusing on the deployment topology and infrastructure needed to support backend operations. It highlights the distribution of components for performance, scalability, and reliability.

4.1.3 Known Issues with View

Some areas in the model need improvement:

- **Error Handling:** Not all error scenarios, like external service failures, are fully shown.
- **Scalability**: The diagram doesn’t yet cover how the system handles heavy traffic.
- **External Services:** More detail is needed on how the system connects to outside services like payment processors.

We made an abstract class diagram to classify classes then grouped these classes into logical packages as follow:

Packages:

1. **User Management**

- Classes:

  - User (Base Class)
  - Customer (Inherits from User)
  - Admin (Inherits from User)

2. **Dashboard**

- Classes:

  - Dashboard (Main Class for User Interaction)
    - **Responsibilities**: Interface for users to manage bookings, view notifications, and access statistics.

3. **Service Management**

- Classes:

  - ServiceBooked (Base Class)
  - Hotel (Inherits from ServiceBooked)
  - Event (Inherits from ServiceBooked)
  - Room (Associates with Hotel)
    - **Responsibilities**: Manages services like hotels and events.

4. **Booking Management**

- Classes:

  - Booking
  - HotelBooking
  - EventBooking
    - **Responsibilities:** Handles all booking-related operations.

5. **Notification Management**

- Classes:
  - Notification
  - NotificationSystem
  - NotificationStatistics
    - **Responsibilities**: Sends notifications, maintains statistics, and manages notification templates and their placeholders directly within the package.

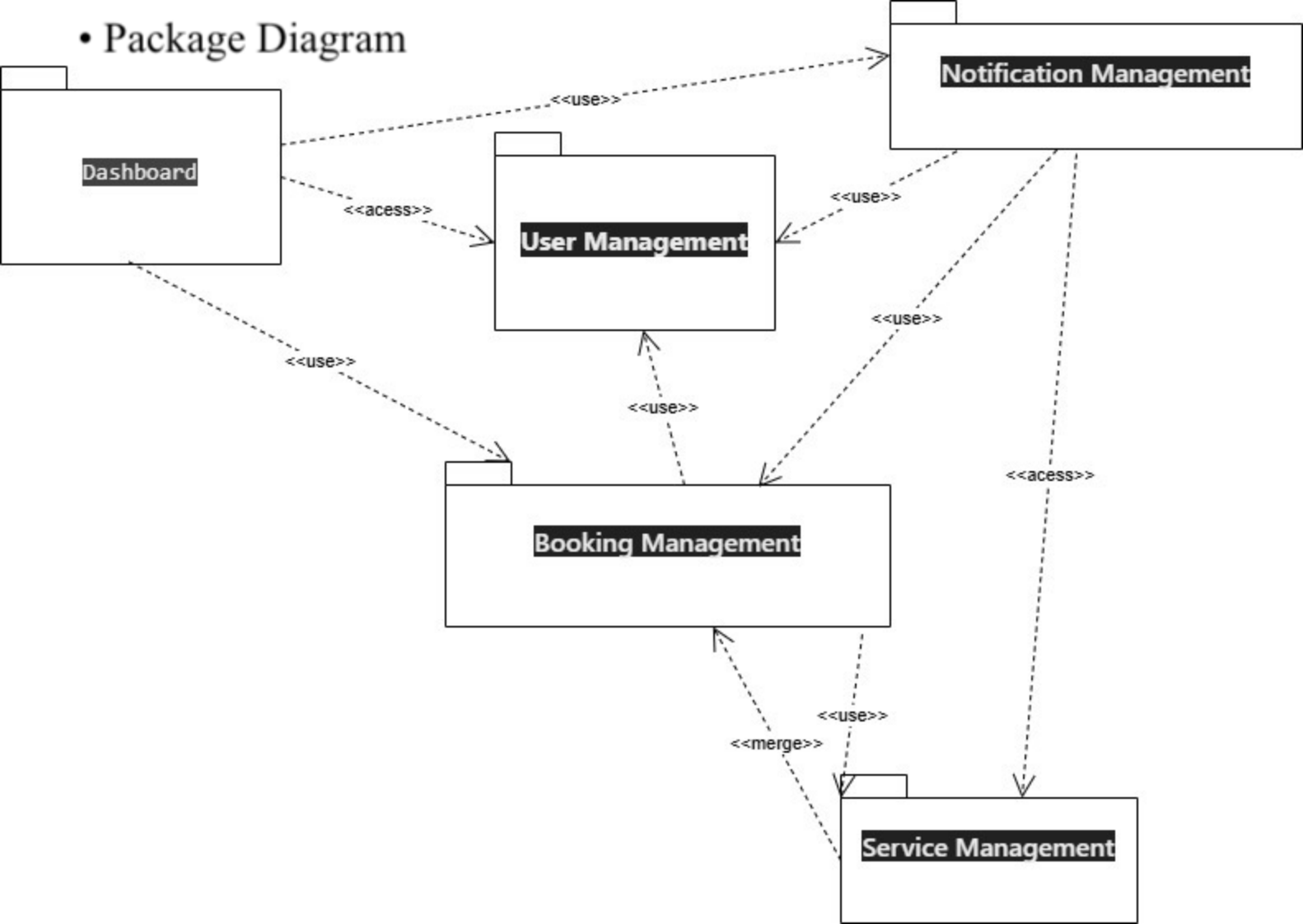
Relations between packages:

1. **Dashboard**:
   a. **Dependencies**: Relies on **Booking Management** to display and manage user bookings and **Notification Management** for updates on bookings, promotions, and events.
   b. **Access**: Retrieves user data from **User Management** to offer personalized experiences.

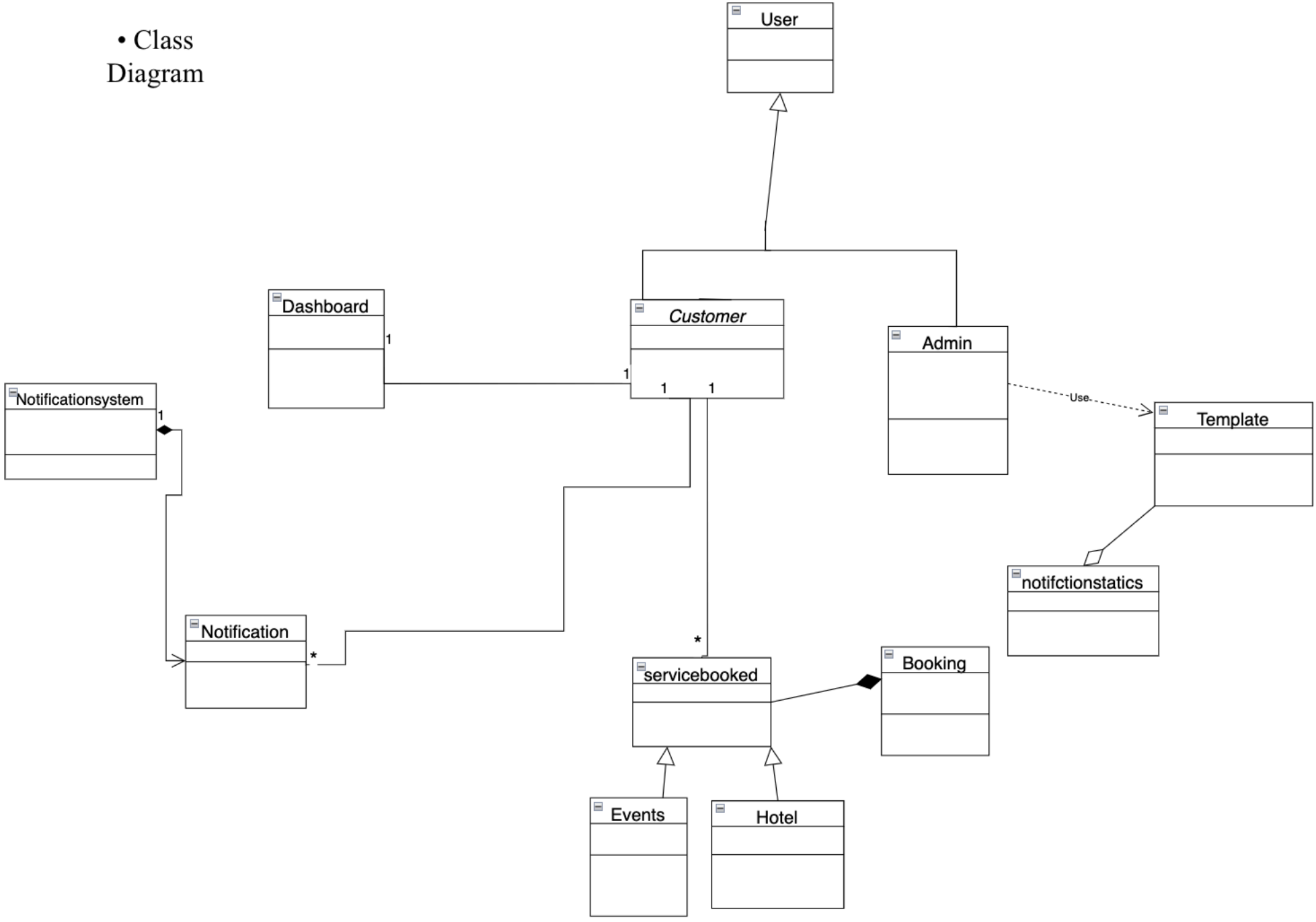
2. **Booking Management**:
   a. **Dependencies**: Uses **Service Management** for booking services (e.g., hotels, events) and **User Management** to link bookings to users and send related notifications.

3. **Notification Management**:
   a. **Dependencies**: Relies on **User Management** for user contact details and **Booking Management** for booking-triggered notifications.
   b. **Access**: Includes data from **Service Management** in notifications for context (e.g., hotel names, event details).

• Package Diagram



• Class Diagram



5 Consistency and Correspondences

5.1 Known Inconsistencies

Currently, there are no identified inconsistencies in the architecture description. The design ensures alignment between the views, models, and system requirements. However, potential issues may arise during the integration of third-party APIs for events and notifications. These will be monitored and resolved during implementation.

5.2 Correspondences in the AD

This architecture description maintains correspondence between its elements to ensure a cohesive design. The following are the identified correspondences:

- **Stakeholders and Concerns**: The architecture addresses primary stakeholder concerns, such as system reliability, scalability, and maintainability, by adhering to a modular design approach.
- **Viewpoints and Models**: The functional view is consistent with the information view, ensuring the Notification module processes user data (e.g., email and phone numbers) securely and effectively.
- **API Design and Module Interactions**: The API endpoints correspond directly with the operations of the Booking and Notification modules. For example, hotel bookings trigger the Notification module to recommend events via APIs.

5.3 Correspondence Rules

The following rules govern the correspondences in this architecture description:

1. **API and Database Consistency**: Any API data fetch must map to a database schema or external API schema, ensuring data integrity between external systems (e.g., hotel providers) and the internal backend.
    - Rule holds: Ensured by schema validation.
2. **Notification Triggers**: Events such as user bookings must reliably trigger notifications via the Notification module.

    - Rule holds: Enforced by event-driven middleware.
3. **Data Security and Privacy**: All user data transmitted between modules or with third-party APIs must adhere to security protocols (e.g., HTTPS).
    - Rule holds: Verified through compliance checks.
4. **Synchronization between Modules**: The Booking module and Dashboard must synchronize booking statuses to reflect real-time updates in user dashboards.
    - Rule holds: Achieved through message queues.