Architecture description template
for use with ISO/IEC/IEEE 42010:2011

Architecture Description of Local Travel Agency for Backend System

“Bare bones” edition version: 2.2

Template prepared by:
Rich Hilliard
r.hilliard@computer.org

Contents

1. Introduction

This chapter describes the architecture for a backend system designed to support a travel agency's online platform. The system will allow users to book hotel rooms, view and book tickets for local events, and receive notifications about their bookings and relevant events. The architecture focuses on providing a scalable, efficient, and user-friendly backend that interacts with the frontend via APIs.

1.1 Identifying information

- **Architecture Name:** Travel Agency Backend Architecture.
- **System of Interest:** Online Booking System for a Travel Agency**.**
- **System Description:** This system is designed to facilitate online booking for users, including hotel reservations and local event ticket purchases. It includes features like user authentication, event and hotel search, recommendations, and notifications.

1.2 Supplementary information

Date of Issue and Status: 27 Nov 2024

Authors: Mohamed khaled,Lodgy Magdy, Rodyna Amr, Noureen Mohamed, Judi Wael

Reviewers: TA Hossam

Approving Authority: …

Issuing Organization: Cairo University

Change History: 2 dec 2024

Scope: This document describes the backend architecture for the travel agency's online booking system, which includes hotel and event booking functionality, notifications, and API integrations.

Context: The system is designed to be integrated into an existing frontend website and will handle user requests via API calls.

Glossary:

Gateway	A tool that connects different systems or services, like processing payments.
Docker	A program that helps run apps in small, portable units called containers.
AES-256	A strong method for encrypting data to keep it secure.
Dynamic Pricing	A way of changing prices based on demand, like charging more during busy times.
PCI DSS-Compliant Gateways	Payment systems that follow strict security rules to protect card details.
GDPR	A law in Europe that protects people's personal data and privacy.
CCPA	A law in California that gives people more control over their personal data.

Version Control Information: Git & github, Trello.

References:

- https://developers.google.com/solutions/content-driven/backend/architecture#:~:text=Backend%20architecture%20refers%20to%20how,incoming%20requests%20and%20create%20responses.
- https://www.geeksforgeeks.org/microservices/

1.3 Other information

1.3.1 Overview

The architecture of the travel agency backend system is designed to provide a seamless and efficient experience for users booking hotels and events. The backend will manage user authentication, hotel and event data, and send notifications to users. The system will interface with external APIs to fetch event details, process bookings, and send out notifications.

- **Purpose:** The primary purpose of the backend system is to facilitate user bookings and provide notifications about events and hotel reservations. The system should allow for real-time updates and scalable handling of user requests.
  - **Scope:** This architecture focuses on the backend, including user management, hotel and event booking systems, and the notification module. It will interact with external systems through APIs.
  - **Context:** The system will integrate with an existing frontend to provide a complete online booking experience. The backend will process requests made from the frontend, update data in the database, and provide real-time feedback via notifications.

1.3.2 Architecture evaluations

Usability	Provide intuitive navigation and minimal steps for booking.	The system ensures usability with a responsive design that works smoothly on both mobile and desktop. Navigation is simplified with clear actions like searching, viewing details, and booking, making tasks easy and quick. User progress is synced across devices, allowing seamless transitions. Instant feedback at every step keeps users confident and in control, providing an efficient, user-friendly experience.
Scalability	The system should be built to handle more users easily and should be able to update without causing downtime.	The system uses Docker to handle more traffic. If more users come in, the system can add new servers without affecting the users. When updates are made, they are done gradually so users can still book hotels and tickets without interruption.

Reliability	Manage API failures from external providers gracefully.	If a hotel API fails while a user is browsing listings, the system will show cached results or notify the user that the information is temporarily unavailable. This ensures users can continue browsing and booking without disruption, while the system's backend team works to resolve the issue.
Availability	The system will work continuously, even during heavy traffic or if something goes wrong.	If the main database stops working, the system will automatically switch to a backup database. This ensures users can still book hotels and buy tickets without any problems. Once the main database is fixed, it will update itself with the latest information from the backup.
Security	Data is protected by encryption, and access to sensitive areas is restricted to authorized users only.	When users enter sensitive information, such as payment details, the system ensures that the data is encrypted to keep it secure. Additionally, to protect access to important features, the system requires extra verification for admin accounts, like confirming their identity with a second step, ensuring only authorized users can access sensitive information.
Modifiability	The system should be flexible enough to support new features and easy to maintain over time.	A new feature, such as a dynamic pricing module, is added to the system to offer users customized pricing based on demand. The modular system allows developers to integrate the new feature with minimal changes to existing features, ensuring the system remains stable and efficient.
Data Integrity	Data should be accurate, free of duplicates, and easy to search quickly and precisely.	If a hotel updates its availability, the change is reflected instantly on the user interface, and there are no duplicate listings. The search system is optimized to return relevant hotel results quickly, ensuring users can book their stays without delays.

1.3.3 Rationale for key decisions

- **Event and Hotel Integration via API:** The system relies on external event APIs and hotel provider APIs to fetch real-time data. This was chosen to minimize the need for maintaining large amounts of data within the backend and to ensure up-to-date event and hotel availability.

- **Notification System:** A robust notification module is essential to keep users informed about bookings, promotions, and relevant local events. Email and SMS-based notifications ensure a high level of user engagement and satisfaction.
- **Database Design:** A relational database will be used to store user data, bookings, and preferences, as it offers the flexibility to manage complex relationships between users, events, and hotels.
- **API Design:** RESTful APIs will be used to ensure that the frontend can easily communicate with the backend system for hotel searches, event bookings, and notifications.

2. Stakeholders and concerns

2.1 Stakeholders

- **Client**: Clients are individuals who utilize the travel agency's application to book hotel rooms and tickets for local events. Their primary goal is to seamlessly plan and manage their travel and event activities. Key responsibilities and interactions include:

  - ☐ **Booking Management:** Searching, filtering, and booking hotel rooms (single, double, or family) and tickets for local events based on preferences and availability.

  - ☐ **Dashboard Interaction:** Accessing a personalized dashboard to view, manage, and print bookings. The dashboard also provides personalized event recommendations and notifications.

  - ☐ **Account Management:** Creating and managing user accounts, including password resets and profile updates, to enable secure access to booking features.

- **Notification Handling:** Receiving timely updates about booking confirmations, event recommendations, and promotional offers through emails, SMS, or in-app notifications.

- **System Testers:** System testers are integral members of the quality assurance team, specializing in evaluating backend systems to ensure they meet both functional and non-functional requirements. Their responsibilities include:

- **Functional Testing:** Verifying that the system behaves as intended and meets the specified requirements, such as correct data processing, API responses, and expected system interactions.

- **Non-Functional Testing:** Assessing system attributes like performance, scalability, reliability, and security to ensure optimal operation under various conditions.

- **Test Design and Execution:** Creating detailed test plans, scenarios, and cases tailored to the backend architecture, followed by methodical execution.

- **Bug Identification and Reporting:** Detecting system defects, documenting them clearly, and collaborating with developers to resolve issues.

- **Regression Testing:** Ensuring that recent changes or updates do not negatively affect existing functionality.

- **Travel Agency Admins:** Travel agency admins are staff responsible for managing customer bookings, addressing queries, and ensuring smooth system operation. They play a critical role in maintaining customer satisfaction and operational efficiency. Key responsibilities include:

    - ☐ **Booking Oversight**: Monitoring and confirming hotel room and event ticket reservations, resolving overbooking issues, and managing cancellations or modifications.

    - ☐ **Customer Support**: Providing assistance to end users by addressing inquiries, resolving complaints, and troubleshooting booking-related problems.

    - ☐ **System Monitoring**: Ensuring the backend system functions properly, including checking notification queues and addressing issues with unsent or failed notifications.

- **System Designers:** System designers are specialists responsible for crafting specific modules or subsystems within a larger software architecture. Their work ensures that each component is well-structured, efficient, and seamlessly integrated into the overall system. Key responsibilities include:

  - ☐ **Module Design**: Developing detailed designs for specific subsystems, such as the notification module, API interaction logic, or database management layer, tailored to meet project requirements.

  - ☐ **Architectural Alignment**: Ensuring that each subsystem adheres to the overall system architecture, maintaining consistency, scalability, and reliability.

  - ☐ **Requirement Analysis**: Collaborating with stakeholders to translate functional and technical requirements into clear, actionable design blueprints.

  - ☐ **Technical Documentation**: Producing comprehensive design documents, diagrams, and specifications to guide developers during implementation.

  - ☐ **Optimization and Innovation**: Identifying areas for improvement, optimizing performance, and integrating innovative solutions to enhance system functionality.

- **Backend Development Team:**
  The Backend Development Team is made up of software developers who specialize in creating and managing the server-side logic, databases, APIs, and integrations that enable the platform to function effectively. They work closely with **frontend developers**, **DevOps engineers**, and **quality assurance (QA) teams** to ensure a seamless user experience and efficient system performance.

  - ☐ **System Design and Architecture**: Designing and building the backend structure, including databases, server-side logic, and interactions between systems.

  - ☐ **API Development and Integration**: Creating APIs for internal and external services, enabling communication between the platform and third-party services (e.g., payment gateways, travel partners).

- ☐ **Database Management**: Managing data storage, ensuring data integrity, optimizing queries, and implementing security measures to protect sensitive data.
- ☐ **Authentication and Authorization**: Implementing secure login and access control mechanisms to protect user data and ensure proper permissions.
- ☐ **Performance Optimization**: Enhancing system efficiency through load balancing, caching, and optimizing server-side code to handle high traffic and ensure fast responses.
- ☐ **CI/CD (Continuous Integration/Continuous Deployment)**: Automating the process of deploying code, ensuring smooth updates, and maintaining version control.

- **Database Administrators:** Database administrators (DBAs) are the people responsible for managing and maintaining the system's database. They ensure that all the data is stored securely, is easy to access, and is updated correctly.

  - ☐ **Organize and store data:** DBAs create and organize the structure of the database to store user information, bookings, hotel details. They make sure that data is stored efficiently and is easy to access when needed.
  - ☐ **Ensure data security:** They protect sensitive data, such as users' personal information and payment details, by using encryption and security protocols to prevent unauthorized access.
  - ☐ **Backup data:** They regularly backup the database so that if there is a system failure or data loss, the information can be recovered quickly.
  - ☐ **Fix data issues:** If there are any issues with the data, such as missing or incorrect information, the database administrator fixes them to maintain data accuracy.

- **Travel Agency Owners:**
  Travel agency owners are the business leaders who manage the overall direction of the travel agency. They make decisions about how the system should help the business grow and meet customer needs.

  - ☐ **Set business goals**: They define what the business wants to achieve with the system, such as attracting more customers, increasing bookings, or expanding into new markets.

  - ☐ **Make decisions about features**: They decide which features should be added to the system based on customer feedback, market trends, and the goals of the business. For example, they may want to add a feature that allows users to book international travel or a feature for discounts and promotions.

  - ☐ **Monitor the system's success**: They keep an eye on how well the system is performing and how it is helping the business grow.

  - ☐ **Ensure user engagement**: They make sure the system provides an excellent user experience so customers keep coming back. This includes ensuring easy navigation, smooth booking, and timely notifications.

- **System Architects:** System architects are the people who design the overall structure of the system. They make sure that the system works efficiently, can handle future growth, and meets the needs of both users and the business.

  - ☐ **Design the system's architecture:** They plan how the different parts of the system (like the backend, frontend, and database) will work together. They decide how to organize and connect all these parts to create a smooth, reliable system.

  - ☐ **Select technologies:** They choose the tools, programming languages, and frameworks that will be used to build the system. They select the best options to ensure the system is reliable, fast, and easy to maintain.

  - ☐ **Focus on performance:** They ensure the system is optimized to handle many requests at once and that it responds quickly to users' actions, like searching for hotels.

❑ **Design for flexibility:** They ensure that the system is flexible, so new features can be added easily in the future.

2.2 Concerns

1. Purpose of the System-of-Interest

- **1.1 Detail:** Provide a seamless platform for users to book hotel rooms and tickets for local events.
  - ❑ **Scenarios:** Users can browse hotel listings and event tickets, compare prices, check availability, and complete bookings without confusion or delays.
- **1.2 Detail:** Establish a comprehensive database system for managing and synchronizing hotel and event listings efficiently.
  - ❑ **Scenarios:** Admins can update inventory in real-time, and users can see live availability.

2. Suitability of the Architecture

- **2.1 Detail:** Support both high-performance requirements (for real-time searches) and high availability (zero downtime).
  - ❑ **Scenarios:** Even during peak traffic, the app maintains its responsiveness and does not crash.
- **2.2 Detail:** Modular architecture to enable adding new features, such as international bookings or loyalty programs.
  - ❑ **Scenarios:** Modules like payment gateways or notification systems can be replaced or upgraded without affecting the entire system.
- **2.3 Detail:** Ensure a consistent user experience across devices (desktop, mobile, tablet).
  - ❑ **Scenarios:** A user starts booking on their phone but finishes it later on their desktop.

3. Feasibility of Construction and Deployment

- **3.1 Detail:** Use scalable backend technologies to accommodate growing user demand.
  - ❑ **Scenarios:** Implement containerized services (like Docker) for faster deployments and scaling.

- **3.2 Detail:** Choose cost-efficient hosting options for initial deployment.
  - ☐ **Scenarios:** Host the application on a cloud provider with a pay-as-you-go model to reduce upfront investment.
- **3.3 Detail:** Minimize downtime during updates through rolling deployments and staging environments.
  - ☐ **Scenarios:** New features or bug fixes are pushed live without interrupting ongoing bookings.

4. Risks and Impacts

- **4.1 Detail:** Manage API failures from external providers gracefully.
  - ☐ **Scenarios:** If a hotel API is unavailable, display cached results or notify users of temporary issues.
- **4.2 Detail:** Avoid negative impacts on user trust and revenue due to system downtime.
  - ☐ **Scenarios:** Implement redundant servers and failover mechanisms to ensure constant uptime.
- **4.3 Detail:** Address security vulnerabilities to prevent data breaches.
  - ☐ **Scenarios:** Encrypt sensitive user data, such as passwords and payment details, with AES-256.

5. Maintenance and Evolution

- **5.1 Detail:** Ensure the system remains adaptable to changes in market trends.
  - ☐ **Scenarios:** Quickly integrate new features like dynamic pricing or hotel recommendations.
- **5.2 Detail:** Maintain and update notification systems for better user engagement.
  - ☐ **Scenarios:** Add push notifications for mobile users in addition to email and SMS.
- **5.3 Detail:** Create a robust system for logging and tracking bugs.
  - ☐ **Scenarios:** Developers have access to detailed logs to diagnose and fix issues quickly.

6. Data Management

- **6.1 Detail:** Ensure database integrity and avoid data redundancy.

- [ ] **Scenarios:** When a hotel updates its availability, the change reflects instantly without creating duplicate entries.
- **6.2 Detail:** Optimize search queries to provide users with fast and accurate results.
  - [ ] **Scenarios:** A user searching for "hotels near city center" receives relevant results within seconds..

7. Notification System

- **7.1 Detail:** Guarantee timely delivery of critical notifications.
  - [ ] **Scenarios:** A user gets booking confirmation emails immediately after payment is completed.
- **7.2 Detail:** Allow users to customize their notification preferences.
  - [ ] **Scenarios:** A user can opt-out of promotional emails but still receive booking updates.

8. User Experience

- **8.1 Detail:** Provide intuitive navigation and minimal steps for booking.
  - [ ] **Scenarios:** A new user can complete a hotel booking within 3 clicks.
- **8.2 Detail:** Offer multilingual support to cater to a diverse user base.
  - [ ] **Scenarios:** A French user sees all labels, buttons, and notifications in French.

9. Security

- **9.1 Detail:** Protect payment data using industry-standard encryption and compliance protocols.
  - [ ] **Scenarios:** Use PCI DSS-compliant gateways for processing payments.
- **9.2 Detail:** Prevent unauthorized access through advanced authentication.
  - [ ] **Scenarios:** Two-factor authentication for admin accounts to secure critical operations.

10. Integration

- **10.1 Detail:** Facilitate seamless integrations with hotel APIs and event ticket platforms.
  - ☐ **Scenarios:** When hotels update room availability, the app syncs instantly.
- **10.2 Detail:** Support third-party integrations like map services and weather forecasts.
  - ☐ **Scenarios:** Show nearby attractions or weather conditions for a user's travel dates.

11. System Monitoring

- **11.1 Detail:** Implement a monitoring system for performance metrics.
  - ☐ **Scenarios:** The admin panel displays live statistics on booking volumes and system health.
- **11.2 Detail:** Set up alert systems for anomalies or failures.
  - ☐ **Scenarios:** Notify the backend team if server response time exceeds 500ms.

12. Compliance

- **12.1 Detail:** Ensure adherence to GDPR, CCPA, or other privacy regulations.
  - ☐ **Scenarios:** Collect explicit consent before storing or processing user data.
- **12.2 Detail:** Maintain audit logs for transparency.
  - ☐ **Scenarios:** All admin actions (e.g., deleting a booking) are logged for compliance reviews.

2.3 Concern–Stakeholder Traceability

	Clients	Travel Agency Owner	Travel Agency Admin	System Designers	System Testers	System Architects	Backend Development Team	Regulatory Authorities	Database Administrator	Hotel Providers	Notification System
Concern 1.1	x	x	x	–	–	–	–	–	–	–	–
Concern 1.2	–	–	x	–	–	–	–	–	x	x	–
Concern 2.1	x	–	–	–	–	x	x	–	–	–	–
Concern 2.2	–	x	–	x	–	x	–	–	–	–	–
Concern 2.3	x	–	–	x	–	–	–	–	–	–	–
Concern 3.1	–	x	–	–	–	x	x	–	–	–	–
Concern 3.2	–	x	–	–	–	–	x	–	–	–	–
Concern 3.3	–	–	–	–	–	x	x	–	–	–	–
Concern 4.1	–	–	–	–	x	x	x	–	–	–	–
Concern 4.2	x	x	x	–	–	–	–	–	–	–	–
Concern 4.3	–	–	–	–	x	–	–	x	x	–	–
Concern 5.1	–	x	–	x	–	–	x	–	–	–	–
Concern 5.2	–	–	x	–	–	–	–	–	–	–	x
Concern 5.3	–	–	–	–	x	–	x	–	–	–	–
Concern 6.1	–	–	x	–	–	–	x	–	x	–	–
Concern 6.2	x	–	–	–	x	–	x	–	–	–	–
Concern 7.1	–	–	x	–	–	–	x	–	–	–	x
Concern 7.2	x	–	x	–	–	–	–	–	–	–	–
Concern 8.1	x	–	–	x	–	–	–	–	–	–	–
Concern 8.2	x	x	–	–	–	–	–	–	–	–	–
Concern 9.1	–	x	–	–	–	–	–	x	x	–	–

	End Users	Travel Agency Owner	Travel Agency Admin	System Designers	System Testers	System Architects	Backend Development Team	Regulatory Authorities	Database Administrato r	Hotel Providers	Notification System
Concern 9.2	–	–	x	–	x	–	–	–	–	–	–
Concern 10.1	–	–	–	x	–	–	x	–	–	x	–
Concern 10.2	x	–	–	x	–	–	–	–	–	–	–
Concern 11.1	–	x	x	–	–	–	x	–	–	–	–
Concern 11.2	–	–	–	–	x	–	x	–	–	–	–
Concern 12.1	–	x	–	–	–	–	–	x	x	–	–
Concern 12.2	–	–	x	–	–	–	–	x	–	–	–

3.Viewpoints+

Conceptual Viewpoint

3.1 Overview

The Conceptual Viewpoint focuses on providing a high-level understanding of the system's structure and purpose. It describes the key components, their relationships, and how they fulfill the system's primary goals. This viewpoint is useful for outlining the major elements of the system without delving into implementation details.

3.2 Concerns and Stakeholders

3.2.1 Concerns

Purpose of the System-of-Interest:

- How does the system enable users to book hotel rooms and event tickets efficiently?
- How does the system ensure real-time data synchronization for hotel and event listings?

Suitability of the Architecture:

- How does the architecture support high performance, modularity, and seamless user experiences?

Integration:

- How will the system integrate with external APIs for hotel and event data?

3.2.2 Typical Stakeholders

Clients: Interested in a smooth and intuitive booking experience.

Travel Agency Owners and Admins: Focused on system reliability, performance, and ease of management.

System Architects and Developers: Concerned with designing and building the system to meet performance and integration needs.

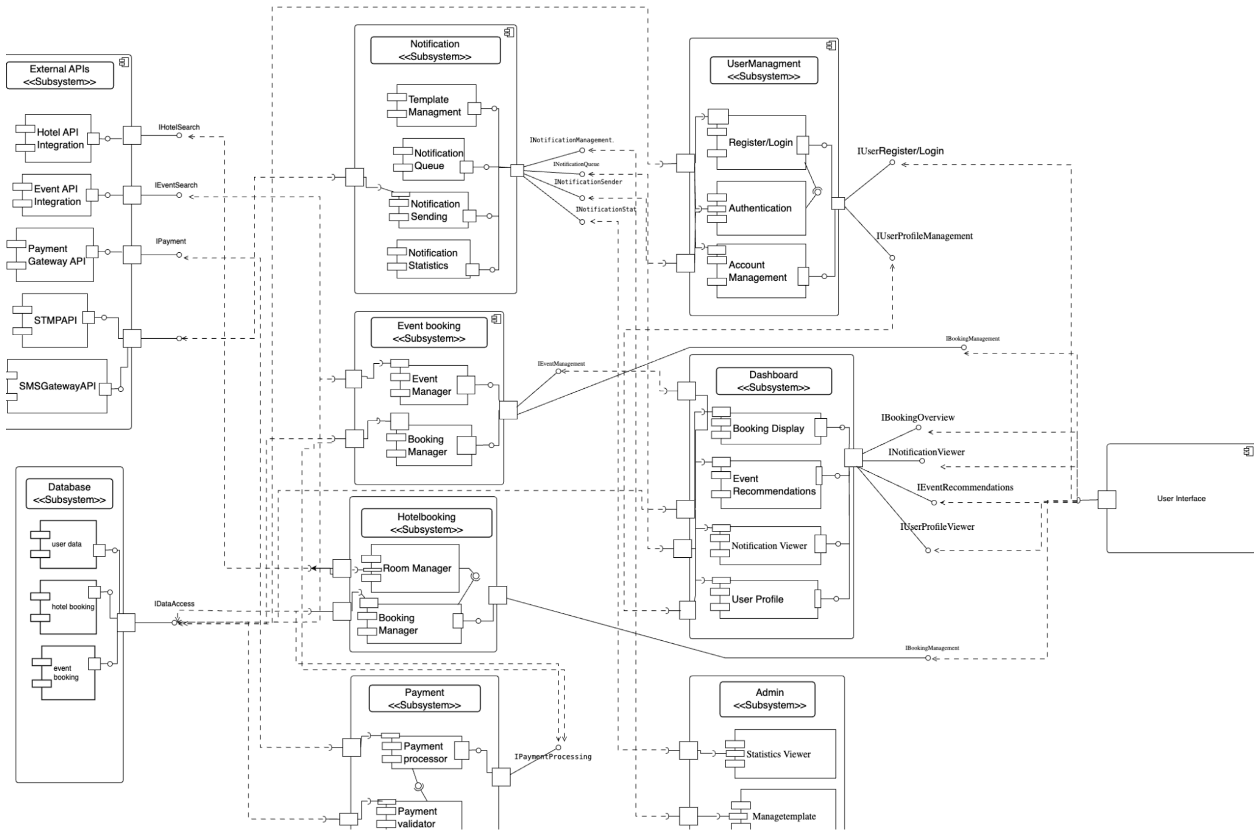
3.3 Model Kinds

The Conceptual Viewpoint includes the following models:

Component Model

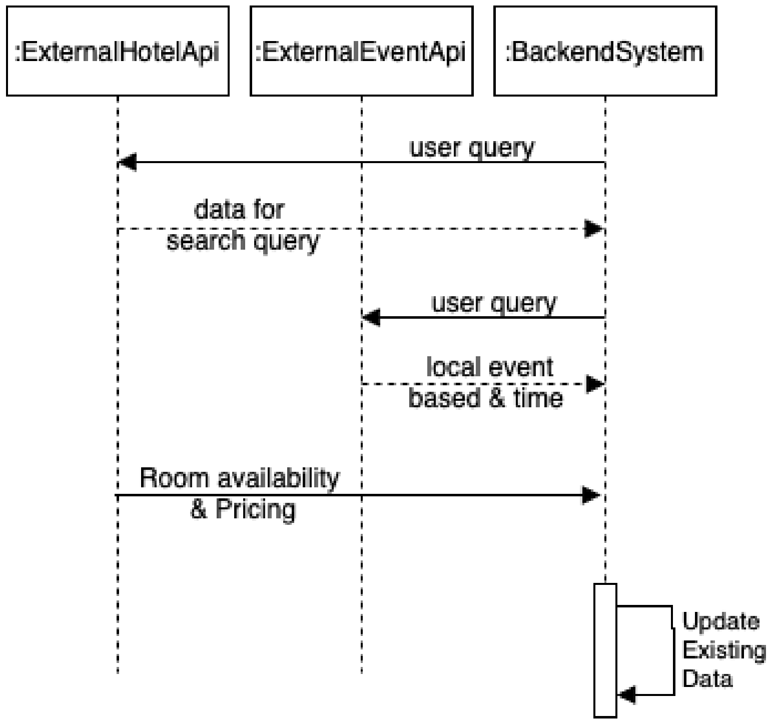
Definition: A high-level representation of the system's core components and their interactions. This model shows the main functional modules and how they communicate internally.

Purpose: To give stakeholders an overview of the system structure and responsibilities of each component.



Definition: Focuses on how the system connects and interacts with external APIs or services. This model describes the data flow between the system and third-party services.

Purpose: To illustrate how the system consumes external data (e.g., hotel availability and event information) and updates its internal processes.



Allocation Viewpoint

3.1 Overview

The Allocation Viewpoint describes how software components are assigned to hardware resources. It ensures efficient resource usage, scalability, and performance by mapping logical components to physical nodes or environments.

3.2 Concerns and Stakeholders

3.2.1 Concerns

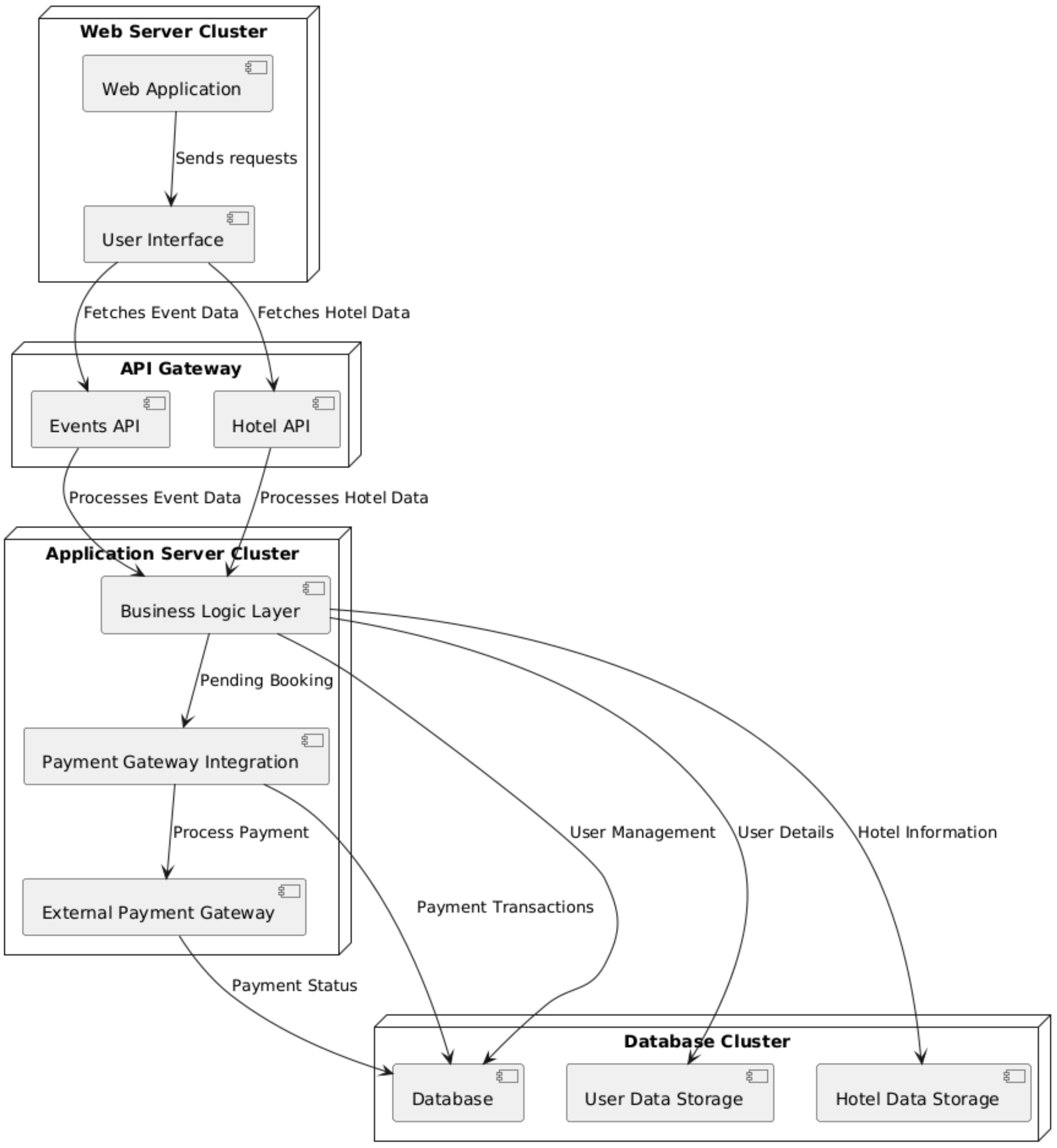
- **Resource Utilization:** How are components distributed across hardware to optimize performance?
- **Scalability:** Can the system scale efficiently by allocating more resources when needed?
- **Redundancy:** How does the system ensure availability and fault tolerance through resource allocation?

3.2.2 Typical Stakeholders

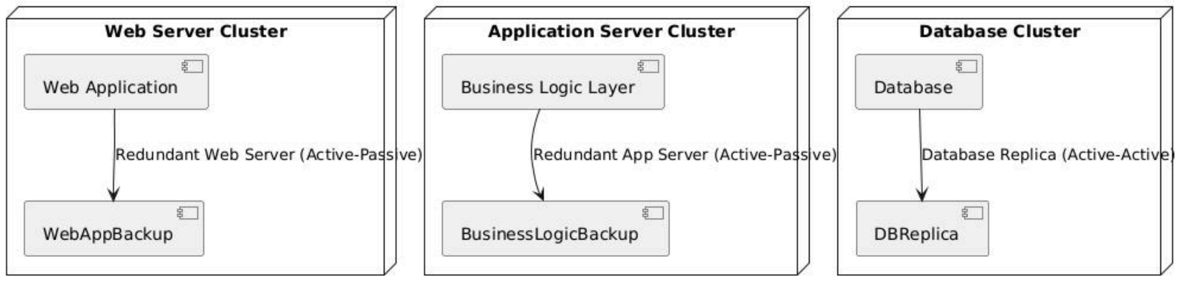
- ➔ **System Administrators**
- ➔ **Infrastructure Engineers**
- ➔ **System Architects**

3.3 Model Kinds

Node Allocation Model: Maps software components to physical or virtual machines.



Redundancy and Backup Model: Shows allocation of redundant components for fault tolerance.


Web Server Cluster
Web Application
Redundant Web Server (Active-Passive)
WebAppBackup
Application Server Cluster
Business Logic Layer
Redundant App Server (Active-Passive)
BusinessLogicBackup
Database Cluster
Database
Database Replica (Active-Active)
DBReplica

Execution Viewpoint

3.1 Overview

The Execution Viewpoint focuses on the system's runtime behavior, describing how components interact during execution. It highlights performance, concurrency, and runtime resource utilization.

3.2 Concerns and Stakeholders

3.2.1 Concerns

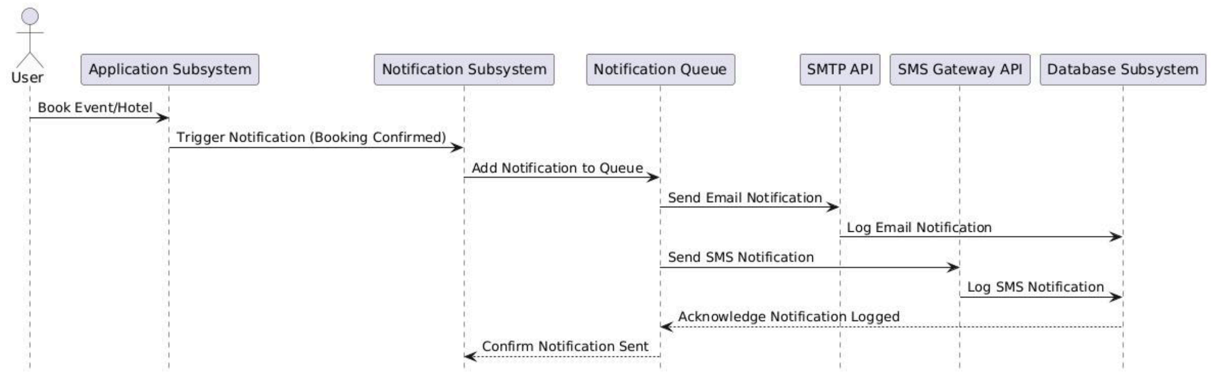
- **Runtime Performance:** How well does the system perform under load?
- **Concurrency:** How does the system handle multiple simultaneous requests or tasks?
- **Fault Tolerance:** How does the system handle failures during runtime?

3.2.2 Typical Stakeholders

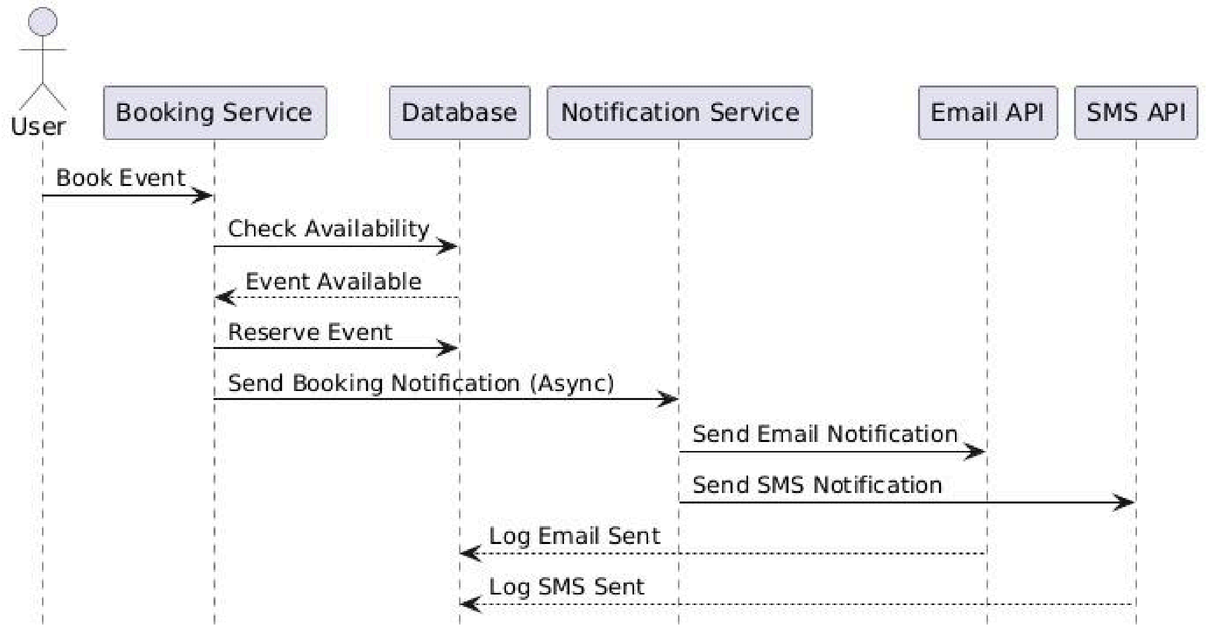
➔ **Developers**
➔ **System Testers**
➔ **System Architects**

3.3 Model Kinds

Execution Flow Model: Shows how different components interact at runtime.



Concurrency Model: Represents how the system manages multiple simultaneous processes or tasks.



4 Views+

4.1 Conceptual View

4.1.1 Models

The Conceptual View contains two primary models:

- **Component Model:**

  Purpose: To illustrate the main components of the system and their roles.

  Includes: Hotel Booking System, Event Booking System, User Dashboard, Notification Module, and Database.

- **Integration Model:**

  Purpose: To show how the system connects with external APIs for hotel availability and event data.

  Includes: External Hotel API, Event API, and how the system synchronizes data with these services.

4.1.2 Known Issues

Final API integration details are subject to changes based on external API specifications.

Allocation View

4.2.1 Models

- **Node Allocation Model**

  Diagram: UML Deployment Diagram

  Description: Maps components like Hotel Booking Module, Event Booking Module, Notification Service, and Database to hardware nodes such as servers and cloud instances.

- **Redundancy and Backup Model**

  Diagram: Redundancy Diagram

  Description: Illustrates how backup servers or services are allocated to ensure high availability in case of failure.

4.2.2 Known Issues with View

- **Scalability Limitations:** Cloud provider limits may restrict the addition of new resources during peak demand.
- **Potential Single Points of Failure:** If certain components (like load balancers) are not redundantly allocated, they could become bottlenecks.
- **Dependency on External Infrastructure:** Downtime from third-party cloud providers could impact system availability.

Execution View

4.3.1 Models

- **Execution Flow Model**

  Diagram: UML Sequence Diagram

  Description: Depicts interactions between components (e.g., User → Booking Service → Database) during a booking process.

- **Concurrency Model**

  Diagram: Concurrency Diagram

  Description: Shows how multiple requests (e.g., hotel searches, bookings) are handled simultaneously by different threads or processes.

4.3.2 Known Issues with View

- **Concurrency Bottlenecks:** Inadequate thread management may lead to performance degradation during high traffic.
- **Latency in API Calls:** External APIs might introduce delays, affecting overall system performance.
- **Error Handling in Real-Time:** Failures in critical components like the booking service may impact multiple users simultaneously if not properly isolated.

5 Architecture decisions and rationale

Decision ID: 1
Decision: Use a Layered Architecture

Concerns Addressed:

- Purpose of the System: Separation of concerns for clear functionality (UI, business logic, data).
- Suitability of the Architecture: Supports modularity and scalability.

Affected Elements:

- User Management Subsystem
- Hotel Booking Subsystem
- Event Booking Subsystem
- Database Subsystem
- Notification Subsystem

Rationale:

Layered architecture divides the system into logical layers (e.g., presentation, business logic, data). It simplifies maintenance, enhances modularity, and allows for independent updates of layers.

Forces and Constraints:

- Clear separation for maintenance but can introduce performance overhead due to layer communication.
- Simplifies unit testing for each layer.

Assumptions: Layers will communicate through defined APIs to reduce coupling.

Considered Alternatives:

- **Monolithic Architecture:** Rejected due to lack of flexibility for scaling individual parts.

- **Event-Driven Architecture:** Considered, but less suitable for this system's synchronous user flow.

Potential Consequences:

- **Positive**: Improved maintainability and flexibility for future feature updates.

- **Negative**: Increased initial setup complexity for defining clear interfaces.

Decision ID: 2
Decision: Use a Microservices Architecture

Concerns Addressed:

- Modifiability and Scalability: Allows independent scaling and updating of subsystems.

- Integration: Easier to integrate external APIs and replace components.

Affected Elements:

- Hotel Booking Subsystem
- Event Booking Subsystem
- Notification Subsystem
- Payment Processing Subsystem

Rationale:

Microservices architecture enables each subsystem to be independently developed, deployed, and scaled, ensuring flexibility and faster updates.

Forces and Constraints:

- **Distributed System Complexity:** Requires more effort to manage communication between services.
- **Operational Overhead:** Service discovery, monitoring, and deployment pipelines.

Assumptions:

Service communication will rely on lightweight protocols.

Considered Alternatives:

- **Monolithic:** Rejected due to scaling limitations.
- **SOA (Service-Oriented Architecture):** Rejected for being heavier and less aligned with modern cloud-native practices.

Potential Consequences:

- **Positive:** Increased system resilience, flexibility for future scaling, and modular updates.
- **Negative:** More complex deployment and monitoring setups.

Decision ID: 3

Decision: Use **API Gateway Layer**

Concerns Addressed:

- **Integration and Security:** Centralized entry point for client requests.

- **Scalability and Routing:** Handles routing to appropriate microservices based on the request.

Affected Elements:

- User Management Service
- Booking Service
- Notification Service
- Search Service

Rationale:

The API Gateway serves as a single point of contact between the client and backend services, reducing complexity for the frontend. It provides centralized features like request routing, load balancing, security (e.g., authentication), and caching.

Forces and Constraints:

- Simplifies the frontend as it only communicates with the gateway.

- Introduces a potential single point of failure, which needs redundancy for high availability.

Assumptions:

Future services can easily be added behind the gateway.

Considered Alternatives:

Direct Client-Service Communication: Avoids additional gateway layer but increases frontend complexity and exposes microservices to security risks.

Potential Consequences:

- **Positive:** Simplifies client-side logic and improves security and scalability.
- **Negative:** Requires monitoring and scaling the gateway itself.

Decision ID: 4

Decision: Use Separate Databases per Service

Concerns Addressed:

- Data Ownership and Isolation: Each microservice manages its own data.
- Scalability and Maintainability: Independent scaling of databases based on service needs.

Affected Elements:

- User Database
- Booking Database
- Events Database
- Notifications Database

Rationale:

Using separate databases ensures that each service is fully independent, reducing the risk of cross-service impact and improving maintainability. This aligns with the microservices architecture principle of decentralized data management.

Forces and Constraints:

- Increases resilience since failures in one service won't affect the others' databases.
- Increases operational overhead due to managing multiple databases.

Assumptions:

- Each service has specific and unique data requirements.
- Data sharing between services is minimal or done via APIs/events.

Considered Alternatives:

Single Shared Database: Easier to manage but creates strong coupling between services, making them harder to scale independently.

Potential Consequences:

- **Positive:** Improved modularity, scalability, and failure isolation.
- **Negative:** Increased complexity in managing multiple databases.

B Appendices

B.1 Functional Requirement

1.1.Search Hotels

The system shall allow users to search for hotels based on location, check-in and check-out dates, room type (single, double, family), and availability.

1.2.View Hotel Details

The system shall display detailed information about hotels, including room types, prices, available facilities.

1.3.Book Hotel Rooms

The system shall allow authenticated users to book available rooms in selected hotels.

1.4.Hotel Recommendations

- **1.4.1** The system shall recommend nearby events based on the user's hotel location and booking dates.
- **1.4.2** The system shall display these event recommendations on the user's dashboard as notifications.

1.5.Search Events

The system shall allow users to search for local events based on location, date, and event type.

1.6.View Event Details

The system shall display event details, including name, description, date, time, location, and ticket price.

1.7.Book Event Tickets

The system shall allow authenticated users to book tickets for available events.

1.8.User Registration

The system shall allow users to create an account using their email and password.

1.9.Login/Logout

- **1.9.1** The system shall authenticate users securely to access their accounts.
- **1.9.2** The system shall allow users to log out securely.

1.10.Forgot Password

The system shall provide a mechanism for users to reset their password via email/SMS.

1.11.User Dashboard

- **1.11.1** The system shall provide a dashboard for authenticated users.
- **1.11.2** The dashboard shall display a list of the user's hotel and event bookings.
- **1.11.3** The dashboard shall display notifications related to bookings, offers, and event recommendations.
- **1.11.4** The dashboard shall allow users to print their bookings.

1.12.Notification Templates Management

- **1.12.1** The system shall allow administrators to create, edit, and store notification templates.
- **1.12.2** The system shall support placeholders in notification templates for dynamic data (e.g., user name, booking details).
- **1.12.3** The system shall support multiple languages for notification templates.
- **1.12.4** The system shall allow administrators to specify the channel (email/SMS) for each template.

1.13.Notification Sending and Queueing

- **1.13.1** The system shall queue notifications for asynchronous processing.
- **1.13.2** The system shall process the queue to send notifications via email or SMS.

1.14.Error Handling

- **1.14.1** The system shall handle failed notifications and log the reason for failure.
- **1.14.2** The system shall implement retry mechanisms for failed notifications.

1.15.Notification Types

The system shall send notifications for the following events:

- Registration confirmation.
- Password reset.
- Booking confirmation (hotels/events).
- Nearby event recommendations.
- Offers and promotions.

1.16.Hotel Providers

The system shall fetch hotel details dynamically by contacting external hotel provider APIs.

1.17.Events API

- **1.17.1** The system shall consume event data from a third-party events API.
- **1.17.2** The system shall store events in the database .

1.18.Statistics for Notifications

- **1.18.1** The system shall provide statistics on the number of successfully sent notifications for each type (email/SMS).
- **1.18.2** The system shall provide statistics on the number of failed notifications and reasons for failure.

1.19.Data Encryption

The system shall encrypt sensitive user data, including passwords and notifications.

B.2 Scenarios

Scenario 1: Viewing and Booking a Hotel Room

As a tourist visiting Cairo for the first time, I want to search for available hotel rooms based on my travel dates and preferences (e.g., single, double, or family room), so that I can find a convenient and affordable place to stay during my visit.

Scenario 2: Event Recommendation Based on Hotel Booking

As a user who has booked a hotel room, I want to receive recommendations for events happening near my hotel during my stay, so that I can make the most out of my visit and explore local attractions.

Scenario 3: Notifications for Event and Hotel Booking

As a user who has booked an event ticket and hotel room, I want to receive a notification on my dashboard and email confirming my bookings, so that I can have a record of my reservations and stay organized.

Scenario 4: Successful Booking Notification

As a user, I want to receive an instant email and SMS after successfully booking a hotel or event, so that I can have confirmation for my records.

Scenario 5: Notification Language Selection

As a user who prefers notifications in a specific language, I want to choose my preferred language for all communications, so that I can understand notifications more easily.

Scenario 6: Event Ticket Booking

As a user interested in attending local events, I want to search for and book event tickets, so that I can secure my spot before they sell out.

Scenario 7: Dashboard Access for Bookings

As a frequent traveler, I want to view all my current and past bookings on my dashboard, so that I can keep track of my reservations and print receipts as needed.

Scenario 8: Forget password recovery

As a user who has forgotten my password, I want to receive a reset password email with a secure link, so that I can regain access to my account.

Scenario 9: Event Search

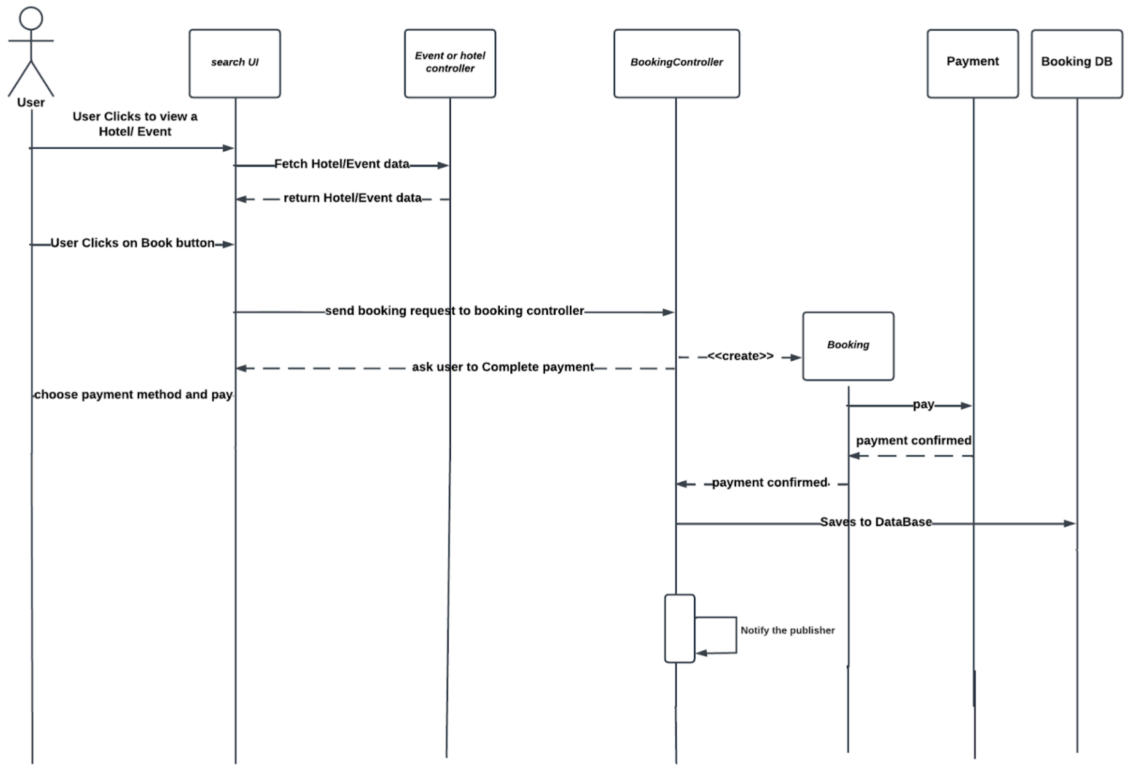
As a user, I want to search for local events based on location and date, so that I can find and attend events that match my interests during my stay.

Scenario 10: Template Management for Notifications

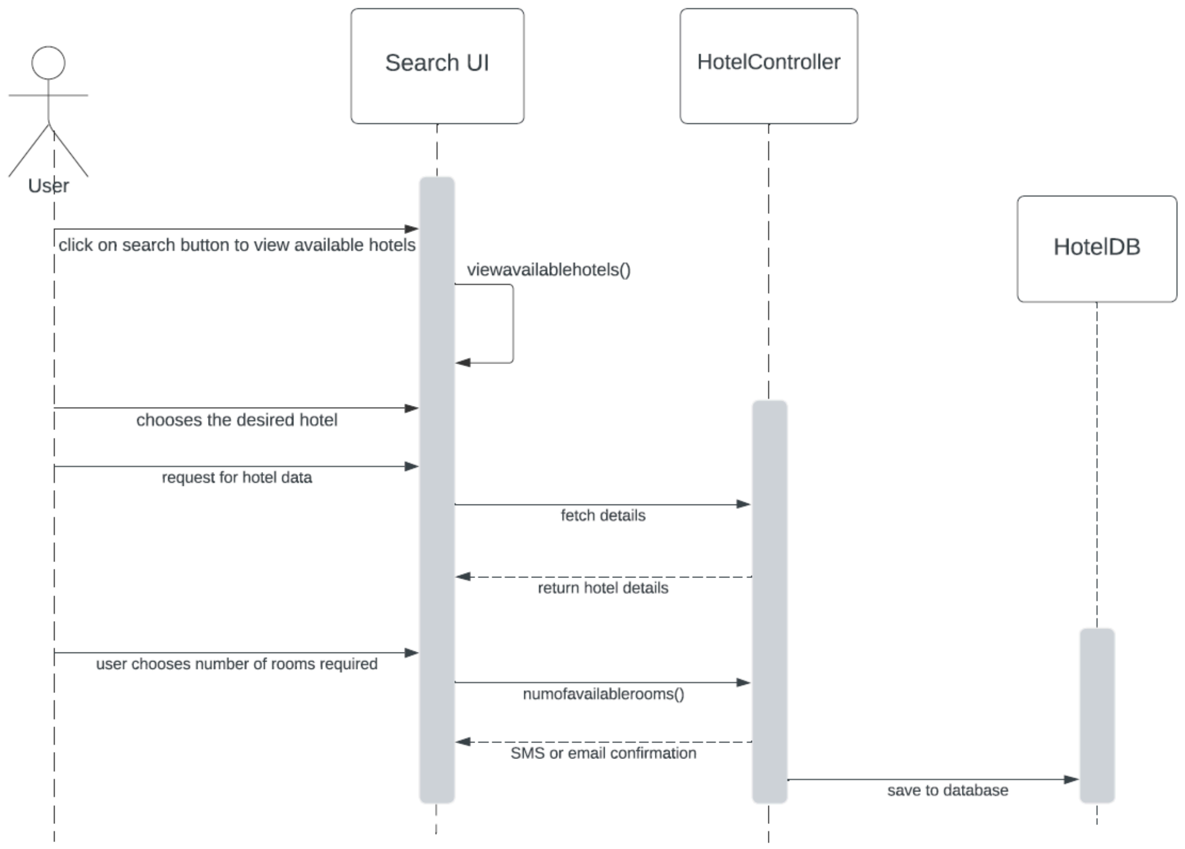
As a product owner, I want to manage notification templates for different use cases (e.g., booking confirmation, password reset), so that I can ensure consistent and personalized communication with users.

B.3 Sequence Diagram

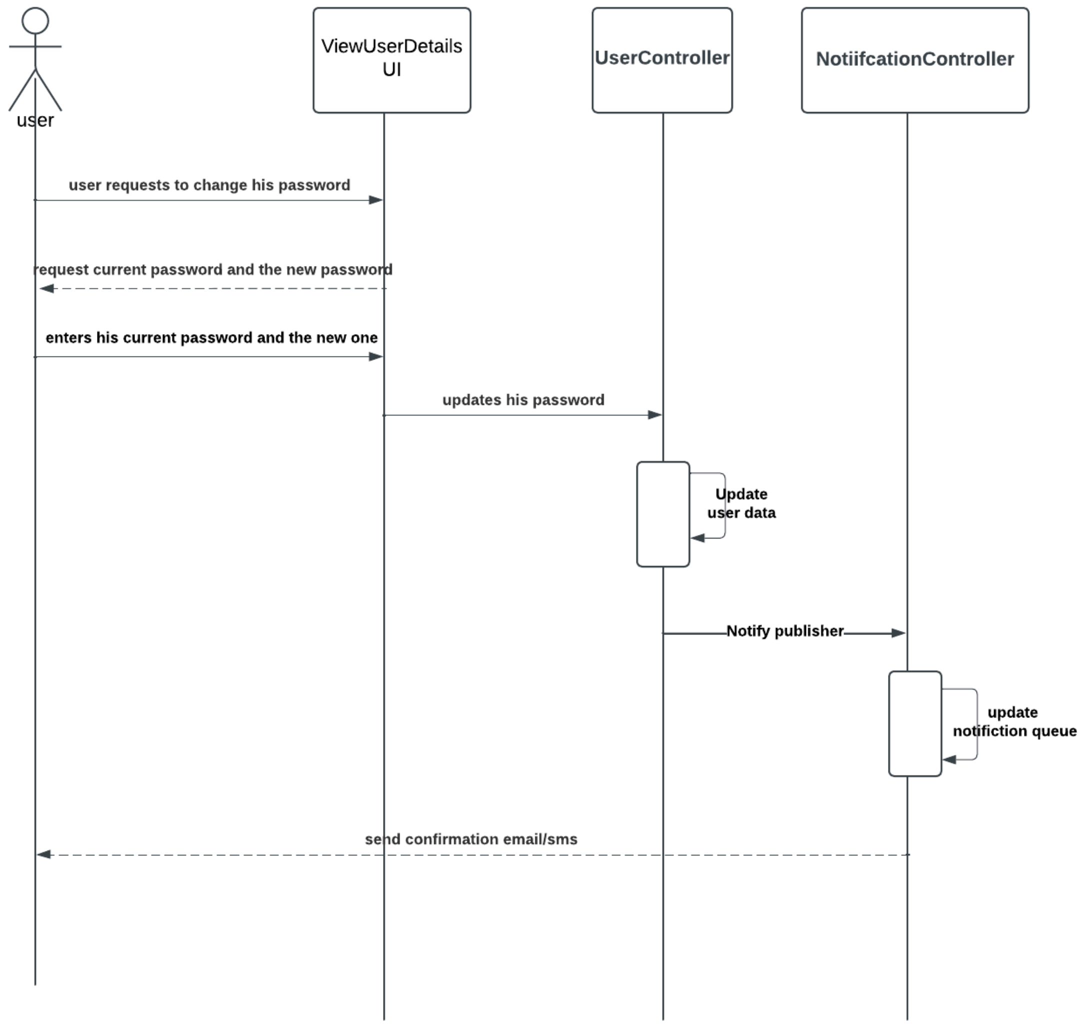
B.3.1 Booking


User
search UI
Event or hotel controller
BookingController
Payment
Booking DB
User Clicks to view a Hotel/ Event
Fetch Hotel/Event data
return Hotel/Event data
User Clicks on Book button
send booking request to booking controller
<<create>>
Booking
ask user to Complete payment
choose payment method and pay
pay
payment confirmed
payment confirmed
Saves to DataBase
Notify the publisher

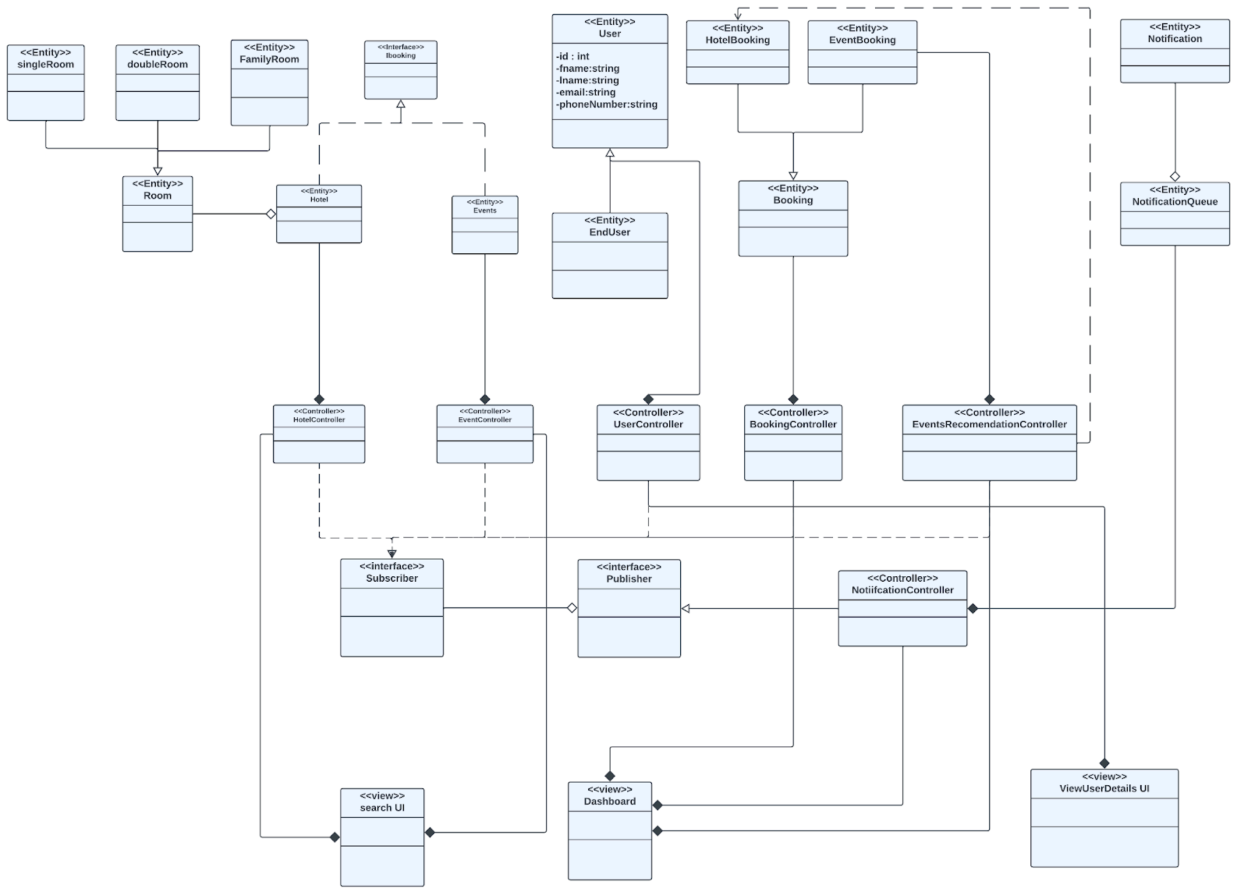
B.3.2 Search / View



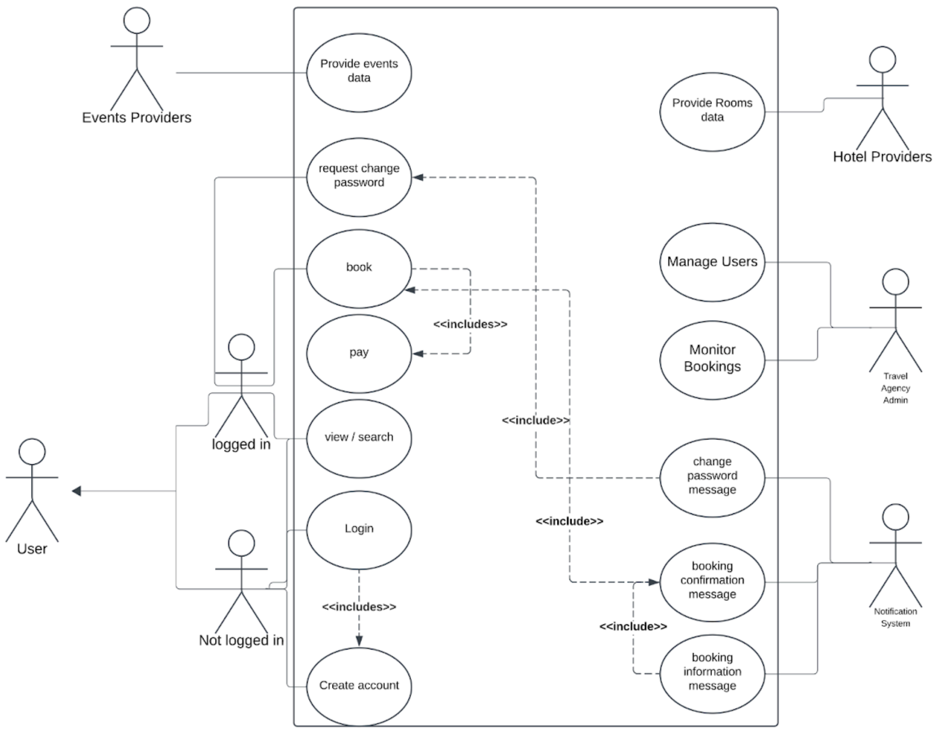
B.3.3 Change Password


user
ViewUserDetails UI
UserController
NotiifcationController
user requests to change his password
request current password and the new password
enters his current password and the new one
updates his password
Update user data
Notify publisher
update notifiction queue
send confirmation email/sms

B.4 Class Diagram

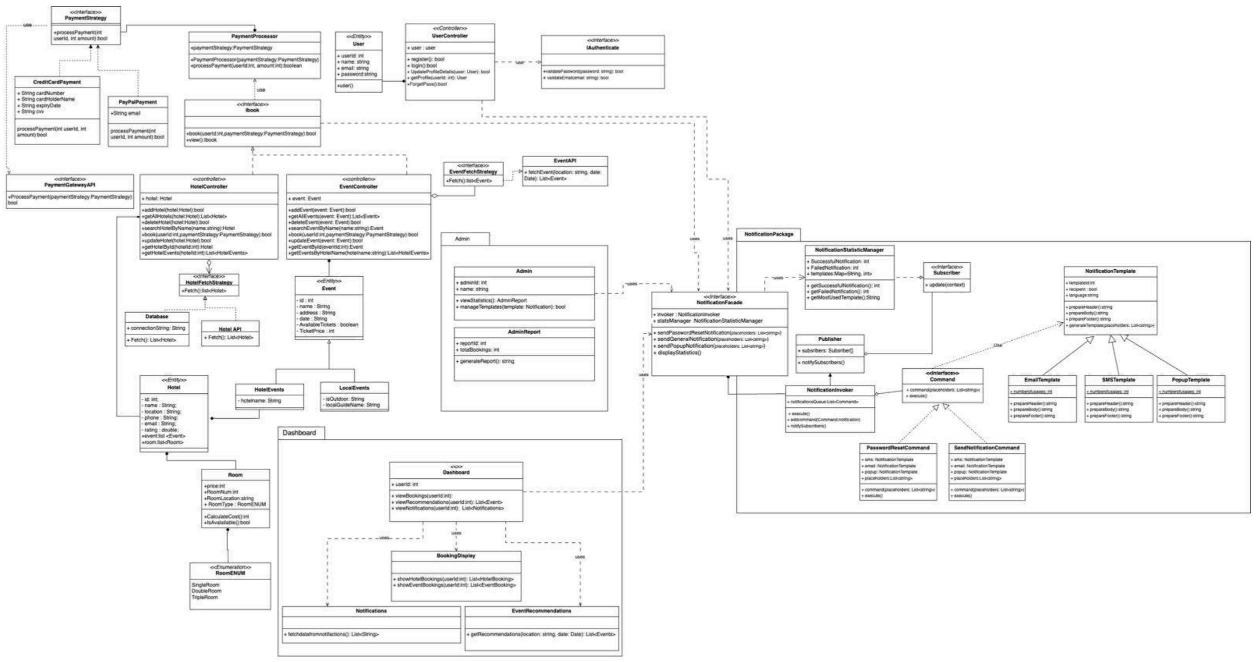


B.5 Use Case Diagram



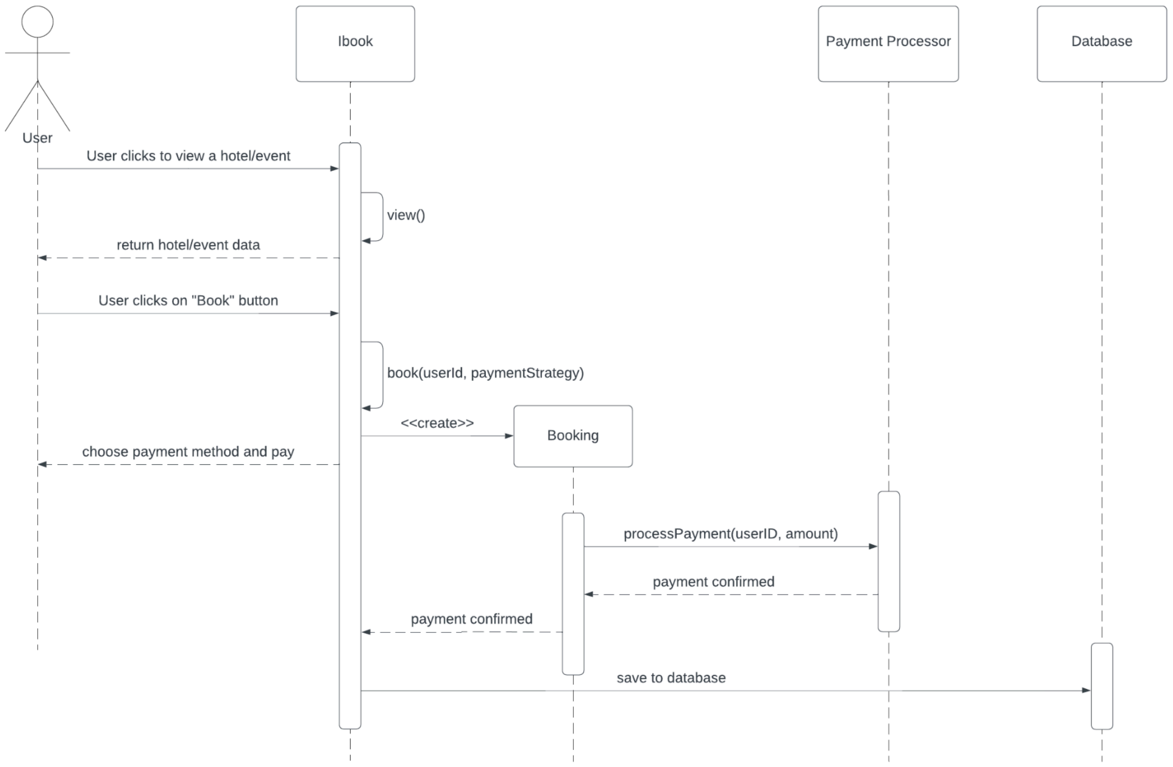
C Appendices Phase2:

C.1 Class Diagram:

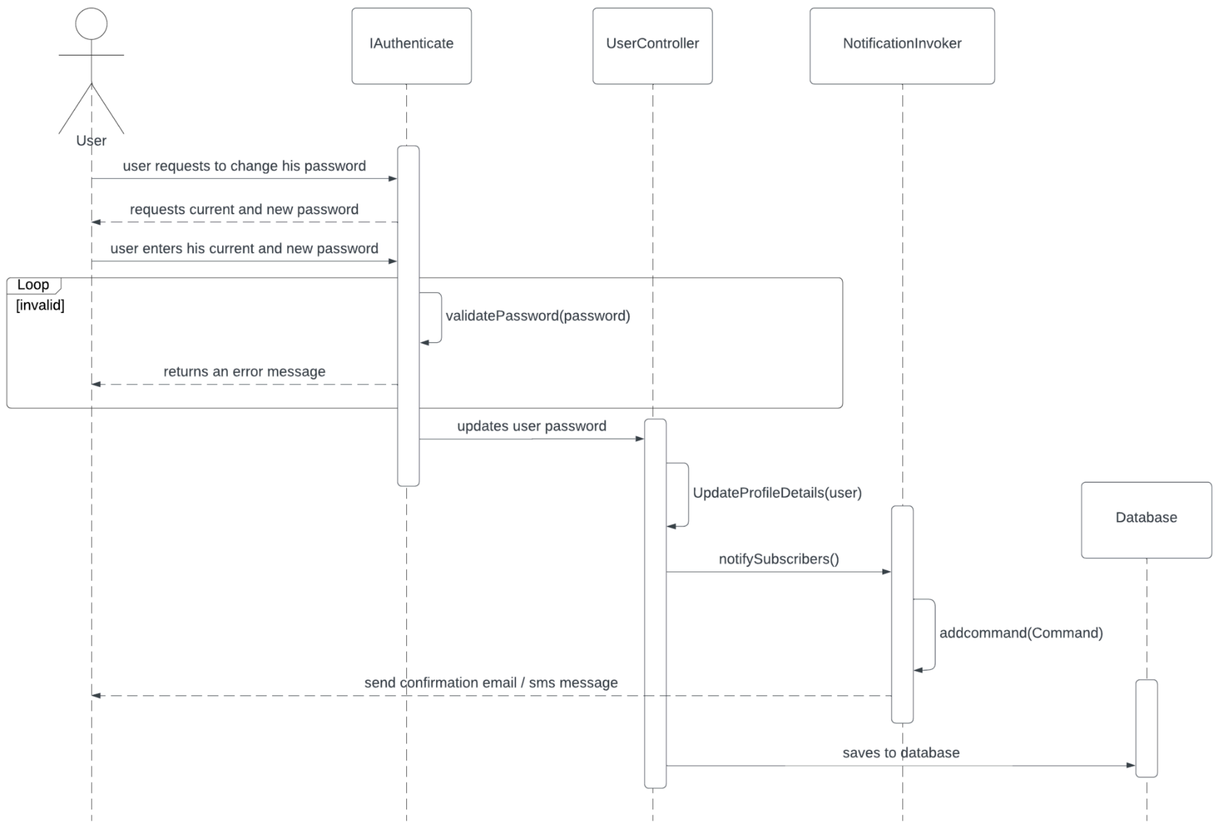


C.2 Sequence Diagram:

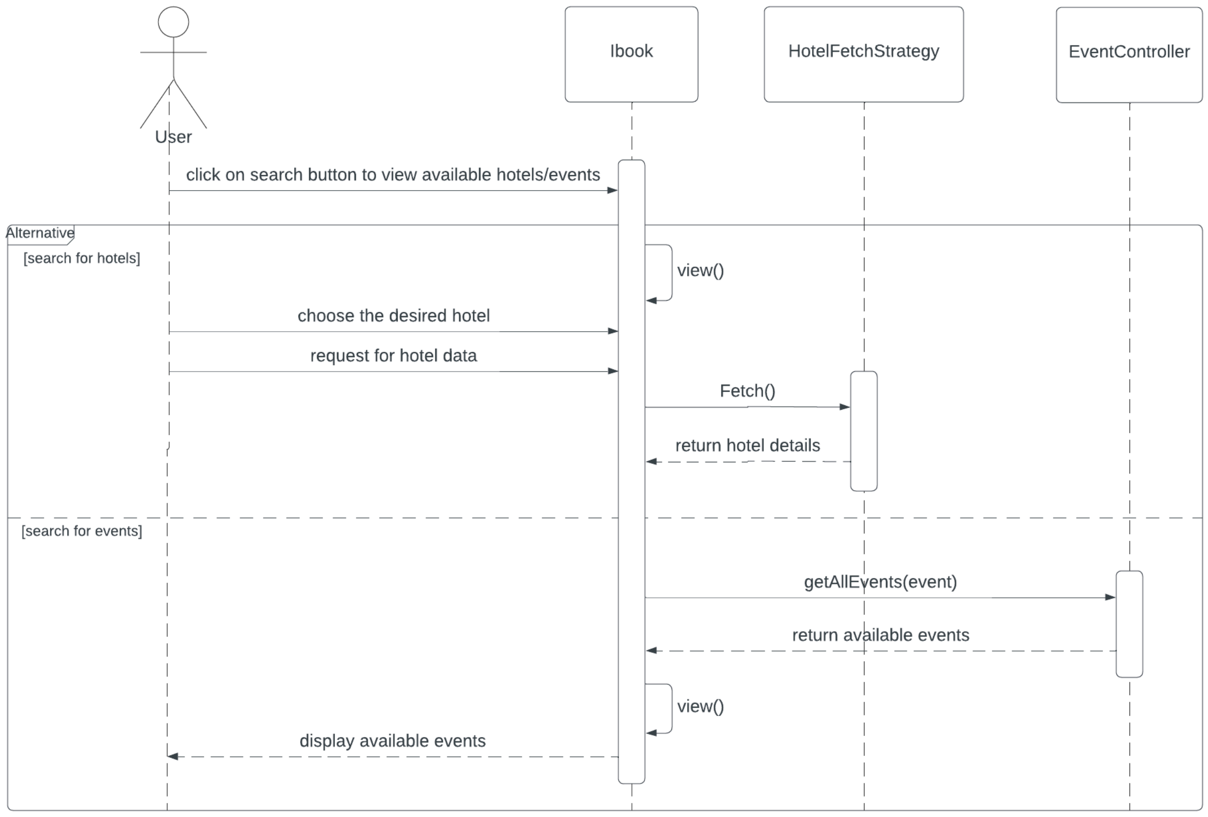
C.2.1 Booking:


Ibook
Payment Processor
Database
User
User clicks to view a hotel/event
view()
return hotel/event data
User clicks on "Book" button
book(userId, paymentStrategy)
<<create>>
Booking
choose payment method and pay
processPayment(userID, amount)
payment confirmed
payment confirmed
save to database

C.2.2 Change Password:


IAuthenticate
UserController
NotificationInvoker
User
user requests to change his password
requests current and new password
user enters his current and new password
Loop
[invalid]
validatePassword(password)
returns an error message
updates user password
UpdateProfileDetails(user)
Database
notifySubscribers()
addcommand(Command)
send confirmation email / sms message
saves to database

C.2.3 Search/View:



Bibliography

[1] Paul C. Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. Documenting Software Architectures: views and beyond. Addison Wesley, 2nd edition, 2010.

[2] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: a framework for integrating multiple perspectives in system development. International Journal of Software Engineering and Knowledge Engineering, 2(1):31–57, March 1992.

[3] IEEE Std 1471, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, October 2000.

[4] ISO/IEC/IEEE 42010, Systems and software engineering — Architecture description, December 2011.

[5] Alexander Ran. Ares conceptual framework for software architecture. In M. Jazayeri, A. Ran, and F. van der Linden, editors, Software Architecture for Product Families Principles and Practice, pages 1–29. Addison-Wesley, 2000.

[6] Nick Rozanski and Eo ´in Woods. Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. Addison Wesley, 2nd edition, 2011.

[7] Uwe van Heesch, Paris Avgeriou, and Rich Hilliard. A documentation framework for architecture decisions. The Journal of Systems & Software, 85(4):795–820, April 2012.