Hoiiday Application

Name: Pîrvulescu Carmen-Gabriela

Group: 30431

Contents

1 Deliverable 1

1.1 Project Specification

Hoiiday is a web-based booking application inspired by a passion for travel and exotic destinations. It is designed to provide a user experience similar to well-known platforms like Booking.com. Users can search for available properties, view detailed information — such as amenities, policies, and rules — and make bookings for specific dates.

A distinctive feature of the application is the "Tips & Tricks" section, where users can find travel advice, destination suggestions based on the season, and packing recommendations. In the future, the application may also offer optional features like car rentals and flight bookings.
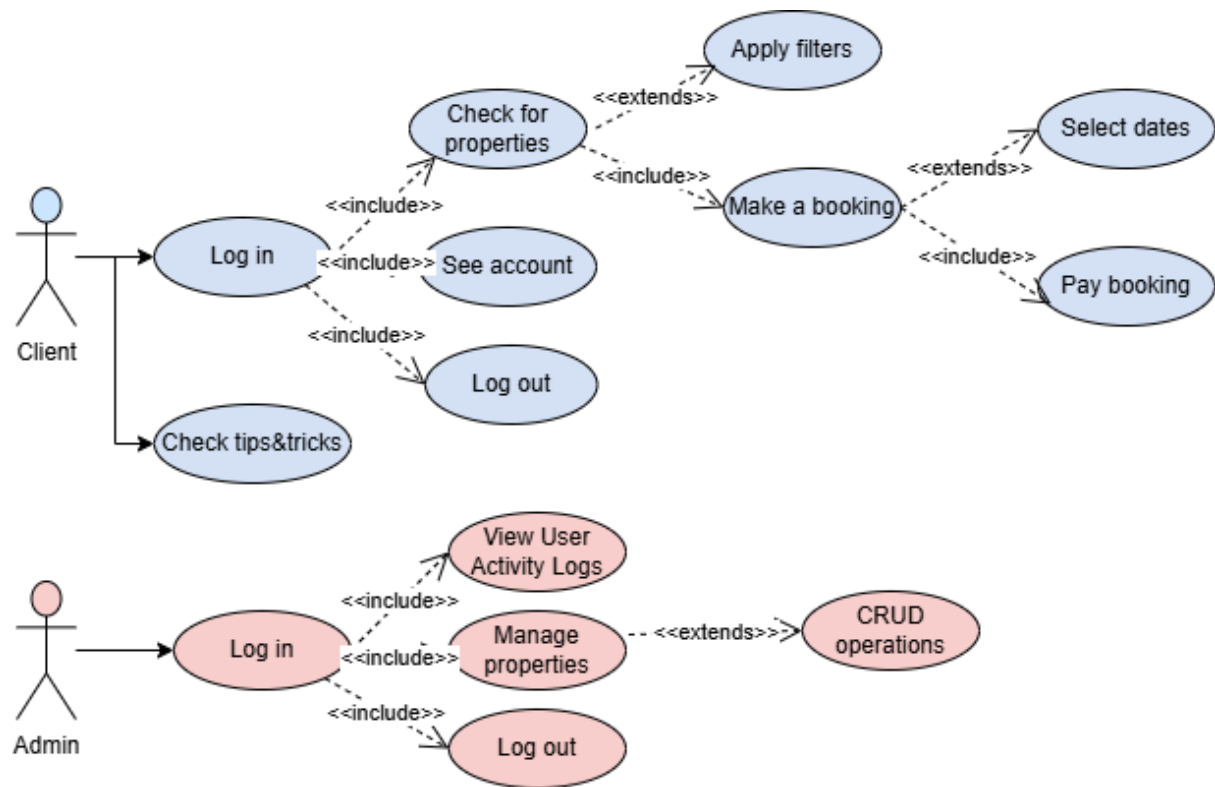
1.2 Functional Requirements

The system supports the following core functionalities:

- Users can create an account and log in, with roles such as client or admin.

- Users can search for properties using filters like location, number of guests, or number of rooms.

- Users can book a property if it is available during the selected period.

- Users can access travel-related tips and destination recommendations.

- Users can make payments by card and receive an invoice after booking.

- Admins can add, update, or delete property listings.

- Admins can view user accounts, monitor activity logs, and manage the platform.
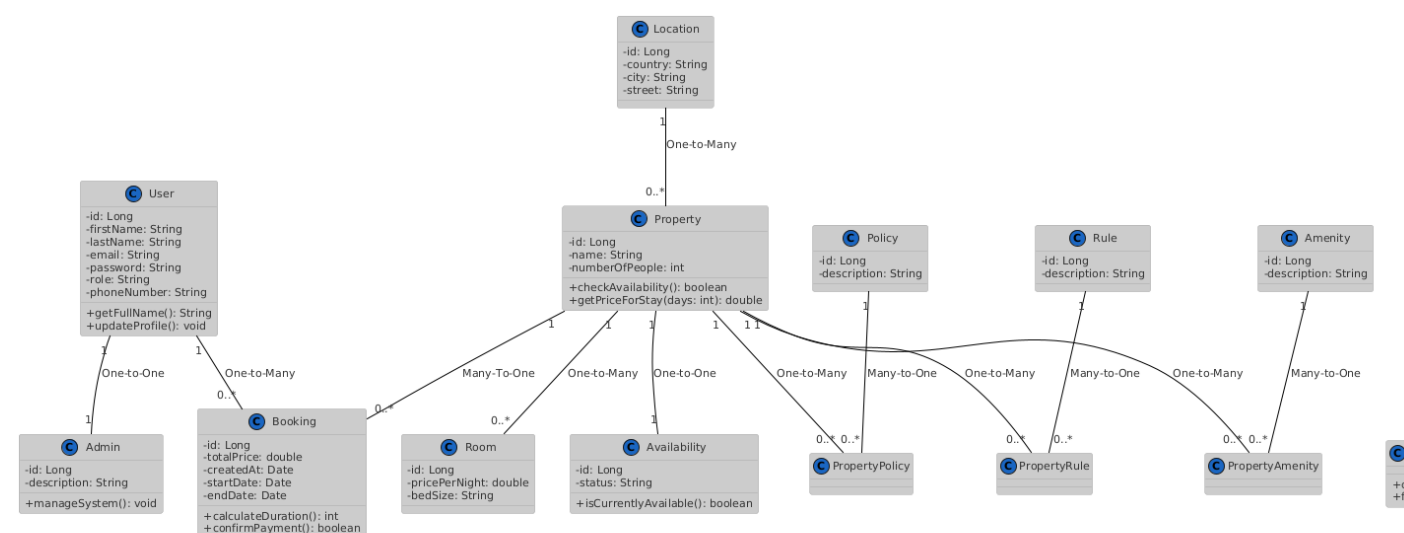
Planned features include booking flights and renting cars.

1.3 Use Case Model

1.3.1 Use Case Identification

1.3.2 UML Use Case Diagram

1.4 Supplementary Specification

1.4.1 Non-functional Requirements

- **Performance**: The application should quickly respond to user actions, ensuring minimal loading times and smooth interaction.

- **Security**: User passwords will be stored securely using encryption techniques. OAuth2 will be used for login to enhance authentication security.

- **Usability**: The interface should be intuitive, modern, and simple to navigate, avoiding unnecessary complexity.

- **Scalability**: The application must support a growing number of listings without a drop in performance.

1.4.2 Design Constraints

- **Programming Language**: The backend will be developed using Java with the Spring Boot framework.

- **Frontend Framework**: The frontend will be built using Angular 18, leveraging standalone components for modularity.

- **Database**: The system will use a MySQL relational database to store user and property data, including encrypted passwords.

- **Authentication**: The login process will use both standard credentials and Google OAuth2 for secure access.

- **Architecture**: The system will follow a RESTful client-server architecture, structured into multiple layers such as the data access layer, service layer, and presentation layer.

1.4.3 Glossary

- **Booking**: Reserving a property for a selected time interval.

- **Admin**: A user with full access rights who can manage properties, users, and system logs.

- **Tips & Tricks**: A section providing users with useful travel recommendations and seasonal advice.

- **Amenity**: A service or feature offered by a property, such as Wi-Fi, pool, or parking.

- **OAuth2**: A secure authentication protocol allowing login via third-party services like Google.

- **Invoice**: A digital receipt issued after payment is completed.

- **REST API**: A style of backend design using HTTP methods for communication between client and server components.

2 Deliverable 2

2.1 Domain Model

The domain model represents the core entities and their relationships within the Hoiiday application. The key entities include **User**, **Admin**, **Property**, **Booking**, **Room**, and **Location**, which form the backbone of the application. These entities are connected through various relationships.

The **Conceptual Class Diagram** is depicted below, showing the classes and their relationships.
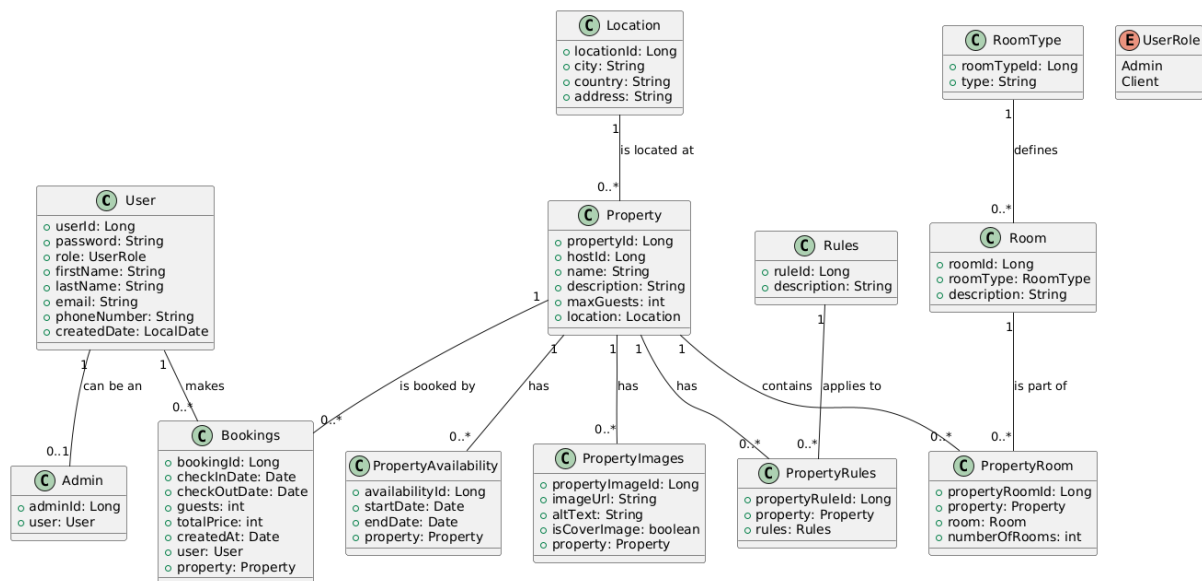

Figure 1: Conceptual Class Diagram for the Hoiiday Application

2.2 Architectural Design

2.2.1 Conceptual Architecture

The conceptual architecture for the Hoiiday application follows the **Layered Architecture Pattern**. It is divided into three main layers: the **Presentation Layer**, **Service Layer**, and **Data Access Layer**.
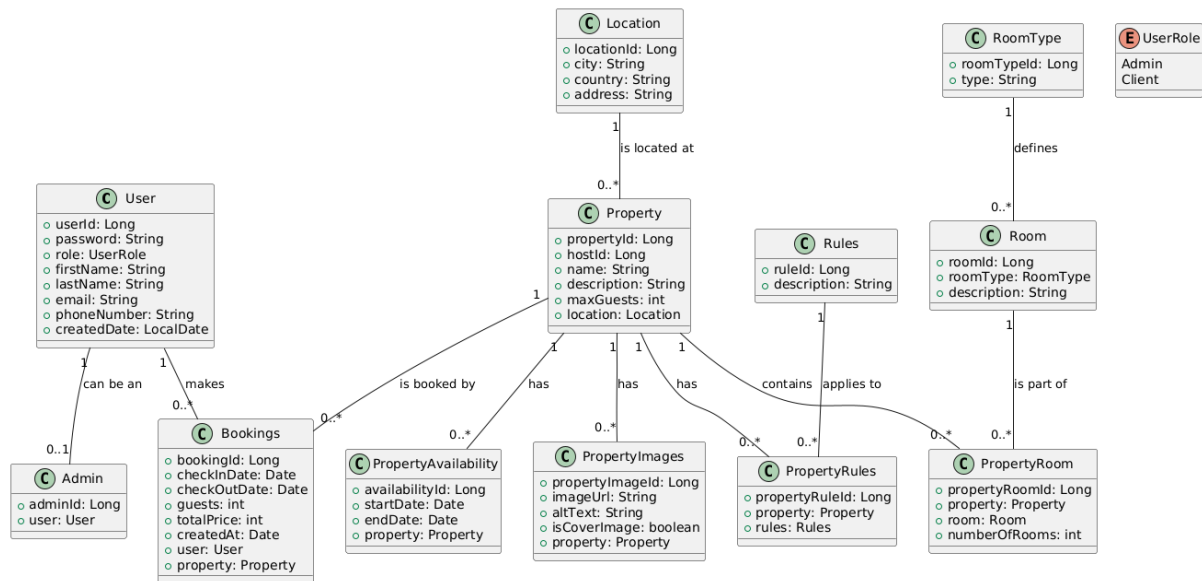
Figure 2: Conceptual Architecture of the Hoiiday Application

2.2.2 Package Design

The package design organizes the Hoiiday application into distinct and logically grouped modules, each responsible for specific functionality. This modular organization helps in maintaining clean code.

- **Controller Package**: - This package contains the **controllers** responsible for handling the incoming HTTP requests from the frontend.

- **Service Package**: - The **service** layer contains the **business logic** of the application. Services communicate with the database layer (via the **repository** package) to fetch or update data.

- **Repository Package**: - The **repository** package is responsible for interacting with the database. It uses **Spring Data JPA** to manage entities. Repositories are responsible for CRUD operations.

- **Model Package**: - The **model** package contains the **entity classes** that represent the core objects of the application. These classes are mapped to the corresponding database tables using **JPA annotations**.

- **DTO Package**: - The **DTO** (Data Transfer Object) package contains the **data transfer objects** that are used to transfer data between layers of the application.

- **Mapper Package**: - The **mapper** package is responsible for converting between **entities** and **DTOs**. It uses a mapping layer to transfer data between the service and the DTO layer.

- **Authentication Package** (optional): - This package handles **authentication and authorization**. It manages user login, logout, token generation (JWT, for example), and role-based access control (RBAC). It ensures that only authorized users can access certain features of the application.
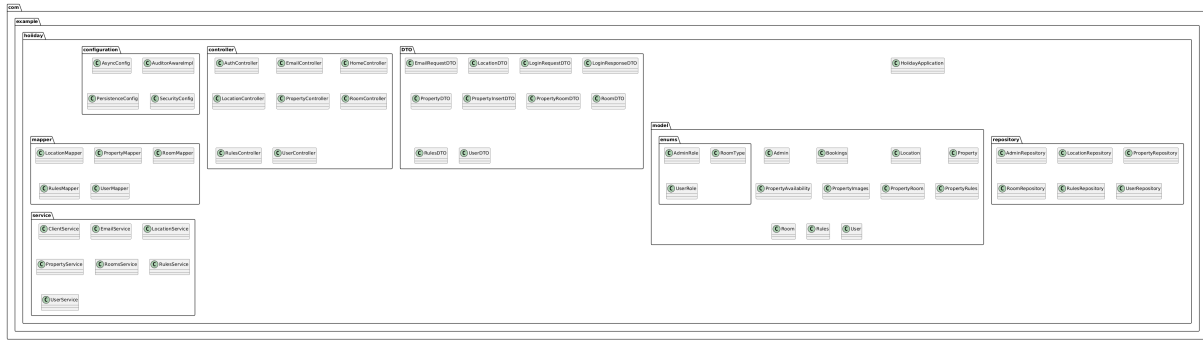
Figure 3: Package Design for the Hoiiday Application

2.2.3 Component and Deployment Diagram

The component and deployment diagrams illustrate the system structure and its deployment across different environments.

- **Component Diagram**: - The **component diagram** shows the system's software components (e.g., **Angular frontend**, **Spring Boot backend**, **database**) and their interactions via well-defined interfaces, such as REST APIs. - It highlights how services like **User Service** and **Booking Service** communicate with the database and each other.

- **Deployment Diagram**: - The **deployment diagram** shows how components are deployed across physical servers. It illustrates the deployment of the **frontend** on a **web server**, the **backend** on an **application server**, and the **database** on a **database server**. - It also shows how these servers communicate, ensuring smooth interaction between components.

Both diagrams provide insight into the system's logical and physical architecture, helping identify potential **bottlenecks** and ensuring **scalability**.
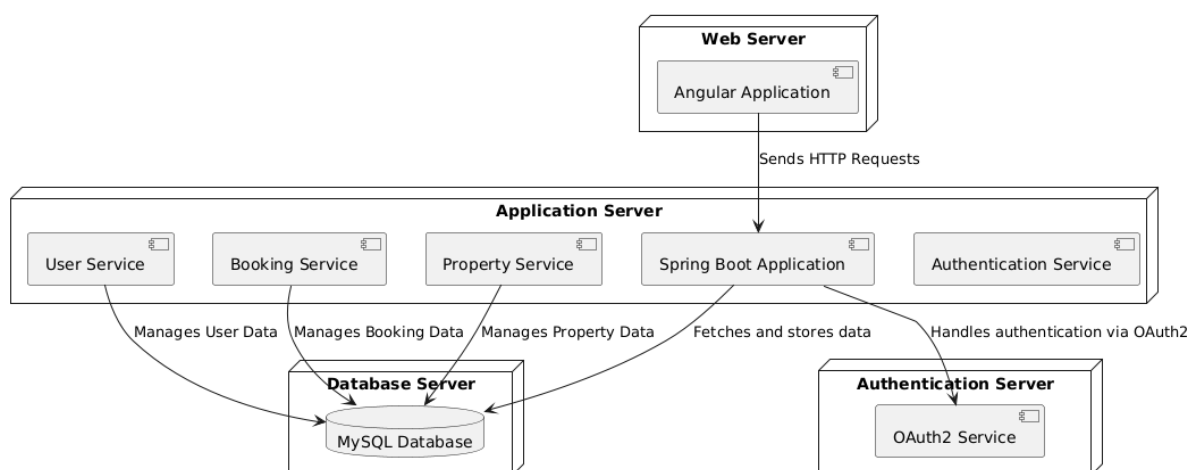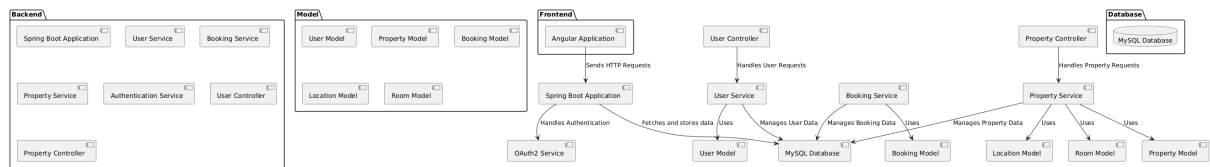

Figure 4: Deployment Diagram for the Hoiiday Application

Figure 5: Component Diagram for the Hoiiday Application