HBnB Project

Comprehensive Technical Document

1. Introduction

1.1 Project Overview

HBnB is a **web-based platform** that allows users to **sign up, create listings, submit reviews, and search for accommodations**. It provides an easy-to-use interface for hosts to list properties and for guests to explore and book available places. The platform incorporates **user authentication, property management, and review-based ratings** to enhance trust and usability.

This document serves as a **technical reference** for the project, outlining its **architecture, design, and core functionalities**. It provides a structured approach to understanding **how different system components interact**, ensuring clarity in development and maintenance. The HBnB platform is designed with scalability in mind, allowing for future enhancements such as **advanced search filters and secure payment integrations**.

1.2 Objective of the Document

This document acts as a **roadmap for the development** of the HBnB platform, detailing its **key architectural components, business logic, and API interactions**. It aims to:

- Provide a **structured overview of the system's design and functionality**.
- Outline **database modeling, API endpoints, and business logic** for developers.
- Ensure **consistent development practices and system scalability**.
- Facilitate **collaboration among engineers, architects, and stakeholders**.

By clearly defining **how different system layers interact**, this document ensures a **cohesive development process**, improving maintainability and performance throughout the lifecycle of the HBnB platform.

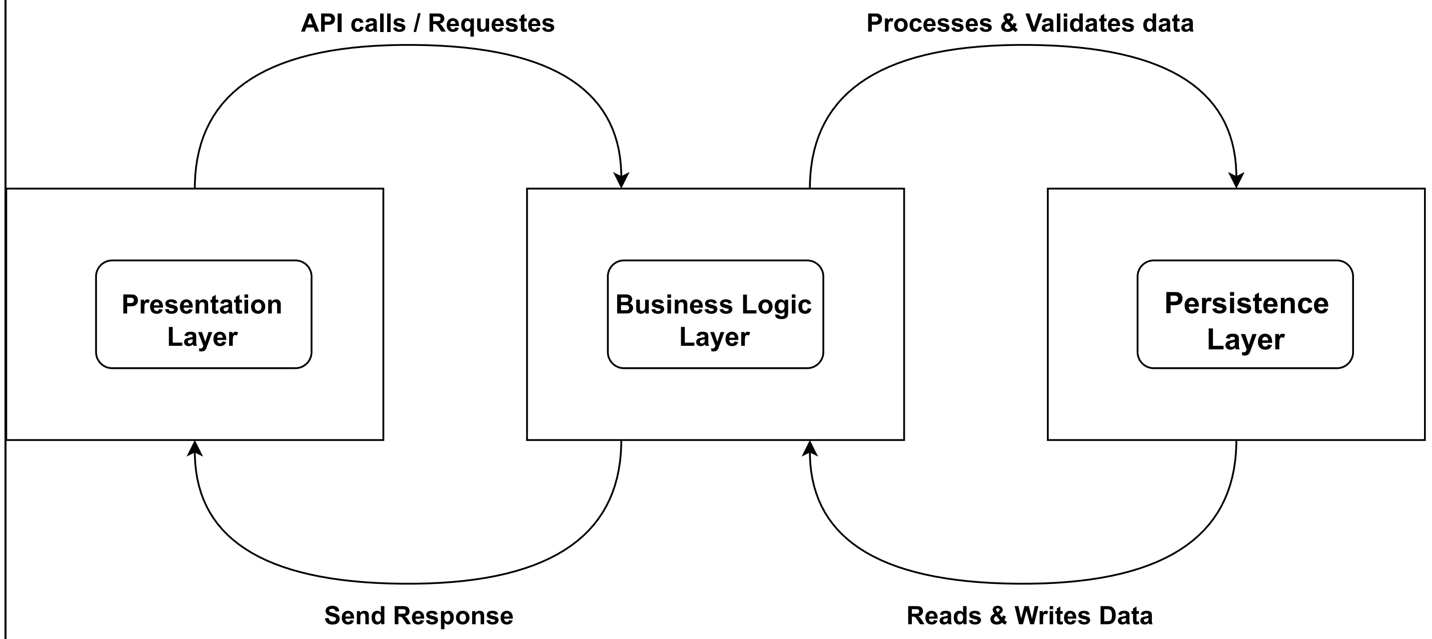
2. System Architecture

2.1 Overview

The system follows a **three-tier architecture** incorporating a facade pattern to maintain modularity and separation of concerns:

- **Presentation Layer (Services, API)**: Manages request handling and client interactions.
- **Business Logic Layer (Models):** Processes incoming data and enforces business rules.
- **Persistence Layer:** Handles information storage and retrieval from the database.

2.2 High-Level Architectural Diagrams

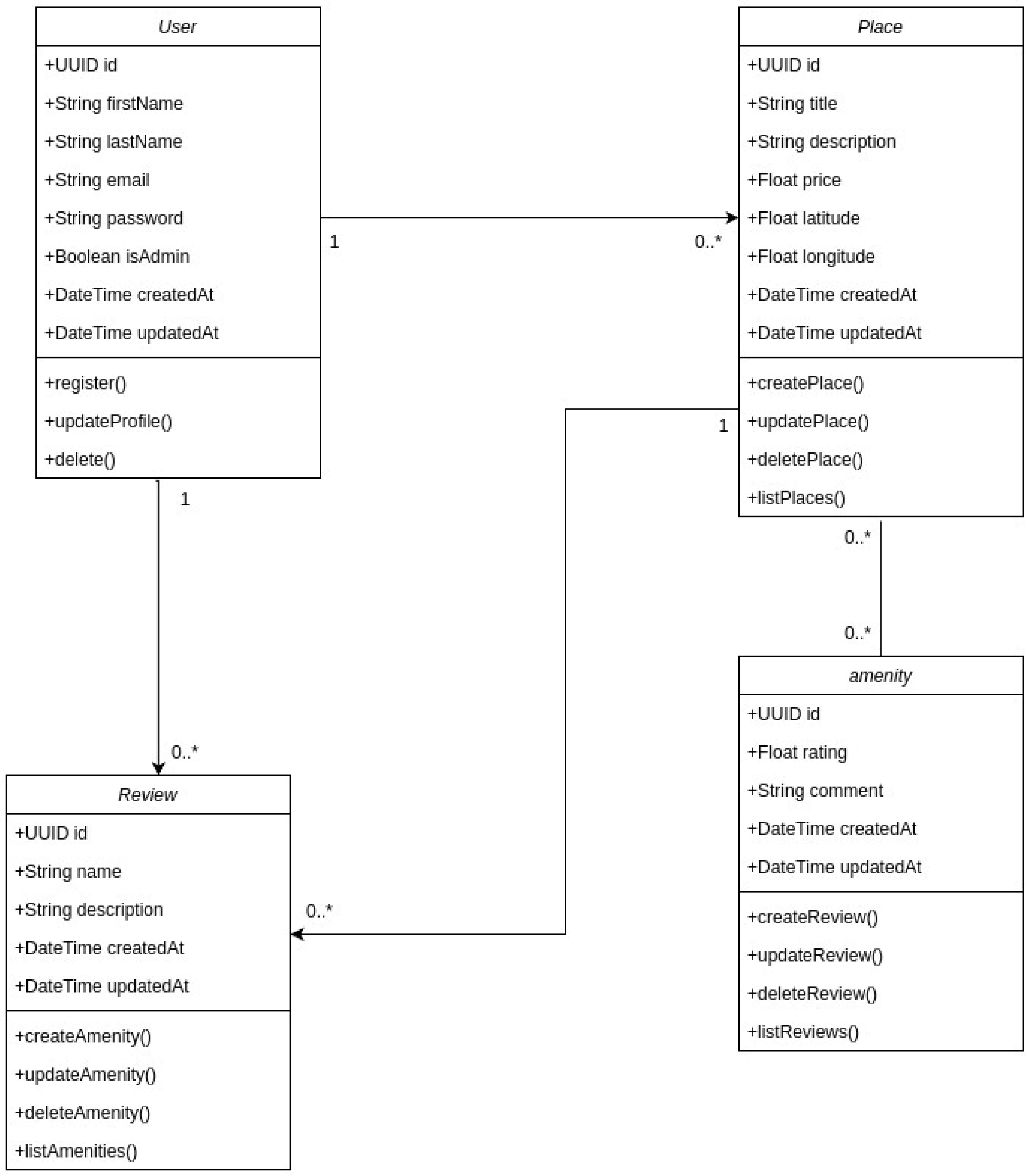


2.3 Component Descriptions

- **Presentation Layer (Services, API):**
    - Accepts API calls from users.
    - Forwards requests to the application logic.
    - Returns processed data to users.
- **Business Logic Layer (Models):**
    - Handles request validation and processing.
    - Implements business rules such as user authentication and data validation.
    - Interfaces with the data storage layer.
- **Persistence Layer:**
    - Maintains all user, place, and review records.
    - Manages data retrieval, updating, and deletion operations.

3. Business Logic and Data Model

3.1 Class Structure Diagram



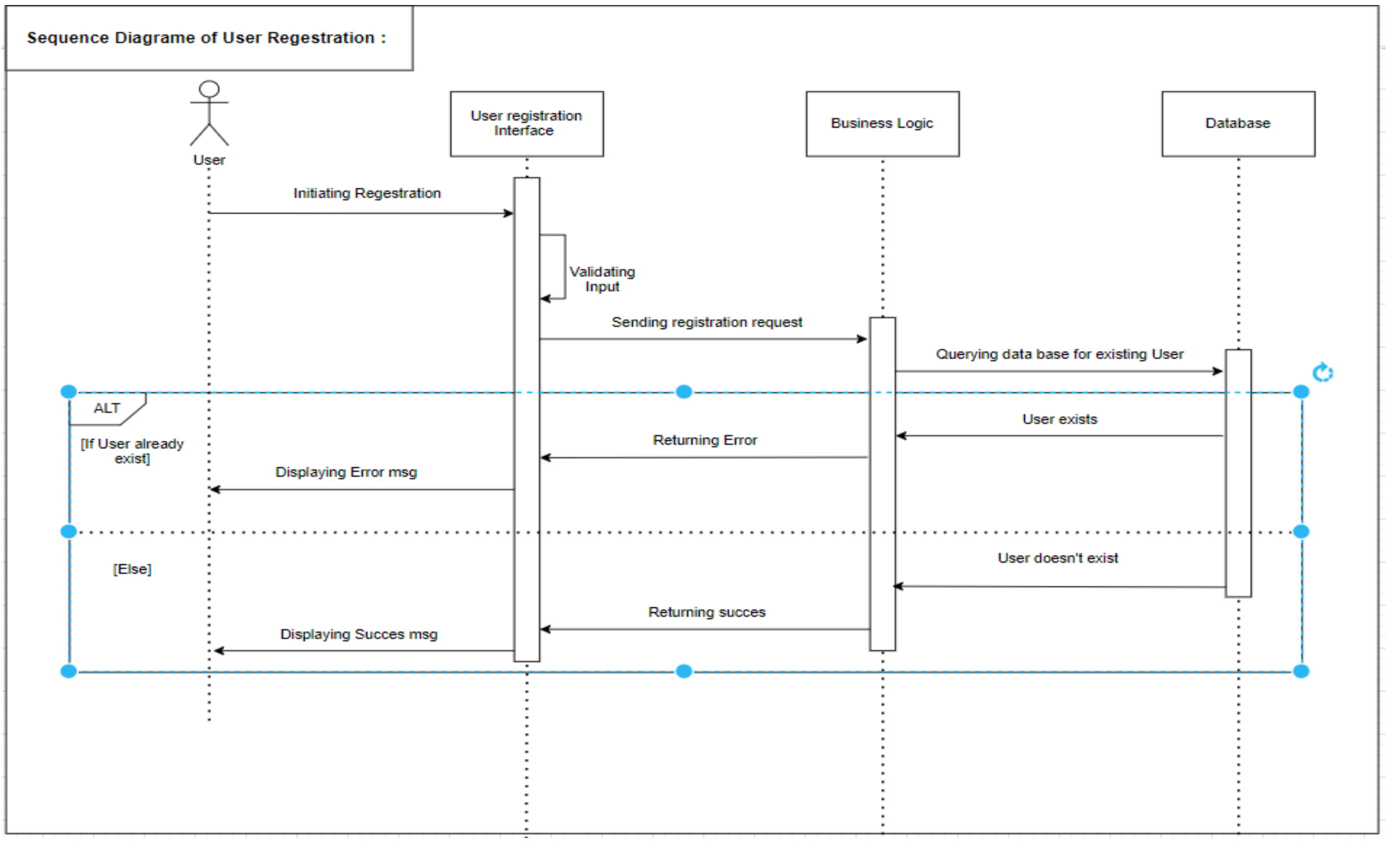
3.2 Explanation of the Data Model

- **User:** Represents registered users with attributes like id, name, email, and password.
- **Place:** Stores accommodation details such as id, name, location, description, and category.
- **Review:** Manages user reviews, including rating, comment, and user_id.
- **Amenity:** Manages operations such as inserting, modifying, and deleting records.

4. API Workflow and Interaction

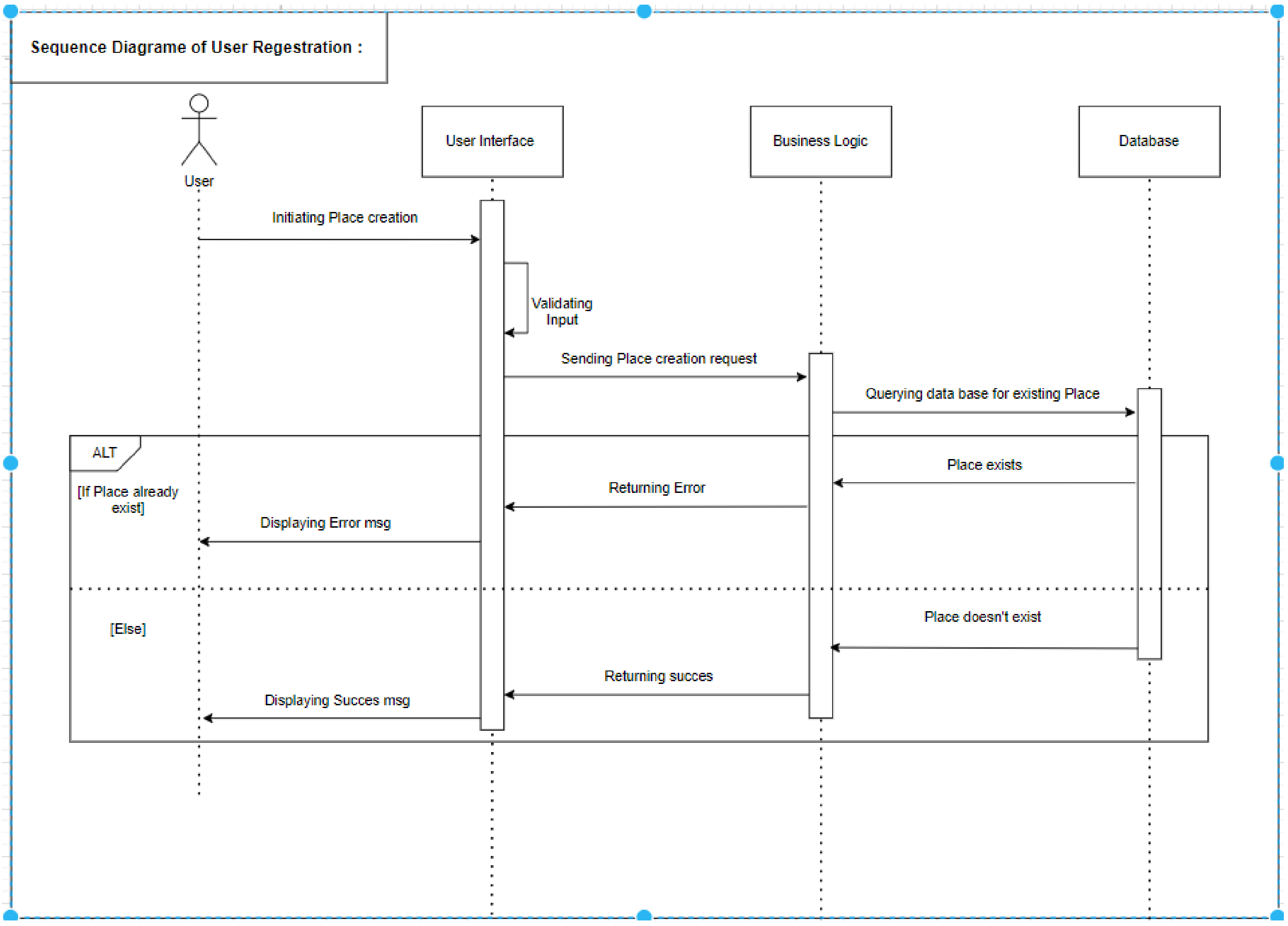
4.1 Sequence Diagrams with Descriptions

4.1.1 User Registration Process



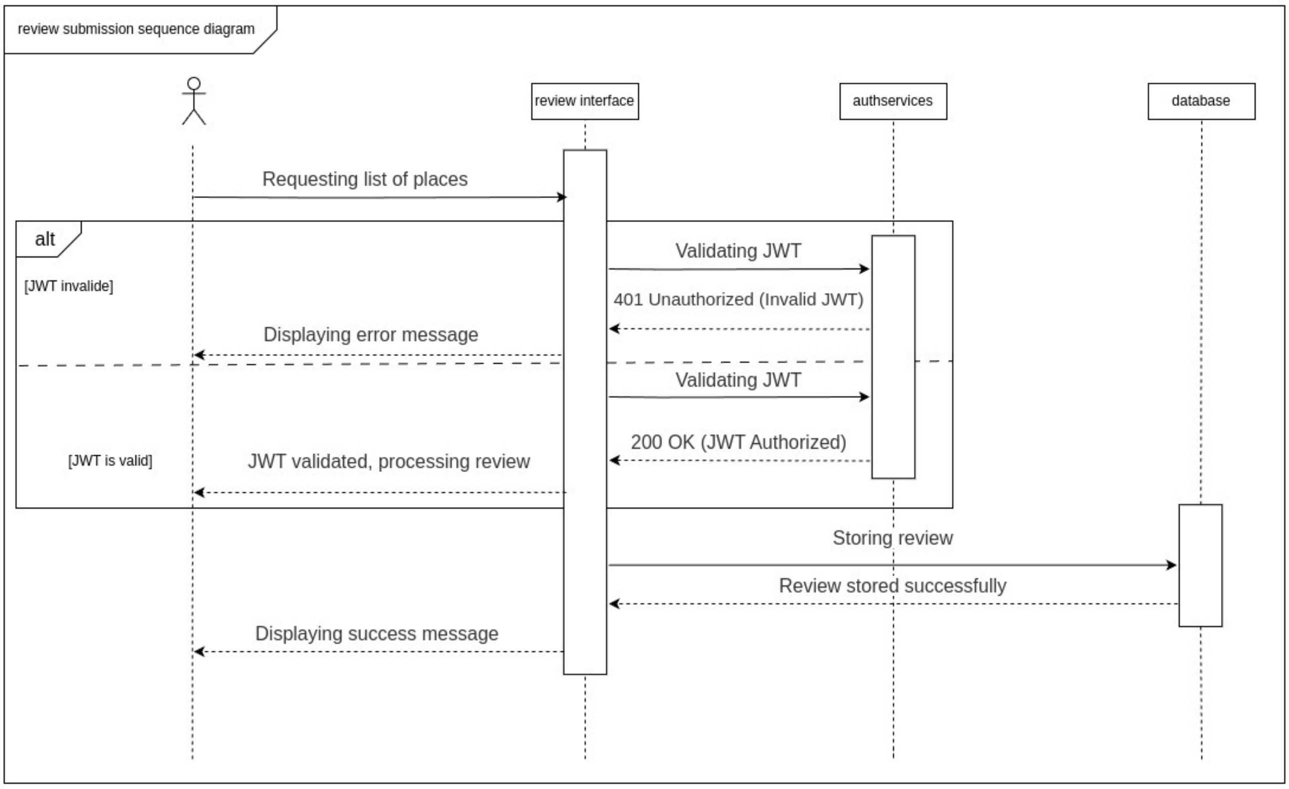
- **Purpose:** Allows users to create new accounts and store their credentials securely.
- **Key Components:** User, Client Application, Business Logic, Database.
- **Flow:**
  - **Step 1:** The user initiates the registration process by submitting a form via the client application.
  - **Step 2:** A POST /api/register request is sent to the backend.
  - **Step 3:** The API validates the user input (e.g., email format, password strength, required fields).
  - **Step 4:** Business Logic queries the database to check if the email is already in use.
  - **Step 5:** If the user already exists, an error message is returned to the client application.
  - **Step 6:** If the user does not exist, the backend hashes the password and stores user details in the database.
  - **Step 7:** If email verification is required, an email service sends a verification email to the user.
  - **Step 8:** A success response is sent back to the client application confirming the registration.
  - **Step 9:** The client application displays the appropriate success or error message to the user.

4.1.2 Place Creation Process



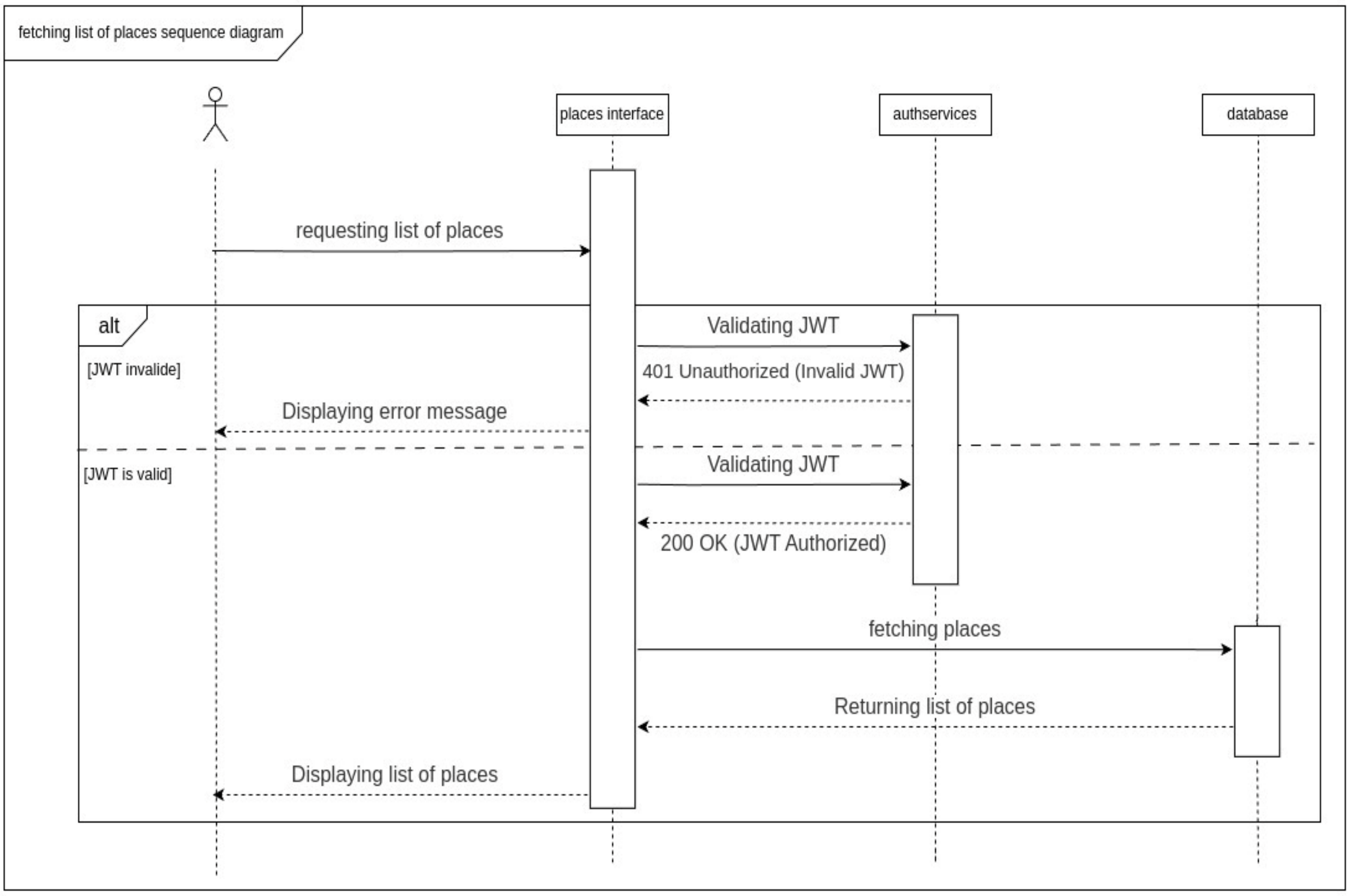
- **Purpose:** Allows users to submit new place listings and ensure data integrity.
- **Key Components:** User, Client Application, Business Logic, Database.
- **Flow:**
  - **Step 1:** The user initiates place creation by submitting details via the client application.
  - **Step 2:** A `POST /api/places` request is made to the backend.
  - **Step 3:** The API validates the input (e.g., place name, location, description, required fields).
  - **Step 4:** Business Logic queries the database to check if the place already exists.
  - **Step 5:** If the place already exists, an error message is returned to the client application.
  - **Step 6:** If the place does not exist, the backend stores the new place details in the database.
  - **Step 7:** A confirmation response is returned to the client application indicating successful creation.
  - **Step 8:** The client application updates the UI to reflect the new place entry and notifies the user.
  - **Step 9:** If additional approval or moderation is needed, the system triggers a review process (if applicable).

4.1.3 Review Submission Process


review submission sequence diagram
review interface
authservices
database
Requesting list of places
alt
[JWT invalide]
Validating JWT
401 Unauthorized (Invalid JWT)
Displaying error message
Validating JWT
[JWT is valid]
200 OK (JWT Authorized)
JWT validated, processing review
Storing review
Review stored successfully
Displaying success message

- **Purpose:** Enables users to submit reviews and ratings for places while ensuring authentication and data integrity.
- **Key Components:** User, Frontend, Authentication Service, Database.
- **Flow:**
  - **Step 1**: The user initiates the review submission process by entering a rating and comment in the frontend application.
  - **Step 2:** The frontend sends a POST /api/reviews request to submit the review along with the user's JWT (JSON Web Token) in the Authorization header.
  - **Step 3:** The authentication service verifies the JWT.
  - **Step 4:** If the JWT is invalid, the authentication service responds with 401 Unauthorized.
  - The frontend displays an error message to the user.
  - **Step 5:** If the JWT is valid, the authentication service returns 200 OK.
  - The frontend processes the review submission.
  - **Step 6**: The frontend sends a POST /api/reviews request to the database to store the review, including the user ID, place ID, rating, and comment.
  - **Step 7**: The database processes the request, inserts the review into the reviews table, and returns 201 Created upon successful storage.
  - **Step 8**: The frontend receives the response and displays a confirmation message to the.

4.1.4 Retrieving Places List



Purpose: Enables users to retrieve a list of available places while ensuring secure access control.
Key Components: User, Frontend, Authentication Service, Database.
Flow:
Step 1: The user requests a list of places via the frontend application.
Step 2: The frontend sends a GET /api/places request with the user's JWT in the Authorization header.
Step 3: The authentication service verifies the JWT.
Step 4:
If the JWT is invalid, the authentication service responds with 401 Unauthorized.
The frontend displays an error message to the user.
Step 5:
If the JWT is valid, the authentication service returns 200 OK.
The frontend proceeds with fetching the list of places.
Step 6: The frontend sends a GET /api/places request to the database to retrieve the places.
Step 7: The database executes a query on the places table and returns a JSON response containing the list of places with a 200 OK status.
Step 8: The frontend displays the retrieved list of places to the user.

5. Summary

This document serves as a structured reference for the HBnB project, ensuring clear understanding of how different system layers interact. The blueprint provided will guide the development and implementation phases effectively, ensuring maintainability and scalability.