HBnB Evolution

Part 1: Technical Documentation

System Architecture & Design

- **Badr Alshaya**
- **Yazeed Aljohani**
- **Mohammed Al-Dosari**

Table of Contents

1. INTRODUCTION

1.1 Purpose

This document delineates the comprehensive technical architecture for the **HBnB Evolution** system. It acts as the authoritative reference for the implementation phase, ensuring strict adherence to the **Layered Architecture** and **Facade Pattern**. The primary goal is to provide a rigid framework that decouples the API presentation from the core business logic and persistence mechanisms.

1.2 Project Overview

HBnB Evolution is a RESTful web application designed to clone the core functionality of AirBnB. The system facilitates a two-sided marketplace between **Owners** (listing providers) and **Customers** (consumers). The backend infrastructure handles JWT-based authentication, property management, and review aggregation within a modular environment.

2. HIGH-LEVEL ARCHITECTURE

The system architecture implements a strictly **Layered Pattern** to enforce separation of concerns. As illustrated in the diagram below, the system is stratified into three distinct execution planes.

Layer Components:

- **Presentation Layer:** Comprises API Endpoints (Controllers) for handling requests and Servicesfor lightweight processing.
- **Business Logic Layer:** Encapsulates the core models (Place, User, Review, Amenity) and business rules.
- **Persistence Layer:** Manages data durability via Storage interface, implementing both Databaseand Repository patterns.

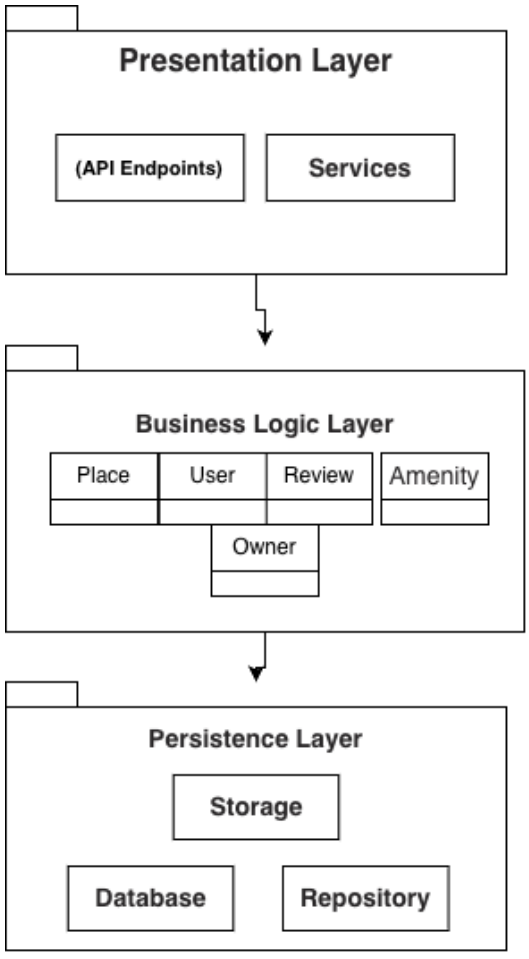

Figure 1: High-Level Package Diagram depicting the Layered Architecture, illustrating the strict downward dependency flow from Presentation to Persistence.

3. BUSINESS LOGIC LAYER

3.1 Model Architecture & Design Rationale

The Business Logic Layer acts as the system's deterministic core, implementing the **Facade Pattern** to decouple the presentation endpoints from the data persistence mechanisms. This layer is responsible for entity lifecycle management, data integrity validation, and business rule enforcement.

By abstracting the storage implementation, the domain model ensures that business operations are persistence-agnostic. The architecture utilizes **Generalization (Inheritance)** to enforce strict typing and minimize code redundancy (DRY Principle), specifically within the user identity management module.

3.2 Detailed Class Design

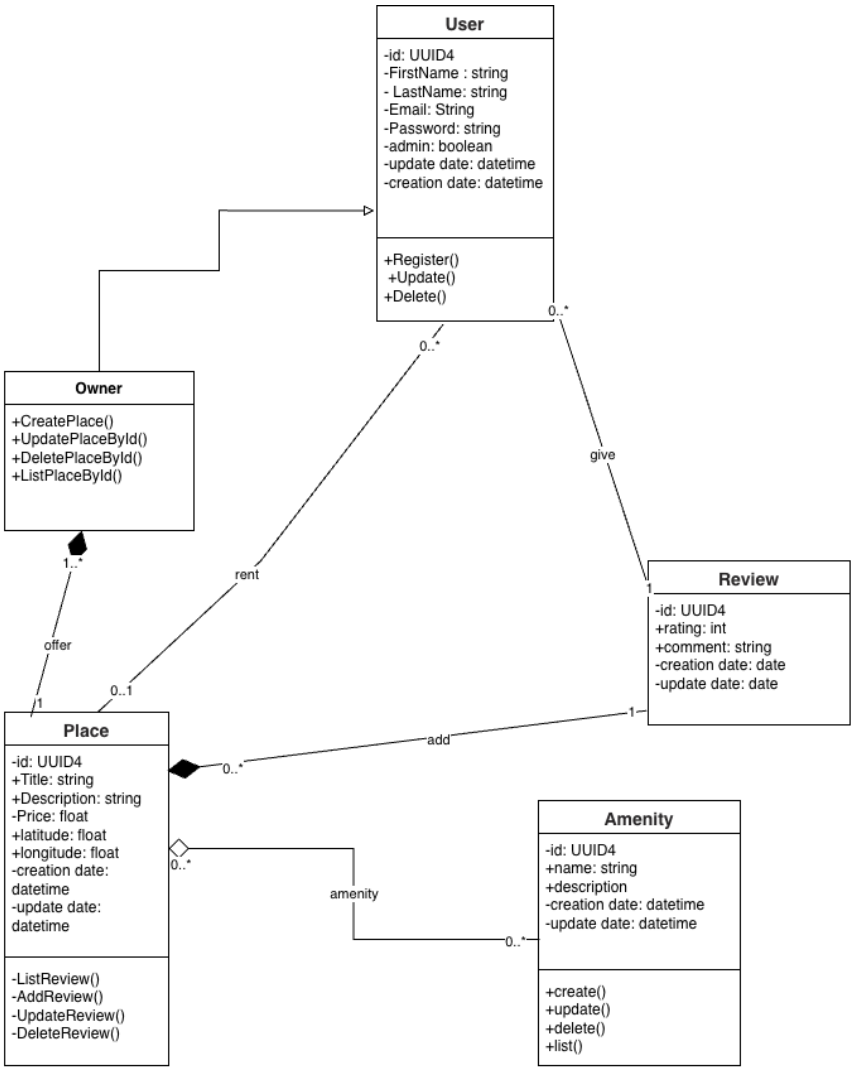

Figure 2: Business Logic Class Diagram illustrating the Domain Object Model, Entity Relationships, and Cardinality constraints.

3. BUSINESS LOGIC LAYER (Cont.)

3.3 Entities & Relationships Analysis

The domain model is structured around key entities defined by their specific roles and interaction constraints:

- **User (Base Entity & Consumer):** Represents the primary authenticated actor within the system. Unlike a purely abstract class, the User entity functionally serves as the **Service Consumer (Guest)**.
    - **Capabilities:** Encapsulates identity attributes (id, email, password) and maintains direct associations with **Reviews** (authorship) and **Places** (booking/renting).
- **Owner (Specialization):** Inherits from User to extend the base capabilities with **Service Provider** privileges.
    - **Relationship:** Maintains a One-to-Many composition with **Place** (1 Owner ↔ 0..* Places). This enforces strict ownership liability, distinguishing an Owner from a standard User.
- **Place (Resource Aggregate):** The central entity representing the listing. It acts as an aggregate root for related descriptors.
    - **Geospatial Data:** Stores latitude and longitude for location services.
    - **Amenity Association:** Interacts via an aggregation relationship with **Amenity**. Modeled to ensure that deleting a Place does not purge Amenities from the global dictionary.
    - **Review Composition:** Composes **Reviews** via a One-to-Many relationship.
- **Review (Association Entity):** Functions as a link between a **User** and a **Place**. It enforces referential integrity, ensuring a review cannot exist without a valid author (User) and a target property.
- **Amenity (Lookup Entity):** Represents standardized features (e.g., Pool, WiFi). Designed for high reusability across multiple Places via an aggregation model.

4. API Interaction Flow

The sequence diagrams below depict the runtime interaction between the Application Layers. The flow adheres to the pattern: **User → API → BusinessLogic → Database**.

4.1 User Registration

Description: The API delegates the registration request to the BusinessLogic layer. The logic layer validates credentials before persisting the new User entity.

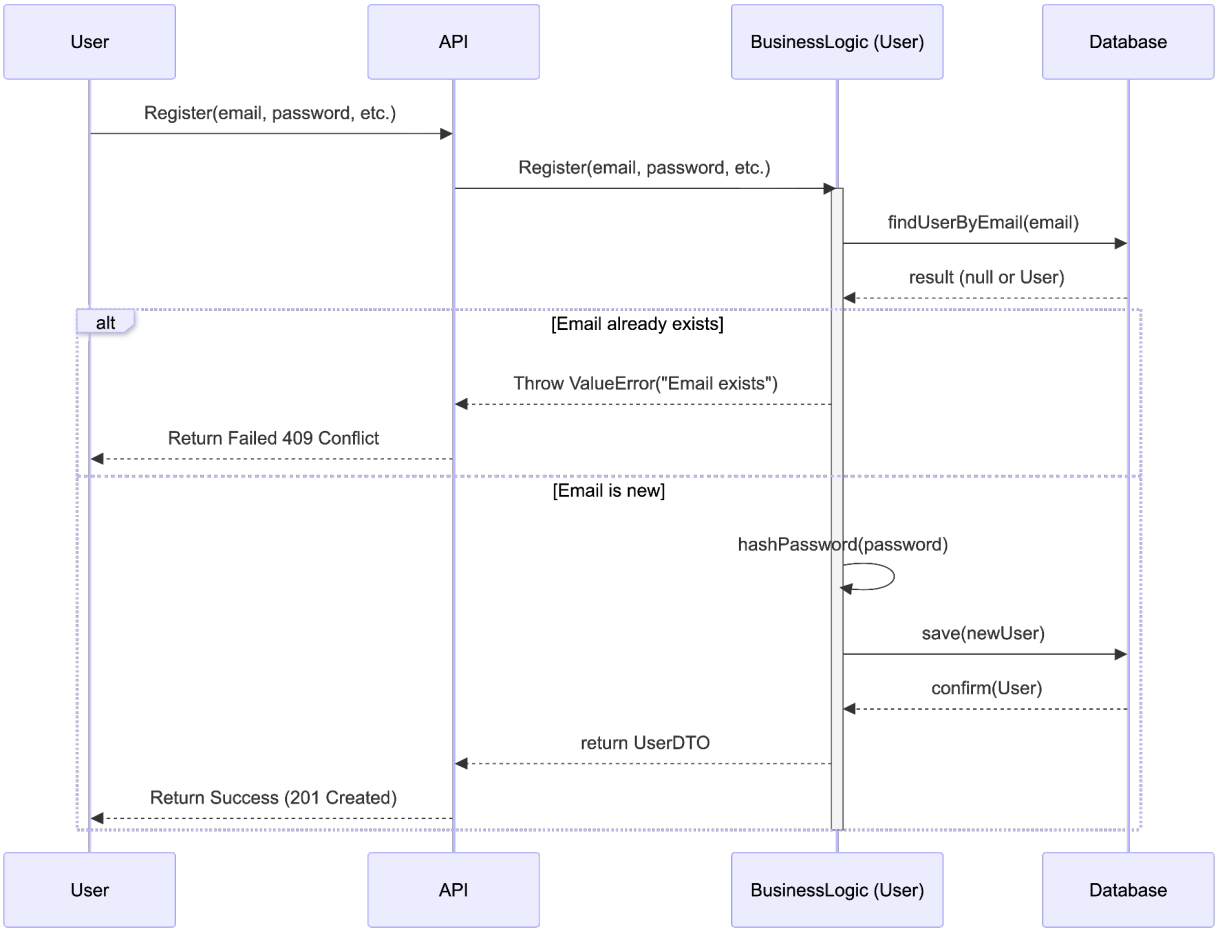

Figure 3: Sequence Diagram for User Registration, detailing the synchronous flow of credential validation, password hashing, and entity persistence.

4. API Interaction Flow (Cont.)

4.2 Place Creation

Description: The API validates the User's role (must be Owner) and input integrity via the BusinessLogic layer before persisting the new Place entity to the database.

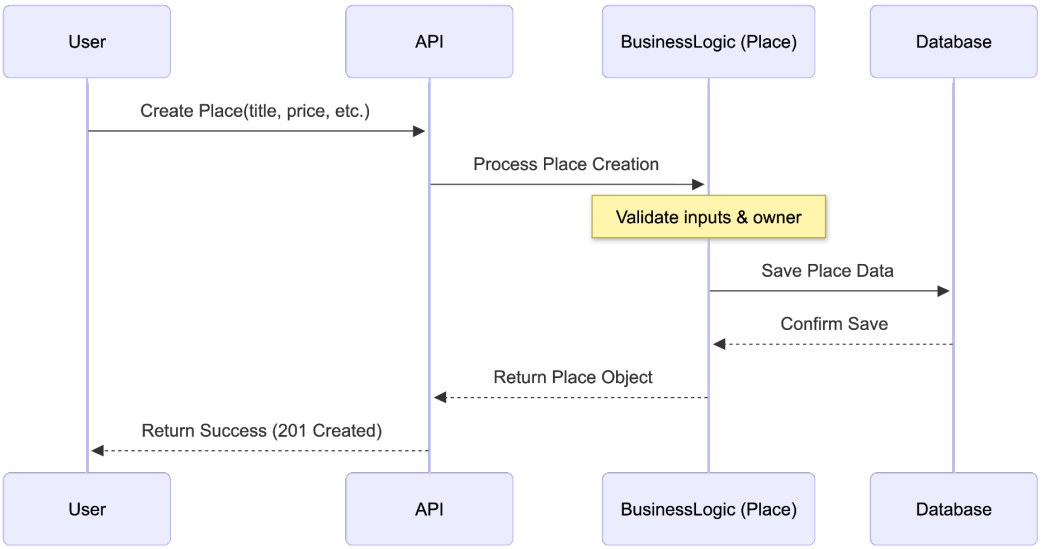

Figure 4: Sequence Diagram for Place Creation, illustrating the Role-Based Access Control (RBAC) enforcement and data propagation to storage.

4.3 Review Submission

Description: When submitting a review, the BusinessLogic layer performs a dependency check to verify the target Place exists. Upon validation, the review is saved.

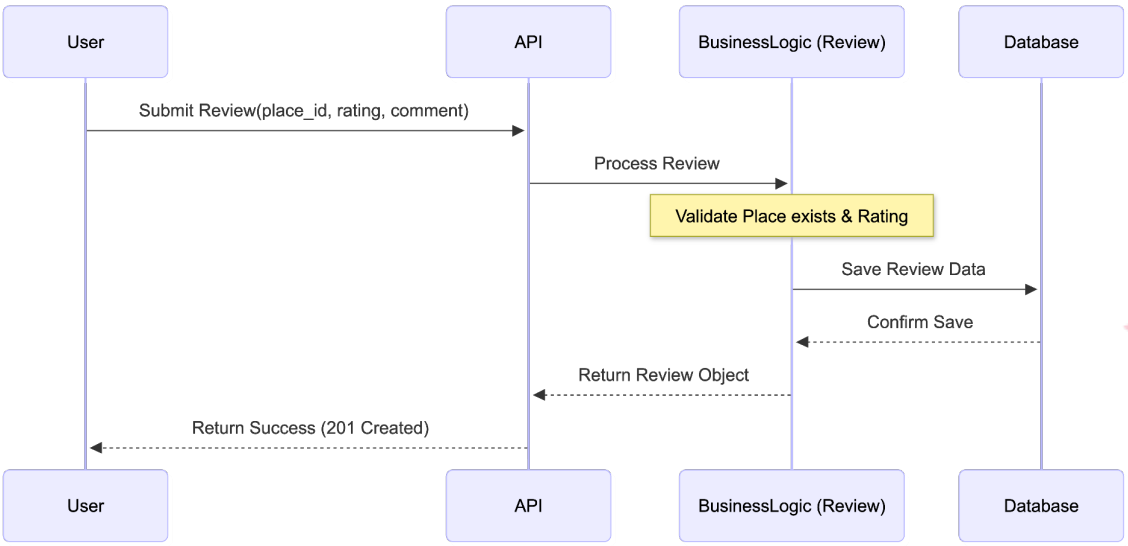

Figure 5: Sequence Diagram for Review Submission, emphasizing the referential integrity validation logic between the Customer and the target Place.

4. API Interaction Flow (Cont.)

4.4 Fetching a List of Places

Description: A read-only operation where the API delegates filtering to the BusinessLogic layer. The system queries the Persistence layer for Place entities and returns a serialized list to the client.

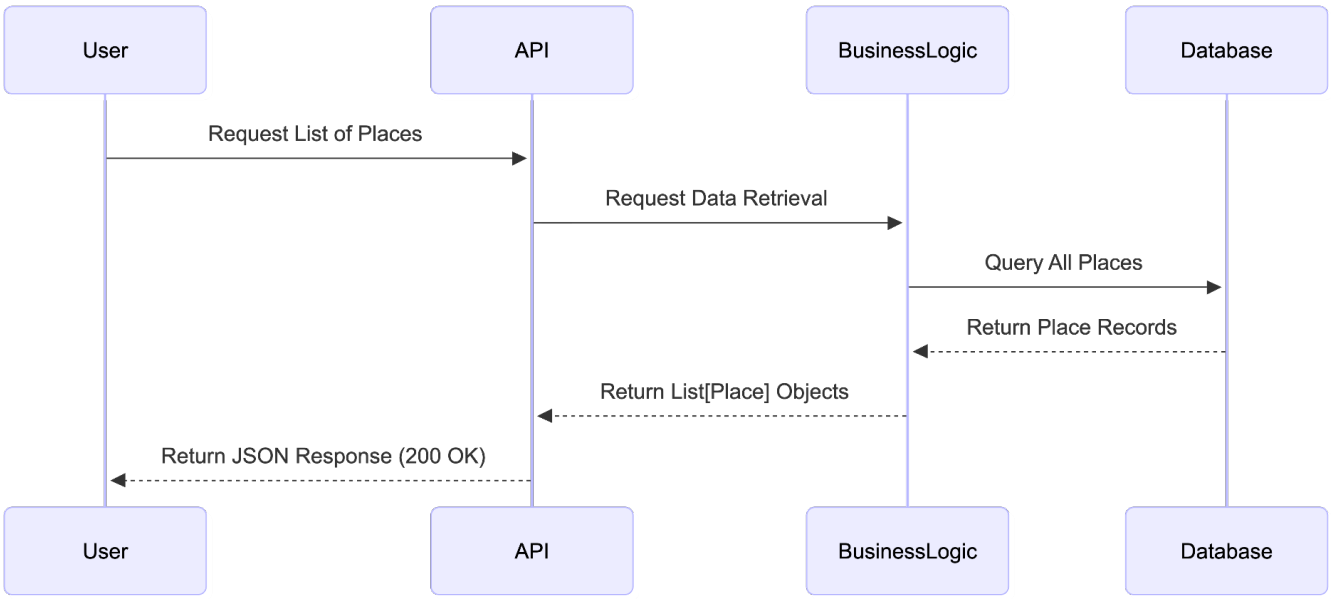

Figure 6: Sequence Diagram for Data Retrieval, demonstrating the query delegation process, database lookup, and DTO serialization pipeline.