Holberton Coding School

HBnB - UML

Hector M. Ruiz
http://www.github.com/hruiz1191
C25, Ponce
Feb 16, 2025

Introduction

This document serves as a comprehensive technical blueprint for the HBnB project, detailing its architecture, business logic, and API interactions. It provides a structured approach to understanding the system's design, ensuring clarity for implementation and future development.

All diagrams in this document have been created using Mermaid.js, a powerful tool for generating UML-style diagrams in a text-based format. The document includes:

- A High-Level Package Diagram illustrating the three-layer architecture and Facade Pattern.
- A Detailed Class Diagram defining the Business Logic Layer, including key entities and relationships.
- Sequence Diagrams depicting API interactions and the data flow between system components.

By providing these visual representations, this document acts as a reference guide for developers, ensuring a scalable, maintainable, and well-structured system.

Project Overview

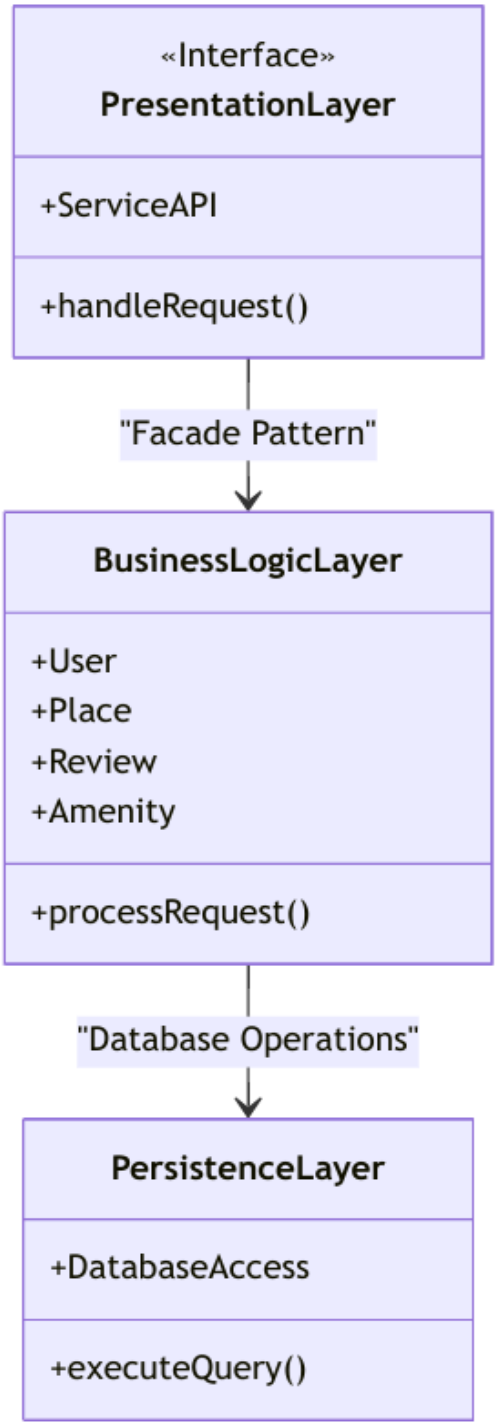
The HBnB project is a full-stack web application designed to facilitate property rentals, where users can register, create listings, and submit reviews. The system follows a three-layer architecture:

1. Presentation Layer – Handles user interactions via APIs and services.
2. Business Logic Layer – Manages the core models and application logic.
3. Persistence Layer – Stores and retrieves data from the database.

To improve modularity and maintainability, the system utilizes the Facade Pattern, simplifying interactions between layers.

This technical document serves as a blueprint for the project, guiding the implementation process with structured diagrams, explanations, and data flow analysis. By following this documentation, developers can ensure a clear, efficient, and scalable implementation of HBnB.

Task 0: High-Level Package Diagram



Objective

Create a high-level package diagram that illustrates the three-layer architecture of the HBnB application and the communication between these layers via the facade pattern. This diagram will provide a conceptual overview of how the different components of the application are organized and how they interact with each other.

Overview

This section presents the High-Level Package Diagram of the HBnB project, showcasing its three-layer architecture: Presentation Layer, Business Logic Layer, and Persistence Layer. The system follows the Facade Pattern, ensuring a structured and maintainable interaction between these layers. The diagram highlights how each layer communicates, emphasizing the clear separation of concerns to improve scalability and maintainability.

```
classDiagram
%% Definición de las capas de la arquitectura

class PresentationLayer {
   <<Interface>>
   +ServiceAPI
   +handleRequest()
}

class BusinessLogicLayer {
   +User
   +Place
   +Review
   +Amenity
   +processRequest()
}

class PersistenceLayer {
   +DatabaseAccess
   +executeQuery()
}

%% Relación entre las capas utilizando el patrón Facade

PresentationLayer --> BusinessLogicLayer : "Facade Pattern"
BusinessLogicLayer --> PersistenceLayer : "Database Operations"
```

Diagram Explanation

Presentation Layer

- Exposes a service (`ServiceAPI`) that handles user requests.
- Calls `handleRequest()` to process incoming requests and delegates them to the business logic layer through the **Facade Pattern**.

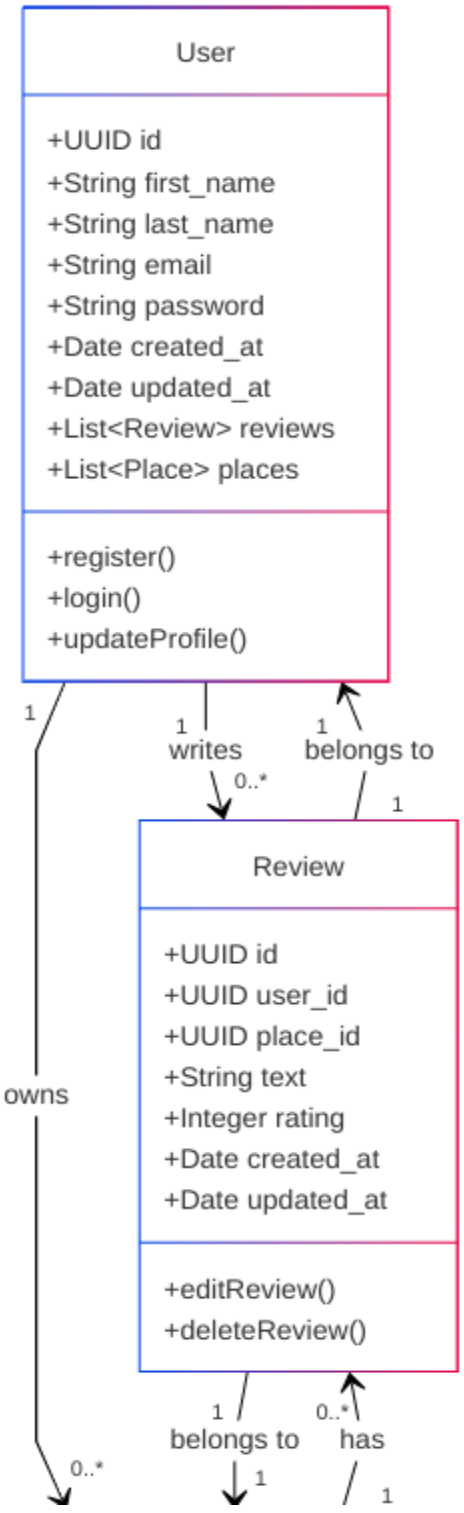
Business Logic Layer

- Defines the core data models (`User`, `Place`, `Review`, `Amenity`).
- Contains methods like `processRequest()`, which implement the application's business logic.
- Communicates with the persistence layer to retrieve or store data in the database.

Persistence Layer

- Contains the `DatabaseAccess` class, which executes queries and manages data persistence.
- Exposes `executeQuery()` to perform read and write operations on the database.

Task 1: Detailed Class Diagram for Business Logic Layer



Objective

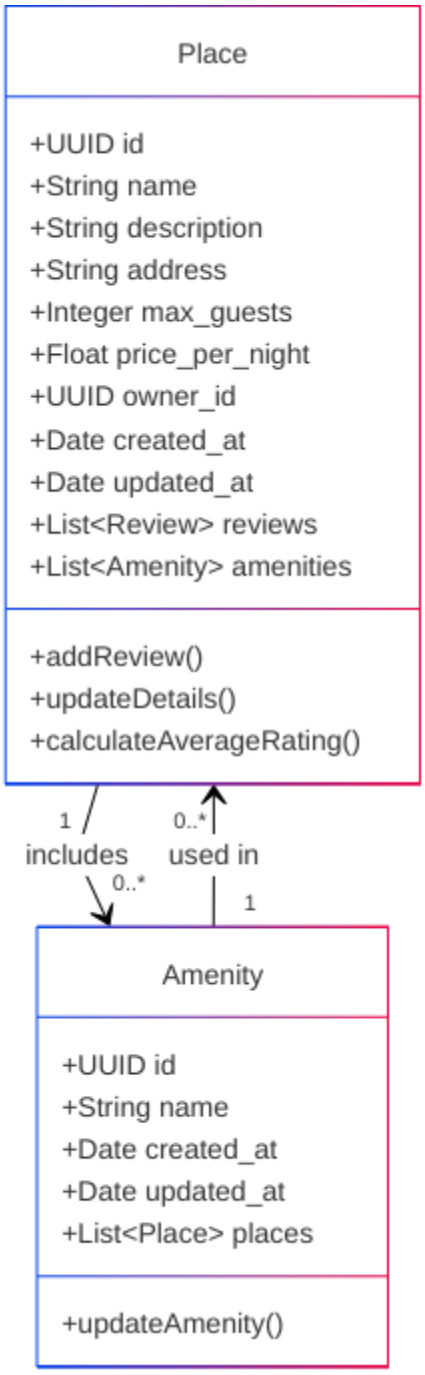
Design a detailed class diagram for the Business Logic layer of the HBnB application. This diagram will depict the entities within this layer, their attributes, methods, and the relationships between them. The primary goal is to provide a clear and detailed visual representation of the core business logic, focusing on the key entities: User, Place, Review, and Amenity.

Overview

This section provides a **detailed class diagram** for the **Business Logic Layer** of the HBnB application. It defines the **core entities** (`User`, `Place`, `Review`, `Amenity`) along with their **attributes, methods, and relationships**. The diagram follows **Object-Oriented Design (OOD) principles**, ensuring **modularity and reusability**. The relationships between models, including **associations, compositions, and generalizations**, are visualized to clarify how data flows within the business logic. This class structure serves as the foundation for implementing the core functionalities of the system.

classDiagram

```
%% User Class
class User {
    +UUID id
    +String first_name
    +String last_name
    +String email
    +String password
    +Date created_at
    +Date updated_at
    +List~Review~ reviews
    +List~Place~ places
    +register()
    +login()
    +updateProfile()
}

%% Place Class
class Place {
    +UUID id
    +String name
```




```
    +String description
    +String address
    +Integer max_guests
    +Float price_per_night
    +UUID owner_id
    +Date created_at
    +Date updated_at
    +List~Review~ reviews
    +List~Amenity~ amenities
    +addReview()
    +updateDetails()
    +calculateAverageRating()
}

%% Review Class
class Review {
    +UUID id
    +UUID user_id
    +UUID place_id
    +String text
    +Integer rating
    +Date created_at
    +Date updated_at
    +editReview()
    +deleteReview()
}

%% Amenity Class
class Amenity {
    +UUID id
    +String name
    +Date created_at
    +Date updated_at
    +List~Place~ places
    +updateAmenity()
}
```

```
%% Relationships
User "1" --> "0..*" Review : writes
User "1" --> "0..*" Place : owns
Place "1" --> "0..*" Review : has
Place "1" --> "0..*" Amenity : includes
Review "1" --> "1" User : belongs to
Review "1" --> "1" Place : belongs to
Amenity "1" --> "0..*" Place : used in
```

Explanation of Each Entity

1. User Class

- Represents a user of the platform.
- Stores essential user details (`first_name`, `last_name`, `email`, `password`).
- Manages relationships with **Review** (users write reviews) and **Place** (users own places).
- Includes methods:
  - `register()`: Register a new user.
  - `login()`: Authenticate user.
  - `updateProfile()`: Modify user details.

2. Place Class

- Represents a rental property.
- Stores details such as `name`, `description`, `address`, `max_guests`, and `price_per_night`.
- Connected to:
  - **User** (A user owns places).
  - **Review** (A place can have multiple reviews).
  - **Amenity** (A place can have multiple amenities).
- Key methods:
  - `addReview()`: Adds a review to the place.
  - `updateDetails()`: Updates the property information.
  - `calculateAverageRating()`: Computes the average rating from all reviews.

3. Review Class

- Represents a user review of a place.
- Stores details like `text` and `rating`.
- Connected to:
  - **User** (Each review belongs to one user).
  - **Place** (Each review belongs to one place).
- Key methods:
  - `editReview()`: Updates an existing review.
  - `deleteReview()`: Removes a review.

4. Amenity Class

- Represents amenities available in a place.

- Stores the `name` of the amenity.
- Connected to:
  - **Place** (A place can have multiple amenities).
- Key method:
  - `updateAmenity()`: Updates the name of the amenity.

Explanation of Relationships

1. **User - Review (1 to Many)**
   - A user can write multiple reviews.
   - Each review belongs to one user.
2. **User - Place (1 to Many)**
   - A user can own multiple places.
   - Each place is owned by one user.
3. **Place - Review (1 to Many)**
   - A place can have multiple reviews.
   - Each review is associated with one place.
4. **Place - Amenity (Many to Many)**
   - A place can have multiple amenities.
   - An amenity can be present in multiple places.
5. **Review - User & Place (Many to One)**
   - A review is linked to one user and one place.

Task 2: Sequence Diagrams for API Calls

Objective

Develop sequence diagrams for at least four different API calls to illustrate the interaction between the layers (Presentation, Business Logic, Persistence) and the flow of information within the HBnB application. The sequence diagrams will help visualize how different components of the system interact to fulfill specific use cases, showing the step-by-step process of handling API requests.

Overview

This section presents **sequence diagrams** for four key **API interactions** in the HBnB application:

1. **User Registration** – A new user creates an account.
2. **Place Creation** – A user lists a new place.
3. **Review Submission** – A user submits a review for a place.
4. **Fetching Places** – A user requests a list of available places.

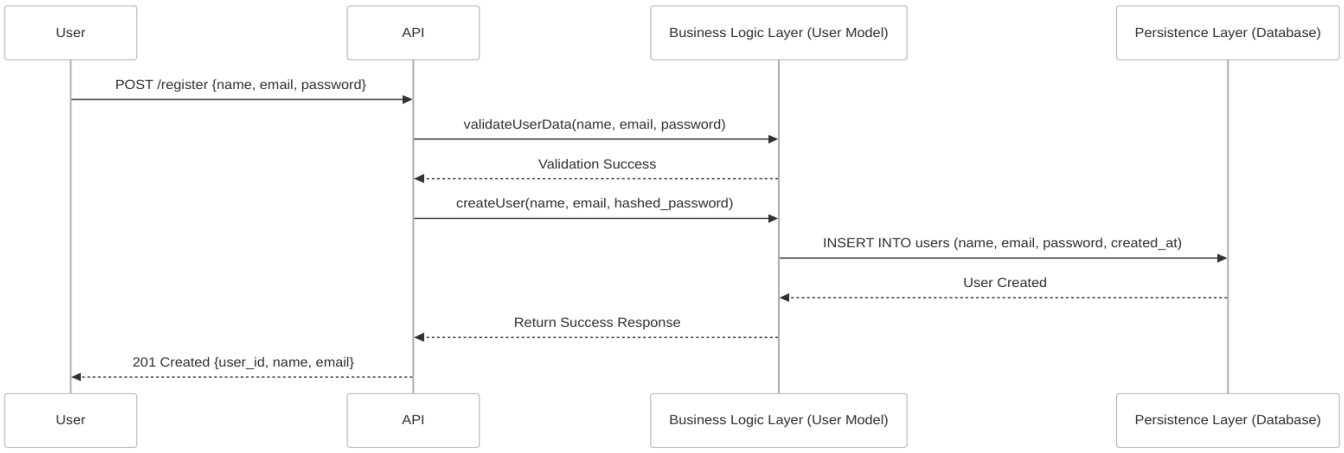
Each diagram illustrates the **flow of interactions** between the **Presentation Layer (API), Business Logic Layer (Models), and Persistence Layer (Database)**. The **Facade Pattern** ensures smooth communication between layers, making the system **scalable and efficient**. These sequence diagrams

I. User Registration

Overview

The User Registration API allows new users to create an account. The process includes validating user input, encrypting passwords, and storing user details in the database. The system ensures data integrity and security before returning a success response.

Description: A new user signs up for an account. The request goes through the API, which validates the input, creates the user, and stores the data in the database.



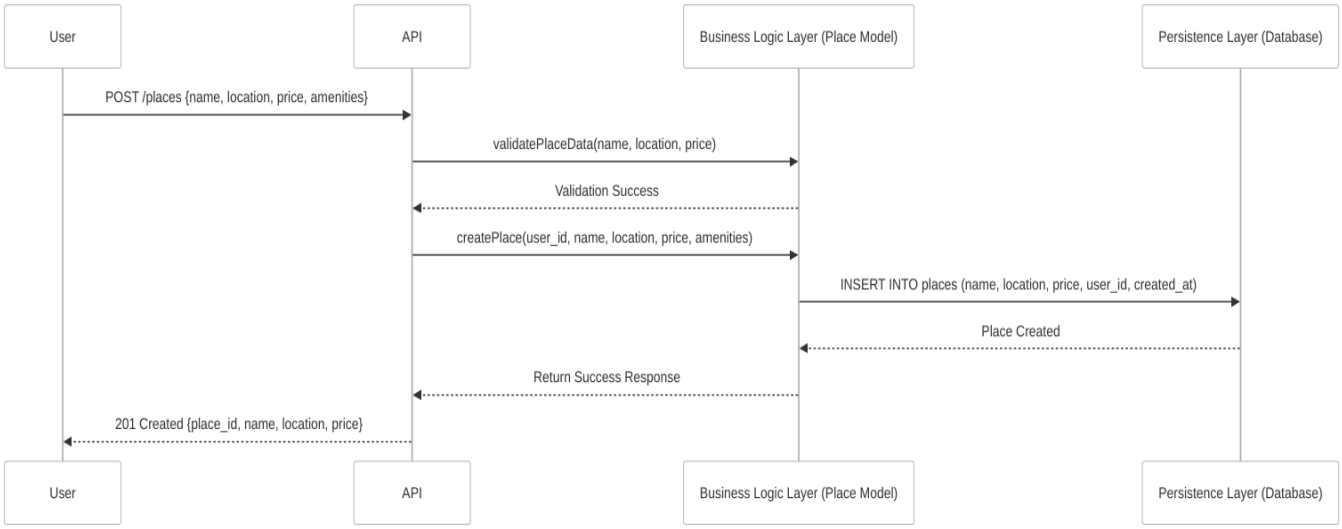
```
sequenceDiagram
    participant User
    participant API
    participant BusinessLogic as Business Logic Layer (User Model)
    participant Database as Persistence Layer (Database)

    User->>API: POST /register {name, email, password}
    API->>BusinessLogic: validateUserData(name, email, password)
    BusinessLogic-->>API: Validation Success
    API->>BusinessLogic: createUser(name, email, hashed_password)
    BusinessLogic->>Database: INSERT INTO users (name, email, password, created_at)
    Database-->>BusinessLogic: User Created
    BusinessLogic-->>API: Return Success Response
    API-->>User: 201 Created {user_id, name, email}
```

II. Place Creation

Overview

The Place Creation API enables users to list new properties. It verifies input data, associates the listing with the user, and saves it in the database. This API ensures that only authenticated users can create listings, maintaining system integrity.

Description: A user creates a new place listing. The request includes details such as name, location, price, and amenities. The API validates the data, creates the listing, and stores it in the database.



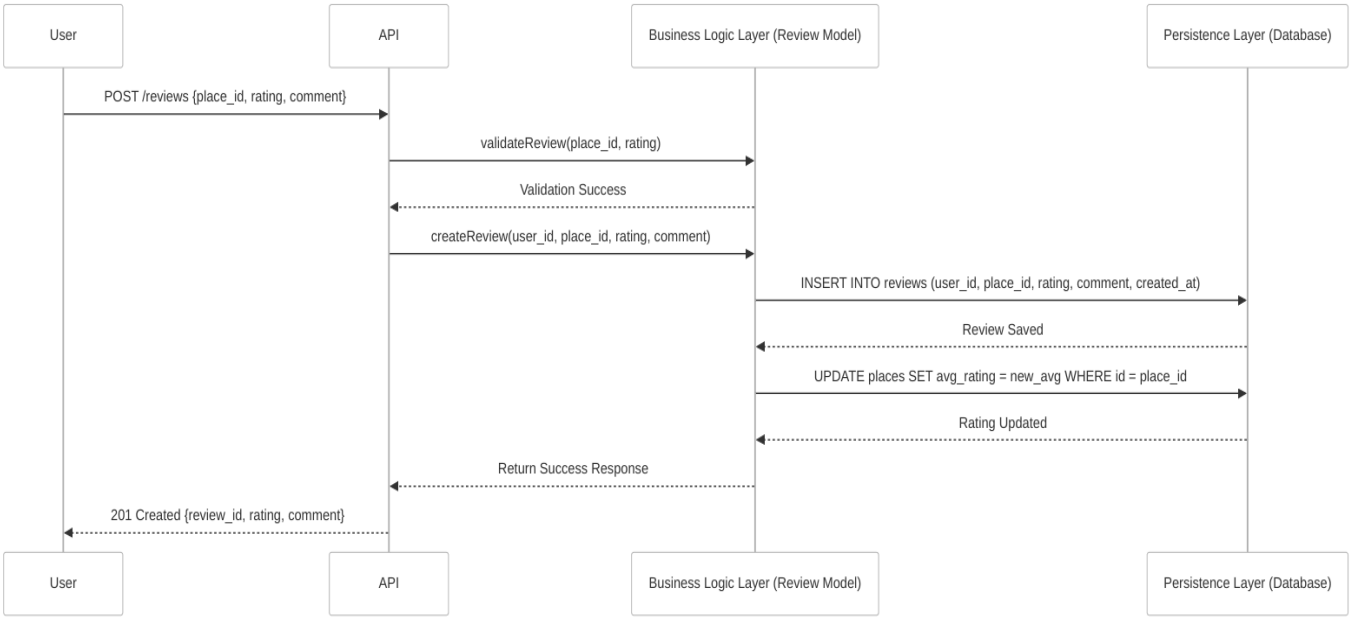
```
sequenceDiagram
    participant User
    participant API
    participant BusinessLogic as Business Logic Layer (Place Model)
    participant Database as Persistence Layer (Database)

    User->>API: POST /places {name, location, price, amenities}
    API->>BusinessLogic: validatePlaceData(name, location, price)
    BusinessLogic-->>API: Validation Success
    API->>BusinessLogic: createPlace(user_id, name, location, price, amenities)
    BusinessLogic->>Database: INSERT INTO places (name, location, price, user_id, created_at)
    Database-->>BusinessLogic: Place Created
    BusinessLogic-->>API: Return Success Response
    API-->>User: 201 Created {place_id, name, location, price}
```

III. Review Submission

Overview

The Review Submission API allows users to submit ratings and comments for a place. It checks if the place exists, ensures valid rating values, saves the review, and updates the average rating for the place.

Description: A user submits a review for a place. The API validates that the place exists, stores the review, and updates the place's rating.



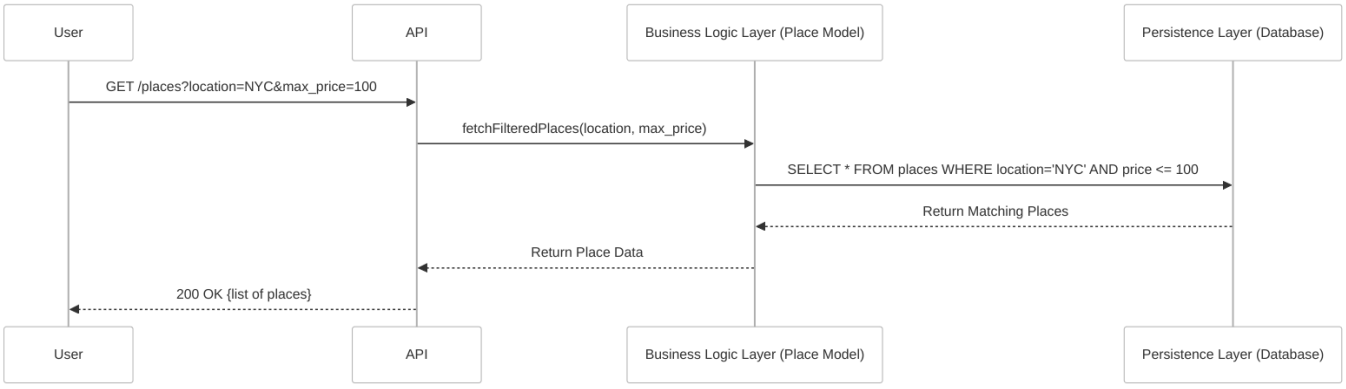
```
sequenceDiagram
    participant User
    participant API
    participant BusinessLogic as Business Logic Layer (Review Model)
    participant Database as Persistence Layer (Database)

    User->>API: POST /reviews {place_id, rating, comment}
    API->>BusinessLogic: validateReview(place_id, rating)
    BusinessLogic-->>API: Validation Success
    API->>BusinessLogic: createReview(user_id, place_id, rating, comment)
    BusinessLogic->>Database: INSERT INTO reviews (user_id, place_id, rating, comment, created_at)
    Database-->>BusinessLogic: Review Saved
    BusinessLogic->>Database: UPDATE places SET avg_rating = new_avg WHERE id = place_id
    Database-->>BusinessLogic: Rating Updated
    BusinessLogic-->>API: Return Success Response
    API-->>User: 201 Created {review_id, rating, comment}
```

IV. Fetching a List of Places

Overview

The Fetching Places API retrieves a list of places based on filters such as location and price. It queries the database for matching results and returns structured data, ensuring fast and efficient searches for users.

Description: A user requests a list of places based on search criteria (e.g., price range, location, amenities). The API retrieves and filters data from the database.



```
sequenceDiagram
    participant User
    participant API
    participant BusinessLogic as Business Logic Layer (Place Model)
    participant Database as Persistence Layer (Database)

    User->>API: GET /places?location=NYC&max_price=100
    API->>BusinessLogic: fetchFilteredPlaces(location, max_price)
    BusinessLogic->>Database: SELECT * FROM places WHERE location='NYC' AND price <= 100
    Database-->>BusinessLogic: Return Matching Places
    BusinessLogic-->>API: Return Place Data
    API-->>User: 200 OK {list of places}
```

Explanations & Key Interactions

1. **User Registration:**
    - The user sends a `POST` request with their details.
    - The API validates the data before creating the user.
    - The new user is stored in the database, and a success response is returned.
2. **Place Creation:**
    - A user submits a request to add a new place.
    - The API validates and passes the request to the **Business Logic Layer**.
    - The place is stored in the database, and the API responds with the new **place ID**.
3. **Review Submission:**
    - A user submits a review for a place.
    - The API verifies the place exists, stores the review, and updates the **average rating** of the place.
4. **Fetching Places:**
    - The user requests a list of places.
    - The API queries the **Persistence Layer** to find matches based on filters (e.g., location, price).
    - The API returns the results as a JSON response.

Conclusion

This technical document provides a structured and detailed overview of the **HBnB project**, covering its **high-level architecture, business logic, and API interactions**. By organizing the system into **three layers**—Presentation, Business Logic, and Persistence—it ensures **scalability, maintainability, and clear separation of concerns**. The **Facade Pattern** simplifies communication between layers, improving efficiency. The class and sequence diagrams clarify relationships and data flow, aiding implementation. This document serves as a **valuable reference**, guiding developers in building a robust and well-structured system. Its clarity and consistency will streamline development, ensuring the project aligns with its design principles and objectives.