Introduction

The HBnB project is a web application designed as a clone of Airbnb, providing users with the ability to create accounts, list accommodations, submit reviews, and browse available places. The primary goal of this project is to develop a **scalable, maintainable, and well-structured application** that models real-world booking and review systems.

This document serves as a **comprehensive technical blueprint** for HBnB, combining high-level architecture, detailed business logic, and interaction flows between components. It provides:

- A **High-Level Package Diagram** illustrating the three-layer architecture (Presentation, Business Logic, and Persistence) and the use of the **Facade Pattern**.
- A **Detailed Class Diagram** for the Business Logic Layer, capturing the entities (`User`, `Place`, `Review`, `Amenity`), their attributes, methods, and relationships.
- **Sequence Diagrams for Key API Calls**, demonstrating how requests flow through the system, from the client to the database and back.

This document is intended as a **reference for developers and stakeholders**, ensuring clarity in the system's design, guiding implementation, and facilitating future maintenance and scalability of the HBnB application.

HBnB Application: Three-Layer Architecture & Facade Pattern

Objective

This document presents a high-level **package diagram** illustrating the **three-layer architecture** of the HBnB application and the **communication between layers** via the **facade pattern**. It provides a conceptual overview of the application's organization and interactions.

1. Layered Architecture Overview

The HBnB application is structured into three main layers:

1.1 Presentation Layer

- **Purpose:** Handles interaction between users and the application.
- **Components:**
  - Services: `UserService`, `PlaceService`, `ReviewService`
  - API endpoints
- **Responsibilities:**
  - Accepts user requests
  - Forwards requests to the **Business Logic Layer** via a **facade**
  - Returns responses to users

1.2 Business Logic Layer

- **Purpose:** Implements the core business rules and manages entities.
- **Components:**
  - Models: `User`, `Place`, `Review`, `Amenity`
  - Facade: `HBnBFacade`
- **Responsibilities:**
  - Process business operations
  - Interact with **Persistence Layer** for data storage/retrieval
  - Expose a simplified interface to the **Presentation Layer**

1.3 Persistence Layer

- **Purpose:** Handles data storage and retrieval.
- **Components:**
  - Data Access Objects (DAOs): `UserDAO`, `PlaceDAO`, `ReviewDAO`, `AmenityDAO`
- **Responsibilities:**
  - Perform CRUD operations on the database
  - Serve requests from the **Business Logic Layer**

---

2. Facade Pattern

The **facade pattern** is used to simplify interactions between the **Presentation Layer** and the **Business Logic Layer**:

- Provides a **single interface** (`HBnBFacade`) for all presentation services.
- Hides the complexity of models and persistence operations.
- Example methods: `createUser()`, `getPlaceDetails()`, `addReview()`.

---

3. Key Components by Layer

Layer	Components
Presentation Layer	`UserService`, `PlaceService`, `ReviewService`, API Endpoints
Business Logic Layer	`HBnBFacade`, `User`, `Place`, `Review`, `Amenity`
Persistence Layer	`UserDAO`, `PlaceDAO`, `ReviewDAO`, `AmenityDAO`

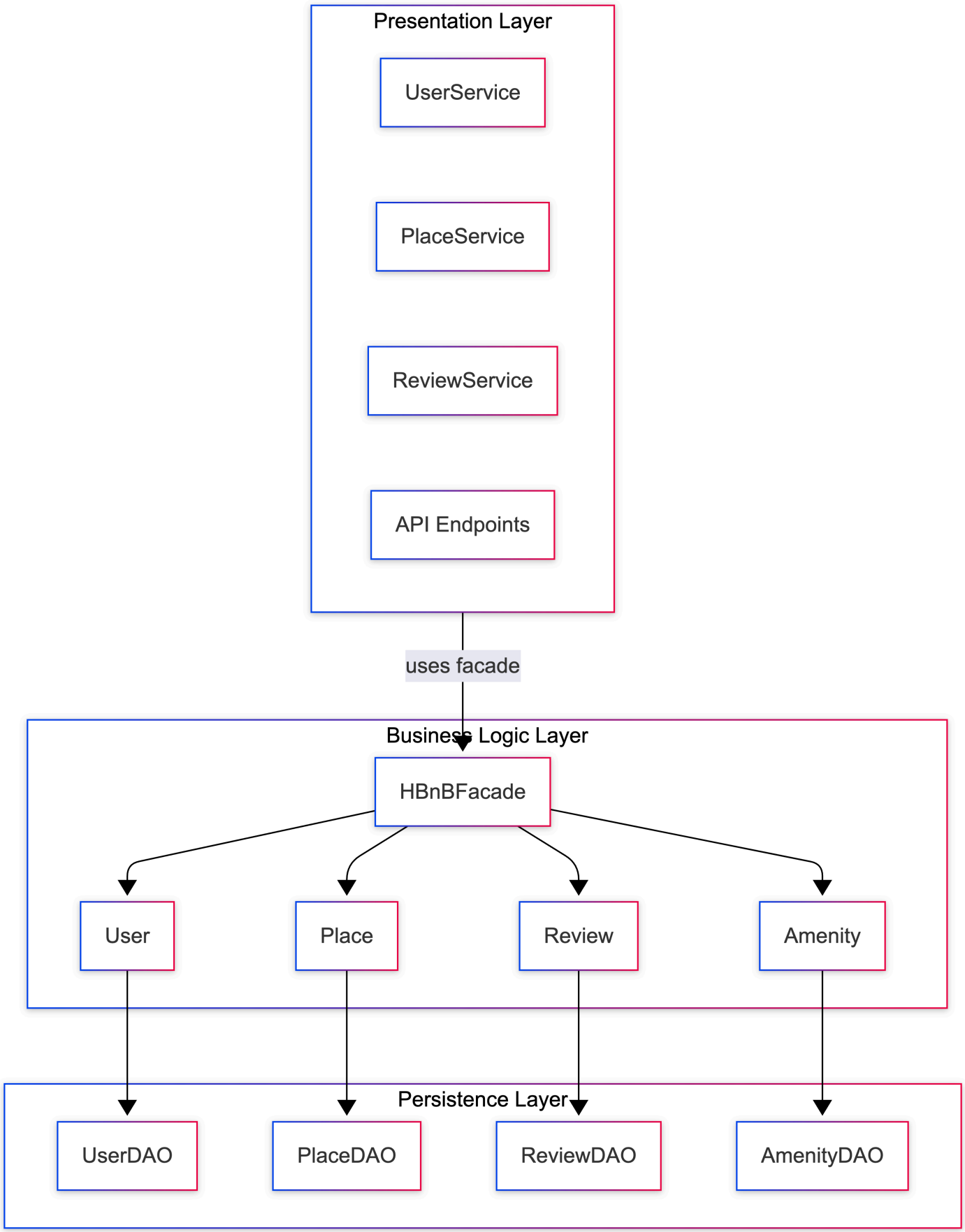
---

4. Communication Flow

1. **Presentation Layer → Facade:** All user requests go through `HBnBFacade`.
2. **Facade → Models:** Facade calls the relevant models to process business logic.
3. **Models → Persistence Layer:** Models or facade request data access objects to retrieve/store data.
4. **Persistence → Business Logic → Presentation:** Response flows back up through the layers.

---

5. Package Diagram



HBnB Business Logic Layer – Detailed Class Diagram

Objective

Design a **detailed class diagram** for the **Business Logic Layer** of HBnB, showing:

- Entities (`User`, `Place`, `Review`, `Amenity`)
- Attributes and methods
- Relationships (associations, compositions, multiplicity)
- Key identifiers (`UUID`), creation, and update timestamps

---

1. Key Entities and Attributes

1.1 User

- **Attributes**
  - `id: UUID` – Unique identifier
  - `name: str`
  - `email: str`
  - `password: str`
  - `created_at: datetime`
  - `updated_at: datetime`
- **Methods**
  - `create_place()`
  - `write_review()`
  - `update_profile()`

1.2 Place

- **Attributes**
  - `id: UUID` – Unique identifier
  - `name: str`
  - `description: str`
  - `city: str`
  - `user_id: UUID` – Owner (association to User)
  - `created_at: datetime`
  - `updated_at: datetime`
- **Methods**
  - `add_amenity()`
  - `add_review()`
  - `calculate_rating()`

1.3 Review

- **Attributes**
  - `id: UUID`
  - `user_id: UUID` – Reviewer
  - `place_id: UUID` – Reviewed place
  - `text: str`
  - `rating: float`
  - `created_at: datetime`
  - `updated_at: datetime`
- **Methods**
  - `edit_review()`

1.4 Amenity

- **Attributes**
  - `id: UUID`
  - `name: str`
  - `description: str`
  - `created_at: datetime`
  - `updated_at: datetime`
- **Methods**
  - `update_details()`

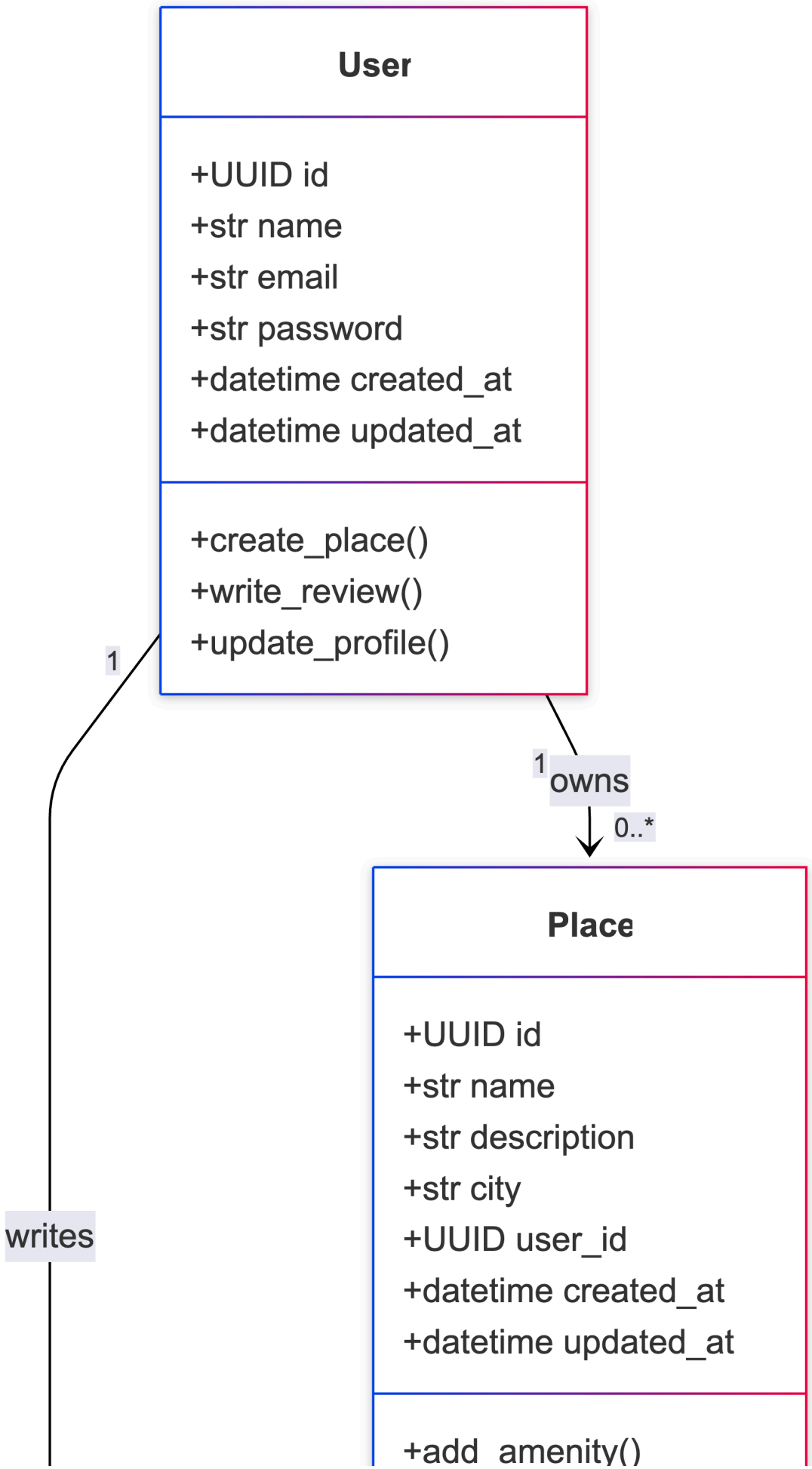
---

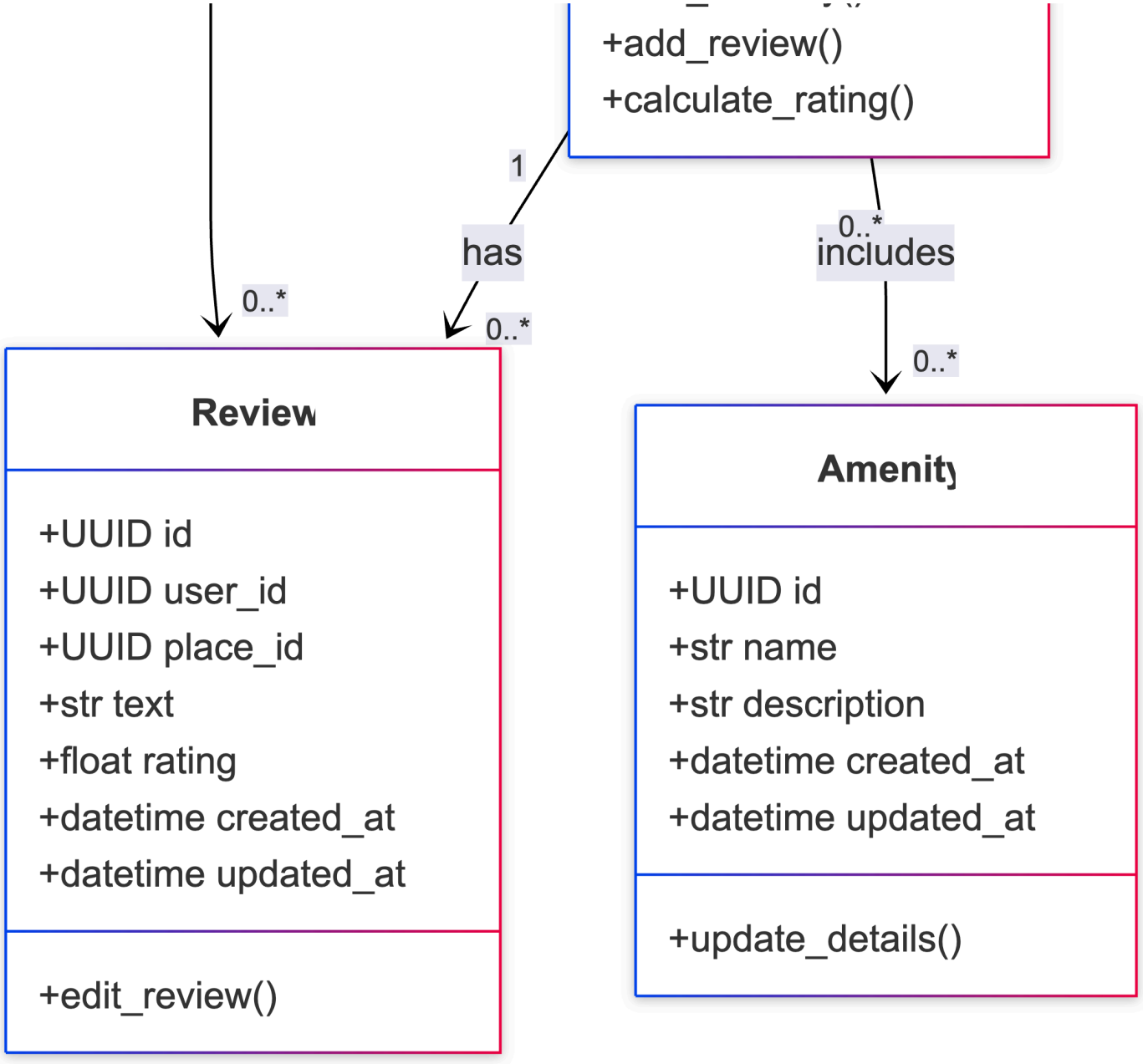
2. Relationships

- **User ↔ Place**
  - One-to-Many: A user can own multiple places (`1..*`)
- **User ↔ Review**
  - One-to-Many: A user can write multiple reviews (`1..*`)
- **Place ↔ Review**
  - One-to-Many: A place can have multiple reviews (`1..*`)
- **Place ↔ Amenity**
  - Many-to-Many: A place can have multiple amenities, and an amenity can belong to multiple places
- All entities have **timestamps** (`created_at`, `updated_at`)
- All entities use **UUIDs** as identifiers

---

3. Business Logic Layer Diagram


User
+UUID id
+str name
+str email
+str password
+datetime created_at
+datetime updated_at
+create_place()
+write_review()
+update_profile()
1
writes
1
owns
0..*
Place
+UUID id
+str name
+str description
+str city
+UUID user_id
+datetime created_at
+datetime updated_at
+add_amenity()



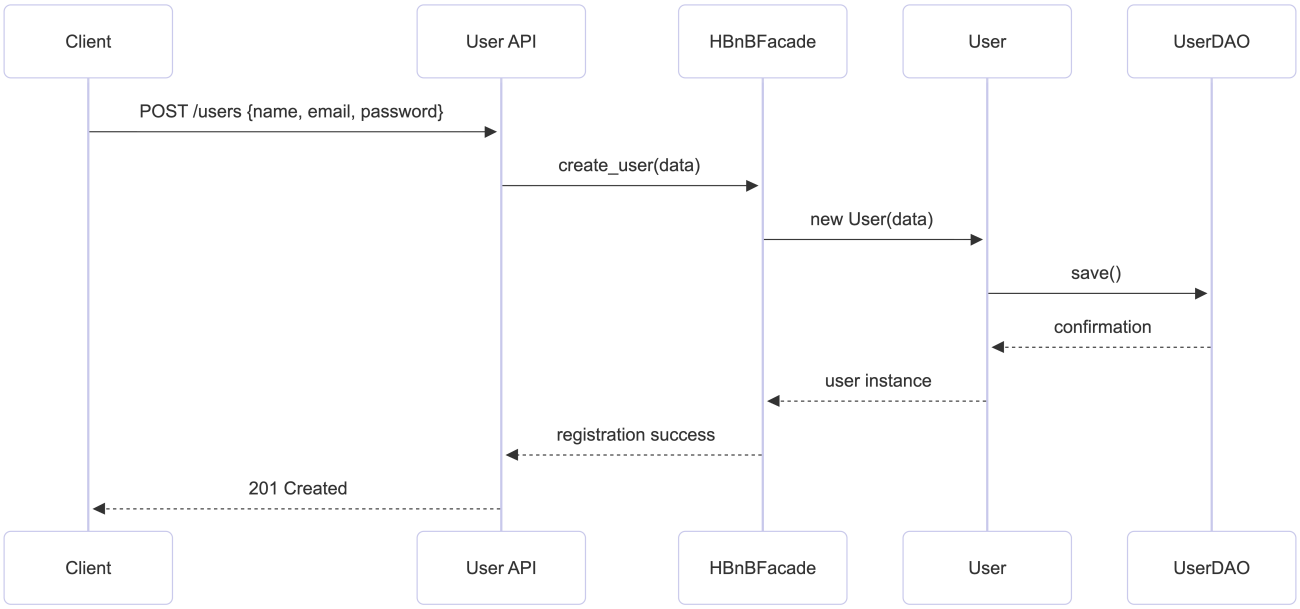
HBnB Application – Sequence Diagrams for API Calls

Objective

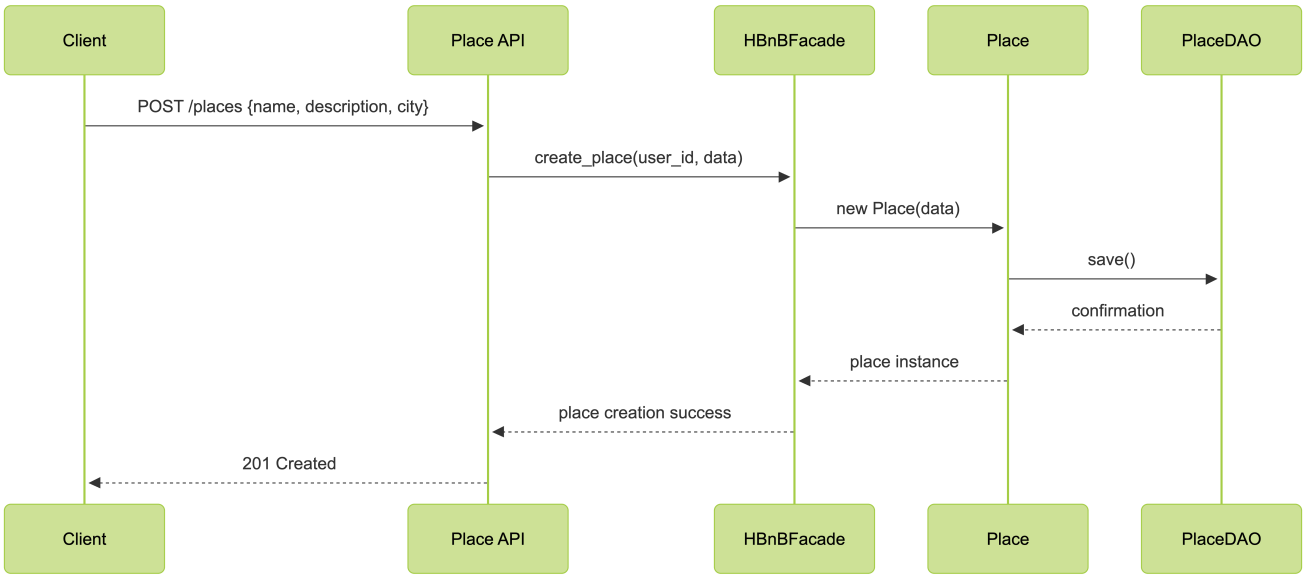
This document provides sequence diagrams for four API calls in the HBnB application. Each diagram illustrates:

- Interaction between layers (Presentation, Business Logic, Persistence)
- Flow of data and method calls
- Step-by-step processing of user requests

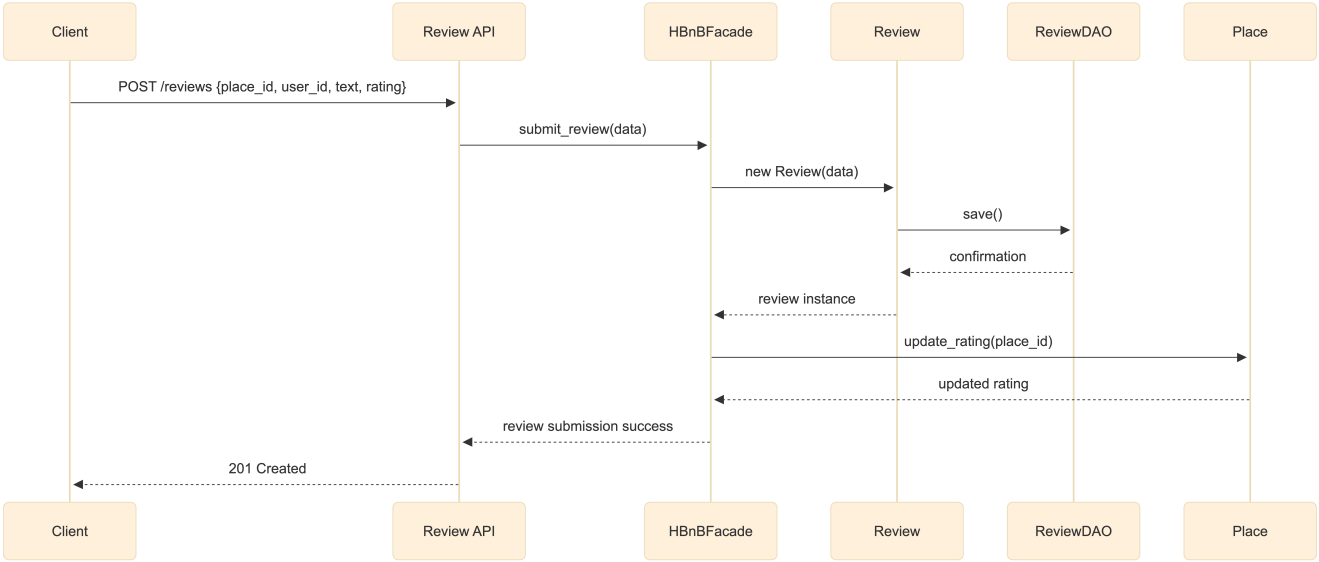
1. User Registration



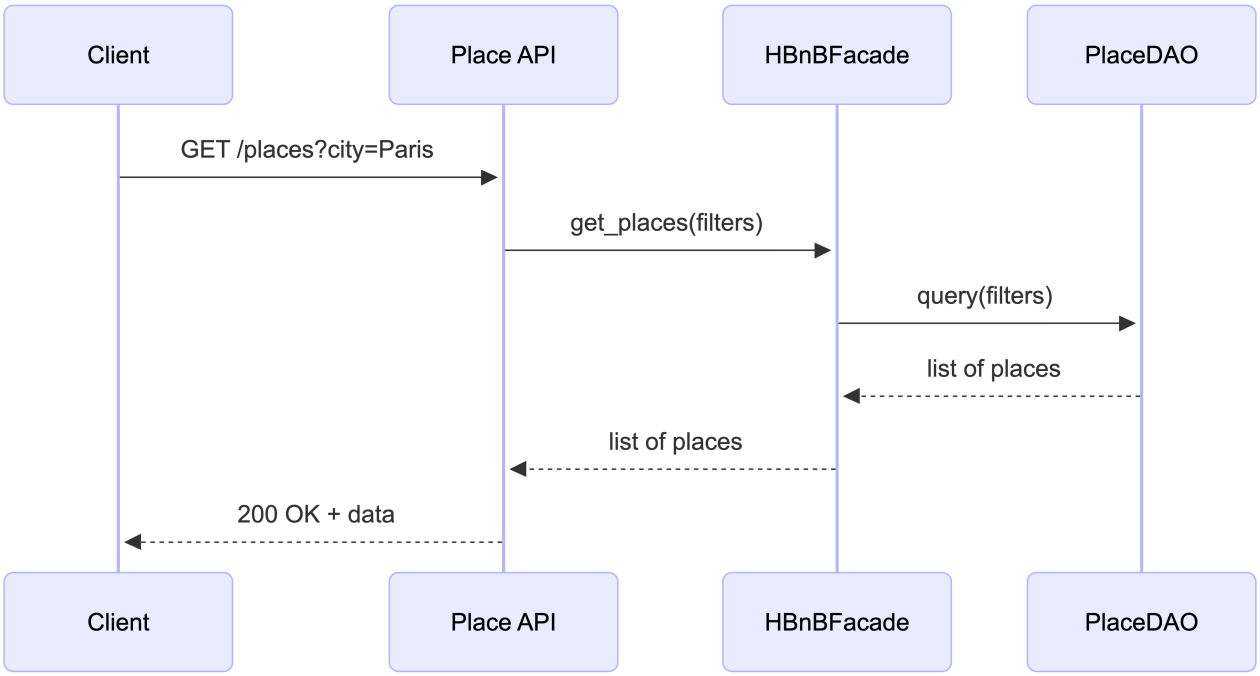
2. Place Creation



3. Review Submission



4. Fetching a List of Places



Conclusion

The HBnB project demonstrates a well-structured and modular approach to building a scalable web application. By organizing the system into **three distinct layers**—Presentation, Business Logic, and Persistence—the architecture ensures a clear separation of concerns, maintainability, and ease of future extension.

Key takeaways from this project include:

- **Layered Architecture:** Each layer has a clear responsibility, reducing coupling and promoting clean, testable code.
- **Facade Pattern:** The use of a unified interface simplifies interactions between layers and enhances modularity.
- **Comprehensive Business Logic:** Core entities such as `User`, `Place`, `Review`, and `Amenity` are clearly modeled with appropriate relationships, ensuring accurate representation of real-world use cases.
- **API Interaction Flow:** Sequence diagrams illustrate the step-by-step processing of requests, from user input to database operations, highlighting the smooth coordination between layers.
- **Documentation and Design Clarity:** Detailed diagrams and explanatory notes provide a robust blueprint that supports implementation, debugging, and future maintenance.

Overall, this project establishes a **solid foundation for HBnB**, offering both a clear conceptual understanding and practical guidance for implementation. The architecture and design choices made in this project ensure that HBnB can scale, adapt to new features, and provide a reliable and maintainable system for users and developers alike.