Documentation des Diagrammes UML

HBnB Project – Reservation Application

Document Type: Technical Documentation

Content: Analysis of UML Diagrams

Version: 1.0

Date: 2026

Table of contents

1. Introduction

This document presents a detailed analysis of the UML diagrams for the Holberton School HBnB project, a lodging reservation application. UML (Unified Modeling Language) diagrams allow for visualizing, specifying, and documenting the system's architecture and behavior.

1.1 Documentation Objectives

- Understand the structure of classes and their relationships
- Visualize the application's layered architecture
- Analyze interactions between system components
- Document the main business process flows

1.2 Types of Diagrams Presented

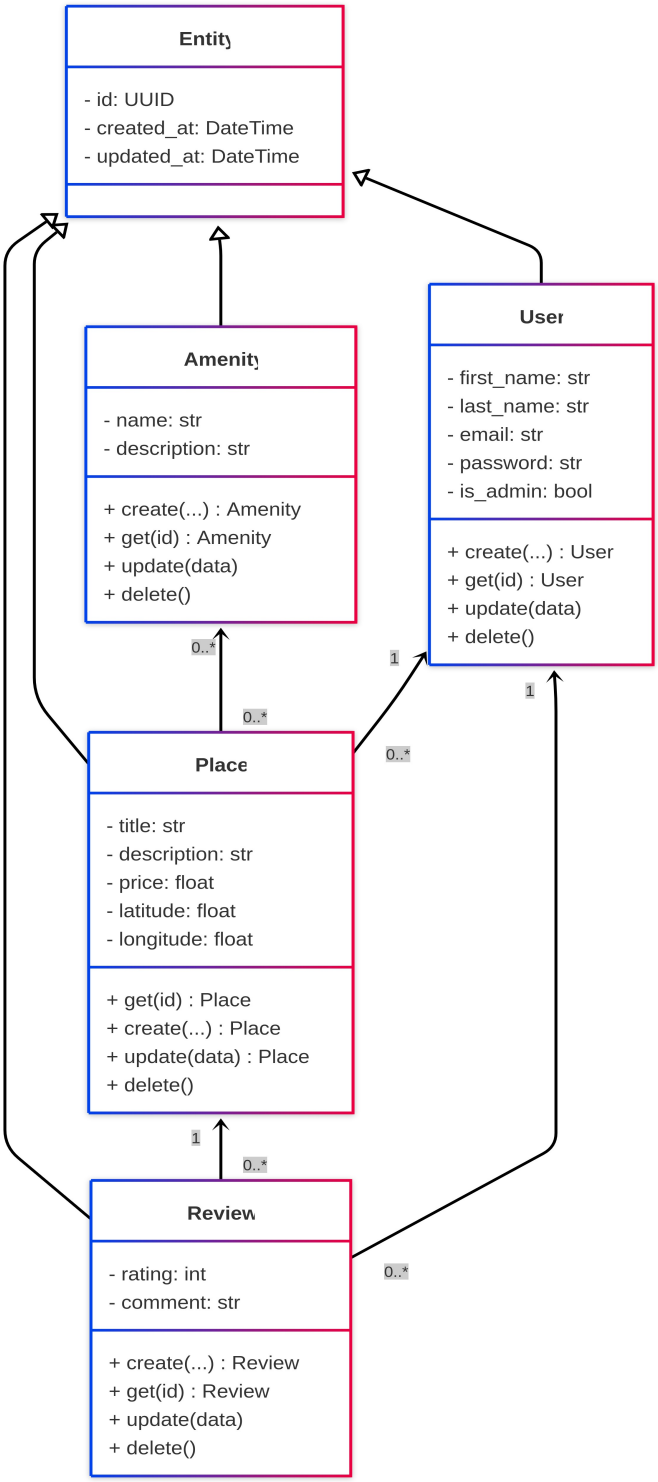
- **Class Diagram:** Represents the static structure of the system, including classes, their attributes, methods, and relationships.

- **Package Diagram:** Illustrates the modular organization of the application in architectural layers.

- **Sequence Diagrams:** Show the temporal interactions between objects for different use case scenarios.

2. Class Diagram

The class diagram represents the fundamental structure of the HBnB system. It defines the main business entities and their relationships, forming the application's domain model.



2.1 Main Classes

User

Represents a system user with their login and profile information.

Main attributes :

- **first_name, last_name** : User's full name
- **email** : Unique email adress
- **password** : Secure password

Main methods :

- **create()** : create a new user
- **get(id)** : retrieve a user by ID
- **update(data)** : Update user information
- **delete()** : Delete the account

Place

Represents a property available for rent

Main Attributes :

- **name** : Name of the place
- **description** : Detailed description
- **address** : Full adress
- **city, country** : Location
- **latitude, longitude** : GPS coordinates
- **number_of_rooms, bathrooms ….** : Features
- **price** : Daily rate per night

Main Methods :

- **create()** : Register a place
- **get(id)** : Retrieve place details
- **update(data)** : Modify information
- **delete()** : Remove the place

Review

Allows users to leave comments and ratings on places

Main Attributes :

- **rating** : Ration from 1 to 5 stars
- **comment** : Textual comment

Main Methods :

- **create()** : Publish a new review
- **get(id**) : View a review
- **update(data)** : Edit the review
- **delete()** : Delete the review

Amenity

Represents the amenities and services available at a place (WiFi, parking, swimming pool, etc.).

Main attributes :

- **name** : Name of the amenity
- **description** : Description of the amenity

Main Methods :

- **create()** : Add a new amenity
- **get(id)** : Retrieve details
- **update(data)** : Modify information
- **delete()** : Remove the amenity

2.2 Relationships Between Classes

- **User – Place (Association):** A user can own multiple places. This allows managing property owners.

- **User – Review (Association):** A user can write multiple reviews for different places.

- **Place – Review (Association):** A place can receive multiple reviews from different users .

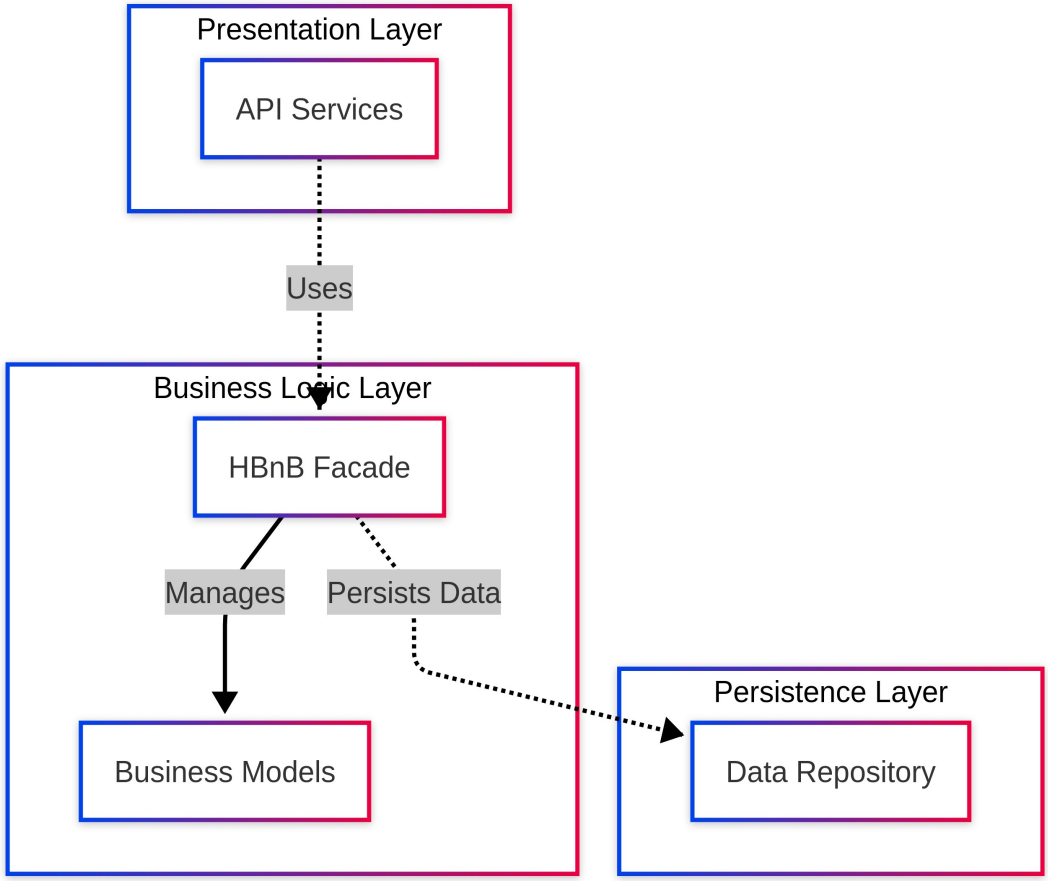
- **Place – Amenity (Association):** A place can have multiple amenities, and an amenity can be present in multiple places.

2.3 Design Principles

- **Encapsulation:** Attributes are private (-) and accessible via public methods (+)
- **Cohesion:** Each class has a single, well-defined responsibility
- **Separation of Concerns:** Business data is separated from application logic
- **Full CRUD:** All entities support Create, Read, Update, Delete operations

3. Package Diagram

The package diagram illustrates the layered architecture of the HBnB application. This modular organization promotes maintainability, testability, and adherence to the separation of concerns principle.



3.1 Three-Layer Architecture

Presentation Layer

Component : API Services

Entry point of the application exposing REST endpoints. This layer handles HTTP requests, input data validation, JSON serialization/deserialization, and HTTP responses. It serves as the interface between the client and the business logic.

Business Logic Layer

Component : HbnB Facade

Core of the application containing the business logic. The Facade acts as a unified entry point to orchestrate complex operations, coordinate interactions between models and the repository, and enforce business rules. It implements the Facade pattern to simplify the interface with subsystems.

Persistence Layer

Component : Business Models + Data Repository

Handles data persistence. The Business Models define the structure of business entities, while the Data Repository abstracts data access by implementing the Repository pattern. This separation allows for easily switching the storage system without affecting the upper layers.

3.2 Dependencies and Data Flows

• **API --> Facade (Uses):** The presentation layer uses the facade to access business functionalities. Unidirectional communication respecting the principle of downward dependency.

• **Facade --> Models (Manages):** The facade manipulates and coordinates business models to implement use cases. It orchestrates operations on entities.

• **Facade --> Repository (Persists Data):** The facade delegates data persistence to the repository. The dashed line indicates an indirect dependency via an interface, allowing dependency injection.

3.3 Advantages of this Architecture

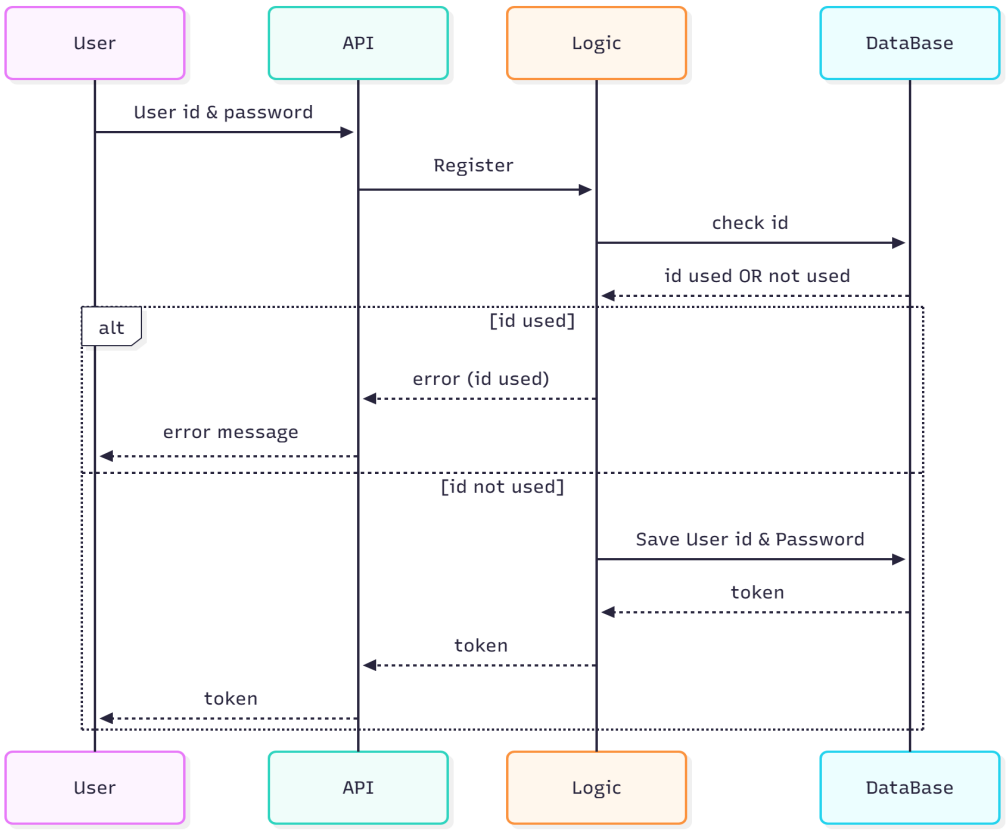
- **Separation of Responsibilities:** Each layer has a clearly defined role
- **Testability:** Layers can be tested independently using mocks
- **Maintainability:** Changes are isolated within the concerned layer
- **Scalability:** Facilitates adding new functionalities
- **Reusability:** Business logic is decoupled from the interface
- **Flexibility:** Technology changes in one layer do not affect others

4. Sequence Diagram

Sequence diagrams illustrate the temporal interactions between the different system components for specific use case scenarios. They show the order of exchanged messages and help understand the dynamic behavior of the application.

4.1 User Registration

This scenario describes the complete process of registering a new user in the system.



Execution Flow :

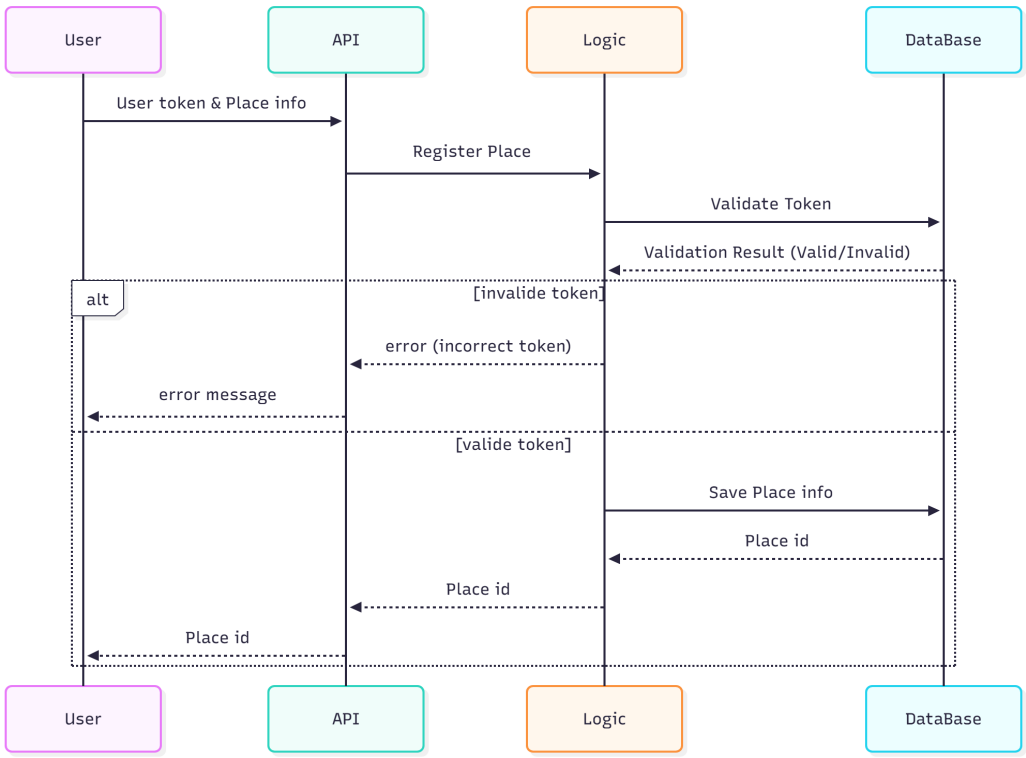
- The client sends a POST request with user data (name, email, password)
- The API Service receives the request and calls the `create_user()` method of the Facade
- The Facade performs business validations (email format, password strength, uniqueness)
- If validations pass, the Facade uses the User model to create the instance
- The User model hashes the password for security
- The Facade persists the user via the Repository
- The Repository stores the data in the database
- A success response with the user ID is returned to the client

Key Points :

- **Cascading Validation:** API → Facade → Model
- **Security:** Password hashing before storage
- **Email Uniqueness Check**
- **Error Handling** at each layer

4.2 Place Registration

This sceanario illustrates the addition of a new rental property by an owner



execution Flow :

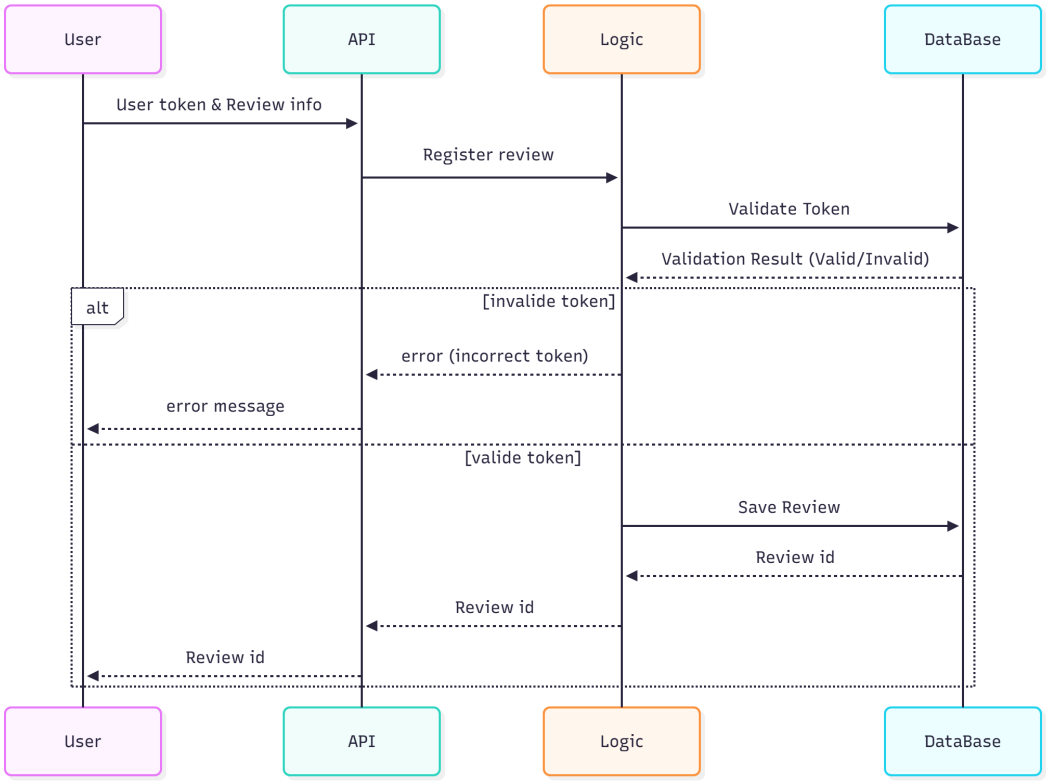
- The authenticated client sends a POST request with the property details
- The API Service checks the owner's authentication and permissions
- The Facade receives the place creation request
- Data validation: name, address, price, capacity, GPS coordinates
- Verification that the user exists and is allowed to create a place
- Creation of the Place instance with all attributes
- Association of the place with the owner (User–Place relationship)
- Persistence via the Repository
- Return of the created place ID to the client

Performed Validations :

- Owner authentication
- Data consistency (positive price)
- Validity of address and GPS coordinates
- Potential uniqueness of the place to avoid duplicates

4.3 Review Registration

This scenario shows how a user can leave a review for a place they have visited.



Execution Flow :

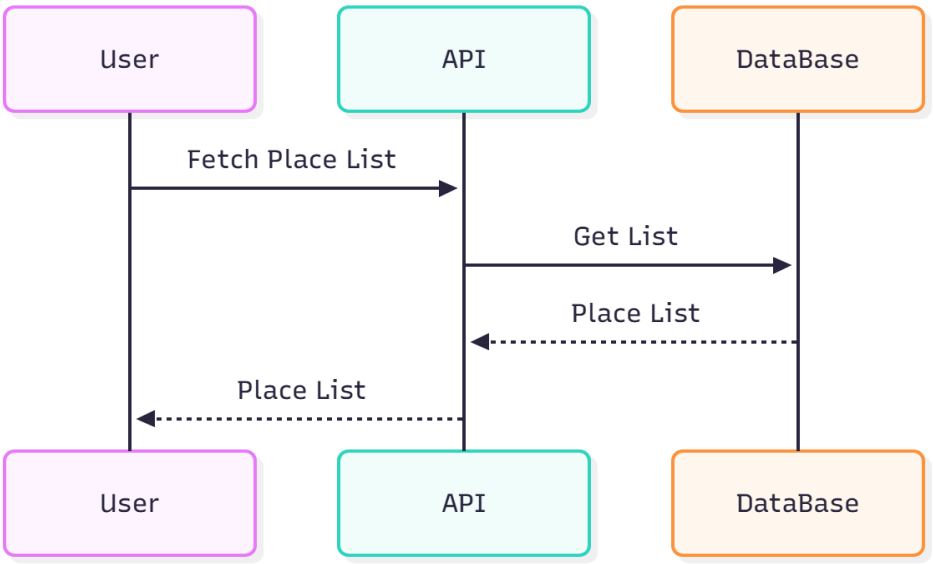
- The client sends a POST request with the place ID, rating, and comment
- The API Service authenticates the user
- The Facade checks that the place exists
- Validation that the user can leave a review
- Rating validation (between 1 and 5)
- Creation of the Review instance with User–Place association
- Return of confirmation to the client

Business Rules :

- A user cannot review their own place
- The rating must be between 1 and 5 stars
- The comment is optional but recommended
- A user can leave only one review per place
- The review is timestamped for traceability

4.4 Place Retrieval

This scenario illustrates viewing the details of a place, including its amenities and reviews.



Execution Flow :

- The client sends a GET request to get all places
- The API Service forwards the request to the Facade
- The Repository loads the places from the database
- Load relationships: owner (User)
- Retrieve the list of associated amenities
- Aggregate all data
- Serialize to JSON and send to the client

Returned Data :

- Place information: name, description, address, price, photos
- Owner details: name, profile picture
- List of available amenities
- Calculated average rating
- Availability (if integrated with the reservation system)

5. Conclusion

This documentation has presented a comprehensive analysis of the UML diagrams for the HBnB project. The proposed architecture demonstrates a robust design based on proven software engineering principles.

5.1 Design Strenghts

- **Clear Layered Architecture:** The Presentation / Business / Persistence separation facilitates maintenance and evolution
- **Coherent Domain Model:** Business classes are well-defined with precise responsibilities
- **Documented Interactions:** Sequence diagrams clarify processing flows
- **Extensibility:** The architecture allows for easy addition of new features
- **Testability:** Layer separation simplifies unit and integration testing

This architecture provides a solid foundation for developing a modern, scalable, and maintainable reservation application. UML diagrams serve as a reference throughout development and facilitate communication among team members.