HBnB - UML Technical Documentation

About this Document

This document provides complete technical documentation for the HBnB (Holberton Airbnb Clone) project. It compiles all the UML diagrams and detailed explanations necessary to understand and implement the application's architecture.

Objective of the HBnB Project

HBnB is a property rental app inspired by Airbnb. It allows users to:

- **Create and manage listings** for properties for rent
- **View available properties** with details and amenities
- **Leave reviews and ratings** for properties
- **Manage their user profile** and bookings

Scope of this Documentation

This technical documentation covers:

1.General Architecture

- Introduction to the three-layer architecture
- Explanation of the Facade pattern
- Communication flow between layers

2.Business Logic Layer

- Detailed Class Diagram
- Main Entities (User, Place, Review, Amenity)
- Relationships between Entities
- Responsibilities of each class

3. API Interaction Flow

- Sequence diagrams for API calls
- Complete data flow
- Interactions between components
- Error handling and validations

Document Structure

1. Introduction (this document)
2. Task 0: High-Level Package Diagram

- Three-Layer Architecture
- Facade Pattern
- Main Components

3. Task 1: Detailed Class Diagram

- Business Entities (User, Place, Review, Amenity)
- Relationships and Associations
- Attributes and Methods

4. Task 2: Sequence Diagrams

- API Flow for User Creation
- API Flow for Property Creation
- API Flow for Review Creation
- Fetching a List of Places

Design Principles

Separation of Responsibilities

Each layer has a well-defined role and communicates only with its neighbors, ensuring easy maintenance and evolution.

Pattern Facade

The Facade acts as a unified interface between the Presentation layer and the Business layer, simplifying interactions and reducing coupling.

Modularity

The architecture allows for the easy addition of new features without affecting existing components.

Testability

Each layer can be tested independently, making it easier to detect and correct bugs.

Target Audience

This documentation is intended for:

- **Developers:** To understand the architecture and implement the features
- **System Architects:** To validate design choices
- **Testers:** To design appropriate tests
- **Maintainers:** To understand and maintain the code

How to Use This Document

1. Start with this introduction to understand the overall context.
2. Consult the package diagram for an overview of the architecture.
3. Study the class diagram to understand the entities and their relationships.
4. Analyze the sequence diagrams to understand the interaction flows.
5. Refer to the detailed explanations for each section.

Conventions and Notations

UML notation

- Classes: Represented by rectangles with attributes and methods
- Relationships: Arrows indicating inheritance, association, or dependency
- Multiplicities: Indicating the number of instances in a relationship

Naming

- Classes: PascalCase (ex: UserRepository)
- Methods: camelCase (ex: getUserById)
- Attributes: snake_case (ex: user_id)

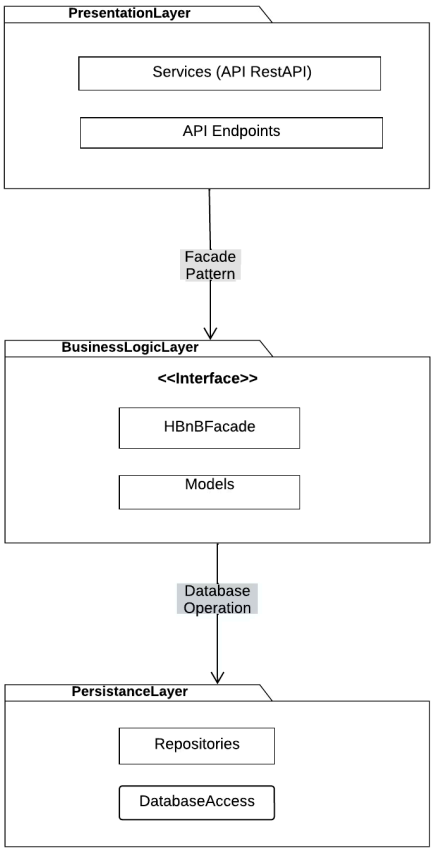
Important Notes

Document Evolution: This document may be updated as the project evolves.
Validation: All diagrams have been validated to ensure accuracy and clarity.
Resources: References and learning resources are provided to deepen understanding of each concept.

Task 0: High-Level Packet Diagram

Three-Layer Architecture

This diagram illustrates the overall architecture of the HBnB project, organized into three distinct layers: Presentation, Business Logic, and Persistence. The Facade pattern is at the heart of this architecture, acting as the unified communication interface between the Presentation layer and the Business Logic layer.



Presentation Layer

- API Endpoints: These are the system's entry points, accessible via HTTP. They receive requests from users or other systems.
- Services: These contain the logic to orchestrate calls between the APIs and the business logic layer. They translate user intent into technical actions.

This layer manages interaction with the outside world: it receives requests, prepares them, and forwards them to the business logic.

Business Logic Layer

- HBNB Facade «Interface»: An interface that centralizes access to the system's main functionalities. It simplifies the use of internal components.
- Models (User, Review, Amenity): Representations of business entities. Each model contains the associated data and behaviors.

This layer contains the system's intelligence: it decides what to do and how, according to business rules.

Pattern Facade

The Facade acts as the communication interface between the Presentation and Business Logic layers.

Data flow

The arrows indicate that data flows from top to bottom:

• The user interacts with the API (PresentationLayer).

• The API calls the business logic (BusinessLogicLayer).

• The business logic uses persistence to access the data (PersistenceLayer).

Key Components

Details of Each Layer

Presentation Layer

- Service API: The primary entry point for services, handles HTTP requests.
- Web API: The web interface for clients, manages sessions, authentication, and authorization.
- Rest API: A complete REST architecture with endpoints for each resource (Users, Places, Reviews, Amenities).

Business Layer - Main Models

User

Attributes: first_name, last_name, email, password,

Methods: registrer(), update(), delete(), validate()

Place

Attributes: title, description, price, rental Methods: create(), update(), delete(), get_reviews()

Review

Attributes: comment, rating

Methods: create(), update(), delete()

Amenity

Attributs: name, description

Méthodes: create(), update(), delete()

Persistence Layer - Repositories

→ UserRepository

- CRUD operations for users
- Search by email

→ PlaceRepository

- CRUD operations for properties
- Location filtering

→ ReviewRepository

- CRUD operations for reviews
- Retrieving reviews by property

→ AmenityRepository

CRUD operations for equipment

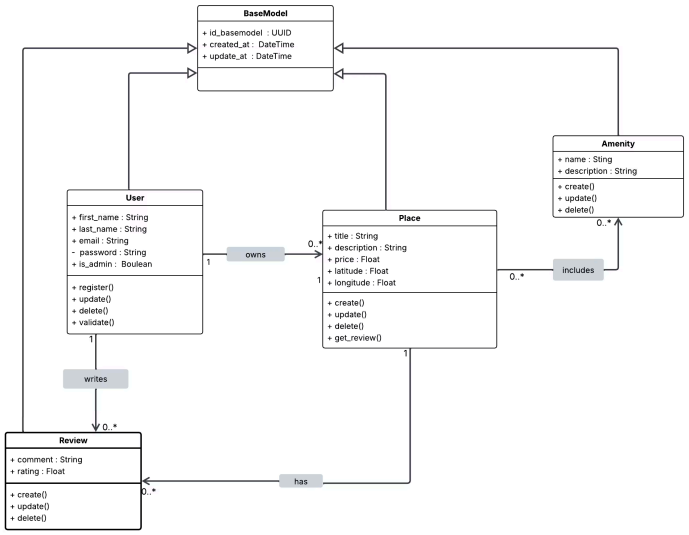
Property association

Task 1: Detailed Class Diagram

Business Logic

This diagram presents the detailed structure of the classes within the business layer. It highlights the main entities such as User, Place, Review, and Amenity, as well as the relationships and interactions between them.

The goal is to provide a clear view of the data organization and the core logic of the application.



Main Entities

- **User** represents a user with attributes such as first_name, last_name, email, password and is_admin.
- **Place** describes a property, including title, description, price, latitude and longitude.
- **Review** records a review of a property, including comment and rating.
- **Amenity** represents an available amenity or service, including name, and description.

Relationships and Associations

- **User and Place:** A user can own multiple properties (User 1:N Place).
- **Place and Review:** A property can receive multiple reviews (Place 1:N Review).
- **User and Review:** A user can post multiple reviews (User 1:N Review).
- **Place and Amenity:** A property can be associated with multiple amenities (Place M:N Amenity - relationship via an intermediate table not shown).

Class Responsibilities

- **User:** Manage user authentication, profile information, and activity history.
- **Place:** Manage property details, location searches, and booking management.
- **Review:** Allow users to create, edit, and delete reviews, and calculate average ratings.
- **Amenity:** Manage the amenities catalog and their associations with properties..

Task 2: Sequence Diagrams

API Interaction Flow

This section is dedicated to creating sequence diagrams, an essential tool for visualizing the communication flow and interactions between the different components of our application. They will allow us to understand how API requests are processed, from their reception by the presentation layer to data persistence and the return of the response to the client.

Sequence diagrams illustrate the chronological flow of messages exchanged between objects or systems.

We will cover the four main creation flows via the API:

- Creating a user
- Creating a property (Place)
- Creating a review
- Retrieving a list of properties

API Flow for User Creation

This flow describes the detailed interactions that occur when a new customer signs up via the API. It highlights the path of a user creation request through the application layers, including validation, business logic, and data persistence.

Here is the step-by-step process:

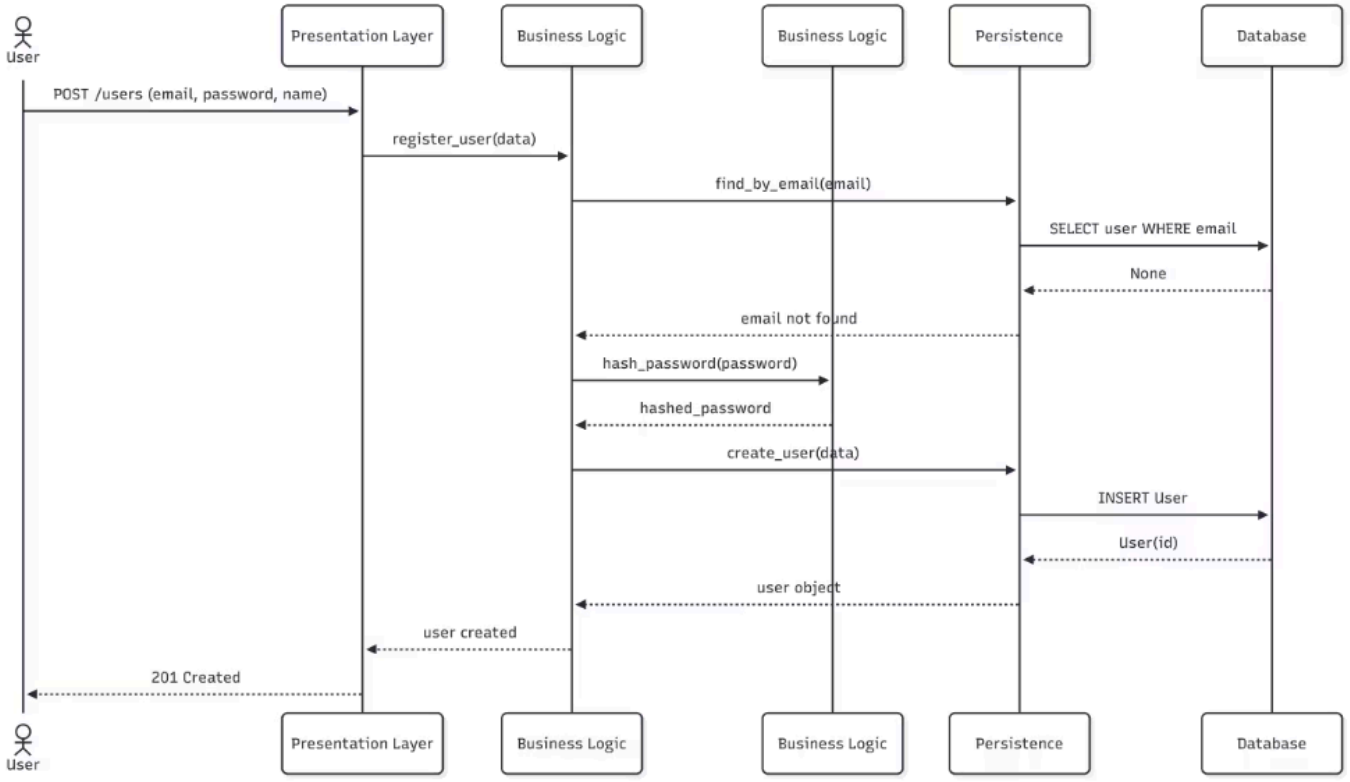
1. **Client Request**: The client sends an HTTP POST request to the */users* endpoint with the user's data (email, password, first name, last name) in the request body.
2. **Reception by the Presentation Layer (RestAPI)**: The *UserController* of the presentation layer (RestAPI) intercepts the incoming request.
3. **Initial Validation:** The RestAPI performs basic syntax and semantic validation of the data (e.g., email format, presence of required fields).
4. **Call to the Business Layer (UserFacade)**: If the initial validation is successful, the RestAPI delegates the request to the *UserFacade* (a component of the business layer).
5. **Business Validation:** The *UserFacade* applies the business logic specific to user creation, including verifying the uniqueness of the email address in the database. It can also hash the password.
6. **Business Object Creation**: If all business validations are successful, the *UserFacade* creates an instance of the User business object.
7. **Call to the Persistence Layer (UserRepository):** The *UserFacade* invokes the *UserRepository*'s *save()* method to persist the new *User* object.
8. **Database Interaction:** The *UserRepository* interacts with the database management system (DBMS) to insert the user's data.
9. **Return from Persistence:** The database returns a success or failure status to the *UserRepository*, which then forwards it to the *UserFacade.*
10. **Business Success/Failure Handling:** The *UserFacade* handles the response. In case of success, it can generate an authentication token or other relevant information. In case of failure (for example, email address already in use), it throws an appropriate business exception.
11. **Return to the Presentation Layer:** The *UserFacade* returns the result (*User* object created or error) to the RestAPI.

12 . **Constructing the HTTP Response:** The RestAPI formats the HTTP response. If successful, it returns a 201 Created status with the User object (potentially without the password) in the response body. If an error occurs, it returns an appropriate status (*400 Bad Request, 409 Conflict*, etc.) with a clear error message.

13. **Sending the Response to the Client**: The RestAPI sends the finalized HTTP response to the client.

Sequence Diagram: User Creation

This diagram illustrates the detailed flow of interactions during user creation, from the initial client request to data persistence in the database, and the return of the response.



This sequence diagram breaks down the user creation process, highlighting the critical steps from the client's initial request to the secure persistence of data. It illustrates the collaboration between the presentation layer, business logic, and the database system. Each phase plays a vital role in ensuring the integrity and security of user information while providing a clear response to the client.

Request Validation

- Client POST request to the */users* endpoint.
- RestAPI performs an initial data validation (format, required fields)

Data Persistence

- The *UserRepository* saves the new user.
- Interaction with the database for insertion

Business Logic

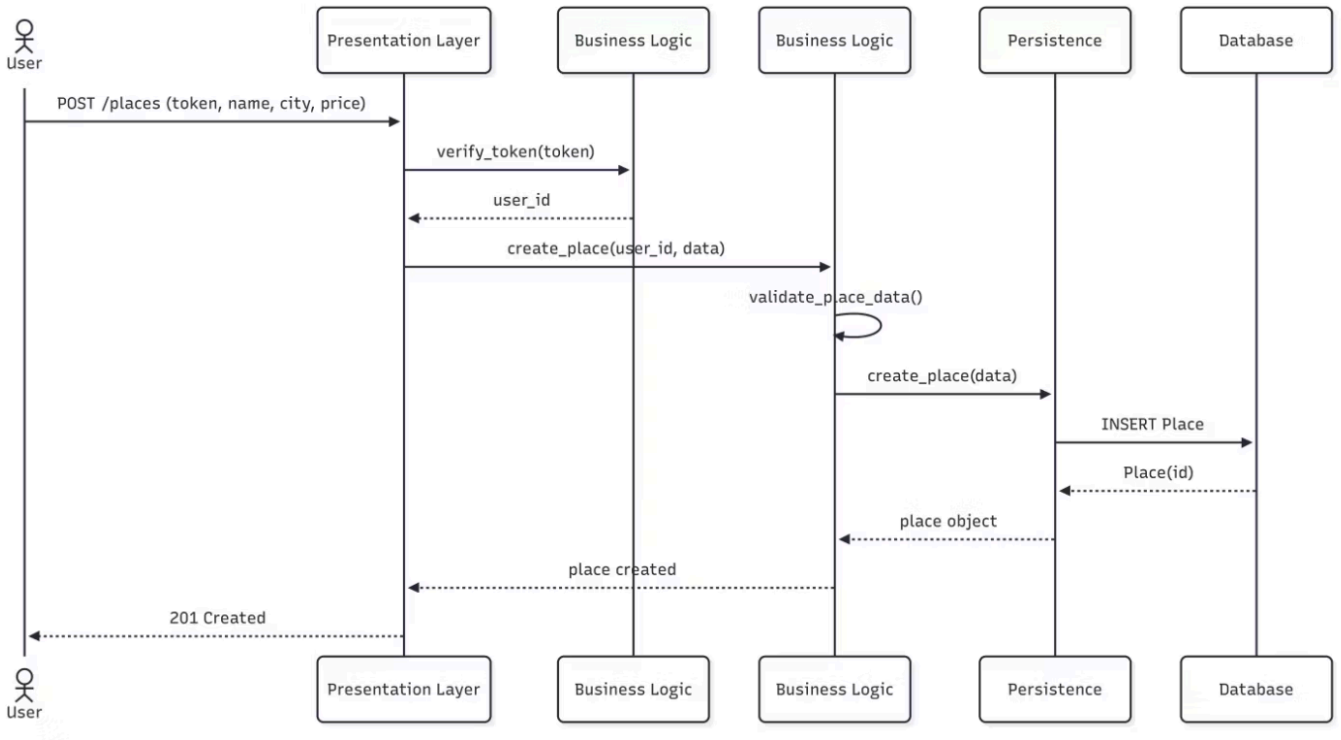
- The *UserFacade* applies the business rules. Checks the uniqueness of the email and hashes the password.
- Creates the *User* object.

Response HTTP

- The RestAPI constructs and sends an HTTP
- Returns 201 Created (with user data) or a 4xx error.

Sequence Diagram: Property Creation

This diagram illustrates the detailed flow of interactions and software responsibilities when creating a new property in the system, from receiving the client request to data persistence and creation confirmation.



This property creation process involves several critical steps, ranging from the secure receipt of client data to their persistence and validation. It ensures that each new property is correctly recorded, validated by business logic, and associated with its owner before confirming the creation to the client.

Validation and Verification

The client initiates the request, which is validated by the RestAPI for syntax and data presence, followed by a check for existence and owner authorization.

Business Logic

The *Facade* orchestrates the creation of the *Place* object, applying business rules and validating property-specific data (e.g., positive price).

Persistence and Associations

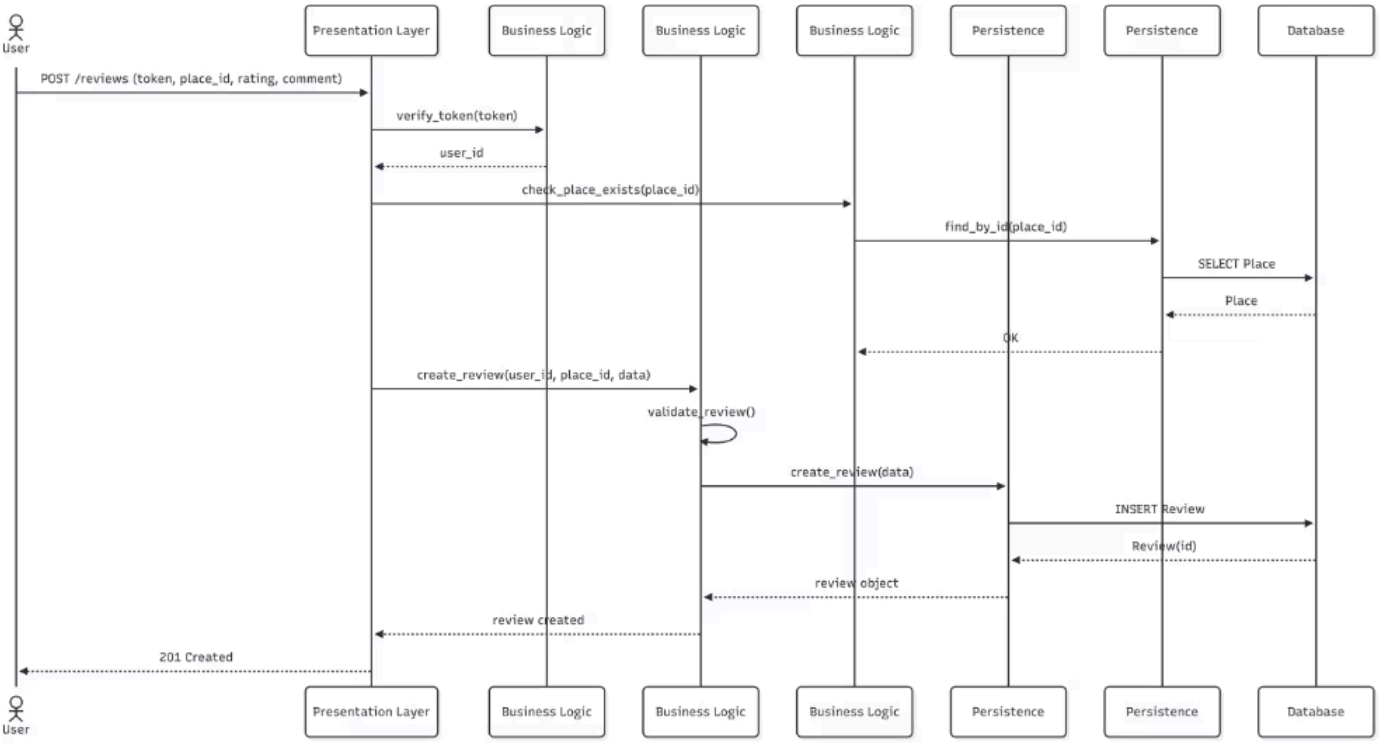
The *PlaceRepository* saves the *Place* object in the database, retrieves its unique ID, and the *Facade* manages the association of amenities.

Response to Client

The RestAPI constructs and sends an HTTP *201 Created* response to the client, including the fully created and validated *Place* object.

Sequence Diagram: Creating Reviews

This diagram presents the detailed flow of software interactions and validations required to create a customer review. It illustrates how the customer request is processed, validated, and persisted through the various layers of the application, until the relevant data is updated.



The review creation process is crucial for maintaining information integrity and rating reliability. It begins with rigorous validation of the query and associated data, followed by the application of business logic to ensure the review's relevance and uniqueness. Finally, the review is saved, and aggregates such as the average rating are updated before the transaction is confirmed to the customer.

Validation & Verification

The client submits a review, which is first validated by the RestAPI. This step includes verifying the existence of the user and the property associated with the review.

Business Logic

The *Facade* applies the business rules, creates the *Review* object, and checks for duplicate reviews for the same user and property.

Persistence & Updates

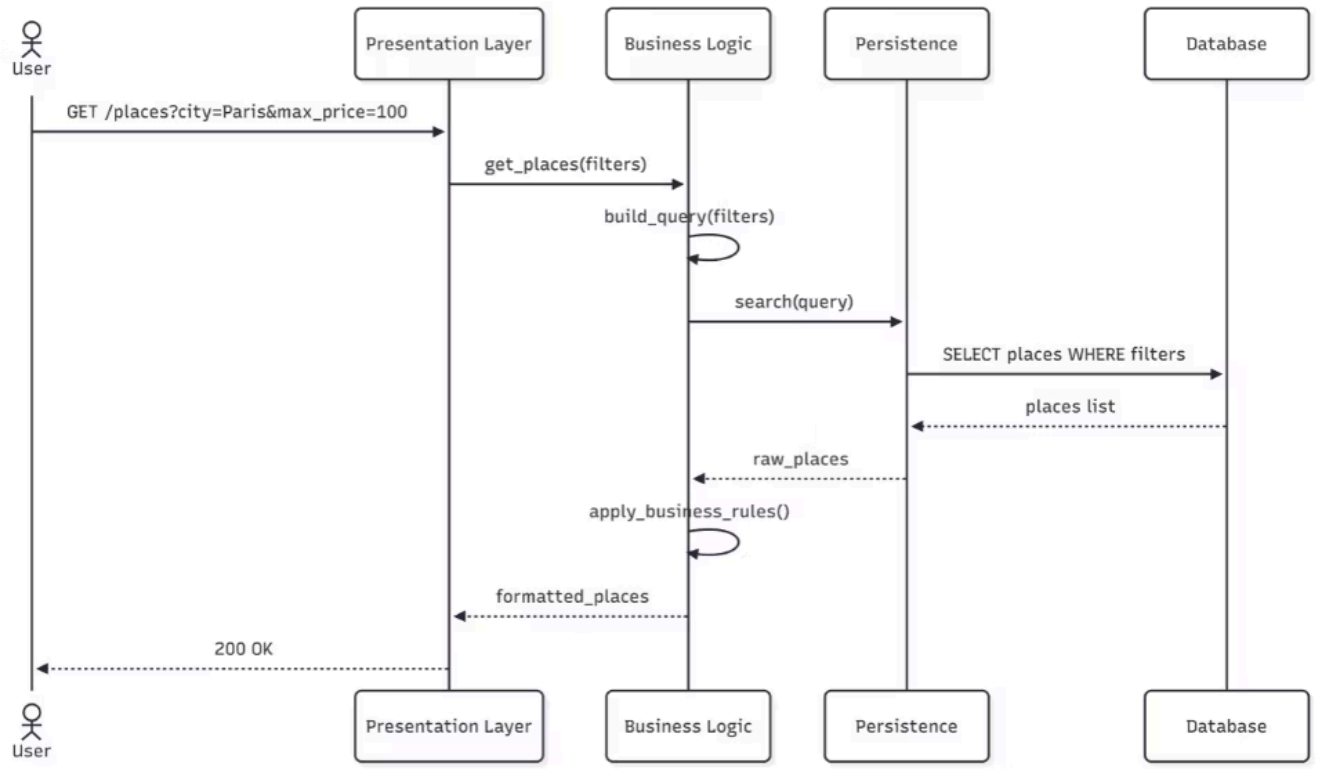
The *ReviewRepository* saves the review in the database. It also triggers the update of the average rating for the property in question.

Response to Client

The RestAPI constructs and sends an HTTP *201 Created response* to the client, confirming the successful creation of the review and providing the data for the created review.

Sequence Diagram: Retrieving a List of Properties

This sequence diagram illustrates the detailed flow of retrieving a list of properties, including the interactions between the client and the different layers of the application. It covers request handling, the application of business logic, and data retrieval, with the option to use filters and pagination.



The process of retrieving a list of properties begins with a client request that can include search and pagination criteria. The RestAPI validates these parameters before forwarding the request to the Facade, which applies business logic and enriches the data. Finally, the Repository queries the database to return the filtered and paginated results to the client, ensuring a fast and relevant user experience.

Request & Validation

The client sends a GET request with filters (location, price) and pagination.

The RestAPI validates these parameters to ensure they are correct.

Business Logic

The Facade applies business filters to refine the search.

It enriches the data, for example by calculating average ratings or adding equipment.

Data Retrieval

The PlaceRepository queries the database using specific criteria.

The database returns a list of properties matching the filters.

Response to Client

The RestAPI sends an HTTP 200 OK response.

The response contains a paginated list of properties and all enriched information.

HBnB - UML Technical Documentation

Complete Technical Documentation

Cette documentation exhaustive présente l'architecture technique, les diagrammes UML détaillés et les flux API complets du projet HBnB. Elle couvre l'ensemble des aspects fondamentaux de la conception et de l'implémentation, offrant une vue d'ensemble claire et approfondie des systèmes et de leurs interactions.

Three-layer architecture

Robust and modular design for improved maintainability and scalability.

Detailed UML diagrams

Clear visual representations of classes, sequences, and use cases.

Complete API flows

Precise descriptions of endpoints, requests, and responses for seamless integration.

Design Patterns

Applying proven solutions to resolve recurring design problems.

Created by:

James Roussel

Tommy Jouhans

Dates: 02/02/2026 at 02/13/2026

HBnB Project