13/02/2026

HBnB Project

BOUATE Mahmoud

PODEVIN Lucas

REGNIER Maxime

SOMMAIRE

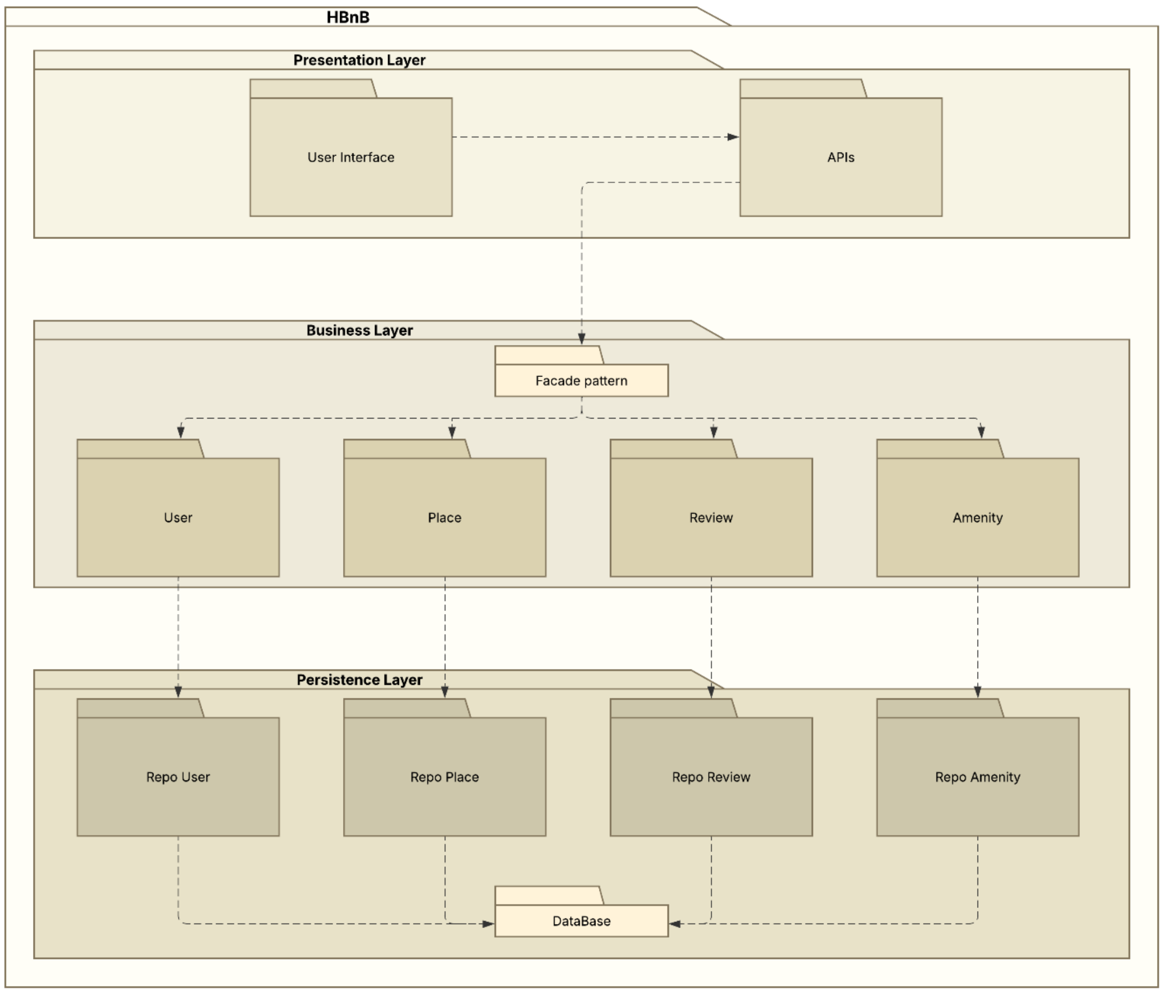
1 Introduction

This document presents the technical documentation for the HBnB Evolution project, a simplified AirBnB-like application designed to manage users, places, reviews, and amenities.

The purpose of this document is to define the overall architecture of the application and describe the structure of its business logic before the implementation phase.

This documentation provides a high-level view of the system architecture, details the core business entities, and illustrates the interactions between the different layers of the application.

2 High-Level Architecture



The HBnB Evolution application follows a layered architecture that separates responsibilities across different parts of the system. This design improves maintainability, readability, and scalability.

The Presentation Layer handles user interactions through APIs and services. It receives requests and delegates processing without containing business rules.

The Business Logic Layer contains the core entities and rules of the application. It is responsible for enforcing business constraints and coordinating operations between entities.

The Persistence Layer manages data storage and retrieval. It abstracts database operations from the rest of the application.

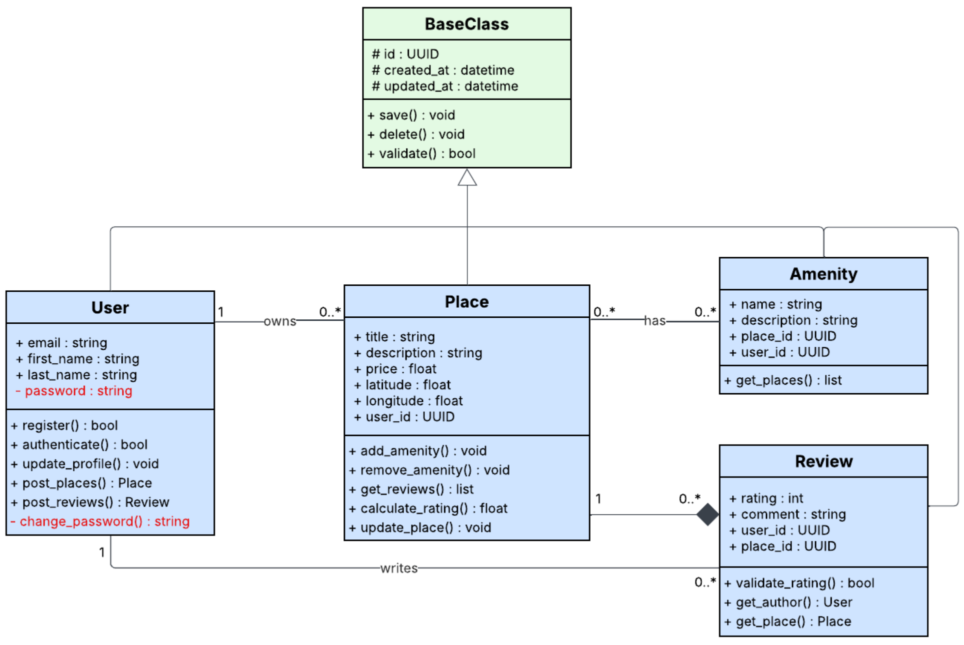
A Facade pattern is used to provide a single-entry point to the Business Logic Layer.

This approach simplifies communication between layers and reduces coupling between components.

3 Business Logic Layer (Class Diagram)

3.1 Presentation

The Business Logic Layer is the core of the HBnB Evolution application. It defines the main entities, their attributes, methods, and relationships. This layer enforces the rules of the system and manages how users, places, reviews, and amenities interact.



3.2 Explanation

3.2.1 BaseClass

A parent class inherited by all other classes.
It provides common fields and methods:

- id, created at, updated_at
- save(), delete(), to_dict(), validate()

3.2.2 User

Represents a system user.

- Attributes: email, first name, last name, password, admin flag
- Methods: register, authenticate, update profile, change password, get places and reviews

Relationship:

- One user owns many places (1 → 0..*)

3.2.3 Place

Represents a property/location listed by a user.

- Attributes: title, description, price, latitude, longitude, user_id
- Methods: manage amenities, get reviews, calculate rating

Relationships:

- Belongs to one user
- Has many amenities
- Has many reviews

3.2.4 Amenity

Represents features of a place (e.g., Wi-Fi, parking).

- Attributes: name, description, place_id, user_id
- Method: get_places()
- Relationship: many-to-many or many with places

3.2.5 Review

Represents feedback from users about a place.

- Attributes: rating, comment, user_id, place_id

- Methods: validate rating, get author, get place

Relationships:

- One review belongs to one place
- One user can write many reviews

3.3 Summary of Relationships

- User → owns → Places
- Place → has → Amenities
- Place → has → Reviews
- User → writes → Reviews
- All classes inherit from BaseClass

4 API – Interaction Flow (Sequence Diagrams)

4.1 Presentation

This section describes how the main API (application programming interface) calls interact with the different layers of the HBnB Evolution application. Sequence diagrams are used to illustrate the flow of information from the user request to the data persistence layer.

These diagrams help to understand how the Presentation Layer communicates with the Business Logic Layer through the Facade, and how data is stored or retrieved from the Persistence Layer.

4.2 Why Sequence Diagrams Are Used

Sequence diagrams show:

- The order of interactions between components
- The responsibilities of each layer
- The flow of data during an API request

They provide a clear view of how a user action is processed by the system.

4.3 User Registration

4.3.1 Description

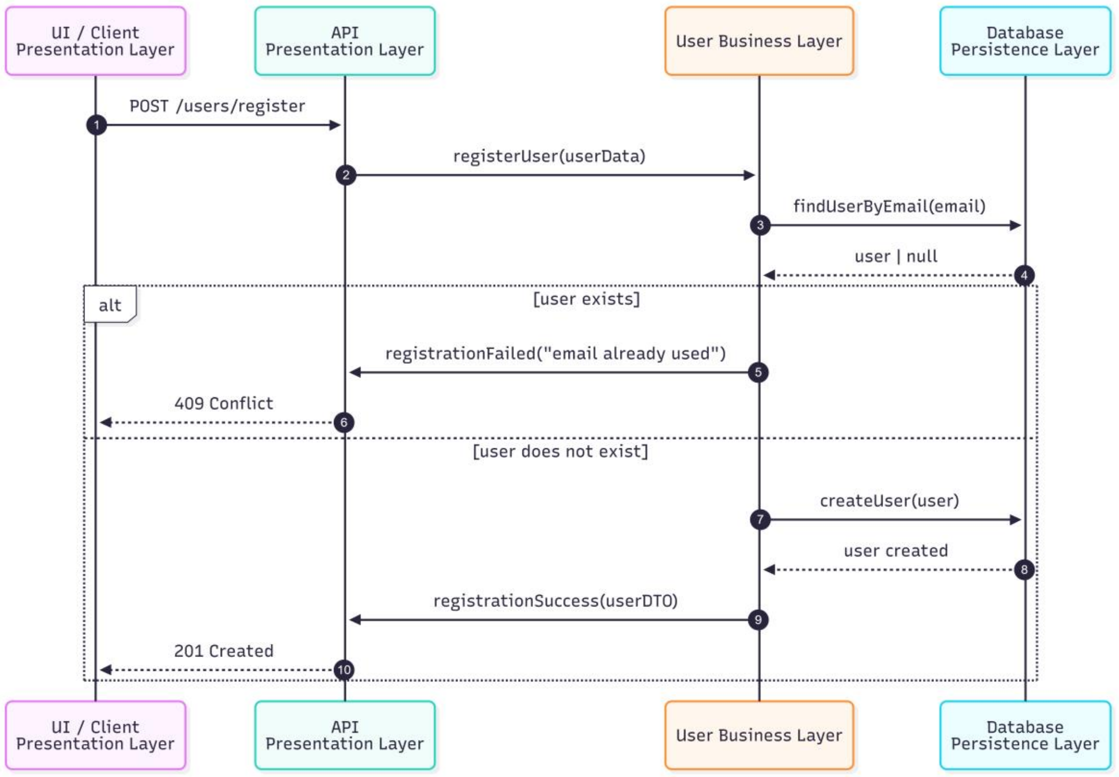
The client sends a user registration request via the API.

The API forwards the request to the User Business Layer, which validates the input data and checks whether the user already exists in the database.

The persistence layer is queried to retrieve the user information.

If the user does not exist, the Business Layer creates and stores the new user, and a success response is returned to the client.

If the user already exists, a conflict response is returned.



4.4 Place Creation

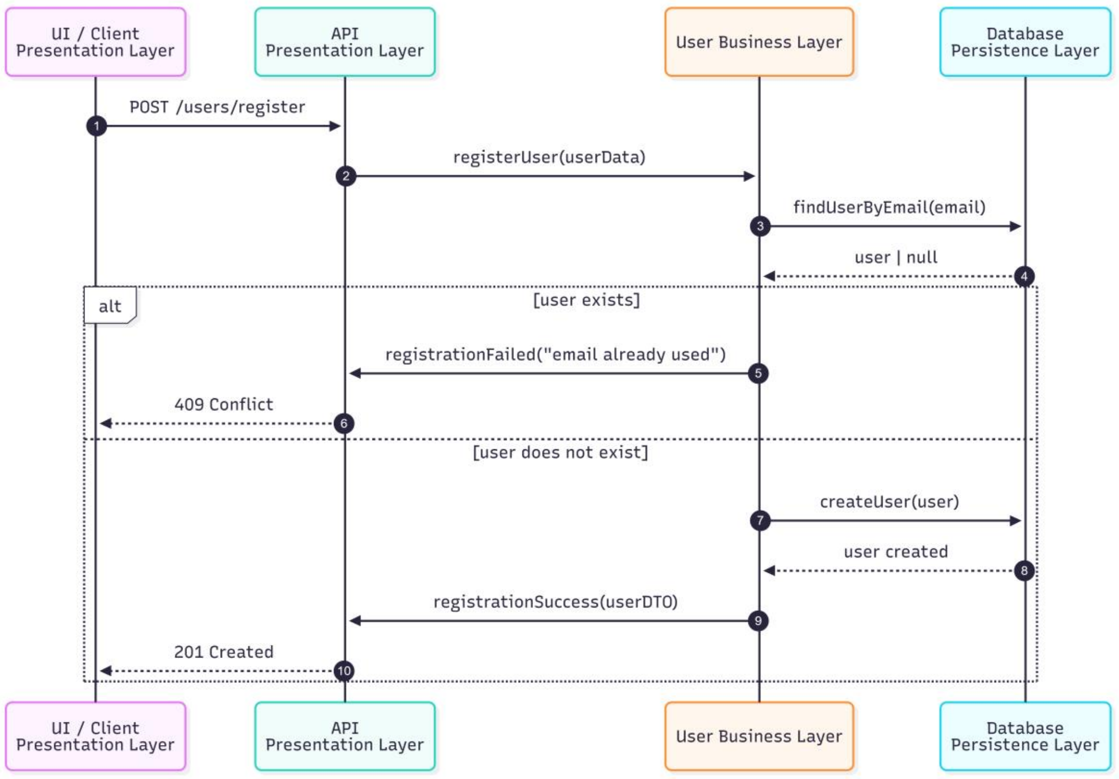
4.4.1 Description

The client sends a place creation request through the API.

The API delegates the request to the Place Business Layer, which applies the business rules and validates the provided data.

If the data is valid, the place is saved in the database via the persistence layer, and a confirmation response is returned to the client.

If the data is invalid, a bad request response is returned.



4.5 Review Submission

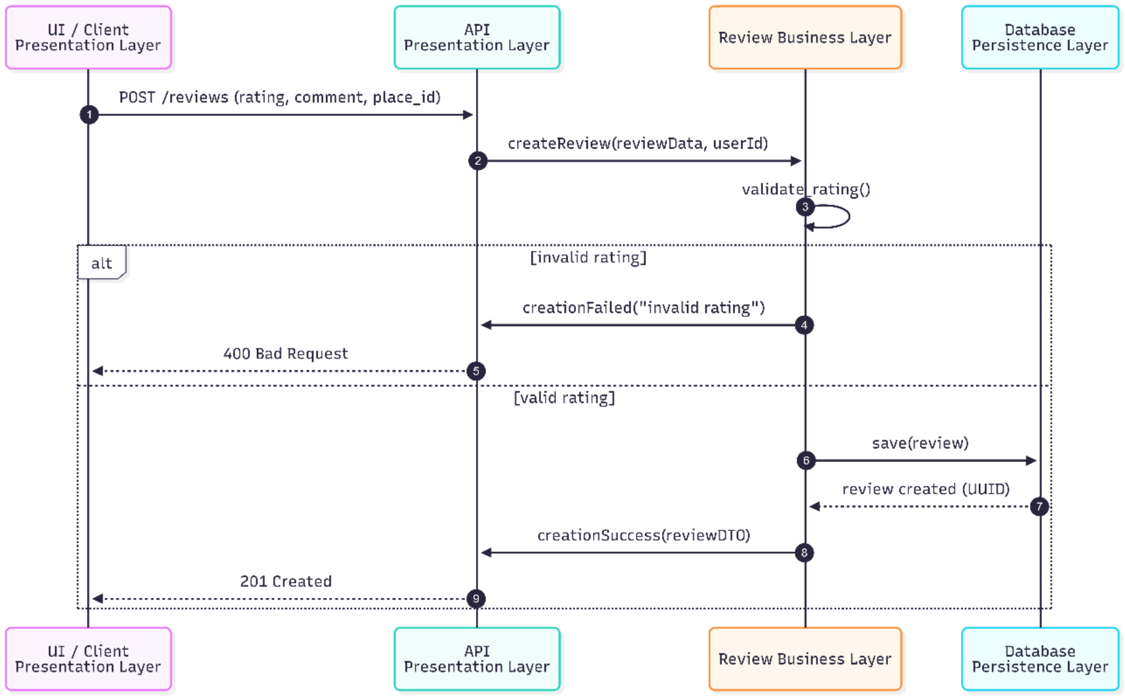
4.5.1 Description

The client submits a review through the API.

The API forwards the request to the Review Business Layer, which validates the review data, including the rating.

If the validation succeeds, the review is stored in the database and a confirmation response is returned to the client.

If the validation fails, a bad request response is returned.


UI / Client Presentation Layer
API Presentation Layer
Review Business Layer
Database Persistence Layer
POST /reviews (rating, comment, place_id)
createReview(reviewData, userId)
validate_rating()
alt
[invalid rating]
creationFailed("invalid rating")
400 Bad Request
[valid rating]
save(review)
review created (UUID)
creationSuccess(reviewDTO)
201 Created

4.6 Fetching a List of Places

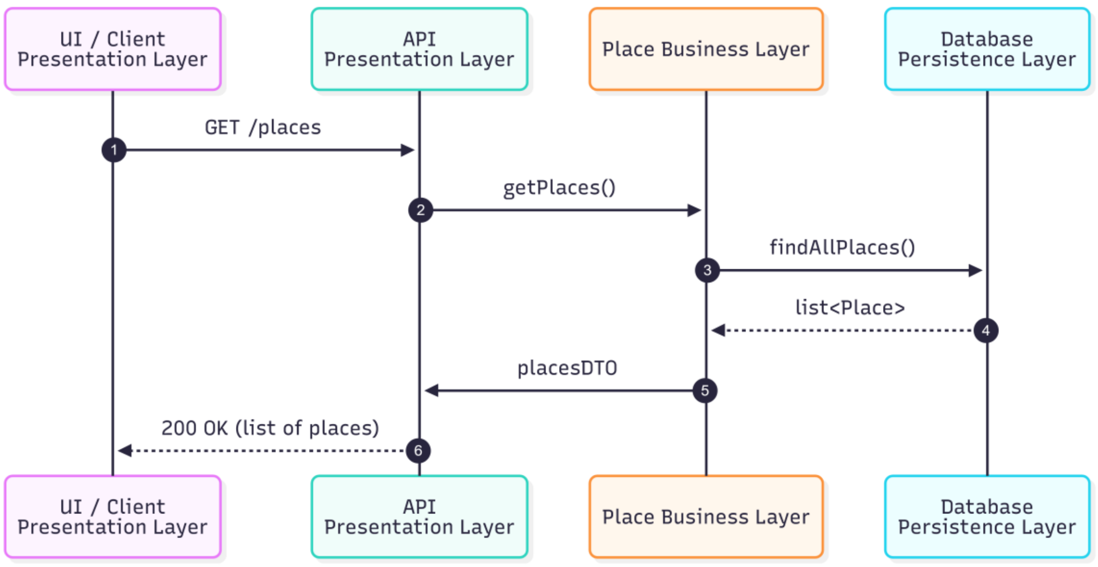
4.6.1 Description

The client sends a request to retrieve all available places.

The API forwards this request to the Place Business Layer, which is responsible for coordinating the operation.

The Business Layer queries the persistence layer to fetch the list of places from the database.

Once the data is retrieved, it is transformed into a suitable response format and returned to the client through the API.



5 Conclusion

The HBnB Evolution project defines a clear and scalable architecture for a simplified Airbnb-like application. By adopting a layered design that separates the Presentation, Business Logic, and Persistence layers, the system ensures maintainability, modularity, and ease of future extensions.

The use of UML class diagrams helped model the core business entities and their relationships, while sequence diagrams illustrated how the different layers interact during typical API operations. This structured approach provides a solid foundation for implementation and reduces potential design issues early in the development process.

Overall, the proposed architecture supports clean separation of concerns, promotes code reusability, and prepares the application for future enhancements such as authentication improvements, additional features, or scaling to larger datasets.