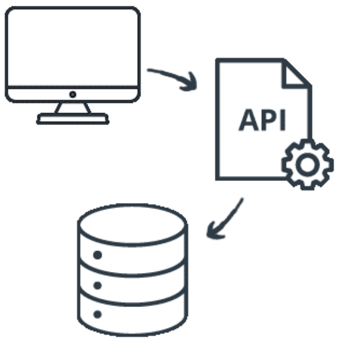
Holberton School Project 13/02/2025

<u>HBnB Evolution – Comprehensive Technical Documentation</u>

Contributors :

Rebati Sara
Planchon Valentin
Rossi Damien



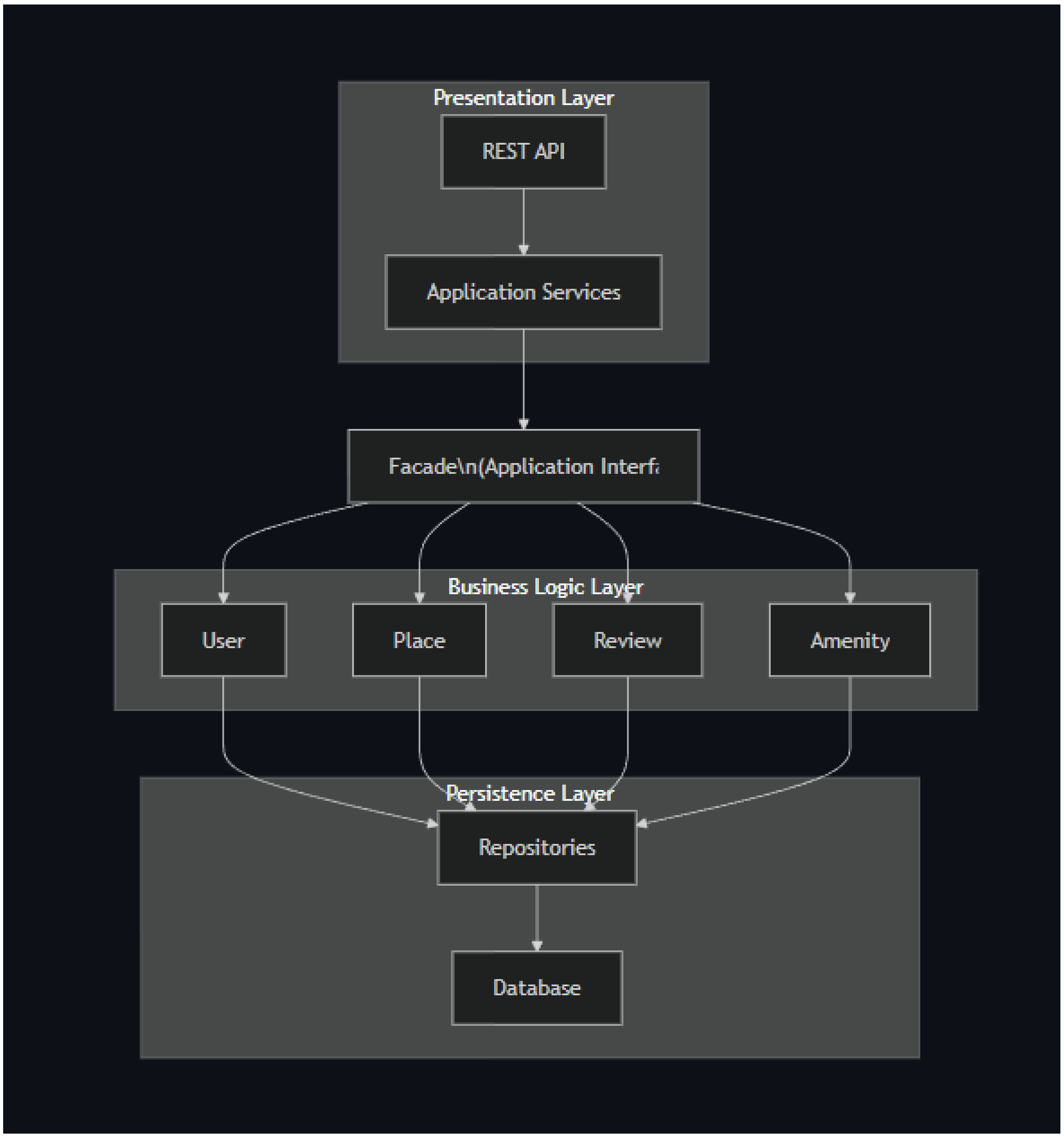
I – Introduction

HBnB Evolution is a simplified property rental platform inspired by modern accommodation marketplace applications. The goal of this project is to design a scalable, maintainable, and well-structured backend architecture that supports user management, place listing management, review submission, and amenity management.
This document serves as a technical blueprint for the HBnB Evolution application. It consolidates architecture diagrams, business logic models, and API interaction flows into a single reference document.
The objective is to provide developers with a clear understanding of system structure, data flow, and design decisions before implementation begins.
The document is divided into three main technical sections: high-level architecture, business logic modeling, and API interaction flows.

II – High-Level Architecture

The HBnB application follows a three-layer architecture composed of the Presentation Layer, Business Logic Layer, and Persistence Layer. This separation ensures modularity, improves maintainability, and simplifies future scalability. Communication between the Presentation Layer and Business Logic Layer is handled through a Facade pattern. The Facade acts as a unified entry point, reducing coupling between external services and internal business rules.

High-Level Package Diagram :

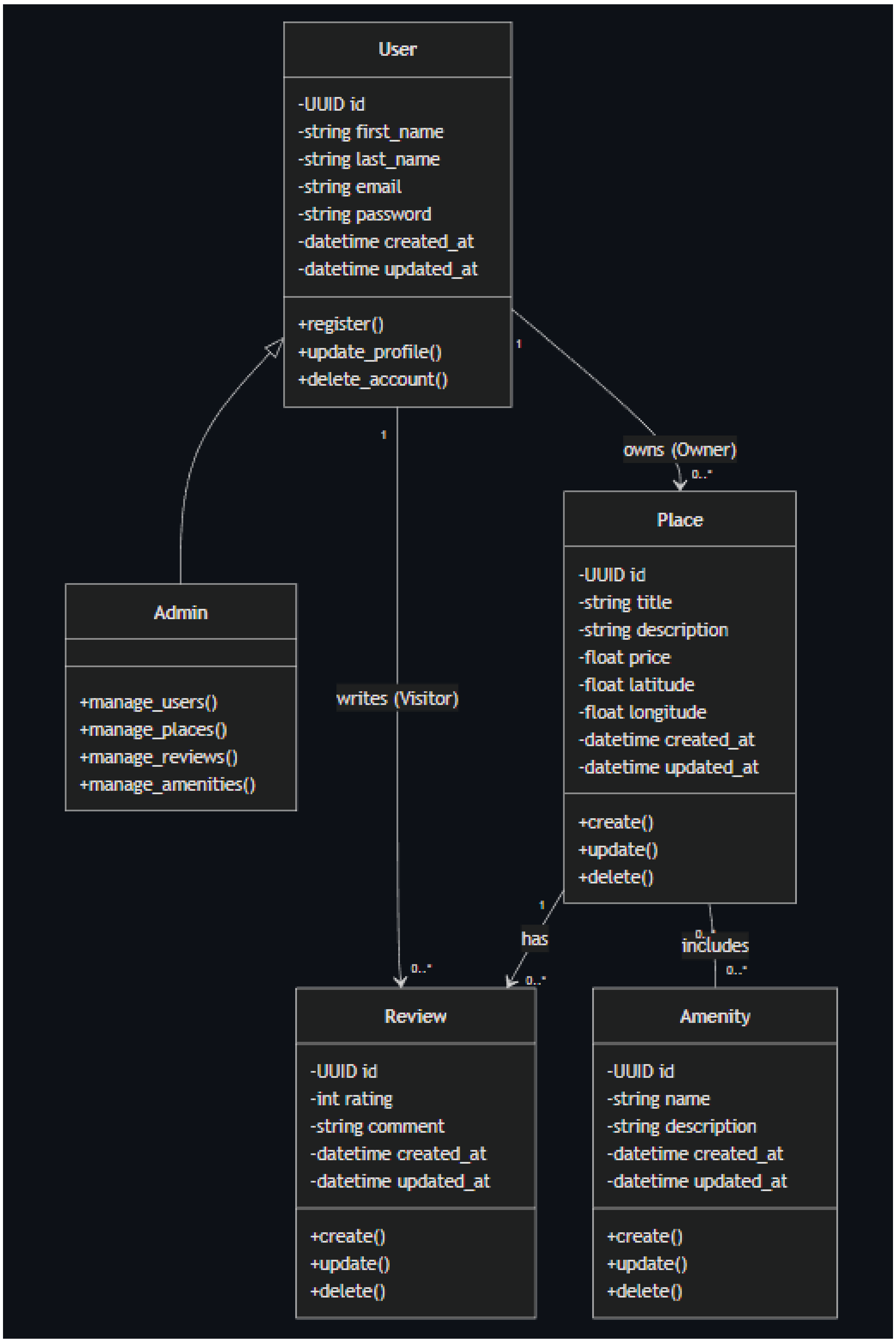


This diagram illustrates how external requests are received by the API, processed through application services, and then delegated to the Facade. The Facade coordinates business entities and ensures that all persistence operations are executed through repository interfaces. This architecture ensures clean separation of responsibilities.

III – Business Logic Layer

The Business Logic Layer contains the core domain entities: User, Place, Review, and Amenity. Each entity includes unique identification and timestamp auditing fields to ensure traceability. Relationships are designed to reflect real-world platform interactions. Users can own places and create reviews. Places can have multiple reviews and multiple amenities. Amenities can be shared across multiple places.

Detailed Class Diagram :

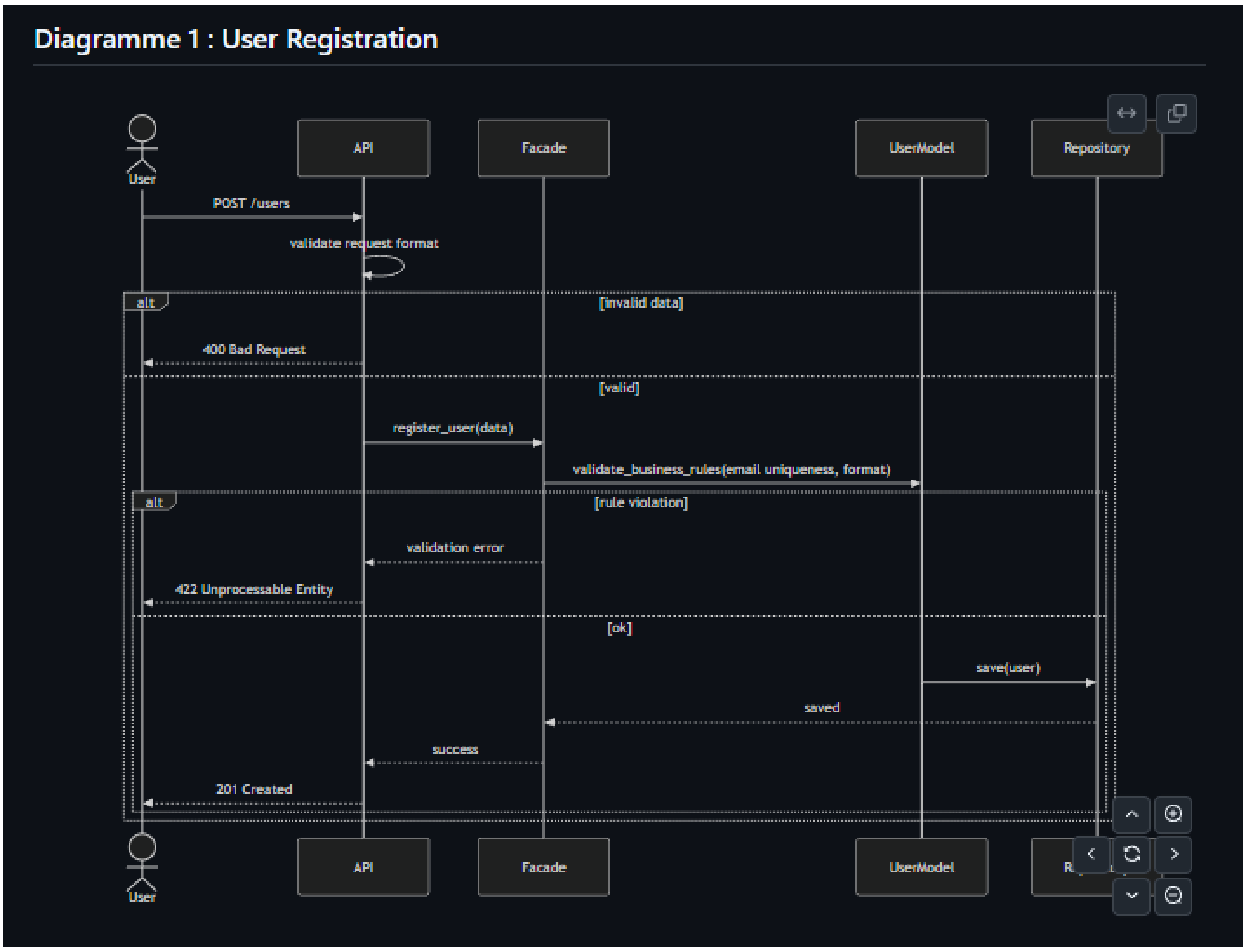


This diagram illustrates how external requests are received by the API, processed through application services, and then delegated to the Facade. The Facade coordinates business entities and ensures that all persistence operations are executed through repository interfaces. This architecture ensures clean separation of responsibilities (also methods are not showned in this diagram).

IV – API Interaction Flow

Sequence diagrams illustrate how different system layers collaborate to process API requests. They provide visibility into validation steps, business rule enforcement, and persistence operations.

User Registration :



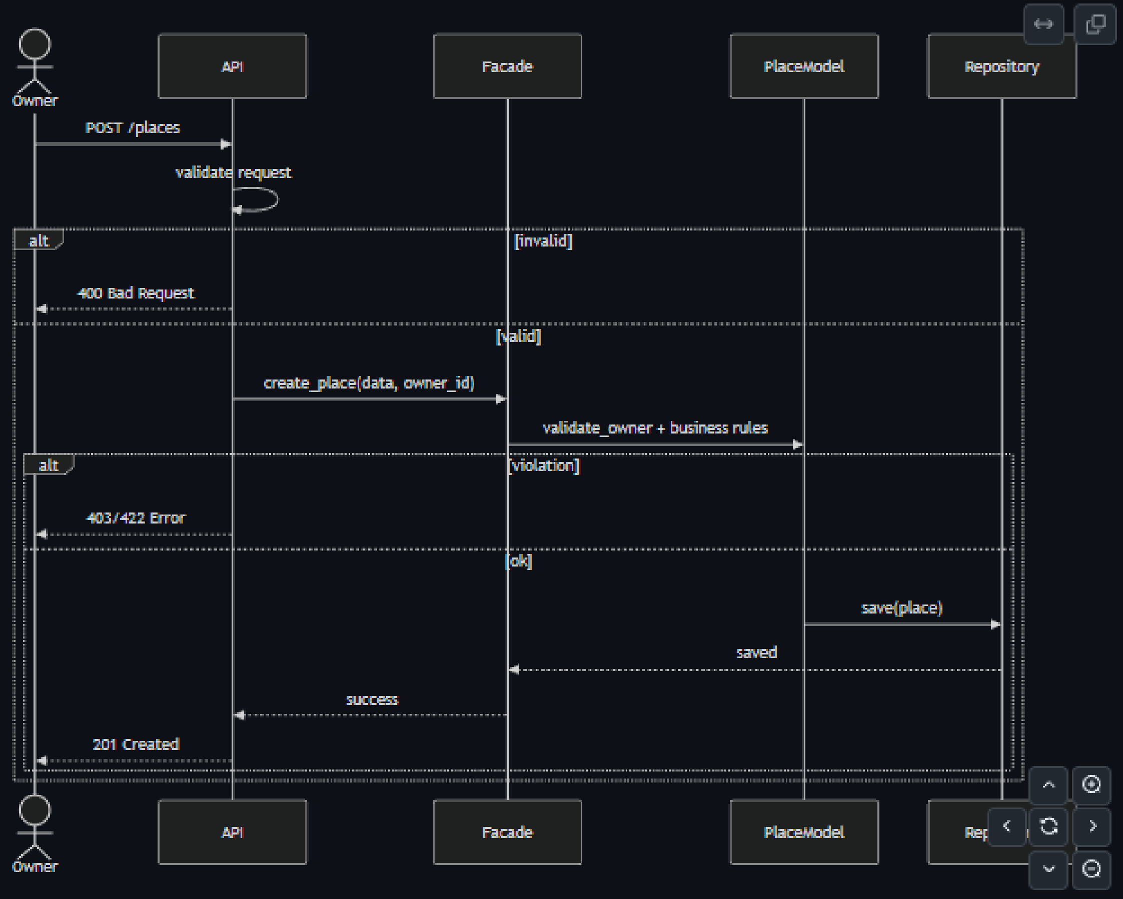
This flow shows how new users are validated at both API and business levels before being stored in the database. The separation ensures data integrity and security.

Place Creation :

This diagram represents the creation of a place by an Owner (`POST /places`).

It emphasizes:

- Validation at the API layer (request structure).
- Authorization and ownership validation at the Business Logic level.
- Multiple possible error responses:
  - **400 Bad Request** → malformed input.
  - **403 Forbidden** → ownership or permission violation.
  - **422 Unprocessable Entity** → business rule violation.
  - **201 Created** → place successfully created.
- Data persistence only after all validations succeed.



Place creation requires ownership validation and business rule verification before persistence. This ensures only authorized users can create listings.

Review Submission :

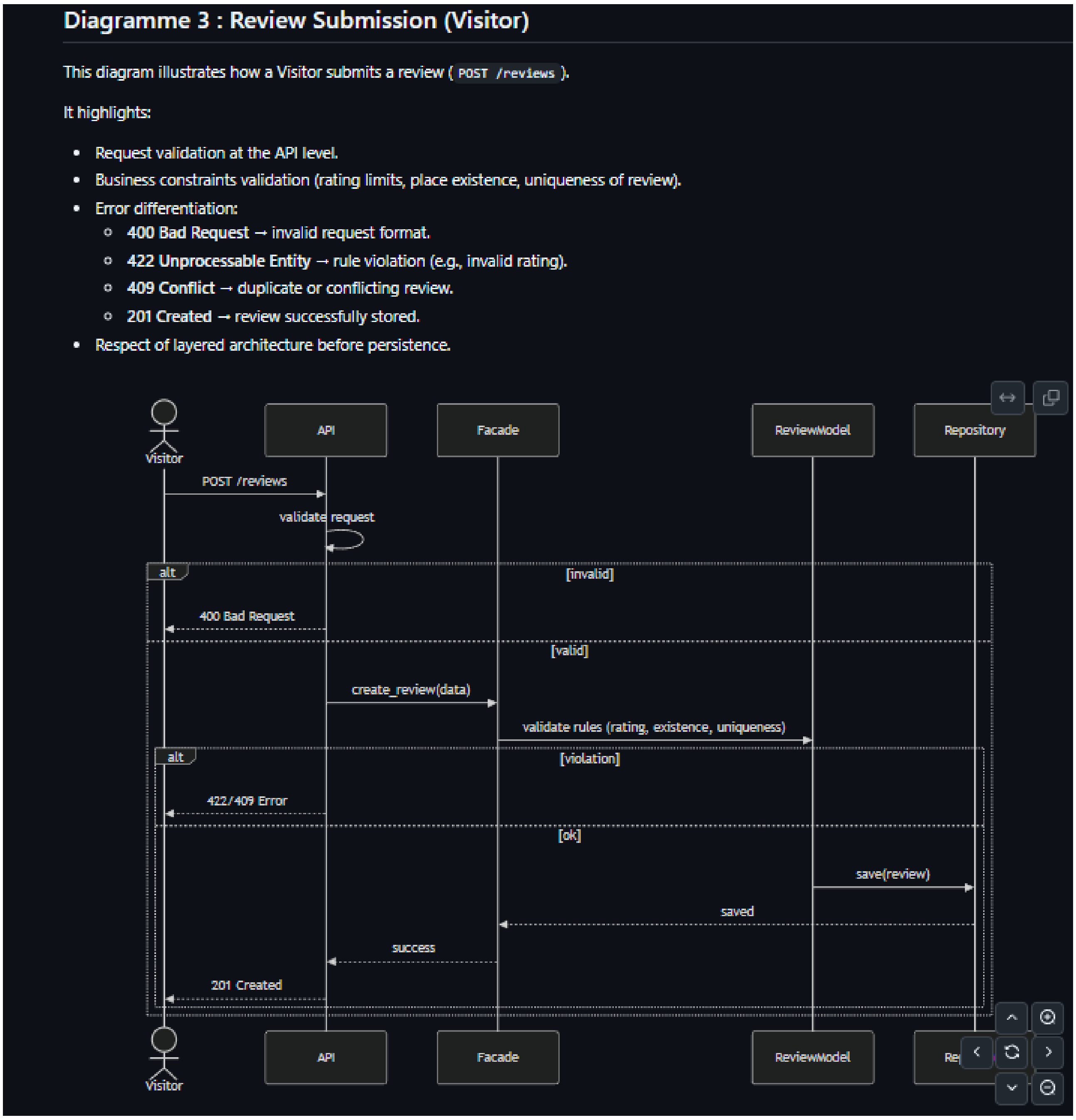


Diagramme 3 : Review Submission (Visitor)

This diagram illustrates how a Visitor submits a review (`POST /reviews`).

It highlights:

- Request validation at the API level.
- Business constraints validation (rating limits, place existence, uniqueness of review).
- Error differentiation:
  - **400 Bad Request** → invalid request format.
  - **422 Unprocessable Entity** → rule violation (e.g., invalid rating).
  - **409 Conflict** → duplicate or conflicting review.
  - **201 Created** → review successfully stored.
- Respect of layered architecture before persistence.

Reviews are validated for rating rules and uniqueness constraints before storage to maintain review reliability.

Fetching Places :

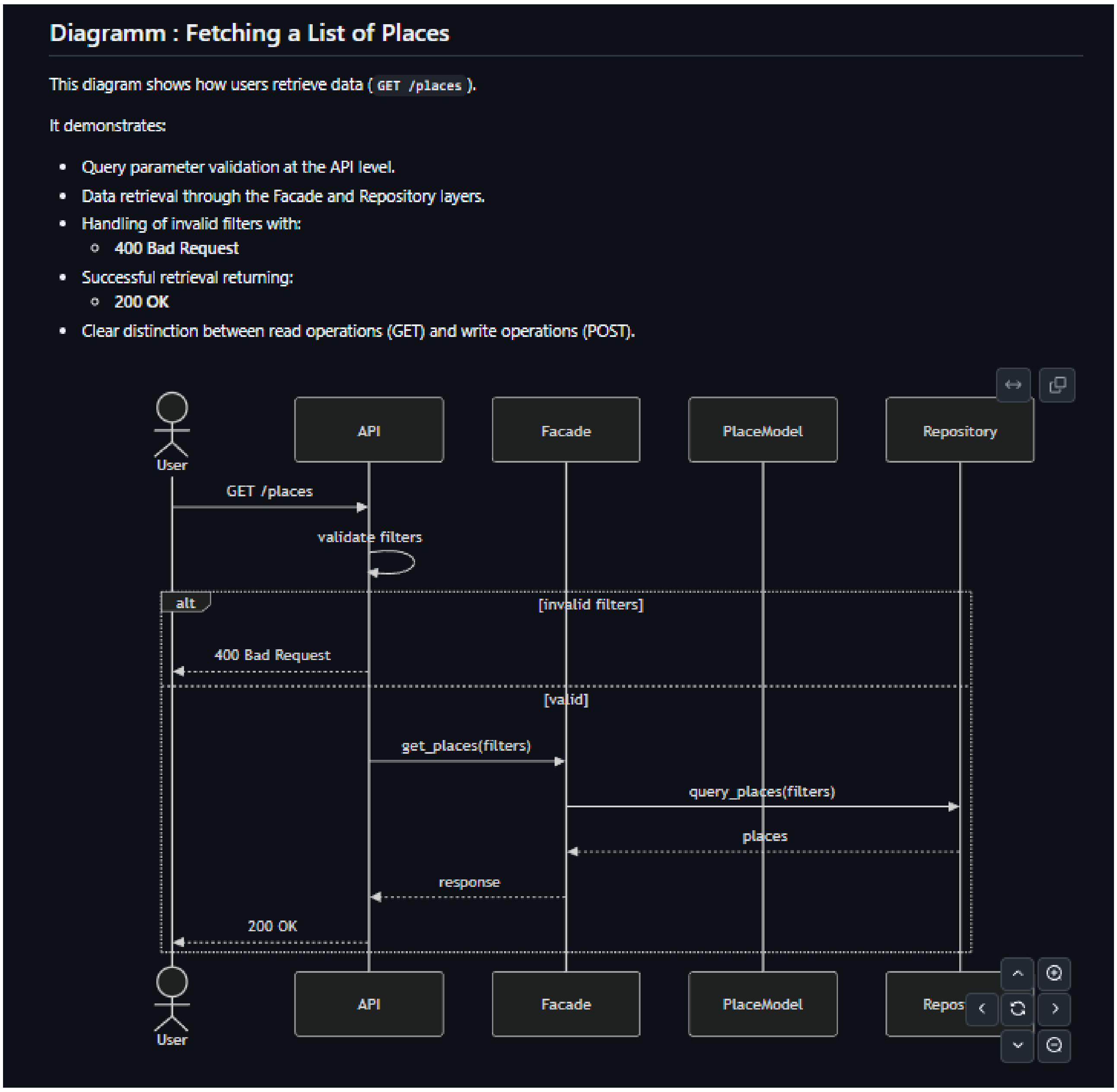


Diagramm : Fetching a List of Places

This diagram shows how users retrieve data (`GET /places`).

It demonstrates:

- Query parameter validation at the API level.
- Data retrieval through the Facade and Repository layers.
- Handling of invalid filters with:
  - 400 Bad Request
- Successful retrieval returning:
  - 200 OK
- Clear distinction between read operations (GET) and write operations (POST).

Read operations follow a simplified path focused on data retrieval while still ensuring filter validation.

V – Conclusion

This technical document provides a complete architectural overview of the HBnB Evolution backend design. By combining high-level architecture diagrams, detailed business logic modeling, and API interaction flows, it establishes a strong foundation for implementation. The layered architecture and facade design pattern improve maintainability, testability, and scalability. The business entity model ensures clarity in data relationships, and the sequence diagrams define predictable system behavior for key API operations. This document will serve as a primary reference throughout development and future system evolution.