HBnB Evolution

Technical Documentation

Norah AlNujidi Shadan Alkharji Rinad AlZuyber

Contents

1. Introduction

Purpose

This document provides the technical architecture and design specifications for the HBnB Evolution application. It serves as a blueprint to guide the implementation phases of the project.

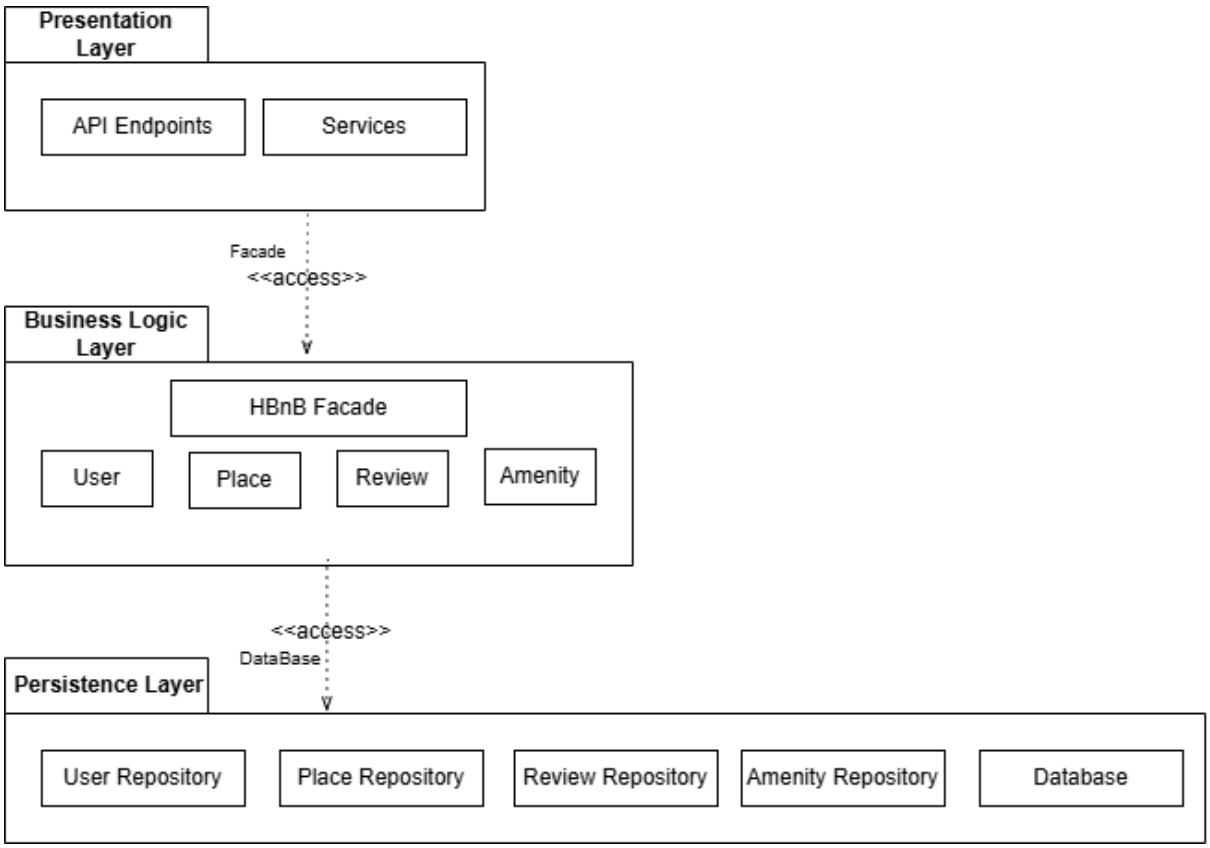
Project Overview

HBnB Evolution is a simplified AirBnB-like application that enables users to list properties, write reviews, and manage amenities.
Core Features
User Management: Registration, profile updates, and admin roles
Place Management: Create and manage property listings with details (title, description, price, location)
Review Management: Submit ratings and comments for places
Amenity Management: Manage amenities associated with places

Technical Requirements

All entities have unique UUID identifiers
Creation and update timestamps for audit purposes
Three-layer architecture with Facade pattern
Database persistence (to be implemented in Part 3)

2. High-Level Architecture



1.1 Purpose

This diagram illustrates the three-layer architecture of the HBnB application and shows communication between layers via the Facade Pattern.
It provides a clear overview of how components are organized and interact, guiding implementation and maintenance.

1.2 Layers and Components

a. Presentation Layer

- Components: API Endpoints, Services
- Responsibility: Handles user interaction and forwards requests to the Business Logic Layer.
- Communication: Via Facade to the Business Logic Layer.

b. Business Logic Layer

- Components: HBnB Facade, User, Place, Review, Amenity
- Responsibility: Implements core business rules and logic.
- Communication: Accesses Persistence Layer through Database Access.
- Notes: Facade provides a single entry point, simplifying interactions for the Presentation Layer.

c. Persistence Layer

- Components: User Repository, Place Repository, Review Repository, Amenity Repository, Database
- Responsibility: Manages data storage and retrieval.
- Notes: Separates data management from business logic, enabling flexibility and maintainability.

1.3 Dependencies

- Presentation → Business Logic: Facade
- Business Logic → Persistence: Database Access
- Inside Business Logic: All models are accessed through HBnB Facade.

1.4 Design Rationale

1. Layered Architecture: Ensures Separation of Concerns and easier maintenance.
2. Facade Pattern: Reduces complexity and provides a unified interface for Presentation Layer.
3. Repositories: Isolate database operations from business logic for flexibility.
4. Hierarchical Structure: Clear flow of data from UI to database.

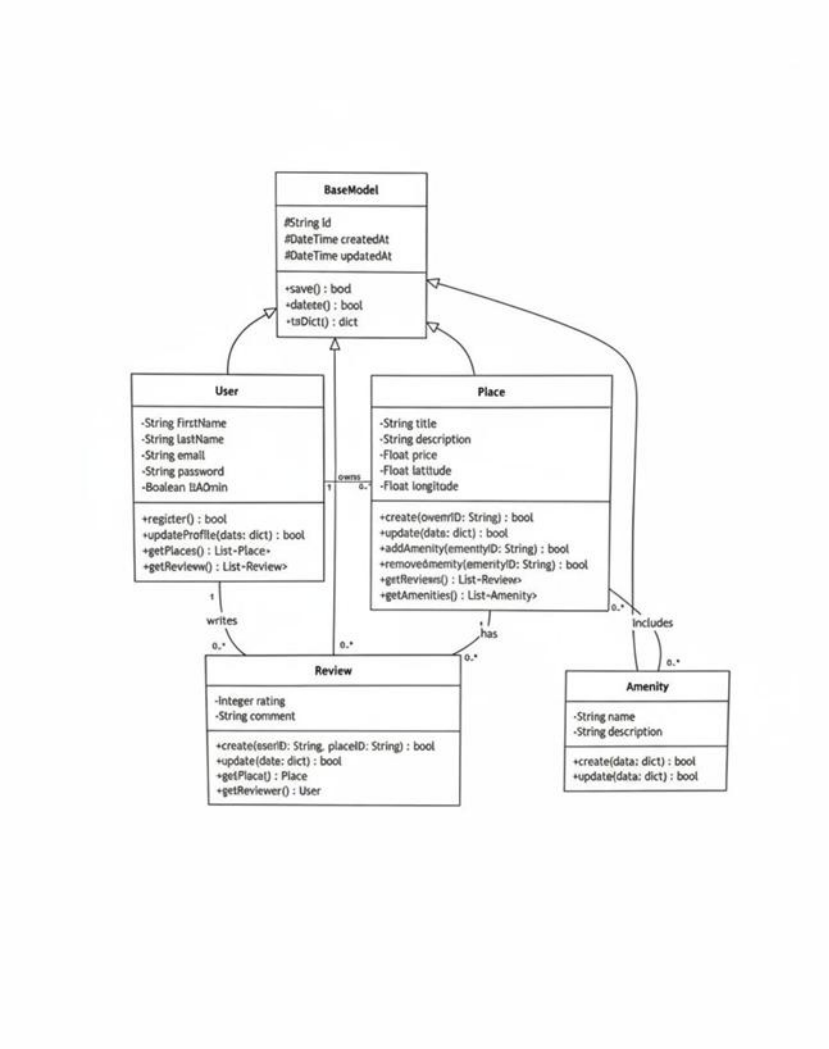
3. Business Logic Layer

3.1 Overview

The Business Logic Layer is responsible for managing the core entities of the system and enforcing the business rules. All entities in this layer inherit from a base class called **BaseModel**, which provides shared attributes such as unique identifiers and timestamp fields, as well as common methods for data persistence and management.

3.2Class Diagram

The class diagram illustrates the main entities of the Business Logic Layer and their relationships. All entities inherit from the BaseModel class, which provides common attributes and shared methods. This design promotes code reusability and consistency across the system.



3.3 Entity Descriptions

BaseModel

The BaseModel class serves as the parent class for all entities in the system, providing shared attributes and common functionality.

-Attributes: Unique identifier (UUID4), creation timestamp, last update timestamp

- Methods: Persist data to the database, remove records, convert entity data to a dictionary representatio

User:

-The User entity represents all platform users, including both regular users and administrators.

-Key Attributes: First name, last name, unique email address, hashed password, administrator flag

-Key Methods: User registration with validation, profile updates, retrieval of owned places and written reviews

-Business Rules: Email addresses must be unique, and passwords must be securely hashed

Place:

-The Place entity represents rental properties available on the platform.

-Key Attributes: Title, description, price, and geographic coordinates (latitude and longitude)

-Relationships: Each place is owned by a single user and may have multiple amenities and reviews

-Key Methods: Creation with an assigned owner, updating property details, and managing associated amenities

-Business Rules: The price must be a positive value, and geographic coordinates must fall within valid ranges

Review:

-The Review entity represents user feedback for rental properties.

-Key Attributes: Rating value (ranging from 1 to 5) and a textual comment

-Relationships: Each review is written by one user and associated with one place

-Key Methods: Creating and updating reviews

-Business Rules: Users are not allowed to review their own properties, and only one review per user is permitted for each place

Amenity:

-The Amenity entity represents features or services offered by properties, such as WiFi or a swimming pool.

-Key Attributes: Unique name and description

-Relationships: Amenities may be associated with multiple places through a many-to-many relationship

-Key Methods: Creating and updating amenity records

3.4 Key Relationships

Inheritance (BaseModel → All Entities)

All entities inherit common attributes and behaviors from the BaseModel class. This approach:

Reduces code duplication by following the DRY principle

Ensures consistency across all entities

Associations:

User → Place (One-to-Many): A single user can own multiple places

User → Review (One-to-Many): A user can write multiple reviews

Place → Review (One-to-Many): A place can have multiple reviews

Place Amenity (Many-to-Many): Places can share amenities, requiring a junction (association) table

3.5 Design Decisions

- Encapsulation: Private attributes combined with public methods ensure controlled access and proper data validation

- Loose Coupling: Entities reference one another using identifiers rather than full object instances

- Single Responsibility Principle: Each entity is designed to fulfill one specific purpose

- toDict() Method: Facilitates JSON serialization for API responsesz

4. API Interaction Flow

API Interaction Flow

This section presents sequence diagrams for four key API operations, showing the interaction between the Presentation Layer, Business Logic Layer, and Persistence Layer.

4.1 User Registration

API Call: POST /api/users/register

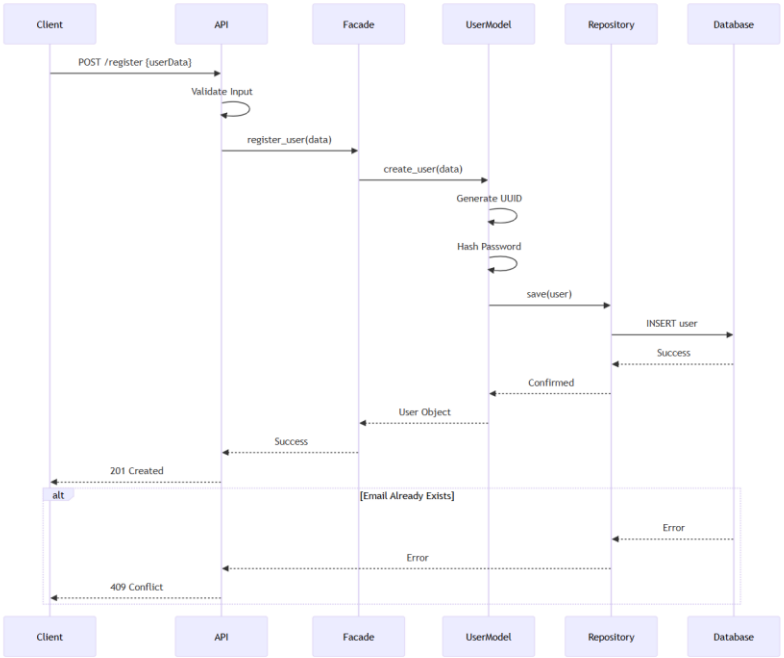
Description:

This sequence shows how a new user registers in the system. The API receives user data, the Facade coordinates with the User Model to validate and create the user, and the Repository persists the data to the database.

Key Steps:

• Client sends registration request with user details
• API validates input data format
• Facade processes the request through User Model
• User Model generates UUID and hashes password
• Repository saves user to database
• Success response returned with user ID

Sequence Diagram:



4.2 Place Creation

API Call: POST /api/places

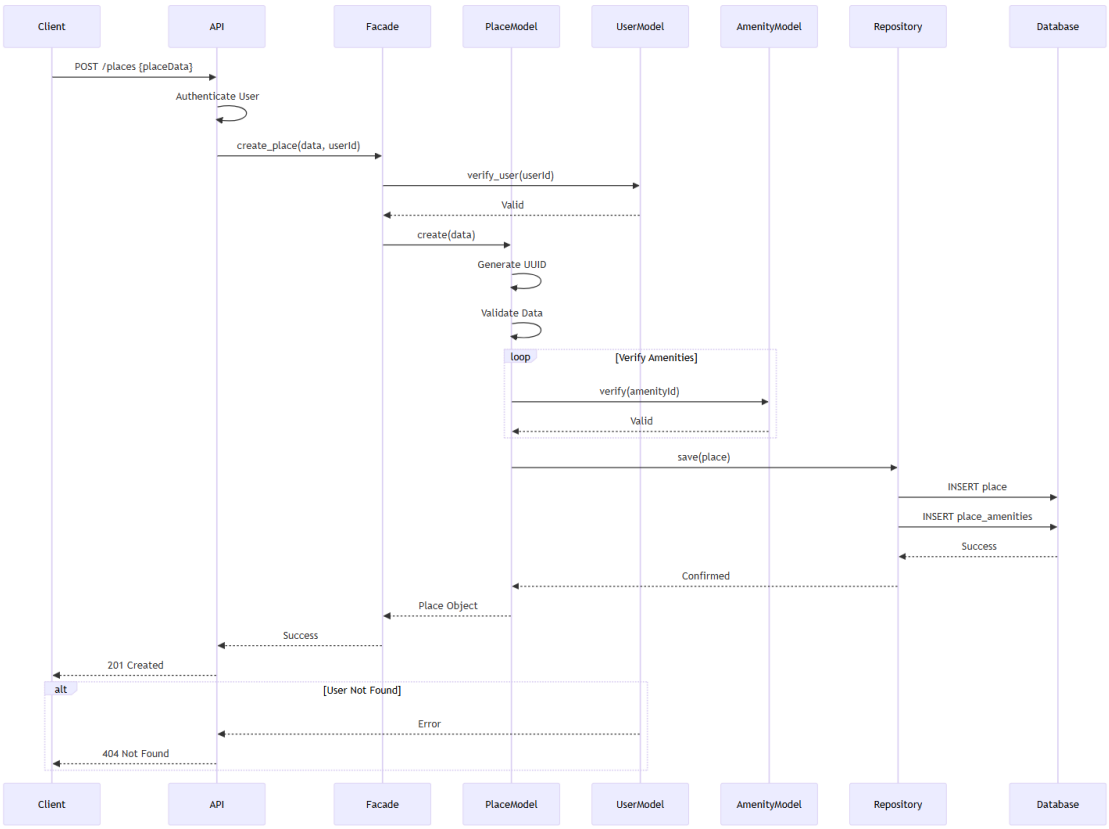
Description:

This sequence illustrates the process of creating a new place listing. The system authenticates the user, validates the place data including coordinates and price, verifies associated amenities exist, and persists the place with its amenity relationships.

Key Steps:

• API authenticates the user
• Facade verifies user exists and is valid
• Place Model generates UUID and validates business rules
• System verifies each amenity in the amenities list
• Repository saves place and creates place-amenity associations
• Created place returned with all details

Sequence Diagram:



4.3 Review Submission

API Call: POST /api/reviews

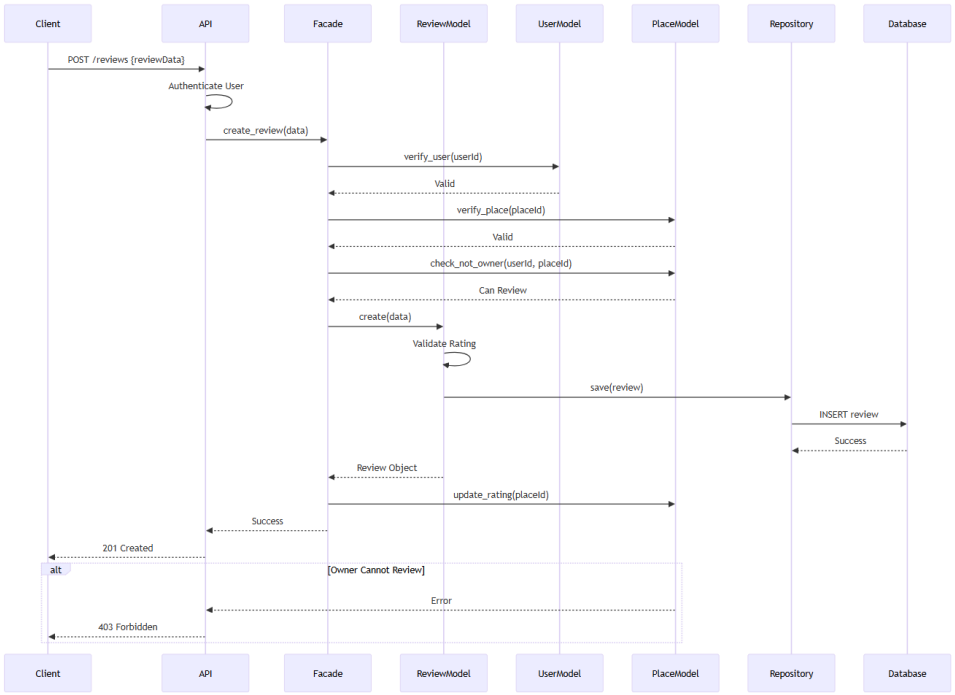
Description:

This sequence shows how users submit reviews for places. The system ensures the user is authenticated, verifies both user and place exist, checks that the user is not the place owner (business rule), validates the rating, and saves the review while updating the place's average rating.

Key Steps:

- API authenticates user
- Facade verifies user and place exist
- System checks user is not the place owner
- Review Model validates rating (1-5) and creates review
- Repository saves review to database
- Place rating is recalculated
- Review details returned

Sequence Diagram:



4.4 Fetching List of Places

API Call: GET /api/places

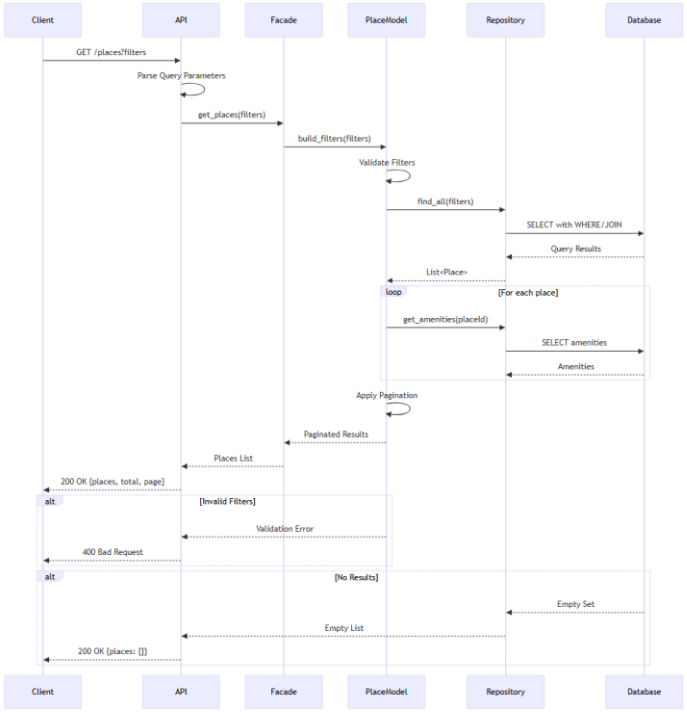
Description:

This sequence demonstrates how the system retrieves a filtered and paginated list of places. The API parses query parameters (filters, pagination), the Business Logic Layer builds and validates filters, the Repository executes the database query with appropriate JOINs for amenities, and results are paginated and returned.

Key Steps:

- Client sends GET request with query parameters (filters, pagination)
- API parses and validates parameters
- Place Model builds filter criteria
- Repository executes filtered query with amenity JOINs
- System retrieves amenities for each place
- Results are paginated
- Formatted list returned with metadata

Sequence Diagram:



End of Documentation.