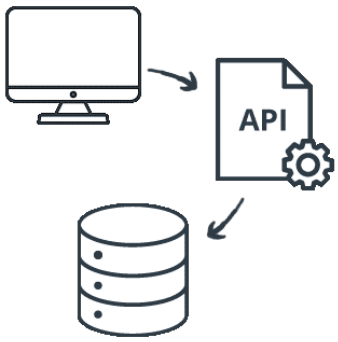
HBnB Evolution – Comprehensive Technical Documentation

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 :

```
flowchart TB

subgraph Presentation["Presentation Layer"]
      API["REST API"]
      Services["Application Services"]
end

Facade["Facade (Application   Interface)"]

subgraph Business["Business Logic Layer"]
      User["User"]
      Place["Place"]
      Review["Review"]
      Amenity["Amenity"]
end

subgraph    Persistence["Persistence Layer"]
      Repository["Repositories"]
      Database["Database"]
end

API --> Services
Services --> Facade

Facade -->  User
Facade --> Place
Facade --> Review
Facade --> Amenity

User --> Repository
Place --> Repository
Review --> Repository
Amenity --> Repository

Repository --> Database
```

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 :

```
classDiagram

class User {
  UUID id
  string first_name
  string last_name
  string email
  string password
  datetime created_at
  datetime updated_at

}

class Admin {
  manage_users()
  manage_places()
  manage_reviews()
  manage_amenities()
}

class Place {
  UUID id
  string title
  string description
  float price
  float latitude
  float longitude
  datetime created_at
  datetime updated_at
}

class Review {
  UUID id
  int rating
  string comment
  datetime created_at
  datetime updated_at
}

class Amenity {
  UUID id
  string name
  string description
  datetime created_at
  datetime updated_at
}

User <|-- Admin
User "1" --> "0..*" Place : owns
User "1" --> "0..*" Review : writes
Place "1" --> "0..*" Review : has
Place "0..*" -- "0..*" Amenity : includes
```

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 :

```
sequenceDiagram
actor User
participant API
participant Facade
participant UserModel
participant Repository

User ->> API: POST /users
API ->> API: Validate request
API ->> Facade: register_user()
Facade ->> UserModel: validate business rules
UserModel ->> Repository: save(user)
Repository -->> API: success
API -->> User: 201 Created
```

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 :

```
sequenceDiagram
actor Owner
participant API
participant Facade
participant PlaceModel
participant Repository

Owner ->> API: POST /places
API ->> Facade: create_place()
Facade ->> PlaceModel: validate ownership and rules
PlaceModel ->> Repository: save(place)
Repository -->> API: success
API -->> Owner: 201 Created
```

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

Review Submission :

```
sequenceDiagram
actor Visitor
participant API
participant Facade
participant ReviewModel
participant Repository

Visitor ->> API: POST /reviews
API ->> Facade: create_review()
Facade ->> ReviewModel: validate rating and constraints
ReviewModel ->> Repository: save(review)
Repository -->> API: success
API -->> Visitor: 201 Created
```

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

Fetching Places :

```
sequenceDiagram
actor User
participant API
participant Facade
participant Repository

User ->> API: GET /places
API ->> Facade: get_places()
Facade ->> Repository: query places
Repository -->> API: results
API -->> User: 200 OK
```

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.