HBnB Evolution App – Techical Document

# Introduction

This document provides a **technical overview of the HBnB application**, serving as a guide for the design and implementation of its core components. Its primary purpose is to clearly define the **architecture, data models, and interaction flows** within the system, enabling developers to understand how the various layers of the application communicate and operate together.

The HBnB project is an **online platform for booking and managing rental properties**, similar in concept to popular home-sharing services. It allows users to **register, browse places, submit reviews, and interact with amenities**, while ensuring that all business rules and data persistence requirements are properly enforced.

This technical document focuses on:

1. **Architectural Structure:** Illustrating the three-layer design of the application (Presentation, Business Logic, Persistence) and how these layers interact via the **facade pattern**.
2. **Data Models:** Defining the entities of the system, their attributes, methods, and relationships, including inheritance, composition, and aggregation.
3. **Interaction Flows:** Showing how the system processes key operations, such as user registration, place creation, and review submission, using **sequence diagrams**.
4. **Implementation Guidance:** Providing a reference for developers to ensure consistency, maintainability, and clarity during the coding process.

By following this document, developers will gain a **comprehensive understanding of HBnB’s system design**, which will facilitate efficient implementation, testing, and future maintenance of the application.

# High-Level Architecture

The HBnB application is designed using a **layered architecture**, which organizes the system into three distinct layers:

1. **Presentation Layer (Services & API)**
   * Handles all user interactions, including API endpoints and service interfaces.
   * Receives requests from clients, processes them via the business logic layer, and returns responses.
2. **Business Logic Layer (Models & Rules)**
   * Contains the **core domain models** such as User, Place, Review, and Amenity.
   * Implements **business rules**, validation logic, and orchestrates interactions between the presentation and persistence layers.
3. **Persistence Layer (Database Access)**
   * Manages data storage and retrieval.
   * Includes DAOs or repository classes responsible for CRUD operations on entities.

**Facade Pattern**

The **facade pattern** is used to **simplify communication between layers**. Instead of exposing all internal operations directly, each layer provides a **unified interface** that hides the underlying complexity:

* The **Presentation Layer** interacts with the **Business Logic Layer** via a facade interface.
* The **Business Logic Layer** accesses the **Persistence Layer** through a facade that abstracts database operations.
* This design promotes **modularity, maintainability, and loose coupling** between components.

A diagram of a business

AI-generated content may be incorrect.

**Diagram Explanation:**

* The **Presentation Layer** sends requests to the **Business Logic Layer** using a unified interface (facade).
* The **Business Logic Layer** executes operations and communicates with the **Persistence Layer** to manage data.
* This structure ensures **clear separation of concerns**, making the system easier to extend, maintain, and test.

# Detailed Class Diagram

**A diagram of a computer

AI-generated content may be incorrect.**

The HBnB system is modeled using **object-oriented design principles**, with entities representing the core objects in the application. The following diagram and explanations show the classes, their attributes, methods, and relationships, and how they support the business logic.

## Entities and Their Roles

1. **BaseEntity (Abstract Class)**
   * Common superclass for all entities.
   * Attributes: ID, CreationDate, LastUpdateDate.
   * Provides **consistency and automatic tracking** of entity metadata.
2. **User**
   * Represents the system users (guests, property owners, or admins).
   * Core methods:
     + register() → creates a new user (class method).
     + updateProfile() → updates personal details.
     + delete() → removes user from the system.
   * **Relationships:**
     + Owns multiple Place entities.
     + Can write multiple Review entities.
3. **Place**
   * Represents rental properties.
   * Core methods:
     + create() → adds a new place.
     + update() and delete() → modify or remove the place.
   * **Relationships:**
     + Owned by a User (composition: place cannot exist without owner).
     + Can have multiple Amenity entities (aggregation).
     + Can contain multiple Review entities (composition).
4. **Amenity**
   * Represents features of a place (e.g., Wi-Fi, pool, parking).
   * Can exist independently of any specific place (aggregation).
   * Methods allow creation, listing, updating, and deletion.
5. **Review**
   * Represents feedback provided by users for a place.
   * Attributes: rating, comment, placeID, userID.
   * **Relationships:**
     + Depends on both User and Place (composition if strict deletion, or aggregation if “deleted user” reviews are allowed).
   * Core methods: create(), listByPlace(), update(), delete().

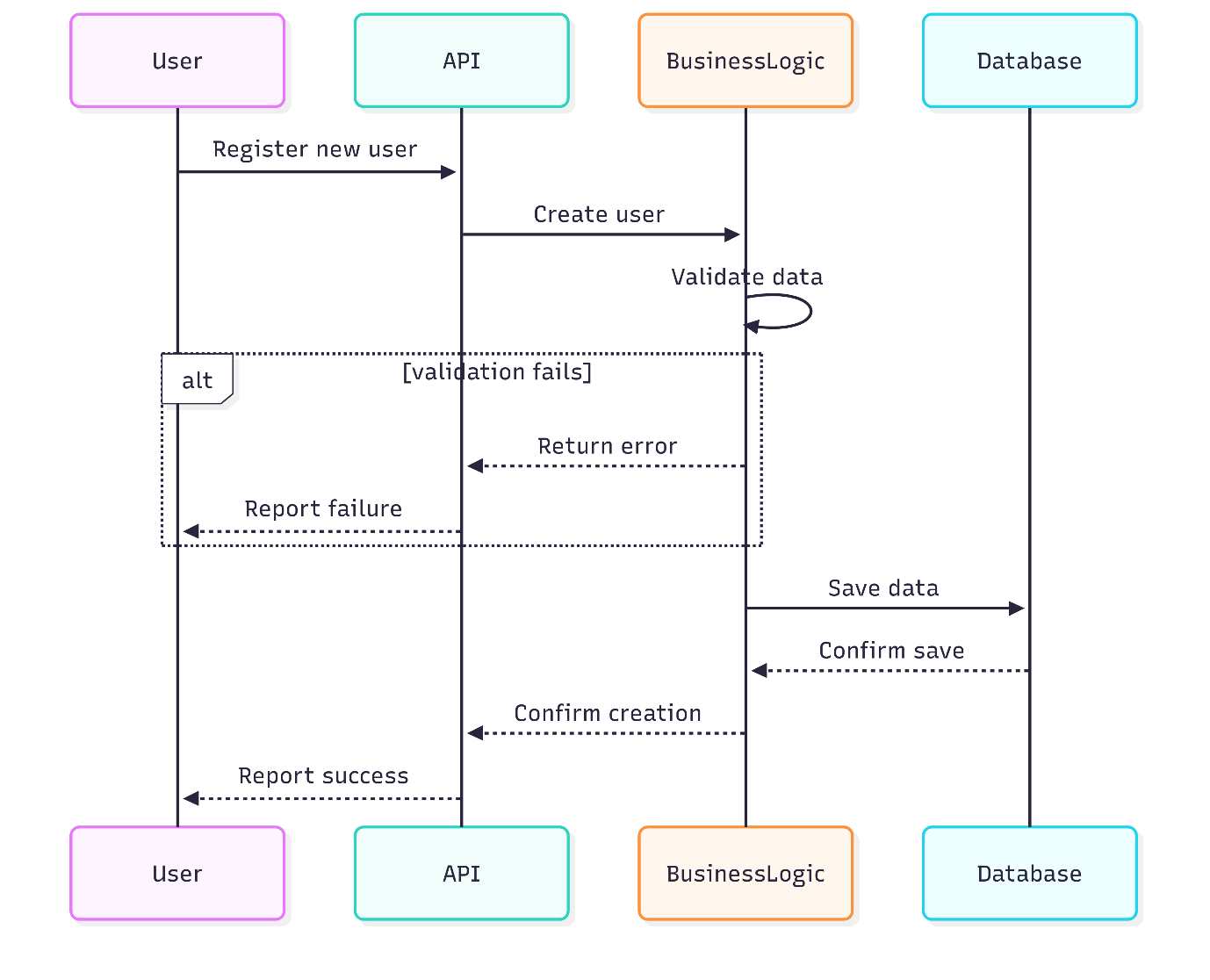
# API Interaction Flow

The **API Interaction Flow** describes the **communication and data exchange between clients (such as web or mobile apps) and the server-side components** of the HBnB system. It focuses on how requests travel through the application, how each layer processes them, and how responses are returned to the client.

Key points about API Interaction Flow:

1. **Purpose:**
   * To provide a **clear picture of how client requests are handled** from start to finish.
   * Helps developers understand the **path of data**, where validation occurs, and how business rules are enforced.
2. **Components Involved:**
   * **Client/Consumer:** The source of API requests (web app, mobile app, or third-party system).
   * **Presentation Layer (API):** Receives requests, performs authentication and authorization, and forwards them to the business logic.
   * **Business Logic Layer:** Applies business rules, orchestrates operations, and ensures that data meets system requirements.
   * **Persistence Layer (Database):** Manages storage, retrieval, and integrity of data entities.
3. **Characteristics:**
   * **Layered Communication:** Requests flow through the API to the business logic, then to the database, and results flow back in reverse.
   * **Validation and Error Handling:** Each layer can detect and handle errors, ensuring only valid operations proceed.
   * **Standardized Interface:** APIs provide a **consistent interface** to clients, hiding internal complexity (often implemented via the **facade pattern**).
   * **Traceability:** Interaction flows can be traced step-by-step, which is useful for debugging, auditing, and optimizing performance.

## User Registration

****

**Explanation:**

* **User** sends a registration request to the **API**.
* **BusinessLogic** validates user details internally (self-call).
* **Validation failure:** Error is immediately returned to the user.
* **Validation success:** User data is persisted in the **Database**, and success confirmation flows back to the user.
* **Key concepts:** Shows the **alt fragment** for conditional flows and internal validation via **self-call activation**.

## Review Submission

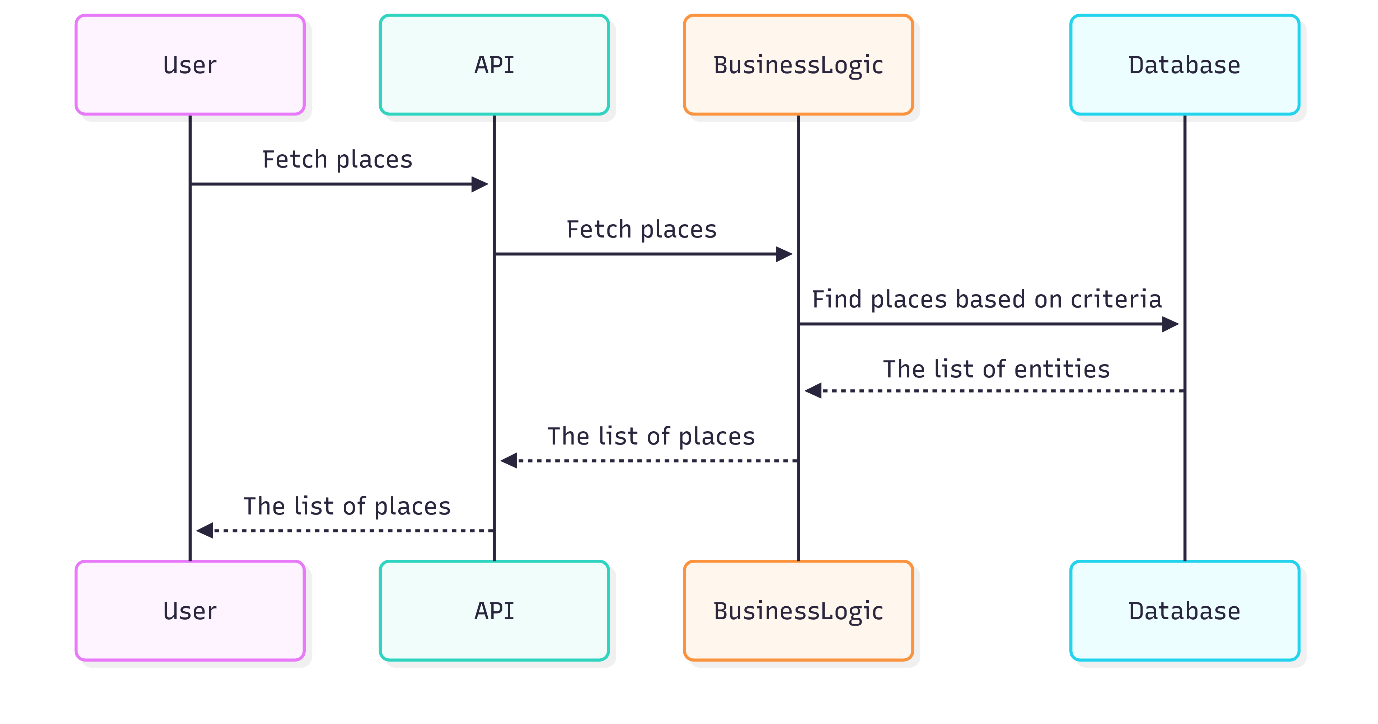
**A diagram of a software company

AI-generated content may be incorrect.**

**Explanation:**

* Users submit reviews through the **API**.
* **BusinessLogic** ensures the place exists, the user is valid, and review content meets requirements.
* Validation failures return immediate error responses.
* Successful reviews are persisted in the database, and confirmation is sent back to the user.

## Fetching Places



**Explanation:**

* Users request a list of places with optional filters.
* **BusinessLogic** queries the database and applies additional filtering or sorting logic.
* The result is returned stepwise back to the **User** through the **API**.

## Place Creation

A diagram of a software company

AI-generated content may be incorrect.

**Explanation:**

* **User** sends a request to create a new property through the **API**.
* **BusinessLogic** validates the **owner** (user exists and is authorized) and the **place details** (title, description, price, location, amenities).
* **Validation failure:** Error response is immediately returned to the user.
* **Validation success:** Place data is persisted in the **Database**.
* Confirmation is sent back to the user, completing the creation flow.