HBnB Evolution

Introduction

Purpose and Scope

This technical document serves as a comprehensive blueprint for the development of the HBnB

Evolution application. The primary objective of this document is to provide a detailed overview of the system's architecture, design, and interactions. It will guide the implementation phases and ensure a clear understanding of the application's structure and functionality.

Project Overview

HBnB Evolution is a simplified version of an AirBnB-like application designed to facilitate various operations related to property rentals. The application allows users to manage their profiles, list properties, leave reviews, and manage amenities associated with properties. The core functionalities include:

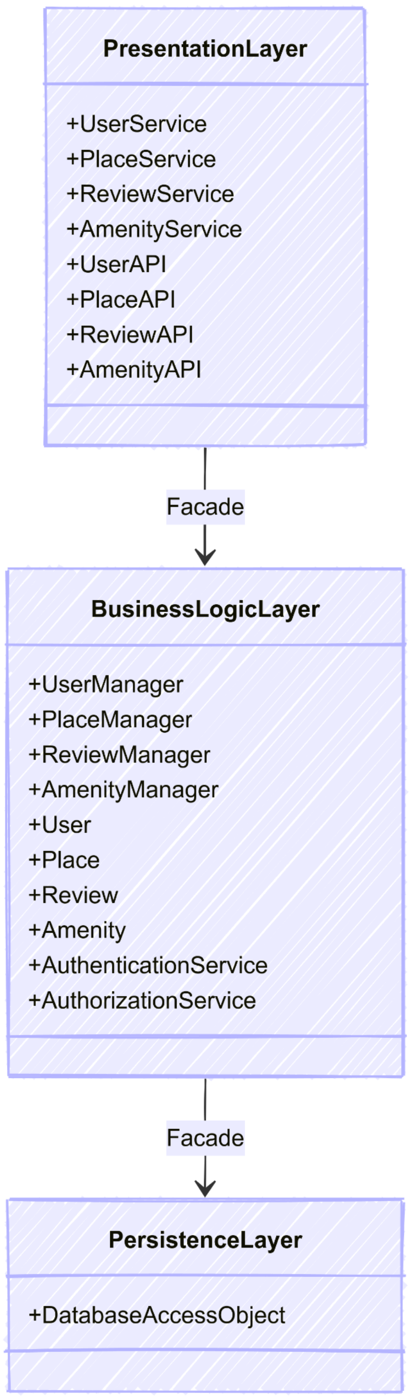
- **User Management**: Users can register, update their profiles, and be identified as either regular users or administrators.
- **Place Management**: Users can list properties they own, specifying details such as name, description, price, and location. Each property can also have a list of amenities.
- **Review Management**: Users can leave reviews for properties they have visited, including a rating and a comment.
- **Amenity Management**: The application manages amenities that can be associated with properties.

Role of the Technical Document

This document consolidates all the diagrams and explanatory notes created during the initial phase of the project. It includes:

- **High-Level Package Diagram**: Illustrates the three-layer architecture of the application and the communication between these layers via the facade pattern.
- **Detailed Class Diagram for the Business Logic Layer**: Focuses on the User, Place, Review, and Amenity entities, including their attributes, methods, and relationships.
- **Sequence Diagrams for API Calls**: Shows the interaction between the layers and the flow of information for key API calls such as user registration, place creation, review submission, and fetching a list of places.

High-Level Package Diagram



Purpose of the diagram

The purpose of the diagram is to visually represent the architecture of the system, specifically showing the different layers and their interactions. This type of diagram helps in understanding the overall structure and organization of the system, as well as the relationships between various components.

Key components or classes involved

The key components or classes involved in the package diagram are:

Presentation Layer:

- UserService: Handles user-related operations.
- PlaceService: Manages property listing operations.
- ReviewService: Manages review-related operations.
- AmenityService: Manages amenities.
- UserAPI: Exposes endpoints for user-related operations.
- PlaceAPI: Exposes endpoints for place-related operations.
- ReviewAPI: Exposes endpoints for review-related operations.
- AmenityAPI: Exposes endpoints for amenity-related operations.

Business Logic Layer:

- UserManager: Handles core business logic for user-related operations.
- PlaceManager: Handles core business logic for place-related operations.
- ReviewManager: Handles core business logic for review-related operations.
- AmenityManager: Handles core business logic for amenity-related operations.
- User: Represents a user entity.
- Place: Represents a place entity.
- Review: Represents a review entity.
- Amenity: Represents an amenity entity.
- AuthenticationService: Manages user authentication (e.g., login, logout).
- AuthorizationService: Manages user authorization.

Persistence Layer:

- DatabaseAccessObject: Handles direct interactions with the database.

Relationships:

- PresentationLayer --> BusinessLogicLayer: Uses the Facade pattern to simplify communication between the presentation and business logic layers.
- BusinessLogicLayer --> PersistenceLayer: Uses the Facade pattern to simplify communication between the business logic and persistence layers.

These components and their relationships are visually represented in the diagram to illustrate the architecture and interactions within the system.

Design decisions and their rationale

These design decisions aim to create a modular, maintainable, and scalable system by clearly separating concerns, simplifying interactions, and defining clear responsibilities for each component.

Layered Architecture:

1. Separation of Concerns:

- Presentation Layer: Handles user interactions and exposes endpoints for various operations. This separation ensures that the user interface and API endpoints are decoupled from the business logic and data access layers.

- Business Logic Layer: Contains the core business logic and operations for different entities. This layer is responsible for processing data and applying business rules, ensuring that the logic is centralized and reusable.

- Persistence Layer: Manages direct interactions with the database. This separation allows for easier maintenance and scalability, as changes to the database schema or access methods can be made independently of the business logic and presentation layers.

Use of Facade Pattern:

2. Simplified Communication:

- Facade Pattern: The use of the Facade pattern between layers simplifies the communication and interaction between them. It provides a unified interface to a set of interfaces in the subsystem, making it easier to interact with the system and reducing the complexity of the interactions.

- PresentationLayer --> BusinessLogicLayer: The Presentation Layer uses a facade to interact with the Business Logic Layer, ensuring that the presentation components do not need to know the details of the business logic implementation.

- BusinessLogicLayer --> PersistenceLayer: The Business Logic Layer uses a facade to interact with the Persistence Layer, ensuring that the business logic components do not need to know the details of the data access implementation.

Class Responsibilities:

3. Clear Responsibilities:

- UserService, PlaceService, ReviewService, AmenityService: These classes handle specific operations related to users, places, reviews, and amenities, respectively. They ensure that the presentation layer has a clear and focused responsibility for each type of operation.

- UserAPI, PlaceAPI, ReviewAPI, AmenityAPI: These classes expose endpoints for user, place, review, and amenity operations, respectively. They provide a clear interface for external interactions with the system.

- UserManager, PlaceManager, ReviewManager, AmenityManager: These classes handle the core business logic for user, place, review, and amenity operations, respectively. They ensure that the business logic is centralized and reusable.

- User, Place, Review, Amenity: These classes represent the entities in the system, encapsulating the data and behavior related to each entity.

- AuthenticationService, AuthorizationService: These classes manage user authentication and authorization, respectively, ensuring that security concerns are handled separately from the business logic and presentation layers.

- DatabaseAccessObject: This class handles direct interactions with the database, ensuring that data access concerns are separated from the business logic and presentation layers.

Consistent Styling:

4. Consistent Styling:

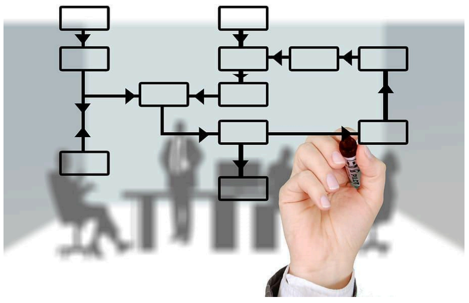
- Configuration: The diagram includes a configuration section that sets the theme and style variables, ensuring that the diagram has a consistent look and feel. This helps in maintaining visual coherence and readability.

How the diagram fits into overall architecture and design of HBnB

The package diagram provides a clear and concise representation of the HBnB system's architecture, highlighting the key components and their interactions. This high-level overview helps developers and stakeholders understand the system's structure and design, facilitating better development, maintenance, and scalability.

Detailed Class Diagram for the Business Logic Layer

Present the detailed class diagram ⬇️



Purpose of the diagram

………

Key components or classes involved

………

Design decisions and their rationale

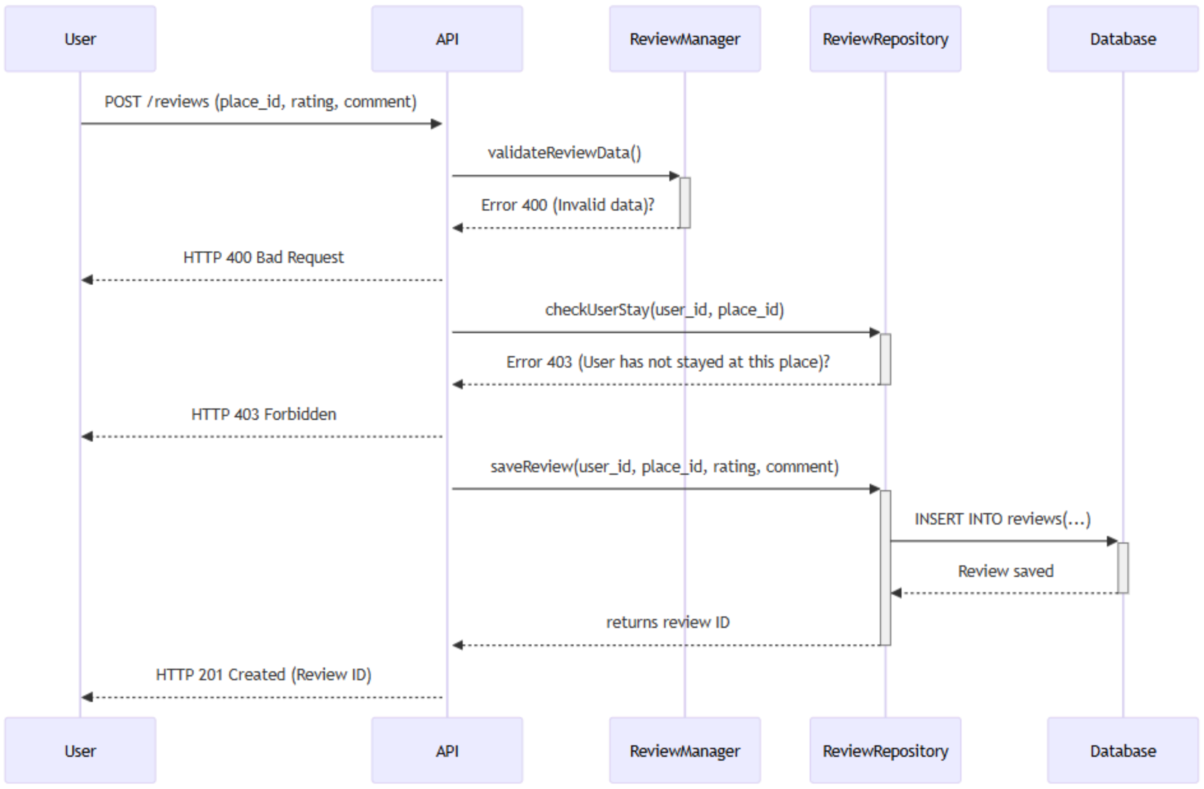
………

How the diagram fits into overall architecture and design of HBnB

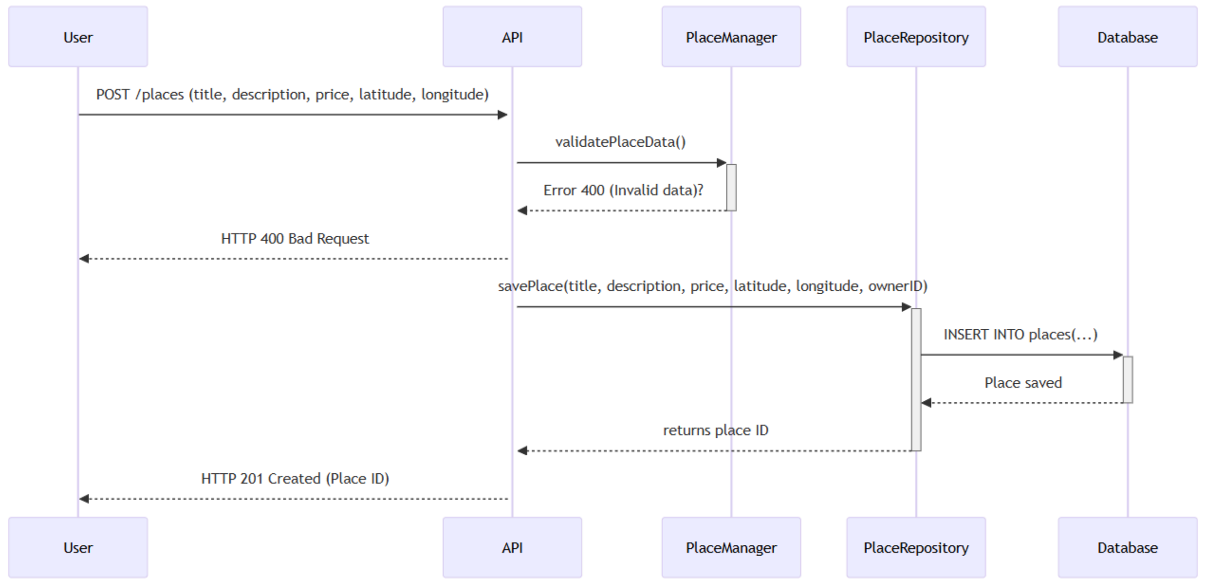
………

Sequence Diagrams for API Calls

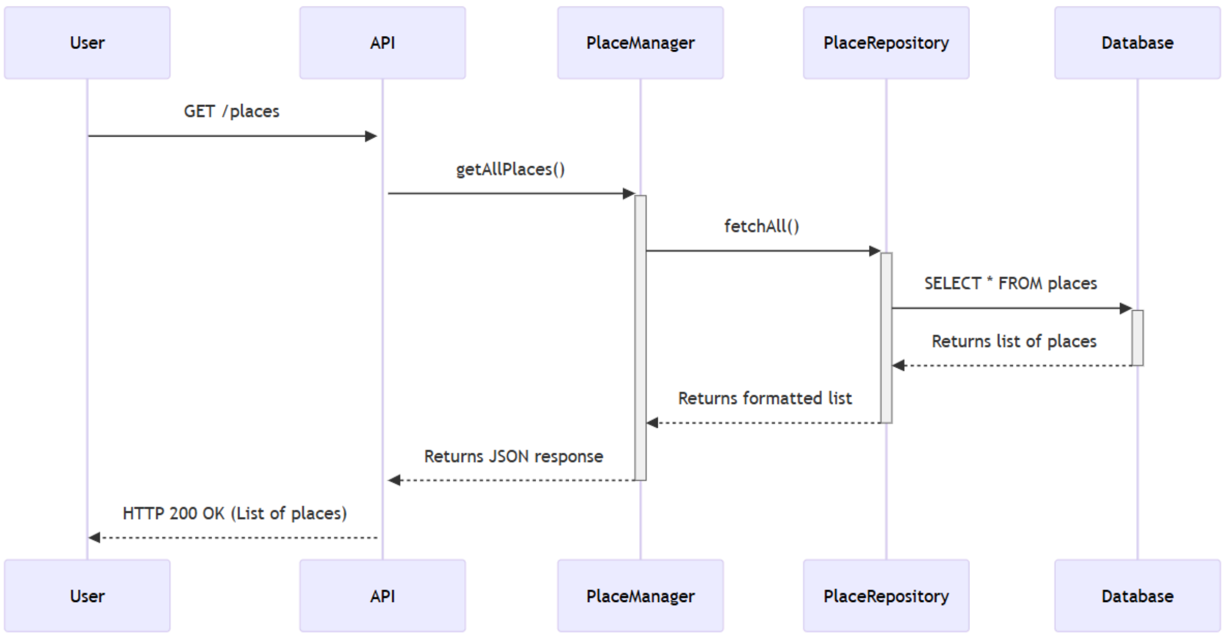
Review submission api calls


User
API
ReviewManager
ReviewRepository
Database
POST /reviews (place_id, rating, comment)
validateReviewData()
Error 400 (Invalid data)?
HTTP 400 Bad Request
checkUserStay(user_id, place_id)
Error 403 (User has not stayed at this place)?
HTTP 403 Forbidden
saveReview(user_id, place_id, rating, comment)
INSERT INTO reviews(...)
Review saved
returns review ID
HTTP 201 Created (Review ID)
User
API
ReviewManager
ReviewRepository
Database

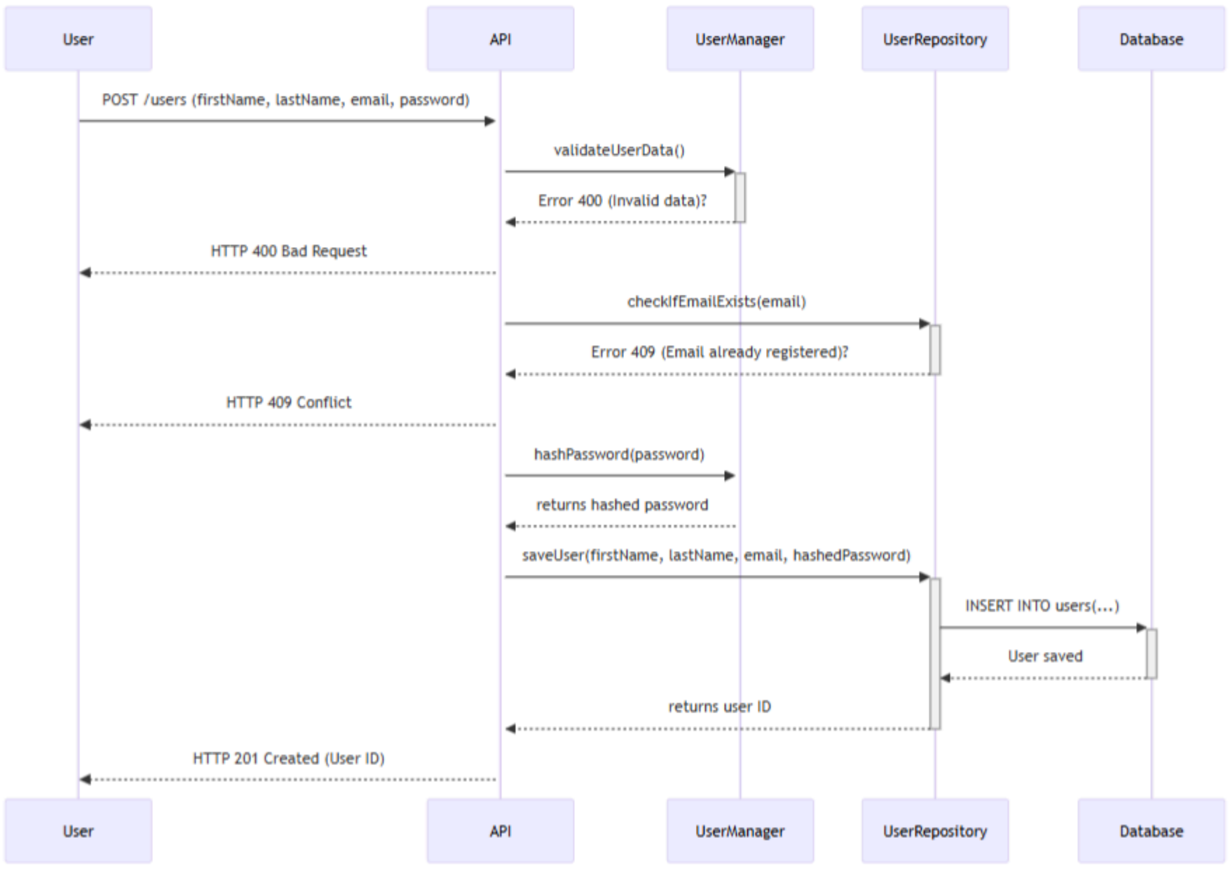
Creation place api calls


User
API
PlaceManager
PlaceRepository
Database
POST /places (title, description, price, latitude, longitude)
validatePlaceData()
Error 400 (Invalid data)?
HTTP 400 Bad Request
savePlace(title, description, price, latitude, longitude, ownerID)
INSERT INTO places(...)
Place saved
returns place ID
HTTP 201 Created (Place ID)
User
API
PlaceManager
PlaceRepository
Database

Places list fetching api calls



User registration api calls



Purpose of the diagram

The sequence diagram for API calls helps visualize how different layers of the architecture (Presentation, Business Logic, Persistence) interact when processing user requests.
Its main goals are to:
- Show the order of operations and data exchanges between components
- Identify validation steps and error handling (e.g., invalid requests, unauthorized access)
- Help developers understand how each API works internally
- Make maintenance and future updates easier with clear documentation

Key components or classes involved

Each sequence involves multiple components spread across three main layers:

Presentation Layer

- **User**: The user interacting with the API through a web or mobile interface.
- **API (Controller Layer)**: Receives and processes HTTP requests. Examples:
  - UserController
  - PlaceController
  - ReviewController

Business Logic Layer

- **Services**: Handle validation and business logic. Examples:
  - UserService
  - PlaceService
  - ReviewService
  - AmenityService
- **HBnBFacade**: A central entry point that groups all services together.

Persistence Layer

- **Repositories**: Manage database access. Examples:
  - UserRepository
  - PlaceRepository
  - ReviewRepository

Design decisions and their rationale

Separation into Three Layers

- Enhances scalability and maintainability.
- Allows modifications in one layer without affecting the others.
- Improves security by preventing direct access to the database.

Use of the Facade Pattern

- Centralizes access to functionalities and reduces coupling between services.
- Example: HBnBFacade allows UserService and PlaceService to interact without direct dependencies.

Detailed HTTP Error Handling

- **400 Bad Request**: Invalid data sent by the user.
- **401 Unauthorized**: User is not authenticated.
- **403 Forbidden**: Access denied (e.g., a user trying to modify a place they don't own).
- **404 Not Found**: Resource not found (e.g., non-existent place or user).
- **409 Conflict**: Data conflict (e.g., email already used during registration).
- **500 Internal Server Error**: Unexpected server-side error.

How the diagram fits into overall architecture and design of HBnB

- They show how each request is processed and how the layers communicate with each other.
- They complement class and package diagrams by illustrating the dynamic execution of interactions.
- They facilitate the implementation of REST APIs by clearly defining data flows and necessary validations.
- They help identify critical points, including error cases and potential optimizations for request handling.