Technical documentation for classes diagram

Overview:

HBnB is a web application inspired by Airbnb that allows users to publish, reserve, and manage rental places. The system is built around a set of classes that model the key entities of the application, their interactions, and the possible actions.

All classes share a common set of attributes:

- id: unique identifier (UUID v4)
- created_at: date and time of creation
- updated_at: date and time of last modification

Entity Descriptions:

- **Facade** (design patter)

The facade is a structural design pattern used to provide a simplified, unified interface to the complex subsystem composed of multiple classes (User, Place, Photo, Review, Amenity).

Role:

- Acts as an entry point to key system features (e.g., account management, place publishing, review handling)
- Hides the internal complexity of the application's class interactions
- Facilitates easier use of the system by external modules or API layers (e.g., front-end or REST endpoints)

Responsibilities:

- Orchestrates operations such as:
- Create/update/delete a user or place
- Upload and retrieve photos
- Post or moderate reviews
- Attach amenities to places
- Ensures appropriate access control logic (based on user roles) when invoking methods from underlying entities

Example Use Case:
When a client requests to "update a place and add two new photos", the Facade handles:

- Authorization (check if the user owns the place)
- Place update
- Photo validation and storage
- Linking photos to the place

This pattern contributes to separation of concerns and makes the system easier to test, extend, and maintain.

- **User** (Abstract)

Represents a generic user of the system.
This is an abstract base class, inherited by both Client and Admin.

Role:
Provides the basic interface for user interactions such as profile management and place handling.

- **Client** (inherits from User)

Represents a standard user of the platform.

Capabilities:
- Manage their own profile
- Add, update, and delete their own places
- Manage their profile picture
- Post reviews on places they visit
- Can book a place

- **Admin** (inherits from User)

Represents a privileged user with elevated rights.

Capabilities:
- Access and manage all user profiles and places
- Moderate or delete any content (except content from other admins)
- Shares all capabilities of a Client

- **Place**

Represents a rental property posted by a user.

Attributes:
- title, description, price, latitude, longitude

Relationships:
- Owned by a User
- Can be linked to multiple Photo, Review, and Amenity objects

- **Photo**

Handles images associated with users (profile picture) and places (galleries).

Role:
- Stores and provides access to the photo URL hosted on an external server
- Used only by User and Place entities

- **Review**

Represents a rating and comment posted by a user on a specific place.

Relationships:
- Associated with a User (author) and a Place (target)

Capabilities:
- Can be created, modified, or deleted
- Can be listed by user or by place

- **Amenity**

Represents the amenities or facilities available in a place (e.g., Wi-Fi, kitchen, pool).
Attributes:
- name, description

- **Booking**

Represents the booking possibilities for a place and allow users to create a reservation.

Role:

- Manage the place's calendar
- Allow users to place reservations
- Check the reservation status

Main Relationships:

- User ⟷ Place: One-to-many (a user can own multiple places or none)
- User ⟷ Photo: Zero-or-one-to-many (for profile picture)
- Place ⟷ Photo: Many-to-many (gallery)
- Place ⟷ Review: One-to-many
- User ⟷ Review: One-to-many
- Place ⟷ Amenity: Many-to-many

Persistence & Access Control:

All entities are stored with temporal tracking via created_at and updated_at attributes to facilitate versioning and auditing.

The model implements a clear separation of user roles (Client vs Admin) to ensure proper rights and access control.
The inheritance structure (User → Client / Admin) is central to managing permissions and system behavior.