Class Diagram

Introduction

The Class diagram presented in Fig. 1 illustrates the static structure of the HBnB Evolution application. This structure is characterized by classes and their relationships or interactions. The classes serve as a blueprint for the entities the application will create and the services it will use to manipulate them for the implementation of the business logic (treatment of the modeled system elements).

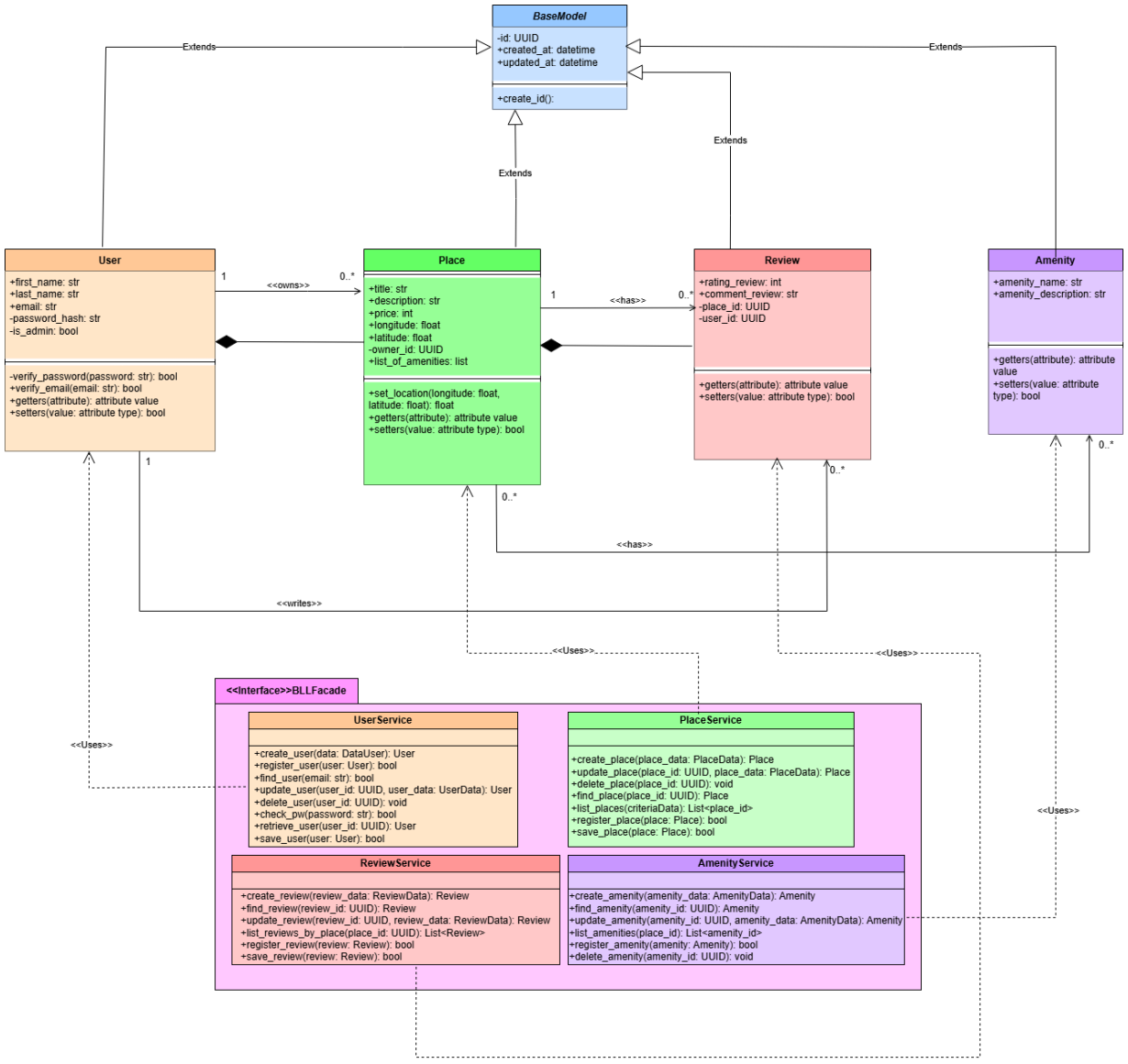

Fig. 1: HBnB Evolution class diagram. The parenthesis next to the attributes state the data type of that attribute. The parenthesis next to the methods state its arguments and their type following the format 'argument: argument type'. The return type of the methods is stated as: '-> return type'.

User

Represents a registered user of the application. Contains basic profile information,login credentials, and makes certain actions available to extract and modify such data.

- **Attributes**
  - **first_name (str)**: Defines the user's first name
  - **last_name (str)**: Defines the user's last name
  - **email (str)**: Defines the user's email address
  - **password_hash (str)**: Defines the user's password. The password is hashed for security reasons, ensuring it's not stored in plain text
  - **is_admin (bool)**: Defines whether the user is an administrator or not
- **Methods**
  - **verify_password(password: str) -> bool**: Verifies the validity of the provided password against the stored password hash. 'password' is a parameter representing the password to be verified
  - **verify_email(email: str) -> bool**: Verifies the validity of the email address. 'email' is a parameter representing the email address to be checked
  - **setters(value: attribute type) -> bool**: Sets corresponding attribute(s) to value and returns success/failure accordingly
  - **getters(attribute) -> attribute type:** Returns the value of the corresponding attribute

Place

Represents a registered location in the system. Each place has an owner and contains the basic information of the location. It also contains a list of the amenities available at that location.

- **Attributes**
  - **title (str)**: Defines the name of the place
  - **description (str)**: Provides a description of the location. A longer-than-usual string is expected for this attribute (this and all other string inputs will have an enforced max length determine case by case)
  - **price (int)**: Defines the price per night for this location. An integer input is enforced for this attribute
  - **longitude (float):** Defines the longitude coordinate of the place's location
  - **latitude (float):** Defines the latitude coordinate of the place's location
  - **owner_id (UUID)**: Defines the ID of the User object corresponding to the place's owner

    - **list_of_amenities (list)**: Defines a list of amenities available at the place

- **Methods**
    - **set_location(longitude: float, latitude: float)** : Sets the values of the longitude and latitude attributes of the Place. The parameters 'longitude' and 'latitude' are floats representing the location coordinates. This method doesn't return any value
    - **setters(value: attribute type) -> bool**: Sets corresponding attribute(s) to value and returns success/failure accordingly
    - **getters(attribute) -> attribute type**: Returns the value of the corresponding attribute

Amenity

Represents the various amenities associated with a place. Each amenity has a name and a description.

- **Attributes**
    - **amenity_name (str)**: Defines the name of the amenity
    - **amenity_description (str)**: Provides a description of the amenity. A longer-than-usual string is expected for this attribute
- **Methods**
    - **setters(value: attribute type) -> bool**: Sets corresponding attribute(s) to value and returns success/failure accordingly
    - **getters(attribute) -> attribute type**: Returns the value of the corresponding attribute

Review

Represents the comments that a user can leave regarding a place and which will be associated with that place. Each review includes a rating and a comment.

- **Attributes**
    - **rating_review (int)**: Provides a rating (given by a specific user) for a place
    - **comment_review (str)**: Provides a comment (given by a specific user) for a place. A longer-than-usual string is expected for this attribute
    - **place_id (UUID)**: Specifies the ID of the place receiving the comment
    - **user_id (UUID)**: Specifies the ID of the user giving the comment
- **Methods**
    - **setters(value: attribute type) -> bool**: Sets corresponding attribute(s) to value and returns success/failure accordingly

- **getters(attribute) -> attribute type**: Returns the value of the corresponding attribute

BaseModel

Represents the basic functionalities inherent to all entities.

- **Attributes**
  - **id (UUID)**: Defines a unique identifier for each entity upon creation
  - **created_at (datetime)**: Provides the creation timestamp of the entity
  - **updated_at (datetime)**: Provides the timestamp of the last modification done to the entity
- **Methods**
  - **create_id() -> UUID:** Generates a unique identifier and assigns it to the entity's 'id' attribute. This method takes no parameters and return no value

BLLFacade

Represents the Facade Pattern, which simplifies and centralizes the interaction of the Business Logic Layer (BLL) with the Presentation Layer (for the user-system interactions) and with the Persistence Layer (for system-database interactions).

- **UserService Methods**
  - **create_user(data: UserData) -> User**: Creates a new user. The 'UserData' parameter represents the necessary parameters expected for the creation of a new user. Returns the newly created 'User' object
  - **register_user(user: User) -> bool**: Communicates with the Persistence Layer. Returns Success/Failure
  - **find_user(email: str) -> UUID | None**: Find the user using their email address. The 'email' parameter is the email to be checked. Returns the user id if the user was found, return None/Null otherwise
  - **update_user(user_id: UUID, user_data: UserData)** -> User: Modifies parameters of a user's profile. 'User_id' allows locating the correct user object, and 'UserData' contains the new attribute values of the concerned user. Returns the updated 'User' object
  - **delete_user(user_id: UUID)**: Deletes the user corresponding to the provided ID. This method returns no value
  - **check_pw(password: str) -> bool :** Checks if the provided password matches the user's password. The 'password' parameter is the password to be checked
  - **retrieve_user(user_id: UUID) -> User:** Retrieve the user object from the Database
  - **save_user(user: User) :** Saves the User object to the database. Returns success/failure

- **PlaceService Methods**
    - **create_place(place_data: PlaceData)** : Allows creating a new place. The 'PlaceData' parameter represents the necessary parameters expected for the creation of a place. Returns the newly created 'Place' object
    - **update_place(place_id: UUID, place_data: PlaceData)** : Modifies parameters of a place. 'Place_id' allows locating the correct place, and 'PlaceData' allows associating the correct elements with the given ID. Returns the updated 'Place' object
    - **delete_place(place_id: UUID)** : Deletes the place corresponding to the provided ID. This method returns no value
    - **list_places(criteriaData)** : Lists all places satisfying given criteria. Returns a list of 'Place' objects IDs
    - **register_place(place: Place) :** Communicates with the Persistence Layer. Returns Success/Failure
    - **find_place(place_id: UUID) :** Finds the place corresponding to the provided ID. Returns the Place
    - **save_place(place: Place) :** Saves the place's object Place to the database

- **ReviewService Methods**
    - **create_review(review_data: ReviewData)** : Allows submitting a comment. The 'ReviewData' parameter helps locate the user ID and/or place ID to be commented on. Returns the newly created 'Review' object
    - **find_review(review_id: UUID)**: Finds a review with the given ID. Return the 'Review' object
    - **update_review(review_id: UUID, review_data: ReviewData)**: Modifies parameters of a review. 'Review_id' allows locating the correct review object, and 'ReviewData' contains the new attribute values of the concerned review. Returns the updated 'Review' object
    - **list_reviews_by_place(place_id: UUID)** : Allows listing comments by place. The 'place_id' parameter helps find the ID of the place whose comments are to be listed. Returns a list of 'Review' objects
    - **register_review(review: Review):** Communicates with the Persistence Layer. Returns Success/Failure
    - **save_review(review: Review) :** Saves the review's object to the database

- **AmenityService Methods**

- **create_amenity(amenity_data: AmenityData)** : Allows creating a new amenity. The 'AmenityData' parameter represents the parameters expected for the creation of a new amenity. Returns the newly created 'Amenity' object
- **find_amenity(amenity_id: UUID)**: Finds an amenity with the given ID. Returns the 'Amenity' object
- **update_amenity(amenity_id: UUID, amenity_data: AmenityData)**: Modifies parameters of an amenity. 'Amenity_id' allows locating the correct amenity object, and 'AmenityData' contains the new attribute values of the concerned amenity. Returns the updated 'Amenity' object
- **list_amenities(place_id)** : Lists all amenities associated with a place. Returns a list of amenity IDs
- **delete_amenity(amenity_id: UUID)** : Deletes the amenity corresponding to the provided ID. This method returns no value

Relationship between entities

Relationship between User, Place, Review, Amenity and BaseModel

All four main entities inherit attributes and methods from BaseModel. All entities are created with a unique identifier, and have attributes for their creation and last modification dates. Since these functions are the same for all entities, BaseModel allows writing the code once instead of repeating it in each entity.

Between User and Place

User can own zero or more places. User is independent of Place, it can exist even if no places are associated with it. Place depends on User, without an associated User a Place can not exist.
A place can only have one owner.

Between Place and Amenity

Place can have zero or more amenities. Amenity can be associated with zero or more places.

Between Place, User and Review

Place and User can have zero or more reviews. Review is associated with only one user and one place.