LACHIE KING
LIANI MCKEOWN
ULIANA DESHIN
ANTHONIA IFOEZE
HOLBERTON SCHOOL

Version 1.0
9TH Feb 2026

HBNB
UML DOCUMENTATION

TABLE OF CONTENTS

OVERVIEW

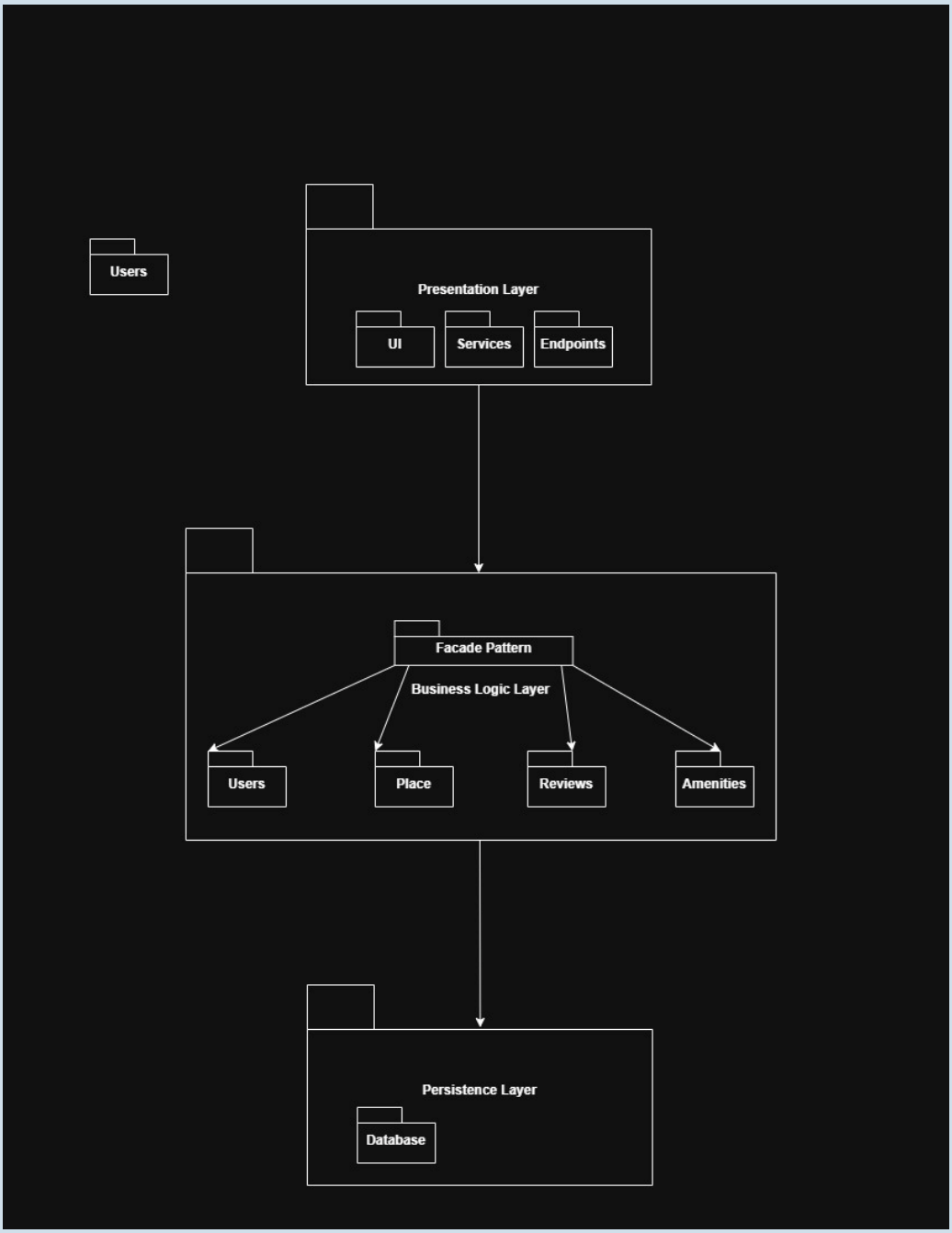
This technical document defines the architecture and design of the HBnB Evolution project. It describes the system structure, core components, and their interactions, serving as a blueprint that guides developers during implementation and ensures consistency across the application.

HBnB Evolution is a web-based property rental and management application that manages users, places, reviews, and amenities. The application follows a three-layer architecture with a Facade Pattern to ensure modularity, maintainability, and clear separation of concerns.

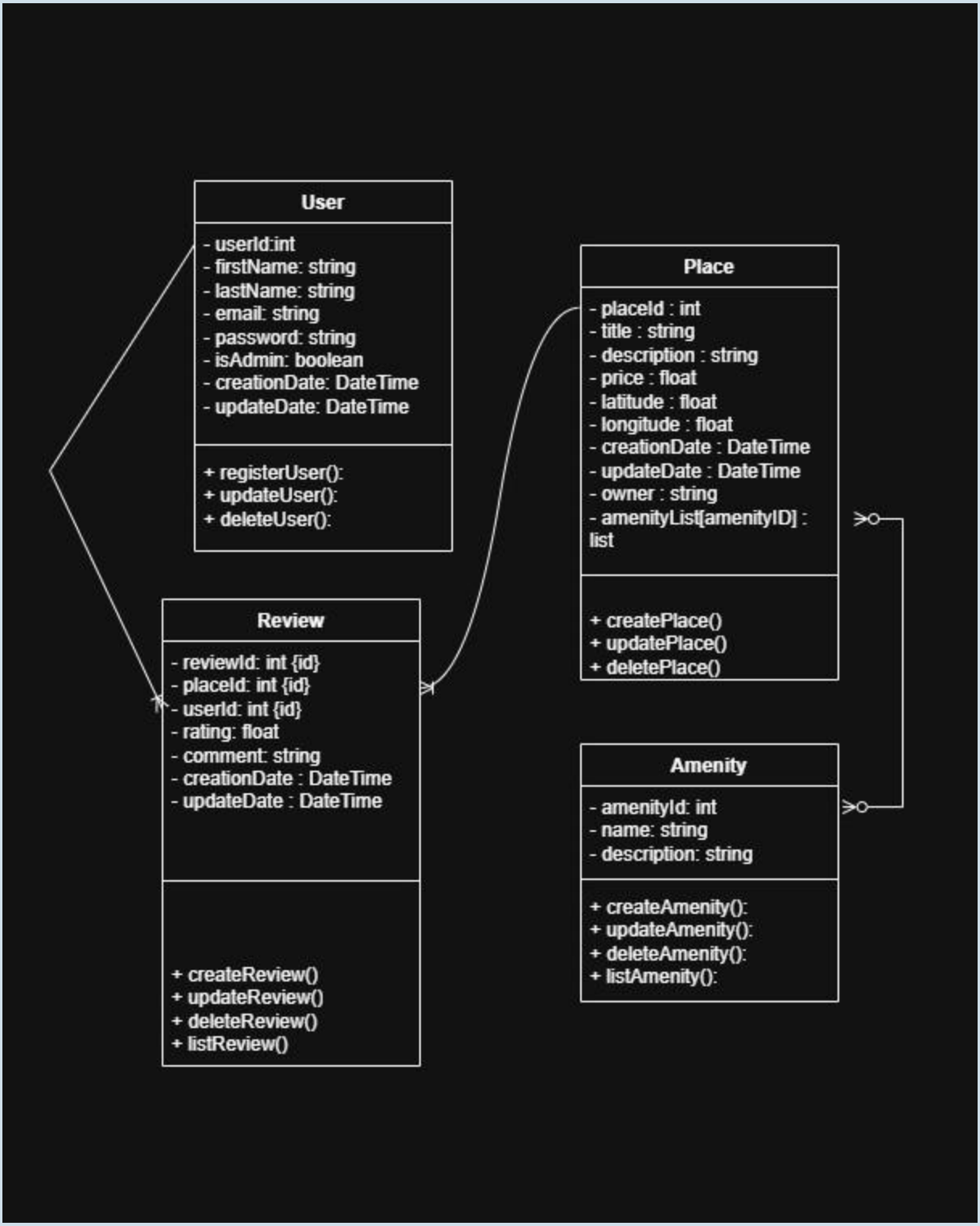
HIGH-LEVEL ARCHITECTURE

The diagram below illustrates the three-layer architecture of the HBnB Evolution application. The Presentation Layer (UI, services, endpoints) handles user interaction. The Business Logic Layer (BLL) contains the core domain logic, and the Persistence Layer is responsible for data storage. Each layer communicates only with adjacent layers; this constraint separates the responsibilities of each layer, which improves maintainability and testability.

Within the BLL, the Facade pattern provides a single, simplified entry point that coordinates complex interactions between the layers.



BUSINESS LOGIC LAYER CLASS DIAGRAM



USER

A User is an entity that has registered into the database, allowing them to perform actions such as creating a place, or leaving a review. Every User is provided with a unique ID which associates reviews with the correct user.

The regular attributes that are associated with user are first name, last name, email, and password. Which are all strings. We also have creationDate and updateDate which are DateTime attributes used to show when the most recent changes were made.

Each user has an is_admin boolean attribute to determine whether or not they are an admin, which will allow access through security checks into limited sections of the application.

The main 3 methods of any User within the application are registerUser(), updateUser(), and deleteUser(), providing freedom of creating an account, updating their information at any time, or deleting their account from the application.

PLACE

Place refers to the locations which registered users can create and sell on the application. A place must have the string values title, description and owner for users to view. It must also have a float value of its price for booking and its longitude and latitude for locational reference in an embedded Maps feature. As with all entities, every place must have a creationDate and updateDate to record the time of creation and/or change.

Every place must have a placeId for future reference; this is due to its relationship with the review entity, which must collect the placeId to determine to which place it is assigned. One place can have more than one review. An amenityList is also created from any amenityId associated with a place, which is then fetched when a user filters while fetching a list of places.

This entity's methods include createPlace(), which submits a new place to the database, updatePlace(), which submits changes to the place listing, and deletePlace(), which removes the listing from the database. All actions, upon successful save, are visible to registered and unregistered users.

REVIEW

A registered user can write and then submit a review of a place. Reviews are presented with a float rating from 0 to 5 and a comment. This is public for registered and unregistered users to view on its associated Place. There is a creationDate and updateDate to reference the date of its creation and any edits.

Every review has its own id for future reference, as well as an associated placeId for the place it is reviewing and the userId of the author.

Methods include createReview(), in which a registered user submits a review to be saved to the database and then presented to other users. Similarly, updateReview() submits a request to edit an existing review for public viewing. deleteReview() removes a chosen review from viewing, and the database, and listReview() allows all reviews associated with one Place to be viewable to registered and unregistered users.

AMENITY

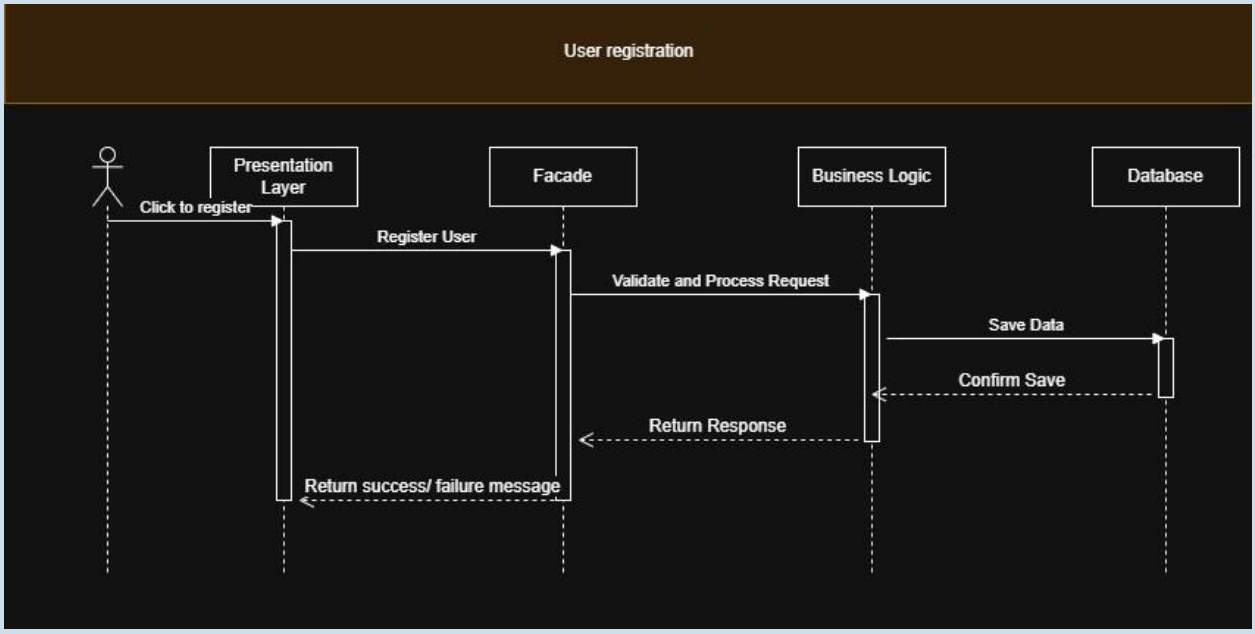
An Amenity is a feature or service that is associated with a place. These can be things like Wi-Fi, Air conditioning, swimming pools, or parking. Amenities allow for structure, helping users understand what is included in their stay, and allowing for better search and filtering capabilities.

Each amenity is provided with a unique ID, and is accompanied by a name and short description. Amenities can be linked to multiple places, and places can contain multiple amenities.

The main methods of an amenity are createAmenity() to create an Amenity, updateAmenity() to update the amenity, deleteAmenity() to delete the amenity or listAmenity() to list the amenities, which is primarily used within the place class..

API INTERACTION FLOW

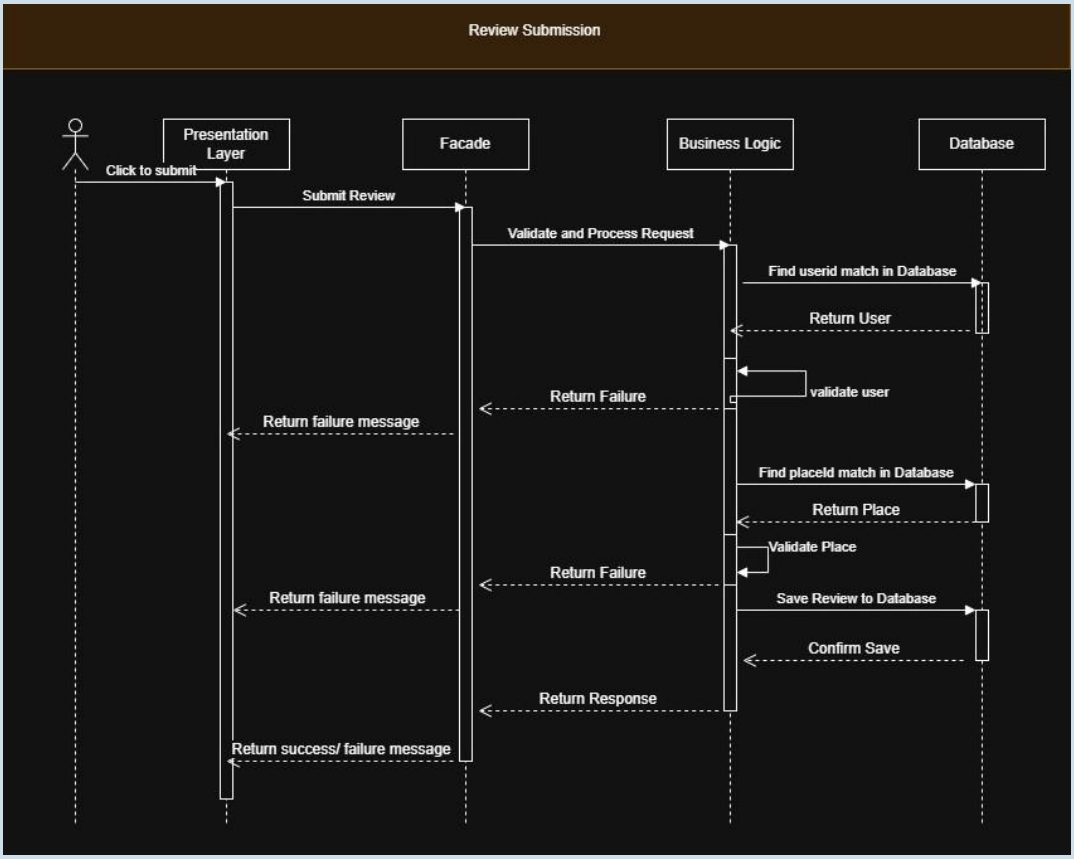
USER REGISTRATION



This sequence diagram illustrates the API call required for user registration. The user first clicks a button to register a new account. Then, through the presentation layer, the 'register user' action is created as a request, which the facade then sends to the Business Logic Layer (BLL) to validate and process. The BLL subsequently sends the user registration data to be saved in the database (i.e. the persistence layer).

From the database, the saved data is either confirmed or denied depending on whether certain criteria are met (e.g. a mandatory field is left empty). The BLL returns the response from the persistence layer to the facade, which then returns a success or failure message to the presentation layer for the user to view.

REVIEW SUBMISSION

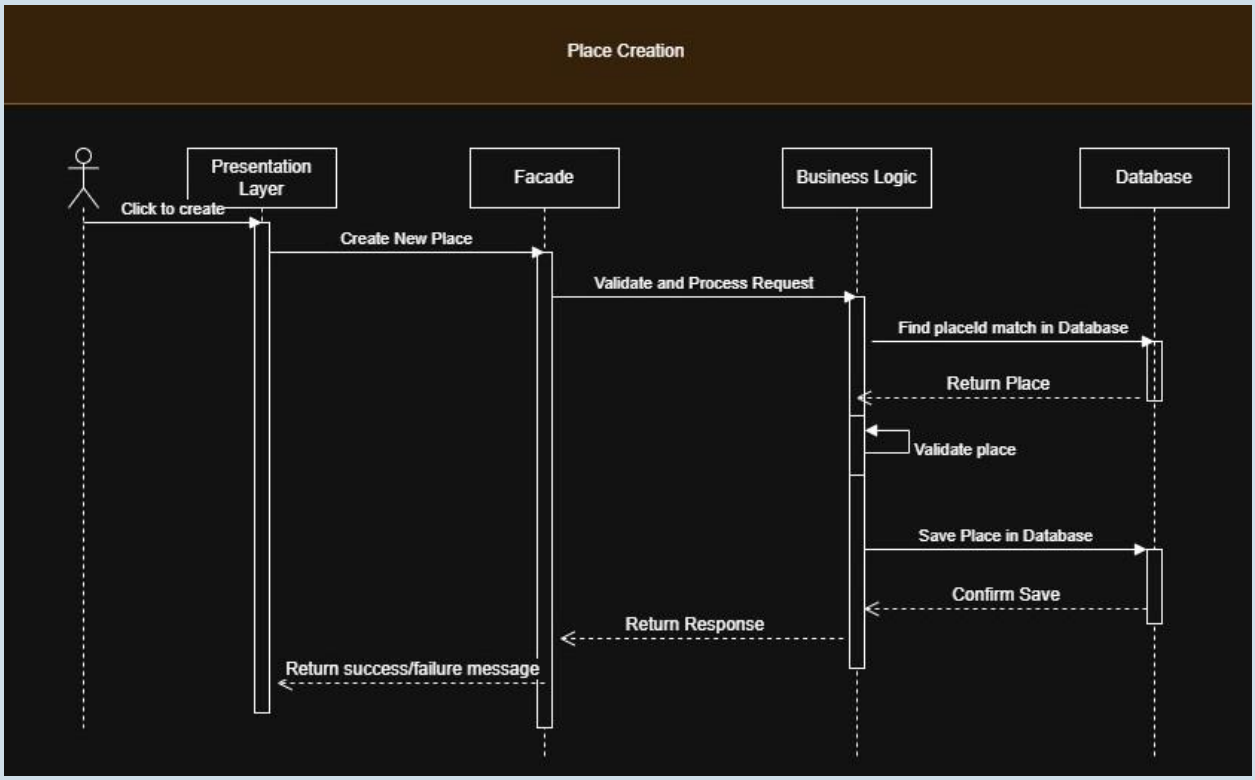


A User will start the process by clicking on the submit button on their review. The Presentation layer collects the information and creates a request for the Facade. The Facade will then send the request to the Business Logic Layer for validation and processing. The Business Logic layer will then send a check to the database to validate that the userId matches what is in the database. It will return the User to the Business Logic layer, which will then run a validation to check the User exists.

If the user does not exist in the database, an error message will be sent to the Facade, and then back to the presentation layer for display. If the user does exist, the Business logic layer will now continue through the process and repeat the same process for the place to see if it exists in the database. After the place validation, the business logic layer will again send a failure message back to the facade and then the presentation layer on failure. Otherwise, it will continue to the next step on success.

Once validation has been completed, the Business Logic Layer will send the information to the Persistence layer to be saved in the database. The Persistence layer will return a confirmation/failure to the business logic layer, which will be returned through the facade back to the presentation layer, showing a success/failure message to the user.

PLACE CREATION



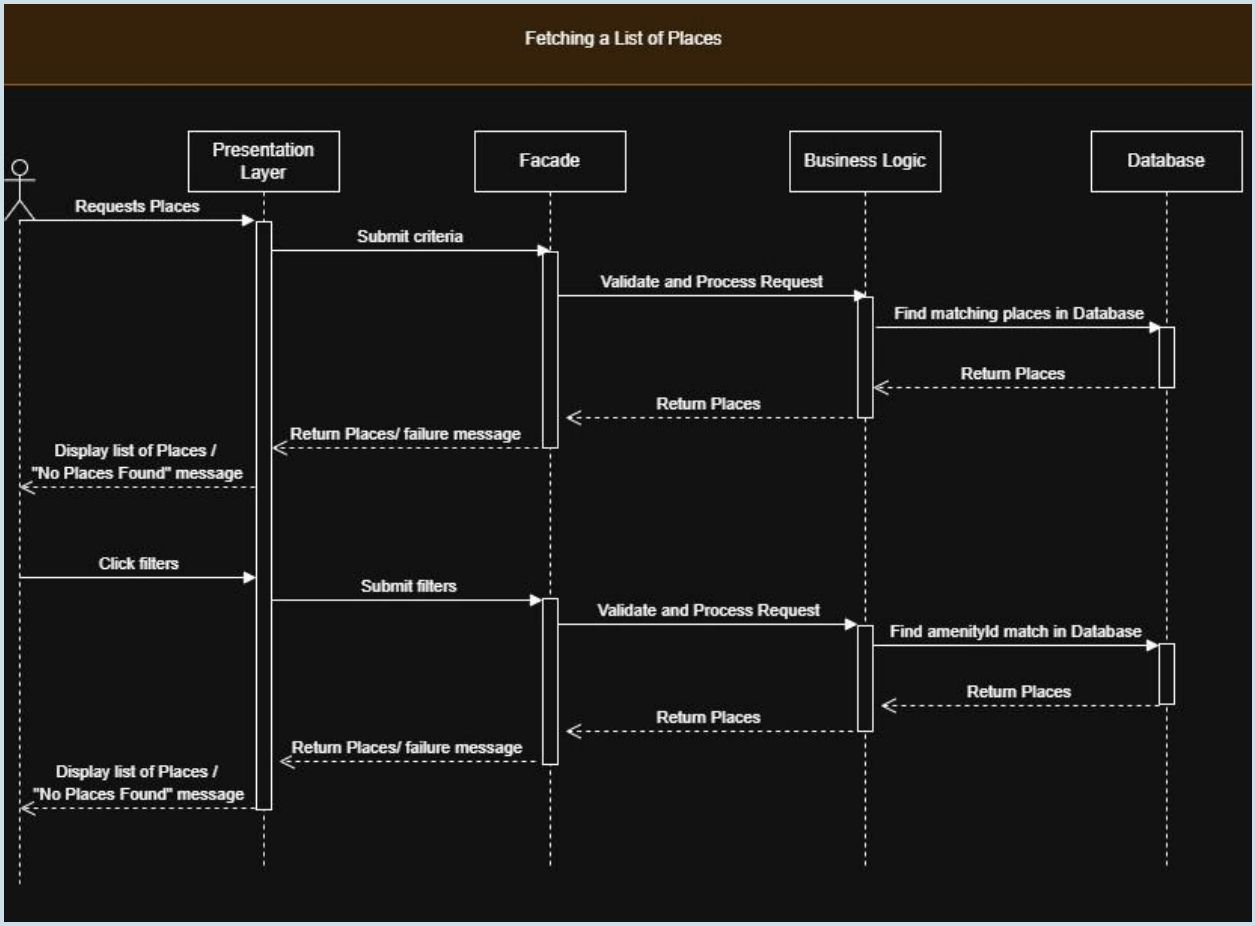
A user clicks a button to create a new place. The Presentation Layer collects the required information and creates a request to the Facade, which forwards the request to the Business Logic Layer (BLL) for validation and processing.

The BLL checks the persistence layer whether a matching place already exists in the database. The persistence layer returns the result of this check to the BLL. If a matching place is found or the validation fails, the BLL generates a failure response and sends it back to the Facade. The Facade then returns this failure message to the presentation layer.

If validation is successful and no matching place exists, the BLL proceeds by sending the new place data to the persistence layer to be saved in the database. The persistence layer confirms whether the save operation was successful or not.

The persistence layer's response is returned to the BLL, which forwards the outcome to the Facade. The Facade then sends a success or failure message back to the presentation layer, allowing the user to view the result of their action.

FETCHING A LIST OF PLACES



A user wants to view a list of Places based on certain criteria. The Presentation Layer will display fields for the user to make this initial request. This will not include any filter options, only Location and Check in/ Check out dates.

A user initiates a request through the Presentation Layer, which submits the search criteria to the Facade. The Facade delegates the request to the relevant handler within the BLL. The BLL validates and processes the request interacting with the Persistence Layer to retrieve matching Places from the database. The resulting list of Places/ or failure message is then sent back through the Facade to the Presentation Layer, where it is displayed to the user.

Once this initial list has been displayed to the user, the user can request that filters be applied to this list through the Presentation Layer which submits these filters to the Facade. The Facade delegates the request to the relevant handler within the BLL. The BLL validates and processes the request interacting with the Persistence Layer to retrieve matching Places with requested filters from the database. The resulting list of Places/ or failure message is then sent back through the Facade to the Presentation Layer, where it is displayed to the user.