

(HBNB



# HBnB UML

## Introduction

What is UML? UML stands for Unified Modeling Language. It's a standardized way to visualize the design of a system. Think of it as a blueprint for your code, much like architectural blueprints for buildings.

This Technical Documentation outlines the design and structure of a simplified Airbnb-like application, named HBnB Evolution which is a places rental platform. The purpose of this document is to provide a clear overview of the system architecture, including its key components, relationships, and functionalities. HBnB is designed to facilitate short-term property rentals by connecting property owners with potential guests. The system allows users to list properties with a list of amenities, search for accommodations and leave reviews.

## Scope

This document covers multiple aspects of HBnB's system design.

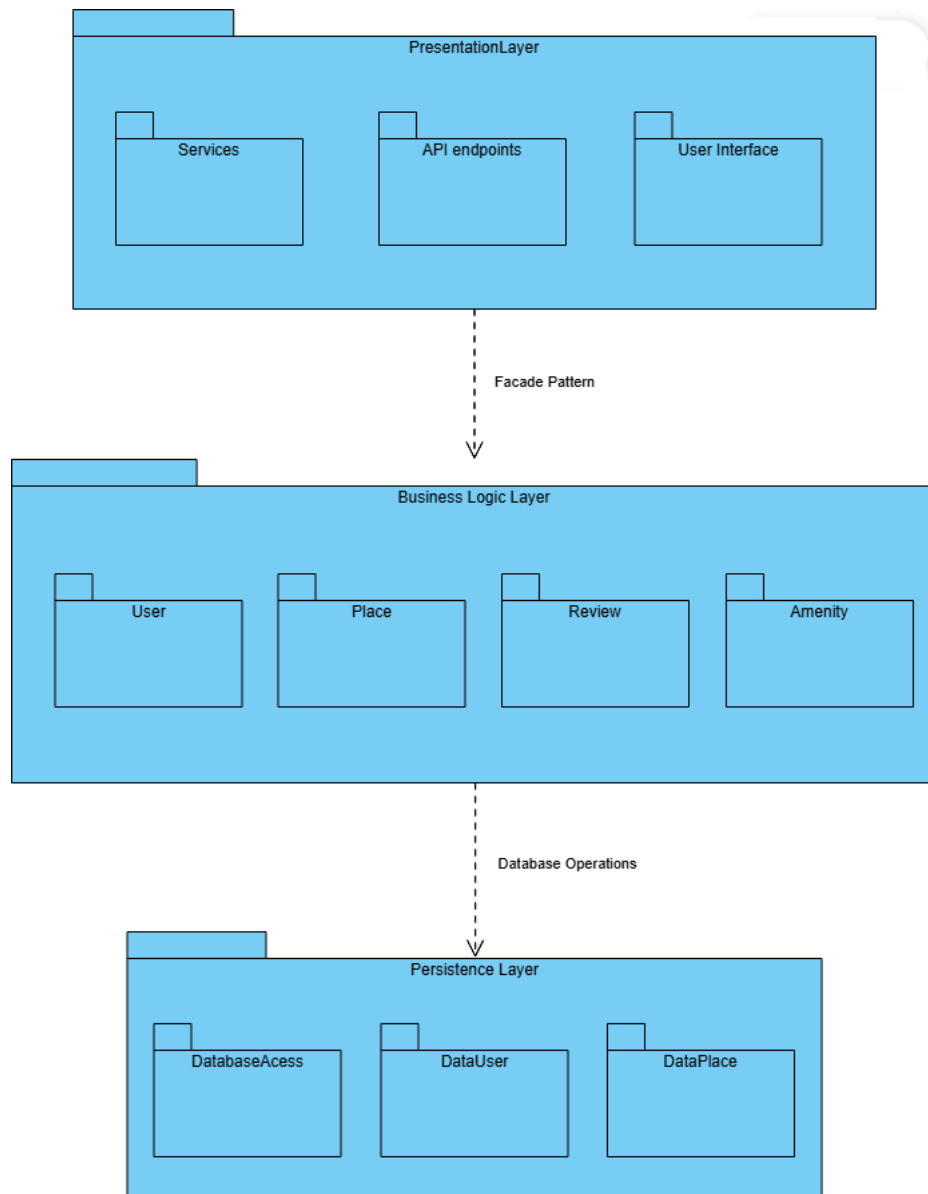
High-Level Package Diagram: illustrates the three-layer architecture of the HBnB application and the communication between these layers via the facade pattern.

Class Diagram: Represents the data model, relationships, and key operations for managing users, places, reviews, and amenities.

Sequence Diagrams for API: Illustrates the flow of requests and responses between system components, detailing key API interactions.

# High-Level Architecture Package Diagram

This diagram represents a three-layered architecture used in software design. It provides a conceptual overview of how the different components of the application are organized and how they interact with each other. It is divided into three layers that each have specific responsibilities and interacts with the layer below it.



## Legends

- Packages appear as rectangles with small tabs at the top,
- The package name is inside the rectangles,
- The dotted arrows are dependencies.

## **Presentation Layer (Top Layer)**

The Presentation Layer is responsible for managing user interactions and displaying information in a user-friendly manner. It communicates with the Business Logic Layer through well-defined API endpoints. This layer follows the Facade Pattern, which simplifies the interface for the UI by providing a cohesive entry point to the system's core functionality.

### **Services**

The services in this layer act as intermediaries between the UI and the backend. They process requests, validate input, and prepare data before sending it to the Business Logic Layer.

### **API Endpoints**

The API Endpoints define the communication interfaces between the frontend and the backend. These endpoints are responsible for handling requests, performing required logic through the Facade in the Business Logic Layer, and returning responses to the UI.

### **User Interface**

The User Interface (UI) is the visible and interactive component of the application, it is responsible for capturing user input, displaying system responses.

## **Business Logic Layer (Middle Layer)**

This layer acts as an intermediary between the Presentation Layer and the Persistence Layer, ensuring that business operations are executed. Business Logic Layer is responsible for enforcing business rules, managing transactions, and coordinating interactions between different components. It contains the core business logic and the models that represent the entities of the system.

### **User**

Handles user-related operations such as registration, authentication, profile updates, and role assignments, as well as security policies, including password encryption and role-based access control.

### **Place**

This class manages entities related to locations, properties or businesses. They are associated with the user who created them (owner) and can have a list of amenities. Handles Create, Read, Update, Delete operations for places.

### **Review**

Processes and validates user-generated reviews and ratings. Each review is associated with a specific place and user. Reviews can be created, deleted and listed by place.

### **Amenity**

Amenity handles all equipments available through all the places. Each has a name and description, can be created, updated, deleted and listed.

## **Persistence Layer (Bottom Layer)**

The Persistence Layer is responsible for managing data storage and retrieval operations within the system.

### **Database Access**

Handles direct interactions with the database, including executing queries, managing transactions, and ensuring efficient data retrieval.

### **DataUser**

Manages user-related data, including user profiles, authentication credentials, and role-based permissions.

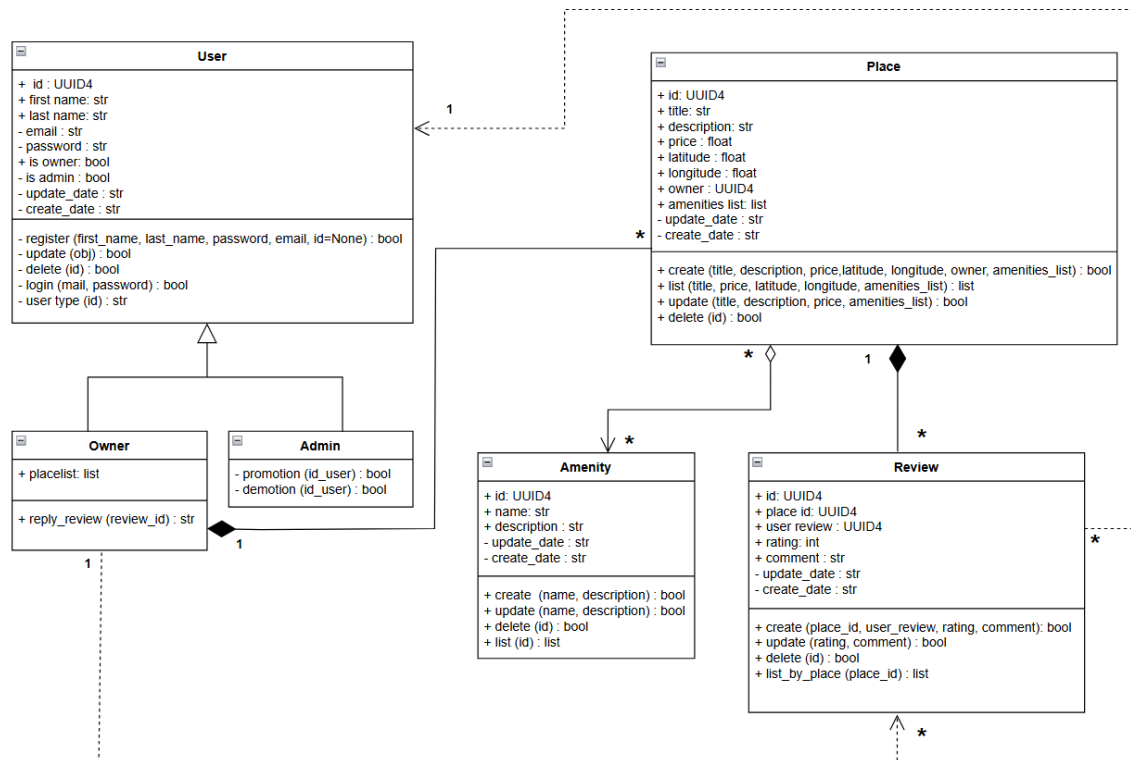
### **DataPlace**

Manages information related to places, including property details, locations, and ownership.

# Business Logic Layer Class Diagram

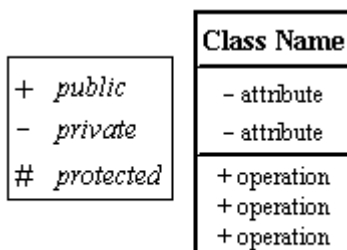
The diagram serves as a blueprint for the hbnb application, detailing the core classes, their attributes, methods, and interactions. It provides structure for managing users, places, reviews, and administrative functions.

Business Logic Layer provides a structured and scalable foundation for the hbnb application, ensuring efficient management of users, places, and reviews while maintaining security and user experience.









## Legends

Entity/Class: Each is represented by a rectangle with up to 3 compartments (class's name, attributes and operations/methods) which visibility can be public (+), private (-) or protected (#):



## Relationships:

Class Diagram Relationship Type	Notation
Association	
Inheritance	
Realization/ Implementation	
Dependency	
Aggregation	
Composition	

- **Association:** Represents a bi-directional relationship between two classes. It establishes a connection between objects of the two classes.
- **Inheritance/Generalization:** Represents an “is-a” relationship, also known as inheritance (ClassA is a subclass of ClassB).
- **Realization/Implementation:** Represents the relationship between a class and an interface. The class (usually concrete) realizes the operations declared by the interface.
- **Dependency:** Indicates that one class depends on another. This dependency can be due to a method parameter, local variable, or even due to the usage of an object as a method.
- **Aggregation:** Represents a “whole-part” relationship. It’s a weaker form of association where one class (the whole) contains objects of another class (the part), but the part can exist independently of the whole.
- **Composition:** A stronger form of aggregation. Represents a “whole-part” relationship where the part cannot exist without the whole. If the whole is destroyed, the part is destroyed as well.

**Multiplicity:** How many objects of each class take part in the relationships and multiplicity can be expressed as:

Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only $n$ (where $n > 1$ )
0..n	Zero to $n$ (where $n > 1$ )
1..n	One to $n$ (where $n > 1$ )



# Class User

The User class is a person registered with personal information.

Attributes	Type
+id	UUID4
+first_name	str
+last_name	str
-email	str
-password	str
-is_admin	bool
+is_owner	bool
-update_date	str
-create_date	str

Methods	Parameters	Return Type	Description
register	first_name, last_name, password, email, id	bool	create a new user
update	obj	bool	update already existing user info
delete	id	bool	delete a user
login	email, password	bool	retrive user data
user_type	id	str	check if user is owner and admin

## Subclass Owner

The Owner class inherits from the User class, Owner class is also dependet of Review class.

Attributes	Type
+placelist	list

Methods	Parameters	Return Type	Description
reply_review	review_id	str	enable user to reply to a review

## Subclass Admin

The Admin class inherits from the User class and have additional rights for actions on regular Users.

Methods	Parameters	Return Type	Description
promotion	id_user	bool	promote a user to admin
demotion	id_user	bool	demote an admin

# Class Place

The Place class is tied to the subclass Owner by Composition, which means that Place cannot exist without an Owner, therefore if an owner delete his account, their place will be deleted as well, they can't exist without an owner.

Place class is also aggregated with the class Amenity, they exist indepedently, amenity exists soly to provide a list to Place.

Attributes	Type
+id	UUID4
+title	str
+description	str
+price	float
+latitude	float
+longitude	float
+owner	UUID4
+amenities_list	list
-update_date	str
-create_date	str

Methods	Parameter	Return Type	Description
create	title, description, price, latitude, longitude, owner, amenities_list	bool	create a new place tied to an owner with a list of amenities
update	title, description, price, amenities_list	bool	update the infos of an already existing place
delete	id	bool	delete a place

# Class Amenity

Amenity class is also aggregated with the class Place, they exist independently, amenity exists solely to provide a list to Place.

Attributes	Type
+id	UUID4
+name	str
+description	str
-update_date	str
-create_date	str

Methods	Parameter	Return Type	Description
create	name, description	bool	create a new amenity
update	name, description	bool	update info of an amenity
delete	id	bool	delete an amenity
list	id	bool	lists the amenities available

## Class Review

The Review class is tied to the class Place by Composition, which means that Review cannot exist without a Place, therefore if a Place is deleted, their reviews will be deleted as well, they can't exist without a Place.

If an Owner deletes his account, it also deletes the reviews in his places since Place is tied to Owner by Composition

Review class is dependent of User, without a user there are no reviews.

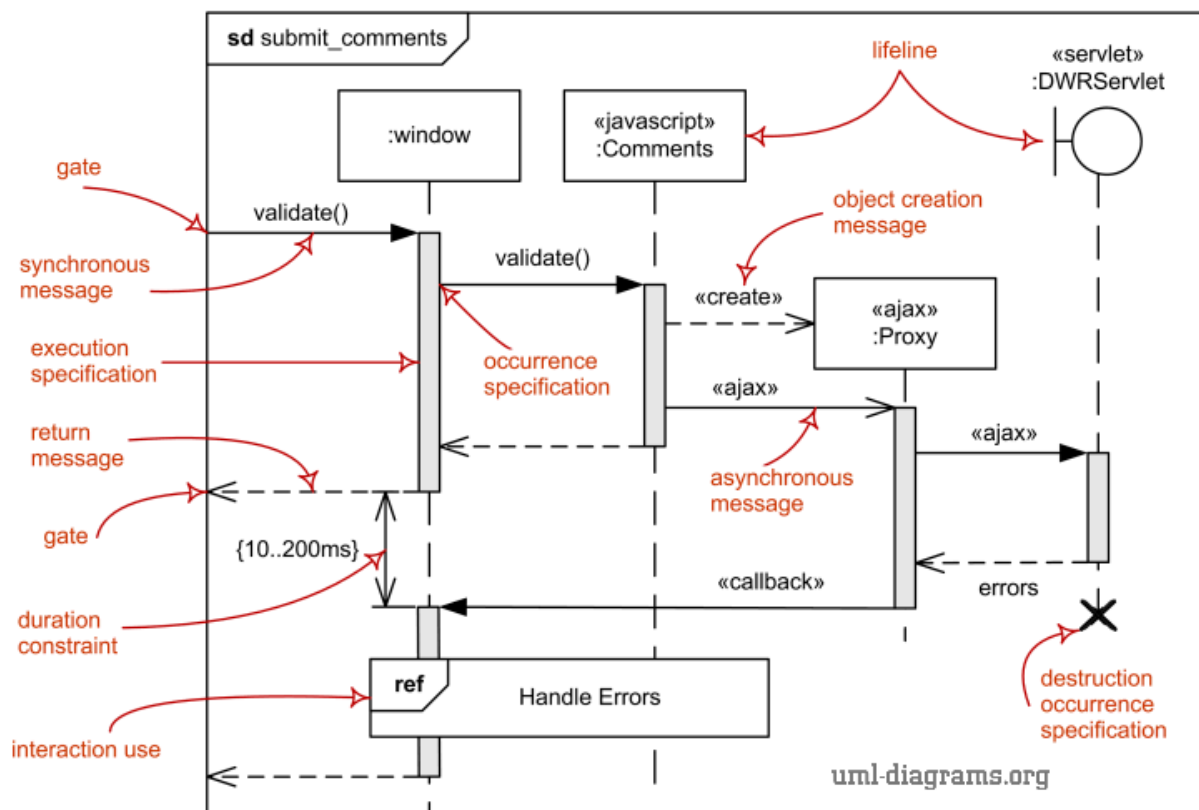
Attributes	Type
+id	UUID4
+place_id	UUID4
+user_review	UUID4
+rating	int
+comment	str
-update_date	str
-create_date	str

Methods	Parameter	Return Type	Description
create	place_id, user_review, rating, comment	bool	create a new review with a rating
update	rating, comment	bool	update the review and rating of an already existing review
delete	id	bool	delete a review
list_by_place	place_id	list	lists the reviews and rating of a place

# API Interaction Flow Sequence diagrams

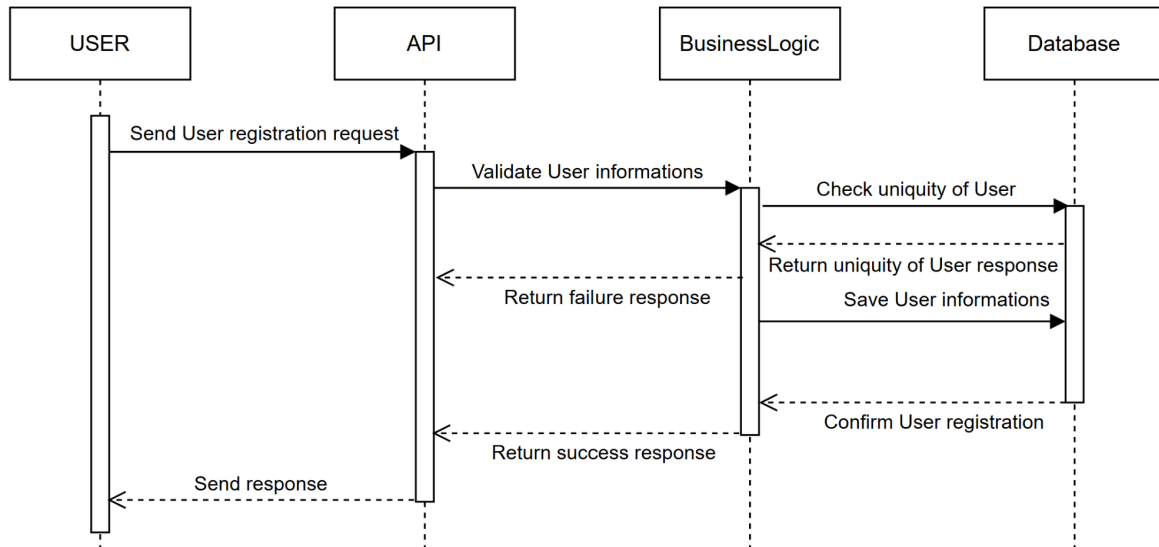
Sequence diagrams illustrate the interaction between the layers (Presentation, Business Logic, Persistence) and the flow of information within the HBnB application. The sequence diagrams will help visualize how different components of the system interact to fulfill specific use cases, showing the step-by-step process of handling API requests. Below are 4 API calls with their sequence diagram associated.

## Legends



# User Registration

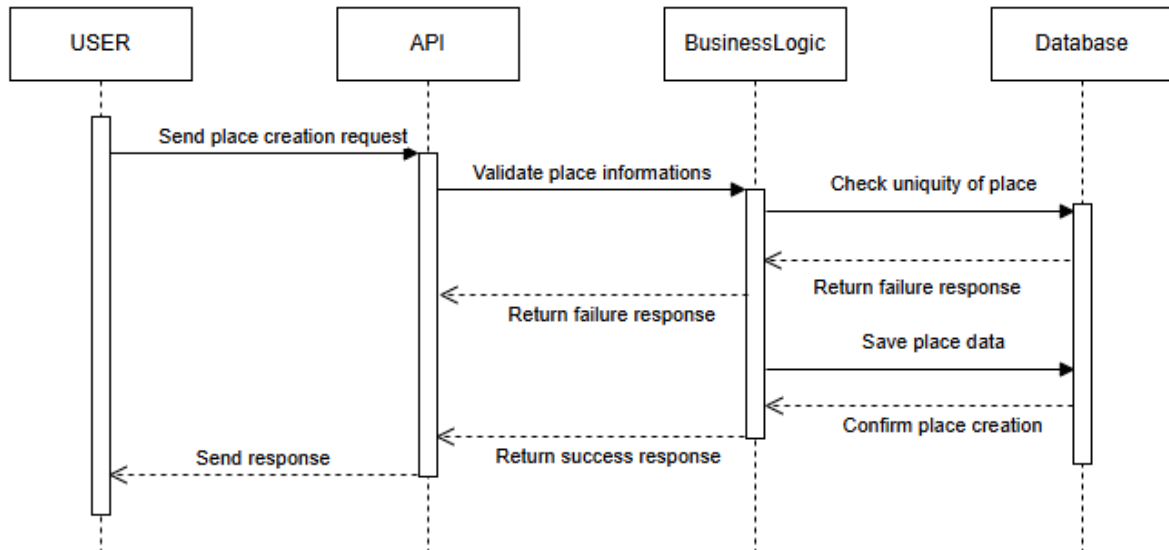
A User signs up for a new account. If the Business Logic Layer detects an issue, it sends a failure response to the API, which then informs the User of the issue.



Component	Role
User	Initiates registration request
API	Acts as an intermediary, forwarding requests and responses
Business Logic	Validates, processes, and coordinates registration logic
Database	Stores user information upon successful validation

# Place Creation

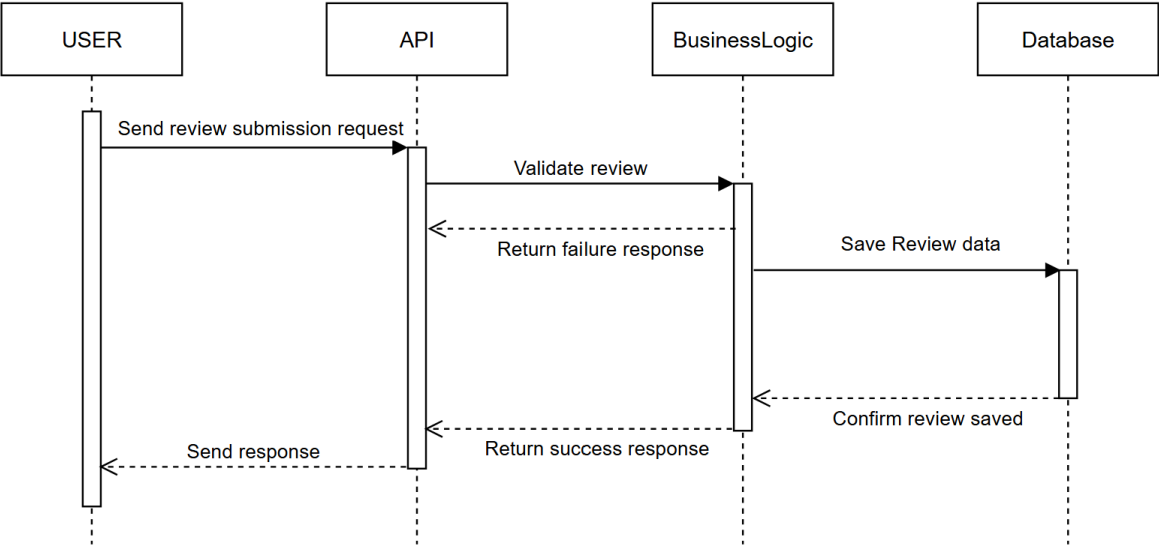
A user creates a new place listing. If the Business Logic Layer detects an issue, it sends a failure response to the API, which then informs the User of the issue.



Component	Role
User	Initiates place creation request
API	Acts as an intermediary, forwarding requests and responses
Business Logic	Validates, processes, and coordinates place creation logic
Database	Stores place information upon successful validation

# Review Submission

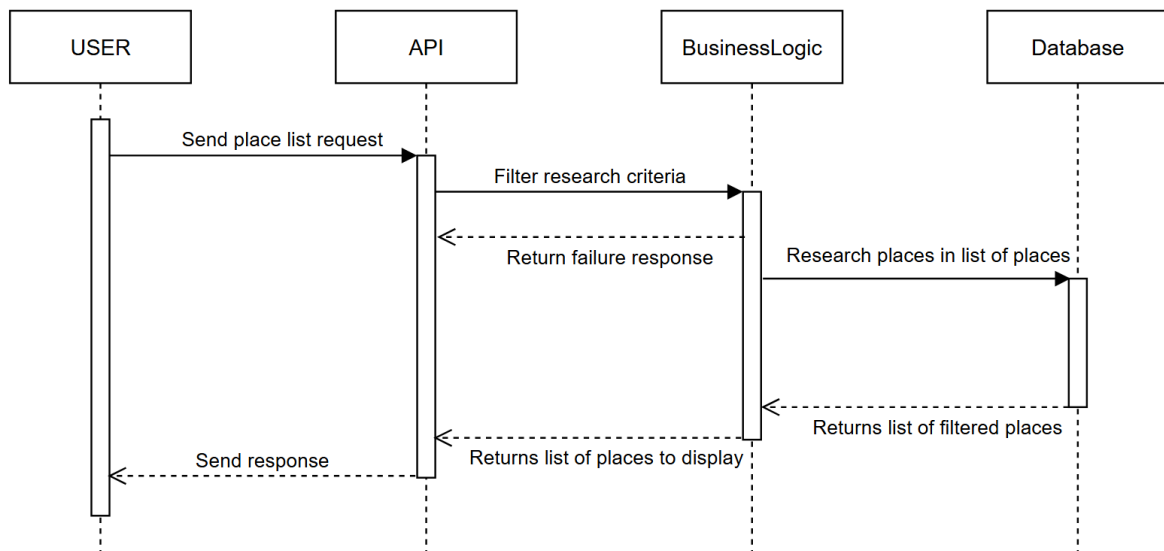
A user submits a review for a place. If the Business Logic Layer detects an issue, it sends a failure response to the API, which then informs the User of the issue.



Component	Role
User	Initiates review submission request
API	Acts as an intermediary, forwarding requests and responses
Business Logic	Validates, processes, and coordinates review submission logic
Database	Stores review information upon successful validation

## Fetching a List of Places

A user requests a list of places based on certain criteria. If the Business Logic Layer detects an issue, it sends a failure response to the API, which then informs the User of the issue.



Component	Role
User	Initiates list of places request
API	Acts as an intermediary, forwarding requests and responses
Business Logic	Validates requests and apply filter
Database	Fetch places based on criteria and return place list

## Authors

- [Tra Mi NGUYEN](#)
- [Tom DIBELLONIO](#)
- [Raphael DOTT](#)