**HBNB Technical Document**

Summary:

Introduction

Step 0

Step 1

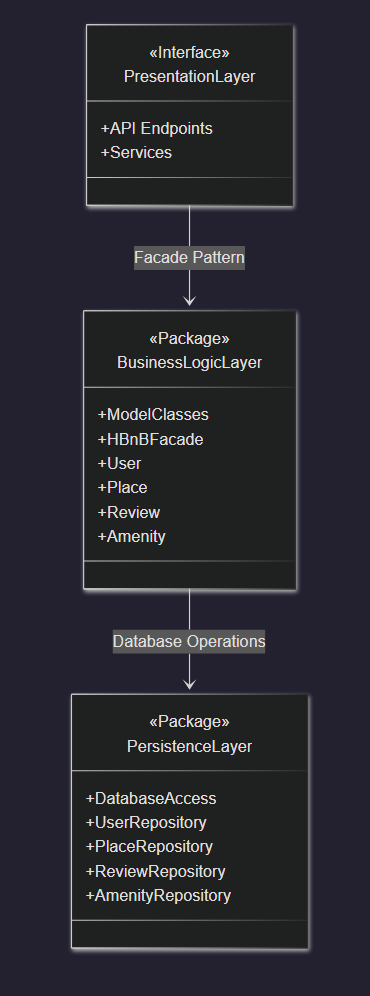
Step 2

**Introduction:**

This is the Technical document of the first part of the HBNB Project. In this document we are reviewing the differents part of the architecture and the differents layers of the project using differents diagrams

**Step 0:**

We created a high-level diagram which illustrates the three-layer architecture that will be used for the creation of the HBnB application. The three layers are: Presentation Layer, Business Logic Layer and Persistence Layer.



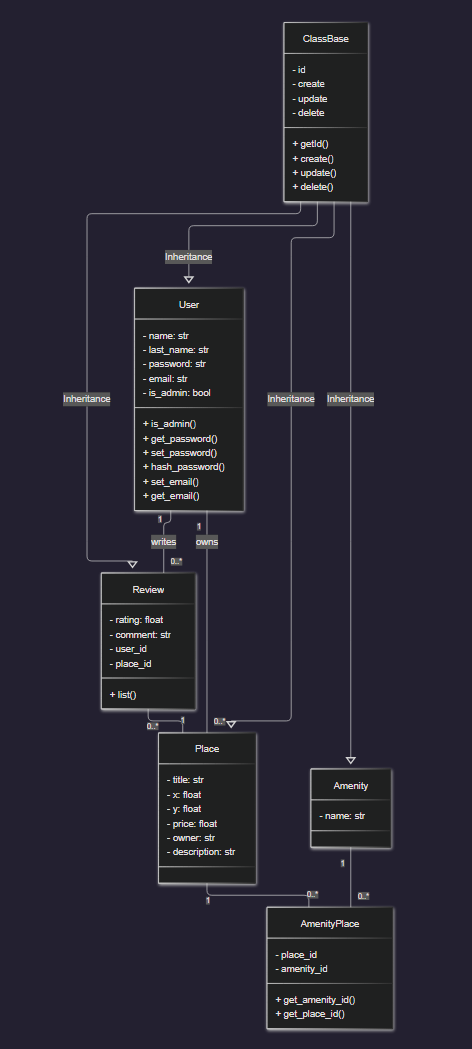
The Presentation Layer is the front of the application, this is the interface between the application and the user. In this layer there are services for user requests and API’s for endpoints and network communication.  
The Business Logic Layer is for the application management rules, it has models (User, Place, Review, Amenity) and the logic.

The Persistence Layer gets and stocks data with DAO (Data Access Objects) or repositories (for request to the database).

*Explanation:*

The Presentation layer calls the facade pattern. The facade pattern calls the business layer and the persistence layer, which means that the business and the persistence layer never communicate directly with the presentation layer.

**Step 1: Detailed Class Diagram**

After creating the high level diagram we started working on the detailed class diagram,this class diagram represents the main components of the system, showing how users, places, reviews, and amenities interact. Together they define the application’s core business logic for user management, place listings, amenities, and reviews.

**ClassBase**

**Role:**

ClassBase is a parent class providing shared attributes and methods across every class

**Key Attributes:**

* **ID:**  Unique identifier for each object.

**Key Methods:**

* **Get() :**  Retrieve an instance by ID.
* **Create() :**  Create new data.
* **Update() :** Update existing data.
* **Delete() :** Remove data.

### **User**

**Role:**Represents a user in the system. A user can own places and write reviews. Administrators are identified by a boolean flag.

**Key Attributes:**

* **name, last\_name, email, password :** Basic user information.
* **Is\_admin :** Determines whether the user is admin or not.

**Key Methods:**

* **is\_admin():** Checks admin rights.
* **get\_password() / set\_password() / hash\_password():** Manage and secure password.
* **get\_email() / set\_email()**: Access and modify email data.

**Relationships:**

* One user **owns** multiple places.
* One user **writes** multiple reviews.

**Place**

**Role:**Represents a property or location listed by a user. Each place is owned by a user and can have associated reviews and amenities.

**Key Attributes:**

* **title:** Name of the place.
* **x, y:** Geographic coordinates (latitude and longitude).
* **price:** Cost of the rent.
* **description:** Details about the place.
* **owner:** Reference to the user who created it.

**Relationships:**

* Each place is **owned by one user**.
* A place **can have multiple reviews**.
* A place **can have multiple amenities**

**Review**

**Role:**Represents feedback left by a user for a specific place. Reviews improve trust and quality transparency for other users

**Key Attributes:**

* **rating:** Rating score.
* **comment:** User comment.
* **user\_id:** ID of the reviewing user.
* **place\_id:** ID of the reviewed place.

**Key Methods:**

* **list():** Retrieve reviews.

**Relationships:**

* A review **belongs to one user**.
* A review **belongs to one place**.

### **Amenity**

**Role:**List all the possible amenities available in a place.

**Key Attributes:**

* name: Amenity name.

**Relationships:**

* Amenities are linked to places with the AmenityPlace class because it’s a many-to-many relationship.

### **AmenityPlace**

**Role:**Acts as an association class that connects places and amenities, enabling many-to-many relationships.

**Key Attributes:**

* **place\_id:** Reference to a place.
* **amenity\_id:** Reference to an amenity.

**Key Methods:**

* **get\_place\_id():** Retrieve the linked place ID.
* **get\_amenity\_id():** Retrieve the linked amenity ID.

**Relationships:**

* Connects each Place to multiple Amenity objects.

**Relationship Overview and Business Logic**

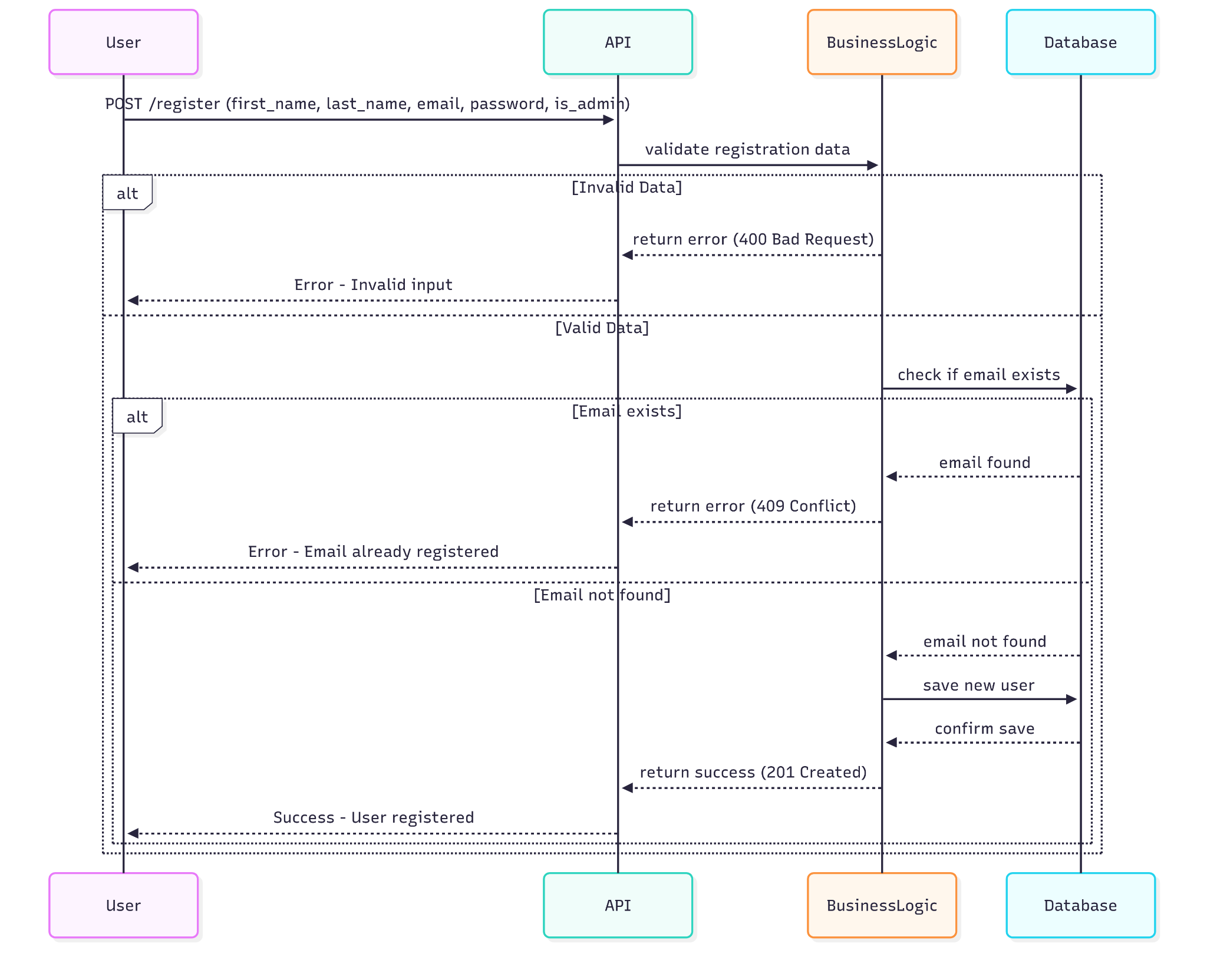
* **Inheritance (Generalization):**All class inherit from ClassBase, this give every other class the CRUD method to every other class
* **User–Place (one to many):** A user can create multiple places. This represents ownership of listings.
* **User–Review (one to many):** A user can write multiple reviews, but each review belongs to one user.
* **Place–Review (one to many):** Each place can receive multiple reviews from different users.
* **Place–Amenity (many to many):** A place can have many amenities, and an amenity can belong to multiple places.

**Step 2:**

Sequences diagrams illustrate the communications between layers and operations required for the process.

Note that at the end of the request the database can have a time out error which means that it doesn't give a return fast enough so an error goes through all the layers.

**Sequence diagram: User Registration**



First the user makes a request to create an account with a given first name, email, password…

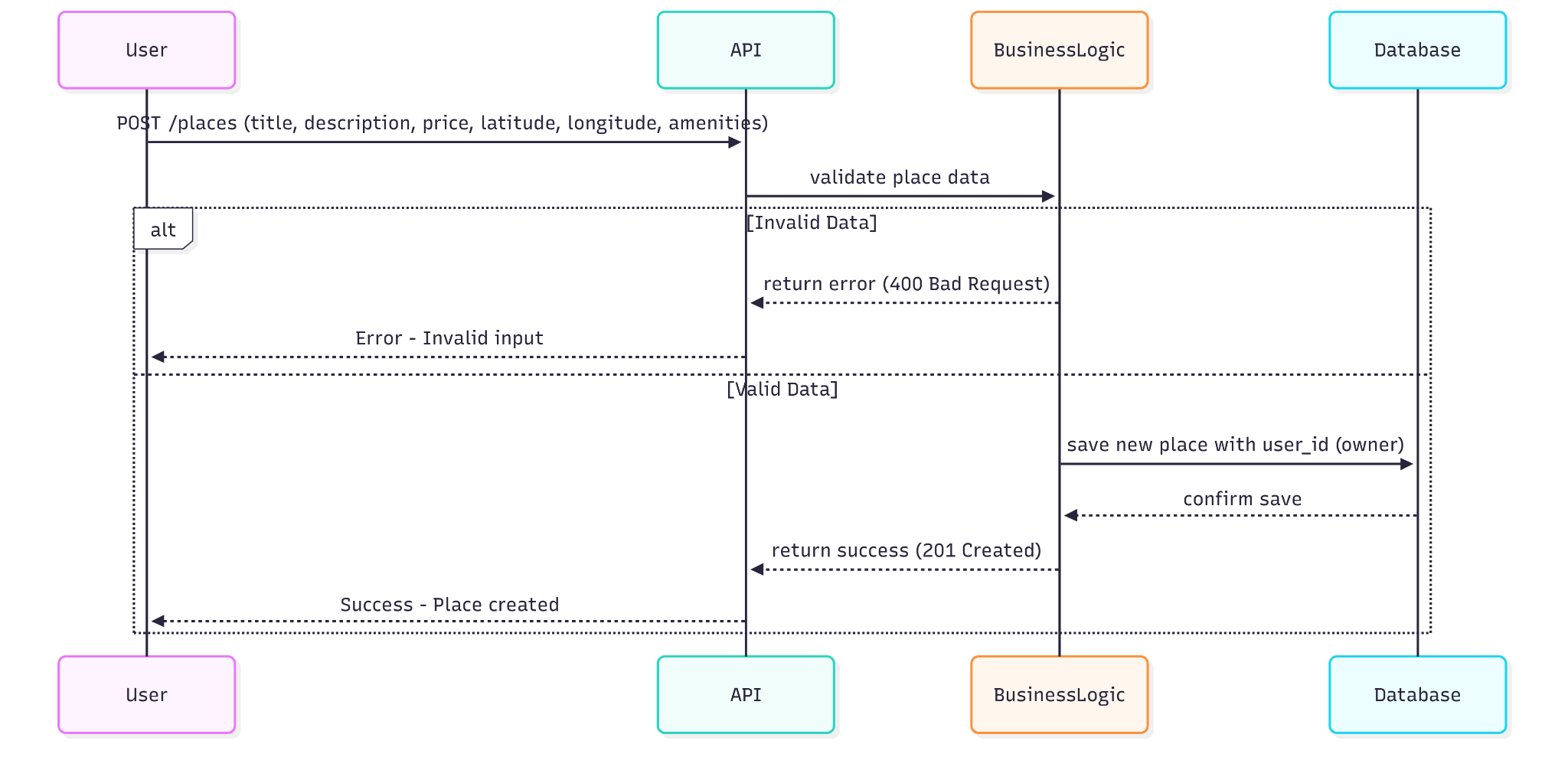
An API request verifies if the input is valid.

If it's invalid the business logic layer returns an error to the API which returns an error with a message invalid input.

If the input is validated we search in the database if the email already exists, if it exists the database returns an error to the business logic layer which returns an error with a message “email already registered”.

If the database doesn't have the email, it hashes the password and saves the new user with his information and confirms the save, the business logic layer returns a success to the API that returns to the user the success.

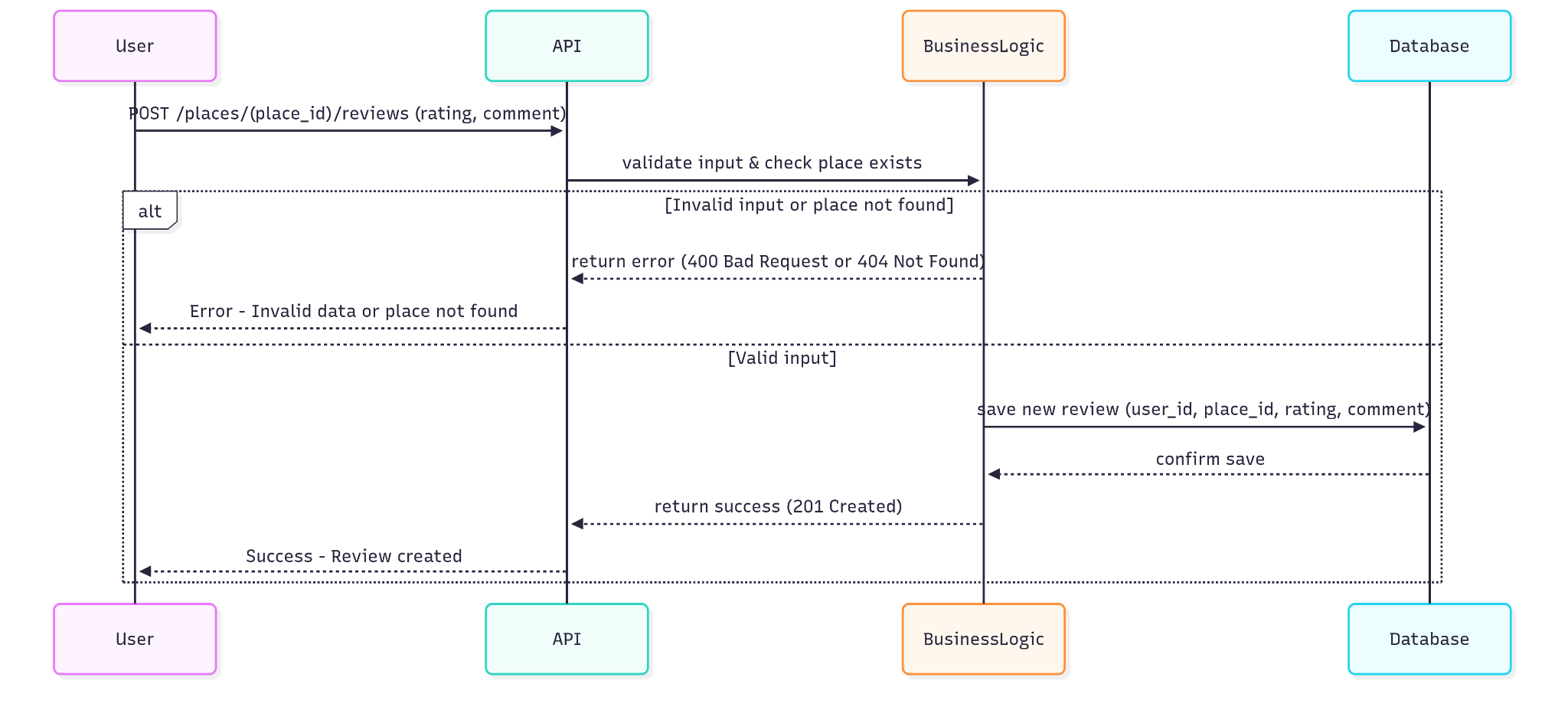
**Sequence Diagram: Place Creation**



First a user makes a request to register a new place with given information (title, description, price…). The API makes a request to validate data, if they are invalid it returns an error and the API returns an error to the facade pattern.

If the data’s are valid, a new place is saved with the user ID in the database, then a success return that goes through every layer.

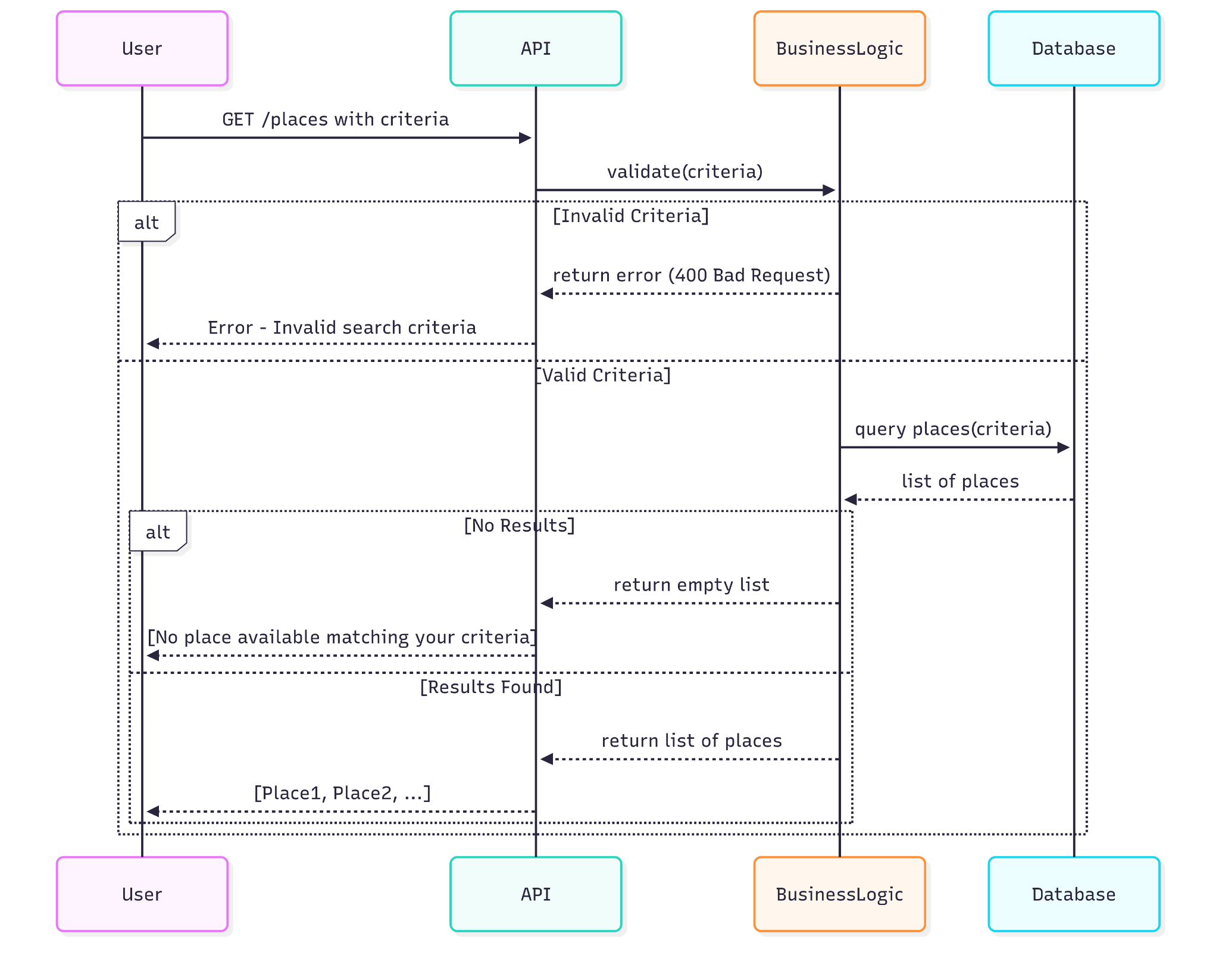
**Sequence Diagram: Review Submission**



A request for posting a review is made by a sign in user with a rating and a comment on the place to review.

An API request goes to verify data’s, if they are invalid then an error return is sent through the layer's, if they are valid a new review is saved in the database linked with the user ID that makes the review and the place ID that is reviewed. A success return is sent.

**Sequence Diagram: Fetching a List of Places**



A research is made by a user with criteria(s) (location, specific amenity…).

The API request for a research with the selected criteria.

Then the database query places with criteria and returns to the business logic layer a list of places that matched.

If no place matches with the user's criteria an empty list is returned, if some places matched the user's criteria then a list of places are shown to the user.

Authors: Théa PROLONGEAU, Dorian OUFER