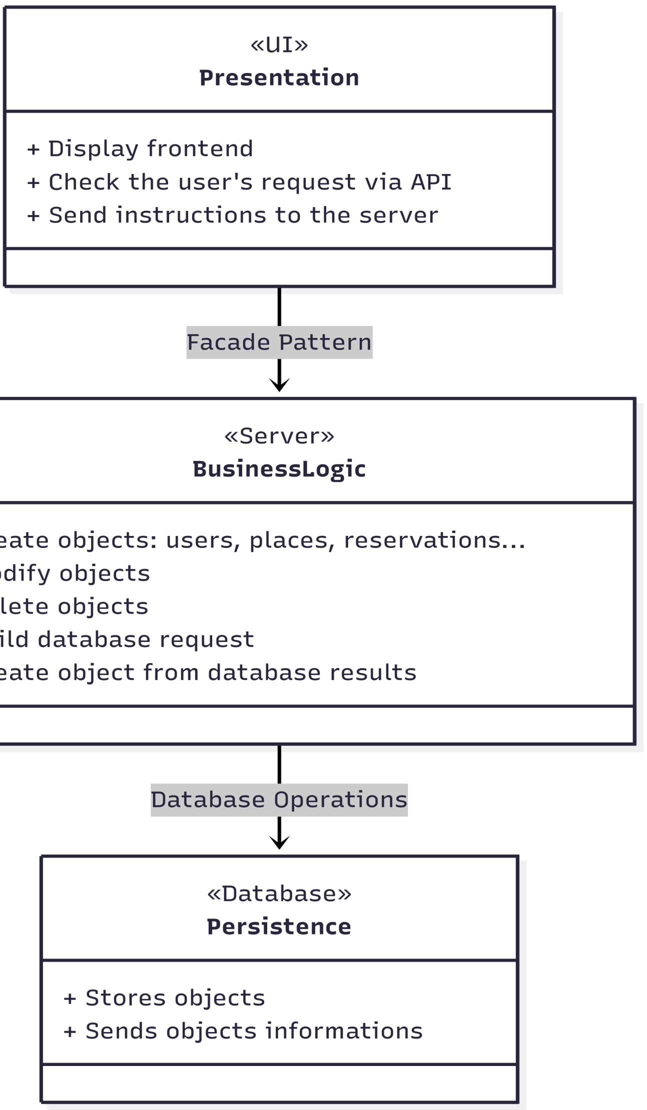
EXPLANATORY NOTES
- HBNB Diagrams -

INTRODUCTION :

The purpose of the HBNB project is to build a simple clone of AirBNB platform on our own. The following diagrams were asked to us as much to anticipate and shape our coding as to document our work and practice those diagrams.

1. HIGH LEVEL PACKAGE DIAGRAM :

The **high level package diagram** is meant to represent **the main components of our system**.



In our project, **three layers** organize it :

- Presentation :
  It's the user interface, our frontend. From there, users will ask for account creation, loging, search results, reservations...
  This layer is coded with HTML, CSS, Javascript
- Business Logic :
  Business Logic will be on the server side. The content and functioning will be detailed in the class diagrams.
  In this layer objects of our business : users, places, reviews, amenities, reservations... are described, and so are their behaviours.
  A piece of code known as « facade » will orchestrate the different objects operations and services in charge of database requests made to the persistence layer.
  It's in this layer that users, places, reviews, reservations, are created before being stored or displayed.
- Persistence :
  Persistence layer is the way created objects are stored or recalled.
  Since our business logic program can't keep in memory every user and reservations, they are periodically erased from the server memory.
  Instead of loosing them, we store them in a database.

2. CLASS DIAGRAMS :

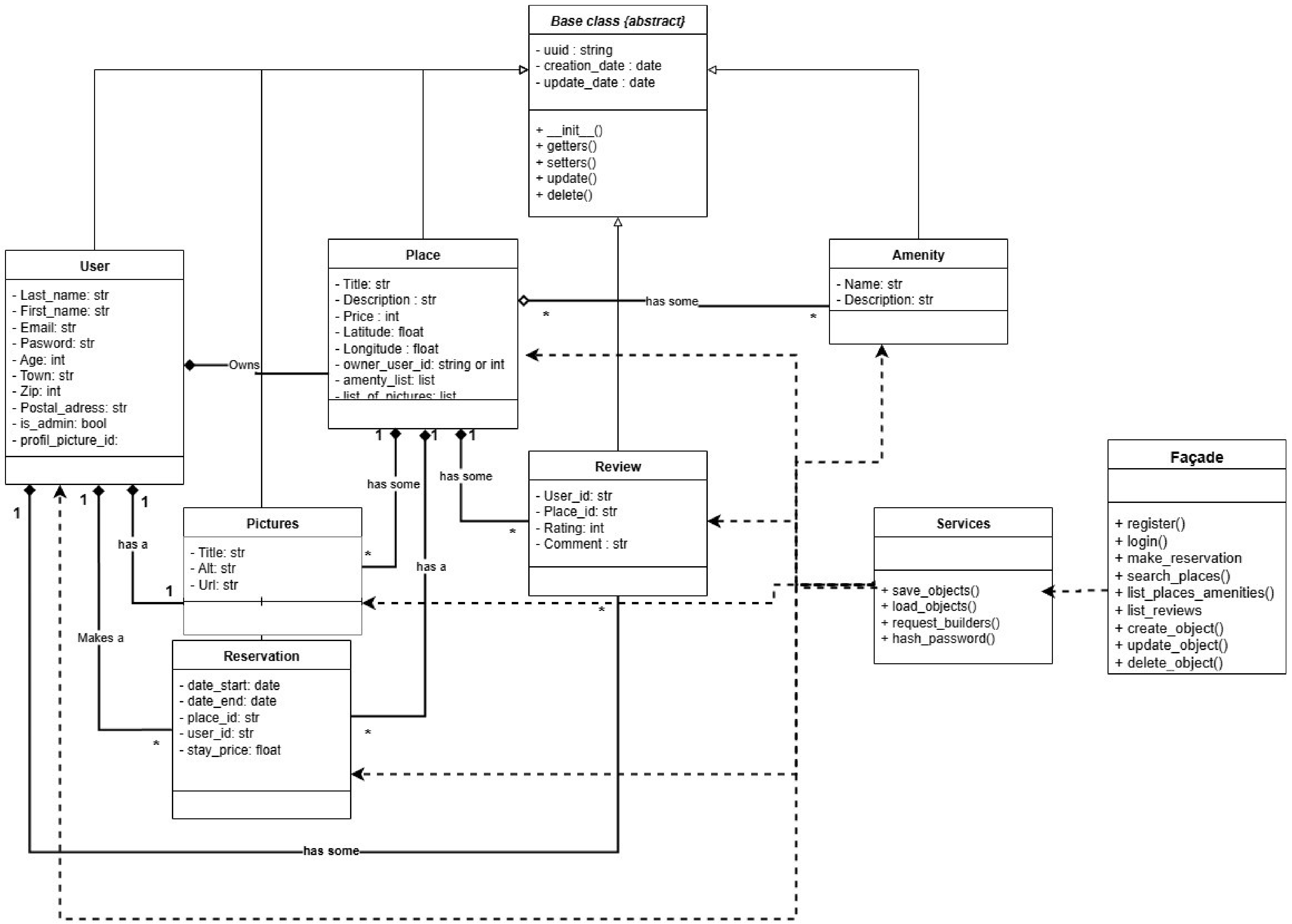
The class diagram is meant to describe the inner system structure : what are the objects composing the system and how they are realted with each others : dependance, inheritance, composition, agregation.
Objects are things we need to manipulate in our system, they are deduced from the purposes and uses of the website.
For example, here is a short description in which we can see deduce objects and methods:

« On Hbnb, any user can look for a place in a town and un a given period of time.

Users can create an account. With this account, he's able to make reservations or to offer a rent for his place.
In order to register, users need to give their firstname, lastname, email, postal adress, age...
They have the possibility to put a profil picture.
Places have a title, a description, a location, a list of comodities, a stay price.
Some reviews can be made about a owner's place, made of rating from 0 to 5 and a comment. »

Class models :

In this description we can deduce some objects : reservations, users, places, reviews, pictures.
Their attributes are mentionned in the details of information requiered (firstname, lastname, etc.).
Verbs related to object can help us deducing some method : create an account, make a reservation, offer a place to rent...
Of course, some basic methods do not appears but are known : object creation, attributes modifications, object deletion, and also methods related to database interactions : database request building, object saving, object loading from database.


Base class {abstract}
- uuid : string
- creation_date : date
- update_date : date
+ __init__()
+ getters()
+ setters()
+ update()
+ delete()
User
- Last_name: str
- First_name: str
- Email: str
- Pasword: str
- Age: int
- Town: str
- Zip: int
- Postal_adress: str
- is_admin: bool
- profil_picture_id:
Place
- Title: str
- Description : str
- Price : int
- Latitude: float
- Longitude : float
- owner_user_id: string or int
- amenty_list: list
- list_of_pictures: list
Amenity
- Name: str
- Description: str
Owns
has some
has some
has some
has a
has a
Makes a
has some
1
1
1
1
1
1
1
1
*
*
*
*
*
*
*
Pictures
- Title: str
- Alt: str
- Url: str
Review
- User_id: str
- Place_id: str
- Rating: int
- Comment : str
Reservation
- date_start: date
- date_end: date
- place_id: str
- user_id: str
- stay_price: float
Services
+ save_objects()
+ load_objects()
+ request_builders()
+ hash_password()
Façade
+ register()
+ login()
+ make_reservation
+ search_places()
+ list_places_amenities()
+ list_reviews
+ create_object()
+ update_object()
+ delete_object()

Some explanation now :

Class roles :

Objects will be responsible of there own creation, their attributes update and their deletion.
Of course, method will be needeed to interact with the database : save the object, call the object, list some objects... Service class will responsible of this part but – even though it's not detailed here, there will be a service class for each object.
And finaly, « façade » will orchestrate the order of methods according to what is asked : database checking, object creation, object saving, etc.

Relations :

Our objects are linked by some relations : inheritance, dependance, aggregation, composition...
All object inherits from Base Class, as an abstract class, making sure each object get some essential attribute such as UUID, creation date, update date... but also basic method to define : creation, attribute update, deletion. Inheritance is represented by

Dependance relation means a class needs to know the class it's arrow is pointing to (in order to call it's method, for example). It's represented by a a dotted line arrow.

Composition is represented by a full diamond. It means that if the class pointed by the diamond is deleted so is the class it's linked to by the composition link. For example, if the user is deleted, so must be the places he created.

Aggregation relation is represented by and empty diamond. It means that many object of a class can be related to an object of another class, but, if the class the diamond is pointing to is deleted, the object linked by aggregation relation can still exist on their own. This is the case for Reviews objects and Places objects.

Multiplicity :

Linked of composition and aggregation are anotated with some numbers representing how many objects belongs of can belongs to another.
For example, a Place can be the property of only 1 user. But a User may have 0 or many Places : 0 or many is represented with this character « * ».

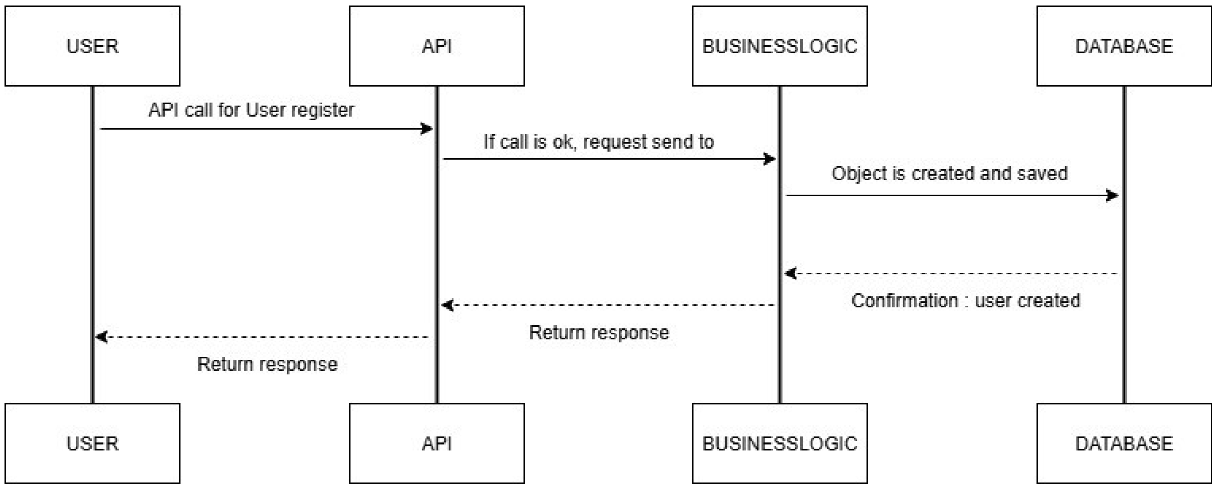
3. SEQUENCE DIAGRAMS :

Sequence diagram purpose is to represent the way different part of the system will interact together over time in order to fulfill a specific user's request : what are they sending to each other, in which order and in which situation ?

Four sequence diagrams where asked to anticipate 4 scenarios :

- User registration
- Place creation
- Review submission
- List of places query

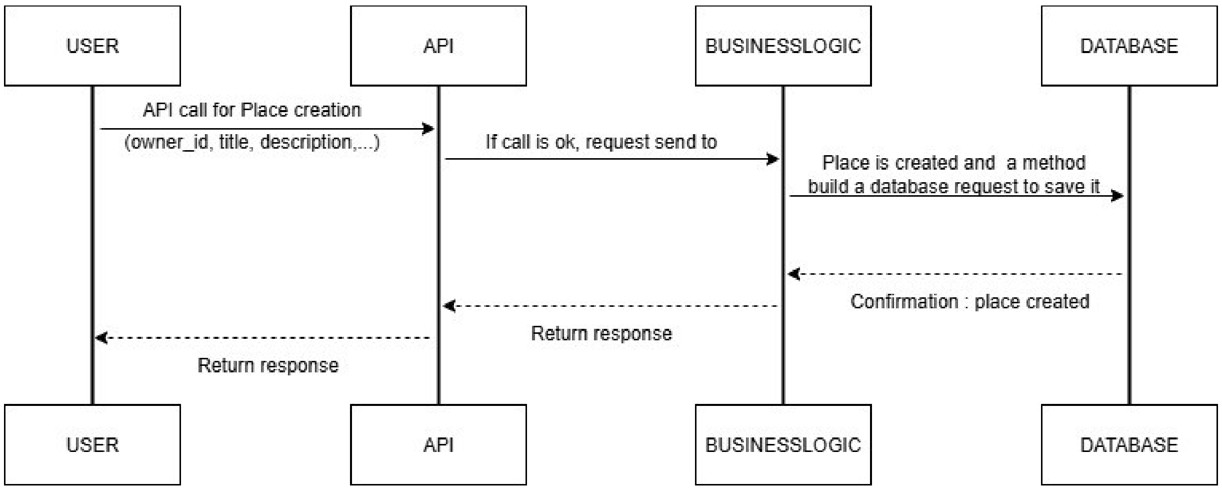
- **User registration :**
  The goal is to see the different messages transmitted beetween each layers, and their orders when a user register.


USER
API
BUSINESSLOGIC
DATABASE
API call for User register
If call is ok, request send to
Object is created and saved
Confirmation : user created
Return response
Return response
USER
API
BUSINESSLOGIC
DATABASE

1. User request a user creation.
2. The request is turned into a JSON and is received by the API which controls it. If the format of the request is good, if permissions are good to, API send it to the right method of the business logic layer.
3. Business logic create the user and calls for a method to build a request that will save the object into the database.
4. Database proceeds to save the object and send back a confirmation.
5. Confirmation is received by BusinessLogic layer which send a confirmation to API.
6. API send the confirmation from BusinessLogic to the UI.

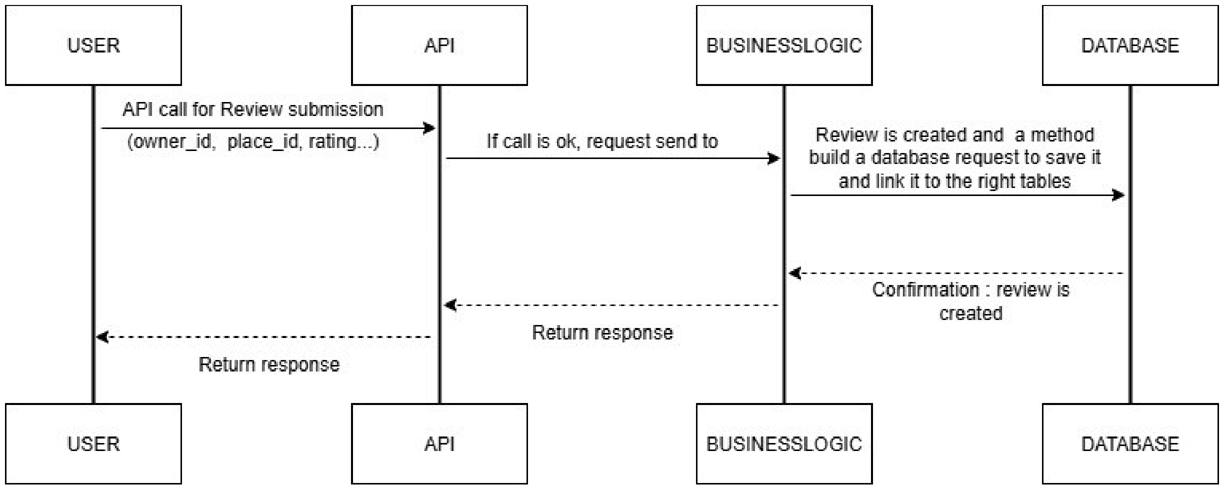
NB : this sequence could be completed with more interaction such as : checking if the email the user gave already exists or not, for example.

- **Place creation :**
  The goal is to see the the goal is to see the different messages transmitted beetween each layers, and their orders when a place is created.


USER
API
BUSINESSLOGIC
DATABASE
API call for Place creation
(owner_id, title, description,...)
If call is ok, request send to
Place is created and a method build a database request to save it
Confirmation : place created
Return response
Return response
USER
API
BUSINESSLOGIC
DATABASE

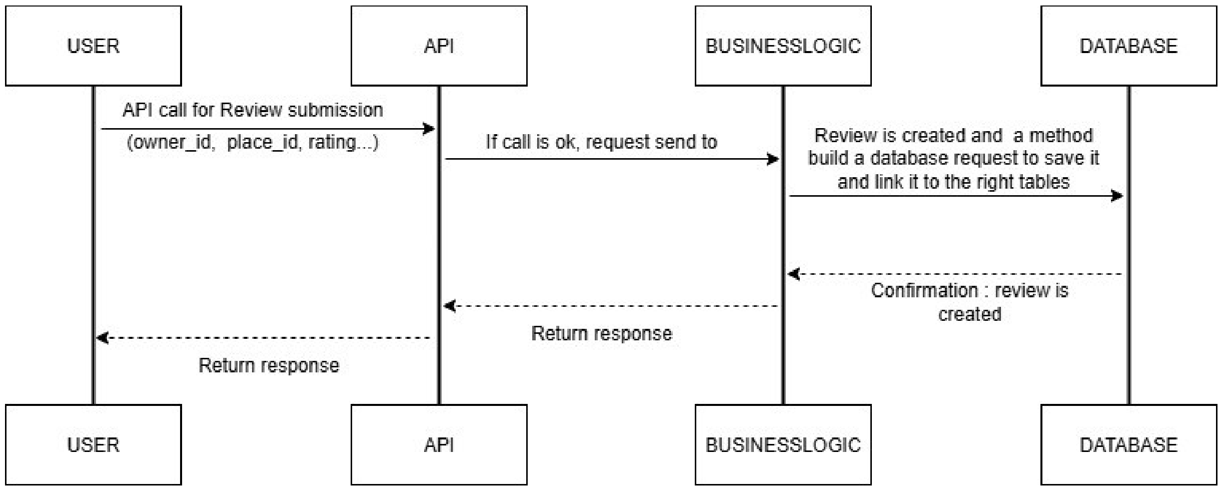
The sequence is pretty similar to user creation, even though methods, messages, error codes... are different.

- **Review submission :**
  The goal is to see the the goal is to see the different messages transmitted beetween each layers, and their orders when a review is created.


USER
API
BUSINESSLOGIC
DATABASE
API call for Review submission
(owner_id, place_id, rating...)
If call is ok, request send to
Review is created and a method build a database request to save it and link it to the right tables
Confirmation : review is created
Return response
Return response
USER
API
BUSINESSLOGIC
DATABASE

The sequence is pretty similar to user creation and place creation, even though methods, messages, error codes... are different.

- **List of places query :**
  The goal is to see the the goal is to see the different messages transmitted beetween each layers, and their orders when user ask to see to see some places based on some specific criterias.


USER
API
BUSINESSLOGIC
DATABASE
API call for Review submission
(owner_id, place_id, rating...)
If call is ok, request send to
Review is created and a method build a database request to save it and link it to the right tables
Confirmation : review is created
Return response
Return response
USER
API
BUSINESSLOGIC
DATABASE

1. User send a place to see some places based on some criterias : location, price, availability dates...
2. Request is send to API which controls if the request format is good, and if user have permissions.
3. BusinessLogic receives the request and call the right method to build the database request for the list a places matching the criterias the user asked.
4. Database returns a table containing the request result and send it to the BusinessLogic layer.
5. BusinessLogic rebuild the places and add them in a list. Returns the list to the API.
6. API send the list to the user in order to the display the places on the frontend