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.

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 « * ».