# UML DIAGRAM

For this project we decided to include some useful information both to the product owner and also to us as a development team. The informations are represented in two diagrams, both written through the graphic language UML: the use cases diagram and the class diagram

*Use cases diagram*

This type of diagram is very useful to fix on paper the features that the product owner wants the application to be able to do. This type of diagram is useful for several reasons, among which the three main ones are: obviously in having the information written in order to remember them well in the time, useful to better explain to the product owner what he wants and also to us to show the product owner that we understood well what he wants.

How we behaved to realize it: we took the pdf files and the directions given on telegram by the professor about the project, we interpreted them as the words said by the product owner to us during the first meeting where he explains what he wants for his application.

First we identified the actors, namely the people and objects that interact directly with the system. We read the content of pdf files and we extrapolated the information about the project requests. We wrote them through use cases and connected them together. Finally we wrote all the various scenarios related to use cases, which explain the application better by representing a possible dialogue between external actors and the system.

Utility for us as a development team: very useful to keep the focus on the demands of the product owner, in this way we avoid inserting more or less elements. Also useful to understand how the various requests are connected, so as to implement them accordingly

Problems encountered: we had some difficulty in extrapolating the requests from the pdf file and how to connect them properly to each other

*Class diagram*

This diagram is very useful for keeping track of the code we are writing, which is how the classes are connected to each other. In this way we can see directly the fields and methods of each class, but above all we see how the various types of objects interact with each other. This last observation is important because it allows us to write new congruent code to the one already written and it also allows us to see if the various objects dialogue in the right way, in the sense of checking if errors can occur that can be resolved by rewriting the code following certain patterns. This last observation is important because it allows us to write new congruent code to the one already written and it also allows us to see if the various objects dialogue in the right way, in the sense of checking whether dialogue errors between objects can occur or if their code is not optimized (any changes to one of them would cause many other changes). These elements can be solved by rewriting the code following certain patterns

How we behaved to realize it: we decided to use Javascript as a programming language, but not using classes with this type of language, we modeled the code replacing the concept of class with the concept of component function, the real protagonist of React (framework used with javascript). We have alternated phases of writing the code, realization of the diagram and consequently a control on it

Utility for us as a development team: very useful to follow the trend and correctness of the code

Problems encountered: we had many problems in starting to realize this type of diagram because there were many difficulties in transporting this class-oriented diagram (and therefore objects) in a code without the use of classes