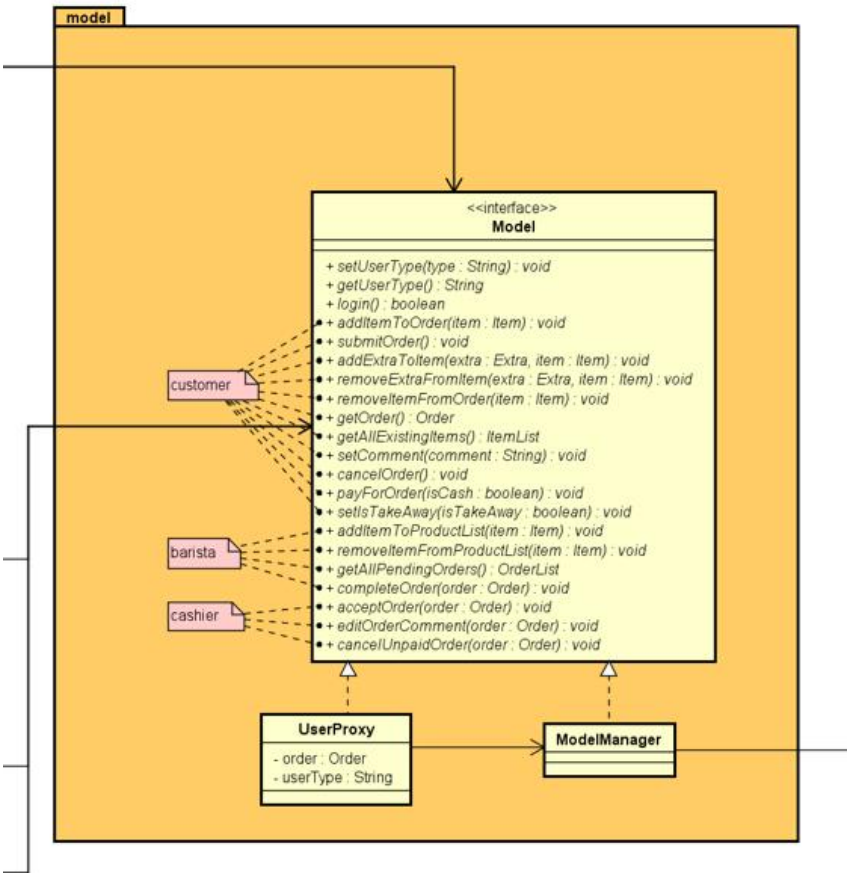
Design of Class Diagram

Foundation of the system

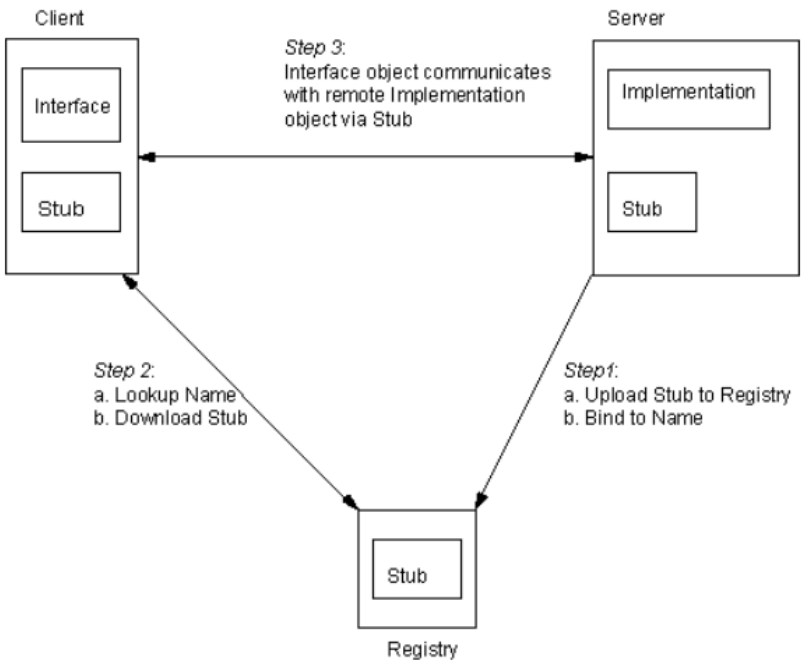
GRASP
SOLID
PROXY
RMI
MVVM Architecture
Factory method for Views



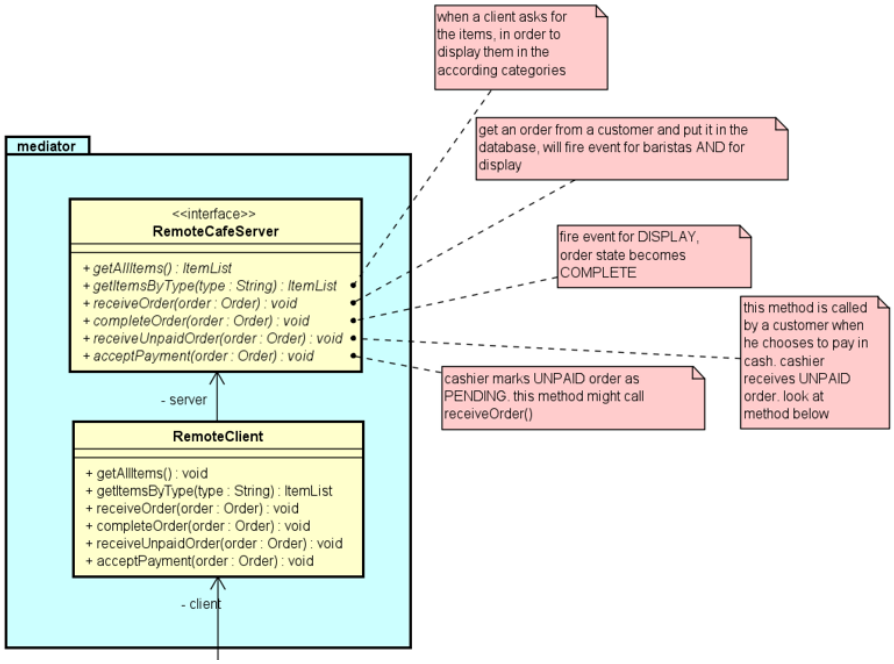
Inevitable part after analysis is design, which was started with creating the first simple class diagram that would be a foundation of the system, later to be extended and modified. Before anything else, the Model interface was built with the necessary methods at the time. Comments about the different possible actions performed by the specific type of user were incorporated. The next step was to invent the utility package which was required due to the methods existing in the Model.

RMI was the design pattern used to create the connection between the Client side and Server side. This solution is instantiating the remote objects which means that the Stub is created and added to the Registry. Binding it to the specific name occurs. All of those events happen in the background and are not visible to the user. As a result of that the TCP connection is acquired and the sockets with the streams are created. After that the Client can call the methods on his computer locally using the exact same interface of the server. Below there are images showing this design pattern.

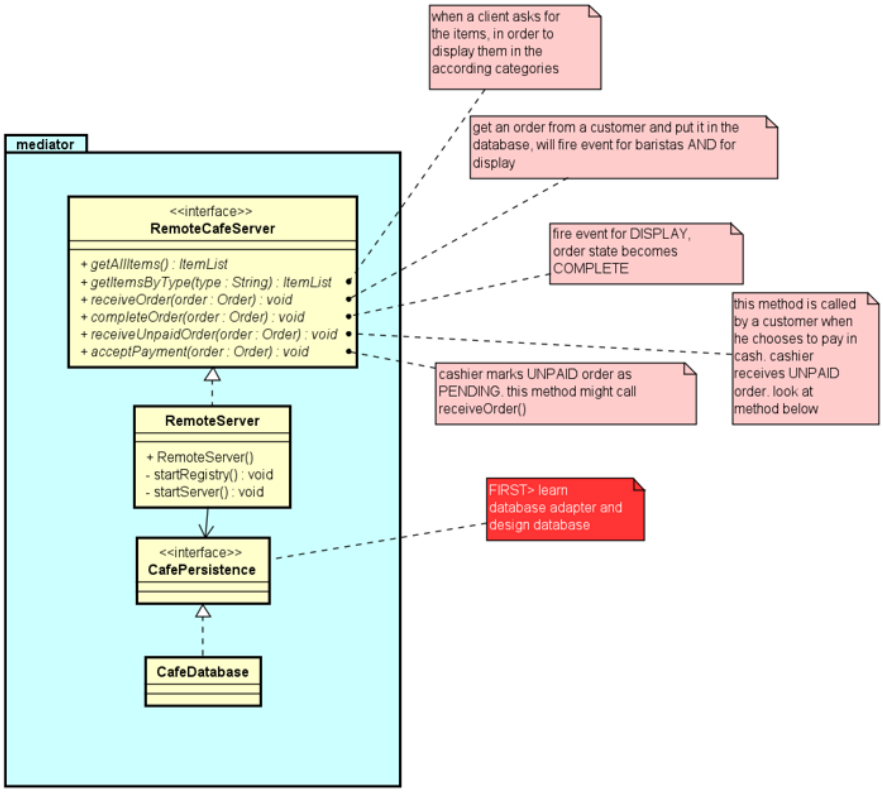
Instantiating the remote objects:



The diagram of the RMI - Client side in this system:



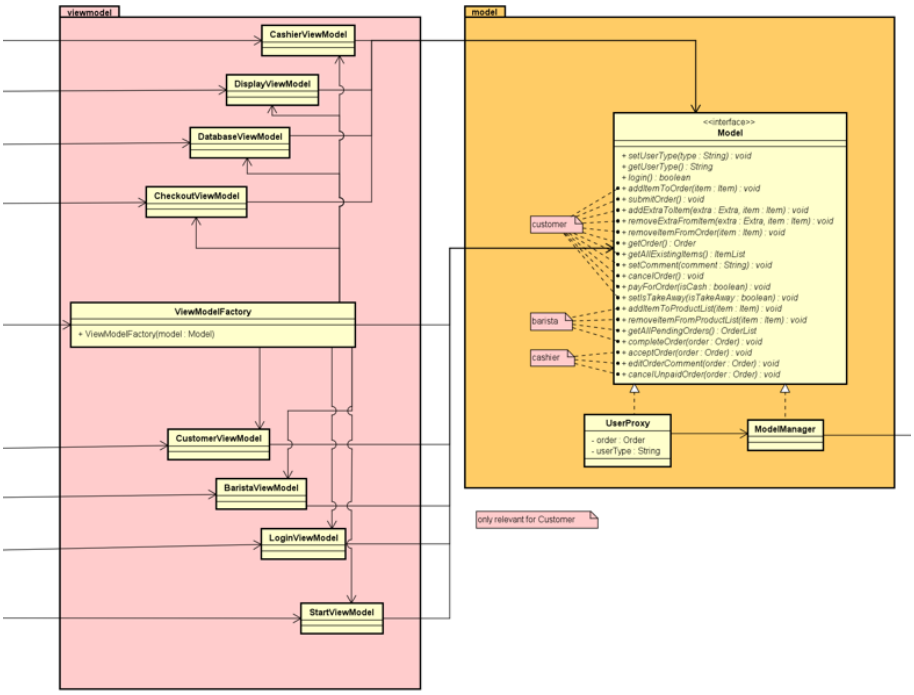
The diagram of the RMI - Server side in our case:



As it can be seen both of the sides are using the exact same interface. As a consequence, all of the logic is happening on the server side and the client side is only delegating the methods which follow the delegation pattern. The notes attached to the classes are slightly explaining the methods and outlining possible extensions.

MVVM was the next design pattern incorporated into the class diagram. The main purpose of this solution is to divide the responsibilities of each package. Model has no association to View or ViewModel and its intention is to retrieve the information requested by the ViewModel or to fire an event when it is necessary. Therefore, the model can also use the Observer pattern which will be described in detail later. ViewModel has only association to Model. Moreover, it is indirectly notifying the View using binding. ViewModel can also use the Observer pattern when being a listener for the Model and the subject. View has the association with the ViewModel and its properties are binded with the ViewModel. It is delegating methods to the ViewModel and it can be a listener for the ViewModel.

The MVVM pattern in this system:



Even though the methods in ViewModel are not shown, it is implied by the association to the Model class that delegation of methods to the Model will occur.

When drafting the first version of the class diagram, two of the GRASP principles were taken into consideration

The creator principle is reflected in the view package of the class diagram. The factory method was used to create new views in the ViewCreator class.The idea of the factory method is to define a general creation of objects while letting the subclasses which objects to create. In this case, the ViewCreator is creating and loading the views, only with an id. This makes it easier to create multiple views later in the project.

Low coupling has been achieved by minimising the dependencies between the classes through the use of interfaces which results in higher reuse of classes.

The SOLID principles have also been taken into consideration when the class diagram was designed.

Each of the currently existing classes have only one responsibility, respecting the single responsibility principle. For instance, in the utility package the Order class only handles information about the order’s comment, date and time, and has an association to the Item class, which similarly only handles data relevant for itself.

Following the open closed principle the created classes are open for extension and closed for modification: for example, instead of modifying every method in the model such that certain types of users have different permissions, the proxy design pattern was used by adding another class that implements the same model, delegating to these methods while checking the user type.