**TCP socket**

* **Architecture:**

For TCP, we build our architecture in the same manner in the project description. We created multiple clients that will send requests to the Middleware, according to what command it is required to be executed, the request is forwarded to the appropriate resource manager (Flight, Room, Car), and then the resource manager will send back a String response to the middleware which will be forwarded to the client. We started by separating the customer from the resource managers and making it handled only in the middleware level however we faced data inconsistency problems since the resource managers also need the customer information to be saved in their storage system. Hence, for requests about customer update, we decided that the best solution is to be handled in the middleware level through calling the three resource managers to update them about the customer state.

2.Deserialization and processing of command

1.Request (command, parameters)

Flight ResourceManage (serverSocketFlight.java)

Client (serverSocketClient.java)

Client (serverSocketClient.java)

Car ResourceManager (serverSocketCar.java)

Middleware (TCPMiddleware.java)

Client (serverSocketClient.java)

Room ResourceManager (serverSocketRoomjava)

3.Serialization and send Request to the appropriate server according to command content.

* **Java serialization:**

The "Request" class is a Java serializable class designed for use in TCP communication networks. It serves as a data structure to encapsulate a command represented as a string and an array of parameters represented as objects. This class enables the easy serialization and deserialization of requests for transmission and processing within the middleware as well as the resource managers.

* **The roadmap for our project is the following:**

The user ‘input is sent to the client; the client will parse this input and create a java serializable class Request. Afterwards, this Request object will be sent to the Middleware using an ObjectOutputStream, the Middleware will unmarshall this Request by accessing its command String and search if it contains Flight, Room, Car, Bundle, Help, Customer.

If the command contains one of the resource managers names it will be directly forwarded to the appropriate one and printing on the Middleware "Forwarding request to the **Resource\_manager\_name** resource manager".

If the command contains **Customer**, then the Request will be forwarded to the three resource managers according to conditions that makes the requests related to Customer be handled. For example, if the three responses are given by the Resource managers are the same, then send only one response to the client, if one of them fails, then send three of them to see the issue.

If the command contains **Help**, then the middleware sends a String response to the client containing the different commands that could be useful for the user.

If the command contains **Bundle**, it is handled between the middleware and three resource managers in the following manner. We go through each of the parameters given to bundle. Starting with checking if we reserve room and/or car. If it is the case, we forward the request to the appropriate resource manager, if reservation is succeeded, we go to the next step. If one of the reservation steps fails, we undo the reservation for the previous steps and we exit the bundle function announcing its failure. To cancel the reservation, we built a new function called **undoReservation that takes the same parameters as reserveItem (used for reservation) and do just the opposite of reserveItem by checking if there are reservation for the given item, if yes, it increments the number of items in storage and increments the number of reserved items, hence it cancels the reservation for the item .**

* **Error handling**
* If the user inputs an invalid command, an error “Command does not exist” will be raised.
* If one of the methods parameters is missing, an error “Missing arguments, Expected: command, <parameter1>, <parameter2> …”
* If the client is disconnected it appears to the middleware, that the client is disconnected and the middleware keeps trying to reconnect until the connection is established again.
* If one of the resource managers is disconnected, the middleware stays listening waiting to connect to them and forward a message to the client ask him to try again later.

Concurrency:

To test the concurrency, we added a small delay around 2ms in one of the clients using Thread.sleep(2000) called ClientSocket\_delayed. We executed the two clients at the same time to ensure some type of concurrency.

When two clients are connected to the server, if multiple cars are added in a location. If one client reserves a car then the other client will see that the number of cars available at this location is decreased by 1.