Spencer Handfield - 260805699
Ethan Itovitch - 260794124
Benjamin Szwimer - 260804222

Architecture & Design RMI

For this project there were three ResourceManagers needed in order to implement the system which consisted of Flights, Cars and Rooms ResourceManager. The purpose of this project was to create a system that was capable of distributing the calls from the client onto the desired ResourceManagers server depending on if the client needed various functionalities from either the Flights, Cars or Rooms ResourceManager and then ultimately return the responses from the server back to the client side. In order to implement the functionality of the client interacting with the appropriate ResourceManager a Middleware was required to do so. The Middlewares responsibility is to essentially route the incoming information from the client to the appropriate ResourceManager. I.e if there was a Car method call only the Cars ResourceManager will be called from the Middleware. For Simplicity sake we felt that in order to implement the customer functionality the easiest way was to give each ResourceManager namely the Flights, Cars and Rooms ResourceManager their own individual copy of the customers in the system in order to not have the need to implement another server or to do this at the Middleware level. This redundancy allowed us to simply create users (by calling the create user on all three resource managers) and getting user information since the customer information on each resource manager would only be associated with that resource manager's respective information. Lastly in order to implement bundling, what was done was if a bundle came in from the client side it would make a call to the Flights ResourceManager for every flight in the list of flights in the bundle. If there was a car in the bundle it would make a call to the Cars ResourceManager and same for the Rooms ResourceManager.

Architecture & Design TCP

In order to explain the architecture for our TCP implementation we will start from the Client side. When a client starts up, a stub of the resource manager is created. This ResourceStub has the same signature of the resource manager however, behind the scenes passes the method name and arguments to its connected server as a string through TCP. From the middleware's perspective, once the ResourceStub is created, the middleware accepts the socket connection, creates three new ResourceStub’s connected to the resource managers and starts a TCPthread with those ResourceStubs for that Client. In order to have the ability to create more than one concurrent call to the

actual ResourceManagers it creates a new socket connection to the ResourceManager for each ResourceManager for each thread. From the ResourceManager's perspective, when the middleware stubs create the socket connection, the ResourceManagers accept the connection and start a TCPThread with the real ResourceManagers (as opposed to the stubs). Now that the connection is established we can now send messages. When a message is sent from the Client, it calls the appropriate message on it's ResourceStub the message is picked up by the Middleware's appropriate TCPThread. The TCPThread then parses the message and passes it to it's ResourceManager which in the middleware's case is another ResourceStub. The ResourceStub then passes the message to the ResourceManager which is picked up by its TCPThread. The ResourceManager's TCPThread then picks up the message and passes it to it's ResourceManager which in this case is the real ResourceManager. The ResourceManager then handles the method and returns the response. The TCPThread then passes the message back to the Middlewares ResourceStub which passes the message back to it's TCPThread which finally passes the message back to the Client's ResourceStub for the Client to then return as output.

Custom functionality

We opted to implement the analytics custom functionality. With the already existing implementation of the client-middleware-server communication we could integrate in the new required commands, cases and appropriate calls in the resource manager rather effectively. Initially a *QueryAnalystics* command is created from the client. Query analytics works by passing in a parameter which is location. If passed all, that will query all the ResourceManagers data and return the Cars, Flights and Rooms based on the availability remaining. It will either be high, medium or low availability remaining. One can also pass a specified location which will return all the Cars, Flights and Rooms at this specific location.

Contributions

We as a group divided the project into three logical sections. The RMI Distribution implementation, The TCP Sockets implementation and lastly the implementation of the custom functionality. Benjamin Szwimer was responsible for the RMI Distribution implementation, Ethan Itovitch was responsible for the TCP Sockets implementation and Spencer Handfield was responsible for the custom functionality. We all worked as a team whenever issues arised or big design decisions were needed. Lastly the report was done by all three members of the team.