DistributedSystemsReadMe

Gary Connelly -G00336837

Due December 10th 2018

Contents

1 Introduction

This is the ReadMe document for the project completed as part of the Distributed Systems module in 4th year Computing in Software Development course.

For this project, we were required to use the JAX-RS/Jersey, Java RMI and JAXB frameworks to develop a simple Car Hire Booking system. A Web Client page should provide users with the ability to Create/Modify/Update/Delete bookings for a specific vehicle for a given set of dates. The Web Client will interact with a RESTful JAX-RS Web Service for bookings which is deployed on Apache Tomcat Server. The RESTful Web Service will act as a RMI client to a RMI Database Server which will handle persistence.

2 Features

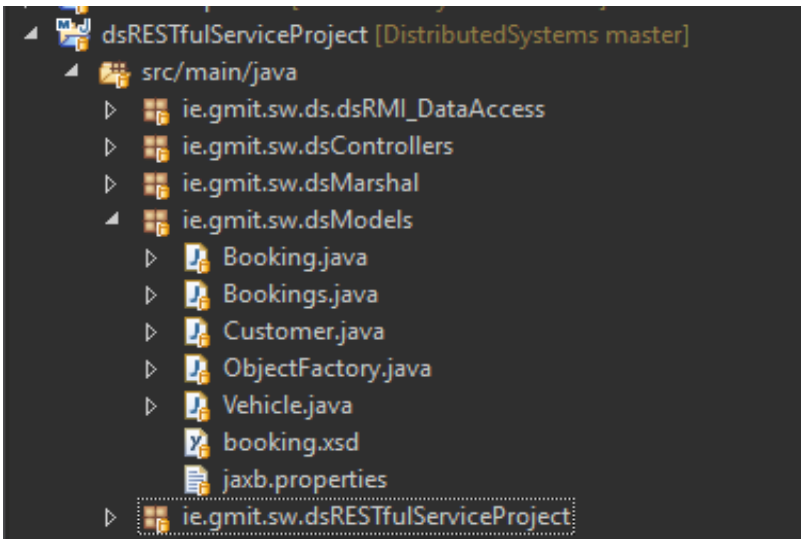
2.1 RMI

The RMI part of the project is contained in the project dsRESTfulServiceProject project in the package called ie.gmit.sw.ds.dsRMI_DataAccess.

This section of the project is responsible for connecting to the database, querying the database and sending data to the RESTful resource via a remote object. The BookingServiceImpl implements the IBookingService. The BookingServiceImpl is the object that can be accessed by the RMI client to interact with the RESTful service.

2.2 Data Models

The data for this project is all stored in a mysql database. The database contains 3 tables; Booking, Customer, Vehicle. These tables are related to each other via foreign keys. I created a .xsd file that would be used as the xml schema for the models to be used in this project. This .xsd file can be found in ie.gmit.sw.dsModels in a file called booking.xsd. This file is used to generate the java classed that will be used as the models for this part of the project.



2.3 RESTful Resources

The RESTful service section of this project is contained in the dsRestfulServiceProject project. This project contains the RMI server and the models which we spoke about earlier. This also contains the business logic that is needed to process the object that is returned from the database and sent to thr RESTful resource and vice versa.

This project also has the resources package which is responsible for getting data and sending data accross the network via HTTP.

This is what the restful recources look like.

```java
package ie.gmit.sw.dsRESTfulServiceProject;



import javax.ws.rs.Consumes;

public interface Resource {

    /*
     * An interface to abstract the business logic from the RESTful annotations.
     * The RESTful resources will implement this interface and process the requests.
     */

    @GET
    @Produces(MediaType.APPLICATION_XML)
    public Response getIt();

    @GET
    @Produces(MediaType.APPLICATION_XML)
    @Path("/{value}")
    public Response getById(@PathParam("value") String value);

    @POST
    @Consumes(MediaType.APPLICATION_XML)
    public Response create(String input);

    @PUT
    @Consumes(MediaType.APPLICATION_XML)
    @Path("{id}")
    public Response update(@PathParam("id") final String id, String input);

    @DELETE
    @Consumes(MediaType.APPLICATION_XML)
    @Path("{id}")
    public Response delete(@PathParam("id") final String id, String input);

}

```

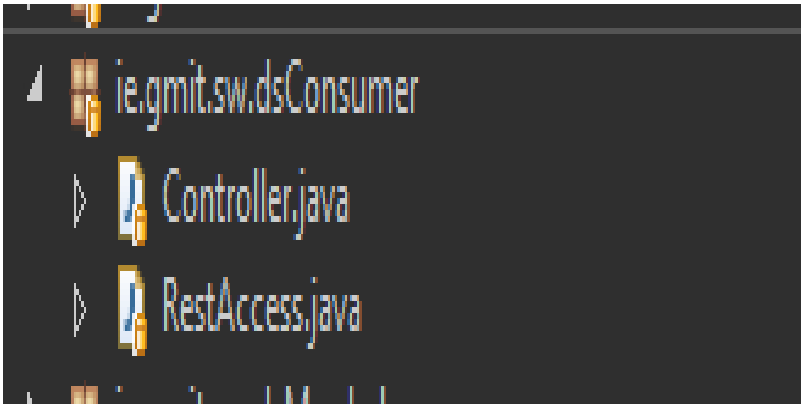
```java
@Path("bookings") // When the client makes a call to the bookings resource
public class MyResource extends BookingMarshal implements Resource {

    BookingController controller = new BookingController(); // Initialize the controller.

    @Override
    public Response getIt() { // Method to return all bookings.
        List<Booking> bookings = new ArrayList<Booking>(); // Initialize a list of booking objects.

        List<ReturnedBooking> returnStatement = controller.getAllBookings(); // Get a list of ReturnedBooking objects

        /*
         * For each ReturnedBooking object in the list, map all the relevant elements a new instance of
         * the Booking model class which can be sent over the network.
         *
         */
        for (ReturnedBooking rb : returnStatement) {
            Booking booking = new Booking();
            Customer customer = new Customer();
            Vehicle vehicle = new Vehicle();
            System.out.println(rb.getBookingId());
            customer.setCustomerId(rb.getCustomerId());
            vehicle.setId(rb.getVehicleId());
            booking.setBookingId(rb.getBookingId());
            booking.setCustomer(customer);
            booking.setVehicle(vehicle);
            booking.setStartDate(rb.getStartDate());
            booking.setEndDate(rb.getEndDate());

            bookings.add(booking);
        }

        Bookings bookingList = new Bookings(); // Initialize the bookings object to contain a list of booking objects
        bookingList.setBooking(bookings); // Set the bookings to the list we just created.
        String msg = getBookingsAsXML(bookingList); // Marshal this list so it can be sent via HTTP.

        return Response.status(200).entity(msg).build(); // Return the xml with a status 200 server message.
    }
```

2.4 Web Client

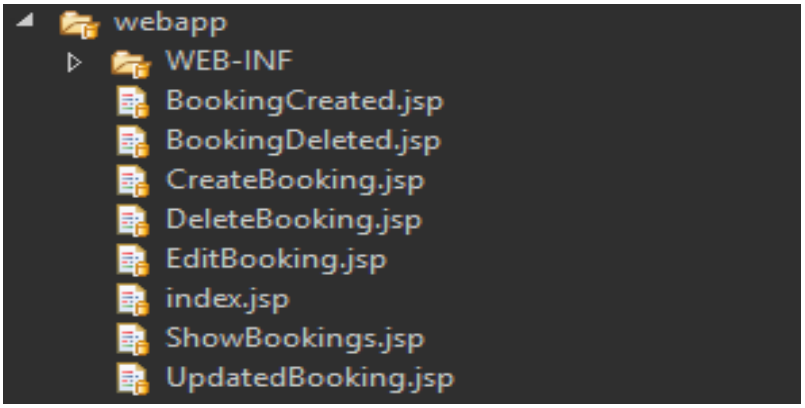
For the web client section of this project, I used JSP. JSP(Java Server Pages) is a web applications technology that allows developers to create dynamic web applications and embed java scriptlets into html pages. This allowed me to easily call external java clases to send data to and from the web pages and, subsequently to and from the RESTful service.

I also needed a copy of the xsd schema on the webclient as I needed it to create the same objects that are on the server so that the xml can be marshalled back to the respective java classes. The web client for this project is contained in the dsClient2 eclipse project. The reason for this is because I was having

configuration issues with the dsClient project. Inside this eclipse project I have a package called ie.gmit.sw.dsConsumer package. Inside this package, are the classes that are responsible for processing requests and responses to and from the service. The class RestAccess is responsible for connecting to the RESTful api. The Controller class is responsible for the business logic between the data access(RestAccess) and the JSP web pages.



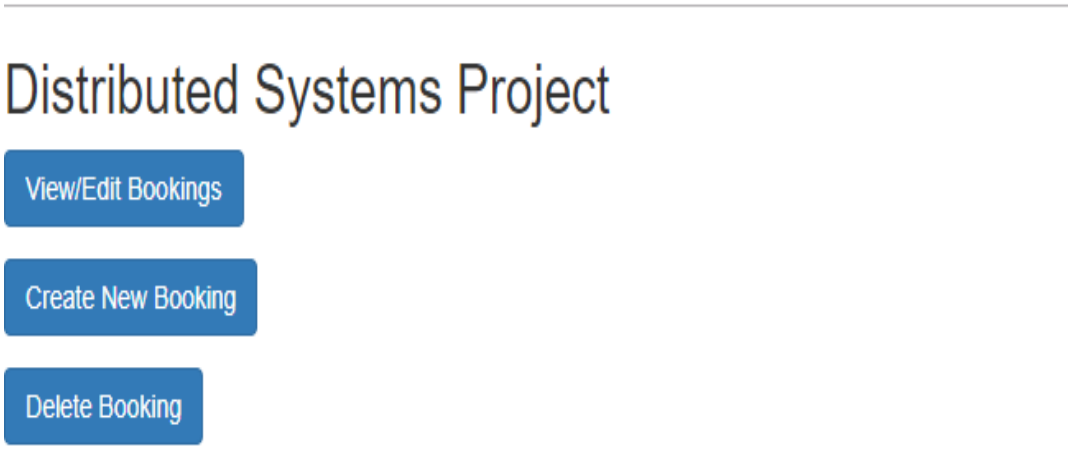
I then have multiple jsp webpages, one for each view, which talk to the Controller class using the embedded java scriptlet capable of calling the controller object.



```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@page import="ie.gmit.sw.dsConsumer.Controller"%>
<%@page import="ie.gmit.sw.dsModels.Booking"%>
<%@page import="java.util.List"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
<script
    src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script
    src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
<style>
table, th, td {
    border: 1px solid black;
}
</style>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Show Bookings</title>
</head>
<body>

    <%
        Controller controller = new Controller();

        List<Booking> bookings = controller.getBookings();
    %>
    <h2>Show All Bookings</h2>
```

All this comes together to generate the various views in the project:

2.4.1 Views



Show All Bookings

ID	Customer_ID	Vehicle_ID	Start_Date	End_Date	Edit
1	1	1	01-01-2018	02-01-2018	1 Edit
4	20	4	01-01-2018	02-01-2018	4 Edit
5	5	5	01-01-2018	02-01-2018	5 Edit
20	20	20	03-03-2019	04-04-2019	20 Edit
15	15	30		01-01-2019	15 Edit

Edit booking Where ID is 1

Booking ID:

1

Customer ID:

1

Vehicle ID:

1

Start Date:

01-01-2018

End Date:

02-01-2018

Update

Create a new Booking(All fields are mandatory)

Booking ID:

00

Customer ID:

00

Vehicle ID:

00

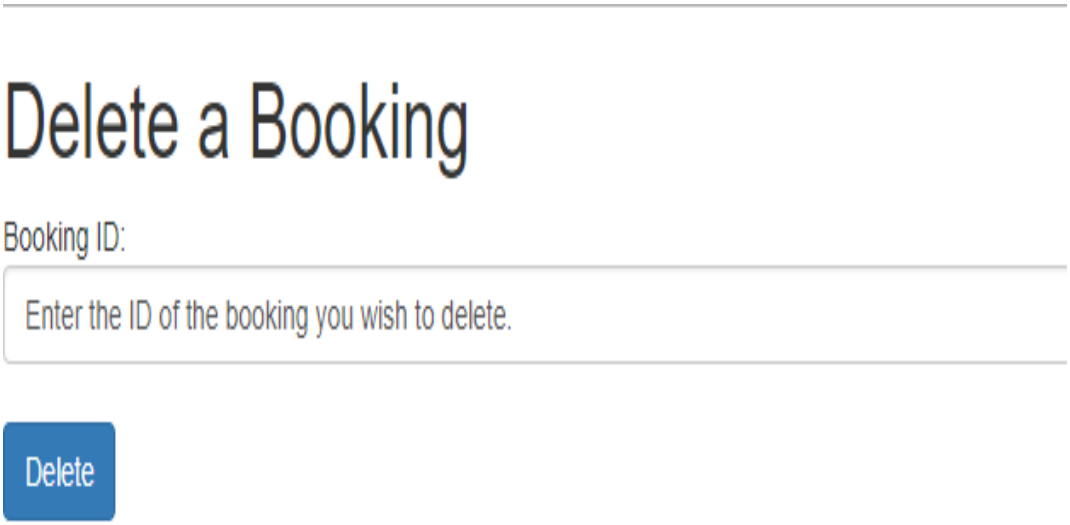
Start Date:

00-00-00

End Date:

00-00-00

Create



2.5 Extra Features

As there is 20% of the marks in this project going for extra features, and because of the fact that I was finished the project a few days early, I decided to add a couple of extra features to it.

2.5.1 Web Security

Web security is a really hot topic at the moment, I decided to implement some sort of web security that would prevent the user from performing an sql injection attack on the database from the forms that appear in the views.

An sql injection attack is where the user simply stypes sql syntax into an input box in the view. If the user new even basic sql, they could perform very damaging attacks on the database by carefully placed semi-colons, which would prematurely end an sql statement, and then input a "drop tables;" query straight after it. This is just one of many examples of the many sql injection attacks the user can perform.

To get around this problem, I created a simple java class that parses the sql query, checking it for semi-colons before it lets the query through to the database. This way, if the user inputs a semi-colon into the input box on the view, the query will never reach the database.

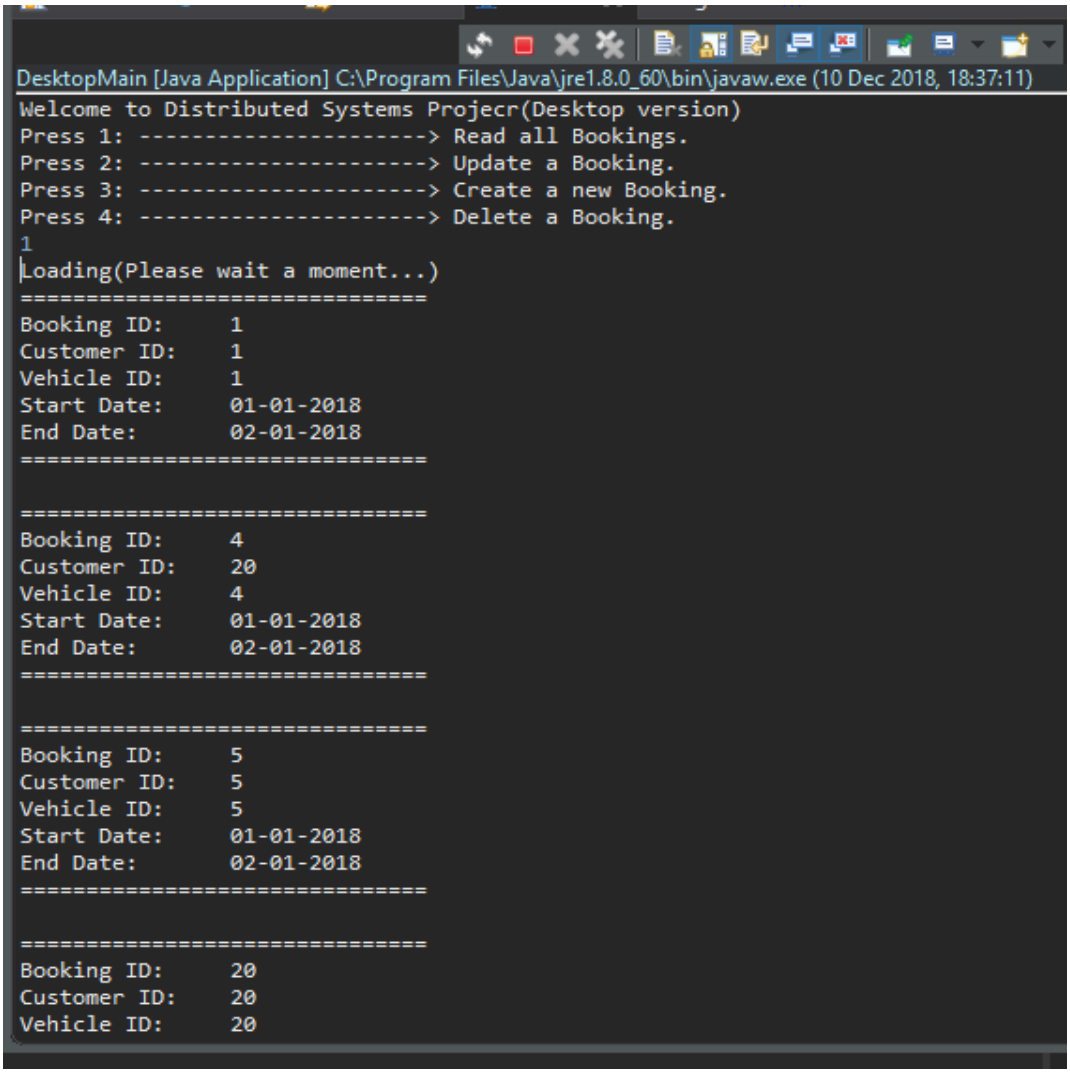
```java
package ie.gmit.sw.dsControllers;

public class SQLParser {
    /*
     * This is a very simple class the provides the application with protection
     * against sql injection attacks. All it does is parse each query and checks
     * each character to see if it contains a ';', meaning that the user can't for
     * example enter "12-12-2018;Drop tables;" into a an input field on a form on
     * the client side, which could be very damaging to the database.
     */

    public String parseSQL(String query) { // Take in an SQL query as a parameter.

        System.out.println(query);

        int syntax = 0; // Initialize a counter and set it to 0.

        for (int i = 0; i < query.length(); i++) { // Create a for loop to loop through each character in the query.
            char c = query.charAt(i); // Initialize a char variable to the current character the loop is on in the
                                      // query.

            if (c == ';') // If the character is a semi-colon.
            {
                syntax++; // Increase the counter by one.
            }
        }

        if (syntax == 0) { // Only if the counter is equal to 0 at the end of the loop.
            return query + ";"; // Append a semi-colon to the end of the query and send it back to the
                                // controller.
        } else { // If the counter is greater than 0.
            return null; // Simply return null, meaning a harmless null query will be sent to the
                         // database.
        }

    }

}
```

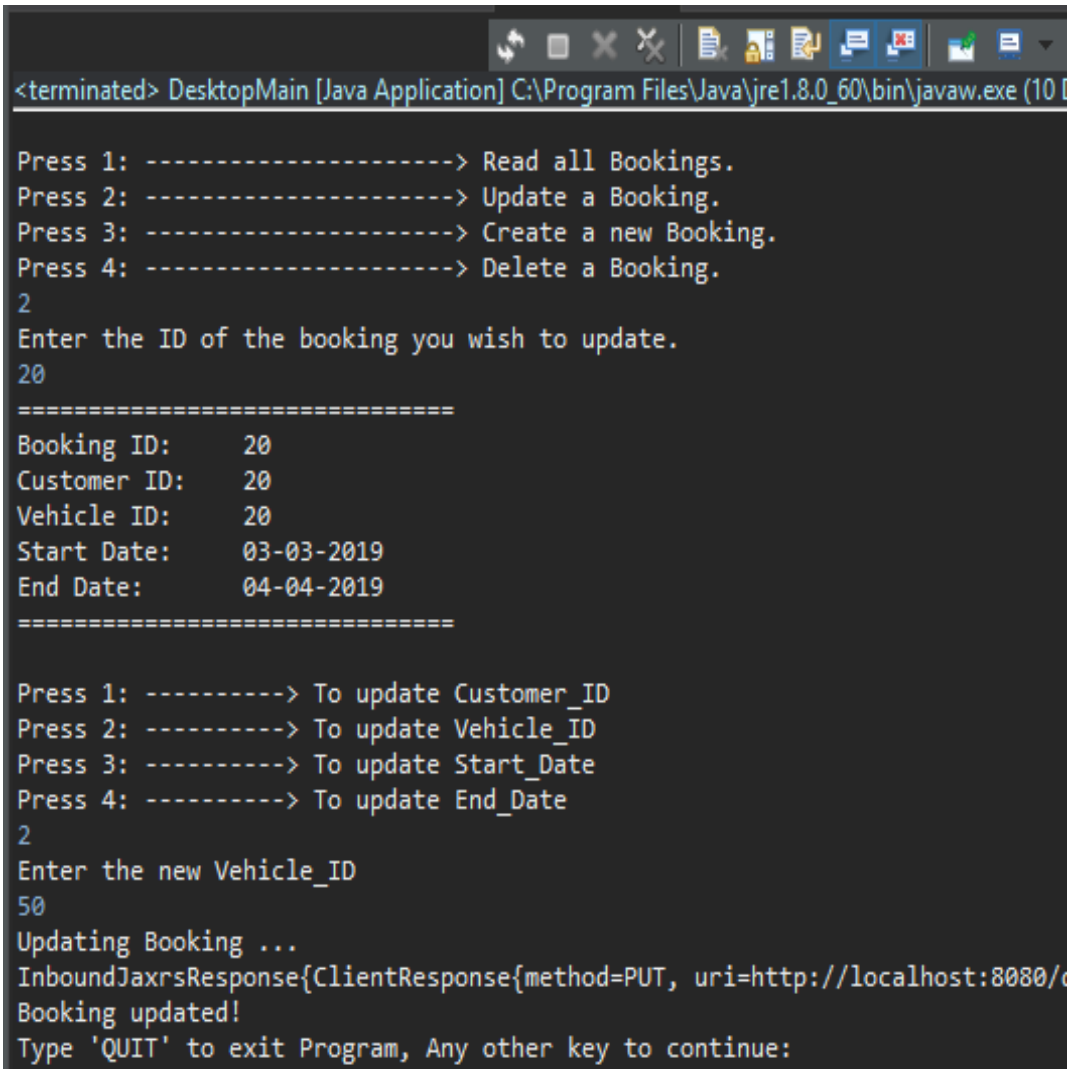
```
public void updateBooking(ReturnedBooking booking) { // Method for updating a booking in the database.

    String query = "UPDATE bookings SET vehicle_id =" + booking.getVehicleId() + ", " + "customer_id ="
            + booking.getCustomerId() + ", " + "start_date =\"" + booking.getStartDate() + "\", " + "end_
            + booking.getEndDate() + "\" WHERE booking_id=" + booking.getBookingId() + ""; // Construct t

    SQLParser parser = new SQLParser();
    query = parser.parseSQL(query);


    try {
        bookingService.updateBooking(query); // Invoke the remote method.
    } catch (RemoteException e) {
        System.out.println("error updating booking in Booking controller"); // Catch any exceptions.
        e.printStackTrace();
    }

}
```

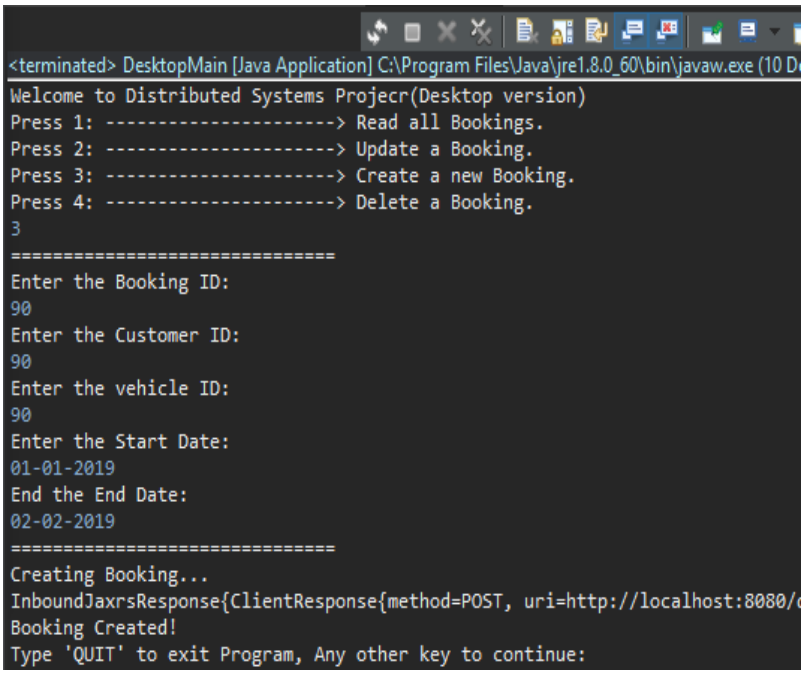
2.5.2 Desktop Client

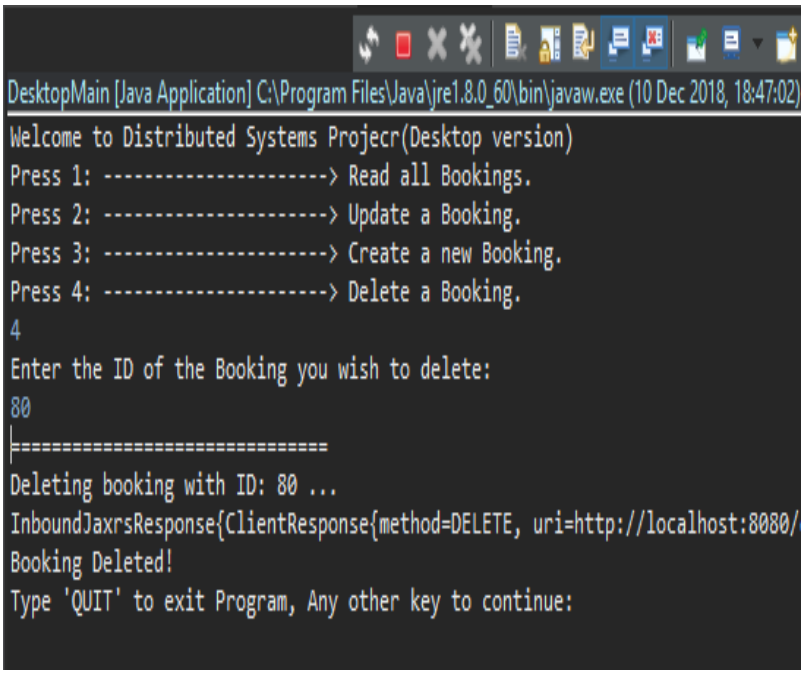
I also created a new client that can be accessed from the users desktop. This client does not have a user interface, the user must interact with the client through commands. This client has all of the same functionality that the web client has(Create, read, update, delete).

The reason I decided to make a desktop client was to show that the restful service is heterogeneous, meaning that it can be accessed by many client side technologies. As long as the client has a copy of the schema, no matter what software technology it uses i.e java, c#, typescript, they can interact with this service.


DesktopMain [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (10 Dec 2018, 18:37:11)
Welcome to Distributed Systems Projecr(Desktop version)
Press 1: ----------------------> Read all Bookings.
Press 2: ----------------------> Update a Booking.
Press 3: ----------------------> Create a new Booking.
Press 4: ----------------------> Delete a Booking.
1
Loading(Please wait a moment...)
Booking ID: 1
Customer ID: 1
Vehicle ID: 1
Start Date: 01-01-2018
End Date: 02-01-2018
Booking ID: 4
Customer ID: 20
Vehicle ID: 4
Start Date: 01-01-2018
End Date: 02-01-2018
Booking ID: 5
Customer ID: 5
Vehicle ID: 5
Start Date: 01-01-2018
End Date: 02-01-2018
Booking ID: 20
Customer ID: 20
Vehicle ID: 20


<terminated> DesktopMain [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (10
Press 1: ----------> Read all Bookings.
Press 2: ----------> Update a Booking.
Press 3: ----------> Create a new Booking.
Press 4: ----------> Delete a Booking.
2
Enter the ID of the booking you wish to update.
20
==========
Booking ID: 20
Customer ID: 20
Vehicle ID: 20
Start Date: 03-03-2019
End Date: 04-04-2019
==========
Press 1: ----------> To update Customer_ID
Press 2: ----------> To update Vehicle_ID
Press 3: ----------> To update Start_Date
Press 4: ----------> To update End_Date
2
Enter the new Vehicle_ID
50
Updating Booking ...
InboundJaxrsResponse{ClientResponse{method=PUT, uri=http://localhost:8080/
Booking updated!
Type 'QUIT' to exit Program, Any other key to continue:


<terminated> DesktopMain [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (10 D
Welcome to Distributed Systems Projecr(Desktop version)
Press 1: ----------> Read all Bookings.
Press 2: ----------> Update a Booking.
Press 3: ----------> Create a new Booking.
Press 4: ----------> Delete a Booking.
3
==========
Enter the Booking ID:
90
Enter the Customer ID:
90
Enter the vehicle ID:
90
Enter the Start Date:
01-01-2019
End the End Date:
02-02-2019
==========
Creating Booking...
InboundJaxrsResponse{ClientResponse{method=POST, uri=http://localhost:8080/
Booking Created!
Type 'QUIT' to exit Program, Any other key to continue:


DesktopMain [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (10 Dec 2018, 18:47:02)
Welcome to Distributed Systems Projecr(Desktop version)
Press 1: ----------> Read all Bookings.
Press 2: ----------> Update a Booking.
Press 3: ----------> Create a new Booking.
Press 4: ----------> Delete a Booking.
4
Enter the ID of the Booking you wish to delete:
80
==========
Deleting booking with ID: 80 ...
InboundJaxrsResponse{ClientResponse{method=DELETE, uri=http://localhost:8080/
Booking Deleted!
Type 'QUIT' to exit Program, Any other key to continue:

3 How to Run

- Download this project from github - https://github.com/garyconnelly1/DistributedSystems.git
- Import the three relevant projects into eclipse - Open the eclipse IDE, go to File -¿ import -¿ Import maven and import dsDesktopClient, dsClient2, and dsRestfulServiceProject.
- Make sure that dsRestfulServiceProject and dsClient2 have Tomcat server as their targeted runtimes.
- Navigate to the dsRestfulServiceProject, and go to the ie.gmit.sw.ds.dsRMI_DataAccess.
- Navigate App.java and run as java application.
- When the console says "Server ready", right click on the whole project and click run on - run on server.
- Wait until the internal web server launches in side eclipse.
- Navigate to the following url - "http://localhost:8080/dsClient2/".

4 Known Errors and Limitations

While this application is very hard to break, it is far from perfect. For example, there is very little validation of inputs on the web client. If the user enters for example tries to delete a booking that does not exist, they will not receive any notification that it does not exist. If there is a problem with the input, it will get handled on the server side but will not notify the user that something is wrong, it just wont update the database.

Another issue that is occurring is that the tomcat server sometimes does not run because it says it timed out. I don't really know why this occurs but it happens very rarely.

When you run the dsRestfulServiceProject, you will sometimes get a stack-trace of errors saying NoSuchMethodError. The server will continue to run even when this method is thrown.