Project Phase-1

1st Adit Shah
Dept. E.C.E.
40172848
Concordia University
adit.shah@mail.concordia.ca

2nd Ishwinder Singh
Dept. E.C.E.
40197552
Concordia University
s_ishwin@live.concordia.ca

3rd Neha Chaudhary
Dept. E.C.E.
40198535
Concordia University
ne_chaud@live.concordia.ca

4th Navaldeep Sandhu
Dept. E.C.E.
40172848
Concordia University
n_sandhu@live.concordia.ca

***Abstract*—The goal of our project is to create a client/server distributed system model which allows all the Ski Resorts to get digital. The idea is to have ski resorts employ RFID lift ticket readers to automatically log the skier's ID and ride duration each time they use a ski lift. So, in our Project, we will create the client side with the help of multi threading and it will all of the data to the server. The server side is deployed by using the Oracle free-tier cloud.**

***Index Terms*—RFID, Ski Resorts, Oracle free tier cloud.**

I. Introduction

The Client-Server distributed system model is created and implemented by using the Spring framework. The framework will be created through spring initializer and selecting the Maven Project and language as Java. After adding all the dependencies, the basic layout of the framework will be generated. After that,we will create java classes to implement the client side with the help of multiThreading. Along with this, different Test Cases will be created to check whether the client side is working properly or not. The purpose of server side is to do the validation of all the parameters and return the response accordingly. The deployment of the server side is done with the help of oracle free cloud tier.

II. Server Implementation

The Fig. 1 represents the three different packages under which we have created different classes. The package repository consists of three classes namely customRepo, customRepoImpl and Resort Repository. The Resort Repository will inherit the features of the MongoRepository which is an interface provided by MongoDB to connect to the database.It will connect the server along with all the POST-API and all the data will be stored automatically to the database. The second package consists of Resort under which we have created three classes namely Resorts, ResortController and ResortDaoService. Resorts.java will declare all the variables and create their Getters and Setters, Constructor and toString methods. ResortController.java will create all the POST methods for the Resorts which are shown in fig. 2 and fig.3. Fig.2 contains first and second POST request and fig.3 contains third and fourth POST request.

- The first POST (@PostMapping("/check_status") method will check the status of server by taking the response Entity in the form of String and return the response of 2XX if the server is working.

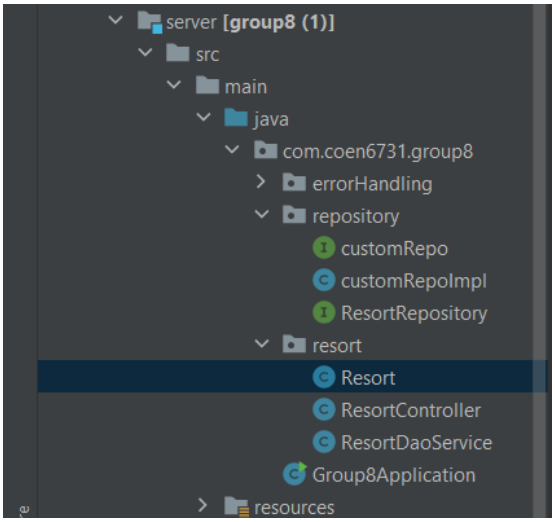

Fig. 1.

- The second POST(@PostMapping("/resorts/{resortID} /seasons") method will take the resortID in the form of URL. Its path variable will contain the resortID and in the request body,it will take Payload which will get the value of the year from the HashMap and then sends a POST Request.

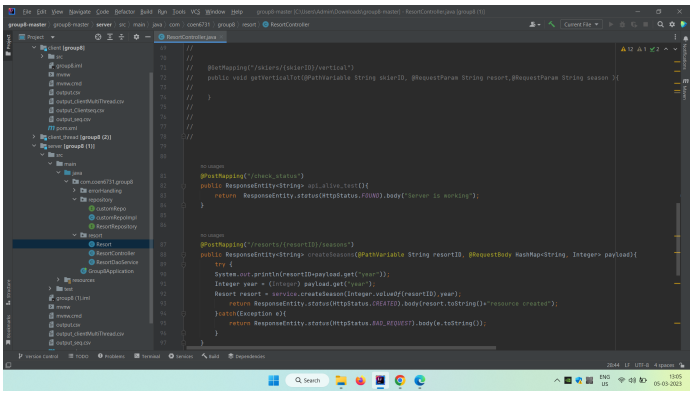

Fig. 2.

- The third POST(@PostMapping("/skiers/{resortID}/seasons /{seasonID}/days/{dayID}/skiers/{skierID}")) method will take the seasons in the form of URL. Its path

variable will contain resortId,seasonID,dayID and skierID and its request body will contain the value of the lift, which will get the value of time and liftId in the form of key-value pairs.

- The fourth POST(@PostMapping("/resorts")) request will take the URL in the form of Resorts and POST all the parameters which are defined in the resorts.java and it will POST it.

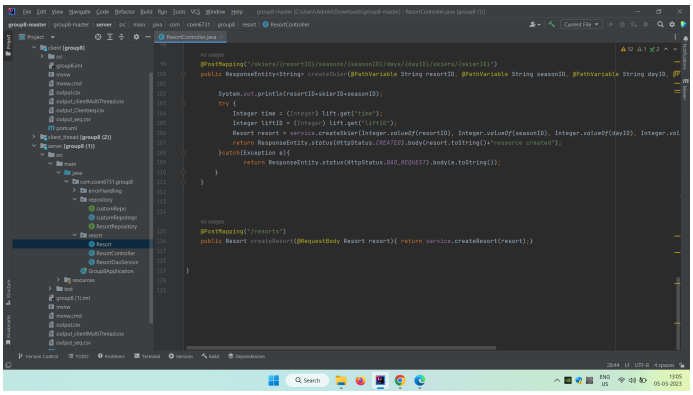

Fig. 3.

The ResortDaoService provides an abstract interface and business logic which is used to create POST methods and store all the data in the database. In this class, there will be two methods namely createSeason and createSkier respectively and these are shown in fig. 4 which will take the data from POSTMapping which is in the form of PathVariable and Request-Body and store all the data into the MongoDB.

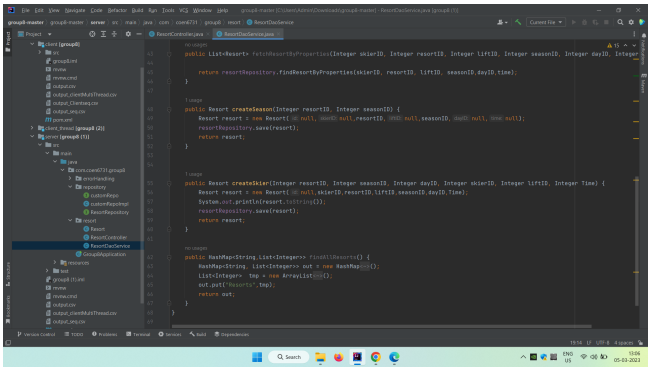

Fig. 4.

A. *Oracle cloud deployment*

The fig. 5 represents the instance information which is used in deploying the server on an oracle cloud. The server is created using Spring framework and is then deployed on the cloud. The fig. 6 shows that the server is implemented and it is running successfully on the cloud which can be depicted by the public IP address.

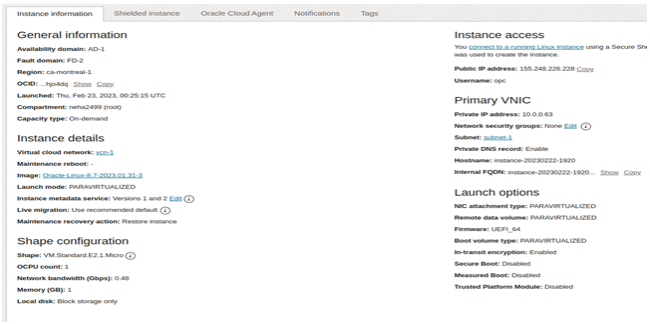

Fig. 5.

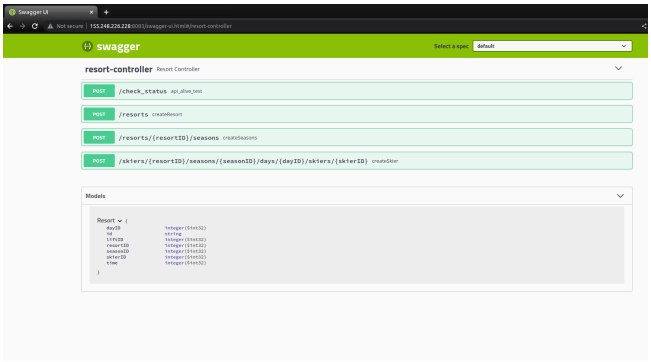

Fig. 6.

III. Client Implementation

The fig. 7 represents the implementation of the Client-side. Here, we have created two clients, one of them is Mutli-

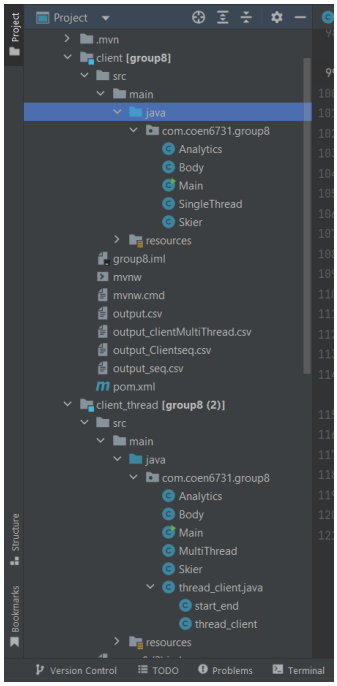

Fig. 7.

threaded and the another one is single threaded. The first Client with group8 will be single-threaded and the second one will be Multi-threaded. For the Multi threaded-client inside the package Coen6731.group8,there will be six different classes namely Analytics,Body,Main, MultiThreaded,Skier and in thread-client.java there will be two classes namely start-end and thread-client.

A. Multi Threading

- Analytics class is created to measure the profiling performance of the client side. Profiling performnace basically includes mean response time, median response time, throughput,99Th percentile, max and min response time and the response time should be in milliseconds. In the first method, we will find the percentile by multiplying the response time with 0.99 and subtracting the result from 1. In the second method, the median can be find by arranging the time in ascending order and applying the formula of median. The third method consists of finding the Max and Min response time.The fourth method will return the status from the server side which should be in the form of 2xx. The last method helps in finding the throughput.
- Body class will contain the variables which are used in the request body defined in the POST mapping. Here, we have declared two variables time and liftId which are used in the third POST mapping.
- As it was mentioned in the project description that we have generated a random data as shown in fig . 8,so, for that we will be creating a skier class in which we will declare all the variables and after that we will create a constructor and using the super() keyword, we will generate the random values for all the variables and these will be sent to the server.

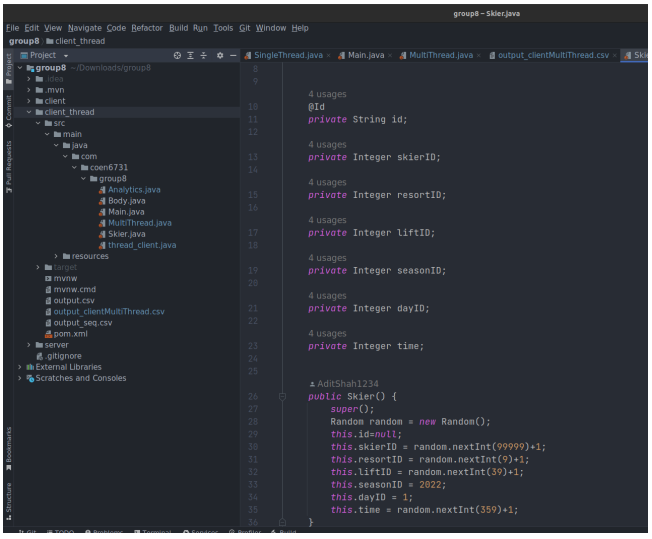

Fig. 8.

For the implementation of the Multi-Threading, we have designed each client per thread in multi-threading environment to simulate multiple client in concurrency. There are total 100 clients in total. Initially we created 32 threads for 32 clients and each client in posting 1000 response. The post implementation of the Web Client is defined in thread_client.java class. If any client out of 32 client completed the 1000 post, 68 (100 - 32) more thread will be created which means 68 more client will do the post operation. A flag is used to supervise this activity as shown in Fig 9.

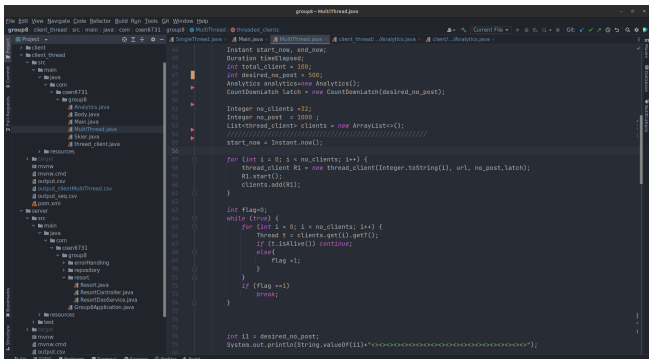

Fig. 9.

Now all the 100 client/thread will perform the post operation until the goal of 10K post has been reached. Since all the thread are posting independently, CountDownLatch class is used to set the counter for the desired number of post.

thread_client class is executed as a thread by implementing runnable interface. The run() method is also implemented which act as the starting point of all the threads and all the business logic is written in the run() method.

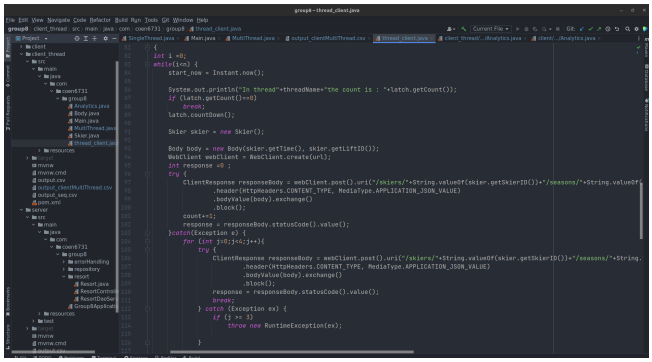

Fig. 10.

For handling we have used try/catch method as shown in Fig. 10. If the Post operation failed once, it will retry the Post request 4 times more before considering as failed request. If there are in total 5 failed request, there would be run time exception handled by the catch body.

B. Single Thread

- For the non multi-threaded client the Analytics, Body, and Skier class is same as that of Multi-threaded client

as explained above.

- For the implementation of non multi-threading, a single client is created that will do all the 10K post operation using for loop as shown in fig. 11.

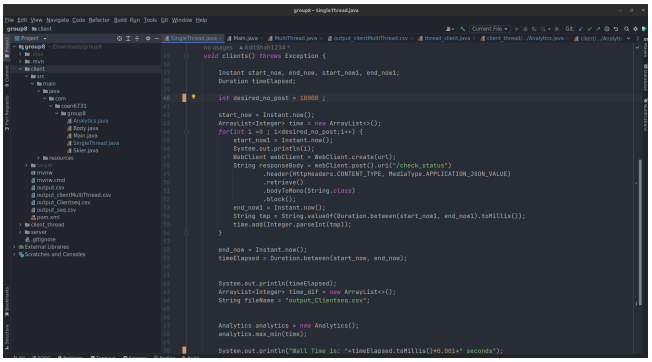

Fig. 11.

IV. Profiling Performance

The fig.12, fig.13,fig.14 represents the profiling performance of the multi-threaded client when the number of requests are 10K, multi-threaded client when the number of requests are 500 and the single-threaded client respectively.

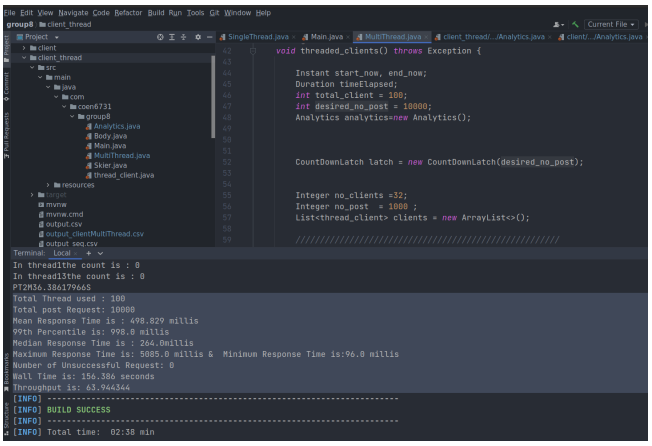

Fig. 12.

- In fig.12, it can be seen that the total number of threads used are 100 and the number of POST request send are 10,000. Accordingly, their mean response time is 498.829 milliseconds, their median response time is 264.0 milliseconds, its 99th percentile is 998.0 milliseconds,maximum response time is 5085.0 milliseconds,minimum response time is 96.0 milliseconds,its wall time is 156.386seconds and by dividing the number of post requests by wall time, the throughput will be 63.944344.
- On comparing it with the single threaded client where the same of POST requests were sent, its wall time is 233 seconds which makes its throughput to 42.91 which is less than multi threaded client.

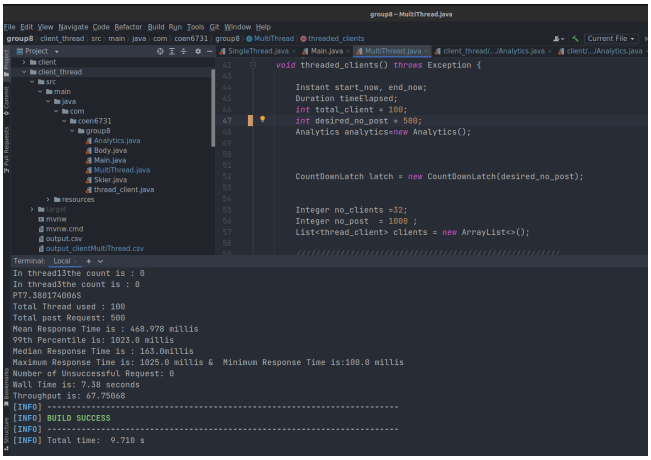

Fig. 13.

The advantages of using multi-threading are

- the system is handling the multiple requests simultaneously while the single thread will handle the requests con-concurrently.
- Through multi-Threading,the response of the server is improved and it minimizes the usage of system resources.
- it helps in better communication and above all,it throughput is always higher than that of single threaded client.

so, due to all these benefits, we prefer to use multi-threading as compared to the single threaded client.

A. Little's Law throughput predictions

As mentioned above, the throughput in multi-threaded clients for 10K post request is 63.94. When the 500 post operation is executed in the same multi-threaded client environment as shown in fig 13, the throughput is 67.750 which is comparable to the 10K post's throughput. Thus the actual throughput is close to the little law prediction.

B. Throughput within the 5 % of the client

- In the multi threaded environment where the server is handling 10K post request, the throughput of the whole system is 63.94 request per second.
- For 5 % of the client which is 500 post operation handled by multi-threaded environment, the throughput is 67.74 request per second. The percentage difference between two throughput values is 4.7 % which is in the range of $\pm 5\%$.

V. Summary

So, we have successfully created the client side,one with multi-threading and one without multi-threading and the server is deployed successfully on an oracle cloud. On comparing the throughput for single threaded and multi-threaded,we can see that the the throughput of multi-threaded client is more than the single-threaded client.

No of Clients	32	1
No of POST	100000	100000
Clients after 1 thread Completion	68	0
Implimentaton type	Multi Thread	Single Threaded
Wall Time(sec)	156	230
Throughput(post/sec)	63.94	42.91
Mean (ms)	498	233
Min (ms)	96	6.8
Meadian(ms)	264	289
Max (ms)	5085	1470
Client at 5%	67.43	39.4
99% percentile(ms)	998	134

Fig. 14.

ID	No of POST	No of Clients	Implimentaton type	Wall Time(sec)	Throughput(post/sec)
1	100000	32	Multi Threaded	156	63.94
2	500	32	Multi Threaded	7.38	67.43
3	100000	1	Single Threaded	230	42.91
4	500	1	Single Threaded	12.69	39.4

Fig. 15.

VI. References

1 https://www.oracle.com/cloud/free/always-free.
2 https://hc.apache.org/index.html.
3 https://www.elastic.co/blog/averages-can-dangerous-use-percentile.
4 https://github.com/neha2499/group8.git.