Ski Resort – Phase One

Pritam Sethuraman - 40230509
Gina Cody School of Engineering and Computer Science
Concordia University
Montreal, Canada
pritam.sethuraman@mail.concordia.ca

Rishi Murugesan Gopalakrisnan - 40200594
Gina Cody School of Engineering and Computer Science
Concordia University
Montreal, Canada
rishi.murugesangopalakrishnan@mail.concordia.ca

***Abstract* — The goal of this project is to create a distributed system that will allow the sport sector to go digital. Suppose ski resorts utilize RFID lift ticket scanners to record the time a skier rides and their skier ID each time they board a lift.**

Keywords — java, servlet, maven, oracle, jetty, multithreading.

I. Introduction

The sports industry has been experiencing a rapid digital transformation in recent years, and this project aims to contribute to this trend by developing a distributed system. Specifically, the system will utilize RFID lift ticket readers in ski resorts to record the time of every ski lift ride and the corresponding skier ID. The system will be designed to collect this data from geographically distributed resorts and store it in a central location. The collected data will then be analyzed to answer various questions related to the usage of ski lifts and the behavior of skiers. This project will start by building a client that generates and sends lift ride data to a server in the Oracle cloud as stage one of the development process.

II. Technology Overview

A. Maven

Maven is a powerful build automation tool for Java projects. It provides a standard way to manage a project's build, dependencies, and documentation. With Maven, developers can automate the process of building, testing, and deploying their applications, making it easier to manage large and complex projects.

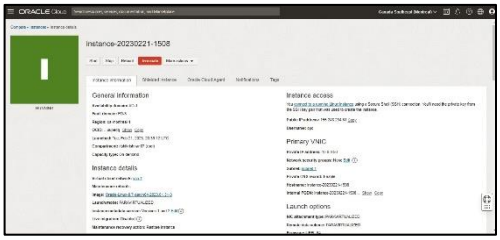
Maven uses a Project Object Model (POM) file to describe the project's structure and dependencies. The POM file is written in XML and contains information such as the project's name, version, and dependencies. Maven uses this information to automatically download the necessary dependencies and build the project according to the specified configuration.

One of the key benefits of using Maven is that it simplifies the process of managing dependencies. Maven uses a central repository of libraries and dependencies, making it easy to include and manage dependencies in your project. Additionally, Maven's plugin system allows developers to extend the functionality of the build process with minimal effort..

B. Oracle Cloud

Oracle Cloud Free Tier is a cloud computing service offered by Oracle that provides users with free access to a range of cloud resources and services. The Free Tier is designed for developers, students, and startups who want to experiment with cloud technology without incurring any costs.

The Oracle Cloud Free Tier includes several core cloud services, such as compute, storage, and networking, along with a range of other services such as databases, analytics, and developer tools. These services are available for free up to certain usage limits, which are generous enough for most small-scale projects like the one that we have implemented.

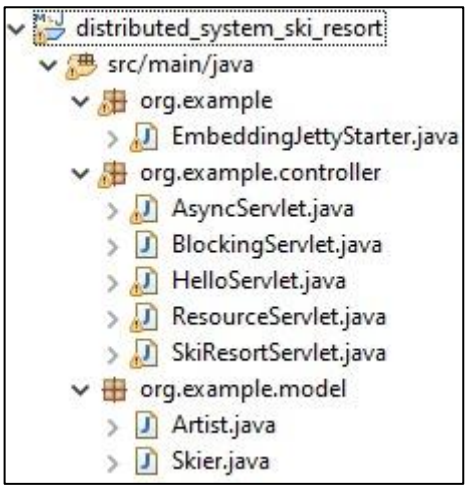


III. Implementation

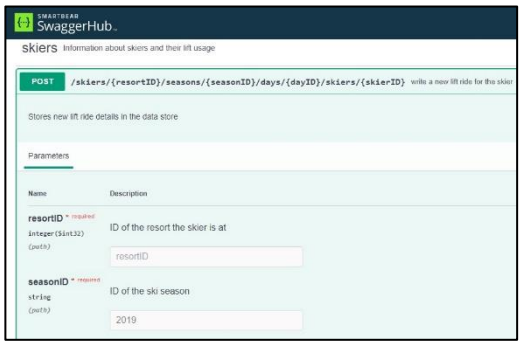
This project has been implemented using Java with the help of Maven build tool. The distributed system consists of two parts – server and client. The server side is hosted on a virtual machine running on Oracle cloud.

A. Implementing the Server API

Implementing a server using Java servlet and the Jetty framework is a popular and efficient way to build web applications. To start, you need to set up your development environment, including installing the JDK and an IDE for developing Java servlets. You also need to download and install the Jetty web server. Once your environment is set up, you can create a new Java class that extends the HttpServlet class and override the doGet() and doPost() methods to handle HTTP GET and POST requests, respectively. You can then write the logic for your server code in these methods. For this project the doGet() and doPost() methods aim to handle request from the client with skier data. The following image shows the code structure of the Java Servlet.



Here the Skier.java is the data model that represents each ski ride. It was developed by following the API documentation provided. The API documentation for skiers endpoint is given in the following image.



The parameters mentioned in this API is accepted in the doPost() method of the Java Servlet. A basic parameter validation is also been done for these parameters in the doPost() method. The following image shows the code snippet for the same.

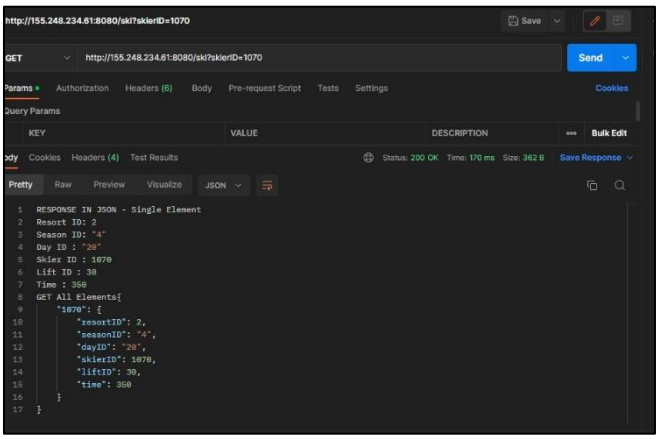
```
// BASIC PARAMETER VALIDATION

if (jsonObject == null || jsonObject.isEmpty()) {
    throw new IllegalArgumentException("The JSON object is null or empty.");
}


if (resortID <= 0) {
    throw new IllegalArgumentException("The resortID must be a positive integer.");
}

if (liftID <= 0) {
    throw new IllegalArgumentException("The liftID must be a positive integer.");
}

if (time <= 0) {
    throw new IllegalArgumentException("The time must be a positive integer.");
}
if (seasonID == null || seasonID.isEmpty()) {
    throw new IllegalArgumentException("The seasonID is null or empty.");
}

if (dayID == null || dayID.isEmpty()) {
    throw new IllegalArgumentException("The dayID is null or empty.");
}
```

Validations have been done for all the parameters and error will be thrown if POST method sends an empty string or a null value or a negative integer.

After validation the response is sent back to the client.

```
//***** SEND RESPONSE : POST METHOD *****
response.setStatus(HttpServletResponse.SC_CREATED);
response.getOutputStream().println("POST RESPONSE: Skier " + skierIDString + " is added to the database.");
```

The APIs are verified using Postman tool, the following images show the results from postman

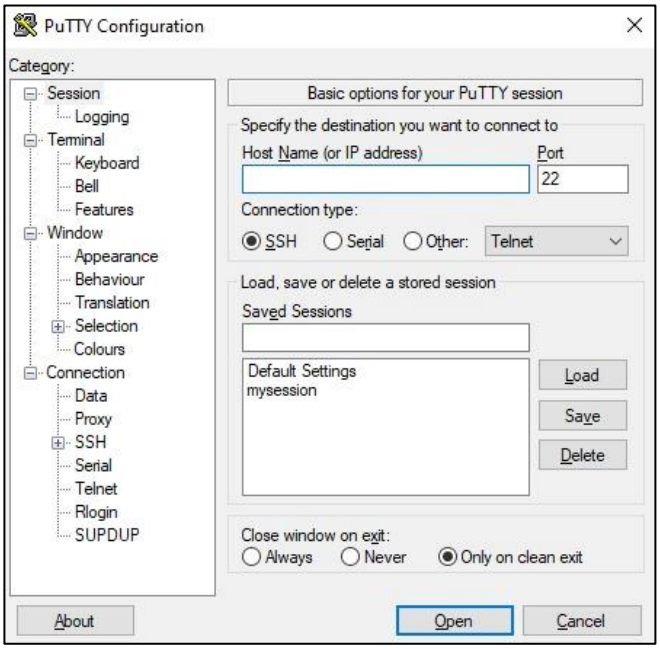


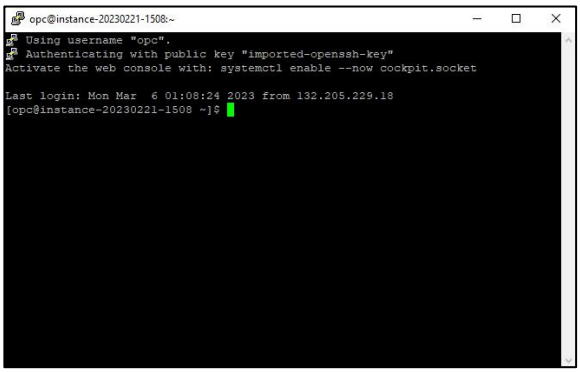
After successful implementation of the Java Servlet it is hosted in Oracle Virtual machine using Jetty.

Name	State	Public IP
instance-20230221-1508	Running	155.248.234.61

Jetty provides a lightweight and scalable web server that is easy to configure and deploy. It also includes many useful features, such as support for servlets, JSPs, and web sockets, as well as security features such as SSL/TLS. Jetty is also highly extensible, allowing developers to add custom modules to enhance the functionality of their servers. By implementing a server using Java servlet and the Jetty framework, developers can build powerful and flexible web applications with ease.

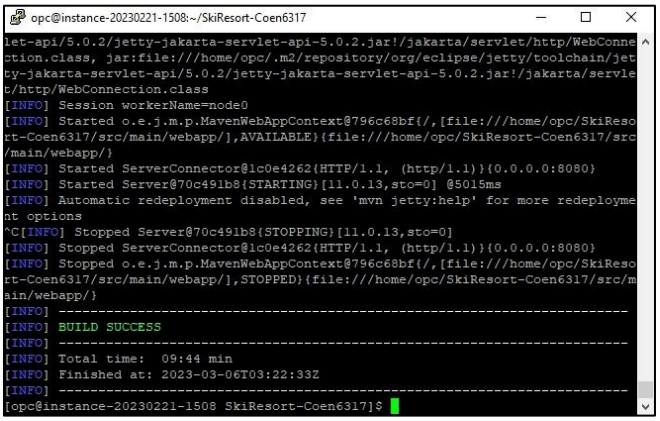
The server hosted in Oracle cloud can be hosted using Putty tool.





The above image shows the front screen for the oracle instance with the instance id - **instance-20230221-1508**.

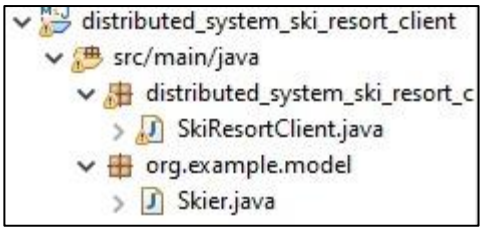
The Server instance is run using **mvn jetty:run** command. The following image depicts successful build of the server.



B. *Building the Client*

A Multithreaded Java Client was implemented to send POST request to the Server. This client is used to upload lift ride data into the server.

The following image shows the code structure of the client application.



Get() and Post() methods are written to send requests to the server.

For this phase, we have implemented the multithreaded client in the post method. As per the requirement we have done a simple test call to check connectivity.

```
public static void test() throws Exception
{
    //***** FOR TESTING CONNECTIVITY *****

    //Calling a simple GET method

    String url_get = "http://155.248.234.61:8080/ski?skierID=1070";
    get(url_get);
    System.out.println("CONNECTED TO SERVER");
}
```

C. *Data Generation*

A Random generator was implemented in the client to generate values randomly for the parameters. The following image shows the randomGenerator(). Data Generated from the random generator will be sent to the server using post request.

```
public static int[] randomGenerator()
{
    // Random Generator for Data Generation

    Random random = new Random();
    int resortID = random.nextInt(10)+1;
    int seasonID = 2022;
    int dayID = 1;
    int skierID = random.nextInt(100000)+1;
    int liftID = random.nextInt(40)+1;
    int time = random.nextInt(360)+1;
    int[] result = {resortID,seasonID,dayID,skierID,liftID,time} ;
    return result;
}
```

D. *Multithreadded Client*

The main objective of this phase is to create a multithreaded client. According to the requirements, we have implemented a 32 threads which will send 1000 POSTS requests until 10K post requests are sent to the server.

A single dedicated thread is created to generate lift data and stacked in a queue. Every other thread will execute values from the created queue.

```
//Creating a Thread pool with 32 Threads
ExecutorService executorService = Executors.newFixedThreadPool(NUM_THREADS);

// Creating a Blocking Queue that will hold the lift ride events
final BlockingQueue<int[]> queue = new LinkedBlockingQueue<int[]>();

// Create a single dedicated thread that will generate lift ride events
Thread liftRideEventGenerator = new Thread(() -> {
    //Generating lift ride events and them to queue
```

E. *Handling Errors*

Errors have been handled in the multithreaded client. If an error occurs the same request is repeated for 5 times and if not accepted, it is terminated. The following code block shows the same.

```
// ***** HANDLING ERRORS *****
int retryCount = 0;
boolean success = false;
while(!success && retryCount <5)
{
    try {

        long st = System.currentTimeMillis();


        int responseCode = post(resortID,seasonID,dayID,skierID,liftID,time);
        long et = System.currentTimeMillis();
        long latency = et-st;

        String[] metrics = {String.valueOf(st), "POST", String.valueOf(latency), String.valueOf(responseCode)};
        FileWriter csvWriter = new FileWriter(csvFilePath, true);
        csvWriter.append(String.join(",", metrics));
        csvWriter.append("\n");
        csvWriter.close();

        if (responseCode == 201)
        {
            success = true;
            int numRequestsSent = requestCount.incrementAndGet();
            if(numRequestsSent >= 1000 )
            {
                int unsuccessful = 10000-numRequestsSent;
                executorService.shutdown();
                liftRideEventGenerator.interrupt();

                // ***** On Completion *****
                long endTime = System.currentTimeMillis();
                long totalTime = endTime - startTime;
                System.out.println(" Number of Successful requests sent : " + numRequestsSent);
                System.out.println(" Number of Unsuccessful requests sent : " + unsuccessful);
                System.out.println(" Total Run Time : " + totalTime);
                double throughput = numRequestsSent/(totalTime/1000);
```

```
            }
        }
        else
        {
            retryCount++;
            Thread.sleep(1000);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
```

On Completion the program should give the following outputs, number of successful requests, number of

unsuccessful requests, total run time, and total throughput time.

```
// ***** On Completion *****
long endTime = System.currentTimeMillis();
long totalTime = endTime - startTime;
System.out.println(" Number of Successful requests sent : " + numRequestsSent);
System.out.println(" Number of Unuccessful requests sent : " + unsuccessful);
System.out.println(" Total Run Time : " + totalTime);
double throughput = numRequestsSent/(totalTime/1000);

System.out.println(" Total throughput in requests per second : " + throughput);
```

For testing and Little's Law prediction, we sent 500 requests to the server and obtained the following output on completion. Our Prediction is 83.0

```
Number of Successful requests sent : 500
Number of Unuccessful requests sent : 0
Total Run Time : 6399
Total throughput in requests per second : 83.0
```

F. Profiling Performance

After implementation of client and server side, we have done performance metrics for the same.

We have calculated latency and stored it in CSV to calculate, mean, median, throughput, 99th Percentile, min and max responses.

```
static String csvFilePath = "C:\\Users\\Admin\\Desktop\\latency.csv";
File csvWriter = new File(csvFilePath);
```

```
String[] metrics = {String.valueOf(st), "POST", String.valueOf(latency), String.valueOf(responseCode)};
FileWriter csvWriter = new FileWriter(csvFilePath, true);
csvWriter.append(String.join(",", metrics));
csvWriter.append("\n");
csvWriter.close();
```

G. GitHub Link

https://github.com/Rishi-M-G/SkiResort-Coen6317

IV. Conclusion

The stage one of the project has been implemented successfully with multithreading client. The mean and median response time were found out to be 292.4755ms and 244ms respectively. The throughput value is 90 and it is closer to our throughput prediction.

RESULTS	
Mean Response Time	292.4775
Median Response Time	244
Throughput Value	90
99th Percentile	663.99
Minimum Response Time	43
Maximum Response Time	1101