Department of Electrical and Computer Engineering

Gina Cody School of Engineering and Computer Science

ASSIGNMENT 1

COEN 6731- Distributed Software System

COURSE INSTRUCTOR:

Yan Liu

Submitted by:

Adit Shah
Id- 40172848
Email- adit.shah@mail.concordia.ca

Navaldeep Sandhu
Id- 40170635
Email- navaldeep.sandhu@mail.concordia.ca

TABLE OF CONTENTS

Table of Contents

1. Objective

The objective of Assignment 1 is to enhance skills in
a. Implementing and running a distributed architecture based on client/server model using Oracle free tier cloud virtual machine;
b. Designing services for open APIs
c. Managing concurrent operations.

2. Introduction

The client/server architecture is implemented using a Web **Servlet**. This process involves selecting the **maven project** and **Java** language, adding dependencies, and generating the framework that stores all dependencies in the pom.xml file. The server is working on jetty. The application uses web servlet Maven 4.0.2, and Java 1.8, and is hosted on the **Oracle free cloud tier**. The VM instance information is first set up to generate the public IP address and other necessary details. The Open API's documentation is created using **Swagger Hub**. For storing data, a **Concurrent HashMap** is used as an in-memory database.

Swagger URL: **https://app.swaggerhub.com/apis/ASHAH9497/Assignment1/1.0.0**

GitHub URL: https://github.com/AditShah1234/distributed_ass1.git

3. Results and Discussion

3.1. Cloud deployment

The below figure Fig.1 shows the information instance created on Oracle cloud for deployment of client-server model.

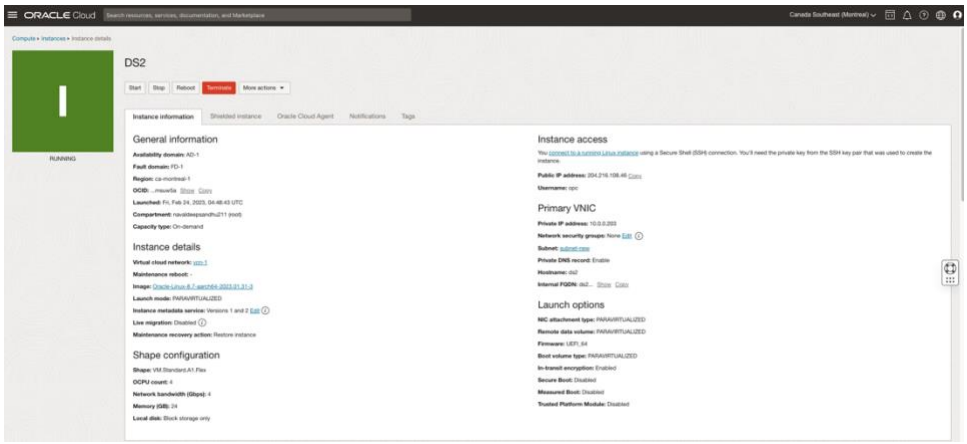

Figure 1 Oracle instance

3.2. Resource Open API design

The below figure Fig.2 shows the get () and set () methods for each data member of the class Audio.java. Open API has been defined for GET and POST methods

```
public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getTrack_title() {
    return track_title;
}

public void setTrack_title(String track_title) {
    this.track_title = track_title;
}

public String getAlbum_title() {
    return album_title;
}

public void setAlbum_title(String album_title) {
    this.album_title = album_title;
}

public String getTrack_num() {
    return track_num;
}

public void setTrack_num(String track_num) {
    this.track_num = track_num;
}

public String getNum_review() {
    return num_review;
}

public void setNum_review(String num_review) {
    this.num_review = num_review;
}

public String getNum_copy_sold() {
    return num_copy_sold;
}

public void setNum_copy_sold(String num_copy_sold) {
    this.num_copy_sold = num_copy_sold;
}
```

Figure 2 get () and set () methods

GET method is being used to obtain the required value whereas POST method is being used to save the value in server.

The POST and GET APIs have been documented through Swagger Hub, and these Open APIs have been made publicly available through the following URL for access.

https://app.swaggerhub.com/apis/ASHAH9497/Assignment1/1.0.0

3.3. API's handling in the server side using servlet

Figures 4 and 5 illustrate the procedures generated within a servlet, located in the Controller package, that manage the GET and POST requests respectively.

```java
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    String name = request.getParameter("name");
    String track = request.getParameter("track");
    String album = request.getParameter("album");
    String artist_all = request.getParameter("artist_all");
    Gson gson = new Gson();

    PrintWriter out = response.getWriter();
    response.setContentType("application/json");
    response.setCharacterEncoding("UTF-8");

    if (name == null && track == null && album == null && artist_all == null) {
        JsonElement element = gson.toJsonTree(audioMasterDB);
        JsonElement elementN = gson.toJsonTree(audioNameDB);
        JsonElement elementT = gson.toJsonTree(audioTrackDB);
        JsonElement elementA = gson.toJsonTree(audioAlbumDB);
        out.println("GET RESPONSE IN JSON - all elements " + element.toString());

        out.flush();
    } else if (artist_all != null) {

        out.println("GET RESPONSE IN JSON - all elements " + audioNameDB.keySet().toString());

        out.flush();
    } else {

        if (name != null) {
            name_record = audioNameDB.get(name);

        }
        if (track != null) {
            track_record = audioTrackDB.get(track);
        }
        if (album != null) {
            album_record = audioAlbumDB.get(album);
        }
        // System.out.println(name+track+album+name_record.toString()+track_record.toString()+album_record.toString());

        union.addAll(name_record);
        union.addAll(track_record);
        union.addAll(album_record);

        if (name == null) {
            name_record.addAll(union);
        }

        if (track == null) {
            track_record.addAll(union);
        }
        if (album == null) {
            album_record.addAll(union);
        }

        HashSet<String> intersection = new HashSet<String>(name_record); // use the copy constructor
        System.out.print(intersection.toString() + "name_record");
        intersection.retainAll(track_record);
        System.out.print(intersection.toString() + "track_record");
        intersection.retainAll(album_record);
        System.out.print(intersection.toString());
        String id = intersection.stream().findFirst().get();

        if (id != null) {
            Audio audio = audioMasterDB.get(id);

            out.println("GET RESPONSE IN JSON - single element: " + gson.toJson(audio));
            out.flush();
        } else {
            out.println("GET RESPONSE IN JSON resource not found");
            out.flush();
        }

    }
}
```

Figure 3 doGet

```java
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    String name = request.getParameter("name");
    String track = request.getParameter("track");
    String album = request.getParameter("album");
    String track_num = request.getParameter("track_num");
    String num_review = request.getParameter("num_review");
    String num_copy_sold = request.getParameter("num_copy_sold");
    last_records += 1;

    Long tmp = Long.parseLong(num_copy_sold);
    total_copies_sold_all += tmp;

    String id = String.valueOf(last_records);
    Audio audio = new Audio(id, name, track, album, track_num, num_review, num_copy_sold);

    audioMasterDB.put(id, audio);

    if (audioNameDB.containsKey(name)) {
        // HashSet<String> given_name_record =new HashSet<>();
        // given_name_record = audioNameDB.get(name);

        // given_name_record.add(id);
        // audioNameDB.put(name, given_name_record);
        audioNameDB.get(name).add(id);

    } else {

        audioNameDB.put(name, new HashSet<>());
        audioNameDB.get(name).add(id);

    }
    if (audioTrackDB.containsKey(track)) {
        audioTrackDB.get(track).add(id);

    } else {
        audioTrackDB.put(track, new HashSet<>());
        audioTrackDB.get(track).add(id);
    }
    if (audioAlbumDB.containsKey(album)) {
        audioAlbumDB.get(album).add(id);

    } else {
        audioAlbumDB.put(album, new HashSet<>());
        audioAlbumDB.get(album).add(id);
    }
    // System.out.print(audioNameDB.toString());

    response.setStatus(200);

    response.getOutputStream().println("POST RESPONSE: Audio " + name + " is added to the database. " + id
            + " total sale " + String.valueOf(total_copies_sold_all));
}
}
```

Figure 4 doPost

3.4. The concurrent Client

The below figure Fig.5 shows execution of multiple requests at one time without hampering execution of other requests.

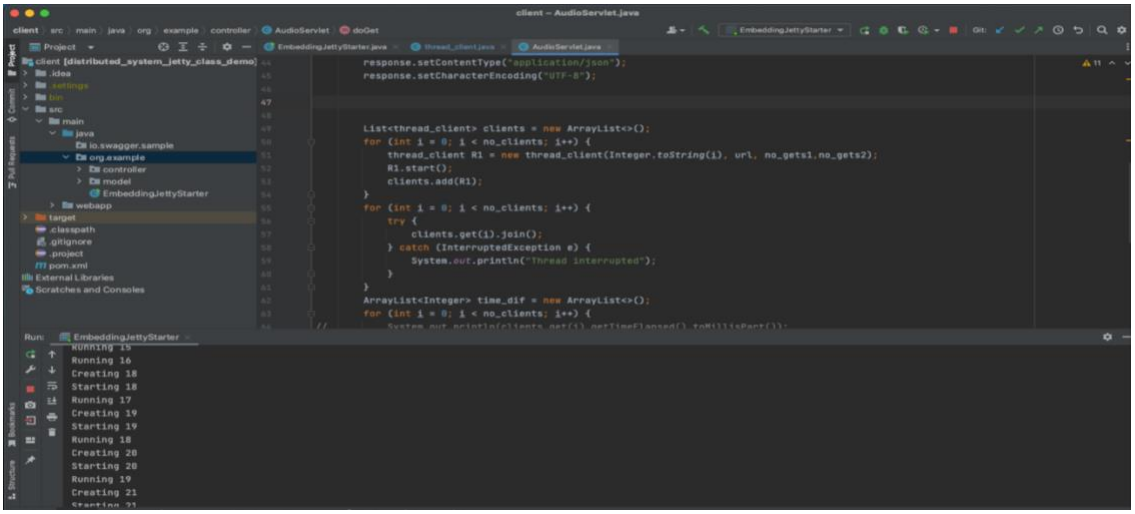

Figure 5 Multiple Requests

- Figure 6 to Figure 11 shows the simulation of GET and POST methods.

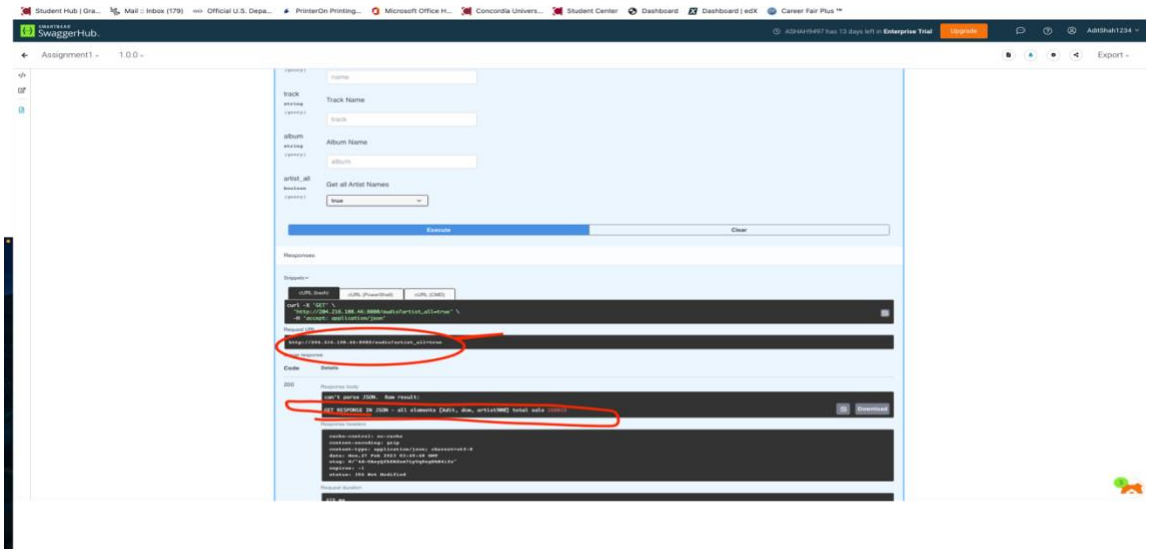

Figure 6 Swagger Documentation for GET

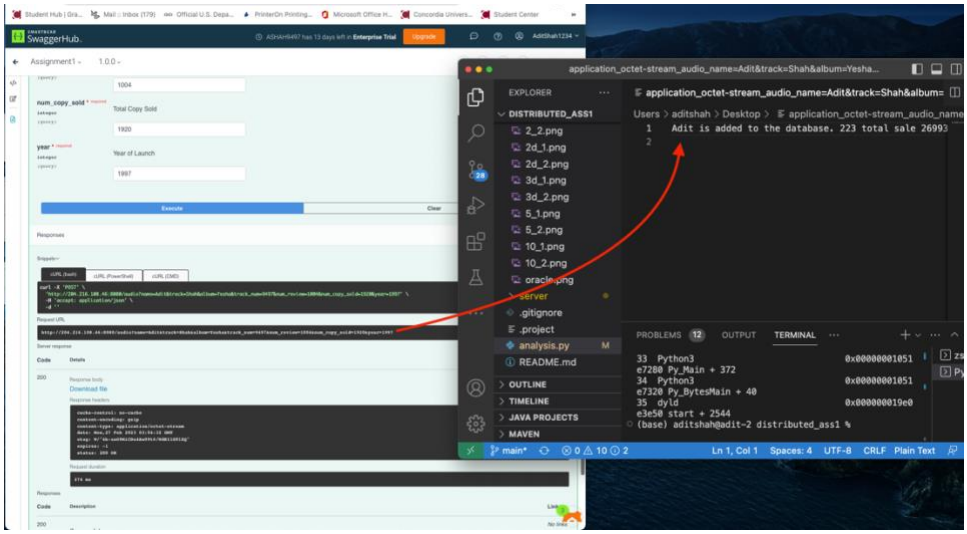

Figure 7 Swagger Documentation for POST

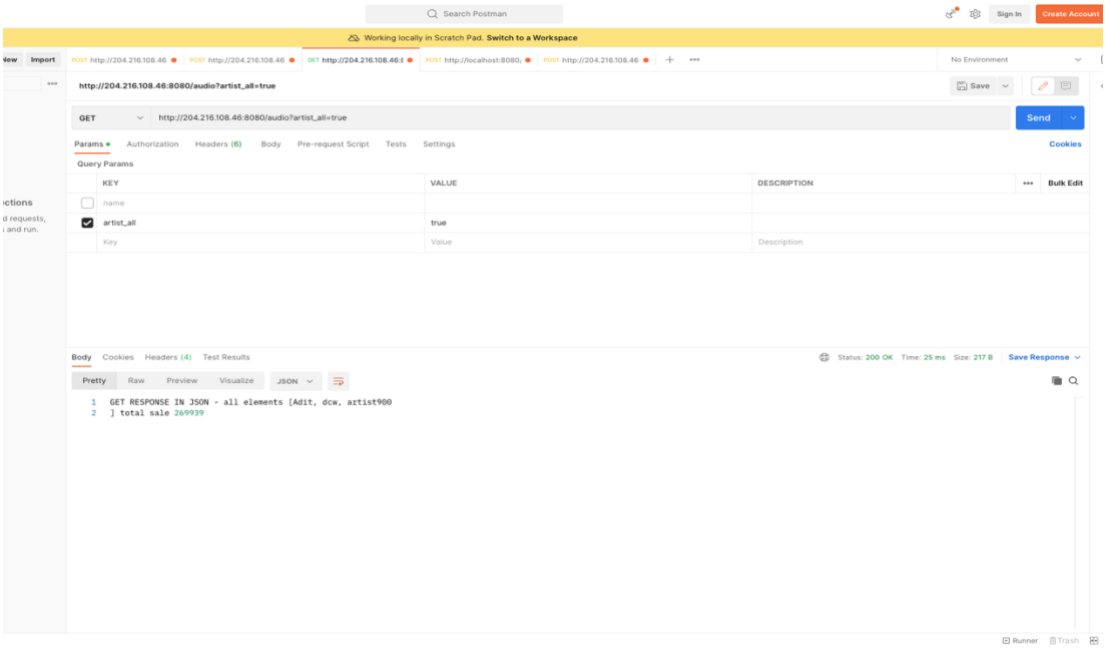

Figure 8 GET request to fetch all the artist

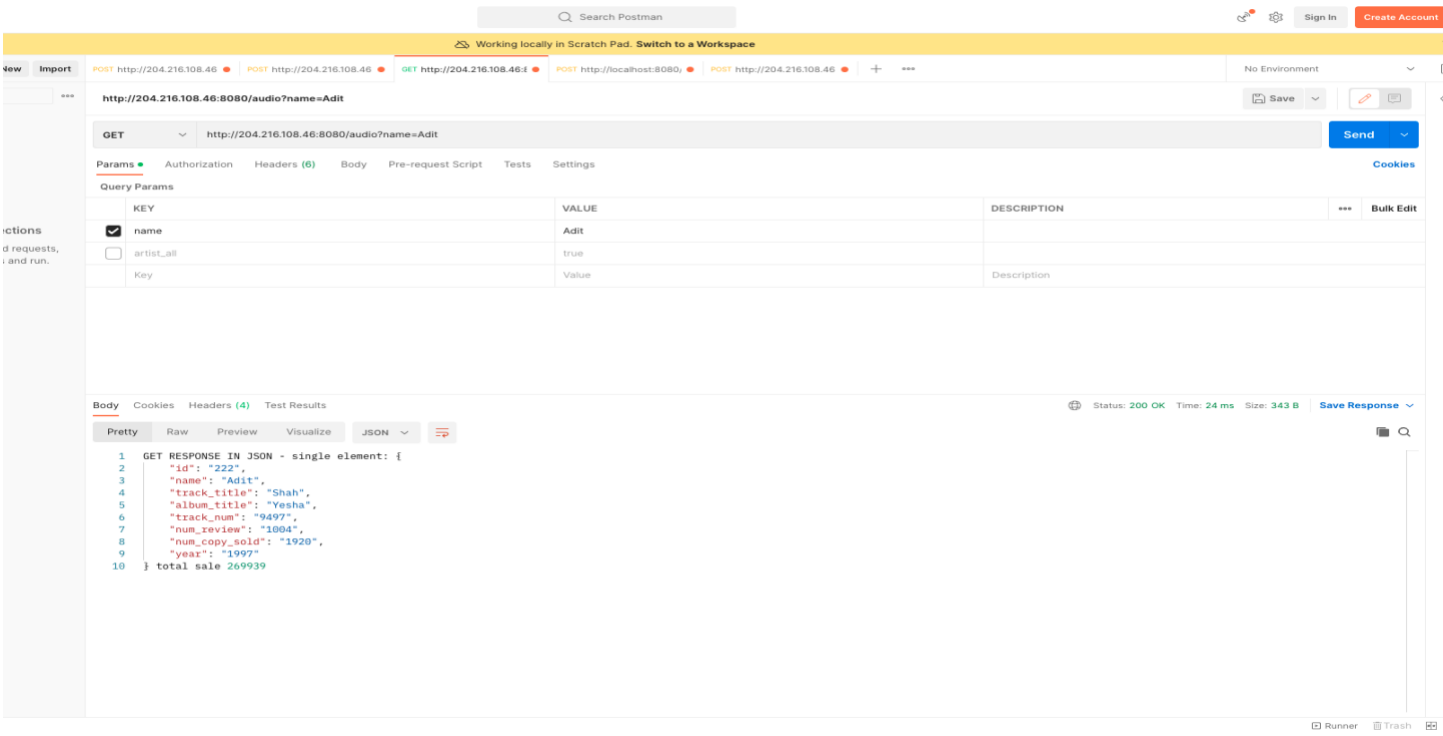

Figure 9 Get request to get record

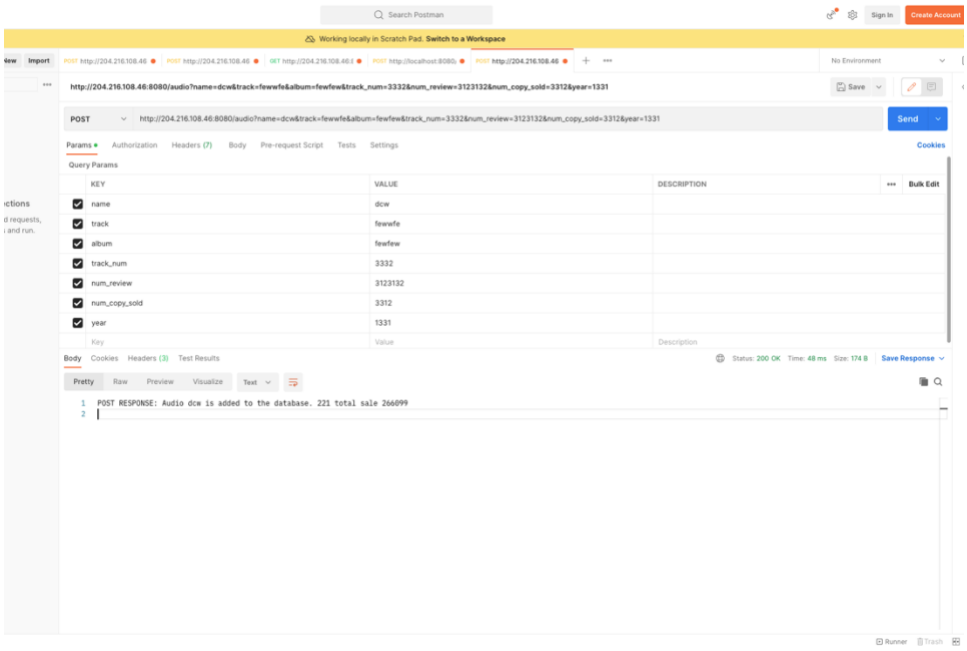

Figure 10 Post request to add to the DB

4. Results

All below graphs represent plotting of round-trip time taken with respect to number of clients given 10, 50 and 100.

The split in GET is no of request to fetch all the artist names to no of GET request to fetch records of the particular artist.

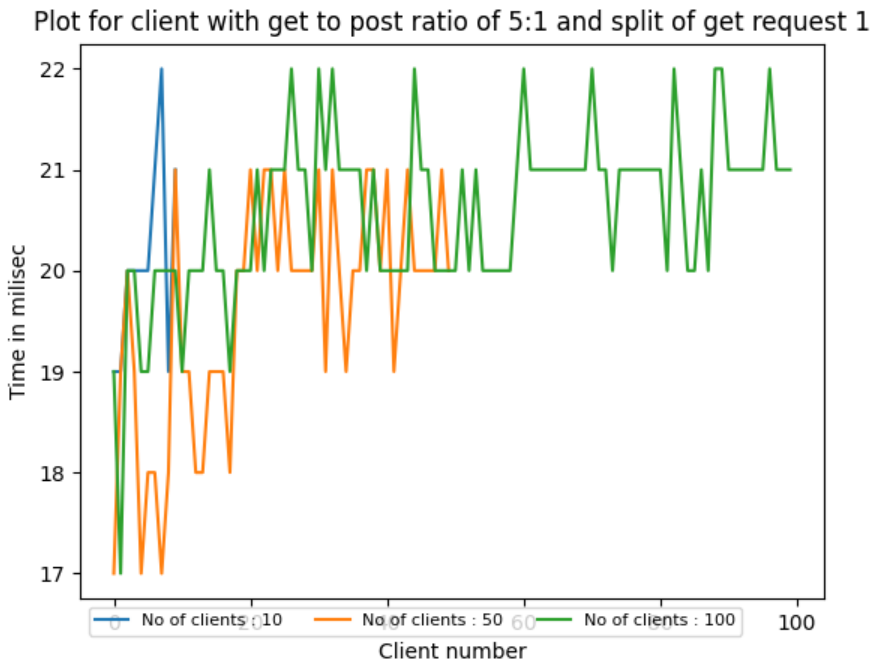

Figure 11

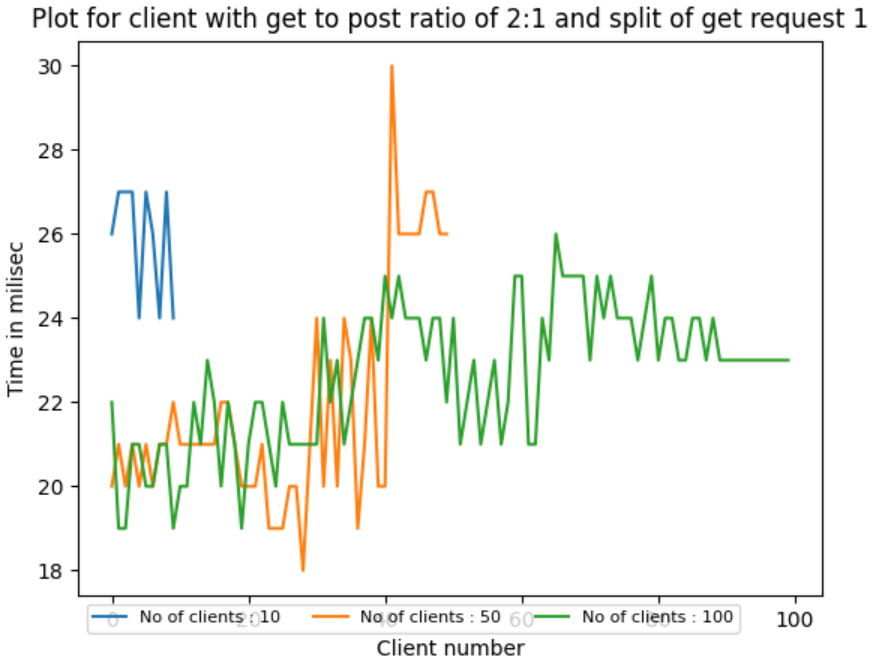

Figure 12

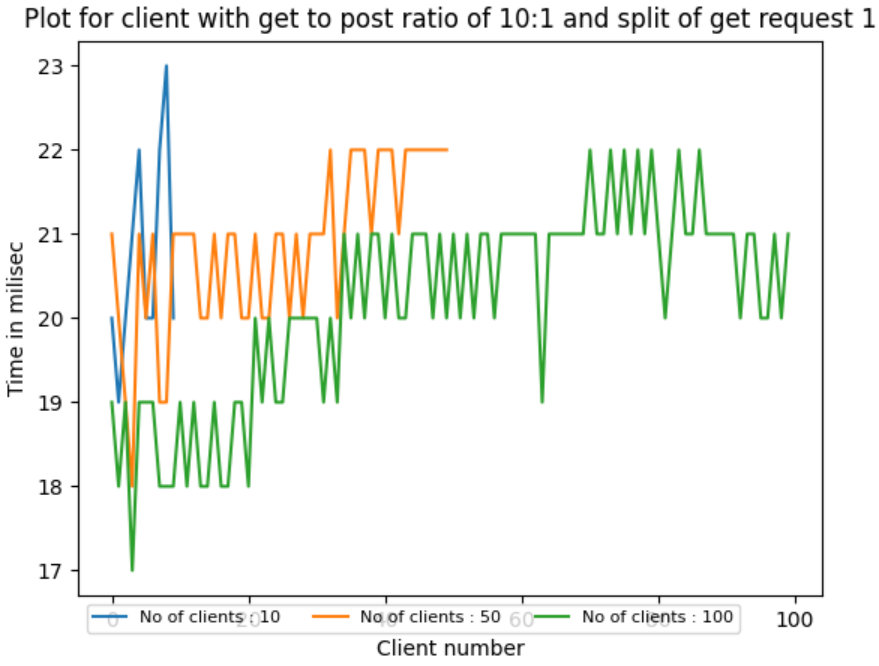

Figure 13

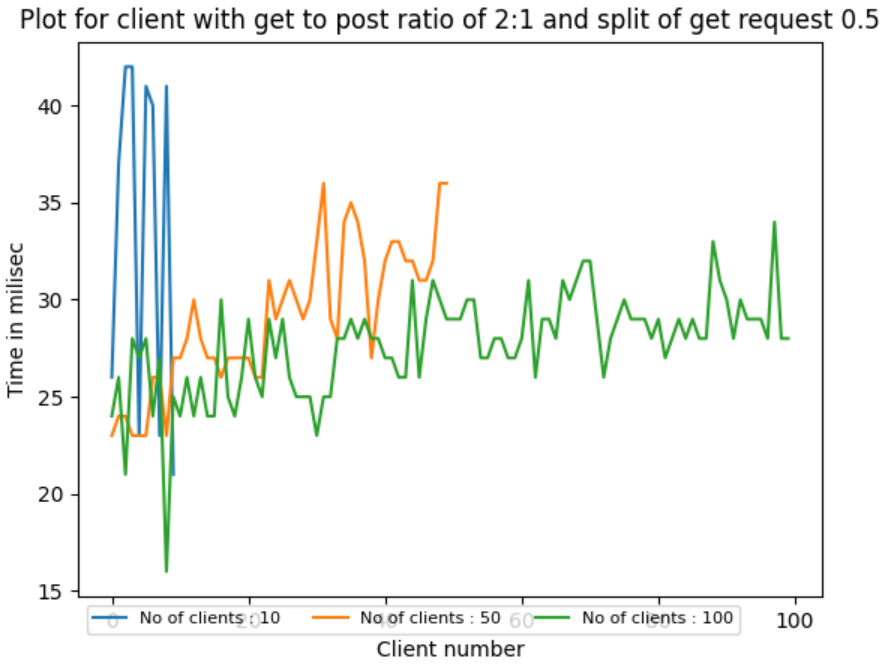
Plot for client with get to post ratio of 2:1 and split of get request 0.5
Time in milisec
Client number
No of clients : 10
No of clients : 50
No of clients : 100
40
35
30
25
20
15
100

Figure 14

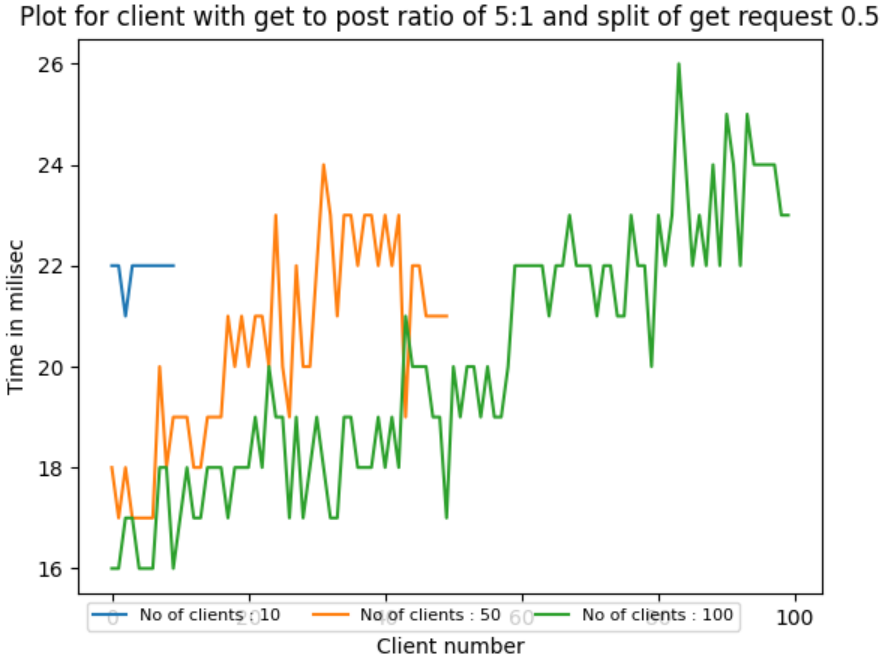

Figure 15

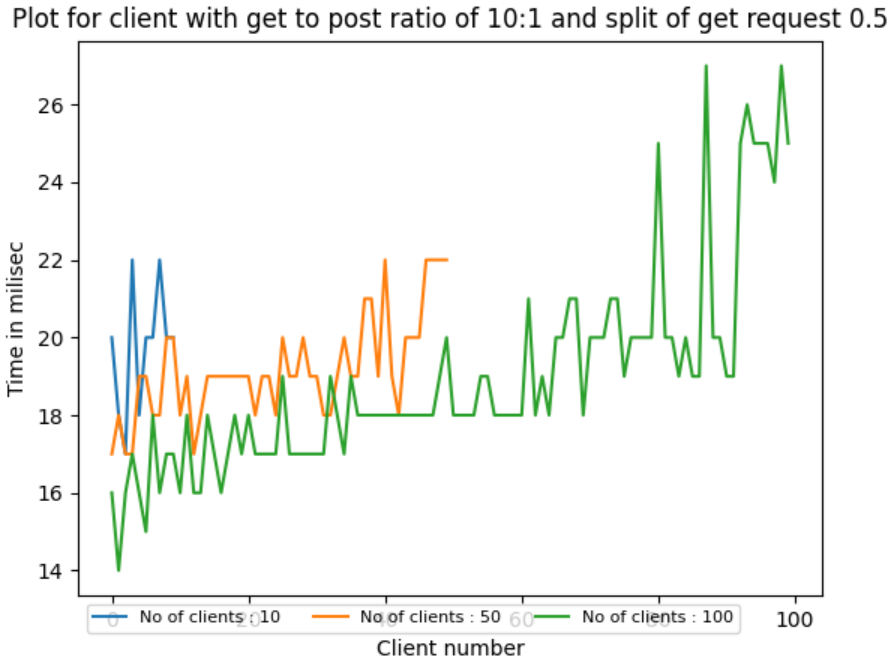

Figure 16

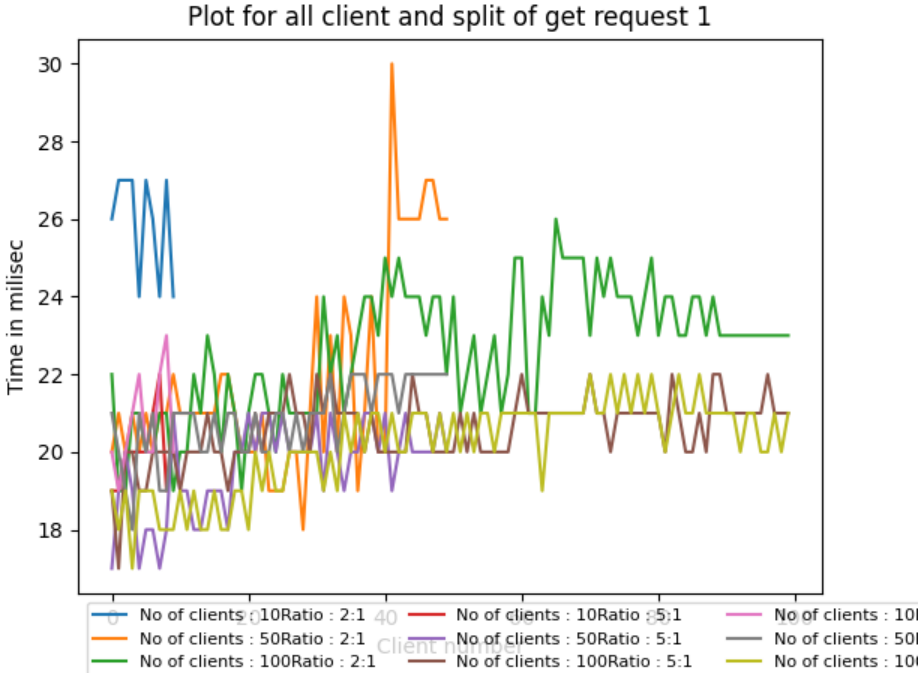

Figure 17

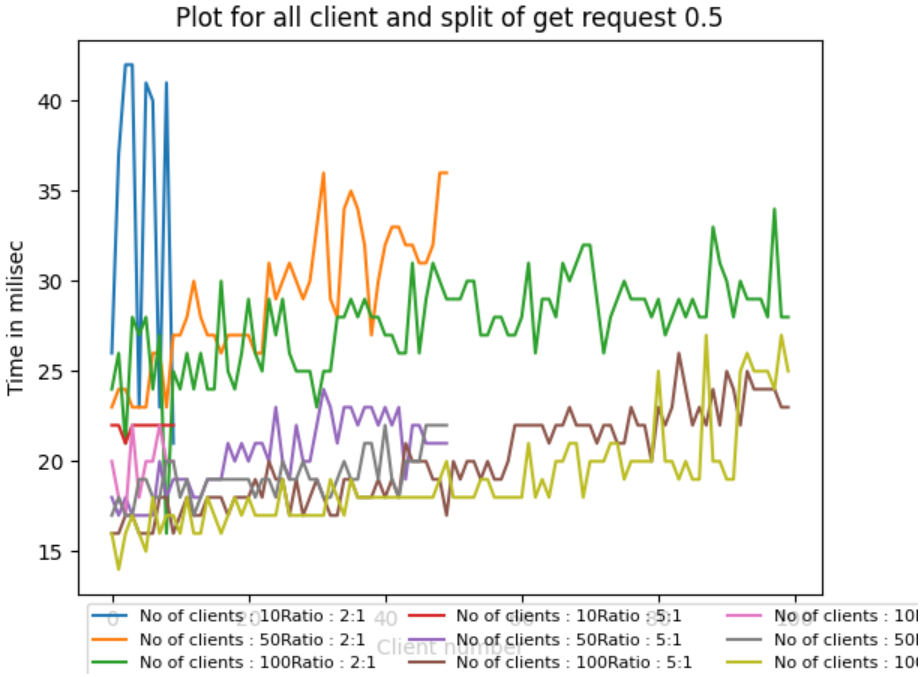

Figure 18

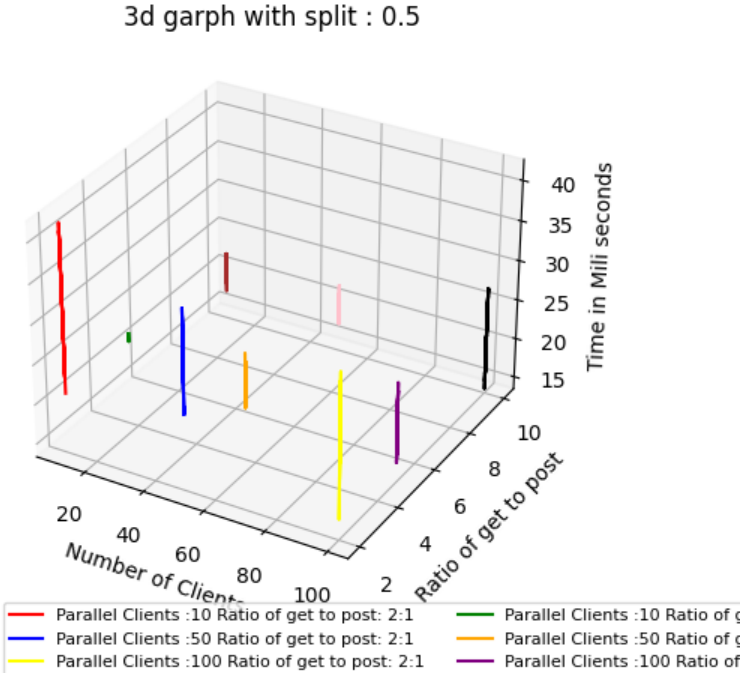

Figure 19

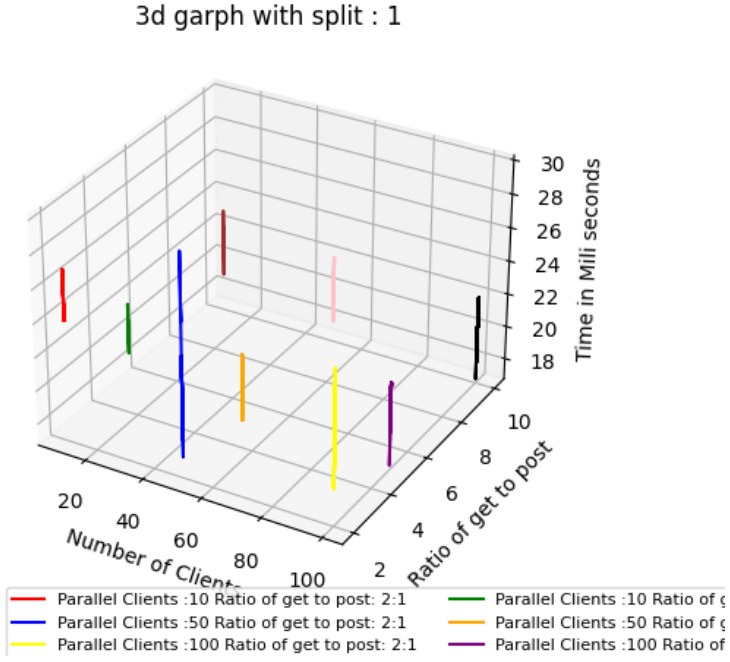

Figure 20

5. Analysis

As we can see the highest time is taken by when there is 2:1 ratio of get to post with only 2 clients in parallel. While we increase the number of 10:1 with 100 client we get the least round trip time.

6. Conclusion

The requirement of designing, developing and deployment of client-server architecture has been fulfilled. Oracle cloud has been used for hosting server. Open APIs for GET and POST method has been defined and used to retrieve and store the data respectively. Swagger Hub has been used for documentation of API. Multithreading gas been used for handling concurrent requests and graph has been plotted.

7. References

- https://github.com/youyinnn/distributed_system_jetty_helloworld
- https://www.tutorialspoint.com/servlets/index.htm
- https://swagger.io/docs/specification/describing-parameters/
- https://www.youtube.com/watch?v=TCd8QIS-2KI ;