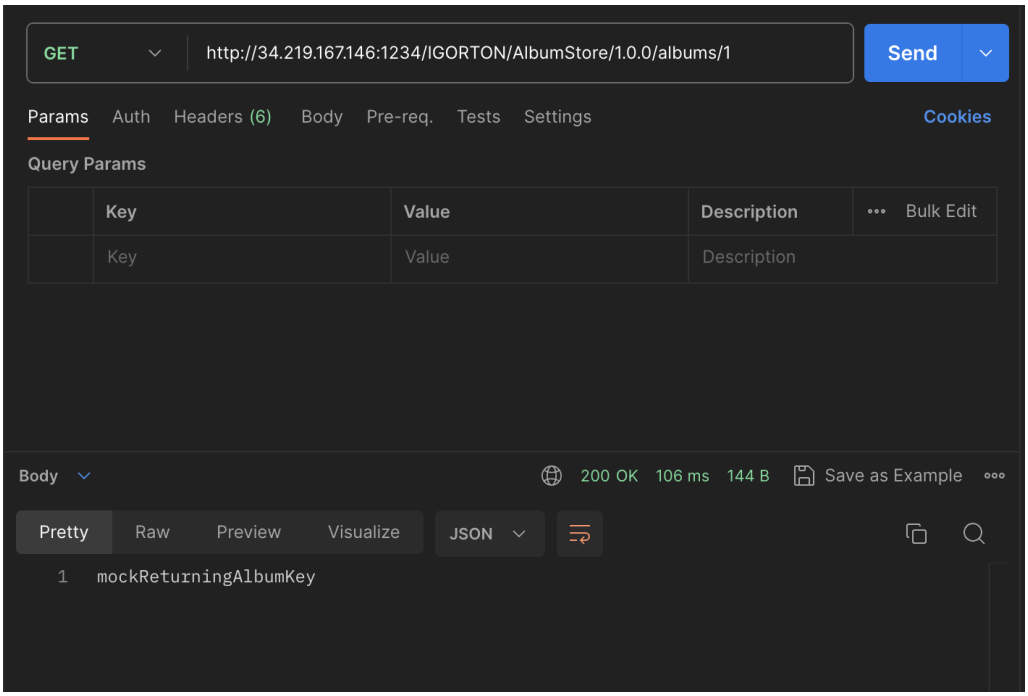
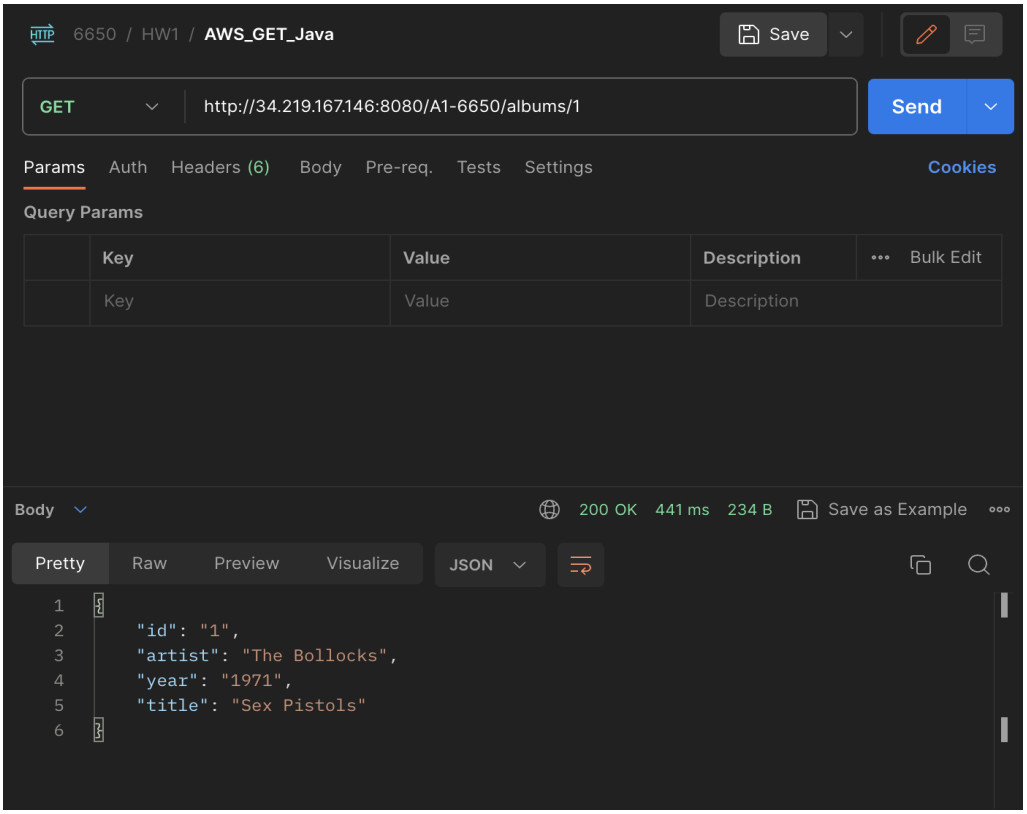
CS6650-Assignment 1

Name: Wanyue Wang
Github URL: https://github.com/wanyuew/6650#6650

- Server implementations working
    - The Go Server has been effectively deployed on AWS. Upon making GET and POST requests through Postman, the connection status indicates success, and the expected items are returned as intended. (Note that the port set for Go Server is 1234)



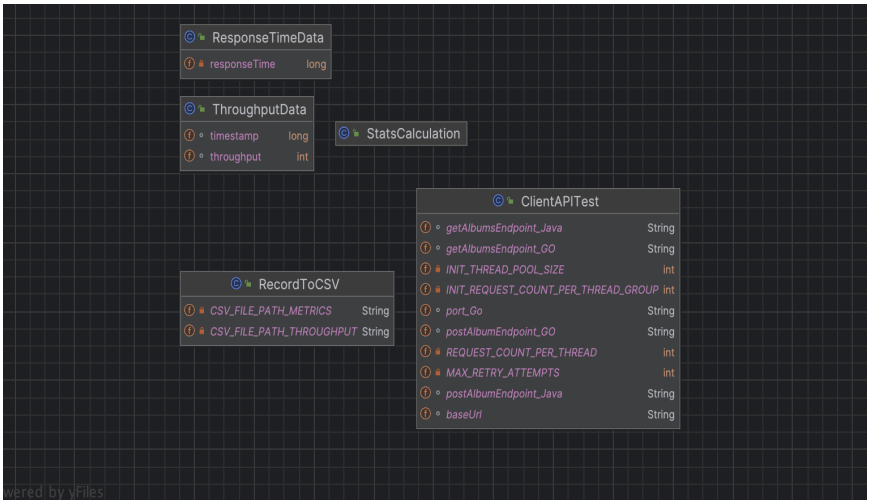
    - The Java Server has been effectively deployed on AWS. Upon making GET and POST requests through Postman, the connection status indicates success, and the expected items are returned as intended. (Note that the port set for Java Server is 8080)



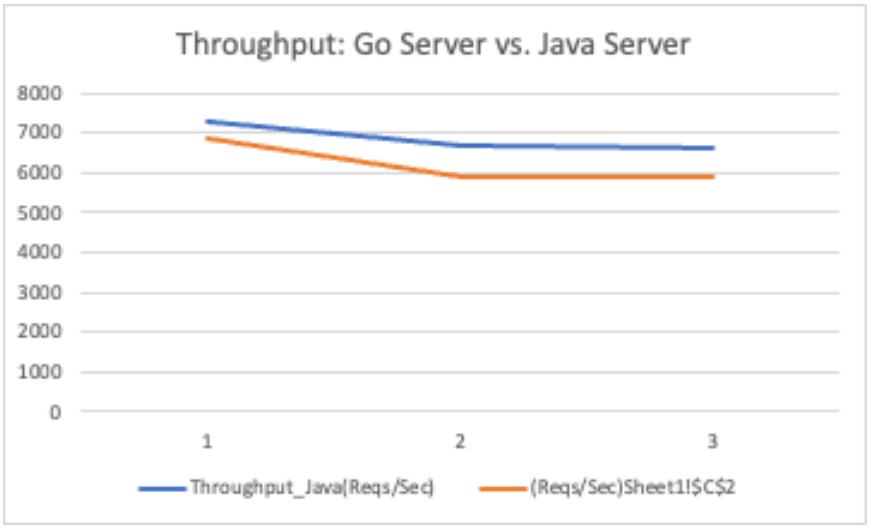
- Client design description (5 points) - clarity of description, good design practices used
    - Client1
        - The class ClientAPI runs threads for sending requests
        - The runThreads function executes a specified number of threads, each of which performs a certain number of HTTP requests. (Excerpt of the Screenshot as below)

```java
private static void runThreads(int numThreads, String url, int numRequests) {
    Thread[] threads = new Thread[numThreads];
    String port = url.substring( beginIndex: url.length() - 4);
    String postUrl;
    String getUrl;
    if (port.equals(port_Go)) {
        postUrl = url + postAlbumEndpoint_GO;
        getUrl = url + getAlbumsEndpoint_GO;
    } else {
        postUrl = url + postAlbumEndpoint_Java;
        getUrl = url + getAlbumsEndpoint_Java;
    }
    for (int i = 0; i < numThreads; i++) {
        threads[i] = new Thread(() -> {
            HttpURLConnection postConnection = null;
            HttpURLConnection getConnection = null;
            try{
                postConnection = createConnection(postUrl, method: "POST");
                getConnection = createConnection(getUrl, method: "GET");
                for (int j = 0; j <numRequests; j++) {
                        sendPostRequest(postUrl, postConnection);
                        sendGetRequest(getUrl, getConnection);
                }
            }catch (IOException e) {
```

- Client2
  - The classes ResponseTimeData and ThroughputData are utilized to define objects representing response time and throughput data, respectively. They serve as data structures to organize and manage key metrics (response time and throughput) collected during the load testing of the Java and Go servers.
  - The class ClientAPITest contains the entry point of threads running and sending requests in Client2
  - The class RecordToCSV is where the functions for loading data into CSV files
  - The class StatsCalculation encapsulate the functions for calculating the mean/median/p99/max/throughput

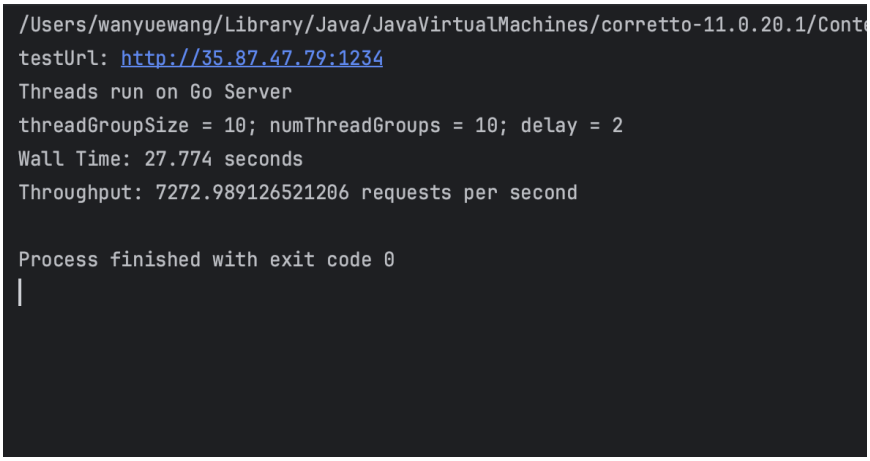


- Client Part 1 - Output window showing best throughput. Somewhere around 2k/sec should be a minimum target at higher loads.
    - Chart illustrating the throughput comparison between Java Server and Go Server.
        - Go Server exhibits a higher throughput compared to the Java Server under the same set of parameters. Additionally, both servers demonstrate a gradual decrease in throughput as the request load increases.

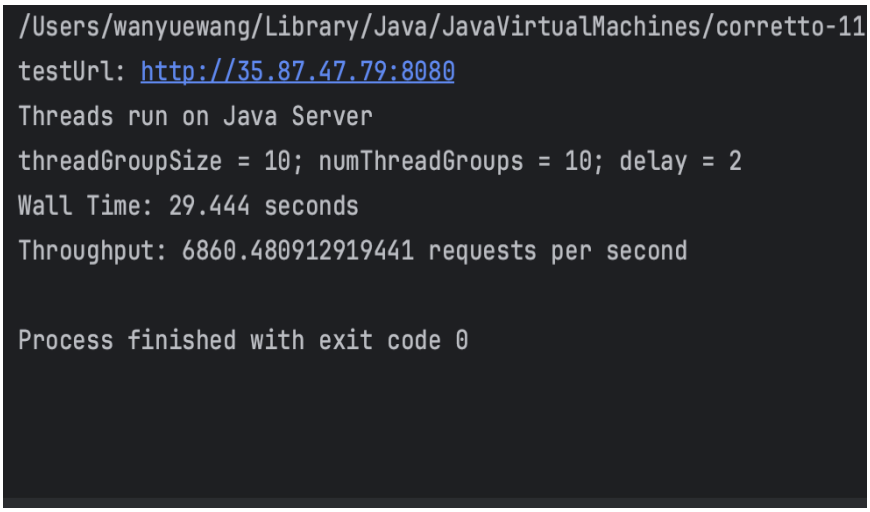


    - threadGroupSize = 10, numThreadGroups = 10, delay = 2

- Go Server



- Java Server

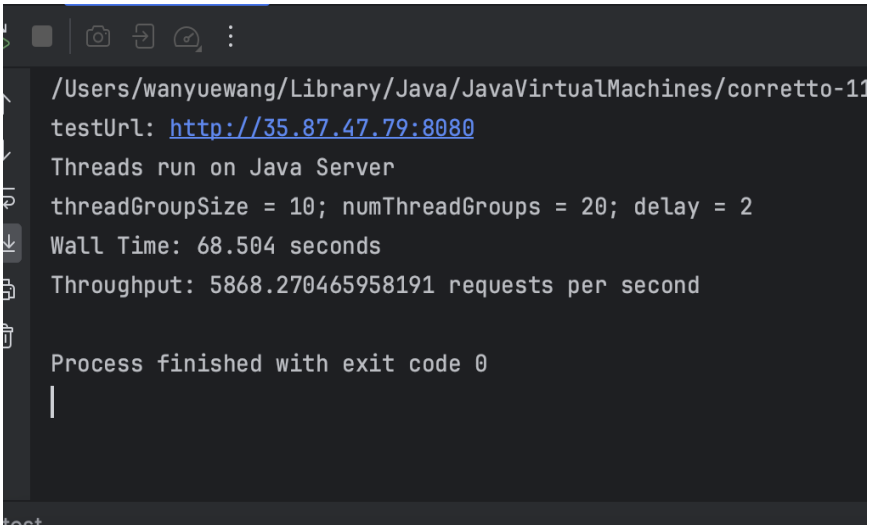


- threadGroupSize = 10, numThreadGroups = 20, delay = 2
  - Go Server

```
/Users/wanyuewang/Library/Java/JavaVirtualMachines/corretto-11.0.20.
testUrl: http://35.87.47.79:1234
Threads run on Go Server
threadGroupSize = 10; numThreadGroups = 20; delay = 2
Wall Time: 60.233 seconds
Throughput: 6674.082313681869 requests per second

Process finished with exit code 0
```

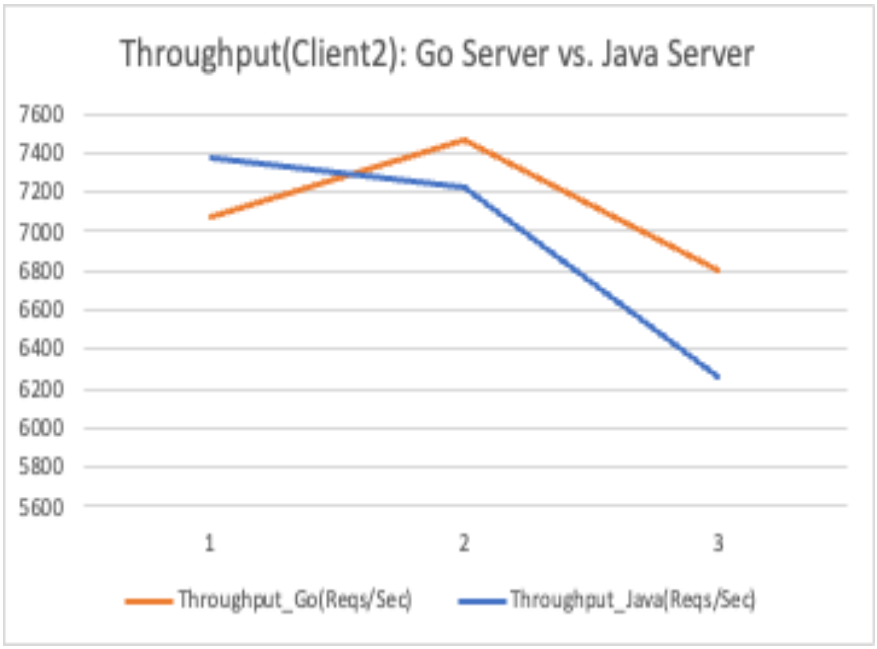
- Java Server



- threadGroupSize = 10, numThreadGroups = 30, delay = 2
  - Go Server

- Java Server

- Client Part 2 - (10 points) - 5 points for throughput within ~5% of Client Part 1. 5 points for calculations of mean/median/p99/max/throughput (as long as they are sensible).
  - Chart illustrating the throughput comparison between Java Server and Go Server.
  - Noted that Go server mostly has a higher throughput than Java Server



- Calculations of mean/median/p99/max/throughput
  - Go Server

```
testUrl: http://34.219.167.146:1234
Mean: 740.1181818181818 milliseconds
Median: 528 milliseconds
p99: 3095.0 milliseconds
Min: 250 milliseconds
Max: 3096 milliseconds
Threads run on Go Server
threadGroupSize = 10; numThreadGroups = 10; delay = 2
Wall Time: 28.559 seconds
Throughput: 7073.076788402955 requests per second

Process finished with exit code 0
```

```
testUrl: http://34.219.167.146:1234
Mean: 559.2809523809524 milliseconds
Median: 420 milliseconds
p99: 1582.0 milliseconds
Min: 201 milliseconds
Max: 1587 milliseconds
Threads run on Go Server
threadGroupSize = 10; numThreadGroups = 20; delay = 2
Wall Time: 53.888 seconds
Throughput: 7459.916864608076 requests per second
```

```
testUrl: http://34.219.167.146:1234
Mean: 706.5967741935484 milliseconds
Median: 467 milliseconds
p99: 2365.0 milliseconds
Min: 170 milliseconds
Max: 2606 milliseconds
Threads run on Go Server
threadGroupSize = 10; numThreadGroups = 30; delay = 2
Wall Time: 88.439 seconds
Throughput: 6806.951684211717 requests per second

Process finished with exit code 0
```

- Java Server

```
testUrl: http://34.219.167.146:8080
Mean: 611.7272727272727 milliseconds
Median: 478 milliseconds
p99: 1472.0 milliseconds
Min: 214 milliseconds
Max: 1475 milliseconds
Threads run on Java Server
threadGroupSize = 10; numThreadGroups = 10; delay = 2
Wall Time: 27.361 seconds
Throughput: 7382.771097547604 requests per second

Process finished with exit code 0
```

```
testUrl: http://34.219.167.146:8080
Mean: 560.3761904761905 milliseconds
Median: 441 milliseconds
p99: 1409.0 milliseconds
Min: 242 milliseconds
Max: 2200 milliseconds
Threads run on Java Server
threadGroupSize = 10; numThreadGroups = 20; delay = 2
Wall Time: 55.679 seconds
Throughput: 7219.957254979436 requests per second

Process finished with exit code 0
```

```
testUrl: http://34.219.167.146:8080
Mean: 853.1322580645161 milliseconds
Median: 548 milliseconds
p99: 3262.0 milliseconds
Min: 168 milliseconds
Max: 4358 milliseconds
Threads run on Java Server
threadGroupSize = 10; numThreadGroups = 30; delay = 2
Wall Time: 96.255 seconds
Throughput: 6254.2205599709105 requests per second

Process finished with exit code 0
```

- Step 6: Plot of throughput over time (5 points)
    - Below is the plot of throughput of Go Server with arguments: threadGroupSize = 10, numThreadGroups = 30, delay = 2