

A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date.

26-9-2022

Firepass – tests

tests

Several thin, curved lines in dark blue and light grey originate from the bottom left and sweep upwards and to the right.

Heiko Morales –Ryan Smith, Edris Rahimi,
FIREPASS - TESTS

INDEX

- 1. TESTING FIREPASS - 2 -
 - 1.1. WEB PAGE TEST - 4 -
 - 1.1.1. Tool..... - 4 -
 - 1.1.2. Results - 5 -
 - 1.1.3. Conclusion - 7 -
 - 1.2. REST API TEST - 8 -
 - 1.2.1. Tool..... - 8 -
 - 1.2.2. Results - 8 -
 - 1.2.3. Conclusion - 8 -
 - 1.3. FIREPASS DATABASE - 9 -
 - 1.3.1. Tool..... - 9 -
 - 1.3.2. Results - 10 -
 - 1.3.3. Conclusion - 10 -

Image Index

IMAGE 1 LOTUS CONFIGURATION	- 4 -
IMAGE 2 LOTUS LOAD TEST	- 4 -
IMAGE 3 FIREPAS PAGE STATUS TEST	- 5 -
IMAGE 4 REQUESTS, RESPONSES AND NUMBER OF USERS PER SECOND	- 5 -
IMAGE 5 100 USERS STATUS TEST	- 6 -
IMAGE 6 TOTAL REQUESTS PER SECOND	- 6 -
IMAGE 7 RESPONSE TIMES IN MS.....	- 6 -
IMAGE 8 NUMBER OF USERS	- 6 -
IMAGE 9 300 USERS STATUS TEST.....	- 7 -
IMAGE 10 300 USERS LOAD TESTING	- 7 -
IMAGE 11 PETITIONS RESULTS	- 8 -
IMAGE 12 STRESS APP.....	- 9 -
IMAGE 13 RESULT OF THE FIRTS TEST OF 100	- 10 -
IMAGE 14 RESULT OF THE LOAD TEST	- 10 -

1. TESTING FIREPASS

In this document we will perform different types of tests on different firepass services in order to ensure a solid and efficient infrastructure.

Software testing is the empirical and technical investigation aimed at providing objective and independent information on the quality of the product to the interested party or stakeholder. It is a further activity in the quality control process.

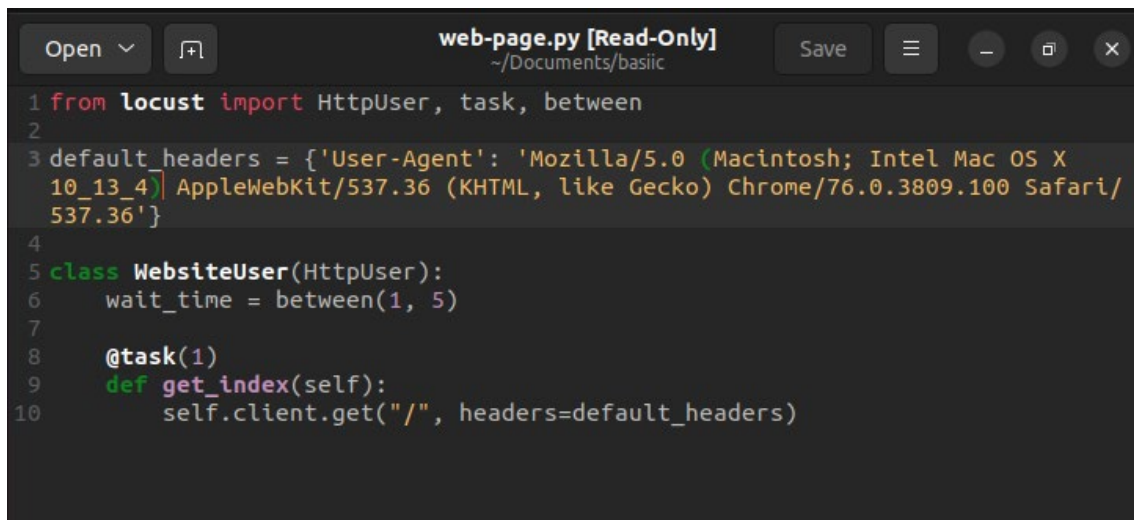
In the case of firepass we will test the 3 most important components of the infrastructure. That is to say, the tests will be for the website, the Rest-API and the database.

1.1. WEB PAGE TEST

The website is one of the keys for customers to contact us in an easy way, that's why we have carried out a load test.

1.1.1. TOOL

For this service we tested our case study web app using Lotus. I started by installing Lotus in to a ubuntu instance. After it was installed, I created a python script to run with Lotus.

A screenshot of a code editor window titled 'web-page.py [Read-Only]' with the file path '~/.Documents/basic'. The editor shows a Python script for configuring a Locust load test. The script imports 'HttpUser', 'task', and 'between' from 'locust'. It defines a 'default_headers' dictionary with a 'User-Agent' string. A 'WebsiteUser' class inherits from 'HttpUser' and sets 'wait_time = between(1, 5)'. A task named 'get_index' is defined with a '@task(1)' decorator, which calls 'self.client.get("/", headers=default_headers)'.

```
1 from locust import HttpUser, task, between
2
3 default_headers = {'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X
4 10_13_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.100 Safari/
5 537.36'}
6
7
8 class WebsiteUser(HttpUser):
9     wait_time = between(1, 5)
10
11     @task(1)
12     def get_index(self):
13         self.client.get("/", headers=default_headers)
```

Image 1 Lotus configuration

I checked the local host and the dashboard was active.

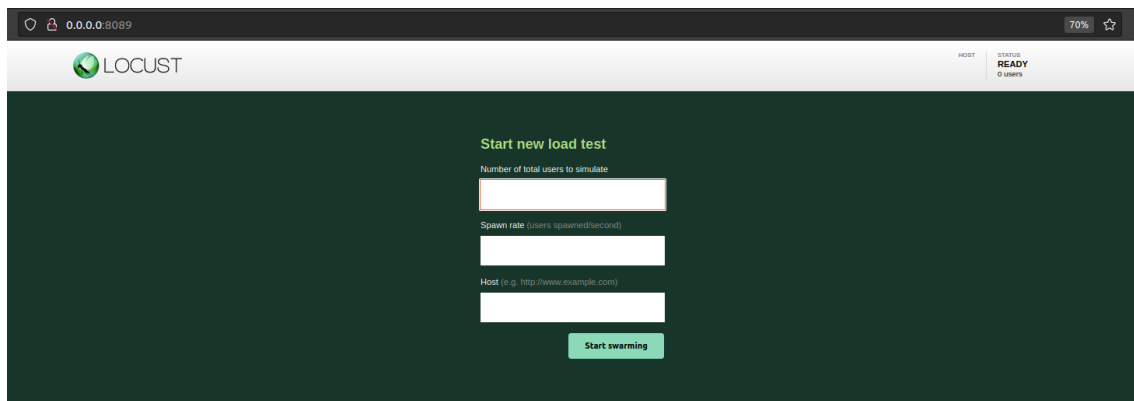


Image 2 Lotus load test

I then targeted the firepass web app and tested how to app would handle different amounts of users.

1.1.1.2. RESULTS

10 users, 2/s:

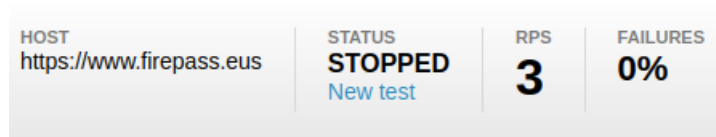


Image 3 Firepas page status test



Image 4 requests, responses and number of users per second

100 users, 2/s:

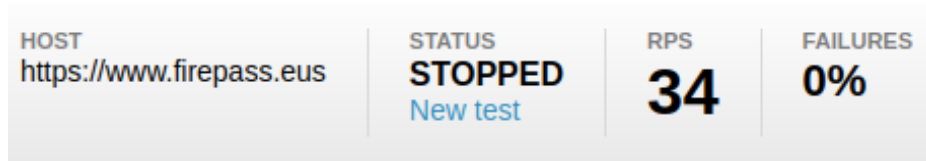


Image 5 100 users status Test

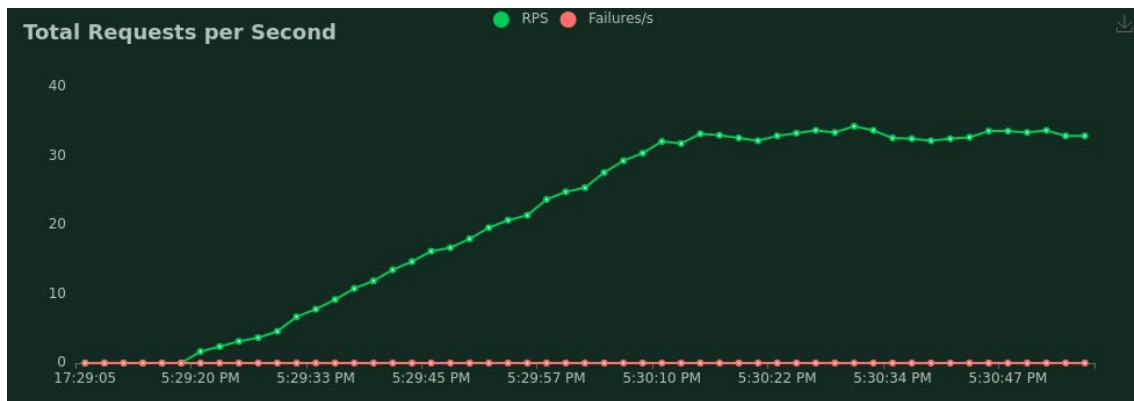


Image 6 Total requests per second

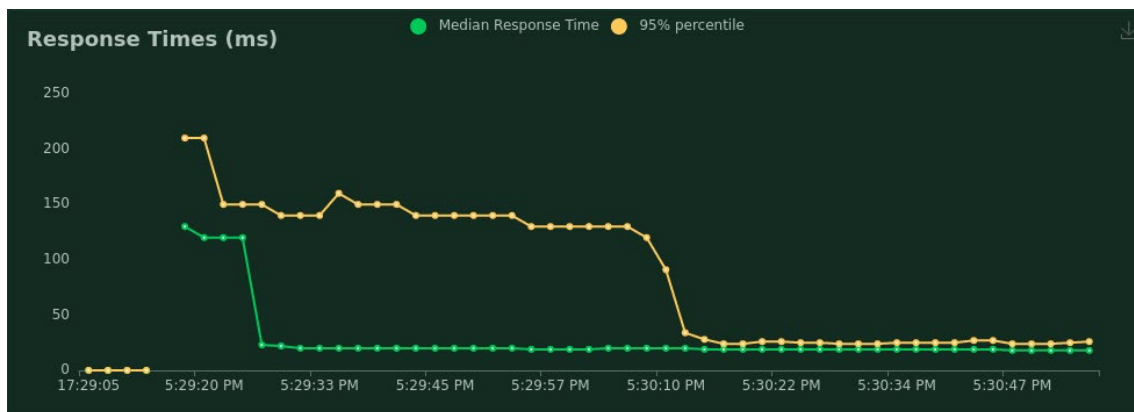


Image 7 Response times in ms

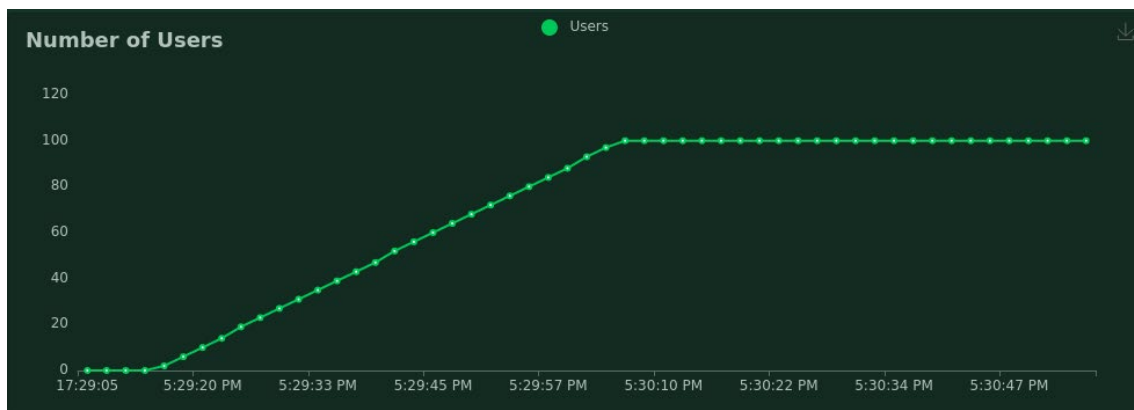


Image 8 Number of users

300 users, 2/s:

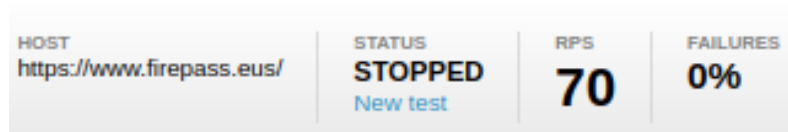


Image 9 300 users status test

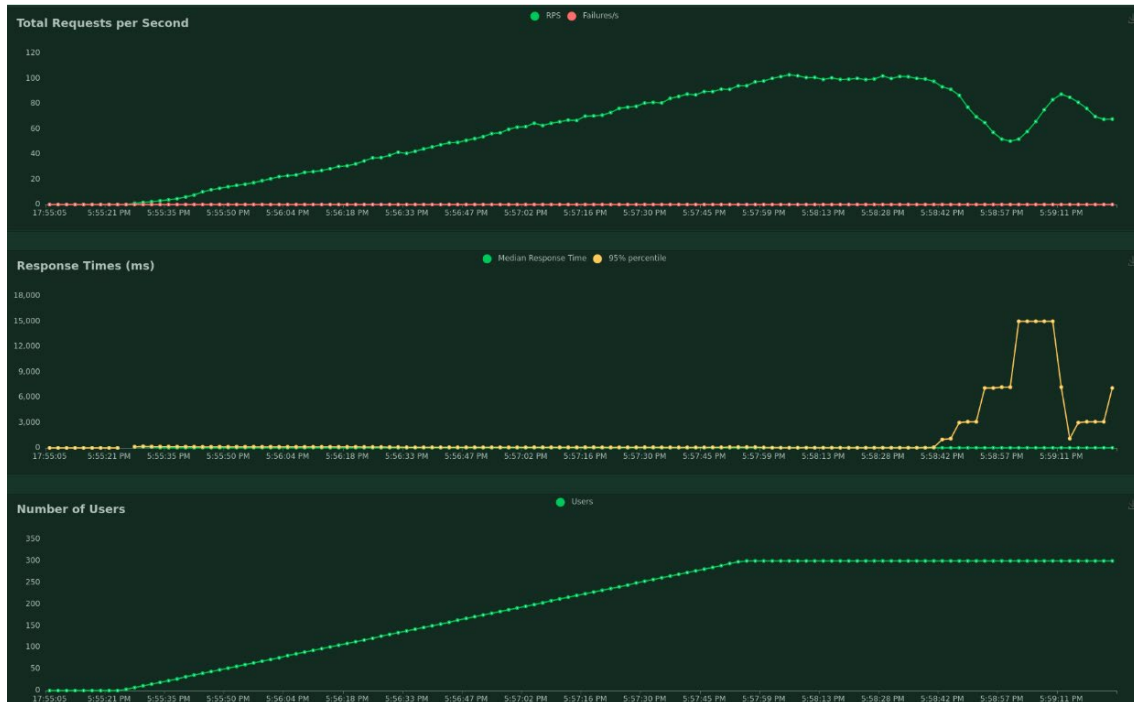


Image 10 300 users load testing

1.1.3. CONCLUSION

The website shows a fairly normal behaviour. As more requests are made per second its performance will decrease, but the user cannot really notice this. It's true that the slowest part is the sending of the mail, but as it is sent at the end, the workload is not noticeable.

1.2. REST API TEST

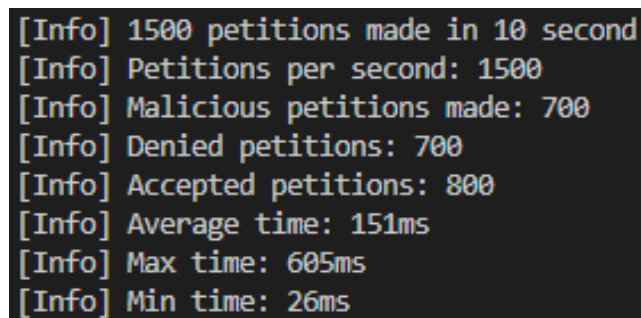
Rest-api is the gateway to firepass services. Therefore, in order to test the api we have decided to generate a load of many users at the same time. With this load we evaluate if the api is able to filter the requests. The function of the api is to filter the requests based on who has validated credentials.

1.2.1. TOOL

The tool used is a tool created by us in java. This programme consists of a variable number of threads that simulate users. We can simulate a load of thousands of users to the api, some with credentials, others without. With this we measure how many requests we can handle per second at most.

1.2.2. RESULTS

1500 requests were made with 700 incorrect credentials:



```
[Info] 1500 petitions made in 10 second  
[Info] Petitions per second: 1500  
[Info] Malicious petitions made: 700  
[Info] Denied petitions: 700  
[Info] Accepted petitions: 800  
[Info] Average time: 151ms  
[Info] Max time: 605ms  
[Info] Min time: 26ms
```

Image 11 Petitions results

1.2.3. CONCLUSION

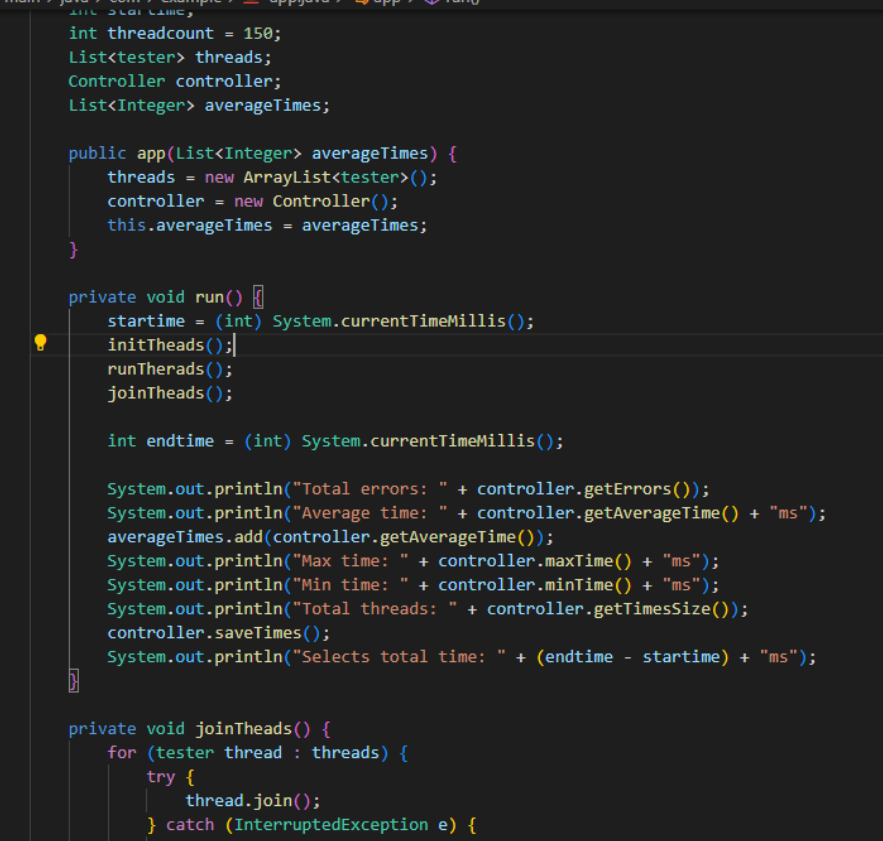
In the case of the api we have conclude that after 1000 requests the response time increases dramatically. The api is in charge of filtering who has credentials and who does not. It is true that it is not a lot of work but creating multiple sessions at the same time prevents the response time to be low. Therefore, we come to the conclusion that from multiple simultaneous communications the requests are slower and make the end user wait.

1.3. FIREPASS DATABASE

The database is one of the most critical things in the system. It is responsible for the encrypted storage of all the credentials of the different users.

1.3.1. TOOL

The tool used is a tool created by us in java. This program consists of a variable number of threads that simulate requests to the database. In other words, we can simulate a load of thousands of requests to the database, both reading and writing.



```
src > main > java > com > example > app.java > app > run()
11
12     int threadcount = 150;
13     List<tester> threads;
14     Controller controller;
15     List<Integer> averageTimes;
16
17     public app(List<Integer> averageTimes) {
18         threads = new ArrayList<tester>();
19         controller = new Controller();
20         this.averageTimes = averageTimes;
21     }
22
23     private void run() {
24         starttime = (int) System.currentTimeMillis();
25         initTheads();
26         runTheads();
27         joinTheads();
28
29         int endtime = (int) System.currentTimeMillis();
30
31         System.out.println("Total errors: " + controller.getErrors());
32         System.out.println("Average time: " + controller.getAverageTime() + "ms");
33         averageTimes.add(controller.getAverageTime());
34         System.out.println("Max time: " + controller.maxTime() + "ms");
35         System.out.println("Min time: " + controller.minTime() + "ms");
36         System.out.println("Total threads: " + controller.getTimesSize());
37         controller.saveTimes();
38         System.out.println("Selects total time: " + (endtime - starttime) + "ms");
39     }
40
41     private void joinTheads() {
42         for (tester thread : threads) {
43             try {
44                 thread.join();
45             } catch (InterruptedException e) {
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\heiko\Desktop\testing> ^C
PS C:\Users\heiko\Desktop\testing>
```

Image 12 Stress app

1.3.2. RESULTS

In this first case, we have carried out a load test:

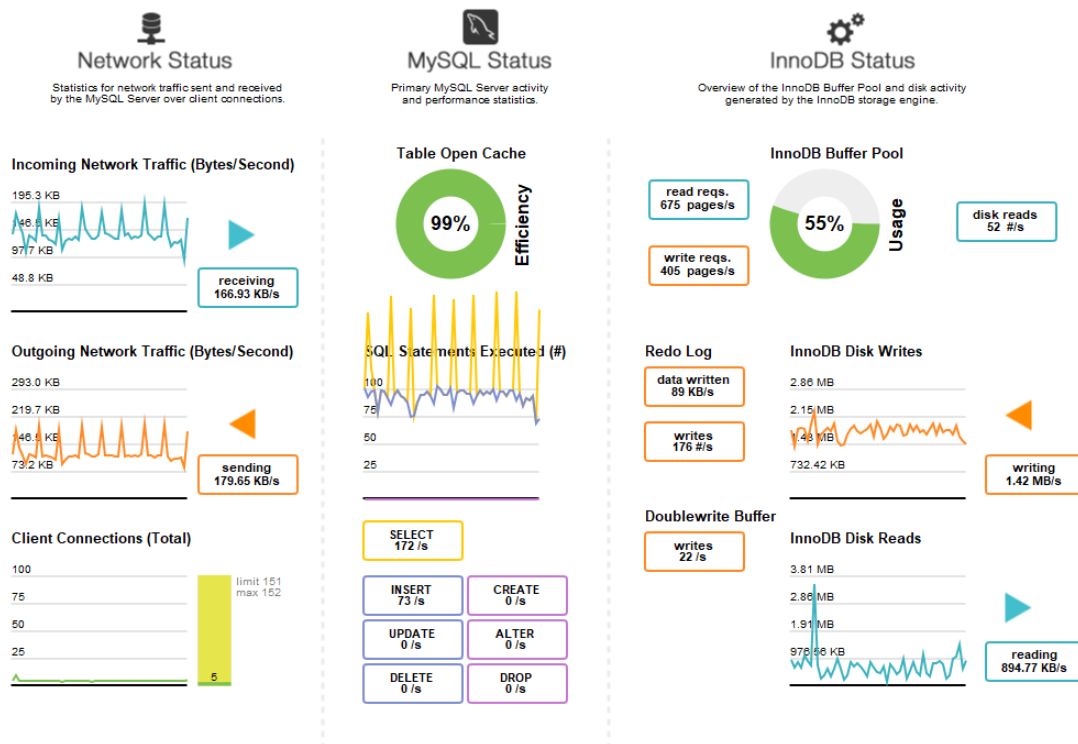


Image 14 Result of the load test

```
[INFO] Time expend on inserts: 21150ms
Total errors: 0
Average time: 271ms
Max time: 496ms
Min time: 74ms
Total threads: 150
2023_01_11_11_08_00
Selects total time: 584ms
```

Image 13 Result of the firts test of 100

1.3.3. CONCLUSION

In the case of the database, we find that read requests are extremely slow compared to read requests. This is evident because when writing, it needs to access and save on the disk. And this is a very slow operation. On the other hand, it is worth noting the efficiency of the database in resolving requests. More than 100 simultaneous requests were made, and the average time went down as you ran the tests. This is due to the data being stored in cache, which is a much faster access memory than disk.