# TP2: TeaStore Performance Efficiency

Group 02

Chun-An Bau
2165883
chun-an.bau@polymtl.ca

Youssef Amine BenDiha
2113648
youssef-amine.ben-diha@polymtl.ca

Jakub Profota
2165556
jakub.profota@polymtl.ca

## Contents

**Abstract**

This document further elaborates on the quality plan for one of the subsystems of TeaStore [1], focusing on the Performance quality characteristic of ISO 25010 standard. The latest version of TeaStore is evaluated based on presented quality metrics and compared with previous versions.

## I. Introduction

In this paper, we continue evaluating our quality assurance plan on the Authentication service of the TeaStore, this time focusing on one specific quality characteristic of ISO 25010: Performance. The Authentication service performance efficiency is crucial as this service is responsible for user account logins and logouts. These accounts contain purchase history. Not being able to keep up with an increased load may cause severe issues like ignoring user purchases. Therefore, extensive system testing under load is mandatory.

## II. Quality Plan

### A. Quality Goals

The specified quality characteristics and their respective sub-characteristics of the Authentication subsystem are presented in Table I. We updated the table with the newly evaluated Performance characteristic.

In the Performance quality characteristic, we focused on Time behavior, Resource utilization, and Capacity sub-characteristics. Time behavior is measured by response times of incoming user requests. We believe the maximum waiting time should be no more than one second. Resource utilization is evaluated through monitoring CPU, RAM, and disk usage. As all the testing was conducted on a minimal virtual machine with reduced resources, we set the objective to not use any of the three resources at 100% capacity throughout the whole benchmark. Finally, for the Capacity sub-characteristic, we measure the ratio of successful and unsuccessful user requests. As the Authentication service is crucial, we believe at least 99% of user requests must pass.

TABLE I
Quality Goals of the Authentication subsystem

| Quality characteristics | Sub-characteristics | Quality measure | Objective |
|---|---|---|---|
| Functional Suitability | Completeness | Compliance with the specifications | All the specified functions of the system must be present |
| | Correctness | Unit Tests | All tests must pass |
| Reliability | Maturity | Delay mean time | Meantime 3 seconds maximum |
| | Availability | Downtime | Downtime 1% maximum under load |
| Maintainability | Modularity | Modularity index | Index reaches 0.5 |
| | Testability | Code coverage | Test cases reach 70% of code coverage |
| Performance | Time behavior | Response time | Average less than 2.5 seconds |
| | Resource utilization | Available resources | Utilization less than 100% all the time |
| | Capacity | Number of successful requests | At least 99% requests passed |

### B. Quality Assurance Strategies

The Performance characteristic is always measured on the final product. Even though there may be testings throughout the development process, what matters is the final product deployment. Therefore, all the newly introduced sub-characteristics in the Table II should be evaluated at the system testing stage. Objective and consistent testing is usually conducted by third parties as any internal performance testing may be biased.

TABLE II
Quality Assurance Strategies of the Authentication subsystem

| Scope | Stage | Roles involved |
|---|---|---|
| Completeness | System testing | Clients |
| Correctness | Development | Developers |
| Maturity | System testing | Third party |
| Availability | System testing | Third party |
| Modularity | Development | Developers |
| Testability | Development | Developers |
| Time behavior | System testing | Third party |
| Resource utilization | System testing | Third party |
| Capacity | System testing | Third party |

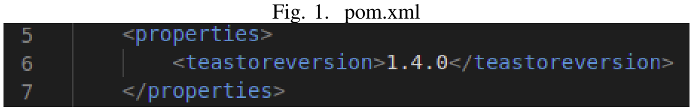
## III. Deployment

There are two methods on how to deploy a Docker image of an older version of TeaStore:

## A. Method 1

To deploy the reduced version of the Docker container, we have to change the version number indicated by lines 5 7 of POM.XML. The latest version is 1.4.0, and we changed it to 1.0.1 for the following experiments.

Fig. 1. pom.xml



```
5   <properties>
6       <teastoreversion>1.4.0</teastoreversion>
7   </properties>
```

After modification, we can launch the reduced version Docker image by the commands below.

```
$ docker run -p 3306:3306 -d --name teastore-db descartesresearch/teastore-db
$ docker run -e "DB_HOST=teastore-db" -p 8080:8080 -d --link
    teastore-db:teastore-db --name teastore-all descartesresearch/teastore-all
```

## B. Method 2

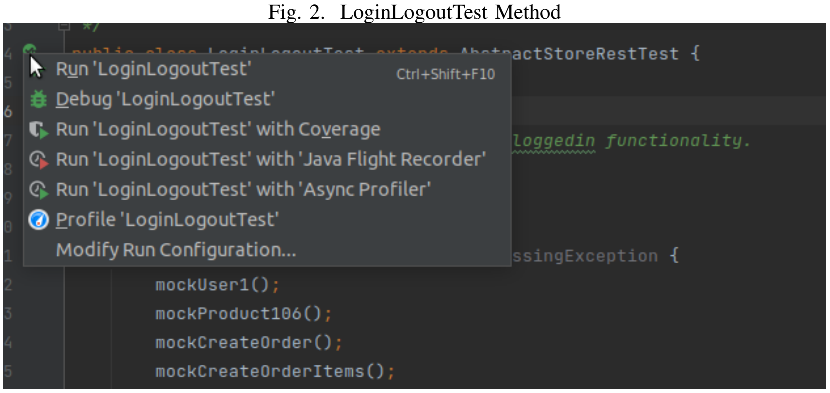
To deploy the reduced version of the docker container, we specify the version while executing the commands. For example, we launch version 1.0.1 by the commands below.

```
$ docker run -p 3306:3306 -d --name teastore-db
    descartesresearch/teastore-db:1.0.1
$ docker run -e "DB_HOST=teastore-db" -p 8080:8080 -d --link
    teastore-db:teastore-db --name teastore-all
    descartesresearch/teastore-all:1.0.1
```

# IV. Performance Profiling

In version v1.0.1, we found a test case LoginLogoutTest.java, which contains the tests of logging in/out, adding items to the cart, removing items from the cart, and placing the order. We profiled by Intellij's plugin, JProfiler [2].

Fig. 2. LoginLogoutTest Method


Run 'LoginLogoutTest'
Ctrl+Shift+F10
Debug 'LoginLogoutTest'
Run 'LoginLogoutTest' with Coverage
Run 'LoginLogoutTest' with 'Java Flight Recorder'
Run 'LoginLogoutTest' with 'Async Profiler'
Profile 'LoginLogoutTest'
Modify Run Configuration...

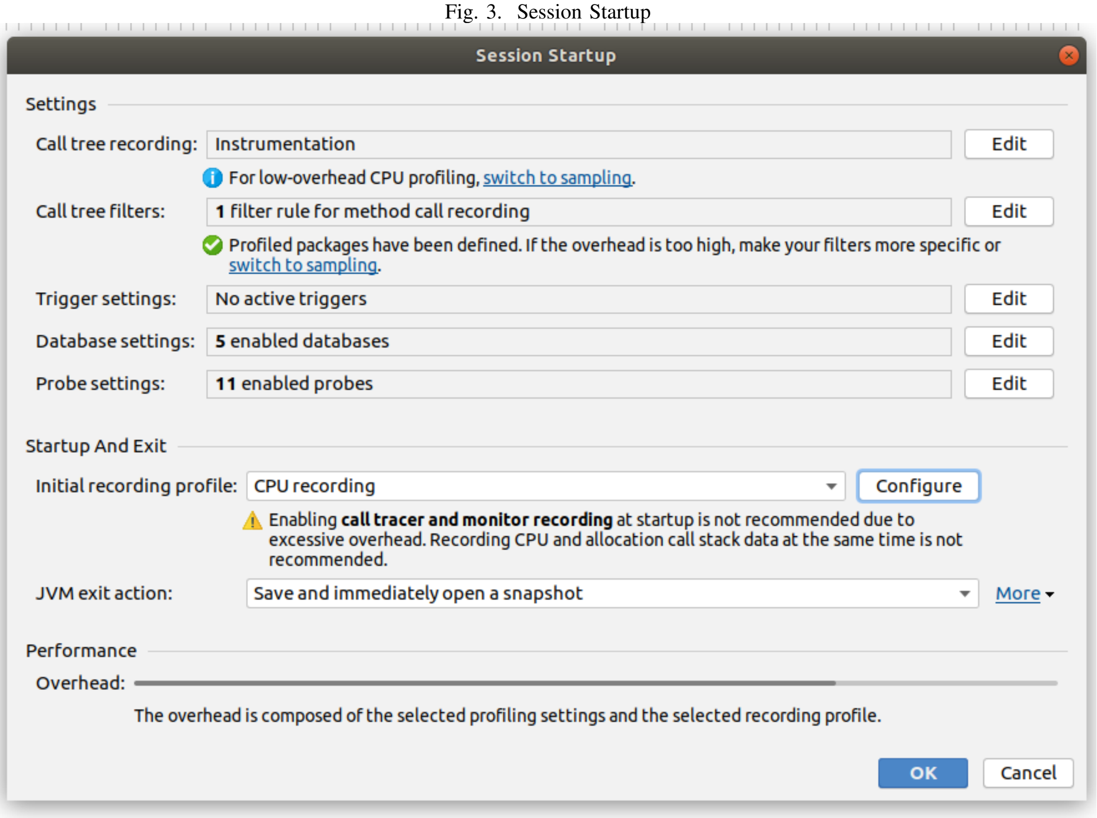
```
public class LoginLogoutTest extends AbstractStoreRestTest {
    loggedin functionality.
    ssingException {
        mockUser1();
        mockProduct106();
        mockCreateOrder();
        mockCreateOrderItems();
```

After selecting *Profile LoginLogoutTest*, the JProfiler UI session will pop up, and we choose the calling methods *instrumentation* and *full sampling* with default settings for experiments as seen on Fig. 3.

## A. Overall Hot Spots

The aggregation level of the hot spot results is class level. The first diagram is instrumentation, and the second is full sampling. We found that class JAVA.UTIL.CONCURRENT.THREADPOOLEXECUTOR$WRAPPINGRUNNABLE occupies the most time slots in both methods, while some classes have significant measure time between the two methods. For example, ORG.APACHE.TOMCAT.UTIL.THREADS.TASKTHREAD$WRAPPINGRUNNABLE occupied 23% of measure times in instrumentation but only occupied less than 1% in full sampling. Also, the result shows that instrumentation takes more time than full sampling, which matches our expectations. Results of instrumentation and full sampling are depicted on Fig. 4 and Fig. 5, respectively.

Fig. 3. Session Startup



Session Startup

Settings

Call tree recording: Instrumentation

For low-overhead CPU profiling, switch to sampling.

Call tree filters: 1 filter rule for method call recording

Profiled packages have been defined. If the overhead is too high, make your filters more specific or switch to sampling.

Trigger settings: No active triggers

Database settings: 5 enabled databases

Probe settings: 11 enabled probes

Startup And Exit

Initial recording profile: CPU recording

Enabling **call tracer and monitor recording** at startup is not recommended due to excessive overhead. Recording CPU and allocation call stack data at the same time is not recommended.

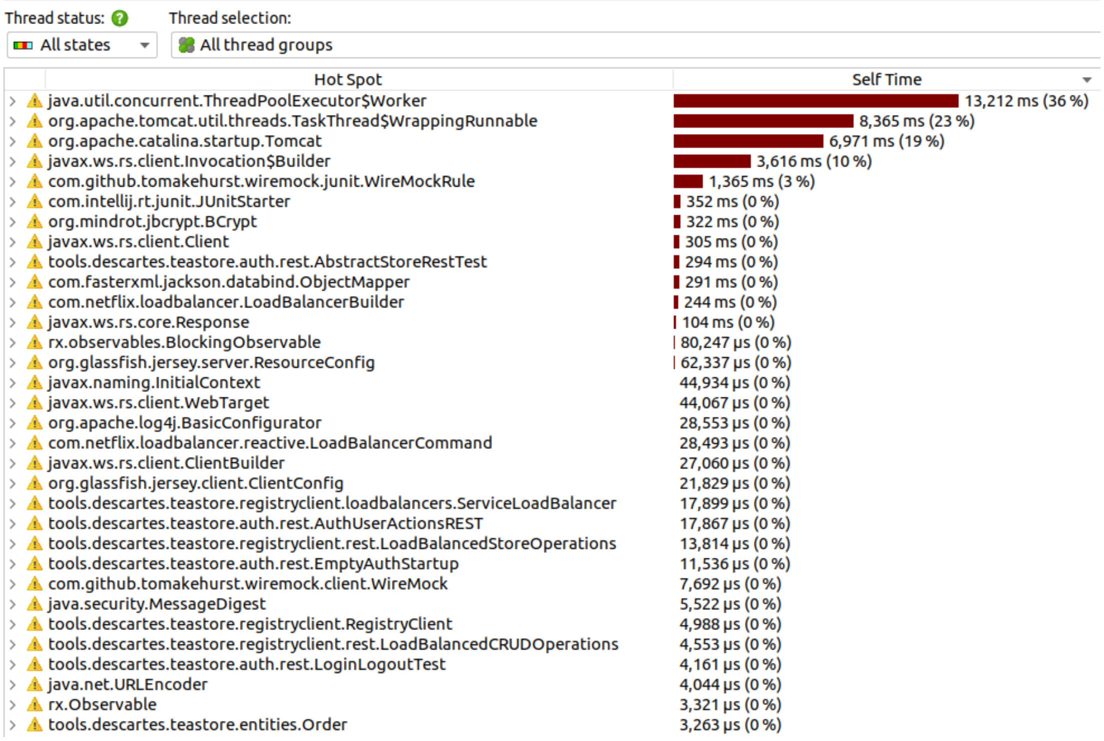
JVM exit action: Save and immediately open a snapshot

Performance

Overhead:

The overhead is composed of the selected profiling settings and the selected recording profile.

Fig. 4. Instrumentation Results

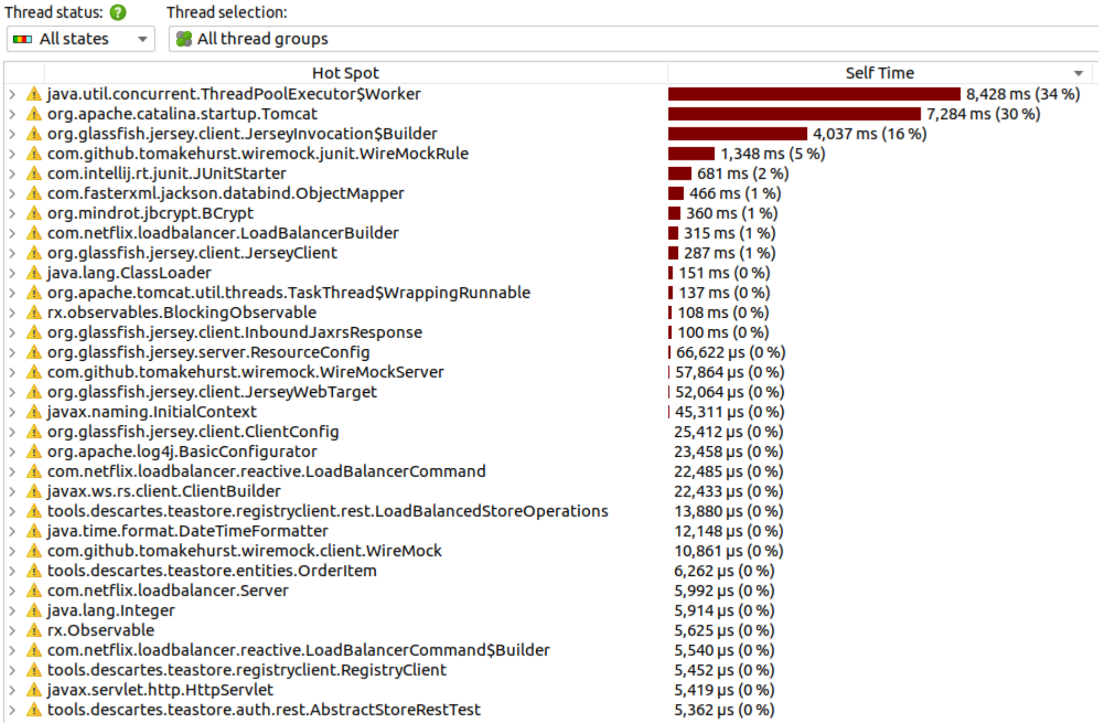


Thread status: All states

Thread selection: All thread groups

| Hot Spot | Self Time |
|---|---|
| java.util.concurrent.ThreadPoolExecutor$Worker | 13,212 ms (36 %) |
| org.apache.tomcat.util.threads.TaskThread$WrappingRunnable | 8,365 ms (23 %) |
| org.apache.catalina.startup.Tomcat | 6,971 ms (19 %) |
| javax.ws.rs.client.Invocation$Builder | 3,616 ms (10 %) |
| com.github.tomakehurst.wiremock.junit.WireMockRule | 1,365 ms (3 %) |
| com.intellij.rt.junit.JUnitStarter | 352 ms (0 %) |
| org.mindrot.jbcrypt.BCrypt | 322 ms (0 %) |
| javax.ws.rs.client.Client | 305 ms (0 %) |
| tools.descartes.teastore.auth.rest.AbstractStoreRestTest | 294 ms (0 %) |
| com.fasterxml.jackson.databind.ObjectMapper | 291 ms (0 %) |
| com.netflix.loadbalancer.LoadBalancerBuilder | 244 ms (0 %) |
| javax.ws.rs.core.Response | 104 ms (0 %) |
| rx.observables.BlockingObservable | 80,247 µs (0 %) |
| org.glassfish.jersey.server.ResourceConfig | 62,337 µs (0 %) |
| javax.naming.InitialContext | 44,934 µs (0 %) |
| javax.ws.rs.client.WebTarget | 44,067 µs (0 %) |
| org.apache.log4j.BasicConfigurator | 28,553 µs (0 %) |
| com.netflix.loadbalancer.reactive.LoadBalancerCommand | 28,493 µs (0 %) |
| javax.ws.rs.client.ClientBuilder | 27,060 µs (0 %) |
| org.glassfish.jersey.client.ClientConfig | 21,829 µs (0 %) |
| tools.descartes.teastore.registryclient.loadbalancers.ServiceLoadBalancer | 17,899 µs (0 %) |
| tools.descartes.teastore.auth.rest.AuthUserActionsREST | 17,867 µs (0 %) |
| tools.descartes.teastore.registryclient.rest.LoadBalancedStoreOperations | 13,814 µs (0 %) |
| tools.descartes.teastore.auth.rest.EmptyAuthStartup | 11,536 µs (0 %) |
| com.github.tomakehurst.wiremock.client.WireMock | 7,692 µs (0 %) |
| java.security.MessageDigest | 5,522 µs (0 %) |
| tools.descartes.teastore.registryclient.RegistryClient | 4,988 µs (0 %) |
| tools.descartes.teastore.registryclient.rest.LoadBalancedCRUDOperations | 4,553 µs (0 %) |
| tools.descartes.teastore.auth.rest.LoginLogoutTest | 4,161 µs (0 %) |
| java.net.URLEncoder | 4,044 µs (0 %) |
| rx.Observable | 3,321 µs (0 %) |
| tools.descartes.teastore.entities.Order | 3,263 µs (0 %) |

Fig. 5. Full Sampling Results



Thread status: All states

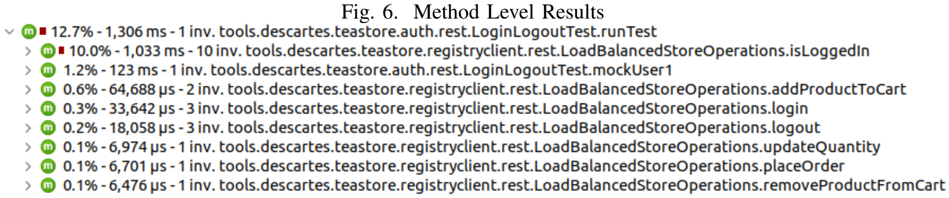
Thread selection: All thread groups

| Hot Spot | Self Time |
|---|---|
| java.util.concurrent.ThreadPoolExecutor$Worker | 8,428 ms (34 %) |
| org.apache.catalina.startup.Tomcat | 7,284 ms (30 %) |
| org.glassfish.jersey.client.JerseyInvocation$Builder | 4,037 ms (16 %) |
| com.github.tomakehurst.wiremock.junit.WireMockRule | 1,348 ms (5 %) |
| com.intellij.rt.junit.JUnitStarter | 681 ms (2 %) |
| com.fasterxml.jackson.databind.ObjectMapper | 466 ms (1 %) |
| org.mindrot.jbcrypt.BCrypt | 360 ms (1 %) |
| com.netflix.loadbalancer.LoadBalancerBuilder | 315 ms (1 %) |
| org.glassfish.jersey.client.JerseyClient | 287 ms (1 %) |
| java.lang.ClassLoader | 151 ms (0 %) |
| org.apache.tomcat.util.threads.TaskThread$WrappingRunnable | 137 ms (0 %) |
| rx.observables.BlockingObservable | 108 ms (0 %) |
| org.glassfish.jersey.client.InboundJaxrsResponse | 100 ms (0 %) |
| org.glassfish.jersey.server.ResourceConfig | 66,622 µs (0 %) |
| com.github.tomakehurst.wiremock.WireMockServer | 57,864 µs (0 %) |
| org.glassfish.jersey.client.JerseyWebTarget | 52,064 µs (0 %) |
| javax.naming.InitialContext | 45,311 µs (0 %) |
| org.glassfish.jersey.client.ClientConfig | 25,412 µs (0 %) |
| org.apache.log4j.BasicConfigurator | 23,458 µs (0 %) |
| com.netflix.loadbalancer.reactive.LoadBalancerCommand | 22,485 µs (0 %) |
| javax.ws.rs.client.ClientBuilder | 22,433 µs (0 %) |
| tools.descartes.teastore.registryclient.rest.LoadBalancedStoreOperations | 13,880 µs (0 %) |
| java.time.format.DateTimeFormatter | 12,148 µs (0 %) |
| com.github.tomakehurst.wiremock.client.WireMock | 10,861 µs (0 %) |
| tools.descartes.teastore.entities.OrderItem | 6,262 µs (0 %) |
| com.netflix.loadbalancer.Server | 5,992 µs (0 %) |
| java.lang.Integer | 5,914 µs (0 %) |
| rx.Observable | 5,625 µs (0 %) |
| com.netflix.loadbalancer.reactive.LoadBalancerCommand$Builder | 5,540 µs (0 %) |
| tools.descartes.teastore.registryclient.RegistryClient | 5,452 µs (0 %) |
| javax.servlet.http.HttpServlet | 5,419 µs (0 %) |
| tools.descartes.teastore.auth.rest.AbstractStoreRestTest | 5,362 µs (0 %) |

## B. Test Case Hot Spots

Fig. 6 shows the result while we only focus on the performance of the test case at the method level.

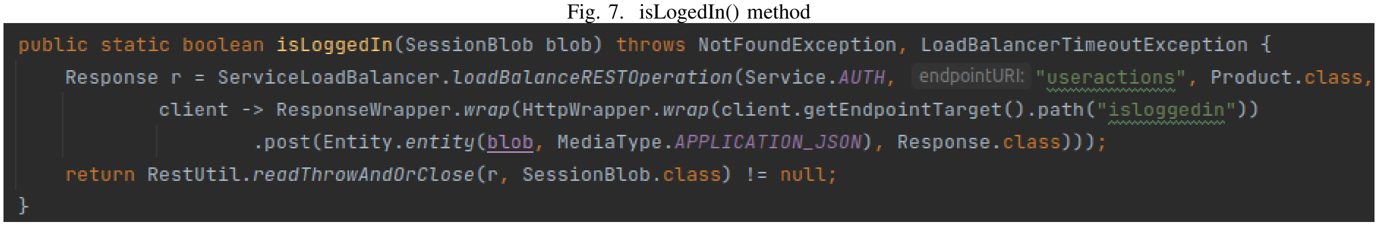
Fig. 6. Method Level Results



```
12.7% - 1,306 ms - 1 inv. tools.descartes.teastore.auth.rest.LoginLogoutTest.runTest
  10.0% - 1,033 ms - 10 inv. tools.descartes.teastore.registryclient.rest.LoadBalancedStoreOperations.isLoggedIn
  1.2% - 123 ms - 1 inv. tools.descartes.teastore.auth.rest.LoginLogoutTest.mockUser1
  0.6% - 64,688 µs - 2 inv. tools.descartes.teastore.registryclient.rest.LoadBalancedStoreOperations.addProductToCart
  0.3% - 33,642 µs - 3 inv. tools.descartes.teastore.registryclient.rest.LoadBalancedStoreOperations.login
  0.2% - 18,058 µs - 3 inv. tools.descartes.teastore.registryclient.rest.LoadBalancedStoreOperations.logout
  0.1% - 6,974 µs - 1 inv. tools.descartes.teastore.registryclient.rest.LoadBalancedStoreOperations.updateQuantity
  0.1% - 6,701 µs - 1 inv. tools.descartes.teastore.registryclient.rest.LoadBalancedStoreOperations.placeOrder
  0.1% - 6,476 µs - 1 inv. tools.descartes.teastore.registryclient.rest.LoadBalancedStoreOperations.removeProductFromCart
```

We can see that the period ISLOGGEDIN() takes is 10-100 times more than other operations, such as LOGIN(), LOGOUT(), and PLACEORDER(). If we look at the code snippet, we will find that the method calls external services via REST API, which needs more time to complete.

Fig. 7. isLogedIn() method



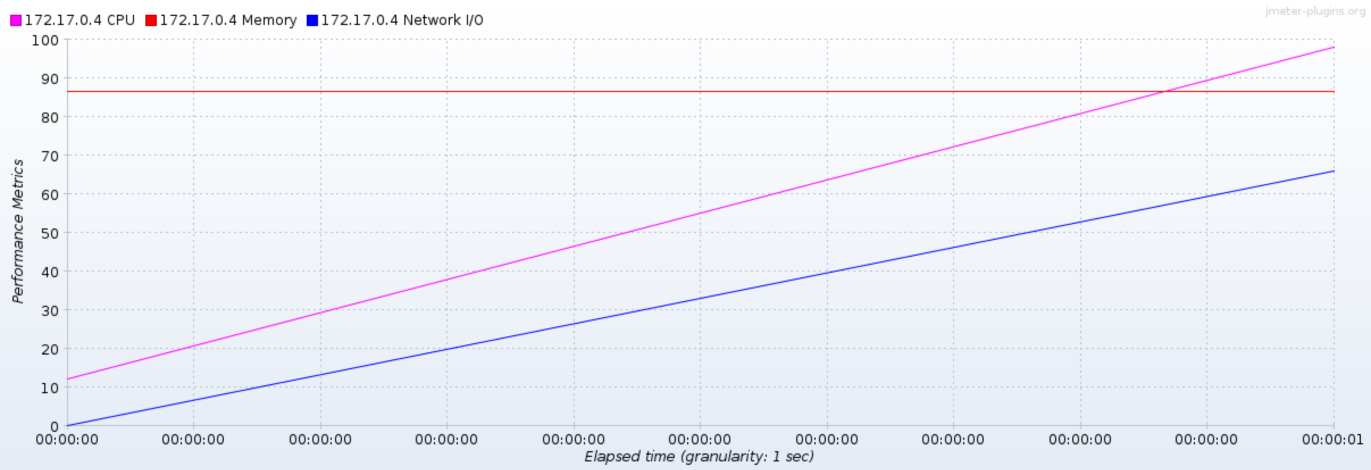
```java
public static boolean isLoggedIn(SessionBlob blob) throws NotFoundException, LoadBalancerTimeoutException {
    Response r = ServiceLoadBalancer.loadBalanceRESTOperation(Service.AUTH, endpointURI: "useractions", Product.class,
            client -> ResponseWrapper.wrap(HttpWrapper.wrap(client.getEndpointTarget().path("isloggedin"))
                    .post(Entity.entity(blob, MediaType.APPLICATION_JSON), Response.class)));
    return RestUtil.readThrowAndOrClose(r, SessionBlob.class) != null;
}
```

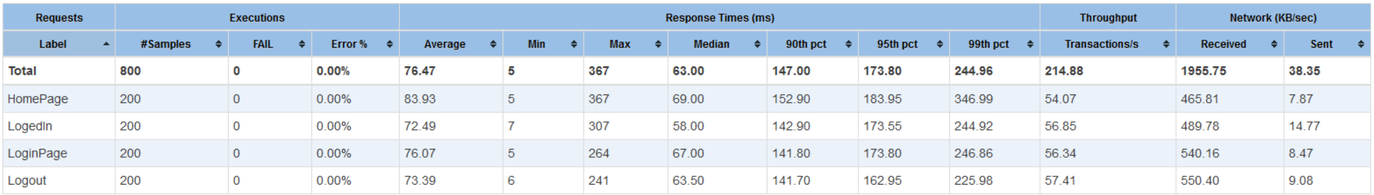
If the developers want to improve the overhead, they may use an alternative way to communicate externally. However, it might sacrifice the convenience of maintaining the project.

# V. VERSION 1.0.1

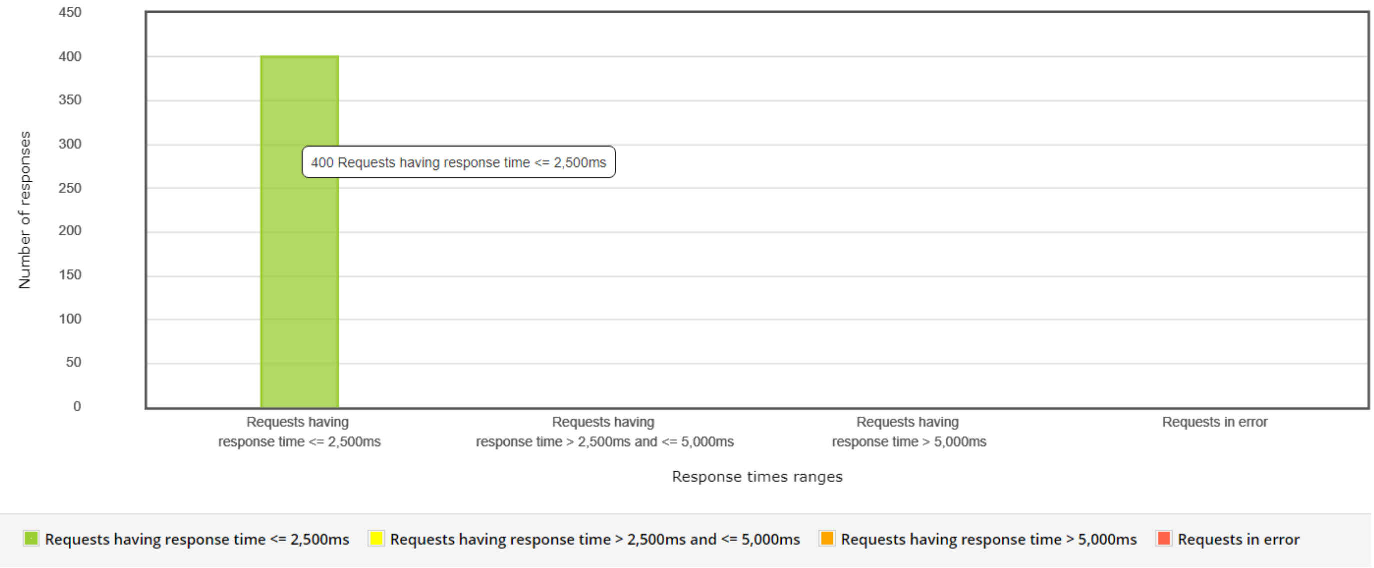
Load testing was conducted on a virtual machine. We used TeaStore version 1.0.1, which was deployed locally as a Docker image. The results of the current version follow in the next section. We simulated a load of 20, 200, 500, and 1000 users. The following diagrams are as follows: resource utilization throughout the benchmark, response times of methods, cumulative number of requests in four response time categories, and finally pass/fail ratio of user requests.

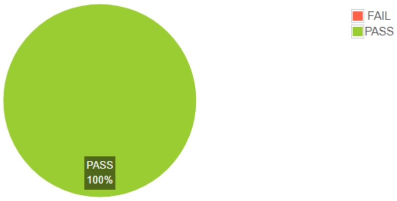
## A. 20 Users



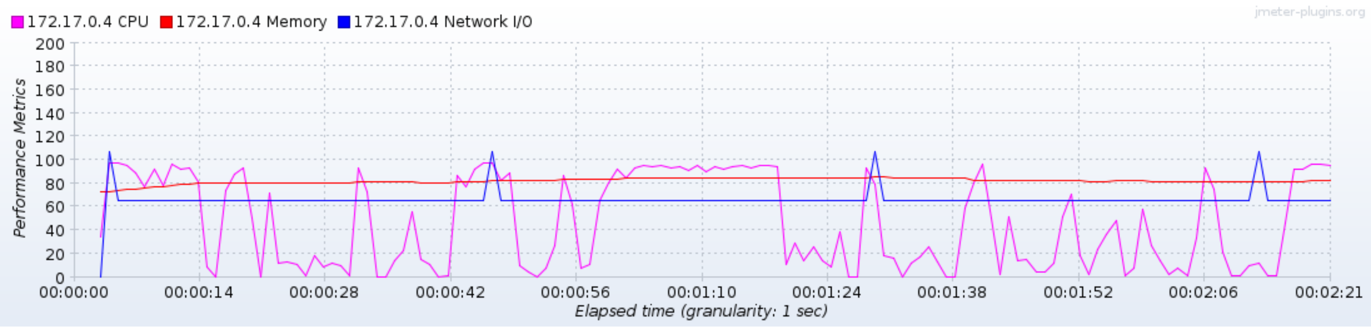


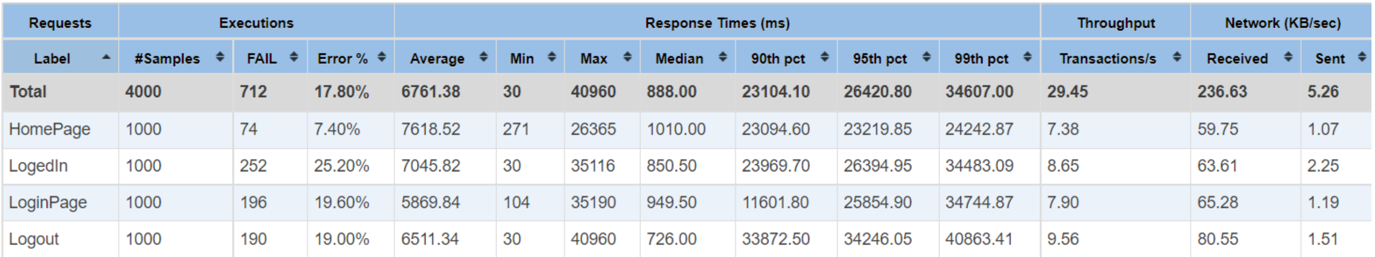
| Requests | Executions | | | Response Times (ms) | | | | | | | Throughput | Network (KB/sec) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Label | #Samples | FAIL | Error % | Average | Min | Max | Median | 90th pct | 95th pct | 99th pct | Transactions/s | Received | Sent |
| **Total** | **800** | **0** | **0.00%** | **76.47** | **5** | **367** | **63.00** | **147.00** | **173.80** | **244.96** | **214.88** | **1955.75** | **38.35** |
| HomePage | 200 | 0 | 0.00% | 83.93 | 5 | 367 | 69.00 | 152.90 | 183.95 | 346.99 | 54.07 | 465.81 | 7.87 |
| LogedIn | 200 | 0 | 0.00% | 72.49 | 7 | 307 | 58.00 | 142.90 | 173.55 | 244.92 | 56.85 | 489.78 | 14.77 |
| LoginPage | 200 | 0 | 0.00% | 76.07 | 5 | 264 | 67.00 | 141.80 | 173.80 | 246.86 | 56.34 | 540.16 | 8.47 |
| Logout | 200 | 0 | 0.00% | 73.39 | 6 | 241 | 63.50 | 141.70 | 162.95 | 225.98 | 57.41 | 550.40 | 9.08 |



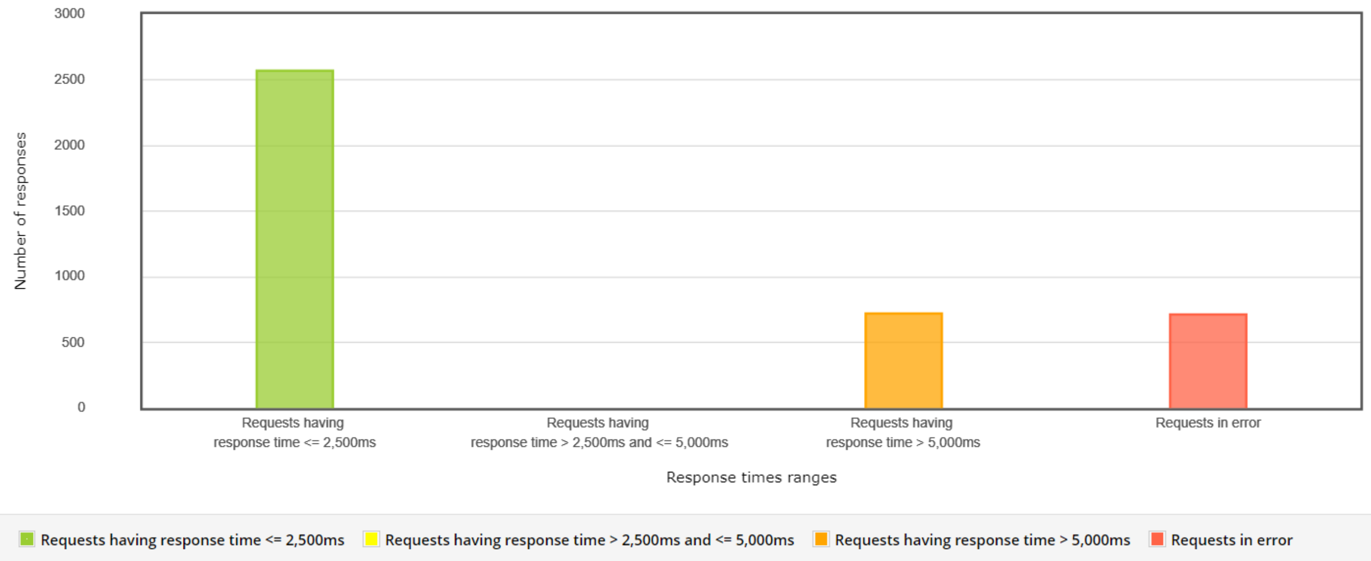


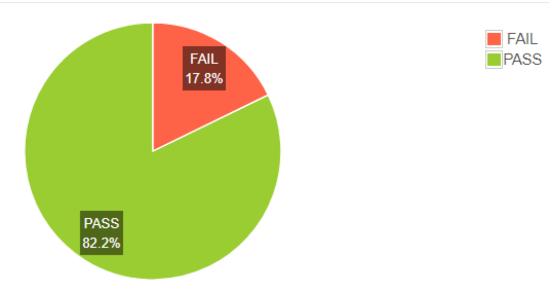
## B. 200 Users



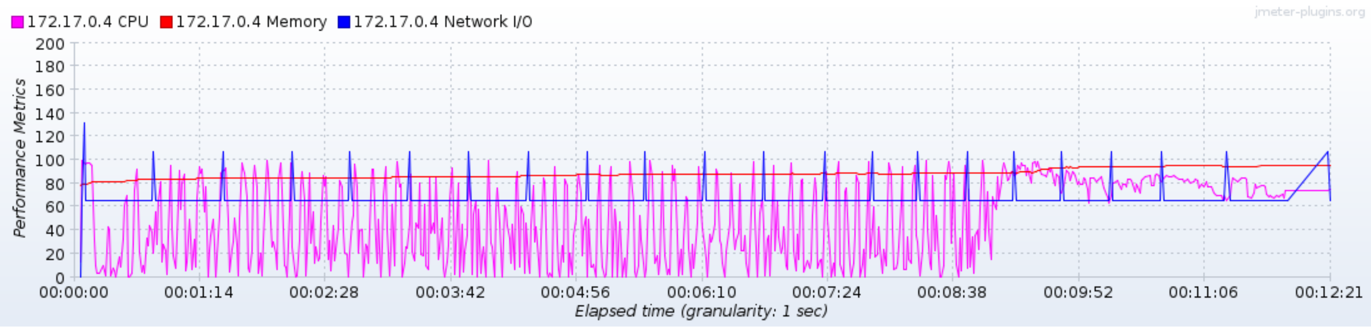


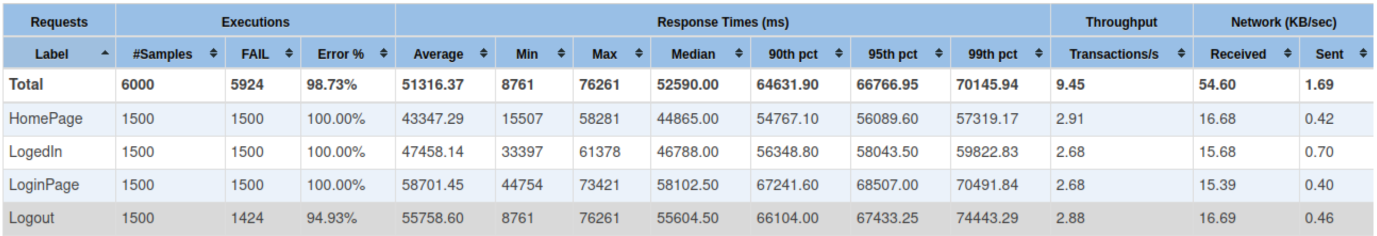
| Requests | Executions | | | Response Times (ms) | | | | | | | Throughput | Network (KB/sec) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Label | #Samples | FAIL | Error % | Average | Min | Max | Median | 90th pct | 95th pct | 99th pct | Transactions/s | Received | Sent |
| **Total** | **4000** | **712** | **17.80%** | **6761.38** | **30** | **40960** | **888.00** | **23104.10** | **26420.80** | **34607.00** | **29.45** | **236.63** | **5.26** |
| HomePage | 1000 | 74 | 7.40% | 7618.52 | 271 | 26365 | 1010.00 | 23094.60 | 23219.85 | 24242.87 | 7.38 | 59.75 | 1.07 |
| LogedIn | 1000 | 252 | 25.20% | 7045.82 | 30 | 35116 | 850.50 | 23969.70 | 26394.95 | 34483.09 | 8.65 | 63.61 | 2.25 |
| LoginPage | 1000 | 196 | 19.60% | 5869.84 | 104 | 35190 | 949.50 | 11601.80 | 25854.90 | 34744.87 | 7.90 | 65.28 | 1.19 |
| Logout | 1000 | 190 | 19.00% | 6511.34 | 30 | 40960 | 726.00 | 33872.50 | 34246.05 | 40863.41 | 9.56 | 80.55 | 1.51 |



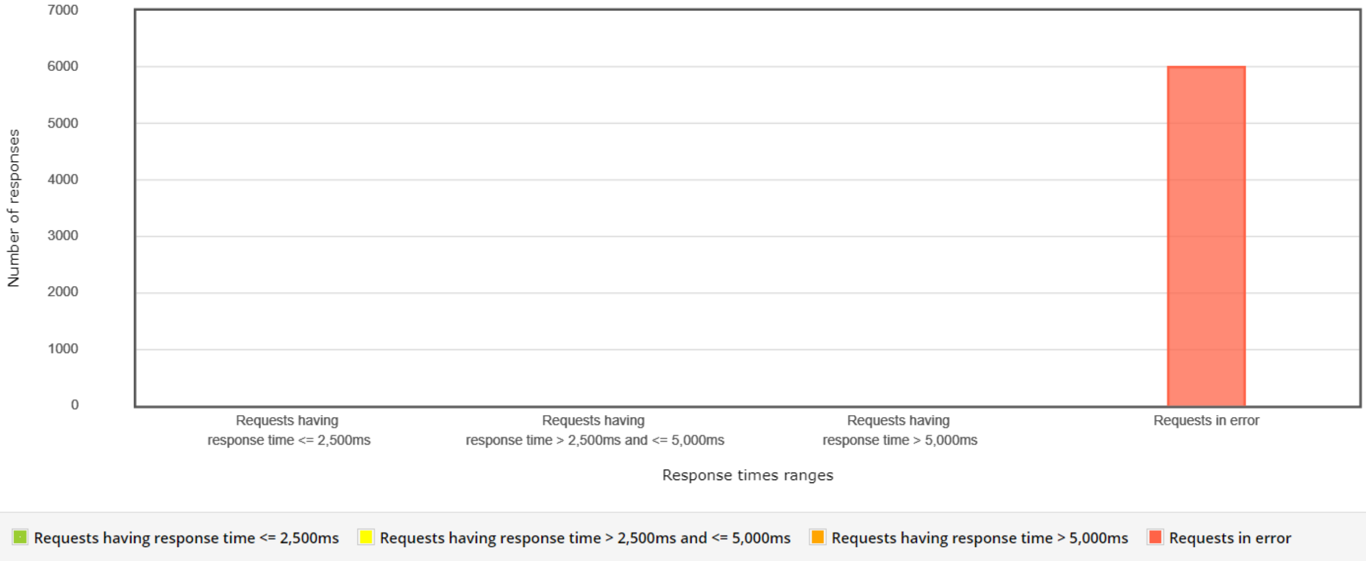


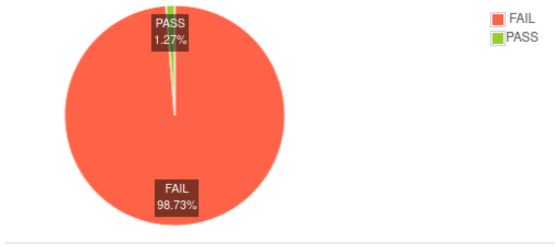
### C. 500 Users



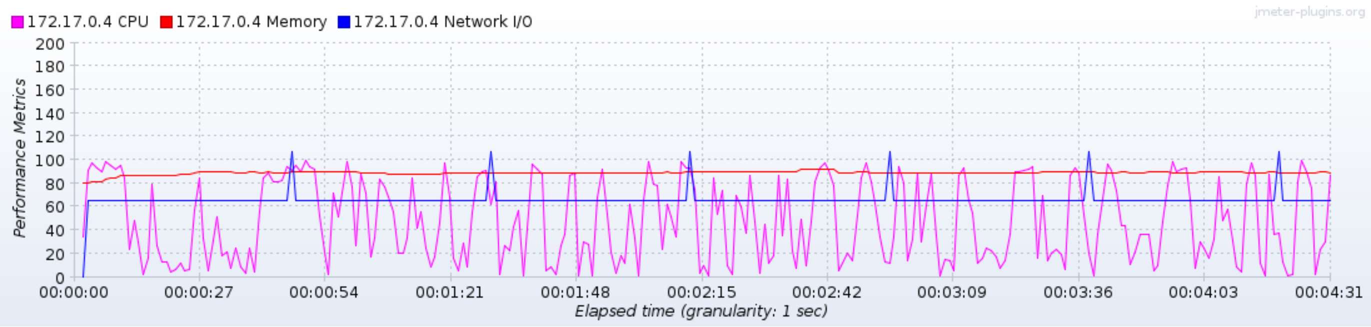


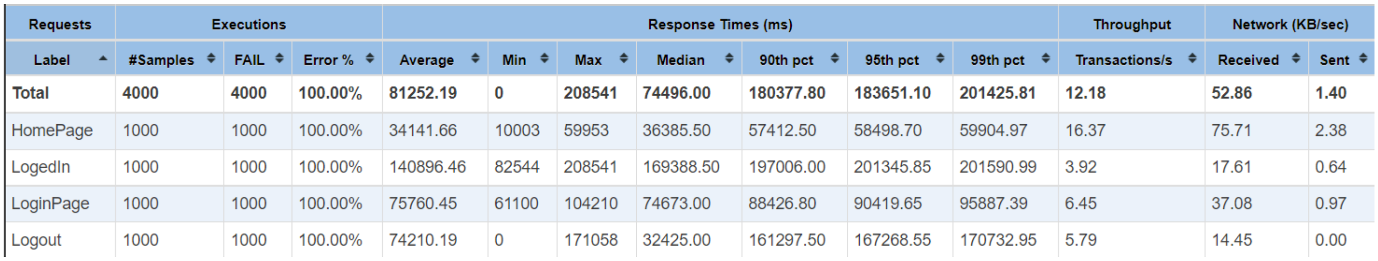
| Requests | Executions | | | Response Times (ms) | | | | | | | Throughput | Network (KB/sec) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Label | #Samples | FAIL | Error % | Average | Min | Max | Median | 90th pct | 95th pct | 99th pct | Transactions/s | Received | Sent |
| **Total** | **6000** | **5924** | **98.73%** | **51316.37** | **8761** | **76261** | **52590.00** | **64631.90** | **66766.95** | **70145.94** | **9.45** | **54.60** | **1.69** |
| HomePage | 1500 | 1500 | 100.00% | 43347.29 | 15507 | 58281 | 44865.00 | 54767.10 | 56089.60 | 57319.17 | 2.91 | 16.68 | 0.42 |
| LogedIn | 1500 | 1500 | 100.00% | 47458.14 | 33397 | 61378 | 46788.00 | 56348.80 | 58043.50 | 59822.83 | 2.68 | 15.68 | 0.70 |
| LoginPage | 1500 | 1500 | 100.00% | 58701.45 | 44754 | 73421 | 58102.50 | 67241.60 | 68507.00 | 70491.84 | 2.68 | 15.39 | 0.40 |
| Logout | 1500 | 1424 | 94.93% | 55758.60 | 8761 | 76261 | 55604.50 | 66104.00 | 67433.25 | 74443.29 | 2.88 | 16.69 | 0.46 |



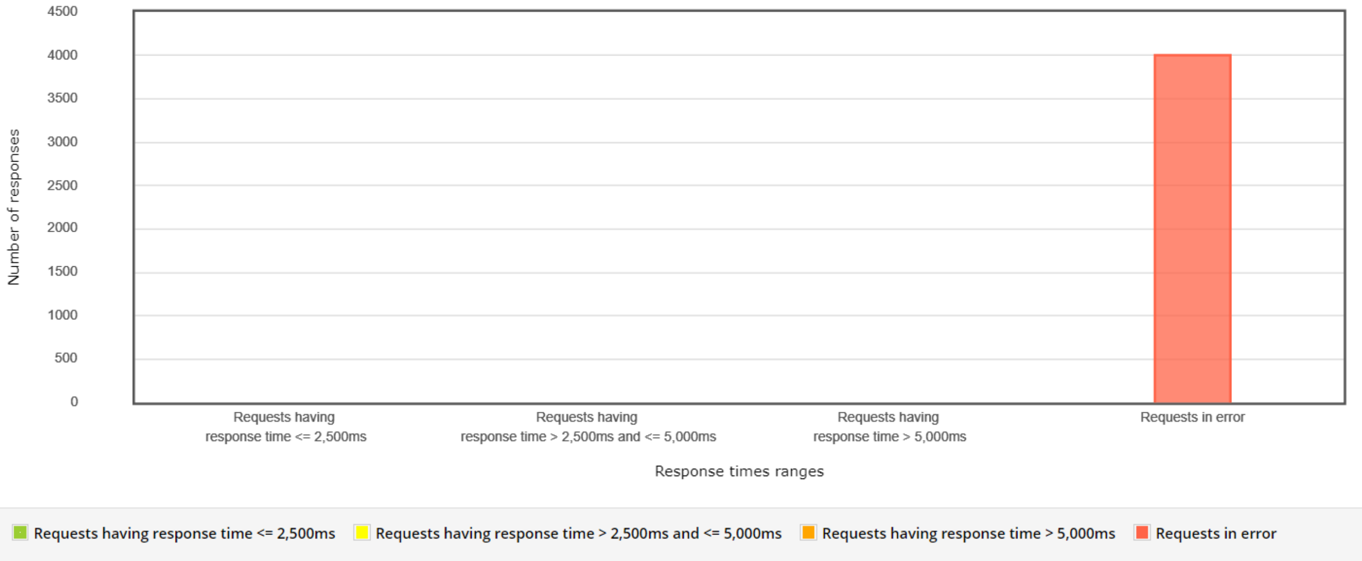


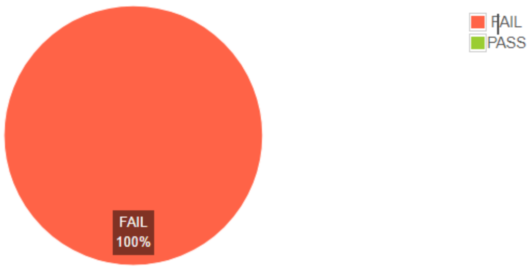
### *D. 1000 Users*





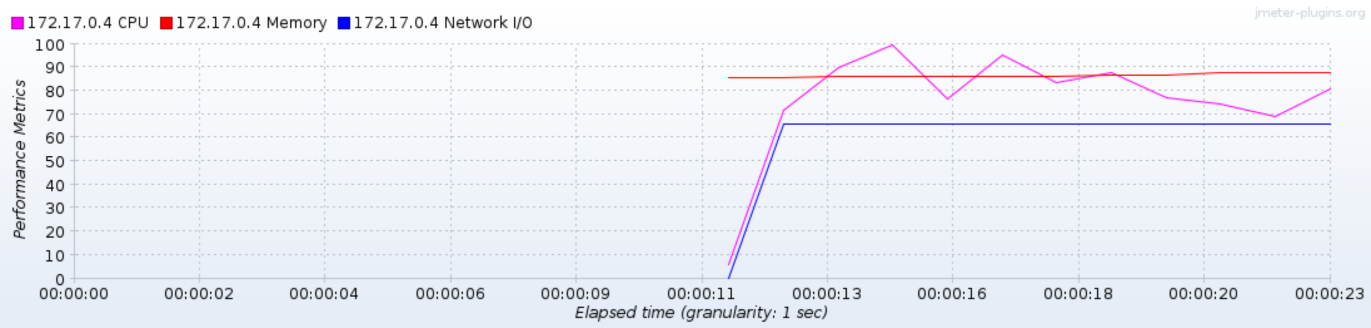
| Requests | Executions | | | Response Times (ms) | | | | | | | Throughput | Network (KB/sec) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Label | #Samples | FAIL | Error % | Average | Min | Max | Median | 90th pct | 95th pct | 99th pct | Transactions/s | Received | Sent |
| **Total** | **4000** | **4000** | **100.00%** | **81252.19** | **0** | **208541** | **74496.00** | **180377.80** | **183651.10** | **201425.81** | **12.18** | **52.86** | **1.40** |
| HomePage | 1000 | 1000 | 100.00% | 34141.66 | 10003 | 59953 | 36385.50 | 57412.50 | 58498.70 | 59904.97 | 16.37 | 75.71 | 2.38 |
| LogedIn | 1000 | 1000 | 100.00% | 140896.46 | 82544 | 208541 | 169388.50 | 197006.00 | 201345.85 | 201590.99 | 3.92 | 17.61 | 0.64 |
| LoginPage | 1000 | 1000 | 100.00% | 75760.45 | 61100 | 104210 | 74673.00 | 88426.80 | 90419.65 | 95887.39 | 6.45 | 37.08 | 0.97 |
| Logout | 1000 | 1000 | 100.00% | 74210.19 | 0 | 171058 | 32425.00 | 161297.50 | 167268.55 | 170732.95 | 5.79 | 14.45 | 0.00 |

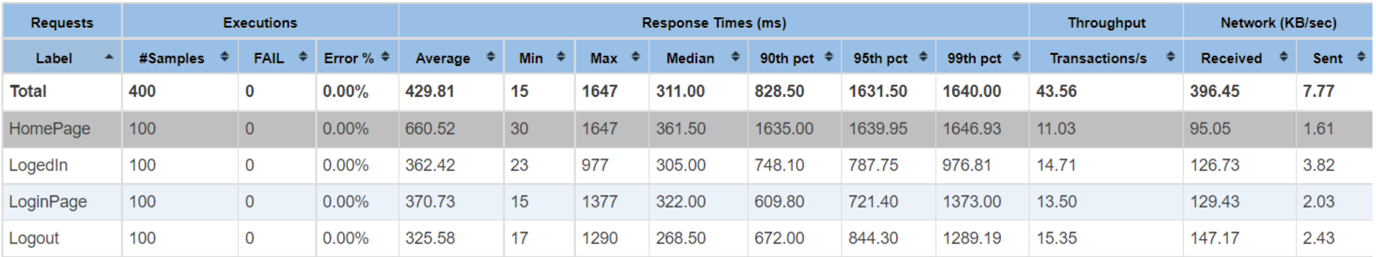




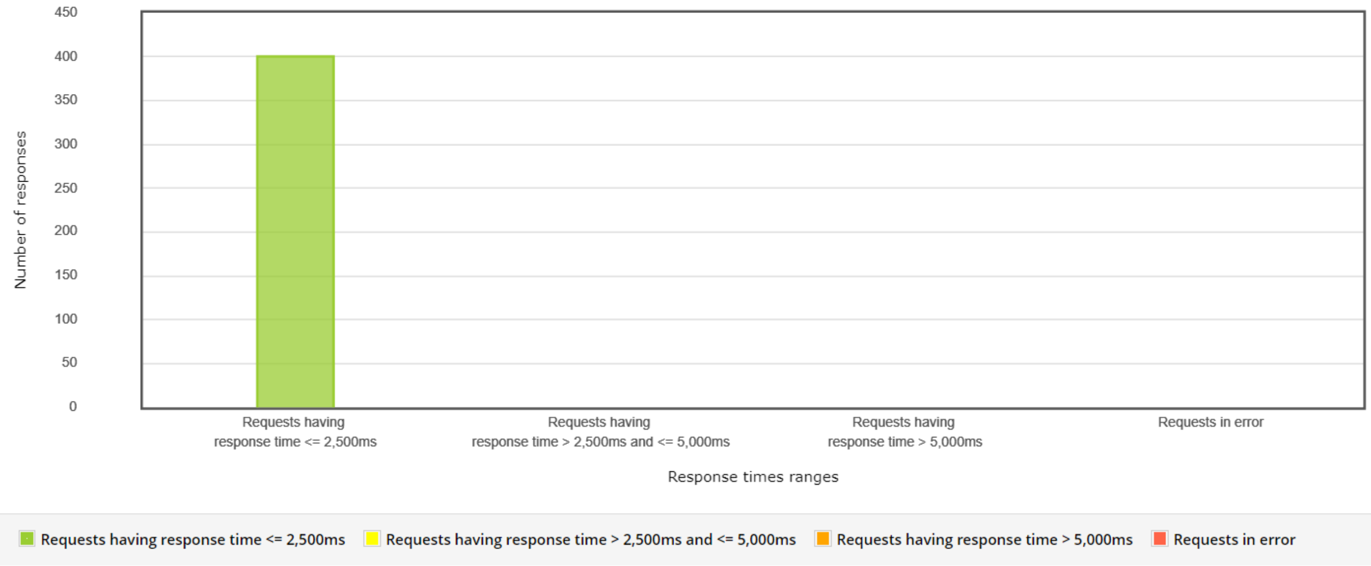
# VI. Current Version

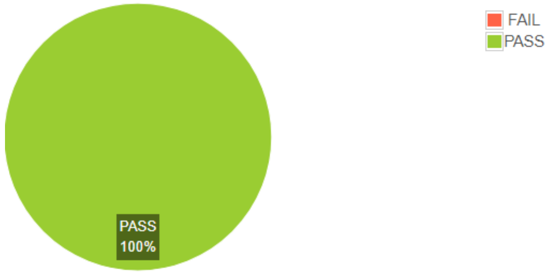
## A. 20 Users



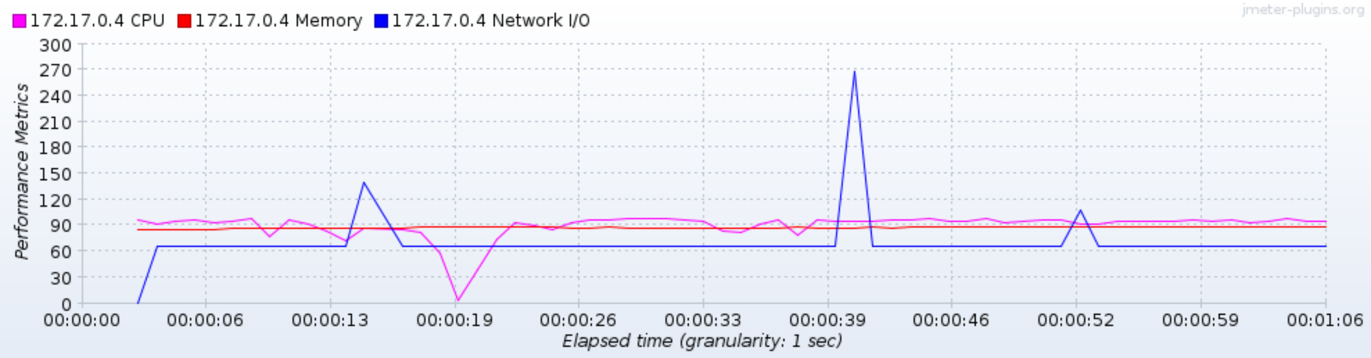


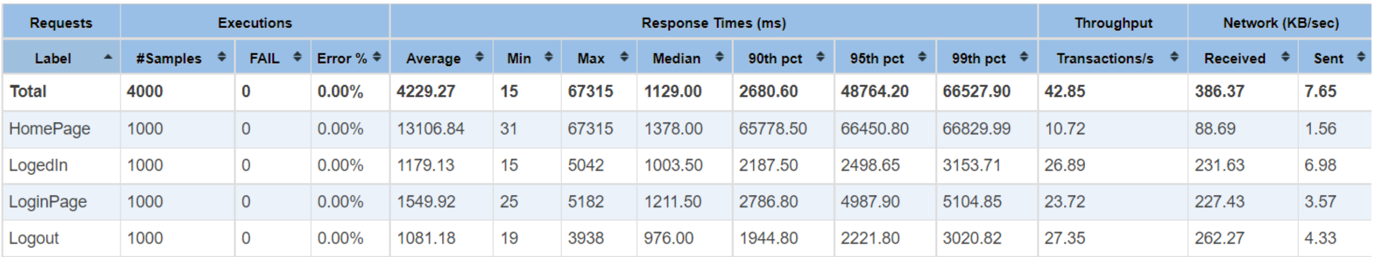
| Requests | Executions | | | Response Times (ms) | | | | | | | Throughput | Network (KB/sec) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Label | #Samples | FAIL | Error % | Average | Min | Max | Median | 90th pct | 95th pct | 99th pct | Transactions/s | Received | Sent |
| **Total** | **400** | **0** | **0.00%** | **429.81** | **15** | **1647** | **311.00** | **828.50** | **1631.50** | **1640.00** | **43.56** | **396.45** | **7.77** |
| HomePage | 100 | 0 | 0.00% | 660.52 | 30 | 1647 | 361.50 | 1635.00 | 1639.95 | 1646.93 | 11.03 | 95.05 | 1.61 |
| LogedIn | 100 | 0 | 0.00% | 362.42 | 23 | 977 | 305.00 | 748.10 | 787.75 | 976.81 | 14.71 | 126.73 | 3.82 |
| LoginPage | 100 | 0 | 0.00% | 370.73 | 15 | 1377 | 322.00 | 609.80 | 721.40 | 1373.00 | 13.50 | 129.43 | 2.03 |
| Logout | 100 | 0 | 0.00% | 325.58 | 17 | 1290 | 268.50 | 672.00 | 844.30 | 1289.19 | 15.35 | 147.17 | 2.43 |



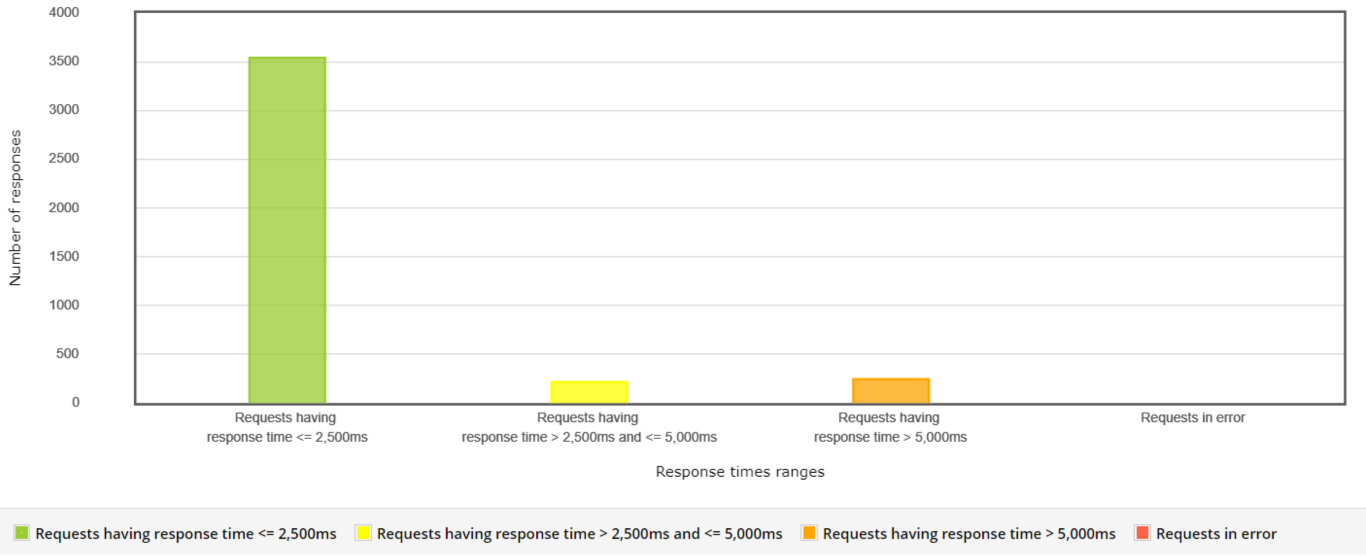


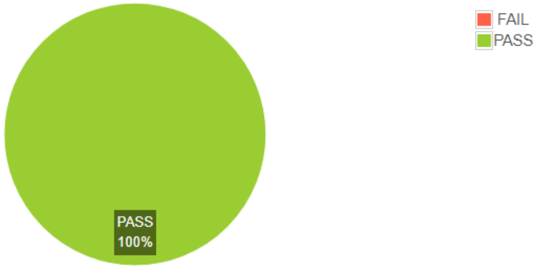
### *B. 200 Users*



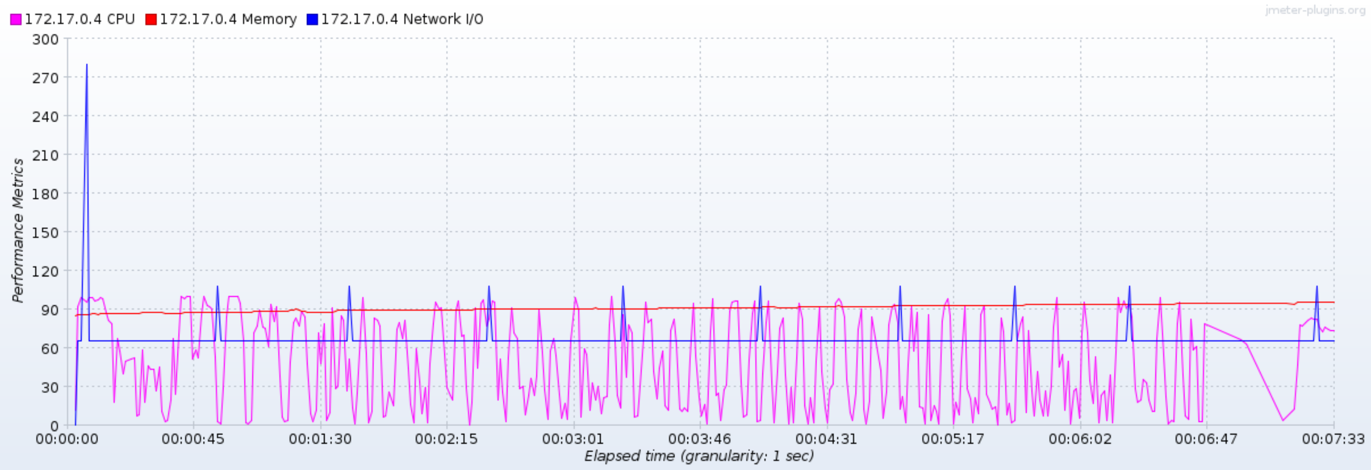


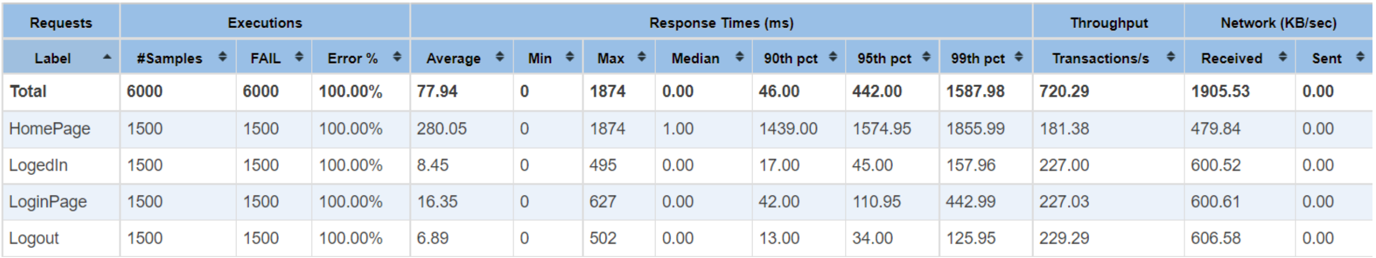
| Requests | Executions | | | Response Times (ms) | | | | | | | Throughput | Network (KB/sec) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Label | #Samples | FAIL | Error % | Average | Min | Max | Median | 90th pct | 95th pct | 99th pct | Transactions/s | Received | Sent |
| **Total** | **4000** | **0** | **0.00%** | **4229.27** | **15** | **67315** | **1129.00** | **2680.60** | **48764.20** | **66527.90** | **42.85** | **386.37** | **7.65** |
| HomePage | 1000 | 0 | 0.00% | 13106.84 | 31 | 67315 | 1378.00 | 65778.50 | 66450.80 | 66829.99 | 10.72 | 88.69 | 1.56 |
| LogedIn | 1000 | 0 | 0.00% | 1179.13 | 15 | 5042 | 1003.50 | 2187.50 | 2498.65 | 3153.71 | 26.89 | 231.63 | 6.98 |
| LoginPage | 1000 | 0 | 0.00% | 1549.92 | 25 | 5182 | 1211.50 | 2786.80 | 4987.90 | 5104.85 | 23.72 | 227.43 | 3.57 |
| Logout | 1000 | 0 | 0.00% | 1081.18 | 19 | 3938 | 976.00 | 1944.80 | 2221.80 | 3020.82 | 27.35 | 262.27 | 4.33 |



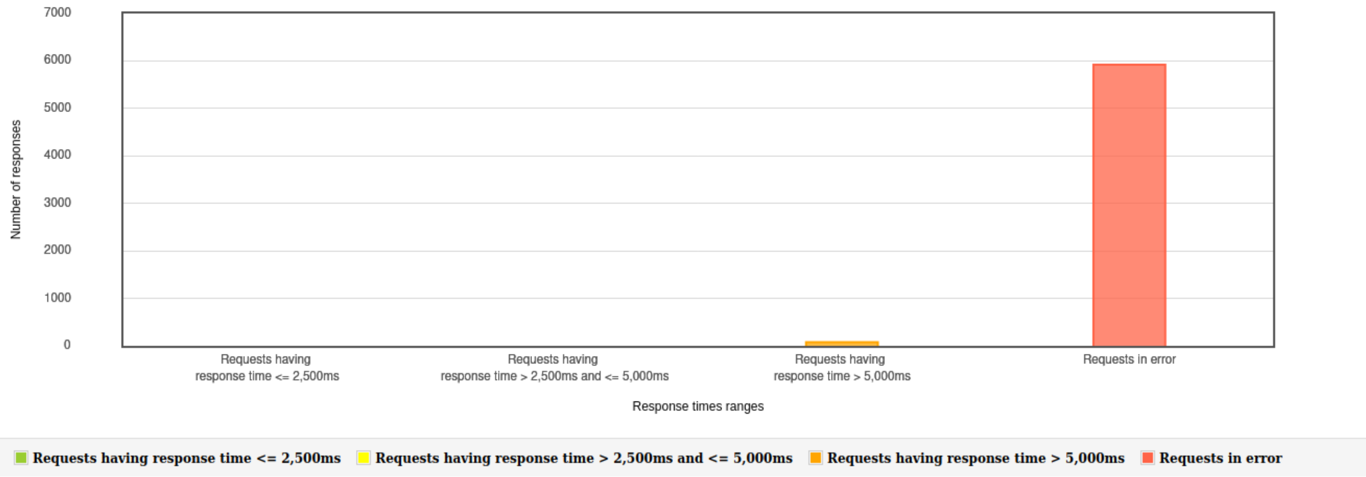


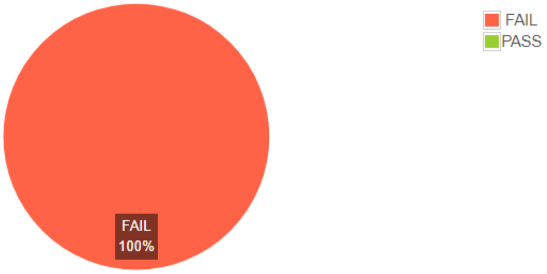
### C. 500 Users



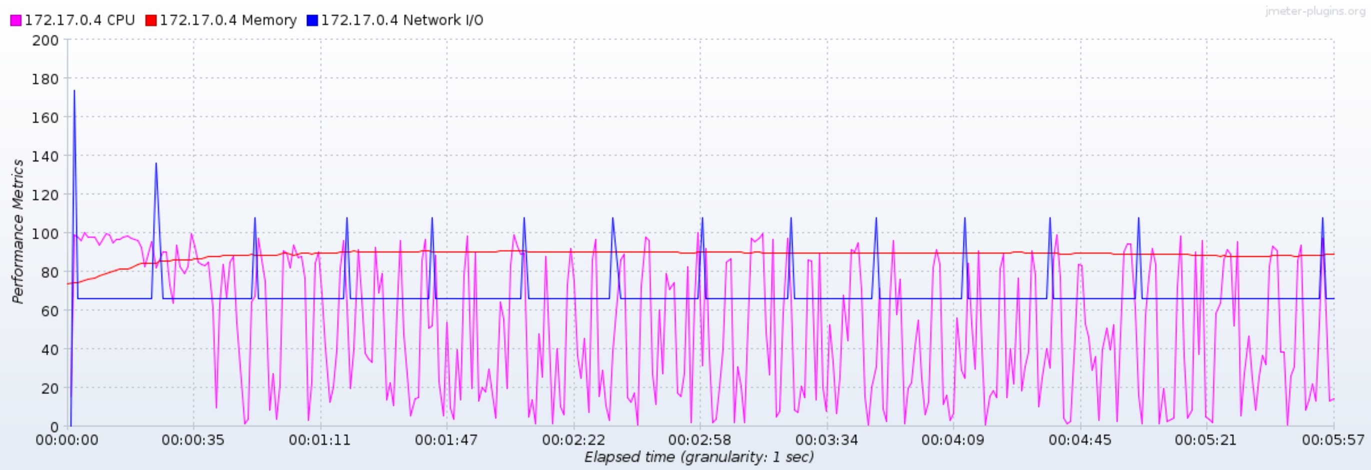


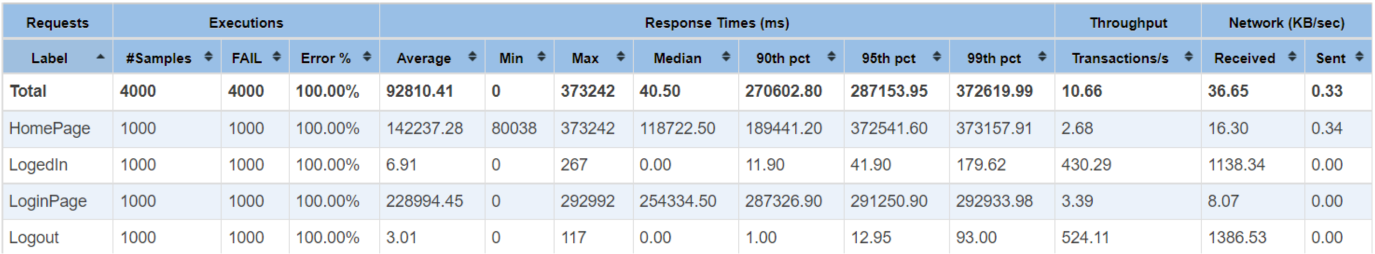
| Requests | Executions | | | Response Times (ms) | | | | | | | Throughput | Network (KB/sec) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Label | #Samples | FAIL | Error % | Average | Min | Max | Median | 90th pct | 95th pct | 99th pct | Transactions/s | Received | Sent |
| **Total** | **6000** | **6000** | **100.00%** | **77.94** | **0** | **1874** | **0.00** | **46.00** | **442.00** | **1587.98** | **720.29** | **1905.53** | **0.00** |
| HomePage | 1500 | 1500 | 100.00% | 280.05 | 0 | 1874 | 1.00 | 1439.00 | 1574.95 | 1855.99 | 181.38 | 479.84 | 0.00 |
| LogedIn | 1500 | 1500 | 100.00% | 8.45 | 0 | 495 | 0.00 | 17.00 | 45.00 | 157.96 | 227.00 | 600.52 | 0.00 |
| LoginPage | 1500 | 1500 | 100.00% | 16.35 | 0 | 627 | 0.00 | 42.00 | 110.95 | 442.99 | 227.03 | 600.61 | 0.00 |
| Logout | 1500 | 1500 | 100.00% | 6.89 | 0 | 502 | 0.00 | 13.00 | 34.00 | 125.95 | 229.29 | 606.58 | 0.00 |



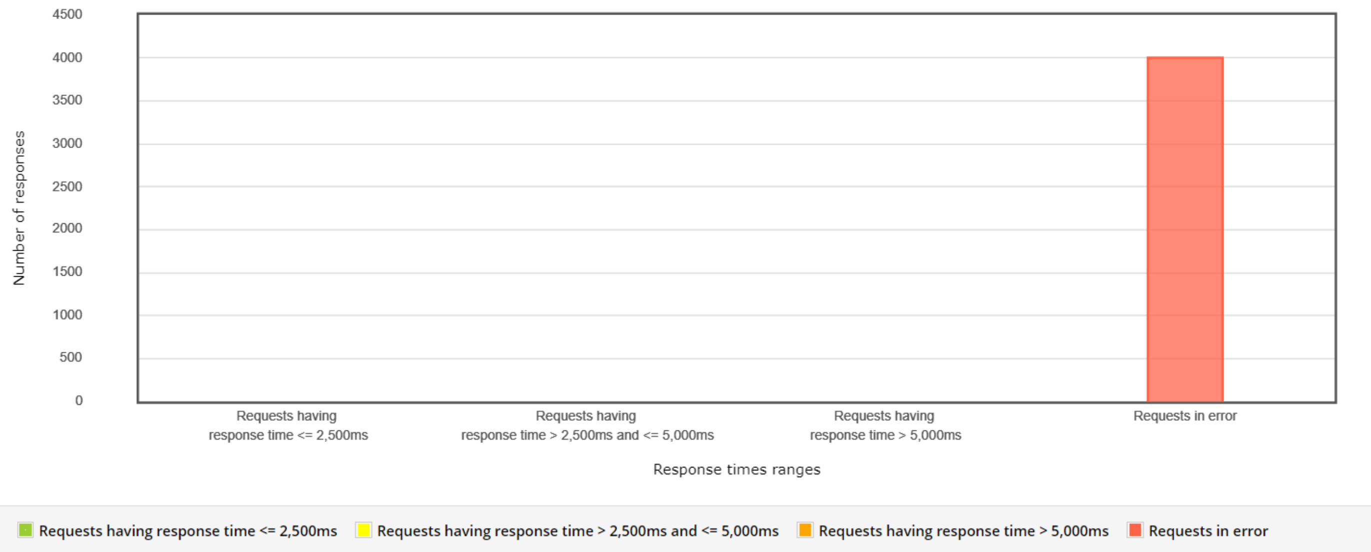


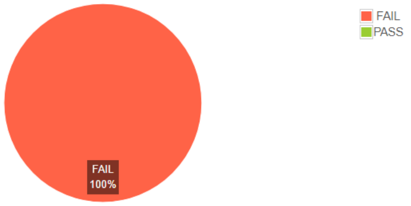
### D. 1000 Users





| Requests | Executions | | | Response Times (ms) | | | | | | | Throughput | Network (KB/sec) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Label | #Samples | FAIL | Error % | Average | Min | Max | Median | 90th pct | 95th pct | 99th pct | Transactions/s | Received | Sent |
| **Total** | **4000** | **4000** | **100.00%** | **92810.41** | **0** | **373242** | **40.50** | **270602.80** | **287153.95** | **372619.99** | **10.66** | **36.65** | **0.33** |
| HomePage | 1000 | 1000 | 100.00% | 142237.28 | 80038 | 373242 | 118722.50 | 189441.20 | 372541.60 | 373157.91 | 2.68 | 16.30 | 0.34 |
| LogedIn | 1000 | 1000 | 100.00% | 6.91 | 0 | 267 | 0.00 | 11.90 | 41.90 | 179.62 | 430.29 | 1138.34 | 0.00 |
| LoginPage | 1000 | 1000 | 100.00% | 228994.45 | 0 | 292992 | 254334.50 | 287326.90 | 291250.90 | 292933.98 | 3.39 | 8.07 | 0.00 |
| Logout | 1000 | 1000 | 100.00% | 3.01 | 0 | 117 | 0.00 | 1.00 | 12.95 | 93.00 | 524.11 | 1386.53 | 0.00 |





## VII. Discussion

### A. Time behavior

We can see that the current version of TeaStore is more performant than version 1.0.1. This information is obtained from the response timetables. Version 1.0.1 clearly runs faster in the 20 users benchmark, with average, maximum, and median being way lower than the same benchmark of the current version. However, for the 200 users benchmark, which represents the average load, the average response time of the current version is better, but not the maximum nor the median. The better performance of the current version is not apparent until we look at another presented graph, the one showing a total count of responses in 4 different response time categories. This graph clearly shows that a more significant ratio of requests takes less than 2.5 seconds compared to the older version. However, both versions could not pass the set objective of 99% response times of requests under 2.5 seconds, even for an average load. If we look at 500 and 1000 users, we can see TeaStore is clearly not able to keep up. Not only the average response times is exceptionally high, but these requests also fail. In conclusion, both versions did not pass the Time behavior metric.

### B. Resource utilization

There is no apparent difference between the two versions. Both versions also passed the set objective. The only noteworthy characteristic we can read from the graphs is the slowly increasing usage of memory. However, this does not present a genuine issue either since TeaStore is written in Java which automatically garbage collects.

### C. Capacity

Looking at the last graph of each benchmark, we can see both versions have serious issues keeping up with the above-average load. Additionally, the older version has issues even on the average load of 200 users. This clearly shows that both versions did not pass the Capacity metric.

## References

[1] https://github.com/DescartesResearch/TeaStore

[2] https://www.ej-technologies.com/products/jprofiler/overview.html