Performance Test Report for www.demoblaze.com

**Executive summary**

Under the indicated workload the overall Domoblaze’ response time (for both UI and API) was within what is considered acceptable for modern web application ([ref](#NNG).).

Domoblaze’ behaviour showed consistent (each test was repeated at least 3 times) across different approaches (UI and API testing), tools and workloads. No errors or substantial delays were observed under the indicated load and duration, and the overall user experience appeared to be pleasant and smooth while the webapp was under load (manual browsing was also conducted). The response time for static content only showed faster (e.g., testing via **Google Lighthouse**) if compared to combined (API and Static, e.g., tested via JMeter**/WebDriver/Firefox**) or the API only (**JMeter**).

Although this was somehow expected, it was also noticed that with an increased load the API gradually returned a higher response time ( double, tripled and picked to even 10+ times over the minimum value) which suggests that in its current configuration the service might have an upper limit (yet to be revealed and not in scope of this [engagement](#Engagement)) after which the performances might start deteriorating. This was confirmed by running the same test for longer (~180sec. The degradation of performances was also observed through to manual testing by recording the response time via ***Google Performance*** (measuring the difference between “before” and “during” the API load test)**,**however it was still considered acceptable from a user experience perspective.

Considering the above findings, the service could benefit from an auto-scaling (adaptive computing power to the service demand) and/or a caching feature, which could act as protective mechanism against unexpected high volume or spikes in demand.

The following are a few recommendations to improve the response further:

* API caching mechanism/Autoscaling
* Server static asset with an efficient cache policy (server did not return Cache-Control: max-age)
* Ensure text remains visible during webfont load (use font-display: swap in CSS font class)
* Consider reducing images sizing
* Consider serving images as AVIF or WebP (better compression quality)
* Consider removing unused JavaScript

**Requirements in scope – Terms of Engagement**

The load test needs to simulate 1000 users who will visit the homepage in a period of 15s. Measure web application response time before and during the test run.

**Context and related Test Approach**

Mainly , there are two components to consider while exploring, understanding, and reporting about web application performances: **UI and an API**

Although they are equally important, they evaluate various aspects of performance.

**UI Performance**

It is vital for the user experience, and it focuses on downloading, processing, and visualising (DOM rendering) content on the client side (mainly web or mobile browser). Generally, this content comes in a form of files. Technically speaking it is called “static” and it is served by web server such as NGINX or similar. They are images (such as jpeg or other modern light ware formats such as AVIF or WebP), JavaScript (that holds the application logic), HTML (layout of the web pages) and CSS (styling of the web pages).

Traditional performance tools, such as JMeter, only download this type of content but do not process them and to understand how the processing of them will impact an end user, we need to use other tools that, in isolation or combined with traditional tools, will be able provide more insights and might help answering some questions that are critical to the end user experience. Some of these questions are (but not limited to):

1. How long did it take for the first visible and useful content to be visible to the end user?
2. How long did it take for the whole page to be visible to the end user?
3. How long did it take for last visual change to fully load?
4. Will performance be impacted by user harder limitations (such as old devices etc.)?

To answer these sort of questions and understand the client-side performances I have used a mixed of tool that will serve different purposes

Sitespeed.io, Google Lighthouse and Google Performance

They all provide insights about speed and performance based on performance best practices and timing metric, ***however*** without any mimicked end user interaction with the target web app. **Google lighthouse** is available directly from the Chrome’s developer tools whereas **sitespeed.io** is an open-source project that can be run easily locally via a docker container.

Last one (Google Performance) allows to record performances directly in the browser by recording response time while reloading a page or performing manual actions. Used the automatic page reloading and record.

JMeter in combination with Wed Driver plugin and Firefox

This combines the power of JMeter (concurrent execution [users/threads] of multiple samplers/requests and reports on key metrics) with the power of Web Driver that help mimicking the **user interaction with the WebApp** (click on links, thinking time etc.)via a web browser (Firefox). This allows to build complex scenarios (e.g., logging in, search of a product, add to a shopping cart and checkout) that are real user patterns throughout the **WebApp** and report of key metrics.

The full definition of each metric can be found here <https://web.dev/performance-scoring/>, however it is important to categorise each metric within a category that will have a different impact on the user experience.

**Technical metrics:** Time To First Byte, Server Response Time, DOM Interactive, DOM Content Loaded, Page Load Time, Fully Loaded

**Visual metrics:** First Paint, First Contentful Paint, First Meaningful Paint, Speed Index, Visually Complete

**Interactive metrics:** First Interactive, Time To Interactive, First Input Delay, Consistently Interactive

API Performance

Also known as server-side performance tests, this type of non-function test in modern environments normally involve REST APIs (aka RESTful API). REST APIs are application programming interfaces that adhere to a REST (Representational state transfer) architecture while interacting with web services. The important part of this architecture is that requests are stateless, meaning that each request is independent from the previous and contains all necessary info to be processed by the accepting sever (that retains no session information whatsoever). The exchange of information between the receiving server and the REST client happens via JSON (JavaScript Object Notation) or XML (extensible markup language) format. The carried Information will be used by the front-end app to populate element of the DOM with content (e.g., Information about patience in a web form, account balance for Banking UI etc)

Performance Testing in the above context means ensuring that backend services respond to client requests within agreed key performance indicators (response time etc.)

Traditional performance tools, such as JMeter, are used within this context and will try to answer questions (but not limited to) like:

1. How will my application respond to different workload (different concurrency and different ramp up and run down, or a combination of both)?
2. What is the application upper limit for a given architecture configuration (CPU, Memory etc)?
3. Will my application fail/recover gracefully if the upper limit is reached/decreased?
4. What is the average, median, 90th pc, 95th pc and 99th pc response time?

To answer some of these questions and understand the server-side performances I have used JMETER.

**Performance Test Results**

**UI Performance**

Google Performance Tab: (Click on F12 and then follow instructions on the screen)

|  |
| --- |
| No JMeter Load |
|  |

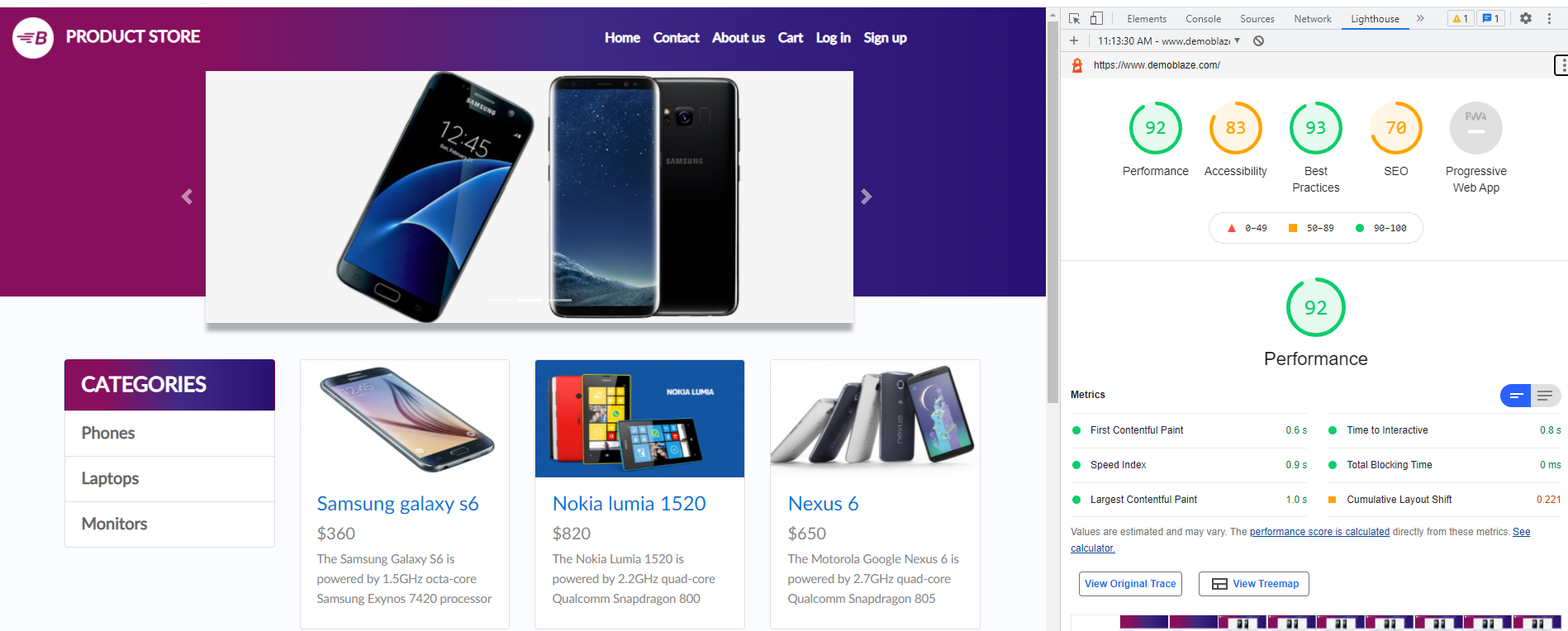
Responsiveness of the Home Page before adding any JMeter load was very fast: ~600ms

Responsiveness of the Home Page during JMeter load was moderately slow: ~6s

Most of the times I observed a response time between ~1.2sec and ~6sec, but a few times when I hit a ~20+sec

|  |
| --- |
| JMeter Load (1000 users for 15 sec) |
|  |

Google Lighthouse Test Results



Overall performance show above acceptable with a good “First Contentful Paint” and “speed”.

However, Lighthouse offers some suggestions to help improving page load time

Immagine che contiene testo

Descrizione generata automaticamente

Full Lighthouse report: <https://git.toptal.com/screening/Fabio-Santoro/-/tree/test-ci/performance/reports/lighthouse>

Available from Chrome Developer Tools.

Instructions:

* Open the target website
* Open Chrome Developer tools (F12 key)
* Go to Light house
* Chose options and click on “Generate Report”

Sitespeed.io Test Results

As comparison to Lighthouse, I have used sitespeed.io

Immagine che contiene tavolo

Descrizione generata automaticamente

Definitions can be found here: <https://git.toptal.com/screening/Fabio-Santoro/-/tree/test-ci/performance/reports/sitespeed.io/help.html>

Full report can be found here: <https://git.toptal.com/screening/Fabio-Santoro/-/tree/test-ci/performance/reports/sitespeed.io>

Instruction:

* CD into a results folder
* Run: docker run --rm -v "$(pwd):/sitespeed.io" sitespeedio/sitespeed.io:19.5.0 -b chrome https://www.demoblaze.com/

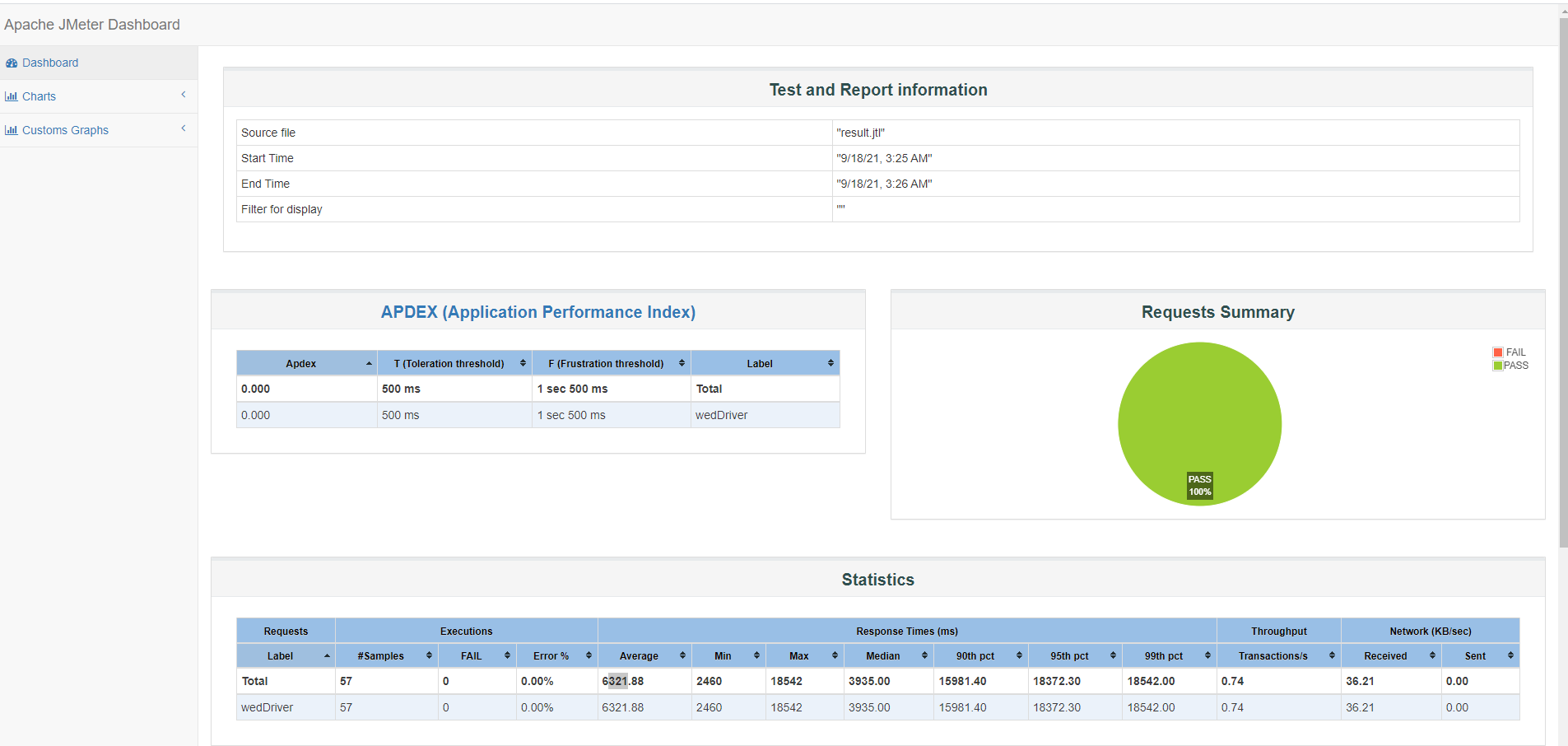
JMeter/WebDriver/Firefox Test Results

As other comparable/non comparable indicators have more significant differences (e.g., First Contentful Paint vs First Visual change etc) I thought it could be a good thing to explore further and conduct a third experiment using JMeter, WebDriver and Firefox with a simple script that opens the home page and wait until the first phone (Samsung) is visible (backend API call returned the data and they have been rendered).

The workload model:

1. 50 concurrent browser sessions for 15 seconds (more than this would require multiple machines as not enough network sockets available on one)
2. Rump up time of 0sec, which means all concurrent users were fired at once

The results confirmed the overall performance already captured by the other tools with an average Response Time of about 6 sec with 4 sec as median



Some, request took longer hence why the 90th, 95th and 99th pc show values in the range between ~16sec to ~18sec

As expected, Response time grew overtime with the number of concurrent threads getting higher, but no errors were observed.

Full report here: <https://git.toptal.com/screening/Fabio-Santoro/-/tree/test-ci/performance/reports/jmeter-reports>

Instructions: Can be found in the README.md file in the project repository under “performance”

Summary of the combined Analysis from a User **p**erspective:

Although the Sitespeed.io and Google Lighthouse show differences (overall score is 92 for Lighthouse and 85 for Sitespeed.io) some commonalities have been found in relation to performances:

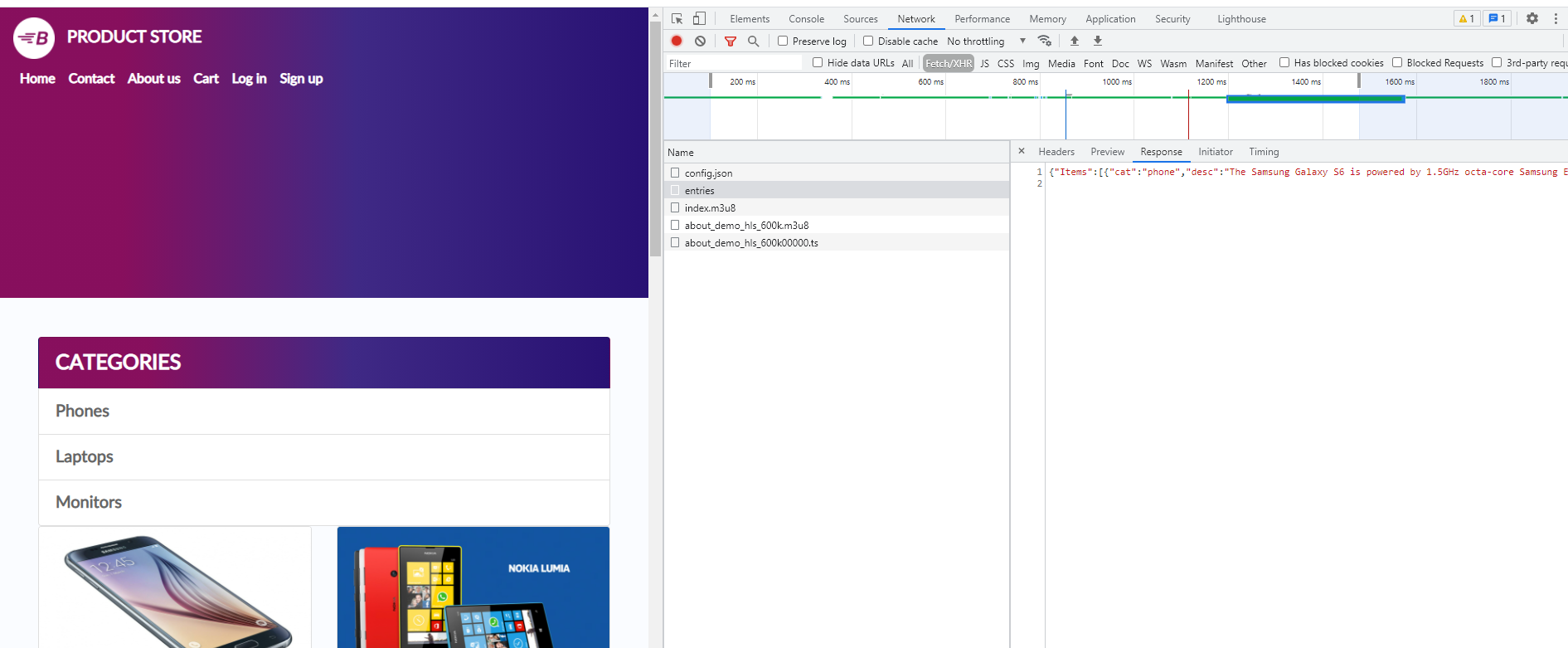
1. Speed Index acceptable/good for both
2. Same Cumulative Layouts shift

Additionally:

* Sitespeed.io shows 7.8sec for “Last Visual Change” (When page was somehow fully load) and ~6.2sec for “Visual Readiness” (time between the first change on the screen until the last on) which are in line with the average figure (~6.3) showed by the JMeter/WebDriver/Firefox (with some concurrency going on).
* Google performance Test also confirms that response time is affected by some decent concurrency if sustained for a period

**API Performance**

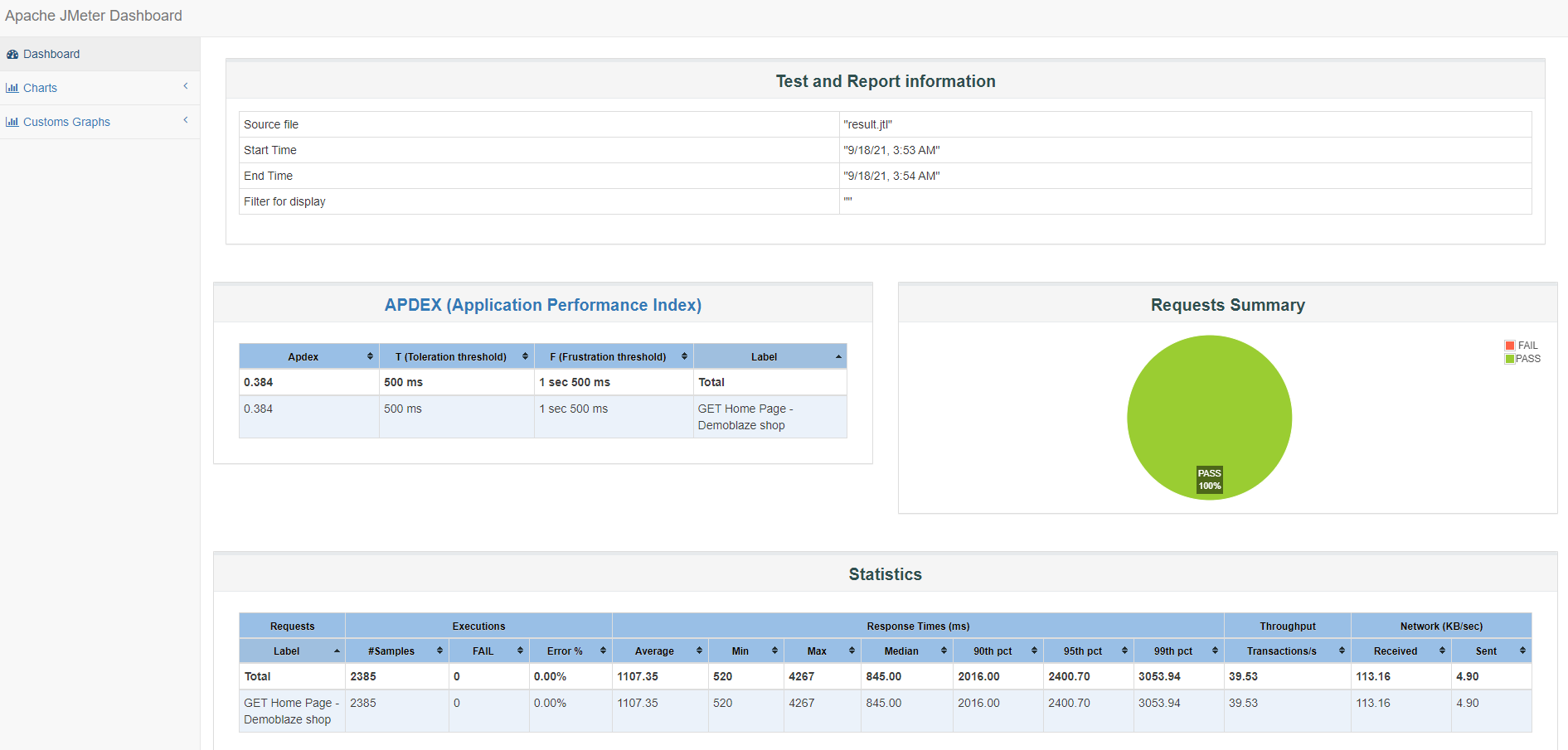
Inspecting the landing page of the target we can see only 4 files are relevant (after filtering for “XHR” request) but only one was an actual HTTP GET request (https://api.demoblaze.com/entries). The remaining were, a config.json file, 2 media files in text format (m3u8) and unexpectedly a .ts file (not sure why was retuned – could possibly be a bug)



A JMeter test plan was built with the only HTTP GET request (/entries) that was found on the home page and run against the target with the following workload model:

1. 1000 concurrent browser sessions for 15 seconds (more than this would require multiple machines as not enough network sockets available on one)
2. Rump up time of 60sec, which means about 16-17 users per second will be loaded until 1000 then the test will commence

Most of the results were line with findings from the other different approaches. The API seemed to be responsive and did not appear to be overwhelmed (no errors) by sudden high workload.

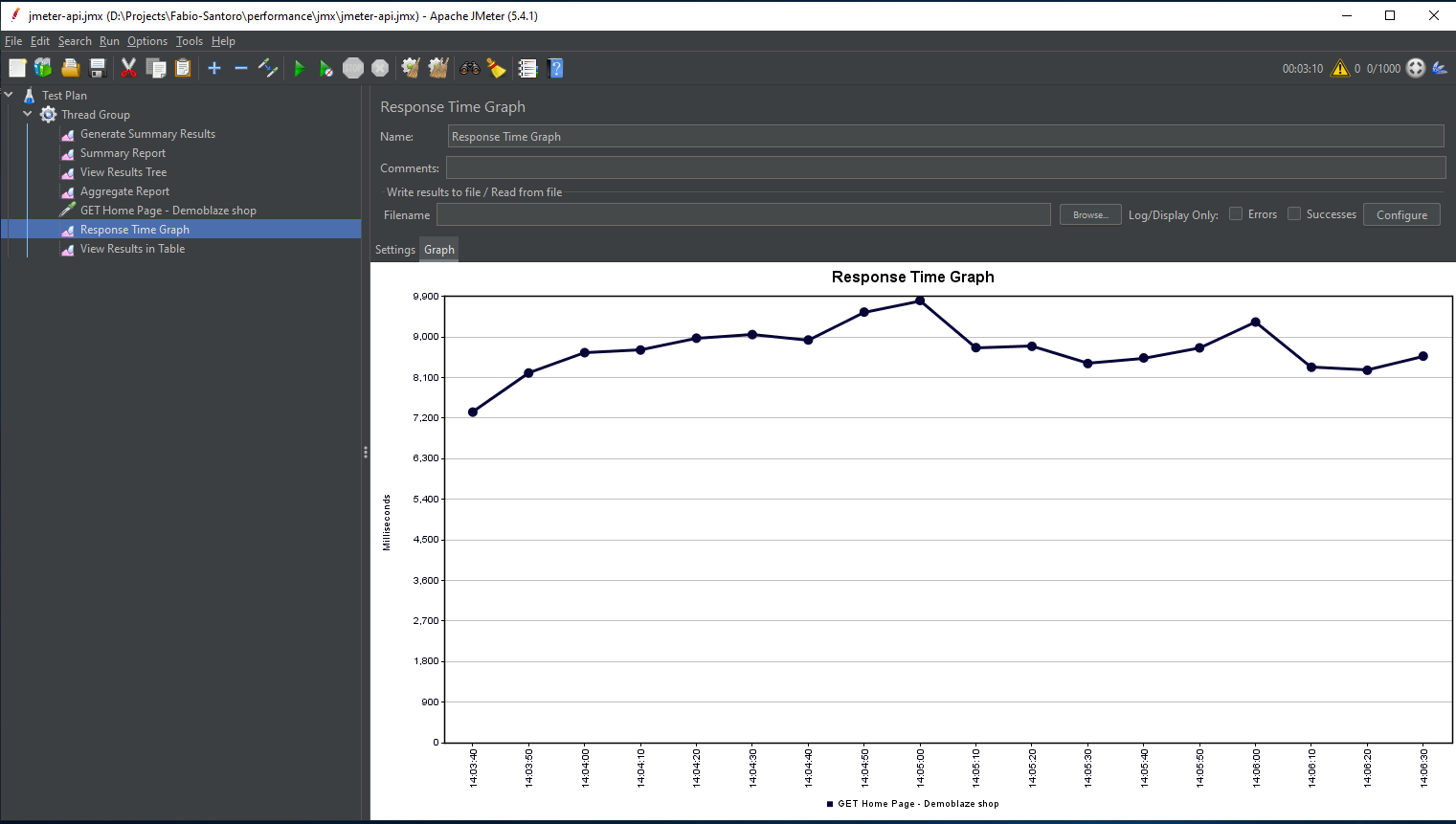


As the experiment was carried out only for a brief period (a spike of 15 sec),

However, as the response time increased after a few seconds I wanted to make sure that the service could run under the same load but for longer, so I repeat the test with same workload but for about 180sec. Interestingly the service managed to run without any errors, however all metrics were affected and so was the User experience. As we can see below the average and the median response time are higher than previously

Immagine che contiene testo

Descrizione generata automaticamente



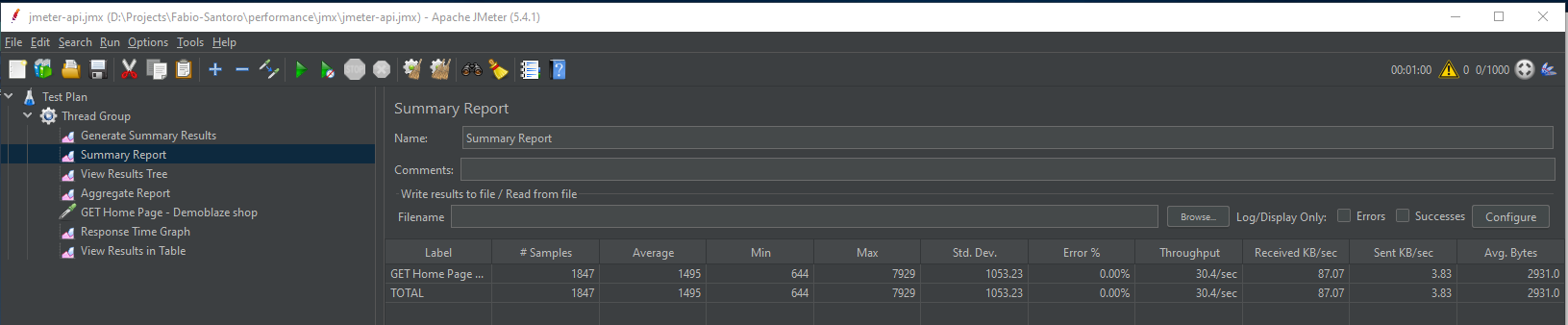
Which it is in line with the other analysis and indicates that the service in its current configuration might have an upper-level limit from which the performances will start deteriorating. An auto-scaling feature (adaptive to the demand) and a caching mechanism could act as protective mechanism against unexpected high-volume traffic or spikes in demand.

Questions:

  a) Explain the test in detail  
  b) Did the load test have an impact on web application response time?  
  c) What is the optimal application response time for modern day web applications?  
  d) Analyse few HTTP/S responses

Answers:

1. I believe the sections withing this document should explain the strategy and approach I followed for this engagement
2. The load did seem to have had somehow an impact on the service ‘s response time after a few seconds in the test exercise. Even if most of the results were consistent with the ones captured with other tools (much lower loads) and no errors were observed, the response increased overtime and the user experience got worse. An auto-scaling feature (adaptive computing power that would scales/descales with increased/decreased demand) and/or a caching mechanism could bring the service to the next level of responsiveness.
3. To my knowledge there is no universally agreed figures, and it all depends on the context (shock exchange orders are different from a small online business). However, speaking at a very high level this [article](https://www.nngroup.com/articles/website-response-times/) from Nielsen Norman Group (founded by pioneers and world class leaders in User Experience) highlight a few interesting key points that are worth considering as general guidelines and could represent a starting point for conversations to have with key stake holders:
   1. 100 milliseconds are perceived by users as instantaneous
   2. Between 1 and 2 seconds keeps the user's flow of thought seamless (users expect a page to be loaded within 2 sec)
   3. After 3 seconds users perceive the delay and start thinking to leave the website
   4. >=10 user will find hard to keep focussed and [engaged](https://www.nngroup.com/articles/how-long-do-users-stay-on-web-pages/)
4. Analysing the metadata of a few JMeter requests we can see that the loading time is very consistent (+- ~100/ms). However, it was observed that a few seconds in the test the loading time increased significantly to phase down again towards the end. See screenshots below for details



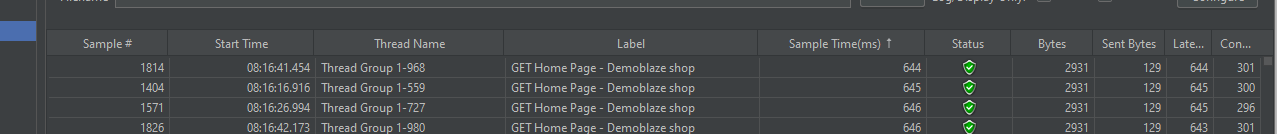


Immagine che contiene testo, elettronico, computer, screenshot

Descrizione generata automaticamente