Instituto Superior de Engenharia do Porto

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Project (First Part)
Individual report

QSOFT

Master in Informatics Engineering - 2024/2025

Porto, December 6, 2024

Pedro Miguel Santos Coelho

`Version 6, 2024-12-05`

Revision History

Revision	Date	Author(s)	Description
1	2024-11-26	Pedro Coelho	Initial version
2	2024-11-26	Pedro Coelho	Maintainability
3	2024-11-30	Pedro Coelho	Accessibility
4	2024-12-02	Pedro Coelho	Visual compatibility
5	2024-12-03	Pedro Coelho	Backend Performance
6	2024-12-05	Pedro Coelho	Backend Performance finish

Contents

List of Figures

1 Introduction

This individual report focuses on the specific contributions made to the second phase of the QSOFT project. The primary objective was to analyze and improve a JHipster-generated monolithic application by addressing key quality attributes: accessibility, visual compatibility, maintainability, performance, and security. My tasks included checking the accessibility using LightHouse and generating a HTML report with the accessibility statment of the application, check the Visual compatibility of the application on different endpoints with different devices and sizes, check the maintainability of the current application and verify the performance of the backend and frontend of the application, ensuring compliance with the Goal-Question-Metric (GQM). The report details my individual approach, methodologies, and results, providing insights into how these tasks contributed to the overall project goals.

2 Accessibility

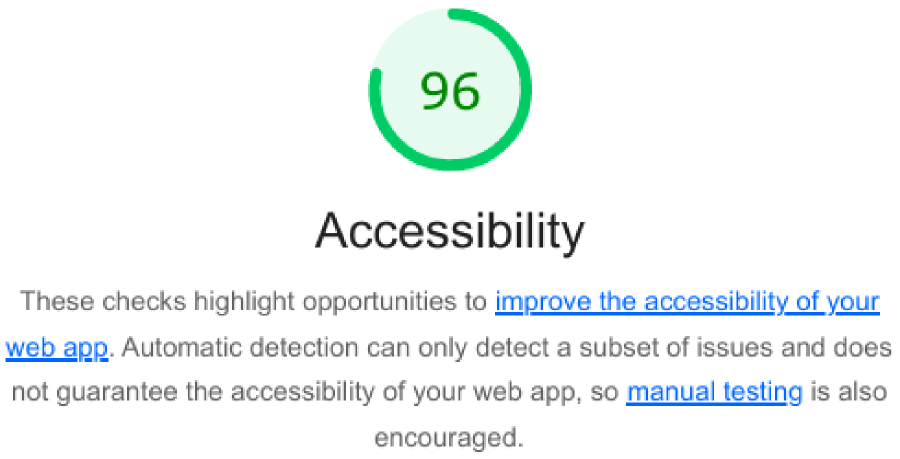
Web accessibility aims to develop digital content and applications that can be used by everyone, regardless of their physical, sensory, or cognitive abilities. This involves creating interfaces and features that are intuitive and functional for any user, promoting a barrier-free digital experience. The core idea is to ensure equal access to information and enable effective and inclusive interaction with the online environment.

2.1 LightHouse

To assess the accessibility of a front-end application, I will use Lighthouse, a tool designed to analyze web applications and provide detailed feedback on accessibility issues. Lighthouse helps identify areas that need improvement, ensuring the application becomes more inclusive for all users.

In addition, ARIA (Accessible Rich Internet Applications) attributes will be applied when necessary. These attributes add semantic meaning to HTML elements, improving navigation and interaction for users who rely on assistive technologies such as screen readers. This process focuses on evaluating and enhancing the accessibility of the selected front-end page.

The page I evaluated was the visualise page for the petType. As shown in Figure 2.1, the application scored 96 out of 100 on the accessibility scale. This high score indicates that the front end developed for this page adheres to key principles of web accessibility, providing a positive and inclusive experience for users.


96

Accessibility

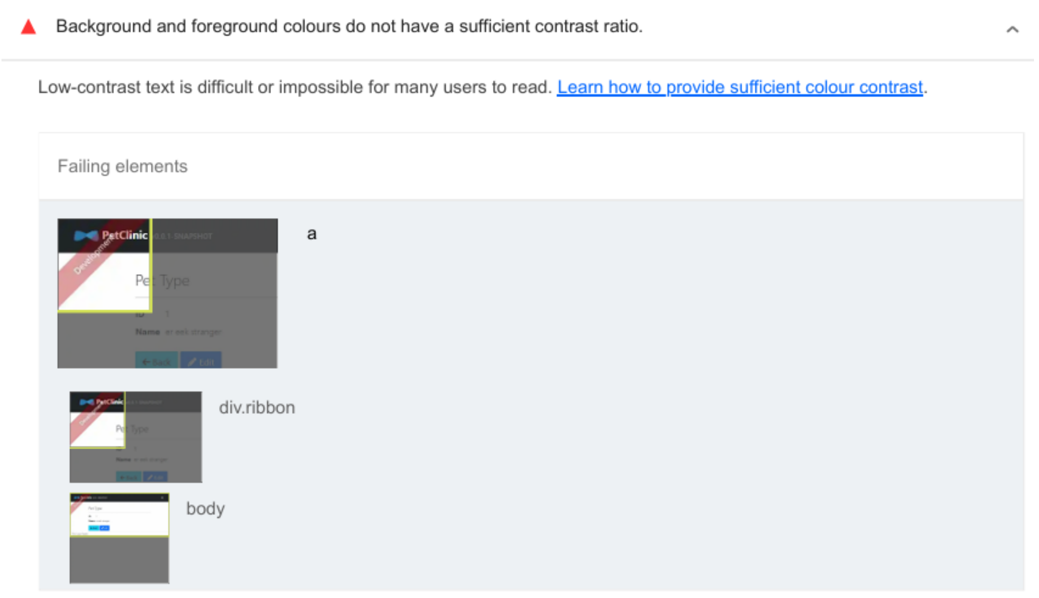
These checks highlight opportunities to improve the accessibility of your web app. Automatic detection can only detect a subset of issues and does not guarantee the accessibility of your web app, so manual testing is also encouraged.

Figure 2.1: Accessibility Level before changes

Following the report generated by the Lighthouse application, I identified that the main issue affecting accessibility is related to color contrast, as seen in the figure 2.3.

Specifically, the contrast between foreground and background colors is insufficient, making it difficult or impossible for many users to read the text.

This issue is due to the application being generated with JHipster, which includes a red development banner by default. The low contrast between the banner's text and background contributes to the reported problem. To address this, adjustments to the color scheme will be required to ensure sufficient contrast and improve readability for all users.


Background and foreground colours do not have a sufficient contrast ratio.

Low-contrast text is difficult or impossible for many users to read. Learn how to provide sufficient colour contrast.

Failing elements

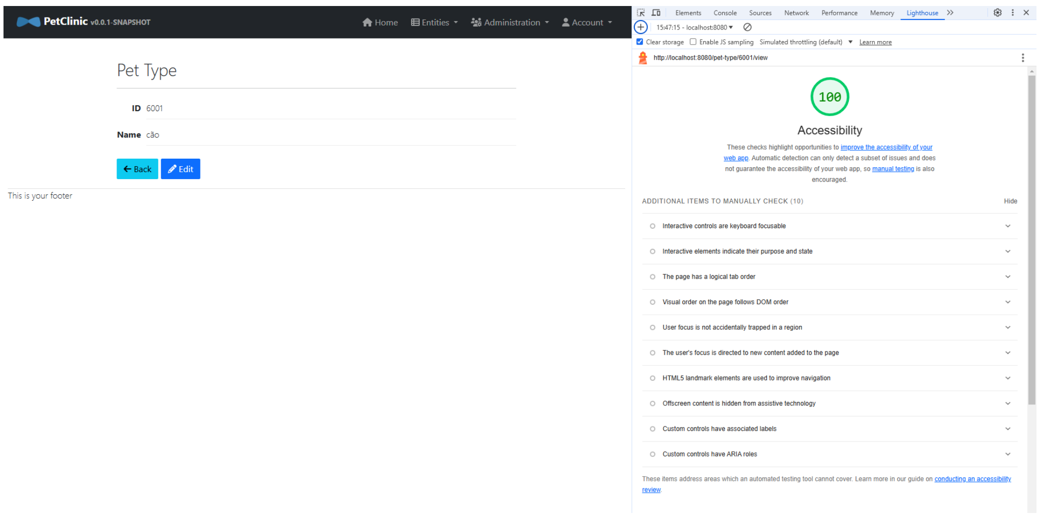
a

div.ribbon

body

Figure 2.2: Contrast report

If we were to remove the development banner added by default in JHipster, the Lighthouse accessibility score would reach 100%, as this is the only issue identified in the report. The banner's insufficient color contrast is the sole factor affecting the accessibility evaluation, and its removal would ensure full compliance with accessibility standards, providing an optimal user experience.



Pet Type

ID 6001

Name cão

This is your footer

15:47:15 - localhost:8080

Clear storage · Enable JS sampling · Simulated throttling (default) · Learn more

http://localhost:8080/pet-type/6001/view

100

Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Automatic detection can only detect a subset of issues and does not guarantee the accessibility of your web app, so manual testing is also encouraged.

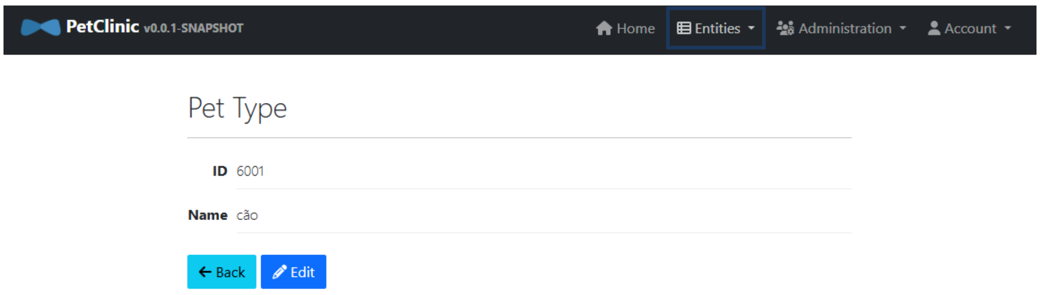
ADDITIONAL ITEMS TO MANUALLY CHECK (10)

- Interactive controls are keyboard focusable
- Interactive elements indicate their purpose and state
- The page has a logical tab order
- Visual order on the page follows DOM order
- User focus is not accidentally trapped in a region
- The user's focus is directed to new content added to the page
- HTML5 landmark elements are used to improve navigation
- Offscreen content is hidden from assistive technology
- Custom controls have associated labels
- Custom controls have ARIA roles

These items address areas which an automated testing tool cannot cover. Learn more in our guide on conducting an accessibility review.

Figure 2.3: Accessibility Level after changes

In addition, I was able to navigate through the entire page using my keyboard, and the selected elements had an outline with a different color. However, in some areas, the color of the outline could be more noticeable as seen in the figure 2.5.

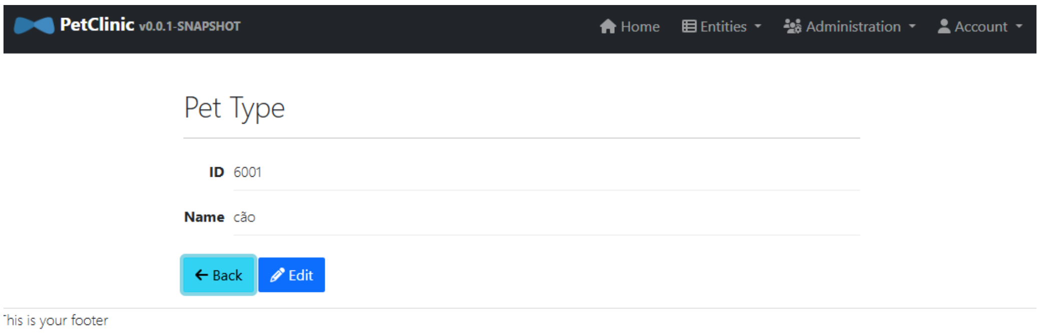


Pet Type

ID 6001

Name cão

Figure 2.4: Navigate to Entities via Keyboard



Pet Type

ID 6001

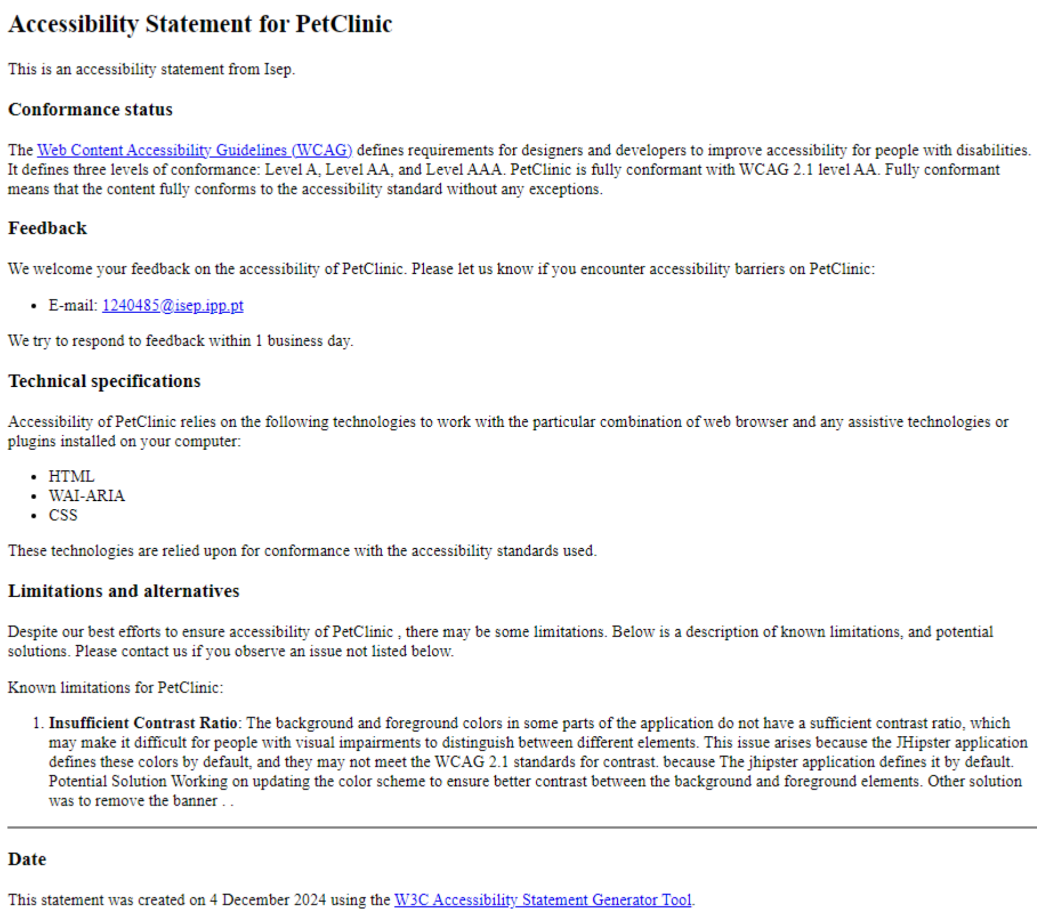
Name cão

'his is your footer

Figure 2.5: Navigate to Back via Keyboard

2.2 Web Accessibility Initiative

By generating the Accessibility Statement from the *https://www.w3.org* website, I ended up with the following HTML page.



Accessibility Statement for PetClinic

This is an accessibility statement from Isep.

Conformance status

The Web Content Accessibility Guidelines (WCAG) defines requirements for designers and developers to improve accessibility for people with disabilities. It defines three levels of conformance: Level A, Level AA, and Level AAA. PetClinic is fully conformant with WCAG 2.1 level AA. Fully conformant means that the content fully conforms to the accessibility standard without any exceptions.

Feedback

We welcome your feedback on the accessibility of PetClinic. Please let us know if you encounter accessibility barriers on PetClinic:

- E-mail: 1240485@isep.ipp.pt

We try to respond to feedback within 1 business day.

Technical specifications

Accessibility of PetClinic relies on the following technologies to work with the particular combination of web browser and any assistive technologies or plugins installed on your computer:

- HTML
- WAI-ARIA
- CSS

These technologies are relied upon for conformance with the accessibility standards used.

Limitations and alternatives

Despite our best efforts to ensure accessibility of PetClinic , there may be some limitations. Below is a description of known limitations, and potential solutions. Please contact us if you observe an issue not listed below.

Known limitations for PetClinic:

1. **Insufficient Contrast Ratio**: The background and foreground colors in some parts of the application do not have a sufficient contrast ratio, which may make it difficult for people with visual impairments to distinguish between different elements. This issue arises because the JHipster application defines these colors by default, and they may not meet the WCAG 2.1 standards for contrast. because The jhipster application defines it by default. Potential Solution Working on updating the color scheme to ensure better contrast between the background and foreground elements. Other solution was to remove the banner . .

Date

This statement was created on 4 December 2024 using the W3C Accessibility Statement Generator Tool.

Figure 2.6: Html accessibility page

Due to time constraints, I was unable to integrate this page into the project at the URL *localhost:8080/pet-type/acessibilidade*. However, the HTML page can be found in the Accessibility folder of the 1240485 student.

2.3 Conclusion

The accessibility assessment of the application using Lighthouse revealed a strong performance, scoring 96 out of 100. The sole issue identified was related to the color contrast of the default "Development" banner generated by JHipster. Addressing this minor issue by either removing or adjusting the banner would result in a perfect accessibility score of 100%, demonstrating the application’s adherence to high accessibility standards.

Based on this, I rate the application of 5 out of 5 on accessibility chapter.

3 Visual compatibility

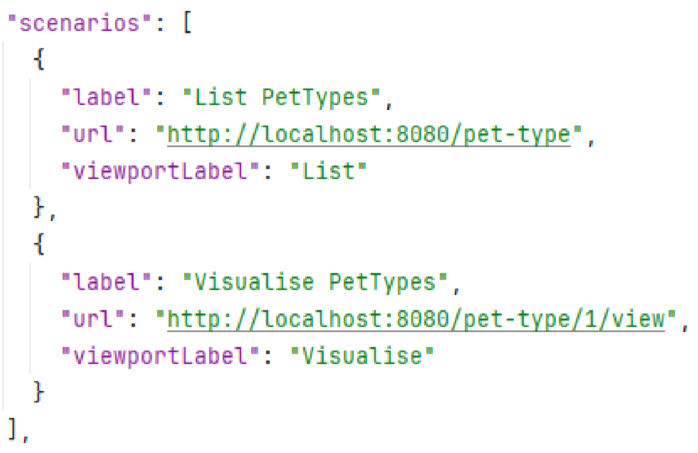
Visual compatibility refers to the seamless interaction of visual elements, ensuring they work harmoniously to create a cohesive and appealing design. It functions by balancing key factors such as color harmony, typography, proportion, layout, and style consistency. When these elements are thoughtfully integrated, they enhance the clarity, usability, and emotional resonance of a design.

3.1 Tests

In the same way that we discussed accessibility in the previous chapter, I will now address the visual compatibility of the Visualise Page of PetTypes, in addition I will also analyze the List Page of the PetTypes. The tests will be conducted with BackstopJS with the intention of verifying if the visualise page of the PetTypes Domain of our web application is readable across different devices, such as mobile phones and tablets or a computer.

3.2 Scenarios

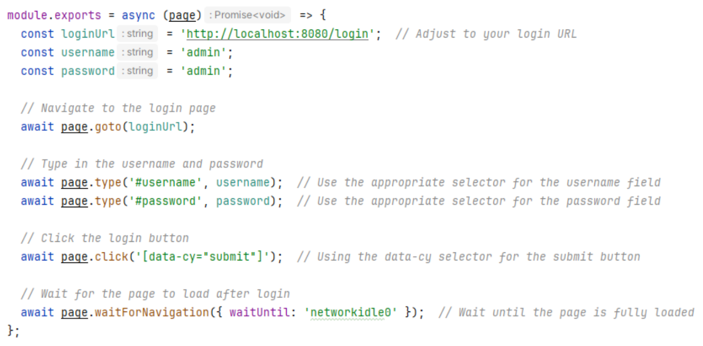
In the figure 3.1 we can see the configuration for endpoints that are being tested in this chapter.



```
"scenarios": [
  {
    "label": "List PetTypes",
    "url": "http://localhost:8080/pet-type",
    "viewportLabel": "List"
  },
  {
    "label": "Visualise PetTypes",
    "url": "http://localhost:8080/pet-type/1/view",
    "viewportLabel": "Visualise"
  }
],
```

Figure 3.1: Configuration on json for the endpoints

Considering that the application has an authentication system, the login was defined in the onBefore file to make the pages accessible during testing. This can be seen and verified in the figure 3.2



```
module.exports = async (page) :Promise<void>  => {
  const loginUrl :string  = 'http://localhost:8080/login';  // Adjust to your login URL
  const username :string  = 'admin';
  const password :string  = 'admin';

  // Navigate to the login page
  await page.goto(loginUrl);

  // Type in the username and password
  await page.type('#username', username);  // Use the appropriate selector for the username field
  await page.type('#password', password);  // Use the appropriate selector for the password field

  // Click the login button
  await page.click('[data-cy="submit"]');  // Using the data-cy selector for the submit button

  // Wait for the page to load after login
  await page.waitForNavigation({ waitUntil: 'networkidle0' });  // Wait until the page is fully loaded
};
```

Figure 3.2: Configuration on onBefore.js for authentication

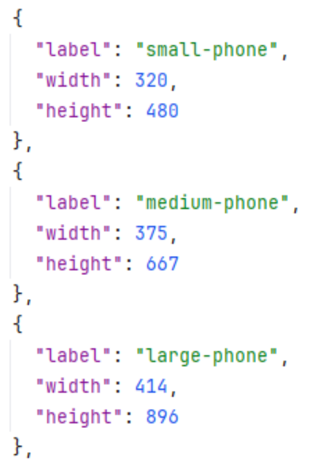
3.3 Viewports

3.3.1 Mobile Phones

In the table below, we can see the dimensions used for the mobile phone tests. Below, in the figure 3.3, we can observe the configuration file for mobile phone testing.

Type	Label	Width (px)	Height (px)
Small Phone	small-phone	320	480

Type	Label	Width (px)	Height (px)
Medium Phone	medium-phone	375	667
Large Phone	large-phone	414	896



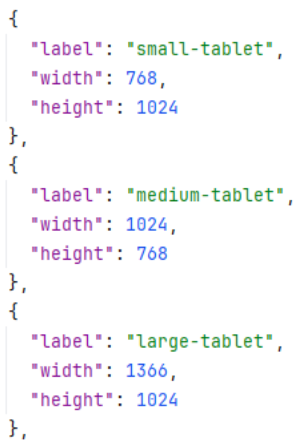
```
{
  "label": "small-phone",
  "width": 320,
  "height": 480
},
{
  "label": "medium-phone",
  "width": 375,
  "height": 667
},
{
  "label": "large-phone",
  "width": 414,
  "height": 896
},
```

Figure 3.3: Configuration on json for Mobile Phones

3.3.2 Tablets

n this section, we present the dimensions used for the tablet tests. Below, in the figure 3.4, we can observe the configuration file for tablet testing.

Type	Label	Width (px)	Height (px)
Small Tablet	small-tablet	768	1024
Medium Tablet	medium-tablet	1024	768
Large Tablet	large-tablet	1366	1024



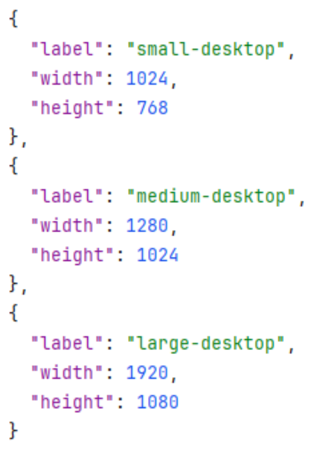
```
{
  "label": "small-tablet",
  "width": 768,
  "height": 1024
},
{
  "label": "medium-tablet",
  "width": 1024,
  "height": 768
},
{
  "label": "large-tablet",
  "width": 1366,
  "height": 1024
},
```

Figure 3.4: Configuration on json for Tablets

3.3.3 Desktops

The table below shows the dimensions used for desktop tests. Below, in the figure 3.5, we can observe the configuration file for desktop testing.

Type	**Label**	**Width (px)**	**Height (px)**
Small Desktop	small-desktop	1024	768
Medium Desktop	medium-desktop	1280	1024
Large Desktop	large-desktop	1920	1080

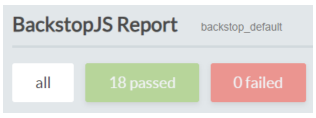


```
{
  "label": "small-desktop",
  "width": 1024,
  "height": 768
},
{
  "label": "medium-desktop",
  "width": 1280,
  "height": 1024
},
{
  "label": "large-desktop",
  "width": 1920,
  "height": 1080
}
```

Figure 3.5: Configuration on json for Desktop

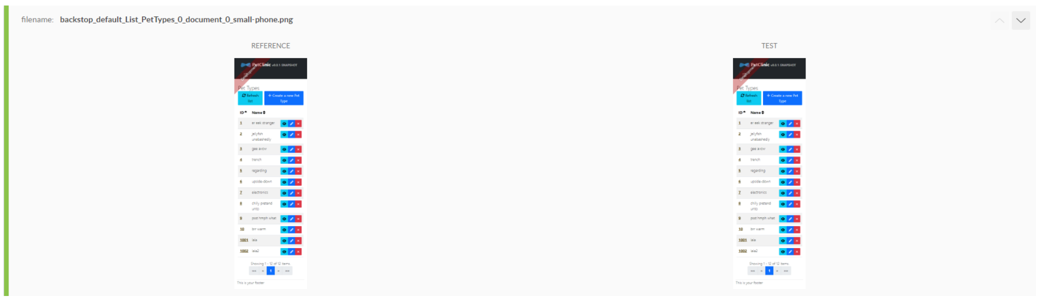
3.4 Test results

In total, 18 tests were performed. These tests covered each endpoint (Visualise and List), across different device types (phone, tablet, desktop), and for various screen sizes (small, medium, large). As we can see in the figure 3.6 all 18 tests passed, which means that the test images that came from the "backstop test" were exactly equal to the references images created from the "backstop reference". An example of this comparison can be seen in the figure 3.7


BackstopJS Report backstop_default

all | 18 passed | 0 failed

Figure 3.6: BackstopJS Results


filename: backstop_default_List_PetTypes_0_document_0_small-phone.png

REFERENCE

TEST

Figure 3.7: BackstopJS Result Example

3.5 Conclusion

The visual compatibility tests conducted in this chapter provide a comprehensive assessment of the application's interface consistency across various devices and screen sizes. Considering that all tests were successful, we can confirm that the application delivers a consistent and reliable visual experience, regardless of the device or screen size. This consistency is essential for maintaining a high-quality user experience across different platforms.

Based on these results, we can confidently conclude that the application meets the necessary standards for visual compatibility, and the overall performance in this area is rated 5 out of 5, reflecting flawless performance across all tested environments.

4 Maintainability

Maintainability refers to the ease with which a software system can be modified to adapt to new requirements, correct defects, or improve performance after deployment. It ensures the long-term sustainability and adaptability of the application. In this phase of the project, relevant maintainability metrics, such as Coupling, Structural Erosion, Size, and Complexity, will be analyzed. The assessment will be conducted using tools and methodologies aligned with best practices, as detailed in the following sections.

4.1 Coupling and Structural Erosion

- **Metric:** Average Component Dependency (ACD)
- **Category:** Cohesion/Coupling (John Lakos)
- **Value:** 4.36

As seen in the Figure 4.1 ACD value of 4.36 indicates that, on average, each component in the system depends on approximately 4 other components, both directly and indirectly. This relatively low level of dependency suggests a good degree of modularity within the architecture, where components are less tightly coupled. Such an architecture facilitates independent functionality and makes it easier to modify or refactor components with minimal risk of unintended interactions.

A lower ACD value generally reflects better maintainability, as the reduced dependencies increase the flexibility of individual components and decrease the likelihood of cascading changes when updates are made. This level of cohesion promotes a system design that is easier to evolve and maintain over time. However, it remains essential to ensure that the reduced dependency does not compromise the necessary interactions and integrations within the system.

Element [1]	ACD
PetClinic_Part2	4,36

Figure 4.1: ACD

4.2 Size and Complexity

- **Metric:** Number of Components/Sources
- **Category:** Size
- **Value:** 107

As seen in the figure 4.2 project consists of 107 components or source files, which provides a structural overview of its size and organization. In the context of a codebase with approximately 5000 lines of code, the distribution of 107 components suggests an average of approximately 47 lines per component. This number indicates a relatively well-organized project structure, where code is compartmentalized across manageable components. Such organization enhances readability, traceability, and modularity, contributing to the maintainability of the system by making individual components easier to understand, modify, and test.

Element [1]	Number of Components/Sources
PetClinic_Part2	107

Figure 4.2: Number Of Components/Sources

4.3 Conclusion

The maintainability analysis of the project demonstrates strong adaptability and long-term sustainability. The Average Component Dependency (ACD) value of **4.36** reflects a modular architecture with low coupling, making the system easier to modify and reducing the risk of unintended interactions. Additionally, with **107 components** and an average of **47 lines of code per component**, the codebase is well-organized and modular, enhancing readability and ease of testing. These characteristics ensure the system's ability to evolve, address defects, and maintain performance over time.

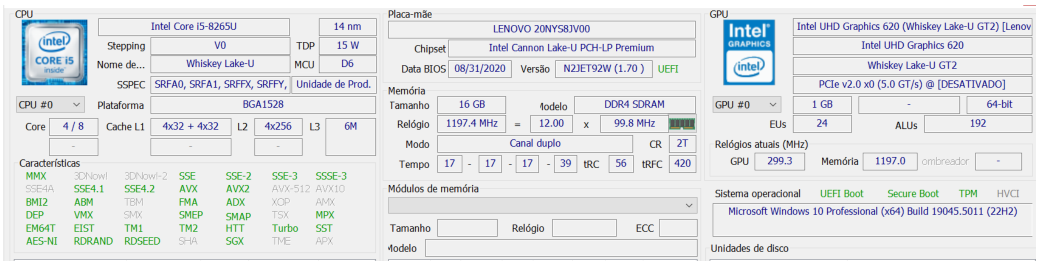
Overall, the maintainability of the project is strong, with a well-organized, modular architecture and low coupling, ensuring long-term sustainability. However, there is still room for optimization, particularly in reducing dependencies and refining the number of components. Combining these factors, maintainability is rated 4 out of 5, reflecting a solid foundation with opportunities for further improvement.

5 Performance

Just like in the Project 1, I will be evaluating the performance of the application. However, unlike the first project, now I will evaluate the front-end performance of the application as well. In order to test the backend I will adapt the tests used in the project 1, both for the jmeter and for k6. To evaluate the performance of the frontend I used LightHouse, the same application used for the Accessibility chapter.

5.1 Testing Environment

All tests conducted for this evaluation were performed on a single machine/computer, with the specifications outlined in the image below (5.1). It's important to note that the computer remained consistently connected to the internet via an Ethernet cable throughout the testing process. This ensures a consistent and standardized testing environment, allowing for accurate and reliable performance assessments under the specified configurations. The use of a wired connection guarantees stable network conditions, contributing to the precision of the results obtained during the evaluation.



```
CPU
            Intel Core i5-8265U                          14 nm
Stepping    V0                                    TDP    15 W
Nome de...  Whiskey Lake-U                        MCU    D6
SSPEC       SRFA0, SRFA1, SRFFX, SRFFY,  Unidade de Prod.
CPU #0      Plataforma  BGA1528
Core  4 / 8   Cache L1  4x32 + 4x32   L2  4x256   L3  6M
      -                 -                 -
Características
MMX     3DNow!   3DNow!-2  SSE    SSE-2   SSE-3    SSSE-3
SSE4A   SSE4.1   SSE4.2    AVX    AVX2    AVX-512  AVX10
BMI2    ABM      TBM       FMA    ADX     XOP      AMX
DEP     VMX      SMX       SMEP   SMAP    TSX      MPX
EM64T   EIST     TM1       TM2    HTT     Turbo    SST
AES-NI  RDRAND   RDSEED    SHA    SGX     TME      APX

Placa-mãe
            LENOVO 20NYS8JV00
Chipset     Intel Cannon Lake-U PCH-LP Premium
Data BIOS   08/31/2020   Versão  N2JET92W (1.70 )   UEFI
Memória
Tamanho  16 GB          Modelo  DDR4 SDRAM
Relógio  1197.4 MHz  =  12.00  x  99.8 MHz
Modo     Canal duplo                  CR    2T
Tempo    17 - 17 - 17 - 39  tRC  56   tRFC  420
Módulos de memória
Tamanho          Relógio          ECC
Modelo

GPU
         Intel UHD Graphics 620 (Whiskey Lake-U GT2) [Lenov
         Intel UHD Graphics 620
         Whiskey Lake-U GT2
         PCIe v2.0 x0 (5.0 GT/s) @ [DESATIVADO]
GPU #0   1 GB        -        64-bit
EUs      24          ALUs     192
Relógios atuais (MHz)
GPU  299.3   Memória  1197.0   ombreador  -
Sistema operacional   UEFI Boot   Secure Boot   TPM   HVCI
Microsoft Windows 10 Professional (x64) Build 19045.5011 (22H2)
Unidades de disco
```

Figure 5.1: System Details

5.2 Backend performance

In the following sections, we will explore specific performance metrics, analyze the results obtained from JMeter and K6 tests, and draw conclusions about the project's suitability for integration into a larger system based on its performance attributes.

5.2.1 Requirements

Normal Conditions

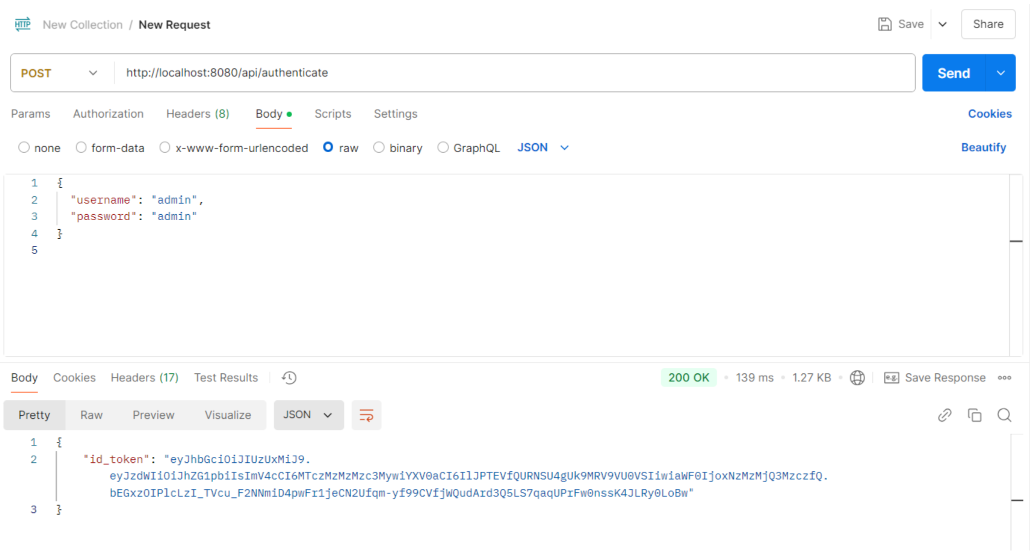
- Supports normal user access of 50 users.
- Respond to any type of request in less than 3 seconds 99% of the time.
- Process more than 250 requests per second.

Heavy Conditions

- Supports heavy user access of up to 200 users.
- Respond to any type of request in less than 5 seconds 99% of the time.
- Process more than 150 requests per second.

5.2.2 Authentication

Due to the application requiring authentication to perform actions on its endpoints, it became necessary to generate a Bearer Token for use in JMeter and K6 tests. To accomplish this, I used Postman, as shown in the figure 5.2. By logging in with valid credentials, the application returns a token, which can then be used to authenticate subsequent requests in JMeter or K6 tests, as demonstrated later in the figure 5.3 for jmeter and in the figure 5.4. This procedure is applied in all the tests presented in this chapter..



POST http://localhost:8080/api/authenticate

Body: none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

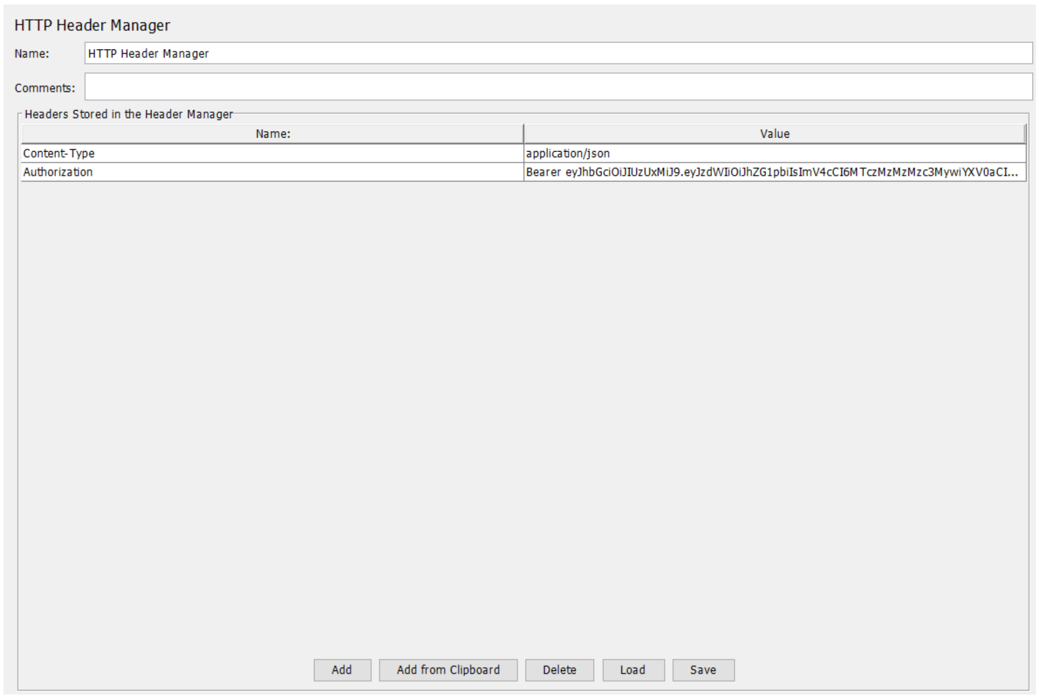
```
{
  "username": "admin",
  "password": "admin"
}
```

Body Cookies Headers (17) Test Results — 200 OK • 139 ms • 1.27 KB

Pretty Raw Preview Visualize JSON

```
{
    "id_token": "eyJhbGciOiJIUzUxMiJ9.
      eyJzdWIiOiJhZG1pbiIsImV4cCI6MTczMzMzMzc3MywiYXV0aCI6IlJPTEVfQURNSU4gUk9MRV9VU0VSIiwiaWF0IjoxNzMzMjQ3MzczfQ.
      bEGxzOIPlcLzI_TVcu_F2NNmiD4pwFr1jeCN2Ufqm-yf99CVfjWQudArd3Q5LS7qaqUPrFw0nssK4JLRy0LoBw"
}
```

Figure 5.2: Authentication via Postman



HTTP Header Manager

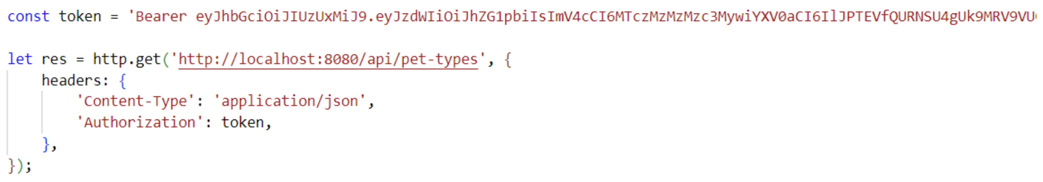
Name: HTTP Header Manager

Comments:

Headers Stored in the Header Manager

Name:	Value
Content-Type	application/json
Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsImV4cCI6MTczMzMzMzc3MywiYXV0aCI...

Figure 5.3: Usage of Token on jmeter tests



```
const token = 'Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsImV4cCI6MTczMzMzMzc3MywiYXV0aCI6IlJPTEVfQURNSU4gUk9MRV9VU

let res = http.get('http://localhost:8080/api/pet-types', {
    headers: {
        'Content-Type': 'application/json',
        'Authorization': token,
    },
});
```

Figure 5.4: Usage of Token on k6 tests

5.2.3 Load Test

Load testing evaluates a system's behavior under typical conditions by simulating realistic user activity. This process enables the early identification of performance issues, such as response time delays and resource overuse, even at normal load levels. Additionally, it provides insights into the system's capacity, supporting scalability planning to ensure it can handle anticipated user growth effectively. In this section we will evaluate the Loads tests to the endpoints POST of petType

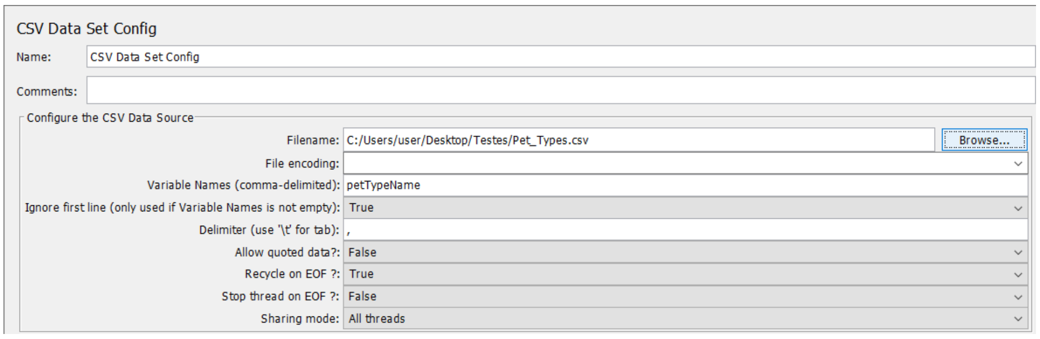
Scenario POST

- 50 users of the web application
- Posting a new PetTypes
- The system returns 201
- Normal Conditions

Jmeter

Test Configuration

In the figure 5.5 we can see the configuration needed to load the csv file on jmeter, later, on the figure 5.6 we can see how to use the same data from the csv for the POST requests. In the figure 5.7 we can see the configuration on jmeter for the test where we can see that this test simulates a ramp-up to 50 virtual users over 10 seconds, maintains this load for 90 seconds, and then ramps down to 0 users over 10 seconds. In the figure 5.8 we can see the configuration for the duration for each request in jmeter of 3 seconds.



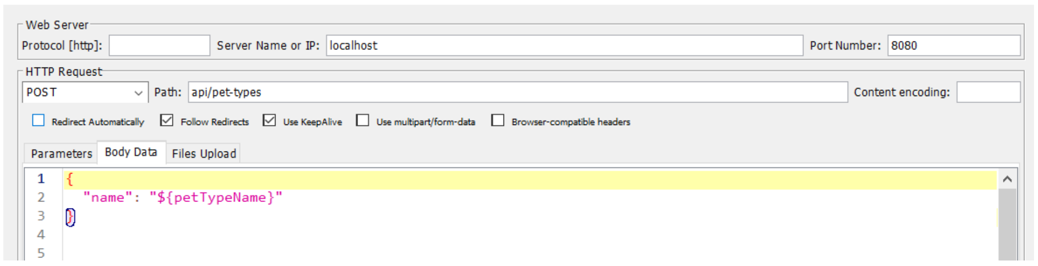
CSV Data Set Config

Name:	CSV Data Set Config
Comments:	

Configure the CSV Data Source

Filename:	C:/Users/user/Desktop/Testes/Pet_Types.csv
File encoding:	
Variable Names (comma-delimited):	petTypeName
Ignore first line (only used if Variable Names is not empty):	True
Delimiter (use '\t' for tab):	,
Allow quoted data?:	False
Recycle on EOF ?:	True
Stop thread on EOF ?:	False
Sharing mode:	All threads

Figure 5.5: Load Data from CSV on jmeter



Web Server

Protocol [http]:	
Server Name or IP:	localhost
Port Number:	8080

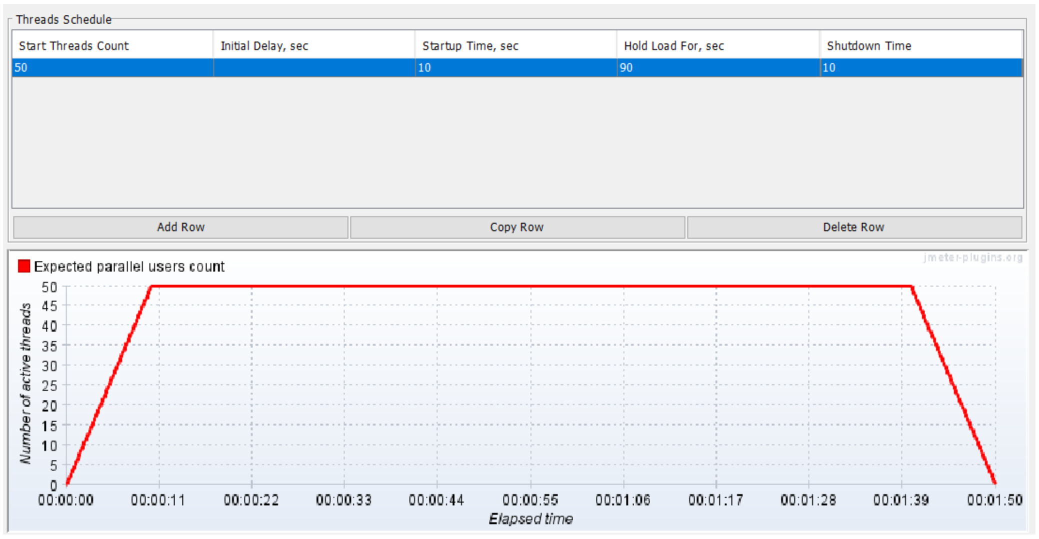
HTTP Request

Method:	POST
Path:	api/pet-types
Content encoding:	

☐ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive ☐ Use multipart/form-data ☐ Browser-compatible headers

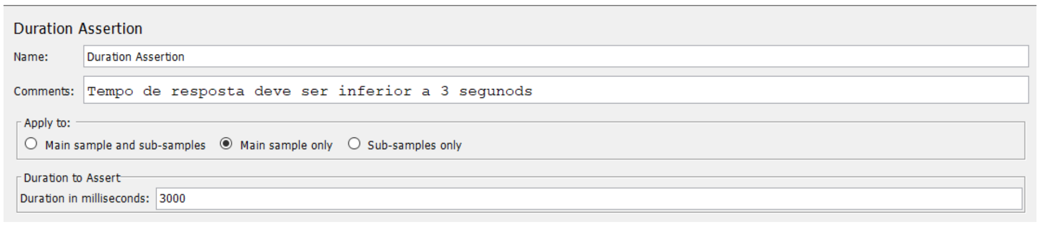
```
{
  "name": "${petTypeName}"
}
```

Figure 5.6: Use Data from CSV on jmeter


Threads Schedule

Start Threads Count	Initial Delay, sec	Startup Time, sec	Hold Load For, sec	Shutdown Time
50		10	90	10

Figure 5.7: Load Test Configuration on jMeter


Duration Assertion

Name: Duration Assertion

Comments: Tempo de resposta deve ser inferior a 3 segunods

Apply to: ( ) Main sample and sub-samples [✓ Main sample only] ( ) Sub-samples only

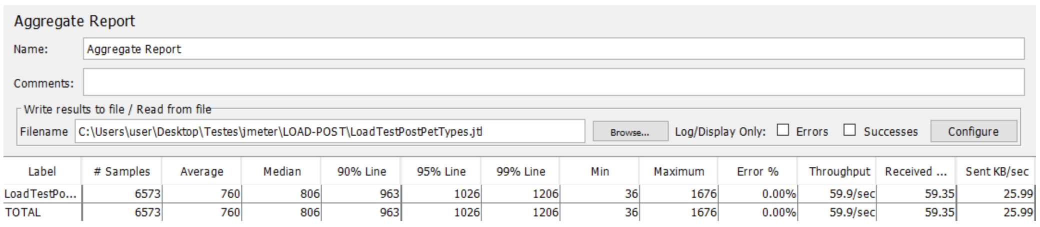
Duration to Assert

Duration in milliseconds: 3000

Figure 5.8: Load Duration Configuration on jMeter

Results

In the figure 5.9 we can see the Aggregate Report of the jmeter test. In the figure 5.10 we can see the response time over time of the jmeter test.


Aggregate Report

Name: Aggregate Report

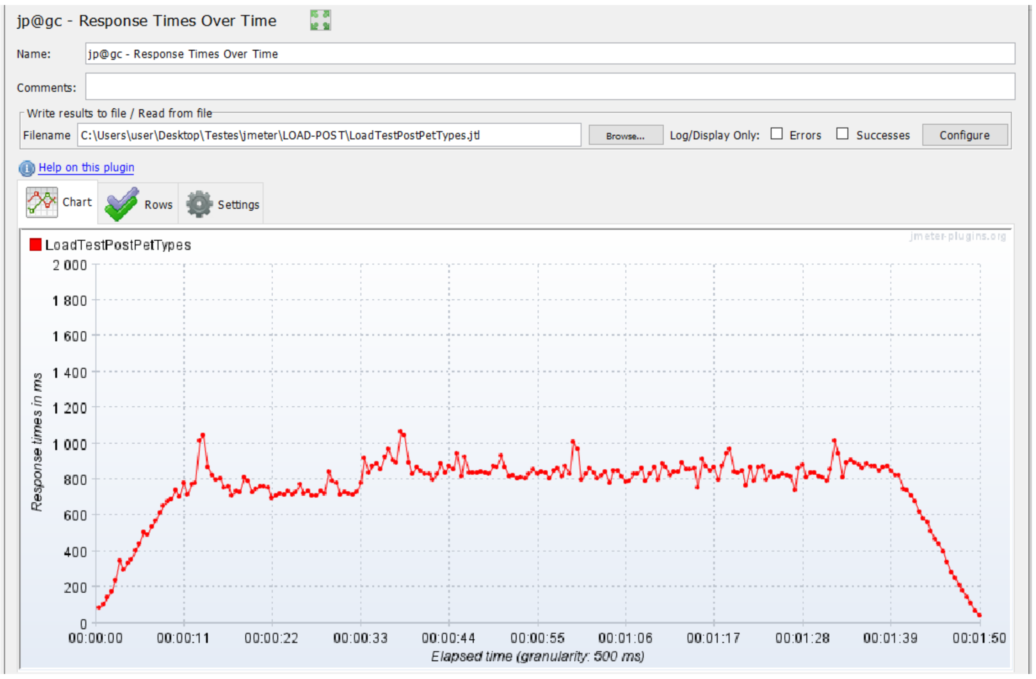
Comments:

Write results to file / Read from file

Filename C:\Users\user\Desktop\Testes\jmeter\LOAD-POST\LoadTestPostPetTypes.jtl Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
LoadTestPo...	6573	760	806	963	1026	1206	36	1676	0.00%	59.9/sec	59.35	25.99
TOTAL	6573	760	806	963	1026	1206	36	1676	0.00%	59.9/sec	59.35	25.99

Figure 5.9: Load Aggregate Report


jp@gc - Response Times Over Time

Name: jp@gc - Response Times Over Time

Comments:

Write results to file / Read from file

Filename C:\Users\user\Desktop\Testes\jmeter\LOAD-POST\LoadTestPostPetTypes.jtl Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

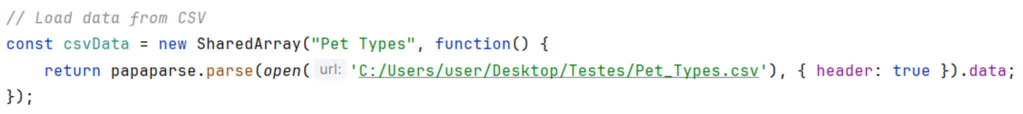
Figure 5.10: Load Response Time

Based on these results, we can conclude that the application demonstrated solid performance throughout the load test. The maximum response time was recorded at 1.6 seconds, with no errors observed during the test. The response time graph indicates that the application maintained consistent performance over the test duration, with no significant deviations or performance degradation. Additionally, other key metrics, such as average response time and throughput, confirm the stability and reliability of the application under load.

K6

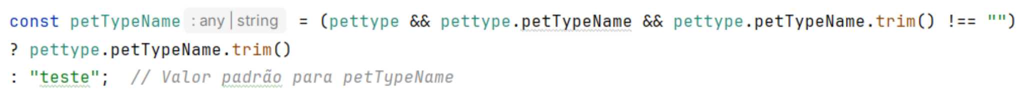
Test Configuration

In the figure 5.11 we can see the configuration needed to load the csv file on K6, later, on the figure 5.12 we can see how to use the same data from the csv for the POST requests. In the figure 5.13 we can see the configuration of the test for K6, where we can see that this test simulates a ramp-up to 50 virtual users over 10 seconds, maintains this load for 90 seconds, and then ramps down to 0 users over 10 seconds. Additionally, a performance threshold is defined to ensure that 99% of HTTP request durations are below 3 seconds.



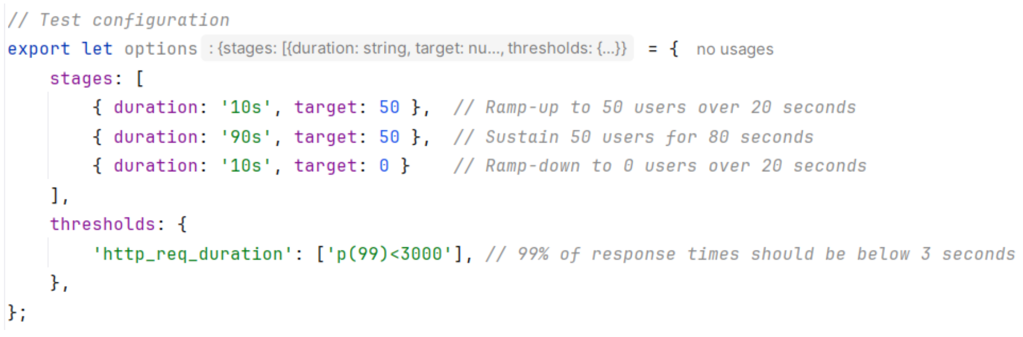
```
// Load data from CSV
const csvData = new SharedArray("Pet Types", function() {
    return papaparse.parse(open( url: 'C:/Users/user/Desktop/Testes/Pet_Types.csv'), { header: true }).data;
});
```

Figure 5.11: Load Data from CSV on K6



```
const petTypeName :any | string  = (pettype && pettype.petTypeName && pettype.petTypeName.trim() !== "")
? pettype.petTypeName.trim()
: "teste";  // Valor padrão para petTypeName
```

Figure 5.12: Use Data from CSV on K6

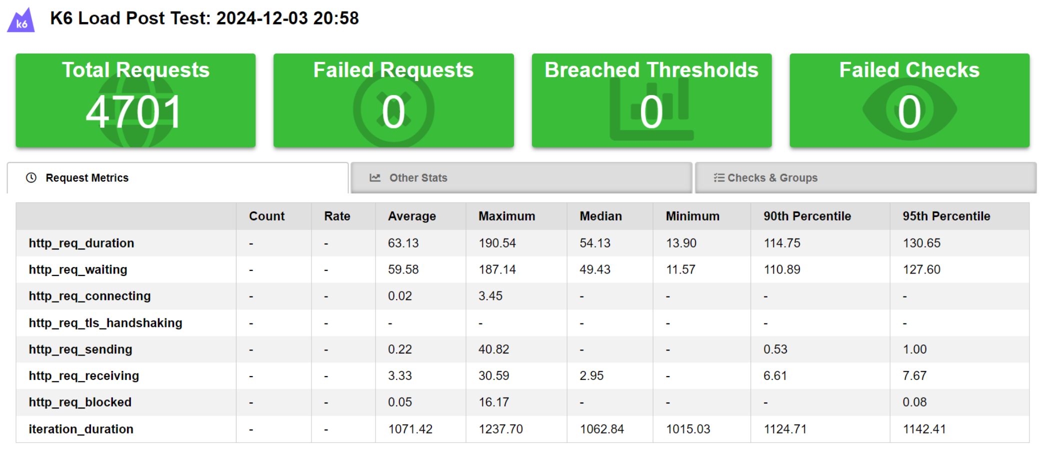


```
// Test configuration
export let options :{stages: [{duration: string, target: nu..., thresholds: {...}}  = {  no usages
    stages: [
        { duration: '10s', target: 50 },  // Ramp-up to 50 users over 20 seconds
        { duration: '90s', target: 50 },  // Sustain 50 users for 80 seconds
        { duration: '10s', target: 0 }    // Ramp-down to 0 users over 20 seconds
    ],
    thresholds: {
        'http_req_duration': ['p(99)<3000'], // 99% of response times should be below 3 seconds
    },
};
```

Figure 5.13: Load Configuration on K6

Results

In the figure 5.14 we can see the results of the k6 tests


K6 Load Post Test: 2024-12-03 20:58

Total Requests	Failed Requests	Breached Thresholds	Failed Checks
4701	0	0	0

	Count	Rate	Average	Maximum	Median	Minimum	90th Percentile	95th Percentile
http_req_duration	-	-	63.13	190.54	54.13	13.90	114.75	130.65
http_req_waiting	-	-	59.58	187.14	49.43	11.57	110.89	127.60
http_req_connecting	-	-	0.02	3.45	-	-	-	-
http_req_tls_handshaking	-	-	-	-	-	-	-	-
http_req_sending	-	-	0.22	40.82	-	-	0.53	1.00
http_req_receiving	-	-	3.33	30.59	2.95	-	6.61	7.67
http_req_blocked	-	-	0.05	16.17	-	-	-	0.08
iteration_duration	-	-	1071.42	1237.70	1062.84	1015.03	1124.71	1142.41

Figure 5.14: Load Results K6

Based on these results, we can confidently state that the application performed with exceptional efficiency, handling a total of 4,701 requests with no errors, failed requests, or breached thresholds. The average response time was an impressive 61.15 milliseconds, and even at the 95th percentile, the response time remained under 121 milliseconds. The maximum response time recorded was just 130.56 milliseconds, showcasing the application's ability to maintain consistently low latency throughout the test.

5.2.4 Soak Test

Soak testing involves running a system at normal load levels for an extended period to identify performance issues that may not be apparent during shorter testing phases. This type of testing helps uncover problems such as memory leaks, resource depletion, and degradation in response times over time. By maintaining a steady load, soak tests provide insights into the system's long-term reliability and stability, ensuring it can handle sustained usage without performance deterioration. In this section, we will evaluate the soak tests for the endpoint GET of petType.

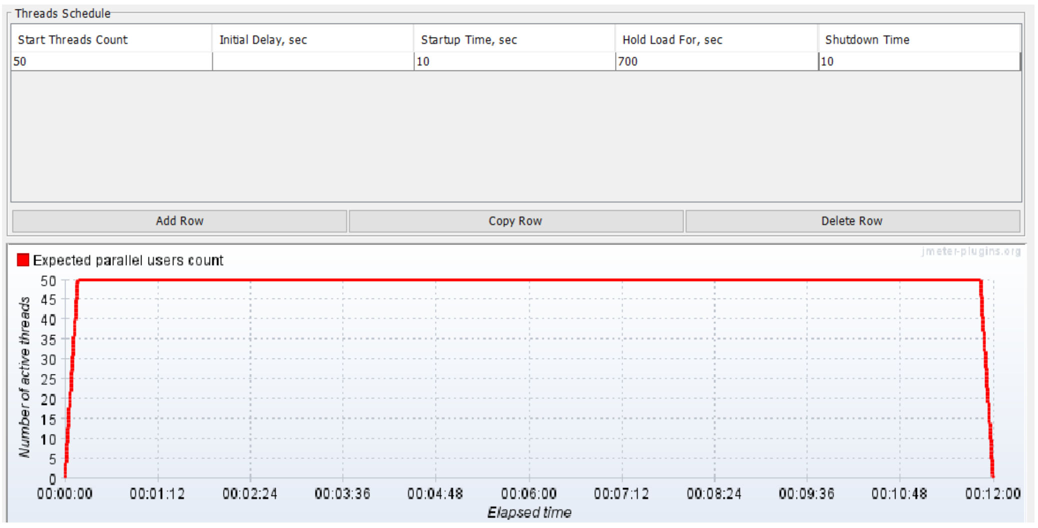
Scenario GET

- 50 users of the web application
- Requesting all PetTypes
- The system returns a list with all PetTypes
- Normal Conditions

Jmeter

Test Configuration

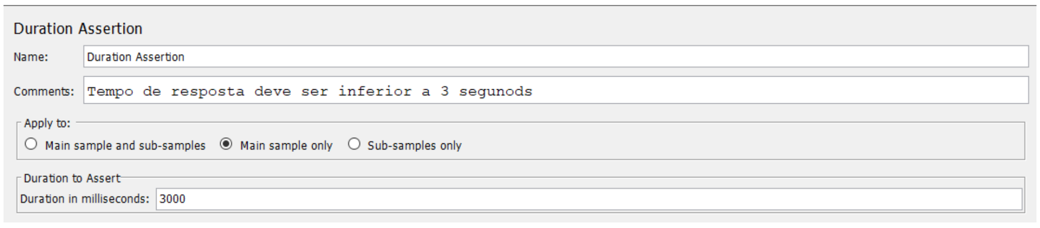
In the figure 5.15 we can see the configuration on jmeter for the test where we can see that the test simulates a ramp-up to 50 virtual users over 10 seconds, maintains this load for 700 seconds, and then ramps down to 0 users over 10 seconds. In the figure 5.16 we can see the configuration for the duration for each request in jmeter of 3 seconds.



Threads Schedule

Start Threads Count	Initial Delay, sec	Startup Time, sec	Hold Load For, sec	Shutdown Time
50		10	700	10

Figure 5.15: Soak Test Configuration on jMeter



Duration Assertion

Name: Duration Assertion

Comments: Tempo de resposta deve ser inferior a 3 segunods

Apply to: ○ Main sample and sub-samples [✓ Main sample only] ○ Sub-samples only

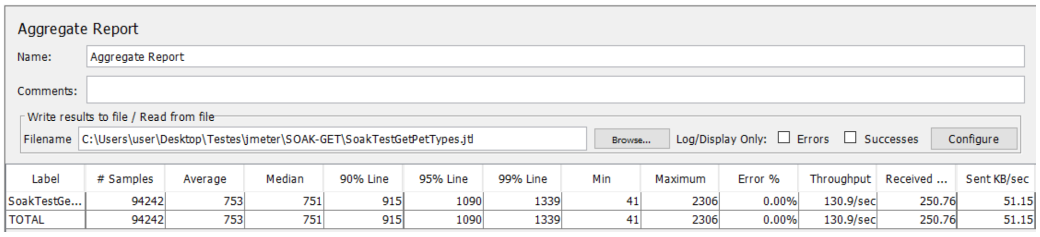
Duration to Assert

Duration in milliseconds: 3000

Figure 5.16: Soak Duration Configuration on jMeter

Results

In the figure 5.17 we can see the Aggregate Report of the jmeter test. In the figure 5.18 we can see the response time over time of the jmeter test.



Aggregate Report

Name: Aggregate Report

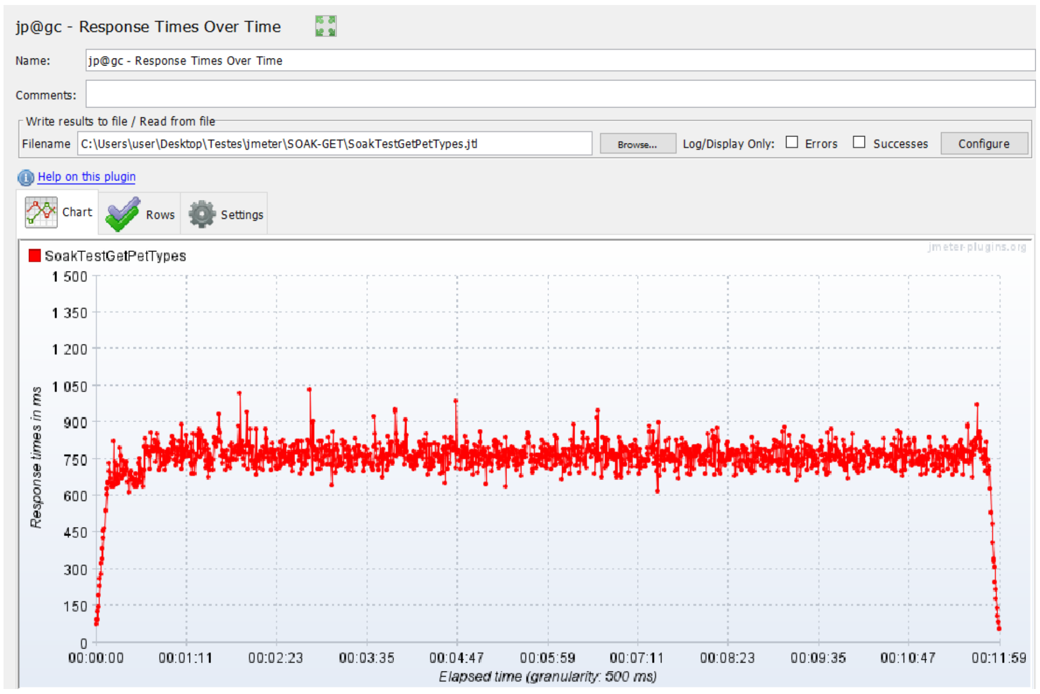
Comments:

Write results to file / Read from file

Filename C:\Users\user\Desktop\Testes\jmeter\SOAK-GET\SoakTestGetPetTypes.jtl Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
SoakTestGe...	94242	753	751	915	1090	1339	41	2306	0.00%	130.9/sec	250.76	51.15
TOTAL	94242	753	751	915	1090	1339	41	2306	0.00%	130.9/sec	250.76	51.15

Figure 5.17: Soak Aggregate Report



jp@gc - Response Times Over Time

Name: jp@gc - Response Times Over Time

Comments:

Write results to file / Read from file

Filename C:\Users\user\Desktop\Testes\jmeter\SOAK-GET\SoakTestGetPetTypes.jtl Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

Figure 5.18: Soak Response Time

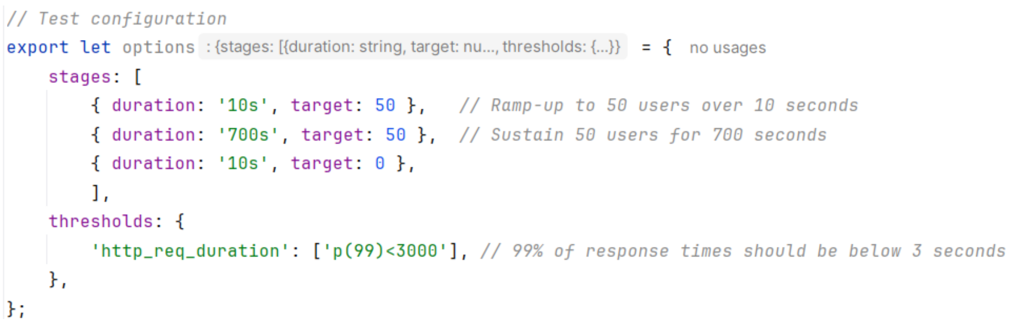
From the results of the soak test, we can confirm that the application maintained stable performance over an extended period. A total of 94242 samples were processed, with a maximum response time of 2,306 second. The average response time was 753 milliseconds, indicating reliable performance under continuous load.

The response time graph shows steady behavior, with minor fluctuations that remained within an acceptable range throughout the test duration. This stability demonstrates the application's ability to handle sustained traffic without any signs of performance degradation.

K6

Test Configuration

In the figure 5.19 we can see the configuration of the test for K6, where we can see that the test simulates a ramp-up to 50 virtual users over 10 seconds, maintains this load for 700 seconds, and then ramps down to 0 users over 10 seconds. Additionally, a performance threshold is set to ensure that 99% of HTTP request durations are below 3 seconds

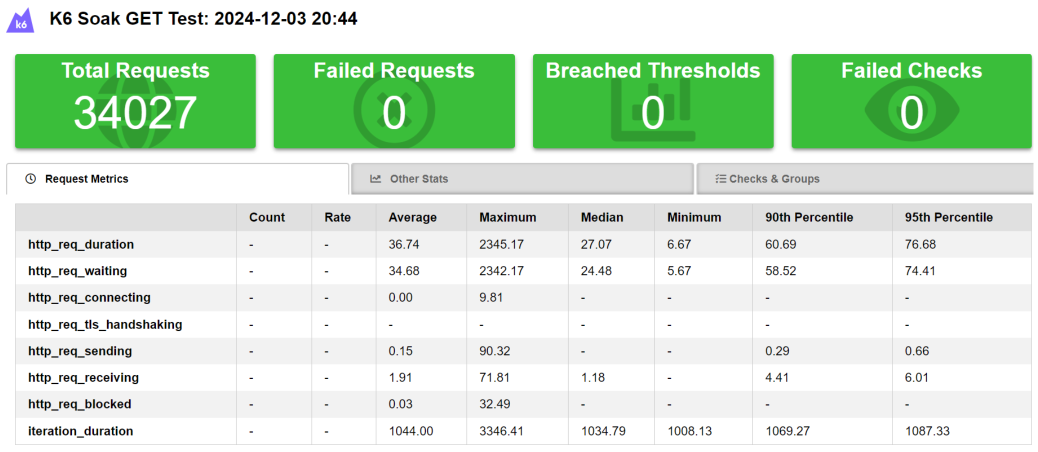


```
// Test configuration
export let options :{stages: [{duration: string, target: nu..., thresholds: {...}}  = {   no usages
    stages: [
        { duration: '10s', target: 50 },   // Ramp-up to 50 users over 10 seconds
        { duration: '700s', target: 50 },  // Sustain 50 users for 700 seconds
        { duration: '10s', target: 0 },
        ],
    thresholds: {
        'http_req_duration': ['p(99)<3000'], // 99% of response times should be below 3 seconds
    },
};
```

Figure 5.19: Soak Configuration on K6

Results

In the figure 5.20 we can see the results of the k6 test.



K6 Soak GET Test: 2024-12-03 20:44

Total Requests 34027 | Failed Requests 0 | Breached Thresholds 0 | Failed Checks 0

Request Metrics | Other Stats | Checks & Groups

	Count	Rate	Average	Maximum	Median	Minimum	90th Percentile	95th Percentile
http_req_duration	-	-	36.74	2345.17	27.07	6.67	60.69	76.68
http_req_waiting	-	-	34.68	2342.17	24.48	5.67	58.52	74.41
http_req_connecting	-	-	0.00	9.81	-	-	-	-
http_req_tls_handshaking	-	-	-	-	-	-	-	-
http_req_sending	-	-	0.15	90.32	-	-	0.29	0.66
http_req_receiving	-	-	1.91	71.81	1.18	-	4.41	6.01
http_req_blocked	-	-	0.03	32.49	-	-	-	-
iteration_duration	-	-	1044.00	3346.41	1034.79	1008.13	1069.27	1087.33

Figure 5.20: Soak Results K6

Based on the results of the test, the application demonstrated excellent performance and stability, successfully handling a total of 34,027 requests with zero failed requests, breached thresholds, or failed checks. This indicates that the application was able to process all incoming traffic efficiently without any errors or performance issues.

The iteration duration metrics further highlight the application's consistency, with a median response time of 1,034 milliseconds and a maximum of 3,346 milliseconds. These results show that while the majority of requests were handled quickly, even the longest processing times remained within acceptable limits.

5.2.5 Stress Test

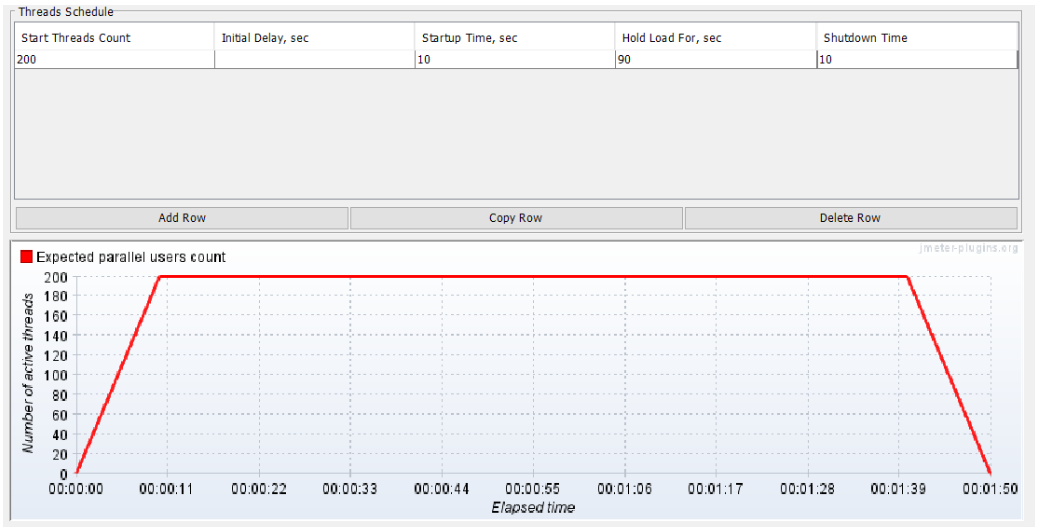
Stress testing is designed to push a system beyond its normal operational capacity to evaluate its robustness and determine its breaking point. Unlike soak testing, which focuses on long-term stability under sustained load, stress testing subjects the system to extreme load levels, such as a high volume of requests or rapid increases in traffic, to identify vulnerabilities and performance bottlenecks under peak conditions. This type of testing helps uncover issues such as response delays, crashes, or failures in handling surges. In this section, I will conduct a stress test for the endpoint GET of petType **Scenario GET**

- 200 users of the web application
- Requesting all PetTypes
- The system returns a list with all PetTypes
- Heavy Conditions

Jmeter

Test Configuration

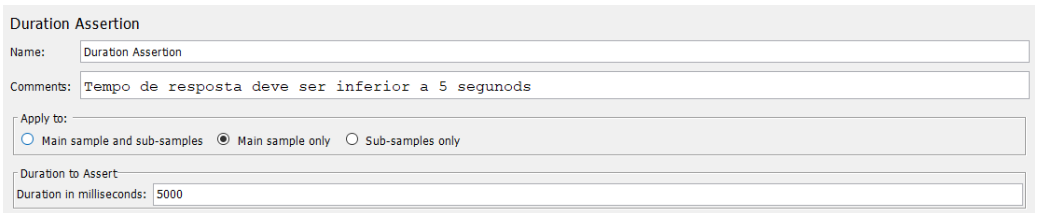
In the figure 5.21 we can see the configuration on jmeter for the test where we can see that the test simulates a ramp-up to 200 virtual users over 10 seconds, sustains 200 users for 90 seconds, and then ramps down to 0 users over 10 seconds. In the figure 5.22 we can see the configuration for the duration for each request in jmeter of 5 seconds.



Threads Schedule

Start Threads Count	Initial Delay, sec	Startup Time, sec	Hold Load For, sec	Shutdown Time
200		10	90	10

Figure 5.21: Stress Test Configuration on jMeter



Duration Assertion

Name: Duration Assertion

Comments: Tempo de resposta deve ser inferior a 5 segunods

Apply to: ○ Main sample and sub-samples [✓ ● Main sample only] ○ Sub-samples only

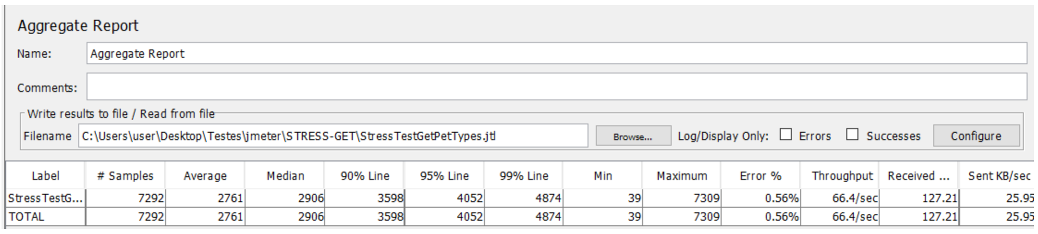
Duration to Assert

Duration in milliseconds: 5000

Figure 5.22: Stress Duration Configuration on jMeter

Results

In the figure 5.23 we can see the Aggregate Report of the jmeter test. In the figure 5.18 we can see the response time over time of the jmeter test.


Aggregate Report

Name: Aggregate Report

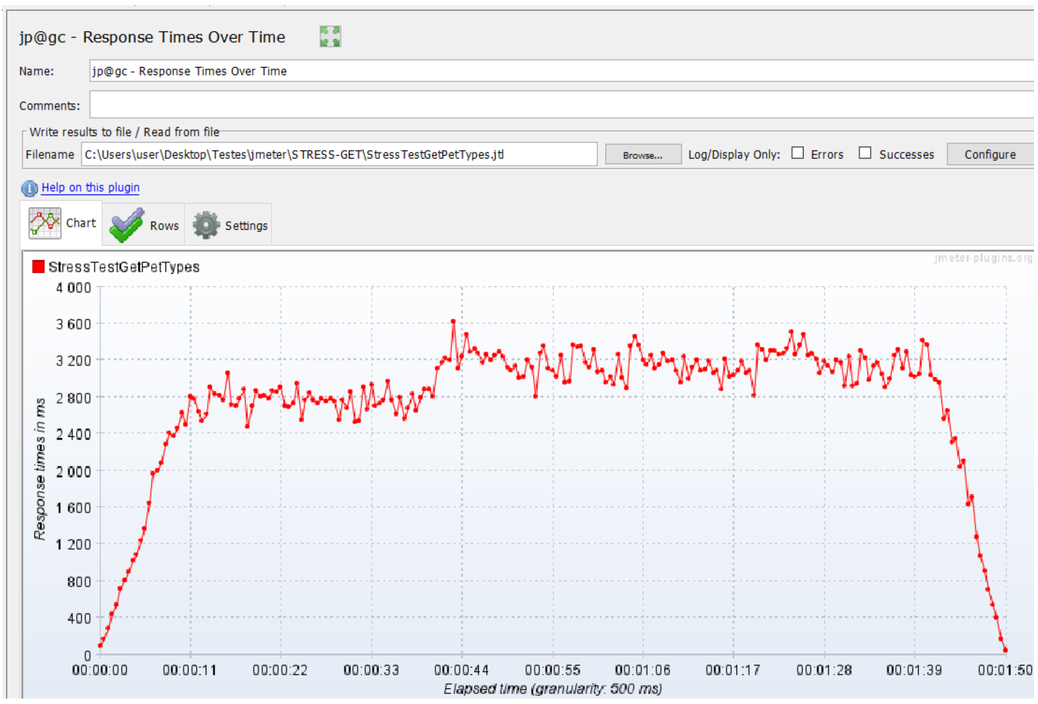
Comments:

Write results to file / Read from file

Filename C:\Users\user\Desktop\Testes\jmeter\STRESS-GET\StressTestGetPetTypes.jtl Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
StressTestG...	7292	2761	2906	3598	4052	4874	39	7309	0.56%	66.4/sec	127.21	25.95
TOTAL	7292	2761	2906	3598	4052	4874	39	7309	0.56%	66.4/sec	127.21	25.95

Figure 5.23: Stress Aggregate Report


jp@gc - Response Times Over Time

Name: jp@gc - Response Times Over Time

Comments:

Write results to file / Read from file

Filename C:\Users\user\Desktop\Testes\jmeter\STRESS-GET\StressTestGetPetTypes.jtl Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

Figure 5.24: Stress Response Time

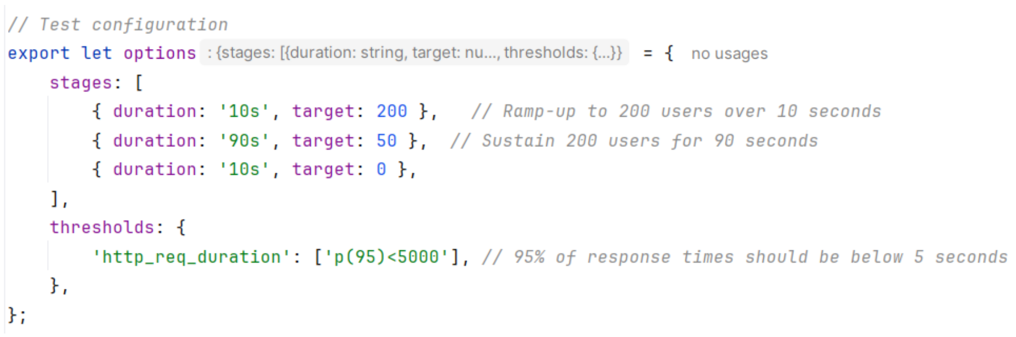
From the results of the stress test, we can confirm that the application maintained stable performance with a heavy load. A total of 7292 samples were processed, with a maximum response time of 7,3 seconds. The average response time was 2761 milliseconds, indicating reliable performance under heavy load.

The response time graph shows steady behavior, with a minor fluctuations around 44 seconds, but remained within an acceptable range throughout the test duration. This stability demonstrates the application's ability to handle sustained traffic without any signs of performance degradation.

K6

Test Configuration

In the figure 5.25 we can see the configuration of the test for K6, where we can see that the test simulates a ramp-up to 200 virtual users over 10 seconds, sustains 200 users for 90 seconds, and then ramps down to 0 users over 10 seconds. Additionally, a performance threshold is defined to ensure that 95% of HTTP request durations are below 5 seconds.

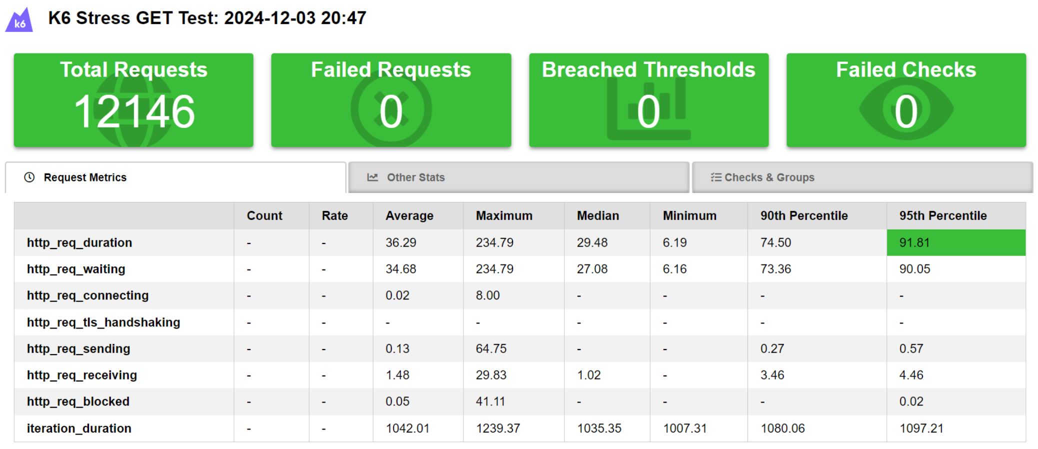


```
// Test configuration
export let options : {stages: [{duration: string, target: nu..., thresholds: {...}}  = {  no usages
    stages: [
        { duration: '10s', target: 200 },   // Ramp-up to 200 users over 10 seconds
        { duration: '90s', target: 50 },  // Sustain 200 users for 90 seconds
        { duration: '10s', target: 0 },
    ],
    thresholds: {
        'http_req_duration': ['p(95)<5000'], // 95% of response times should be below 5 seconds
    },
};
```

Figure 5.25: Stress Configuration on K6

Results

In the figure 5.26 we can see the results of the k6 test.



K6 Stress GET Test: 2024-12-03 20:47

Total Requests 12146 | Failed Requests 0 | Breached Thresholds 0 | Failed Checks 0

	Count	Rate	Average	Maximum	Median	Minimum	90th Percentile	95th Percentile
http_req_duration	-	-	36.29	234.79	29.48	6.19	74.50	91.81
http_req_waiting	-	-	34.68	234.79	27.08	6.16	73.36	90.05
http_req_connecting	-	-	0.02	8.00	-	-	-	-
http_req_tls_handshaking	-	-	-	-	-	-	-	-
http_req_sending	-	-	0.13	64.75	-	-	0.27	0.57
http_req_receiving	-	-	1.48	29.83	1.02	-	3.46	4.46
http_req_blocked	-	-	0.05	41.11	-	-	-	0.02
iteration_duration	-	-	1042.01	1239.37	1035.35	1007.31	1080.06	1097.21

Figure 5.26: Stress Results K6

Based on the results of the test, the application demonstrated excellent performance and stability, successfully handling a total of 12,146 requests with zero failed requests, breached thresholds, or failed checks. This indicates that the application was able to process all incoming traffic efficiently without any errors or performance issues.

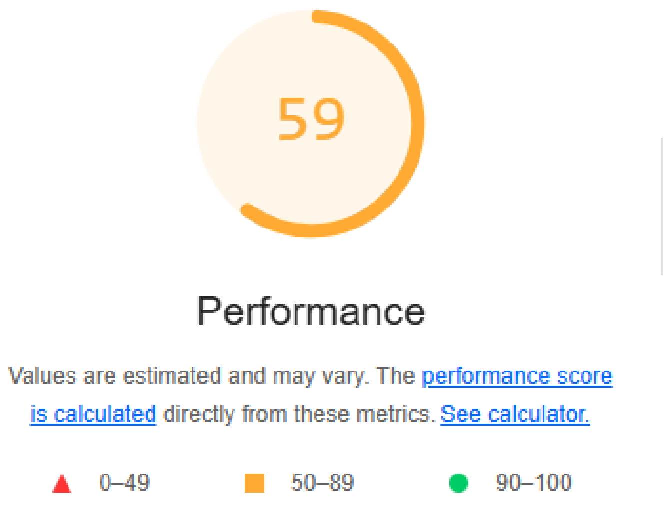
The iteration duration metrics further highlight the application's consistency, with a median response time of 1,042 milliseconds and a maximum of 1,239 milliseconds. These results show that the requests were handled quickly, even the longest processing times remained within acceptable limits.

5.3 Frontend Performance

In this chapter, I focus on evaluating the frontend performance of the application, adopting a user-centric approach to ensure an optimal experience. Guided by the RAIL model Response, Animation, Idle, and Load—and aligned with Core Web Vitals, the analysis aims to highlight key metrics that impact user experience. By examining these aspects, I aim to identify strengths and areas for improvement, ensuring the frontend is well-suited to meet modern performance standards.

5.3.1 LightHouse Report

The frontend performance received an overall score of 59/100, as seen in the figure 5.27 indicating room for improvement in achieving a smoother and faster user experience.



59

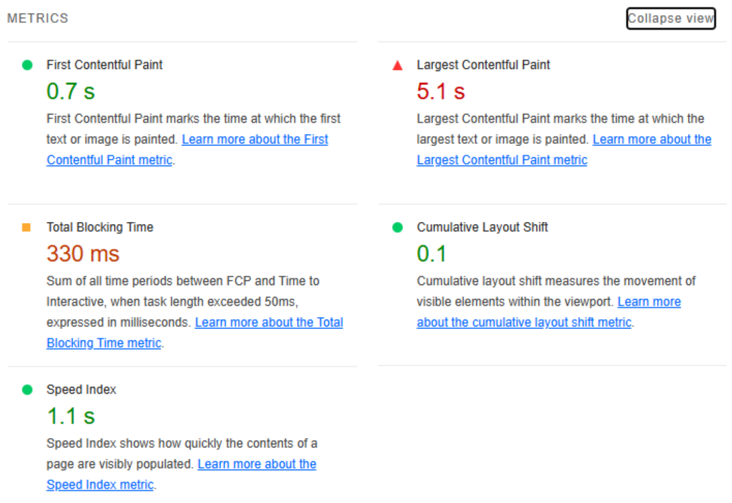
Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

Figure 5.27: LightHouse performance Report

In the figure 5.28 we have a breakdown of each metric evaluate, and on the figure 5.29 we have the impact of each of those metrics for the final result.



METRICS

● First Contentful Paint
0.7 s
First Contentful Paint marks the time at which the first text or image is painted. Learn more about the First Contentful Paint metric.

▲ Largest Contentful Paint
5.1 s
Largest Contentful Paint marks the time at which the largest text or image is painted. Learn more about the Largest Contentful Paint metric

■ Total Blocking Time
330 ms
Sum of all time periods between FCP and Time to Interactive, when task length exceeded 50ms, expressed in milliseconds. Learn more about the Total Blocking Time metric.

● Cumulative Layout Shift
0.1
Cumulative layout shift measures the movement of visible elements within the viewport. Learn more about the cumulative layout shift metric.

● Speed Index
1.1 s
Speed Index shows how quickly the contents of a page are visibly populated. Learn more about the Speed Index metric.

Figure 5.28: Lighthouse Metrics Report

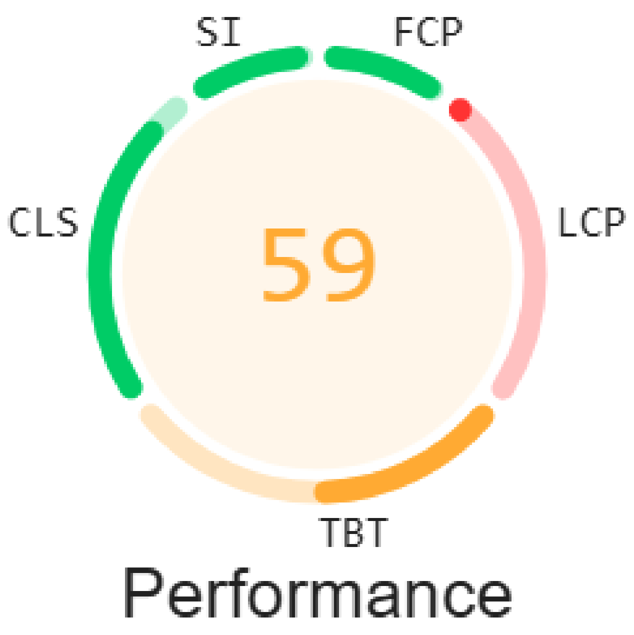
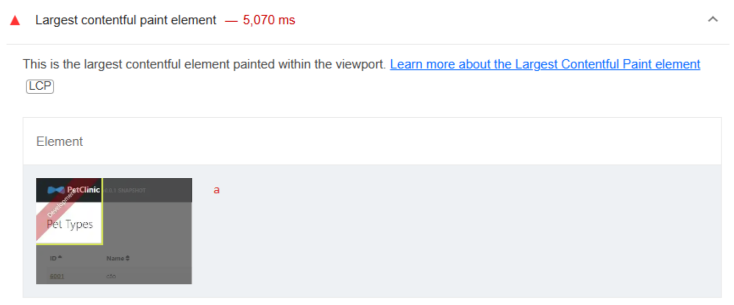


Figure 5.29: Lighthouse Impact Report

Below is a breakdown of the key metrics evaluated by Lighthouse:

- **First Contentful Paint (FCP):** At 0.7 seconds (green), this metric is excellent, as it measures the time it takes for the first piece of content to appear on the screen. A fast FCP helps create a good initial impression for users.
- **Largest Contentful Paint (LCP):** At 5.1 seconds (red), this metric is too slow, measuring the time to render the largest visible element. It directly impacts how quickly users feel the page is fully loaded, and optimization is needed.
- **Total Blocking Time (TBT):** At 330 milliseconds (yellow), this metric measures delays caused by tasks that block interactivity. While acceptable, it can be improved by reducing JavaScript execution or deferring non-critical scripts.
- **Cumulative Layout Shift (CLS):** At 0.1 (green), this measures layout stability, ensuring that content doesn't unexpectedly shift during loading. This result is excellent.
- **Speed Index:** At 1.1 seconds (green), this measures how quickly visible content is rendered. The score is outstanding, indicating fast rendering and a smooth visual experience.

By analyzing the diagnostics, on the figure 5.30 I found that one of the reasons for such poor LCP, was due to the Development Flag. This was also an issue on the accessibility. Removing this flag would most likely improve the Frontend Performance.



▲ Largest contentful paint element — 5,070 ms

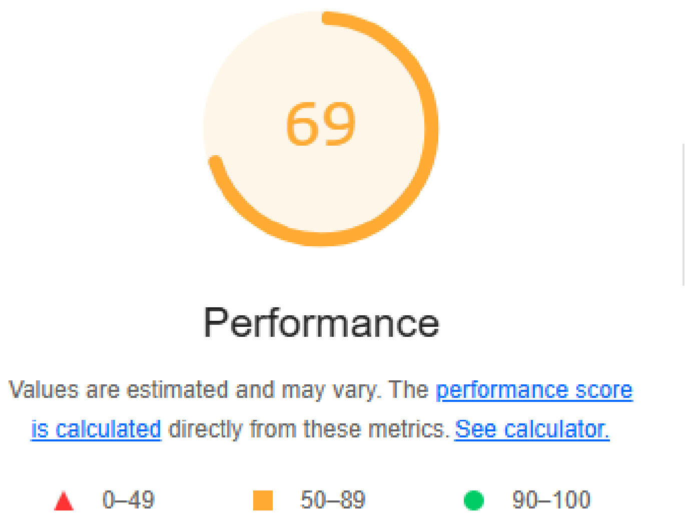
This is the largest contentful element painted within the viewport. Learn more about the Largest Contentful Paint element LCP

Element

a

Figure 5.30: LCP Element causing the problem

By removing the banner the results had a significant improvement. The application score a 69 out of 100.



69

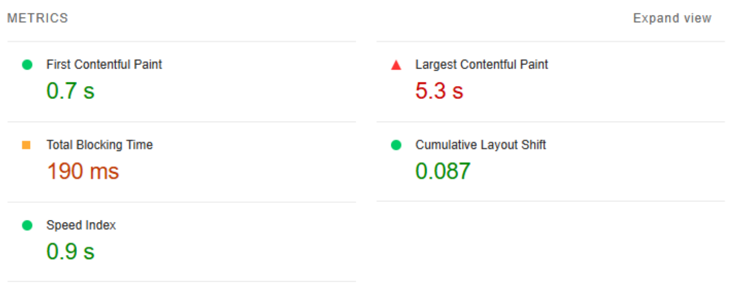
Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

Figure 5.31: LightHouse performance Report after changes

In the figure 5.32 we have a breakdown of each metric evaluate, and on the figure 5.33 we have the impact of each of those metrics for the final result.


METRICS

Expand view

● First Contentful Paint
0.7 s

▲ Largest Contentful Paint
5.3 s

■ Total Blocking Time
190 ms

● Cumulative Layout Shift
0.087

● Speed Index
0.9 s

Figure 5.32: Lighthouse Metrics Report after changes

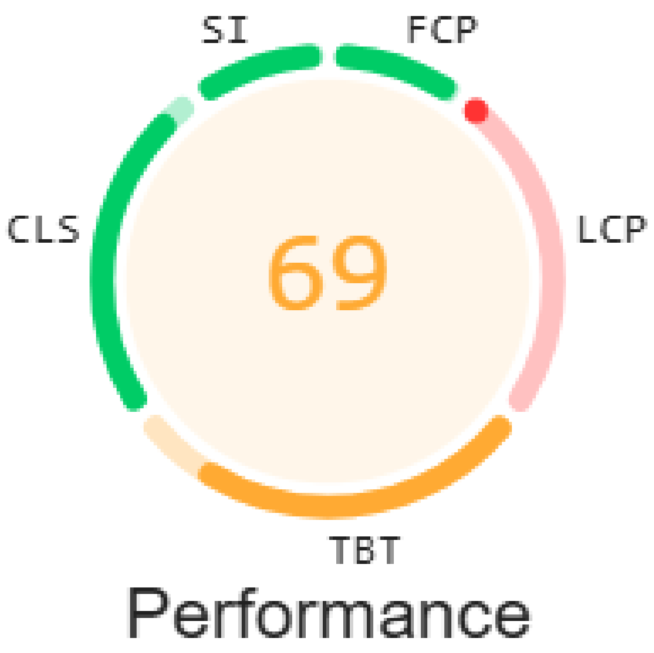


Figure 5.33: Lighthouse Impact Report after changes

Although the changes were primarily intended to improve the LCP, the results showed a different outcome. Three metrics (TBT, CLS, and Speed Index) performed better; however, the metric we aimed to improve remained unchanged.

So, even though the result was not the intended one, I observed improvements in other areas, which suggests that the changes made had a positive overall impact.

5.4 Conclusion

In conclusion, the performance analysis revealed a mixed outcome. The frontend performance, as measured by Lighthouse, scored 59/100, reflecting significant issues with page responsiveness and loading times, particularly due to a poor Largest Contentful Paint (LCP) and Total Blocking Time (TBT). These metrics indicate the need for substantial optimization to improve the user experience.

The backend performance, however, demonstrated strong stability and reliability in most tests. Across various scenarios—load, stress, and soak tests—the system consistently processed requests without errors or failed checks.

Overall, while the backend shows promise, with reliable request handling and strong stability under heavy loads, response times could be further optimized. The frontend, on the other hand, requires considerable attention to meet modern performance standards. However, after the changes we saw the performance have a boost in its ratings. Combining both aspects, the performance for this iteration is rated 4 out of 5, acknowledging the solid backend performance but noting significant areas for improvement in the frontend.

6 Security

[Description, for what under individual responsibility, that inform the goal, the questions related to security, metrics used and their values, with partial response to questions, and goal achievement analysis. Explicitly mention the tool report(s).]

7 Conclusions

The individual report centered on analyzing a JHipster-generated application, focusing on key aspects such as accessibility, visual compatibility, maintainability, and performance. Both the frontend and backend were thoroughly tested using various tools, with each test and its methodology detailed in the respective chapters.

Through analysis and adjustments, it can confidently be stated that the application is in a solid state. There are several positive aspects highlighted across the tested topics, both of the frontend and backend, demonstrating its potential as a reliable foundation. While some areas require improvement, these shortcomings do not overshadow the overall quality and readiness of the application. It serves as a strong starting point for future development and enhancement, providing a robust base upon which more complex features and refinements can be built. ...

References