TESTING REPORT

Imagen en blanco y negro

Descripción generada automáticamente con confianza media

DELIVERABLE 4

DESING AND TESTING 2

2023-2024

|  |  |
| --- | --- |
| Date | Version |
| 07/07/2024 | V1.0 |

|  |  |
| --- | --- |
| Group: 21 | |
| Members | Email |
| Fernández Rodríguez, Jesús | [jesferrod1@alum.us.es](mailto:jesferrod1@alum.us.es) |
| García Rodríguez, Javier | [javgarrod5@alum.us.es](mailto:javgarrod5@alum.us.es) |
| González Ortiz, Miguel | [miggonort1@alum.us.es](mailto:miggonort1@alum.us.es) |
| Palomo García, Miguel | [migpalgar1@alum.us.es](mailto:migpalgar1@alum.us.es) |
| Periáñez Franco, Luis Javier | [luiperfra1@alum.us.es](mailto:luiperfra1@alum.us.es) |

GitHub repository: <https://github.com/JesusFern/Acme-SF-D04>

TABLE OF CONTENTS

[Executive Summary 3](#_Toc171247712)

[Revision Table 3](#_Toc171247713)

[Introduction 3](#_Toc171247714)

[Contents 3](#_Toc171247715)

[Functional Tests 3](#_Toc171247716)

[Banner 4](#_Toc171247717)

[Performance Testing 4](#_Toc171247718)

[Conclusion 6](#_Toc171247719)

[Bibliography 7](#_Toc171247720)

# Executive Summary

This report provides detailed information obtained through the execution of functional and performance tests for deliverable D04 of the project. In this way, we can gain a thorough understanding of the methodology to be followed for conducting these tests and the conclusions we can draw from them.

# Revision Table

|  |  |  |
| --- | --- | --- |
| Revision Number | Date | Description |
| v1r0 | 07/07/2024 | - |
|  |  |  |
|  |  |  |

# Introduction

The report consists of two main sections. The first section focuses on functional testing, verifying that the system's functionalities meet the specified requirements. The second section concentrates on performance testing, ensuring that the system operates within the established performance parameters.

# Contents

## Functional Tests

The development of these tests was carried out following the methodology proposed in the course slides. The highest possible coverage was achieved by discarding cases where our natural intelligence indicated that attempting to cover certain code instructions would be pointless.

Throughout the testing, 100% coverage has never been achieved because the lines “assert object != null;” can never encounter a null object. However, it is good practice to keep these lines of code as recommended.

## Banner

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* Show: The tests for this functionality are quite simple. The process followed logging in as an administrator and accessing an already created banner, and for the hacking we tried to show a banner with a not logged in user. We achieved a coverage of 95.5%.
* Create: The test for this functionality are more complex, for each attribute we had to check for limits and all possible combinations in the creation form were covered to ensure that our validation methods were indeed correct. For the create hacking test we tried to create a banner with a not logged in user. We achieved a coverage of 92.9%.
* Update: For this test we had to create a new object and check all the restrictions already proven on the create test, all possible combinations in the creation form were covered to ensure that our validation methods were indeed correct. For the update hacking test we tried to update a banner with a not logged in user. We achieved a coverage of 93.6%.
* Delete: The delete test are quite simple, trying to delete a created banner and for the hacking test we tried to delete a banner with a not logged in user. We achieved a coverage of 89,0%.
* List: This was a simple test, trying to list banners logged in as administrator and for the hacking we tried to list them with a not logged in user.

## Performance Testing

During the performance analysis, the performance before and after the indexes were implemented was analyzed.

* Analyzing performance

Average of the times obtained after tester#replayer launcher execution:

Gráfico, Gráfico de barras

Descripción generada automáticamente

As can be seen, the grand average is 14,93 milliseconds. It can also be seen that the methods with the highest average milliseconds are administrator/banner/update and are administrator/banner/delete.

Aplicación, Tabla, Excel

Descripción generada automáticamente

This is the result of the data analysis, the confidence interval in milliseconds is [ 13.75083893 ms , 17.04319654 ms ] this meets our 1-second sample requirement comfortably.

Due to the fact that in the queries we used only the id attribute, the framework creates indexes for them automatically, no additional indexes have been added in the testing of this functionality.

* Profiling software

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

This was the result obtained after profiling software. As can be seen the execution time of bind methods of the create and update services are higher than the rest of the methods. However, its self time is 0 so it’s not that method that consumes too much time, but the method that it invokes.

* Profiling hardware

Gráfico, Gráfico de líneas

Descripción generada automáticamente

This was the result obtained after profiling hardware.

The memory of this computer is being moderately used, but it is not a clear bottleneck, the CPU and the network are little used so they are far from a bottleneck as well as the other components.

# Conclusion

In summary, we have been able to evaluate our code rigorously and understand its functioning in detail, identifying possible dead code and bugs. Additionally, thanks to the performance tests, we have been able to identify which functionalities we may need to improve if a response time requirement is presented by the client.

# Bibliography

Intentionally blank