TESTING REPORT

Imagen en blanco y negro

Descripción generada automáticamente con confianza media

DELIVERABLE 4

DESING AND TESTING 2

2023-2024

MIGUEL PALOMO GARCÍA

|  |  |
| --- | --- |
| Date | Version |
| 06/07/2024 | 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](#_Toc167735018)

[Revision Table 3](#_Toc167735019)

[Introduction 3](#_Toc167735020)

[Contents 3](#_Toc167735021)

[Functional Tests 3](#_Toc167735022)

[Code Audit: 4](#_Toc167735023)

[Audit Record: 4](#_Toc167735024)

[Performance Testing 5](#_Toc167735025)

[Conclusion 7](#_Toc167735026)

[Bibliography 7](#_Toc167735027)

# 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 | 06/07/2024 | - |
|  |  |  |
|  |  |  |

# 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.

The only setback during development was the appearance of FAILED messages in the console due to the banner and its changing ID, but this did not pose any impediment to the correct execution of the tests. Below are the details of how each functionality was tested, as well as the proposed hacking attempts.

## Code Audit:

* Show: The tests for this functionality are quite simple. The process followed involved registering as an auditor and accessing each already created code audit. We achieved a coverage of 97.8%.
* 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. We achieved a coverage of 92.2%.
* 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. We achieved a coverage of 92.4%.
* Publish: For the publish test we checked both validations, “Need at least one audit record” and “the mark should be above F” We achieved a coverage of 94.2%.
* Delete: The delete test are quite simple, trying to delete a code audit already published and another one just created. We achieved a coverage of 88.8%.

I’ve tried to complete 100% coverage but it was impossible because the validations needed a null value of user or null Code Audit

## Audit Record:

* Show: The tests for this functionality are quite simple. The process followed involved registering as an auditor and accessing each already created audit record. We achieved a coverage of 95.8%.
* 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. We achieved a coverage of 94.9%.
* 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. We achieved a coverage of 94.4%.
* Delete: The delete test are quite simple, trying to delete a code audit already published and another one just created. We achieved a coverage of 88%.

I’ve tried to complete 100% coverage but it was impossible because the validations needed a null value of user or null Audit Record

# 

## Performance Testing

These graphs show the evolution over time of the execution before adding indexes to the tables and after.

Gráfico, Gráfico de barras

Descripción generada automáticamente

Gráfico, Gráfico de barras

Descripción generada automáticamente

And here we can see the difference in measurements.

Tabla

Descripción generada automáticamente

7

Tabla

Descripción generada automáticamente

Since the critical z-value (two-tailed) is between 0.05 and 1, we can say that the data sample is not large enough for the comparison to be reliable.

# 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