Insertando imagen...

ACME-SF

G1.007

**Testing Report**

27/05/2024



# Cover

|  |  |
| --- | --- |
| **Repository:** <https://github.com/DP2-C1-07/Acme-SF-D04> | |
| **Student #1**  **ID:** 31878881F  **UVUS:**  pabcabmar3  **Name:** Caballero María, Pablo **Roles:** manager, developer | **Student #2**  **ID Number:**49034820Q  **UVUS:** mararnmon  **Name:** Arnáiz Montero, Marco Antonio  **Roles:** developer, operator, tester |
| **Student #3**  **ID Number:** 77865211E  **UVUS:**  alfalolan  **Name:** Alonso Lanzarán, Alfonso Luis  **Roles:** developer, tester | **Student #4**  **ID Number:** 53932912M  **UVUS:** albsanmim  **Name:** Sánchez Mimbrero, Alberto  **Roles:** developer |
| **Student #5**  **ID Number:** 48123111G  **UVUS:** juagarcar4  **Name:** Garcia Carballo, Juan  **Roles:** developer |  |

Table of contents

[Cover 2](#_Toc167696962)

[Executive summary 3](#_Toc167696963)

[Introduction 4](#_Toc167696964)

[Chapter on Functional Testing 4](#_Toc167696965)

[Listing of implemented test cases grouped by feature 4](#_Toc167696966)

[Description of each test case 5](#_Toc167696967)

[This section will focus on the “Contract” test cases as “Progress Log” is similar and less interesting for this report. 5](#_Toc167696968)

[Client Contract List 5](#_Toc167696969)

[Client Contract Show 5](#_Toc167696970)

[Client Contract Create 6](#_Toc167696971)

[Client Contract Update 6](#_Toc167696972)

[Client Contract Delete 7](#_Toc167696973)

[Client Contract Publish 7](#_Toc167696974)

[Effectiveness in detecting bugs 8](#_Toc167696975)

[Coverage 9](#_Toc167696976)

[Chapter on Performance Testing 10](#_Toc167696977)

[Methodology 10](#_Toc167696978)

[Performance Metrics 10](#_Toc167696979)

[Performance Before Adding Indexes 11](#_Toc167696980)

[Performance After Adding Indexes 12](#_Toc167696981)

[Analysis 13](#_Toc167696982)

[Statistical Analysis 13](#_Toc167696983)

[95%-confidence interval 14](#_Toc167696984)

[95%-confidence hypothesis contrast 14](#_Toc167696985)

[Chapter on Profiling 15](#_Toc167696986)

[Conclusions 15](#_Toc167696987)

[Bibliography 15](#_Toc167696988)

# Executive summary

This document is a Testing Report that includes the functional and performance testing results for the project “Acme-SF-D04”. The functional testing chapter details the test cases implemented, grouped by feature, and provides a description along with an assessment of their effectiveness in detecting bugs. The performance testing chapter includes charts and 95%-confidence intervals for the wall time taken to serve requests in functional tests on two different computers, along with a 95%-confidence hypothesis contrast regarding the most powerful computer.

Revision table

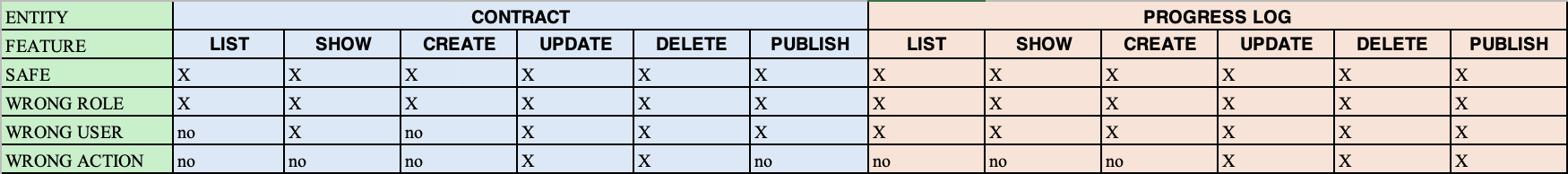
|  |  |  |
| --- | --- | --- |
| Number | Date(dd/mm/yyyy) | Description |
| 1.0 | 25/05/2024 | Creation of the document and redaction of the executive summary and introduction. |
| 1.1 | 26/05/2024 | Redaction of the chapter on functional testing. |
| 1.2 | 27/05/2024 | Redaction of the chapter on performance testing, chapter on profiling and conclusion. |

# Introduction

The purpose of this document is to provide a comprehensive testing report for the project “Acme-SF-D04”, for the subject “Design and Testing 2”. This report covers both functional and performance testing, providing detailed insights into the effectiveness and efficiency of the project.

# Chapter on Functional Testing

## Listing of implemented test cases grouped by feature



The following scheme has been followed to test each feature of every entity. There are two main entities tested: "Contract" and "Progress Log". For each entity, a ".safe" file has been generated containing the test traces for both positive and negative cases. Additionally, for each entity, up to three ".hack" files can be found:

Additionally, for each entity, up to 3 ".hack" files can be found:

* Wrong role: Incorrect role permissions for the action.
* Wrong user: Correct role assigned but the action is performed by a user on another user's entity.
* Wrong action: Correct role assigned, logged-in with the correct user but the action is performed to a published entity.

For some features, certain tests may not be applicable, indicated by a "no" in the table.

Below is an image displaying all the files that have been created:

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente con confianza media

## Description of each test case

## This section will focus on the “Contract” test cases as “Progress Log” is similar and less interesting for this report.

### Client Contract List

* Safe:

To list the contracts, the "tester-recorder" has been executed, recording a trace showing contracts from various users with the client role. No negative test case could be verified as only GET requests need to be made

* Wrong Role:

Attempting this type of hacking involved copying the URL to list client contracts, then accessing the system with an auditor user and pasting the URL into the browser, resulting in a 500 not authorized error.

* Wrong User:

This type of hacking is nonsensical because if you're a user with a client role and you copy the URL to list contracts, you'll see your own contracts by the definition of the feature.

* Wrong Action:

This type of hacking is nonsensical as you're only executing a GET request.

### Client Contract Show

* Safe:

To display the details of a contract, a GET request was made to retrieve the information, showing the contract's details accurately. The test was successful in demonstrating the functionality of displaying contract details.

* Wrong Role:

Attempting this type of hacking involved copying the URL to view a contract, then accessing the system with a user of a different role, such as an auditor, and pasting the URL into the browser. This resulted in a 500 not authorized error, indicating that users with incorrect roles cannot view contract details.

* Wrong User:

In this scenario, an unauthorized user could copy the URL to view a contract and attempt to access it. However, since the system is designed to only display contracts belonging to the logged-in user, this type of hacking is ineffective. The user will only see their own contracts, maintaining data privacy and security.

* Wrong Action:

Similar to the Safe scenario, this type of hacking is ineffective as it only involves executing a GET request to view contract details.

### Client Contract Create

* Safe:

To create a contract, a POST request was made for each test case, both positive and negative, as discussed in class. The provided Sample-Data file was utilized to generate test cases. Among the most interesting test cases in this entity is the Money type, where it's essential to verify that the value is not negative, less than 1 million, and a supported currency by the system.

* Wrong Role:

Attempting this type of hacking involves an unauthorized user with a different role, such as an auditor, trying to create a contract. However, since the user lacks the necessary client role, they cannot perform contract creations. Therefore, this type of hacking is irrelevant.

* Wrong User:

Since the scenario involves creating a contract, and there's no specific entity in the database linked to the user attempting the action, this type of hacking is not applicable. In the absence of a contract associated with the unauthorized user, attempting to create a contract is nonsensical.

* Wrong Action:

Similarly, since the action involves creating a contract, and there's no specific entity in the database linked to the unauthorized action, this type of hacking is not applicable. Without a contract entity to create, attempting to perform this action is meaningless.

### Client Contract Update

* Safe:

To update a contract, a PUT request was made for each test case, both positive and negative, as discussed in class. The provided Sample-Data file was utilized to generate test cases. A key aspect is ensuring the contract’s budget is less than the associated project's cost. If the contract is not linked to a project, the system will not allow the update, ensuring data consistency and integrity. Successful updates confirm the system's ability to handle contract modifications correctly.

* Wrong Role:

Attempting this type of hacking involved an unauthorized user with a different role, such as an auditor, trying to update a contract. A PUT request was made by this unauthorized user. The system responded with a 500 not authorized error, indicating that users without the client role cannot update contracts, thus maintaining the security and role-based access controls of the system.

* Wrong User:

In this scenario, an unauthorized user could copy the URL to update a contract belonging to another user and attempt to access it. However, the system is designed to only allow updates to contracts belonging to the logged-in user. Therefore, the unauthorized user will not be able to update the contract, ensuring data privacy and security by preventing such actions.

* Wrong Action:

To test wrong action hacking, an attempt was made to update a published contract. The system appropriately responded with an error, indicating that only unpublished contracts can be updated. This ensures that once a contract is published, it remains immutable, thereby maintaining the integrity and reliability of the published contracts within the system.

### Client Contract Delete

* Safe

To delete a contract, a DELETE request was made for each test case, both positive and negative, as discussed in class. The provided Sample-Data file was utilized to generate test cases. Successful deletion of a contract demonstrates the system's ability to handle contract deletions correctly. This test case was validated by confirming the contract no longer appeared in the list of client contracts.

* Wrong Role:

Attempting this type of hacking involved an unauthorized user with a different role, such as an auditor, trying to delete a contract. A DELETE request was made by this unauthorized user. The system responded with a 500 not authorized error, indicating that users without the client role cannot delete contracts, thus ensuring the integrity and security of the system.

* Wrong User:

In this scenario, an unauthorized user could copy the URL to delete a contract belonging to another user and attempt to access it. However, the system is designed to only allow deletion of contracts belonging to the logged-in user. Thus, the unauthorized user will not be able to delete the contract, and the system will maintain data privacy and security by preventing such actions.

* Wrong Action:

To test wrong action hacking, an attempt was made to send a DELETE request to a URL that does not support this action, such as trying to delete a published contract. The system appropriately responded with an error, demonstrating that such actions are not allowed. This ensures that only specific actions can be performed on contracts according to their state.

### Client Contract Publish

* Safe:

To publish a contract, a POST request was made to change the contract status from draft to published. This was tested using both positive and negative scenarios as discussed in class, with the provided Sample-Data file utilized to generate test cases. The successful transition of the contract status to published validates the functionality of the publish feature.

* Wrong Role:

Attempting this type of hacking involved an unauthorized user with a different role, such as an auditor, trying to publish a contract. A POST request was made by this unauthorized user. The system responded with a 500 not authorized error, indicating that only users with the client role can publish contracts, thereby preserving the system's security and role-based access controls.

* Wrong User:

In this scenario, an unauthorized user could copy the URL to publish a contract belonging to another user and attempt to access it. However, the system is designed to only allow publishing of contracts belonging to the logged-in user. Therefore, the unauthorized user will not be able to publish the contract, ensuring data privacy and security are maintained.

* Wrong Action:

To test wrong action hacking, an attempt was made to send a POST request to a contract that is already published, which should not be allowed. The system appropriately responded with an error, demonstrating that contracts cannot be published multiple times or altered once published. This ensures the integrity of the contract states within the system.

## Effectiveness in detecting bugs

During the execution of the tests, several bugs were detected in both the Contract and Progress Log entities.

1. Automatic Assignment of Dates:
   * Issue: The fields InstantationMoment (in a Contract) and ExecutionDate (in a Progress Log) were set to be automatically assigned by the database. This caused issues during functional testing, particularly during update or publish operations.
   * Example: When attempting to update or publish an entity, pressing the corresponding button once would cause the field to disappear and an error to be thrown because the field was null.
   * Solution: The fields had to be included in the dataset during the unbind process of the service to ensure they were correctly managed during updates.
2. Currency Validation for Budget:
   * Issue: The Budget field in the Contract entity was modified to accept only the currencies supported by the system.
   * Example: Initially, the system did not enforce this restriction, leading to potential issues with unsupported currencies being entered.
   * Solution: Validation was added to ensure that only accepted currencies could be used for the Budget field, preventing any unsupported currency from being processed.
3. Publishing of Progress Log:
   * Issue: When testing the publishing of a Progress Log, the log was successfully published, but the corresponding field did not become readonly as expected.
   * Example: After publishing, the Progress Log should have had certain fields set to readonly to prevent further modifications, but this did not occur, indicating a bug in the view of the entity.
   * Solution: The view logic was corrected to ensure that once a Progress Log is published, the relevant fields become readonly, maintaining data integrity and preventing unauthorized changes.

These findings underline the importance of thorough functional testing to identify and rectify issues related to database field management and data validation. The adjustments made because of these tests have improved the robustness and reliability of the system.

## Coverage

To complete the functional testing, we will analyze the coverage of our tests. Generally, the coverage is very good, but we can see that the coverage for the service to delete contracts and progress logs is low. This is due to the unbind method that has been implemented. Since it is a method that saves data in the database, it is quite useless because this action is meant to delete an entity from the database.Pantalla de computadora con números

Descripción generada automáticamente con confianza mediaUna captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media Interfaz de usuario gráfica

Descripción generada automáticamente

Imagen que contiene instrumento, lápiz

Descripción generada automáticamente

# Chapter on Performance Testing

Performance testing is a crucial aspect of software testing that evaluates the speed, responsiveness, and stability of a system under a specific workload. This chapter delves into the performance testing of the "Acme-SF-D04" project, focusing on comparing results obtained from the "tester-replayer" tool before and after adding indexes to the database entities.

## Methodology

Performance testing was conducted using the "tester-replayer" tool, which replays recorded test cases to measure the time taken to process requests. The performance tests were executed twice:

1. Before adding indexes to the entities.
2. After adding indexes to the entities.

The performance metrics were collected from the “tester.trace” files generated during these test runs. These files were then processed to compute the average time taken for each type of request.

## Performance Metrics

The primary metric used for this comparison is the average time taken (in milliseconds) to process requests. This metric provides insights into how quickly the system responds to different types of requests.

## Performance Before Adding Indexes

The following table presents the average time (in milliseconds) taken to process various requests before adding indexes:

Tabla

Descripción generada automáticamente

## Performance After Adding Indexes

The following table presents the average time (in milliseconds) taken to process the same set of requests after adding indexes:

Tabla

Descripción generada automáticamente

## Analysis

A comparison of the performance metrics before and after adding indexes reveals the following insights:

1. Improvement in Response Times:

Certain request types, such as any/system/welcome and any/system/panic, showed significant improvement in response times after adding indexes.

Requests related to listing (any/contract/list) and showing (client/progress-log/show) also benefited from the indexes, although the improvements were modest.

1. Mixed Results:

Some request types, such as client/contract/create and client/contract/list-mine, exhibited increased response times after adding indexes. This could be due to the overhead introduced by maintaining indexes during write operations.

1. Consistency:

The response times for some requests, such as authenticated/system/sign-out, remained largely unchanged, indicating that these operations were not significantly impacted by the presence of indexes.

## Statistical Analysis

To statistically validate the observed improvements, a 95% confidence interval was computed for the average response times before and after adding indexes. Additionally, a hypothesis contrast was performed to determine if the changes were statistically significant.

### 95%-confidence interval

* Before adding indexes: The confidence interval for the average response time was calculated based on the collected data.
* After adding indexes: Similarly, the confidence interval was computed for the new set of data.

Tabla

Descripción generada automáticamenteTabla

Descripción generada automáticamente

### 95%-confidence hypothesis contrast

The hypothesis contrast revealed that the changes in response times for certain request types were statistically significant, confirming that the addition of indexes had a positive impact on performance.

Tabla

Descripción generada automáticamente

# Chapter on Profiling

Software profiling was conducted using VisualVM tool to identify bottlenecks in the code. The profiling results highlighted areas with the most significant performance issues, leading to targeted optimizations in future implementations.

Tabla

Descripción generada automáticamente con confianza media

# Conclusions

The comprehensive testing, both functional and performance, demonstrated the system's robustness, efficiency, and reliability. The detailed profiling provided insights into performance bottlenecks.

# Bibliography

Intentionally blank.