**INFORME DE PRUEBAS**



**Grado en Ingeniería Informática – Ingeniería del Software**

**Diseño y Pruebas 2**

**Curso 2023-2024**

Lucía Campos Díez

### Índice

1. [Executive summary 4](#_Toc28482)

[2. Table Review 5](#_Toc28483)

[3. Introduction 6](#_Toc28484)

[4. Contents 7](#_Toc28485)

[5.1 Funcionalm Testing 7](#_Toc28486)

[5.2 Performance Testing 14](#_Toc28487)

[6. Conclusions 17](#_Toc28488)

[**7. Bibliography** 18](#_Toc28489)

# 

# Executive summary

Testing is a fundamental element in software development and maintenance, helping development team members verify the proper functioning of the system and detect errors.  
This document presents a report on the testing carried out by Student 5, organized into sections on functional testing and performance testing; it also includes an analysis of the test coverage.

# Review Table

|  |  |  |
| --- | --- | --- |
| **Version** | **Description** | **Date** |
| v1.0 | Initial Version | 26/05/2025 |
| v2.0 | Initial Version | 26/05/2025 |

# Introduction

In the first section, focused on functional testing, the implemented test cases will be discussed grouped by implementation, including their effectiveness in detecting errors.  
The second section addresses the system’s performance through graphs and statistical analysis of the request response times, using 95% confidence intervals before and after refactoring with corresponding metrics, along with a hypothesis test based on the analysis.

# Contents

## Funcional Testing

Below is a brief summary of the implemented test cases, with additional explanations provided for those tests that uncovered errors in the functionality implementation. Please note that in cases where a test failed due to a detected error or produced an unexpected result, the error was fixed and the test was repeated to ensure proper operation.

### InvolvedIn

* **createInvolvedIn.hack**: We first log in as Technician 1 and then proceed to test various ways to exploit the system. Examples include manipulating the inspection form to submit invalid values—such as an empty maintenance record, an empty task, a published maintenance record, or missing values like the InvolvedIn ID.
* **deleteInvolvedIn.hack**: After logging in as Technician 1, we attempt to hack the system by modifying values in the inspection form. For instance, changing the maintenance record ID to one that has already been published.
* **fulfill.hack** and **fulfill2.hack**: These tests are executed after logging in as Technician 1 and are designed to complete the code coverage for the "fulfill" functionality, including potential misuse scenarios.
* **showInvolvedIn.hack**: Logged in as Technician 1, we try to access or manipulate data not owned by the user. For example, changing the involvedInId to one that doesn’t belong to the logged-in technician, using an empty or invalid value (e.g., characters or symbols).
* **show2InvolvedIn.hack**: This test suite is used to cover additional edge cases and complete the percentage coverage of the "show" feature.
* **updateInvolvedIn.hack**: After logging in as Technician 1, we test edge cases such as submitting an empty involvedInId, modifying data linked to a published maintenance record, and using various invalid values like nonexistent IDs or empty fields.

### MaintanenceRecord

* **createRecord.hack**: Logged in as Technician 1, we attempt to create a maintenance record with invalid data (e.g., empty or incorrectly formatted IDs using letters or symbols).
* **deleteRecord.hack**: This test checks the system’s response when trying to delete a record using invalid or tampered values (e.g., a malformed maintenance record ID).
* **publishRecord.hack**: We attempt to publish a maintenance record with invalid conditions: e.g., not owned by the logged-in technician, records without any tasks, or only containing unpublished tasks.
* **showRecord.hack**: We test scenarios where a user tries to view a record not owned by them, with invalid or empty recordId values.
* **updateRecord.hack**: We simulate various hacking scenarios such as submitting an empty or invalid recordId, editing published records, or providing incorrect/unauthorized IDs.

### Task

* **createTask.hack** and **create2Task.hack**: Logged in as Technician 1, we attempt to create tasks using invalid input—e.g., empty or malformed task IDs, recordIds that don’t belong to the user, or combinations of symbols and characters.
* **deleteTask.hack**: We test the system’s validation by trying to delete a task using an invalid or tampered task ID.
* **publishTask.hack**: We try to publish a task linked to a maintenance record not owned by the logged-in technician or that doesn't meet publishing requirements (e.g., record without tasks or with only unpublished tasks).
* **showTask.hack**: We check for improper access by using task IDs not linked to the logged-in technician, or using invalid values such as symbols or letters.
* **taskListRecord.hack**: These tests complete the coverage of the "task list by record" functionality, simulating edge cases or unauthorized access attempts.
* **updateTask.hack**: We attempt to update tasks using empty, incorrect, or unauthorized task IDs, including values that do not exist or do not belong to the technician.

Program

* **program.safe**: This test ensures that all the requirements specified in the provided documentation are properly fulfilled. It includes several invalid requests to verify system robustness, such as entering incorrect form values (e.g., a past date, negative numbers, overly long inputs, etc.).

## Performance Testing

Pre-refactoring tests with metrics

Tabla

Descripción generada automáticamente

Post-refactoring tests with metrics

Tabla

Descripción generada automáticamente

It can be observed by comparing both graphs that the requests requiring the most time are the update of the entity maintenanteRecord.

Additionally, it is evident that, overall, before refactoring, the requests took longer to process compared to after the refactoring, where the average response time is significantly lower. However, the operations that previously took the most time now take slightly longer.

The confidence intervals with a 95% confidence level are (23, 47) before refactoring and (29, 34) after refactoring.

**Hipótesis de contraste**

Tabla

Descripción generada automáticamente

With this results we can say that the use of indixes makes the performance worse because

# Conclusions

The report includes all the tests carried out by Student 5, which have been useful for detecting some errors in the code and verifying its correct functionality, as well as for measuring performance. This has allowed us to draw conclusions about the efficiency of our system in handling requests.

# **Bibliography**

Intentionally in blank.