Workgroup C1.067

Testing report

24/05/2025

Aarón Jesús Acuña Bellido – [aaracubel@alum.us.es](mailto:aaracubel@alum.us.es)

https://github.com/javiarellanoo/Acme-ANS-D04

Table of Contents

[EXECUTIVE SUMMARY: 2](#_Toc199089501)

[REVISION TABLE 3](#_Toc199089502)

[INTRODUCTION 4](#_Toc199089503)

[FUNCTIONAL TESTING 4](#_Toc199089504)

[Operations on Tasks by Technicians 4](#_Toc199089505)

[Operations on Maintenance Records by Technicians 8](#_Toc199089506)

[Operations on Maintenance Records Tasks by Technicians 12](#_Toc199089507)

[PERFORMANCE TESTING 14](#_Toc199089508)

[Mean Confidence Interval 15](#_Toc199089509)

[Contrasting information 16](#_Toc199089510)

[CONCLUSIONS 17](#_Toc199089511)

[BIBLIOGRAPHY 18](#_Toc199089512)

# EXECUTIVE SUMMARY:

This document serves as a comprehensive and complete overview of the testing procedures applied to validate the functionality and performance of the features involved in Student #5 requirements, with the aim of assessing complete security within the application, complying with “Ley Orgánica 3/2018”.

The functional testing section of the document includes brief descriptions of the test procedures performed for each feature available to technicians, with respect to Tasks, Maintenance Records and the intermediate entity for the many to many relationships between the previous ones (MaintenanceRecordsTasks). To carry out said testing, a set of legal and hacking scenarios have been recorded and reproduced. Across these tests, particular attention was given to edge cases within the stablished ranges of attributes, input validation, navigation attributes and access control. With respect to read-only attributes, test cases have not been recorded, since the related entities do not have any of them.

The second and last part of the report is devoted to performance testing. By computing performance in two different hardware environments, this report establishes the required confidence intervals and compares the execution times to determine the most powerful machine.

# REVISION TABLE

|  |  |  |
| --- | --- | --- |
| Revision Number | Date | Description |
| 1.0 | 25/05/2025 | Initial version of the document |
|  |  |  |
|  |  |  |

# INTRODUCTION

The goal of this document is to provide an analysis of the testing procedure of the application for the requirements related to Student #5. The document will be structured in two parts.

The first one includes information related to the functional testing process, including a list of all test cases implemented, grouped by feature. For each test case, a succinct description of the tests conducted and an indication of whether bugs have been discovered will be provided.

The second section includes the information related to performance testing, including charts regarding this performance as well as a 95% confidence interval for the time taken by the project to serve the requests of the functional tests in two different hardware environments. Finally, the most powerful computer will be determined.

# FUNCTIONAL TESTING

## Operations on Tasks by Technicians

List

The following safe cases have been implemented:

* Go into the list of tasks of each user and check every page to make sure that entries are rendered properly. This has been done across different test cases, since it was also required to be able to access the “show” feature of each Task.
  + Detection of bugs: no bugs were detected.
* Go into the list of published tasks and check every page to make sure that entries are rendered correctly. This has been done across different test case, since it was also required to be able to access the “show” feature of each Task.
  + Detection of bugs: no bugs were detected

The hacking cases include:

* Request the task listing feature with “published” value null as a technician and request it again, with “published” value equals to true, false and null, as any other realm, for example non authenticated or administrator. An exception of authorization was thrown.
  + Detection of bugs: no bugs were detected.

List Tasks associated with a Maintenance Record

The following safe cases have been implemented:

* Go into the details of a Maintenance Record that is in draft mode that belongs to the technician that is currently logged in and press the button to show the tasks associated with that maintenance record and check every page to make sure that entries are rendered properly.
  + Detection of bugs: no bugs were detected.
* Go into the details of a Maintenance Record that is published and press the button to show the tasks associated with that maintenance record and check every page to make sure that entries are rendered properly.
  + Detection of bugs: no bugs were detected.

The hacking cases implemented include:

* Requesting the feature for a non-existing maintenance record. Check that an authorization exception is thrown.
  + Detection of bugs: none were detected.
* Requesting the feature for a maintenance record that is in draft mode and with any other principal than with the technician who created it. Check that an authorization exception is thrown.
  + Detection of bugs: none were detected.

Show

The safe cases include:

* Request the feature for all tasks in the sample data, by the technician that created them, checking that everything renders properly.
  + Detection of bugs: no bugs were detected
* Request the feature for a published task by a technician that did not create it, checking that it is visible.
  + Detection of bugs: no bugs were detected.

The hacking cases implemented include:

* Request the feature for a task in draft mode with a realm distinct to the one associated with said task (i.e. a different technician or a non-authenticated user). An authorization exception is thrown
  + Detection of bugs: none were detected
* Request the feature for a non-existent task. An authorization exception is thrown.
  + Detection of bugs: none were detected.

Create

The following safe test cases were performed:

* Sending the form with all null values to check that no exceptions are thrown.
  + Detection of bugs: no bugs were detected.
* For each attribute of the form, as many invalid and valid data as possible have been considered. Said variations were taken from the “Sample-Data” file located in the Scrapbook folder of the workspace.
  + Detection of bugs: none were detected.
* Submit a form with all valid data to see that the task is created properly:
  + Detection of bugs: none were detected.

The hacking cases implemented include:

* Requesting the feature with other realms. Checking that an authorization error is thrown
  + Detection of bugs: none were detected

Update

The safe cases implemented were:

* Submit an empty form to check that no exceptions were thrown and errors were reported in the correct attributes.
  + Detection of bugs: no bugs were detected.
* For each attribute of the form, as many invalid and valid data as possible have been considered. Said variations were taken from the “Sample-Data” file located in the Scrapbook folder of the workspace.
  + Detection of bugs: none were detected.
* Submit a form with all valid data to see that the task is updated properly:
  + Detection of bugs: none were detected.

The hacking cases implemented include, among others:

* Requesting the feature for a Task that has already been published (with both the technician that created it and another one). Check that an authorization exception is thrown.
  + Detection of bugs: none were detected
* Requesting the feature for a Task in draft mode that does not belong to the logged in technician. Check that an authorization exception is thrown.
  + Detection of bugs: none were detected
* Request the feature for a non-existing Task.
  + Detection of bugs: none were detected.

Publish:

The safe cases implemented were:

* Submit an empty form to check that no exceptions were thrown and errors were reported in the correct attributes.
  + Detection of bugs: no bugs were detected.
* For each attribute of the form, as many invalid and valid data as possible have been considered. Said variations were taken from the “Sample-Data” file located in the Scrapbook folder of the workspace.
  + Detection of bugs: none were detected.
* Submit a form with all valid data to see that the task is published properly:
  + Detection of bugs: none were detected.

The hacking cases implemented include, among others:

* Requesting the feature for a Task that has already been published (with both the technician that created it and another one). Check that an authorization exception is thrown.
  + Detection of bugs: none were detected
* Requesting the feature for a Task in draft mode that does not belong to the logged in technician. Check that an authorization exception is thrown.
  + Detection of bugs: none were detected
* Request the feature for a non-existing Task.
  + Detection of bugs: none were detected.

Delete:

The following safe cases were implemented:

* Requesting the feature normally, with a task in draft mode. Check that the task is properly deleted
  + Detection of bugs: none were detected

The hacking cases include, among others:

* Requesting the feature for a non-existing task. Check that an authorization exception is thrown.
  + Detection of bugs: no bugs were found
* Request the feature for an already published task. Check that an authorization error is thrown.
  + Detection of bugs: no bugs were found.
* Request the feature directly through the URL. Check that the GET phase of the delete request has been restricted with an authorization error so that only requests coming from the application are valid.
  + Detection of bugs: no bugs were detected
* - Request the feature for a task in draft mode with a different principal than the one who created it. Check that it returns an authorization error.
  + Detection of bugs: no bugs were found.

## Operations on Maintenance Records by Technicians

List

The following safe cases have been performed:

* Go into the list of Maintenance Records of each technician and check every page to make sure that entries are rendered properly. This has been done across different test cases, since it was also required to be able to access the “show” feature of each Maintenance Record.
  + Detection of bugs: no bugs were detected.
* Go into the list of published Maintenance Record and check every page to make sure that entries are rendered correctly. This has been done across different test cases, since it was also required to be able to access the “show” feature of each Maintenance Record.
  + Detection of bugs: no bugs were detected

The hack cases include:

* Request the maintenance record listing feature with “published” value null as a technician and request it again, with “published” value equals to true, false and null, as any other realm, for example non authenticated or administrator. An exception of authorization was thrown.
  + Detection of bugs: no bugs were detected.

Show

The following safe cases have been performed:

* For each Maintenance Record in the sample data, it has been checked that they can be correctly seen, by their creator if they are in draft mode and by all technicians if the maintenance record is published.
  + Detection of bugs: none were detected

The following hack cases have been implemented, among others:

* Trying to show a non-existing maintenance record. It has been checked that an authorization exception is thrown.
  + Detection of bugs: none were found.
* Trying to show a non-published maintenance record by another principal other than the one who created the maintenance record. It has been checked that an authorization exception is thrown.
  + Detection of bugs: none were found

Create

The following safe cases have been performed:

* Submit an empty form to check that no exceptions were thrown and errors were reported in the correct attributes.
  + Detection of bugs: no bugs were detected.
* For each attribute of the form, as many invalid and valid data as possible have been considered. Said variations were taken from the “Sample-Data” file located in the Scrapbook folder of the workspace. Natural intelligence has been used to generate different variations of the “moment” and “nextInspectionDate” attributes, to check that the moment in which the maintenance record was created must be in the past, the next inspection due date must be in the future and that also the next inspection due date must be after the moment of creation of the maintenance record.
  + Detection of bugs: none were detected.
* Submit a form with all valid data to see that the maintenance record is created properly:
  + Detection of bugs: none were detected.

The hacking cases implemented include, among others:

* Hack “aircraft” navigation attribute using the “DevTools” and using an Id belonging to a non-existing aircraft (i.e. 999 or -1).
  + Detection of bugs: none were encountered

Update

The following safe cases have been performed:

* Submit an empty form to check that no exceptions were thrown and errors were reported in the correct attributes.
  + Detection of bugs: No bugs were detected.
* For each attribute of the form, as many invalid and valid data as possible have been considered. Said variations were taken from the “Sample-Data” file located in the Scrapbook folder of the workspace. Natural intelligence has been used to generate different variations of the “moment” and “nextInspectionDate” attributes, to check that the moment in which the maintenance record was created must be in the past, the next inspection due date must be in the future and that also the next inspection due date must be after the moment of creation of the maintenance record.
  + Detection of bugs: none were detected.
* Submit a form with all valid data to see that the maintenance record is updated properly:
  + Detection of bugs: none were detected.

The hacking cases implemented include, among others:

* Requesting the feature for a non-existing maintenance record. Check that an authorization exception is thrown.
  + Detection of bugs: no bugs were found
* Requesting the feature for a Maintenance Record that has already been published (with both the technician that created it and another one). Check that an authorization exception is thrown.
  + Detection of bugs: none were detected
* Requesting the feature for a Maintenance Record that is in draft mode with a technician other than the one who created it. Check that an authorization exception is thrown.
  + Detection of bugs: none were detected
* Hack navigation attributes “aircraft” using the “DevTools” and using an Id belonging to a non-existing aircraft (i.e. 999 or -1).
  + Detection of bugs: none were detected

Publish

The following safe cases have been performed:

* Submit an empty form to check that no exceptions were thrown and errors were reported in the correct attributes.
  + Detection of bugs: no bugs were detected.
* For each attribute of the form, as many invalid and valid data as possible have been considered. Said variations were taken from the “Sample-Data” file located in the Scrapbook folder of the workspace. Natural intelligence has been used to generate different variations of the “moment” and “nextInspectionDate” attributes, to check that the moment in which the maintenance record was created must be in the past, the next inspection due date must be in the future and that also the next inspection due date must be after the moment of creation of the maintenance record.
  + Detection of bugs: none were detected.
* Submit a form with all valid data to see that the maintenance record published properly:
  + Detection of bugs: none were detected.

The hacking cases implemented include, among others:

* Requesting the feature for a Maintenance Record that has already been published (with both the technician that created it and another one). Check that an authorization exception is thrown.
  + Detection of bugs: none were detected.
* Requesting the feature for Maintenance Record in draft mode that cannot be published, that it does not have any associated task or that some of the tasks associated are not published. Check that an authorization exception is thrown.
  + Detection of bugs: none were detected.
* Requesting the feature for a Maintenance Record in draft mode that does not belong to the logged in technician. Check that an authorization exception is thrown.
  + Detection of bugs: none were detected.
* Request the feature for a non-existing Maintenance Record.
  + Detection of bugs: none were detected.
* Hack navigation attributes “aircraft” using the “DevTools” and using an Id belonging to a non-existing aircraft (i.e. 999 or -1).
  + Detection of bugs: none were detected.

Delete

The following safe cases were conducted:

* Requesting the feature normally, with a maintenance record in draft mode. Check that the maintenance record is properly deleted
  + Detection of bugs: none were detected.

The hacking cases include, among others:

* Requesting the delete feature for a non-existing maintenance record. Check that an authorization exception is thrown.
  + Detection of bugs: no bugs were found
* Request the feature for an already published maintenance record. Check that an authorization error is thrown.
  + Detection of bugs: no bugs were found.
* Request the feature directly through the URL. Check that the GET phase of the delete request has been restricted with an authorization error so that only requests coming from the application are valid.
  + Detection of bugs: no bugs were detected
* - Request the feature for a maintenance record in draft mode with a different principal than the technician who created it. Check that it returns an authorization error.
  + Detection of bugs: no bugs were found.

## Operations on Maintenance Records Tasks by Technicians

Create

The following safe cases have been performed:

* Submit an empty form to check that no exceptions were thrown and errors were reported in the correct attributes.
  + Detection of bugs: no bugs were detected.
* Submit a form with all valid data to see that the task is added to the maintenance record properly.
  + Detection of bugs: none were detected.

The hacking cases implemented include, among others:

* Hack “task” navigation attribute using the “DevTools” and using an Id belonging to a non-existing task (i.e. 999 or -1). Check that an authorization exception is thrown.
  + Detection of bugs: none were encountered.
* Hack “task” navigation attribute using the “DevTools” and using an Id belonging to a task that is not published and that belongs to other technician than the one who created the maintenance record. Check that an authorization exception is thrown.
  + Detection of bugs: none were encountered.
* Hack “task” navigation attribute using the “DevTools” and using and Id belonging to a task that has already been associated with the maintenance record. Check that an authorization exception is thrown.
  + Detection of bugs: none were encountered.
* Requesting the feature with a non-existing maintenance record. Check that an authorization exception is shown.
  + Detection of bugs: none were encountered.
* Requesting the feature with a maintenance record that is published. Check that an authorization exception is shown.
  + Detection of bugs: none were encountered.
* Requesting the feature with a maintenance record that is in draft mode and with a principal different from the technician who created the maintenance record. Check that an authorization exception is shown.
  + Detection of bugs: none were encountered.

Delete

The following safe cases were conducted:

* Submit an empty form to check that no exceptions were thrown and errors were reported in the correct attributes.
  + Detection of bugs: no bugs were detected.
* Submit a form with all valid data to see that the task is removed from the maintenance record properly.
  + Detection of bugs: none were detected.

The hacking cases include, among others:

* Hack “task” navigation attribute using the “DevTools” and using an Id belonging to a non-existing task (i.e. 999 or -1). Check that an authorization exception is thrown.
  + Detection of bugs: none were encountered.
* Hack “task” navigation attribute using the “DevTools” and using an Id belonging to a task that is not published and that belongs to other technician than the one who created the maintenance record. Check that an authorization exception is thrown.
  + Detection of bugs: none were encountered.
* Hack “task” navigation attribute using the “DevTools” and using and Id belonging to a task that has not been associated with the maintenance record. Check that an authorization exception is thrown.
  + Detection of bugs: none were encountered.
* Requesting the feature with a non-existing maintenance record. Check that an authorization exception is shown.
  + Detection of bugs: none were encountered.
* Requesting the feature with a maintenance record that is published. Check that an authorization exception is shown.
  + Detection of bugs: none were encountered.
* Requesting the feature with a maintenance record that is in draft mode and with a principal different from the technician who created the maintenance record. Check that an authorization exception is shown.
  + Detection of bugs: none were encountered.

# PERFORMANCE TESTING

In this last section, we will evaluate the performance of the project by measuring the elapsed time taken to complete requests during functional testing. The objective is to assess how quickly the system responds under real conditions to determine which computer performs the best.

To collect the data required, we will run all functional tests for technicians’ features. We will be using the following devices:

* HP 15s-fq4xxx: 16 GB RAM, 500 GB SSD memory
* Lenovo Ideapad 5: 32 GB RAM, 512 GB SSD

## Mean Confidence Interval

A graph with blue and black text

AI-generated content may be incorrect.After cleaning up and treating the data obtained from the trace file that is generated as a result of replaying test suites, I have performed an analysis using the procedure explained in the subject to obtain an average time for each of the requests executed.

For the first computer, we have obtained a grand average of 71.6 ms. As can be appreciated, we see that the MIR is the maintenance record list, whose average is 483.36 ms.

Using the data analyzer extension from Excel, we obtain that the amplitude of the confidence interval at 95% is 10.17 ms. By removing and adding this value from the average, we obtain the confidence interval: [0.061 s – 0.081 s]

A graph with blue and black lines

AI-generated content may be incorrect.

When doing the same with the Lenovo PC, we obtained a similar graph. In this case, the grand average stands at 86.41 ms. The MIR continues to be on the maintenance record list, with an average of 609.01 ms.

Using the data analyzer, we get that the amplitude of the confidence interval at 95% is 12.53 ms. We would obtain the following confidence interval: [0.074 s – 0.099s]

## Contrasting information

Given the previous results, we can induce that the computer with the best performance is the HP 15s-fq4xxx, as in general terms, all its average times are lower. Let us use the Z-Test with alpha 0.05 (one minus confidence) to verify our hypothesis.

Once the Z-Test has been conducted, the results are the following:

A table with numbers and symbols

AI-generated content may be incorrect.

As can be appreciated, the p-value, which is 0.07, is in interval (0.05, 1.00], then there is no significant improvement between each other and the differences amongst the times are not significantly different since they are globally the same.

# CONCLUSIONS

This document serves as a means of reporting the results of test cases for future references. The analysis provides a detailed evaluation of the test cases in order to identify possible bugs in the code and make informed decisions to improve the system as well as to ensure protection against possible hacking attempts.

Each test case was recorded systematically, grouped by feature and the bugs found, if any, have been reported. This serves as a transparent reference of the testing process.

Through the analysis of the performance on the two different computers, we have demonstrated that the average of the HP 15s-fq4xxx computer has consistently obtained a better performance. As expected, both systems showed similar behavior in terms of request distribution, however, the obtained MIR has been different in each case. Since the difference on average of the MIR obtained for the second PC and the one obtained for the first one in the first case are very close, this can be a punctual case.

This testing report provides concise descriptions of the tests performed, in hopes that they serve as solid foundation for quality assurance in the near future.

# BIBLIOGRAPHY

Design & Testing II Slides – S02 Performance Testing