Testing Report

Acme AirNav Solutions, Inc

Group Number: C1.066
Repository: `https://github.com/mquirosq/DP2-C1.066`

Members:
María Quirós Quiroga, marquiqui@alum.us.es
Guillermo Rodríguez Narbona, guirodnar@alum.us.es
Ignacio Mora Pérez, ignmorper1@alum.us.es
Daniel Herrera Urbano, danherurb@alum.us.es
Alejandro Parody Quirós, aleparqui@alum.us.es

May 26, 2025

Contents

Executive Summary

This document provides an overview of the tests performed to validate the functionality and performance of the application features related to Student #5. The goal was to assess the security of the project for legal and illegal actions.

The document is divided in two sections; the first, functional testing, includes a description of every test performed over the entities and their corresponding functionalities. Each functionality was tested for safe and hacking actions. We have tested against edge cases, input validation and navigation, and read-only attributes.

For performance testing, covered in the second section of the report, we evaluate how the application performs under two different conditions. Comparison of performance in two different hardware environments, establishing confidence intervals, and statistically comparing execution times to determine the most powerful device.

Revision History

Revision	Date	Description
1.0	2025-05-26	Initial draft

1. Introduction

The purpose of this document is to record and provide information on the tests performed on the entities of the fifth student with the intention of assessing the quality of their related requirements.

The first section of this document is focused on functional testing. It includes a list with the test cases implemented, grouped by entity and feature. For each test case, a brief description is provided.

In the second section, it includes information about performance testing. This will be done by providing charts and a 95% confidence interval for the time taken to serve the requests. The information required for this analysis will be obtained from two different computers to determine which is the most powerful.

2. Functional Testing

This section consists of a list of the test cases implemented for each entity.

2.1. Operations of technicians on maintenance records

2.1.1 List

For the safe cases implemented, we have the following:

- Go to each technician and check that every page that included a listing of the entity is rendered correctly.
- No bugs were detected in this case.

The hacking cases implemented include:

- Request the list feature via URL from other realms, specifically not authenticated, and as an administrator.
- In this case, no bugs were detected.

2.1.2 Show

For the safe cases implemented, we have the following:

- Request the show feature for every maintenance record found in the sample data, checking that they are rendered correctly.
- In this case, no bugs were detected.

The hacking cases implemented include:

- Change the URL navigation attribute with an empty value, a non-integer value, and an ID that does not belong to any maintenance record in the database. Check that an authorization exception is thrown.
- Request show for a non-published maintenance record that does not belong to the active technician. Check that an authorization exception is thrown.
- Request show for a maintenance record from a non-technical profile, using for this case both an administrator profile and as not logged in. Check that an authorization exception is thrown.
- In this case, no bugs were detected.

2.1.3 Create

For the safe cases implemented, we have the following:

- Submit the creation form empty, checking that no exceptions were thrown and the errors reported were shown in the correct attributes.
- For each attribute of the form, fill it while leaving the rest blank, testing both valid and invalid data variations. Verified that the inspection date and moment-time restrictions were correctly shown.
- Submit a form with valid data to check that the creation of a maintenance record is done properly.
- In this case, no bugs were detected.

The hacking cases implemented include:

- Request the create feature from other realms (administrator and not logged in) and check that the authorization exception is thrown.
- Hacking the 'aircraft' navigation attribute using DevTools. Changed the value to a non-existing aircraft, empty value, and non-numeric value. Check that the authorization exception is thrown.
- In this case, no bugs were detected.

2.1.4 Update

For the safe cases implemented, we have the following:

- Submit the update form empty, checking that no exceptions were thrown and the errors reported were shown in the correct attributes.
- For each attribute of the form, fill it while leaving the rest blank, testing both valid and invalid data variations. Verified that the inspection date and moment-time restrictions were correctly shown.
- Submit a form with valid data to check that the update of a maintenance record is done properly.
- In this case, no bugs were detected.

The hacking cases implemented include:

- Request the update feature from other realms (administrator and not logged in) and check that the authorization exception is thrown.
- Hacking the 'aircraft' navigation attribute using DevTools. Changed the value to a non-existing aircraft, empty value, and non-numeric value. Check that the authorization exception is thrown.
- Request the update feature on a maintenance record that has already been published. Check that the authorization exception is thrown.
- Request the update feature on a maintenance record that belongs to another technician, both published and not. Check that the authorization exception is thrown.
- Request the update feature on an empty id, non-existing maintenance record, and a non-numeric value. Check that the authorization exception is thrown.
- Change the values of the read-only attributes in the form and proceed with the submission. Check that the values introduced were ignored.
- In this case, no bugs were detected.

2.1.5 Delete

For the safe cases implemented, we have the following:

- Request the delete feature on a valid maintenance record, checking that the intermediate entities linked to that record were correctly deleted.

- In this case, no bugs were detected.

The hacking cases implemented include:

- Request the delete feature from other realms (administrator and not logged in) and check that the authorization exception is thrown.
- Request the delete feature on a maintenance record that has already been published. Check that the authorization exception is thrown.
- Request the delete feature on a maintenance record that belongs to another technician, both published and not. Check that the authorization exception is thrown.
- Request the delete feature on an empty id, non-existing maintenance record, and a non-numeric value. Check that the authorization exception is thrown.
- Change the values of the read-only attributes in the form and proceed with the submission. Check that the values introduced were ignored.
- In this case, no bugs were detected.

2.1.6 Publish

For the safe cases implemented, we have the following:

- Check the conditions for publishing a maintenance record by trying to publish a maintenance record with no tasks associated or not all tasks published. Check that the record was not published.
- Publish a maintenance record with all associated tasks published. Check that the record was published correctly.
- In this case, no bugs were detected.

The hacking cases implemented include:

- Request the publish feature from other realms (administrator and not logged in) and check that the authorization exception is thrown.
- Request the publish feature on a maintenance record that has already been published. Check that the authorization exception is thrown.
- Request the publish feature on a maintenance record that belongs to another technician, both published and not. Check that the authorization exception is thrown.

- Request the publish feature on an empty id, non-existing maintenance record, and a non-numeric value. Check that the authorization exception is thrown.
- Change the values of the read-only attributes in the form and proceed with the submission. Check that the values introduced were ignored.
- In this case, no bugs were detected.

2.2. Operations of technicians on tasks

2.2.1 List

For the safe cases implemented, we have the following:

- Go to each technician and check that every page that included a listing of the entity is rendered correctly.
- In this case, no bugs were detected.

The hacking cases implemented include:

- Request the list feature via URL from other realms, specifically not authenticated, and as an administrator.
- In this case, no bugs were detected.

2.2.2 Show

For the safe cases implemented, we have the following:

- Request the show feature for every task found in the sample data, checking that they are rendered correctly.
- In this case, no bugs were detected.

The hacking cases implemented include:

- Change the URL navigation attribute with an empty value, a non-integer value, and an ID that does not belong to any task in the database. Check that an authorization exception is thrown.

- Request show for a task that does not belong to the active technician. Check that an authorization exception is thrown.
- Request show for a task from a non-technical profile, using for this case both an administrator profile and as not logged in. Check that an authorization exception is thrown.
- In this case, no bugs were detected.

2.2.3 Create

For the safe cases implemented, we have the following:

- Submit the creation form empty, checking that no exceptions were thrown and the errors reported were shown in the correct attributes.
- For each attribute of the form, fill it while leaving the rest blank, testing both valid and invalid data variations.
- Submit a form with valid data to check that the creation of a maintenance record is done properly.
- In this case, no bugs were detected.

The hacking cases implemented include:

- Request the create feature from other realms (administrator and not logged in) and check that the authorization exception is thrown.
- In this case, no bugs were detected.

2.2.4 Update

For the safe cases implemented, we have the following:

- Submit the update form empty, checking that no exceptions were thrown and the errors reported were shown in the correct attributes.
- For each attribute of the form, fill it while leaving the rest blank, testing both valid and invalid data variations.
- Submit a form with valid data to check that the update of a task is done correctly.
- In this case, no bugs were detected.

The hacking cases implemented include:

- Request the update feature from other realms (administrator and not logged in) and check that the authorization exception is thrown.
- Request the update feature on a task that has already been published. Check that the authorization exception is thrown.
- Request the update feature on a task that belongs to another technician, both published and not. Check that the authorization exception is thrown.
- Request the update feature on an empty id, non-existing task, and a non-numeric value. Check that the authorization exception is thrown.
- In this case, no bugs were detected.

2.2.5 Delete

For the safe cases implemented, we have the following:

- Request the delete feature on a valid task, checking that the inter- mediate entities linked to that record were correctly deleted.
- In this case, no bugs were detected.

The hacking cases implemented include:

- Request the delete feature from other realms (administrator and not logged in) and check that the authorization exception is thrown.
- Request the delete feature on a task that has already been published. Check that the authorization exception is thrown.
- Request the delete feature on a task that belongs to another technician, both published and not. Check that the authorization exception is thrown.
- Request the delete feature on an empty id, non-existing task, and a non-numeric value. Check that the authorization exception is thrown.
- In this case, no bugs were detected.

2.2.6 Publish

For the safe cases implemented, we have the following:

- Publish a task. Check that the task was published correctly.
- In this case, no bugs were detected.

The hacking cases implemented include:

- Request the publish feature from other realms (administrator and not logged in) and check that the authorization exception is thrown.
- Request the publish feature on a task that has already been published. Check that the authorization exception is thrown.
- Request the publish feature on a task that has already been published. Check that the authorization exception is thrown.
- Request the publish feature on an empty id, non-existing task, and a non-numeric value. Check that the authorization exception is thrown.
- In this case, no bugs were detected.

2.3. Operations of technicians on task records

2.3.1 List

For the safe cases implemented, we have the following:

- Go to each technician's task record and check that every page that included a listing of the entity is rendered correctly.
- In this case, no bugs were detected.

The hacking cases implemented include:

- Request the list feature via URL from other realms, specifically not authenticated, and as an administrator.
- In this case, no bugs were detected.

2.3.2 Show

For the safe cases implemented, we have the following:

- Request the show feature for every task record found in the sample data, checking that they are rendered correctly.
- In this case, no bugs were detected.

The hacking cases implemented include:

- Request the show feature for every task found in the sample data, checking that they are rendered correctly.
- Request show for a task record that belongs to a non-published maintenance record of another technician. Check that an authorization exception is thrown.
- Request show for a task record from a non-technical profile, using for this case both an administrator profile and as not logged in. Check that an authorization exception is thrown.
- In this case, no bugs were detected.

2.3.3 Create

For the safe cases implemented, we have the following:

- Submit the creation form empty, checking that no exceptions were thrown and the errors reported were shown in the correct attributes.
- Submit a form with valid data to check that the creation of a maintenance record is done properly.
- In this case, no bugs were detected.

The hacking cases implemented include:

- Request the create feature from other realms (administrator and not logged in) and check that the authorization exception is thrown.
- In this case, no bugs were detected.

2.3.4 Delete

For the safe cases implemented, we have the following:

- Request the delete feature on a valid task record, checking that the inter- mediate entities linked to that record were correctly deleted.
- In this case, no bugs were detected.

The hacking cases implemented include:

- Request the delete feature from other realms (administrator and not logged in) and check that the authorization exception is thrown.
- Request the delete feature in a task record that belongs to a maintenance record that has already been published. Check that the authorization exception is thrown.
- Request the delete feature on a task record that belongs to another technician's maintenance record, both published and not. Check that the authorization exception is thrown.
- Request the delete feature on an empty id, non-existing task, and a non-numeric value. Check that the authorization exception is thrown.
- In this case, no bugs were detected.

3. Performance testing

In this section, we will evaluate the performance of the project by measuring wall time, the elapsed real time taken to complete requests during functional tests. The goal is to assess how quickly the system responds under real conditions to determine which of the two computers perform better.

To collect the required data for the analysis, we will run all the functional tests for technician feature. The devices used are the following:

- Tabletop PC: 32 GB RAM and 1TB of memory.
- HP Victus: 32 GB RAM and 1TB of memory.

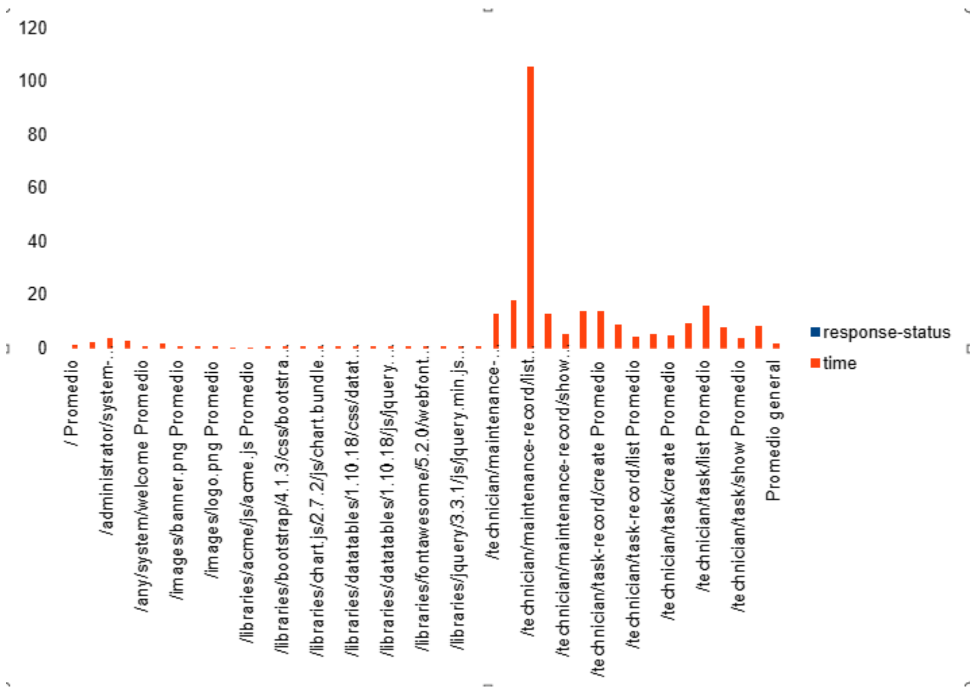

Figure 1: Graph showing the execution time of Tabletop PC

3.1. Mean confidence interval

After treating the data obtained from the test.trace file, we performed the analysis using Excel and following the proceedings in the slides to obtain the average wall time for each of the requests executed in the tests.

For the tabletop PC, we obtained a grand average of 1,87 ms. As we can see in graph 1, the MIR is the maintenance record listing, whose average stands at 105 ms, much higher than the rest of the requests.

Using the Excel data analyser, we obtain the amplitude of the confidence interval at 95%, 2.018395 ms. By removing this amount from the average and adding it, we obtain the confidence interval: 1.721920 ms.

For the HP Victus we obtained a grand average of 25,37 ms. As we can see in graph 2, the MIR is the maintenance record listing, whose average stands at 162 ms, much higher than the rest of the requests.

Using the Excel data analyser, we obtain the amplitude of the confidence interval at 95%, 28.23862957 ms. By removing this amount from the average and adding it, we obtain the confidence interval: 22.51075462 ms.

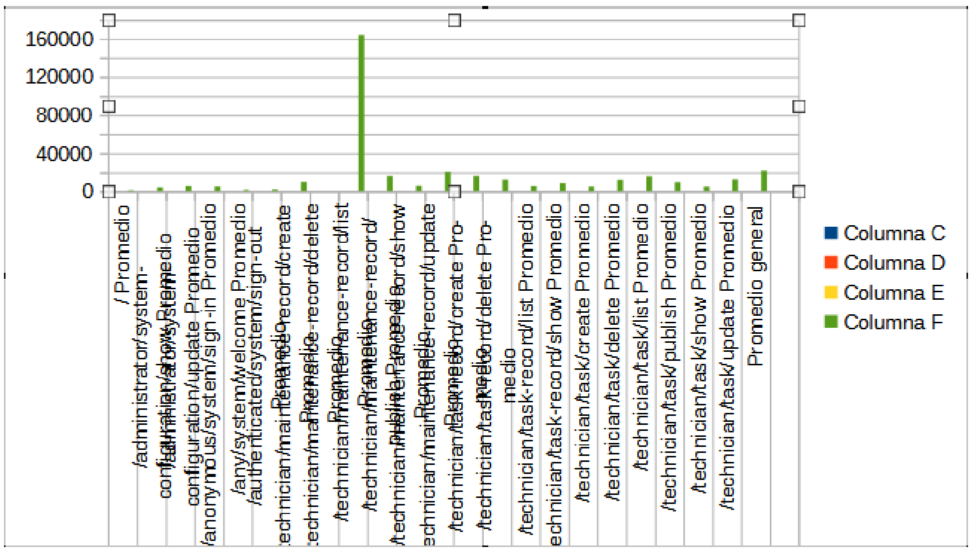

Figure 2: Graph showing the execution time of HP Victus

3.2. Hypothesis contrast

Once we have processed the data for both computers, we will prepare their performance. To do this, we will perform a 95% confidence hypothesis contrast regarding which the most powerful computer is.

Given the results above, we can intuitively think that the computer with the best performance is the tabletop PC, as, in general, all of its average times are lower. To ratify this, we used the Z-test with an alpha value of 0,05. We introduced the tabletop PC as the first one and the Victus data as the second one and obtained a p-value of 0,0005317. As the p-value is between zero and the alpha value, we can ensure that the best performing computer will be the one with the lowest average wall, that is, the tabletop PC.

4. Conclusions

This document serves as a means to construct and report the information obtained from performing the test cases on the entities of Student 5. The analysis recorded in this document provides an evaluation that can be used to identify potential bugs in the code, and from that generate potential solutions, making an informed decision.

Each test was recorded in a structured manner, serving as a transparent reference for the testing process.

Through the wall-time analysis, we have evaluated the performance of the project on two different devices. With this we have also demonstrated that one of those performed better than the other through their grand average.

References

[1] *05 - Requirements - Student #5*, Enseñanza Virtual, 2025. Available on Enseñanza virtual in the course insioe the Project Statement folder.