Testing Report

Acme AirNav Solutions, Inc

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

Student #2:
María Quirós Quiroga, marquiqui@alum.us.es

May 25, 2025

Contents

Executive Summary

This document provides a comprehensive overview of the testing procedures applied to validate the functionality and performance of the application features associated with Student #2. The goal was to assess the security of the project for both legal and illegal actions.

The functional testing section includes a succinct description of the test performed for each feature available to customers regarding bookings, booking records, passengers, and recommendations. Each functionality was tested through a set of safe (legal actions) and hacking (illegal actions) scenarios. Across these tests, particular attention was paid to edge cases, input validation, navigation and read-only attributes and access control.

Performance testing, covered in the second part of the report, evaluates how efficiently the application responds under operational conditions. By comparing performance across two different hardware environments, the report establishes confidence intervals and statistically compares the execution times to determine the most powerful device, which was the HP Victus. The MIR of the features was the booking listing.

Revision History

Revision	Date	Description
1.0	2025-05-22	Document initialization
1.1	2025-05-24	Add functional and performance testing sections

1. Introduction

The purpose of this document is to provide and record the tests performed to assess the quality of the application for the requirements associated to Student #2. This report is structured in two chapters.

The first section, includes information relative to functional testing. A listing with the test cases implemented, grouped by feature will be provided. For each test case, a succinct description plus an indication of how effective it was at detecting bugs is included.

The second section includes information relative to performance testing. Charts showing this performance, as well as a 95%-confidence interval for the wall time taken by the project to serve the requests of the functional tests in two different computers. Finally, a 95%-confidence hypothesis compares the performance of both computers to conclude which is the most powerful.

2. Functional testing

This section consists of a list of the tested cases implemented , grouped by feature.

2.1. Operations of customers on bookings

2.1.1 List

The safe cases implemented include:

- Go into the list for each of the users and check every page to make sure the entries render correctly. This has been done in Spanish and in English.
  - Detection of bugs: Did not detect any bugs.

The hacking cases implemented include:

- Request the booking list feature from other realms, specially as any (not authenticated) and as administrator. Check that an authorization exception is thrown.
  - Detection of bugs: Did not detect any bugs.

2.1.2 Show

The safe cases implemented include:

- Request the feature for every booking in the sample data. Check that everything is rendered correctly in both Spanish and English.
  - Detection of bugs: Detected a bug in the rendering where the "Edit" and "Publish" buttons disappeared when the publish failed.
- Change the system date to the future, in which there is no published flight with departure date after the current moment and check that bookings render correctly for each associated flight.
  - Detection of bugs: Detected a bug in which the list of flights in the show only included flights in the future. When the current date was after the scheduled departure date, the show threw an exception.

The hacking cases implemented include:

- Change the navigation attribute in the URL to an empty value, a non-integer value ("XXX") and an ID that does not correspond to a booking in the database. Check that an authorization exception is thrown.
  - Detection of bugs: Found bug caused by a method being called on the booking before checking if it was null.
- Request show for a booking from another customer. Check that an authorization exception is thrown.
  - Detection of bugs: No bugs were detected.
- Request show for a booking from a non-customer profile, especially any (not logged in) and administrator. Check that an authorization exception is thrown.
  - Detection of bugs: No bugs were detected.

2.1.3 Create

The safe cases implemented include:

- Submit an empty form to check that no exceptions were thrown and errors were reported in the correct attributes.
  - Detection of bugs: Did not detect any bugs.
- For each attribute on the form, leave the rest blank and try as many invalid data as possible, then try as many valid data variations as possible. The data variations were taken from the data provided in the "Sample-Data" file and generated using natural intelligence for strings matching a pattern. Special restrictions like the uniqueness of the locator code attribute were also checked.

  - Detection of bugs: Did not detect any bugs.

- Submit a form with valid data to check that the booking is created properly.
  - Detection of bugs: Did not detect any bugs.

The hacking cases implemented include:

- Request the booking create feature from other realms, specially as any (not authenticated) and as administrator. Check that an authorization exception is thrown.
  - Detection of bugs: Did not detect any bugs.
- Hack the "flight" navigation attributes using the "DevTools". Changed the value to a non-numeric value ("XXX"), a non-existent flight, a flight in draft mode and a published flight with departure date before the current date. Check that the application returned an authorization exception.
  - Detection of bugs: Did not detect any bugs.

2.1.4 Update

The safe cases implemented include:

- Submit an empty form to check that no exceptions were thrown and errors were reported in the correct attributes.
  - Detection of bugs: Did not detect any bugs.
- For each attribute on the form, leave the rest blank and try as many invalid data as possible, then try as many valid data variations as possible. The data variations were taken from the data provided in the "Sample-Data" file and generated using natural intelligence for strings matching a pattern. Special restrictions like the uniqueness of the locator code attribute were also checked.
  - Detection of bugs: Did not detect any bugs.
- Submit a form with valid data to check that the booking is updated properly.
  - Detection of bugs: Did not detect any bugs.

The hacking cases implemented include:

- Request the booking update feature from other realms, specially as any (not authenticated) and as administrator. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.

- Request the booking update feature on an owned draft booking whose associated flight had already departed and on an owned published booking. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.
- Request the booking update feature on a not-owned draft booking and a not-owned published booking. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.
- Request the booking update feature on a non-integer id ("XXX"), an empty id and an id that does not correspond to a booking. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.
- Change the value of read-only attributes in the form (draft mode, purchased date and price) and submit the form. Check that the values introduced were ignored.
  - Detection of bugs: Did not detect any bugs.
- Change the value of navigation attributes in the form (flight) using the "DevTools". Changed the value to a non-numeric value ("XXX"), a non-existent flight, a flight in draft mode and a published flight with departure moment before the current moment. Check that the application returned an authorization exception.
  - Detection of bugs: Did not detect any bugs.

2.1.5 Publish

The safe cases implemented include:

- Check the conditions for publishing a booking by trying to publish a booking with no credit card nibble associated, no passengers associated and no credit card nibble or passengers associated. Checked that the booking was not published and the corresponding errors were shown.
  - Detection of bugs: Did not detect any bugs.
- Published a booking with passengers and with a stored credit card nibble. Check that the booking was published correctly.
  - Detection of bugs: Did not detect any bugs.

The hacking cases implemented include:

- Request the booking publish feature from other realms, specially as any (not authenticated) and as administrator. Check that an authorization exception is thrown.

  - Detection of bugs: Did not detect any bugs.

- Request the booking publish feature of a not-owned booking in draft mode and in a not-owned published booking. Check that an authorization exception is thrown.
  - Detection of bugs: Did not detect any bugs.

- Request the booking publish feature for an owned published booking and for an owned draft booking associated to a flight that has already departed. Check that an authorization exception is thrown.
  - Detection of bugs: Did not detect any bugs.

- Request the booking publish feature on a non-integer id ("XXX"), an empty id and an id that does not correspond to a booking. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.

- Hack all read-only attributes of the form (all attributes are read-only) by using the "DevTools". Check that the values introduced are ignored.
  - Detection of bugs: Did not detect any bugs.

2.2. Operations of customers on booking records

2.2.1 List

The safe cases implemented include:

- Go into the list for each of the users and each of the bookings and check every page to make sure they entries render correctly. This was done in Spanish and in English.
  - Detection of bugs: Did not detect any bugs.

The hacking cases implemented include:

- Request the booking record list feature from other realms, especially as any (not authenticated) and as administrator. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.

- Request the booking record list feature for a not-owned draft booking and a not-owned published booking. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.

- Request the booking-record list feature for a non-integer booking id ("XXX"), an empty id, and an id that does not correspond to a booking. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.

2.2.2 Show

The safe cases implemented include:

- Go to the list for each of the customer users and request the show feature for every booking record in them. This was done in Spanish and in English.
  - Detection of bugs: Did not detect any bugs.

The hacking cases implemented include:

- Request the booking record show feature from other realms, specially as any (not authenticated) and as administrator. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.
- Request the booking record show feature for a booking from another customer. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.
- Request the booking record show feature for a non-integer booking id ("XXX"), an empty id and an id that does not correspond to a booking. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.

2.2.3 Create

The safe cases implemented include:

- Submit an empty form to check that no exceptions were thrown and errors were reported in the correct attributes.
  - Detection of bugs: Did not detect any bugs.
- For each attribute on the form, leave the rest blank and try as many invalid data as possible, then try as many valid data variations as possible.

  - Detection of bugs: Did not detect any bugs.

- Submit a form with valid data to check that the booking-record is created properly.
  - Detection of bugs: Did not detect any bugs.

The hacking cases implemented include:

- Request the booking record create feature from other realms, especially as any (not authenticated) and as administrator. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.
- Request the booking record create feature for a booking that is not the the user's. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.
- Request the booking-record create feature for an owned booking that is already published. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.
- Change the value of navigation attributes in the form (passenger) using "DevTools". Changed the value to a non-numeric value ("XXX"), a non-existent passenger, a passenger in draft mode, and a passenger that is already in the booking. Check that the application returns an authorization exception.
  - Detection of bugs: Did not detect any bugs.

2.2.4 Delete

The safe cases implemented include:

- Delete multiple booking records. Check that they are properly deleted.
  - Detection of bugs: Did not detect any bugs.

The hacking cases implemented include:

- Request the booking record deletion feature from other realms, especially as any (not authenticated) and as administrator. Check that the application returns an authorization exception.
  - Detection of bugs: Did not detect any bugs.

- Request the booking record deletion feature for a not-owned booking record. Check that the application returns an authorization exception.
  - Detection of bugs: Did not detect any bugs.
- Request the booking record deletion feature on a booking record corresponding to a published booking. Check that the application returns an authorization exception.
  - Detection of bugs: Did not detect any bugs.
- Request the booking record deletion feature for a non-integer booking-record id ("XXX"), an empty id, and an id that does not correspond to a booking record. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.
- Hack all read-only attributes of the form (passenger) by using the "DevTools". Check that the values introduced are ignored.
  - Detection of bugs: Did not detect any bugs.

2.3. Operations of customers on passengers

2.3.1 List

The safe cases implemented include:

- Go into the list for each of the customers and check every page to make sure they entries render correctly. The work was done in Spanish and English and for both the all=true and all=false lists.
  - Detection of bugs: Did not detect any bugs.

The hacking cases implemented include:

- Request the passenger list feature from other realms, specially as any (not authenticated) and as administrator. Check that the application returns an authorization exception.
  - Detection of bugs: Did not detect any bugs.
- Request the list feature for a non-boolean "all" value and an empty "all" value. Check that an authorization exception was thrown.
  - Detection of bugs: Not using a boolean value caused an exception to be thrown that was not an authorization exception, which was not admissible.

2.3.2 Show

The safe cases implemented include:

- Go to the list for each of the customer users and request the show feature for every passenger in them. This was done in Spanish and in English.
  - Detection of bugs: Did not detect any bugs.

The hacking cases implemented include:

- Change the navigation attribute in the URL to an empty value, a non-integer value ("XXX") and an id that does not correspond to a passenger in the database. Check that an authorization exception was thrown.
  - Detection of bugs: No bugs were detected.
- Request show for a passenger from another customer. Check that an authorization exception was thrown.
  - Detection of bugs: No bugs were detected.
- Request show for a booking from a non-customer profile, especially any (not logged in) and administrator. Check that an authorization exception was thrown.
  - Detection of bugs: No bugs were detected.

2.3.3 Create

The safe cases implemented include:

- Submit an empty form to check that no exceptions were thrown and errors were reported in the correct attributes.
  - Detection of bugs: Did not detect any bugs.
- For each attribute on the form, leave the rest blank and try as many invalid data as possible, then try as many valid data variations as possible. The data variations were taken from the data provided in the "Sample-Data" file and generated using natural intelligence for strings matching a pattern (passport).
  - Detection of bugs: Did not detect any bugs.
- Submit a form with valid data to check that the passenger is created properly.
  - Detection of bugs: Did not detect any bugs.

The hacking cases implemented include:

- Request the passenger create feature from other realms, especially as any (not authenticated) and as administrator. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.

2.3.4 Update

The safe cases implemented include:

- Submit an empty form to check that no exceptions were thrown and errors were reported in the correct attributes.
  - Detection of bugs: Did not detect any bugs.
- For each attribute on the form, leave the rest blank and try as many invalid data as possible, then try as many valid data variations as possible. The data variations were taken from the data provided in the "Sample-Data" file and generated using natural intelligence for strings matching a pattern (passport).
  - Detection of bugs: Did not detect any bugs.
- Submit a form with valid data to check that the passenger is updated properly.
  - Detection of bugs: Did not detect any bugs.

The hacking cases implemented include:

- Request the passenger update feature from other realms, especially as any (not authenticated) and as administrator. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.
- Request the passenger update feature on an owned published passenger. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.
- Request the passenger update feature on a not-owned draft passenger and a not-owned published passenger. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.
- Request the passenger update feature on a non-integer id ("XXX"), an empty id, and an id that does not correspond to a passenger. Check that an authorization exception was thrown.

  - Detection of bugs: Did not detect any bugs.

- Change the value of the read-only attributes in the form (draft mode) and submit the form. Check that the values introduced are ignored.
  - Detection of bugs: Did not detect any bugs.

2.3.5 Publish

The safe cases implemented include:

- Published a passenger in draft mode. Check that the passenger was published correctly.
  - Detection of bugs: Did not detect any bugs.

The hacking cases implemented include:

- Request the passenger publish feature from other realms, especially as any (not authenticated) and as administrator. Check that an authorization exception is thrown.
  - Detection of bugs: Did not detect any bugs.
- Request the passenger publish feature for a not-owned passenger in draft mode and for a not-owned published passenger. Check that an authorization exception is thrown.
  - Detection of bugs: Did not detect any bugs.
- Request the passenger publish feature for an owned published passenger. Check that an authorization exception is thrown.
  - Detection of bugs: Did not detect any bugs.
- Request the passenger publish feature on a non-integer id ("XXX"), an empty id and an id that does not correspond to a booking. Check that an authorization exception was thrown.
  - Detection of bugs: Did not detect any bugs.
- Hack all read-only attributes of the form (all attributes are read-only) by using the "DevTools". Check that the values introduced are ignored.
  - Detection of bugs: Did not detect any bugs.

2.4. Operations of customers on recommendations

2.4.1 List

The safe cases implemented include:

- Go into the list for each of the customers and check every page to make sure they entries render correctly. This was done in Spanish and in English.
  - Detection of bugs: Did not detect any bugs.

The hacking cases implemented include:

- Request the recommendation list feature from other realms, specially as any (not authenticated) and as administrator. Check that the application returns an authorization exception.
  - Detection of bugs: Did not detect any bugs.

2.4.2 Show

The safe cases implemented include:

- Go to the list for each of the customer users and request the show feature for every recommendation in them. This was done in Spanish and English.
  - Detection of bugs: Did not detect any bugs.

The hacking cases implemented include:

- Change the navigation attribute in the URL to an empty value, a non-integer value ("XXX") and an id that does not correspond to a recommendation in the database. Check that an authorization exception was thrown.
  - Detection of bugs: No bugs were detected.
- Request recommendation that does not match any of the current customer's bookings city and country of destination. Check that an authorization exception was thrown.
  - Detection of bugs: No bugs were detected.
- Request show for a booking from a non-customer profile, especially any (not logged in) and administrator. Check that an authorization exception was thrown.
  - Detection of bugs: 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 two computers performs better.

To collect the data required to compute the mean confidence interval and perform the hypothesis contrast, we will run all functional tests for customer features. We will be using the following devices:

- HP Victus: 32GB RAM and 1TB of memory.
- Lenovo Legion: 16GB RAM 1TB of memory.

3.1. Mean confidence interval

After treating the data obtained from the *"tester.trace"* file, I have performed an 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.

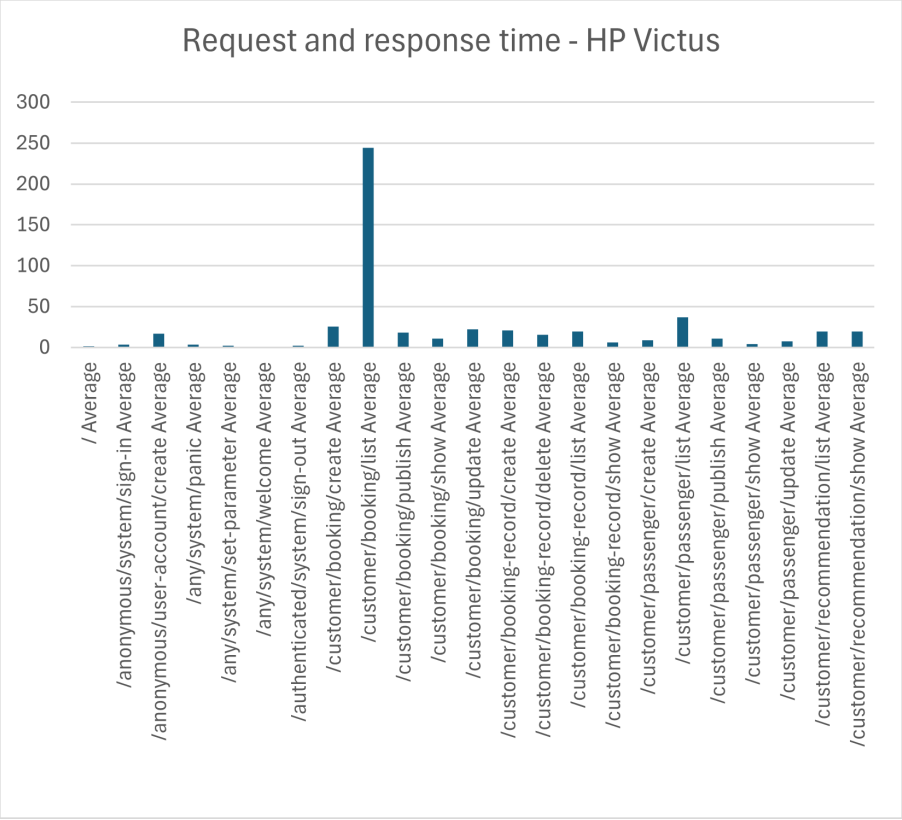

Figure 1: Graph showing average execution time for each request in the HP Victus PC

For the HP Victus PC we obtained a grand average of 30,1 ms. As we can see in the graph1, the MIR (Most Inefficient Request) is the booking listing, whose average stands at 244,35 ms, much higher than the rest of the requests.

Using the Excel data analyzer, we obtain the amplitude of the confidence interval at 95%, 2,6 ms. By removing this amount from the average and adding it, we obtain the confidence interval: [27,5 ms - 32,703 ms] or [0,0275 s - 0,032703 s].

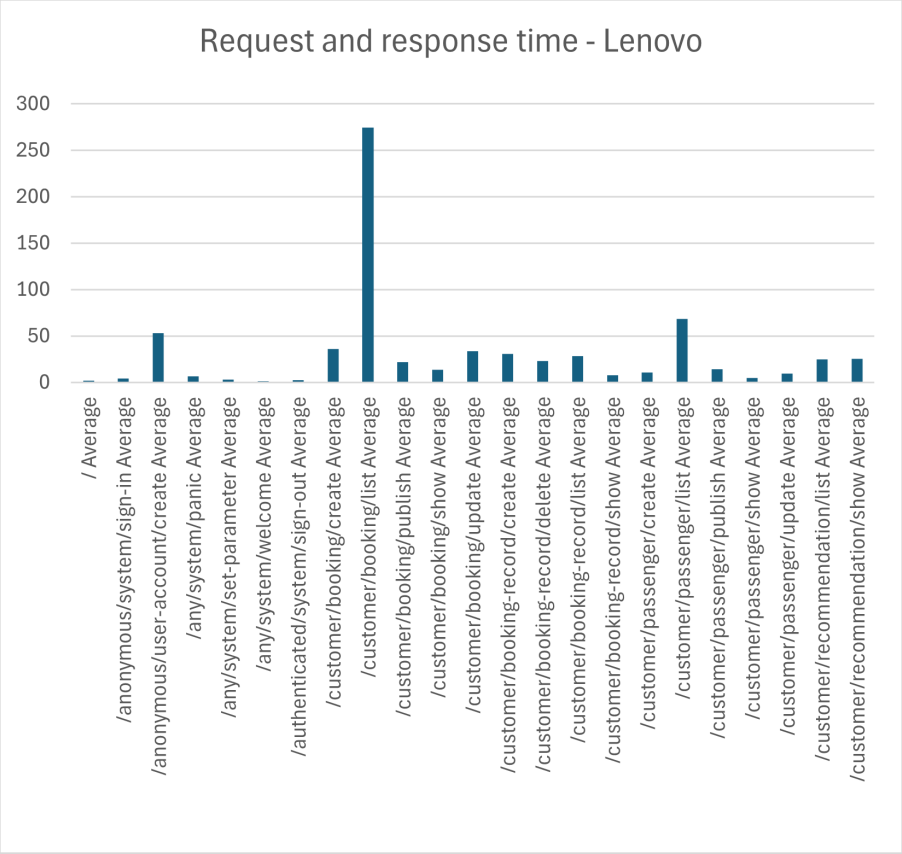

Figure 2: Graph showing average execution time for each request in the Lenovo PC

On the other hand, when doing the same with the Lenovo PC we obtained a graph 2 with a similar shape. In this case, the grand average stands at 37,67 ms, higher than for the first laptop. Once again, the MIR is the bookings listing, with an average of 274,53 ms.

Using the Excel data analyzer, we obtain the amplitude of the confidence interval at 95%, 2,98 ms. By removing this amount from the average and adding it, we obtain the confidence interval: [34,68 ms - 40,64 ms] or [0,03468 s - 0,040643 s].

3.2. Hypothesis contrast

Once we have processed the data for both computers, we will compare 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 HP Victus 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 (one minus the confidence). We introduced the HP Victus data as the first one and the Lenovo data as the second one and obtained a p-value of 0,000181133. 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 time.

	Victus	Lenovo
Average:	30,09921806	37,65804371
Best?	TRUE	FALSE

Figure 3: Conclusion of the comparison of average wall time.

As we can see in the above image 3, the laptop with the best performance is the HP Victus laptop, as it has a lower average wall time, which confirms our initial hypothesis.

4. Conclusions

This document serves as a means of reporting the test cases performed for future reference. The analysis recorded in this document provides a detailed evaluation of the test cases in order to identify possible bugs in the code, evaluate potential solutions, and make informed decisions to improve the quality of the code base and ensure protection against security breaches.

Each test case was recorded in a structured manner, grouped by feature, and reporting on the bugs that were detected thanks to this test case. This report serves as a transparent reference for the testing process.

Through the analysis of wall time across functional tests, we have evaluated the performance of the project on two different devices. The HP Victus consistently demonstrated lower times, with a lower grand average of 30,1 ms compared to the 37,67 ms of the Lenovo Legion. As expected, both systems showed similar behavior in terms of request distribution, with the booking listing being the MIR in both cases.

The testing report provides a clear explanation for the tests performed. In the future, these tests will serve as a solid foundation for continuous improvement and quality assurance.

References

[1] *02 - Requirements – Student #2*, Enseñanza Virtual, 2025. Available on Enseñanza Virtual in the course inside the Project Statement folder.