Testing Report – Student #2

* **Group Number**: C1.034
* **Repository**: https://github.com/CarlosCerdaMorales/dp2-c1-034
* **Workgroup Members** **and Corporate Emails:**

Loubna Founoun Elaoud: [loufouela@alum.us.es](mailto:loufouela@alum.us.es)

Carlos Cerdá Morales: [carcermor@alum.us.es](mailto:carcermor@alum.us.es)

José Luis Moraza Vergara: [josmorver@alum.us.es](mailto:josmorver@alum.us.es)

Claudia Oviedo Govantes: [claovigov@alum.us.es](mailto:claovigov@alum.us.es)

Sergio Cantillo Blanco: [sercanbla@alum.us.es](mailto:sercanbla@alum.us.es)

* **Date**: Sunday 25th, May 2025

**Table Of Contents**

[1. Executive Summary 3](#_451ewtno8qvm)

[2. Revision Table 3](#_ov3mb6vwut6t)

[3. Introduction 3](#_r2bem3ub6oc7)

[4. Contents 3](#_nh08ol2sdh4r)

[4.1 Título 1 3](#_h69on2nqubkm)

[4.2 Título 2 3](#_xn4c25wc84mn)

[5. Conclusions 3](#_2l2wgk3amr06)

[6. Bibliography 3](#_a255vkbpqjez)

# Executive Summary

In this report, I present the functional and performance testing I carried out on my student. The objective was to ensure that all features behave as expected and to assess how quickly the system responds under normal conditions.

For the functional testing, I organized the test cases by feature. Each case targets a specific functionality and was essential in verifying that the application behaves correctly.

For the performance testing, I followed the methodology outlined in the session guide: I collected execution times from the .trace files and processed the data using Excel. I generated charts and calculated 95% confidence intervals to evaluate whether the system’s response times remain within acceptable limits. The tests were executed in two different configurations: one using the database without any additional indexing, and another with relevant indexes applied. I then performed a statistical comparison between both setups to determine the impact of indexing on performance.

In summary, this report reflects the testing I have conducted on the key features of the application, supported by performance data that provides a solid understanding of how the system behaves in real-world conditions.

# Revision Table

|  |  |  |
| --- | --- | --- |
| **Revision Number** | **Date** | **Description** |
| 1.0 | 25/05/2025 | Final version |

# Introduction

The purpose of this report is to document the testing I carried out on my project, covering both functionality and performance. My goal was to ensure that each feature works as intended and that the system can handle requests efficiently.

To achieve this, I first ran a set of functional tests grouped by feature, verifying that the application behaved as expected. Then, I focused on performance testing by analyzing how long the system took to respond to various requests. This helped me identify potential bottlenecks and compare performance across different configurations.

Overall, this testing phase gave me a clearer understanding of how the system behaves under real-world conditions.

# Contents

## 4.1 Functional testing

For these tests, I evaluated the functionalities that my project, Acme-ANS, provides to customers, specifically regarding their bookings and passengers. The tests are documented through trace files located in the /src/test/resources/customer/booking and /src/test/resources/customer/passenger directories. All test cases achieved 100% coverage, as shown in the image below.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto. These are the trace files that demonstrate the code coverage, included just below, and created following the recommendations provided in the course. I tested both .safe traces—covering positive and negative test cases—and .hack traces, which simulate attempts to hack the application. All hacking attempts were correctly handled and resulted in an "Access Unauthorised" response.

In the following sections, the specific test cases are specified and how each was addressed.

Tabla

El contenido generado por IA puede ser incorrecto.

|  |  |
| --- | --- |
| Customer Booking features | |
| CREATE SERVICE SAFE | |
| TC-01: Create an empty booking | An attempt was made to create an empty booking, which correctly triggered error messages from the application, indicating that certain fields "must not be null". |
| TC-02: INVALID LOCATOR CODE | An invalid locator code was submitted when creating a booking. The code did not match the expected pattern and violated both lower and upper boundary constraints. |
| TC-03: REPEATED LOCATOR CODE | A locator code that already existed in the database was used in a booking creation attempt. The system properly rejected the request, as duplicate locator codes are not allowed. |
| TC-04: VALID LOCATOR CODE | A locator code that complied with the required pattern was tested and was correctly accepted by the system. |
| TC-05: INVALID TRAVEL CLASS | An attempt was made to create a booking with a null travel class, which the system correctly rejected as invalid input. |
| TC-06: VALID TRAVEL CLASS | The same test was conducted using the two available travel class values, business and economy. Both were accepted successfully by the system. |
| TC-07: INVALID LAST NIBBLE | Attempts were made to store the lastNibble field with values breaking the pattern by exceeding upper and lower limits, as well as including letters and non-numeric characters. These attempts were rejected by the database. |
| TC-08: VALID LAST NIBBLE | A lastNibble value conforming exactly to the required 4-digit pattern was tested and correctly accepted by the application. |
| TC-09: INVALID FLIGHT | An attempt was made to create a booking without assigning any flight, resulting in an error indicating that the flight field must not be null. |
| TC-10: VALID FLIGHT | An attempt was made to assign a valid flight not in draft mode within the application, which functioned correctly as expected. |
| BUGS | No bugs were found while recording these tests. |
| CREATE SERVICE HACK | |
| TC-11: INVALID ID FOR CREATE | An attempt was made to modify the id field of the new booking being created, aiming to overwrite an existing booking in the database instead of creating a new one. This attempt resulted in an "Access Unauthorised" response from the application. |
| TC-12: INVALID TRAVEL CLASS | Attempts were made to modify the travelClass field with values not accepted by the enumeration, including numbers, empty strings, and other invalid entries. All these attempts resulted in an "Access Unauthorised" response from the application. |
| TC-13: INVALID FLIGHT | An attempt was made to assign an unpublished flight to the booking, resulting in an "Access Unauthorised" response from the application. The same protocol and handling were applied when attempting to assign a flight that does not exist in the database. |
| BUGS | Bugs were encountered when attempting to fully cover TC-12, as some values received from the view could not be entirely tested. Eventually, complete coverage was achieved, and unauthorized access is now properly handled. |
| DELETE SERVICE SAFE | |
| TC-14: DELETE BOOKING | An attempt was made to delete a booking. Since the only restriction is that it must be the user’s own booking, the deletion was performed successfully after verifying this condition. |
| BUGS | No bugs were found while recording these tests. |
| DELETE SERVICE HACK | |
| TC-15: DELETE NON-EXISTENT BOOKING | An attempt was made to delete a non-existent booking by accessing the /delete endpoint with an ID that does not exist in the database. The application correctly handled the request and responded with an "Access Unauthorised" error. |
| TC-16: DELETE BOOKING NOT MINE | An attempt was made to delete a booking that did not belong to the current user by accessing the /delete endpoint with a valid ID. The system correctly identified the unauthorized action and responded with an "Access Unauthorised" error. |
| TC-17: DELETE PUBLISHED BOOKING | An attempt was made to delete a booking that belonged to the current user but was already published. The application correctly rejected the operation, responding with an "Access Unauthorised" error, as published bookings cannot be deleted. |
| BUGS | No bugs were found while recording these tests. |
| LIST SERVICE SAFE | |
| TC-18: LIST MY BOOKINGS | An attempt was made to list the bookings of a customer. This operation was always successful, as the database query is executed based on the ID of the currently logged-in user. |
| BUGS | No bugs were found while recording these tests. |
| LIST SERVICE HACK | |
| THERE IS NO hacking attempt applicable to this specific FEATURE. | |
| PUBLISH SERVICE SAFE | |
| TC-19: PUBLISH an empty booking | An attempt was made to publish an empty booking, which correctly triggered error messages from the application, indicating that certain fields "must not be null". |
| TC-20: INVALID LOCATOR CODE | An invalid locator code was submitted when publishing a booking. The code did not match the expected pattern and violated both lower and upper boundary constraints. |
| TC-21: REPEATED LOCATOR CODE | A locator code that already existed in the database was used in a booking publishing attempt. The system properly rejected the request, as duplicate locator codes are not allowed. |
| TC-22: VALID LOCATOR CODE | A locator code that complied with the required pattern was tested and was correctly accepted by the system. |
| TC-23: INVALID TRAVEL CLASS | An attempt was made to publish a booking with a null travel class, which the system correctly rejected as invalid input. |
| TC-24: VALID TRAVEL CLASS | The same test was conducted using the two available travel class values, business and economy. Both were accepted successfully by the system. |
| TC-25: INVALID LAST NIBBLE | Attempts were made to store the lastNibble field with values breaking the pattern by exceeding upper and lower limits, as well as including letters and non-numeric characters. These attempts were rejected by the database. |
| TC-26: VALID LAST NIBBLE | A lastNibble value conforming exactly to the required 4-digit pattern was tested and correctly accepted by the application. |
| TC-27: EMPTY LAST NIBBLE | An attempt was made to publish a booking without the lastNibble being stored. This resulted in an error, as the requirements state that only bookings with a properly stored lastNibble can be published. |
| TC-28: INVALID FLIGHT | An attempt was made to create a booking without assigning any flight, resulting in an error indicating that the flight field must not be null. |
| TC-29: VALID FLIGHT | An attempt was made to assign a valid flight not in draft mode within the application, which functioned correctly as expected. |
| TC-30: EMPTY FLIGHT | An attempt was made to publish a booking with no flight assigned. This resulted in a controlled error triggered by the service’s validator. |
| TC-31: NO PASSENGERS | An attempt was made to publish a booking without any passengers assigned. This resulted in a controlled error triggered by the service’s validator. |
| TC-32: VALID PUBLISH | Finally, a booking was successfully published after ensuring that all required fields were valid and properly completed, and all restrictions were checked. |
| BUGS | Bugs were encountered when attempting TC-30. A validation error occurred during the price calculation process, as the flight field was null. |
| PUBLISH SERVICE hack | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## 4.2 Título 2

Texto

# Conclusions

It is expected to be 200-word long in most cases.

# Bibliography

If there’s no relevant bibliography, write “intentionally blank”.