Testing Report

* **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**: Saturday 24th, May 2025

**Table Of Contents**

[1. Executive Summary 3](#_Toc199006209)

[Revision Table 3](#_Toc199006210)

[2. Introduction 3](#_Toc199006211)

[3. Contents 4](#_Toc199006212)

[3.1 Functional testing 4](#_Toc199006213)

[3.2 Performance testing 6](#_Toc199006214)

[4. Conclusions 8](#_Toc199006215)

[5. Bibliography 8](#_Toc199006218)

# Executive Summary

In this report, we go over the functional and performance testing we did on our project. The goal was to make sure everything works as expected and to see how fast the system can respond under normal conditions.

For the **functional part**, we grouped our test cases by feature. Each one checks a specific piece of functionality and helped us confirm that the app behaves correctly.

For the **performance part**, we followed the steps from the session guide: we collected the execution times of our requests using the “.trace” files and processed everything in Excel. We generated charts and calculated confidence intervals to see if our app’s speed stays within acceptable limits. We ran the tests on two different computers and compared the results statistically to see which one performs better.

In short, this report shows that we tested the main features thoroughly, and we’ve got performance data that gives a pretty good idea of how the system behaves in real time.

# Revision Table

|  |  |  |
| --- | --- | --- |
| **Revision Number** | **Date** | **Description** |
| 1.0 | 24 May 2025 | Final draft |

# Introduction

The purpose of this report is to document the testing we carried out on our project, covering both functionality and performance. We wanted to check that each feature works as intended and that the system can handle requests efficiently.

To do this, we first ran a set of functional tests grouped by feature, making sure everything behaved correctly. Then, we focused on performance testing by analyzing how long it took the system to respond to different requests. This helped us spot potential slowdowns and compare performance across different machines.

Overall, this testing phase gave us a clearer view of the system’s behavior

# Contents

## 3.1 Functional testing

To carry out these tests, the various features that Acme-ANS provides to administrators in relation to airport management have been thoroughly evaluated. All test executions are documented in the traces located in the *‘src/test/resources/administrator/airport’* directory, with 100% coverage achieved across all the cases listed below.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Texto, Carta

El contenido generado por IA puede ser incorrecto. Test result Traces

**NOTES**: All value variations were derived from the *'Sample Data'* spreadsheet provided as part of the testing resources. Likewise, all anti-hacking tests consistently triggered the expected **"Access Unauthorised"** response, confirming that the access control mechanisms are properly enforced.

- Airport creation. To validate this functionality, the following testing strategies were applied:

1. Submitting an empty form, ensuring that no unintended errors were triggered and that all relevant validation messages were correctly displayed.

2. Individually testing each field, exploring valid and invalid values, as well as edge cases and boundary conditions.

· **String airportName**: Tested for upper and lower boundary lengths, injection attacks, and the use of non-Latin character sets.   
 · **String IATACode**: Evaluated with valid IATA formats, excessively short and long strings, lowercase letters, numeric characters, and duplicate codes.   
 · **Enum operationalScope**: Assessed with all possible enumeration values and invalid inputs such as null, empty strings, and unrelated values.   
 · **String city**: Tested for upper and lower boundary lengths, injection attacks, and the use of non-Latin character sets.   
 · **String** **country**: Tested for upper and lower boundary lengths, injection attacks, and the use of non-Latin character sets.   
 · **String website**: Tested with valid and malformed URLs.   
 · **String email**: Checked against valid and invalid email formats.   
 · **String contactPhoneNumber**: Evaluated with valid formats, overly short and long inputs, and alphanumeric combinations.

3. Attemping to tamper with the creation form using various hacking techniques to assess the robustness of input validation and security controls. To test potential attack vectors:

· **ID Tampering**: Attempts were made to manipulate the entity's ID in the form submission, forcing it to match an existing database entry in order to overwrite it rather than create a new one.   
 · **Enum Injection**: Malicious values were injected into the operationalScope field, including values outside the declared enumeration, null, empty strings, and numeric types.

- Airport listing. This functionality was tested by simply attempting to retrieve and display all airports stored in the database. Since the listing process is entirely managed by the underlying framework, it does not involve any complex logic nor expose any potential security vulnerabilities. Consequently, no hacking attempts were applicable or required for this feature.

- Airport details (show). The "show" functionality was tested by retrieving a single airport record and verifying the correct mapping and data binding from the database to the view. No additional functional tests were required, as the retrieval logic is straightforward and relies on the framework’s default behavior. However, several anti-hacking tests were conducted to assess the system’s resilience against improper access attempts:

· **Omitting the ?id=x** parameter from the URL resulted in an unauthorized access error.

· **URL Manipulation**: Manually modifying the id parameter in the URL to reference a non-existent airport. This correctly resulted in an **"Access Unauthorised"** response, preventing unauthorized access or potential data leaks.

- Airport update. To validate this functionality, the following testing strategies were applied:

1. Individually testing each field, exploring valid and invalid values, as well as edge cases and boundary conditions.

· **String airportName**: Tested for upper and lower boundary lengths, injection attacks, and the use of non-Latin character sets.   
 · **String IATACode**: Evaluated with valid IATA formats, excessively short and long strings, lowercase letters, numeric characters, and duplicate codes.   
 · **Enum operationalScope**: Assessed with all possible enumeration values and invalid inputs such as null, empty strings, and unrelated values.   
 · **String city**: Tested for upper and lower boundary lengths, injection attacks, and the use of non-Latin character sets.   
 · **String** **country**: Tested for upper and lower boundary lengths, injection attacks, and the use of non-Latin character sets.   
 · **String website**: Tested with valid and malformed URLs.   
 · **String email**: Checked against valid and invalid email formats.   
 · **String contactPhoneNumber**: Evaluated with valid formats, overly short and long inputs, and alphanumeric combinations.

2. Attemping to tamper with the update form using various hacking techniques to assess the robustness of input validation and security controls. To test potential attack vectors:

· **URL Manipulation**: Manually modifying the id parameter in the URL to reference a non-existent airport. This correctly resulted in an **"Access Unauthorised"** response, preventing unauthorized access or potential data leaks.

· **Enum Injection**: Malicious values were injected into the operationalScope field, including values outside the declared enumeration, null, empty strings, and numeric types.

## 3.2 Performance testing

We ran our functional tests on two different computers and collected response times. The following graphs show the obtained results.

Gráfico, Gráfico en cascada

Descripción generada automáticamente

Gráfico, Gráfico de barras

Descripción generada automáticamente

As shown in the charts, PC 1 consistently shows lower response times across all endpoints compared to PC 2. For example, the /*administrator/airport/update* request, which is one of the most demanding, takes around 42 ms on PC 2, while only around 36 ms on PC 1. Similarly, the */administrator/airport/show* take over 160 ms on PC 2, but remain below 20 ms on PC 1.

After cleaning and processing the data, we calculated the 95% confidence interval for both sets of results. These are the results:

**Confidence interval calc in PC1:**

Interfaz de usuario gráfica, Aplicación, Tabla

Descripción generada automáticamente

**Confidence interval calc in PC2:**

Interfaz de usuario gráfica, Aplicación, Tabla, Excel

Descripción generada automáticamente

To determine if there was a real performance difference, we configured and ran a Z-Test following the steps from the lecture. The test returned a p-value of 0.99456, which falls in the interval (0.05, 1.00]. This means that the differences between the two computers are not statistically significant—they're most likely due to random variation. These are the findings:

Tabla

Descripción generada automáticamente

Because of this, we didn’t continue with software or hardware profiling, as it wouldn’t lead to any meaningful improvements in performance.

In addittion, it was not necessary to add custom indexes to repository queries, as these operations are limited to simple findAll calls and lookups by primary key (id). Since the id field is automatically indexed by default, no additional indexing was required to ensure optimal performance.

# Conclusions

# We tested the main features of the Acme-ANS system thoroughly, covering all the important parts related to airport management. Our tests included checking normal use cases, edge cases, and trying different hacking attempts. The system handled everything as expected, valid inputs worked fine, and invalid or malicious inputs were blocked with proper “Access Unauthorised” responses. This shows the app is working well and has good security controls.

# On the performance side, PC 1 generally showed faster response times than PC 2 when looking at the raw data. However, statistical analysis revealed that the difference was not significant. The confidence intervals overlapped, and the Z-Test gave a high p-value (0.99456), indicating that the performance differences are most likely due to random variation rather than actual hardware differences. As a result, we decided not to proceed with profiling, since it would not lead to meaningful improvements.

Overall, the system works reliably, is secure, and performs similarly across different hardware environments.

# Bibliography

Intentionally blank.