Documentation Template

* **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 Cerda 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**: 25/05/2025

**Table Of Contents**

[1. Executive Summary 3](#_7vr8dm8obw4e)

[2. Revision Table 3](#_k3zitsyjd32s)

[3. Introduction 3](#_ntcc82giiwut)

[4. Contents 3](#_djssrbod0h7w)

[4.1 Título 1 3](#_mqdn0fx8pyzb)

[4.2 Título 2 3](#_o33kx9hpwvqb)

[5. Conclusions 3](#_lvumujzd1b06)

[6. Bibliography 3](#_6cht5p8bjj88)

# 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 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 and further optimization by applying the course content about software and hardware profiling. I then made a statistical comparison between both setups to determine the impact of indexing on performance.

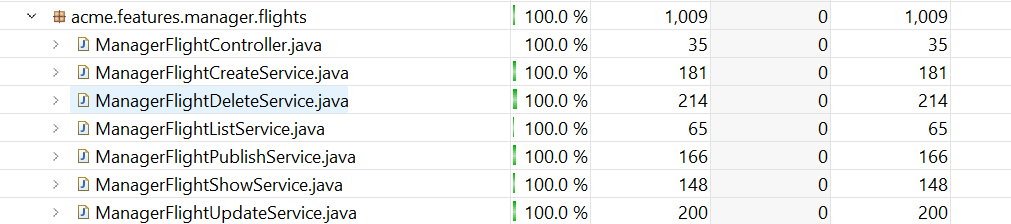
# Revision Table

|  |  |  |
| --- | --- | --- |
| **Revision Number** | **Date** | **Description** |
| 1.0 | [25/05/2025] | Initial version |

# Contents

The following is a list of functional tests performed for the features Flight:

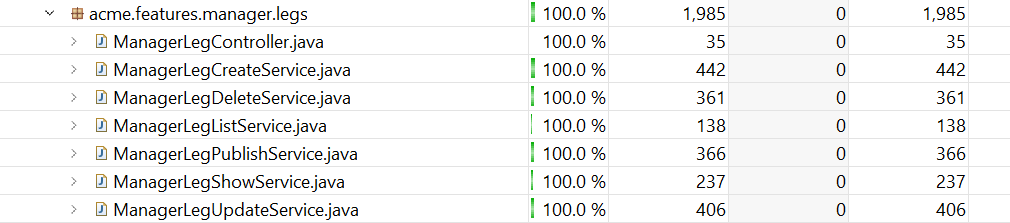
* List.safe: Verifies the ability of a manager to list their flights. The test confirms that all flights associated with manager1 are correctly retrieved and displayed.
* show.safe: Ensures the correct display of flight details belonging to manager1, verifying that individual flight information is properly rendered.
* Create.safe: Tests the creation of flights using valid input data. The system's validation mechanisms are evaluated by attempting to create a flight with empty fields, expecting appropriate error messages. Further tests introduce partially valid data combined with invalid entries, such as excessively long strings or incorrect price values.
* Update.safe: Data from several flights belonging to manager1 is updated using both valid inputs accepted by the system and invalid inputs, similarly to the tests in create.safe.
* Delete.safe: Confirms that manager1 can delete their flights and that associated entities are also removed. The system is tested to ensure that published flights cannot be deleted.
* Publish.safe: Assesses the ability to publish flights. Successfully publishes valid flights and verifies that flights lacking legs or containing unpublished legs cannot be published, with appropriate error messages displayed.
* show.hack: Tests unauthorized access attempts by injecting invalid or unauthorized IDs into the URL, including flights that either do not exist or belong to other managers. Attempts are made to access unpublished flights while logged in as a different manager.
* Create.hack: Simulates misuse of browser developer tools by manually altering the ID of a flight during the creation process to an existing one, potentially resulting in unauthorized updates to published or non-owned flights.
* Update.hack: Includes POST-based attack simulations by altering request payloads and manipulating URLs to attempt updates on non-existing or published flights.
* Delete.hack: Verifies the system’s protection against unauthorized deletions by modifying request parameters and attempting to delete published or non-existent flights.
* Publish.hack: Confirms that already published flights cannot be re-published and that publishing attempts with invalid or unauthorized IDs are appropriately rejected.

The test coverage achieved for the entity Flight is 100%.   
  
All lines of this entity and its features have been fully tested.

The following is a list of functional tests performed for the features of leg:

* List.safe: Verifies that a manager can retrieve and list all their associated leg records. This test confirms that legs linked to manager 1 are correctly fetched and displayed.
* List.show: Ensures the correct display of leg details belonging to manager1, verifying that individual leg information is properly rendered.
* Create.safe: Tests the creation of legs using valid input data accepted by the system. It also includes attempts to create a leg with no input (expecting appropriate validation error messages). Additional scenarios test the system’s response to invalid inputs such as creating a leg with an invalid date format, attempting to use a duplicate flight number, etc. Each of these cases is expected to return relevant and informative error responses.
* Update.safe: Data from several legs belonging to manager1 is updated using both valid inputs accepted by the system and invalid inputs, similarly to the tests in create.safe.
* Delete.safe: Confirms that manager1 can successfully delete selected leg records. Verifies the correct removal of the targeted entries.
* Publish.safe: Tests the publication of previously unpublished legs. Ensures that eligible legs are correctly transitioned to the published state.
* List.hack: GET hacking tests are performed by accessing URLs associated with managers from other users and from a non-logged in user. Attempts are also made to access legs lists of unpublished flights of other managers while logged in as one of them.
* show.hack: Simulates client-side manipulation using browser developer tools (e.g., F12) to alter relationship IDs (such as aircraft or airports) during the creation of a leg. These tests confirm that the system detects and blocks illegitimate attempts, returning appropriate error messages.
* Create.hack: Simulates client-side manipulation using browser developer tools (e.g., F12) to alter relationship IDs (such as aircraft or airports) during the creation of a leg. These tests confirm that the system detects and blocks illegitimate attempts, returning appropriate error messages.
* update.hack:Like create.hack, this test involves POST requests with modified relationship IDs using developer tools. Attempts to update unauthorized or non-existent legs should be prevented and met with proper validation responses.
* delete.hack: Attempts unauthorized deletion of leg records by manipulating request parameters or relationship IDs. Tests are conducted while logged in as an unauthorized user or when not logged in at all. The system should prevent such deletions.
* published.hack:Simulates publishing attempts using modified relationship IDs or targeting unauthorized legs.

The test coverage achieved for the features of the leg is 100%.

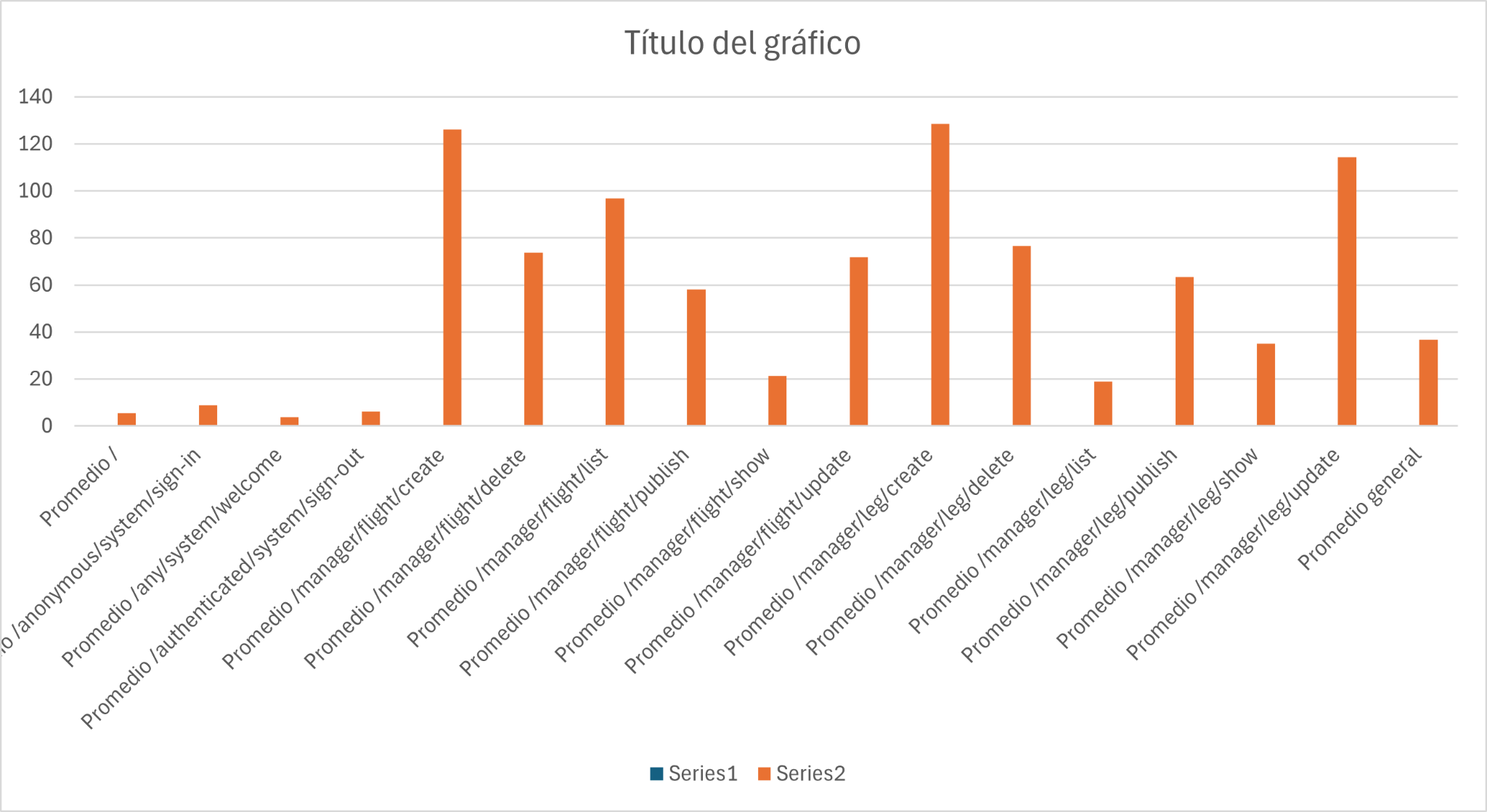


All lines of this entity and its features have been fully tested.

## Performance Testing

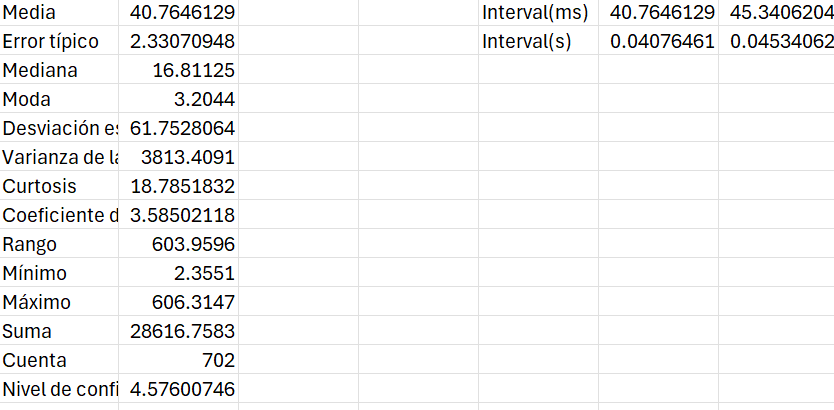
A performance analysis of the system will now be carried out through the execution of the previously mentioned functional tests. The tests have been carried out under two different scenarios:

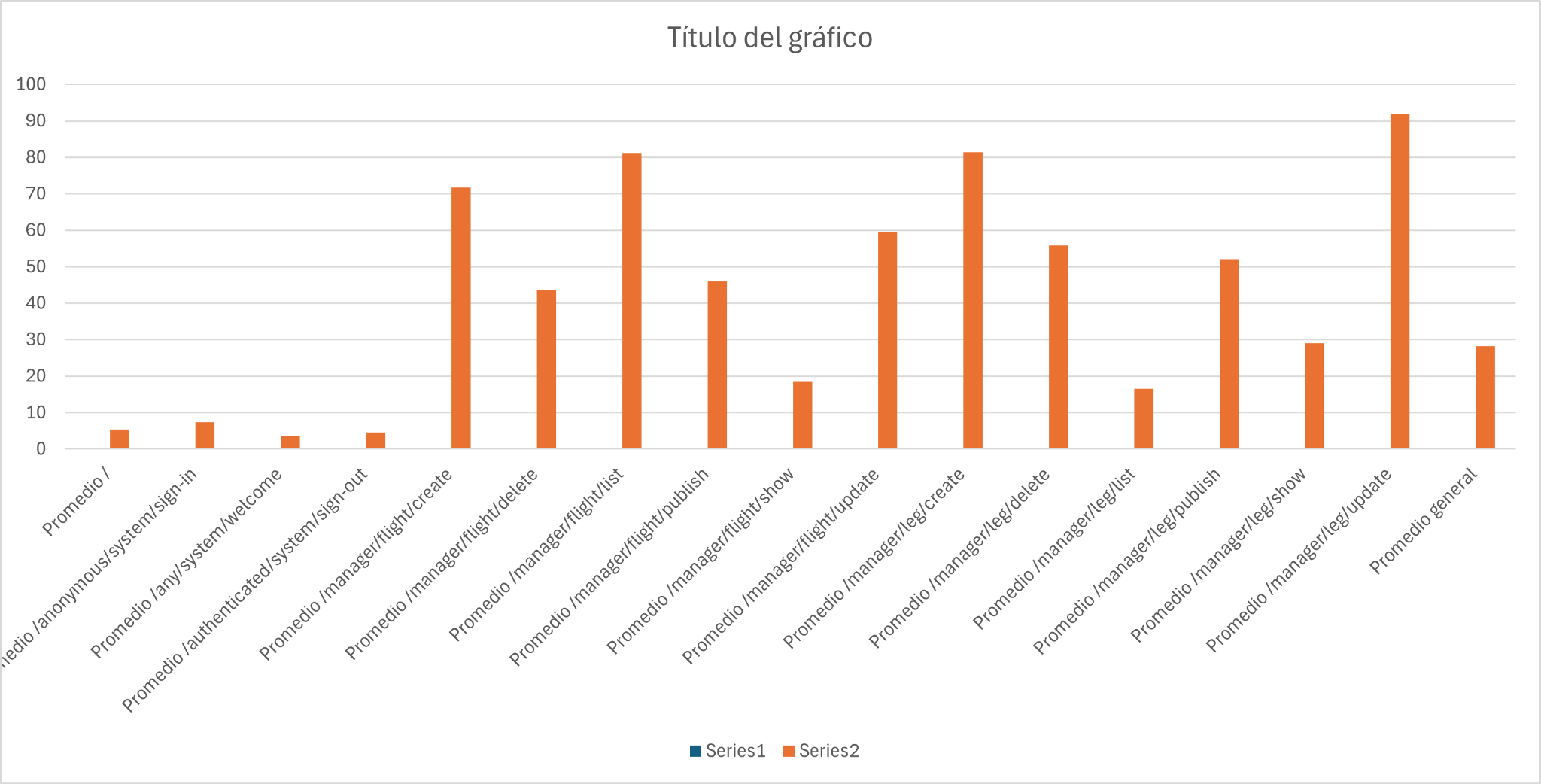
- Without indexes for query optimization: The Excel file “tester-performance-clean.xlsx” contains the average performance results of the test operations as illustrated in the following chart.



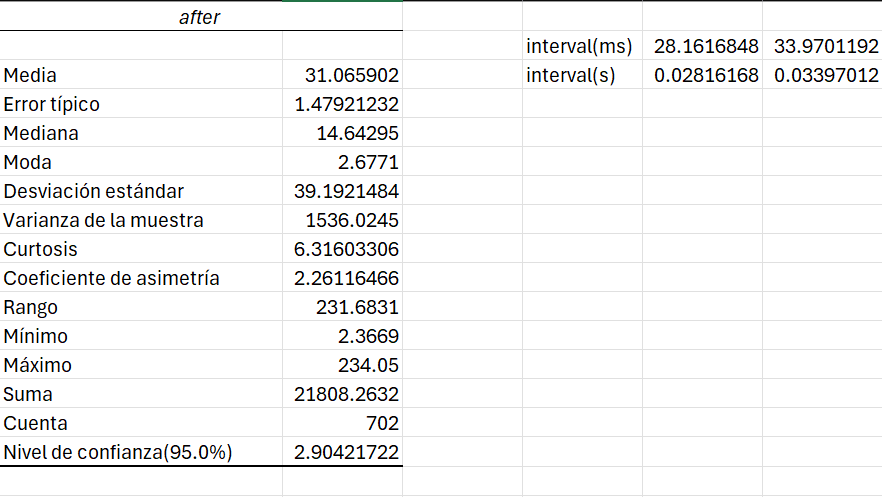
As observed, the most time-consuming operation on average is the create function of legs and flights, which takes nearly 130 milliseconds.

Below are some statistics regarding the operations:



* With indexes for query optimization and software and hardware profiling :The Excel file “tester-performance-clean-indexes.xlsx” contains the average performance results of the test operations, as illustrated in the following chart.  
  

There’s been a noticeable performance improvement, thanks to adding a few well-placed indices, since most of my queries are based on single IDs. But mainly this boost came from applying techniques covered in the course material on software and hardware profiling, which helped identify where things could be optimized therefore achieving much greater performance.



## 

The pvalue obtained from the z-test is 0.0012083. This value is much lower than alpha, which in this case is 0.05. We can conclude that the changes are significant in terms of system performance.

# Conclusions

More than 100% test coverage was achieved for the code related to the flight and leg features developed by Student 1. This testing helped uncover and fix hidden issues, including unused code. Additionally, statistical analysis confirmed that the implemented features meet the non-functional requirement: on average, operations on the leg and flight entities complete in under one second.

# Bibliography

intentionally blank.