Universidad de Sevilla  
Escuela Técnica Superior de Ingeniería Informática

**Testing report D04**



Degree in Computer Engineering – Software Engineering  
Design and Testing II.

Academic year 2024 – 2025

|  |  |  |
| --- | --- | --- |
| **Laboratory group C2.005** | | |
| **Alphabetised authors** | **Role** | **Role description** |
| González Benito, Claudio – clagonben@alum.us.es | Manager  Developer  Tester | Makes plans, creates and supervises tasks, initialises the repository and writes reports  Creates development configuration, customises the starter, implements features, does informal testing and writes reports  Does formal testing and writes reports |
| Ramos Vargas, Alba – albramvar1@alum.us.es | Operator  Developer  Tester | Creates deployment configurations, deploys the application, keeps the application running and writes reports  Creates development configuration, customises the starter, implements features, does informal testing and writes reports  Does formal testing and writes reports |

# Versioning

|  |  |  |
| --- | --- | --- |
| Date | Version | Description |
| 10/05/2025 | v1.0.0 | Creation of the report |
| 26/06/2025 | V1.0.1 | Updated coverage |
| 02/07/2025 | V2.0.0 | Updated version due to new call |

Table of contents

[Versioning 2](#_Toc199173048)

[Executive Summary 3](#_Toc199173049)

[Introduction 3](#_Toc199173050)

[Functional testing 3](#_Toc199173051)

[Specific Test cases 5](#_Toc199173052)

[Flight Assignment 5](#_Toc199173053)

[Activity Log 9](#_Toc199173054)

[Coverage 12](#_Toc199173055)

[Performance testing 13](#_Toc199173056)

[First study case: charging vs not charging 14](#_Toc199173057)

[Second study case: no indexes vs indexes 15](#_Toc199173058)

[Conclusions 16](#_Toc199173059)

[Bibliography 17](#_Toc199173060)

# Executive Summary

The purpose of this report is to study the testing done on the project, specifying test cases, their expected and real results, the coverage gotten and the comparison of performance once indexes were added.

Due to the formatting decisions of this report, a PDF version will be added to ensure that the report is readable.

# Introduction

In this report we will specify the test cases produced, that is, list, show, create, update, delete and publish for both feature modules, flight assignment (usually referred to as simply ‘assignment’) and activity log (usually referred to as ‘log’).

I will expose what each test case is supposed to test in general and in particular, specifying validation constraints and expected exception. In addition, the coverage provided by these tests will be discussed.

Finally, the second chapter of this report will study the performance, considering three distinct cases that were considering worth studying.

# Functional testing

The generic cases that will be considered are:

* List: the user must receive a list of the entity in question.
* Show: the user must be shown all relevant information of the entity, this includes all properties and a singular property of any linked entity that’s considered the most representative. In this case, the property chosen of legs is their flight code and the property of crew members are their identifier code, as it is an alternative primary key for both entities.
* Create: the user must be able to create an entity following some constraints.
* Update: the user must be able to update any preexisting entity following the same constraints as the create proceeding.
* Delete: the user must be able to delete the entity if the requirements allow it.
* Publish: the user must be able to publish the entity if the requirements allow it.

In addition, there will be two kind of tests considered, ‘safe’ cases and ‘hack’ cases. The former comprise only ‘legal’ actions and cannot ever get any exception; the latter’s purpose is to get exception, specifically 500 Access not authorized exception, in their majority.

## Specific Test cases

## Flight Assignment

|  |  |  |  |
| --- | --- | --- | --- |
| Test case |  | Expected result | Bug detection |
| List | **SAFE** | Show three different lists:   * All assignments * Future assignments * Past assignments   Future and past assignments will be classified considering the leg’s scheduled departure and arrival.  For users registered as lead attendants of a specific leg, all assignments of that leg will be shown, regardless of the crew member assigned.  For users not registered as lead attendants, only their assignment will be shown. | No bugs detected |
| **HACK** | When trying to access the list by another realm, there should be a not authorized exception.  Seeing the list of another user can never be done as the system uses the user’s id to give the list of assignments, the user does not provide any hackable information other than the user account, what is out of our purvey. | No bugs detected |
| Show | **SAFE** | Show a user’s assignment, accessing it through the lists specified above.  Lead attendants are shown all assignments of the leg they are in charge of and can access the edit buttons (delete, update and publish).  Not lead attendants will be shown only their assignment and will have no buttons to interact with, as they are not authorized to do so. | No bugs detected |
| **HACK** | The system should provide a 500 Access not authorized exception in the following cases:   * A user tries to access an assignment that is not theirs * A user tries to access an object that is not theirs, regardless of the class * A user does not specify an id to be shown * A user tries to be shown an object with an id not compatible (i.e. a string id) | No bugs detected |
| Create | **SAFE** | A user can create an assignment in these two specific cases:   * The user is the lead attendant of the leg given * The user is creating an assignment with a leg with no assignments and a lead attendant duty[[1]](#footnote-1)   The validation considered should be:   * Only the remarks value can be null * There can be only one lead attendant, pilot and co-pilot by leg * The last update should be a past moment before the leg’s scheduled departure * The leg given must be from the same airline the user is an employee of * The assignee must be from the same airline the user is and must be available | Reordered validations so that all are shown. |
| **HACK** | There should be a invalid value validation message for the following hacking attempts:   * Selecting a Duty that does not exist. * Selecting a leg or a crew member with a not numeric id.   There should be a 500 Access not authorized message for the following possible hacking attempts:   * A user not registered as a crew member tries to access the create form * Selecting a leg or a crew member with a not provided id. | No bugs detected |
| Update | **SAFE** | A user registered as the leg attendant of the leg can update the assignments with the following constraints and validations:   * Only the remarks value can be null * There can be only one lead attendant, pilot and co-pilot by leg * The last update should be a past moment before the leg’s scheduled departure * The leg cannot be changed * The assignee must be from the same airline the user is and must be available | Fixed a bug when trying to update an assignment with no leg selected. |
| **HACK** | There should be an invalid value validation message for the following hacking attempts:   * Selecting a Duty or Status that does not exist. * Selecting a leg or a crew member with a not numeric id.   There should be a 500 Access not authorized message for the following possible hacking attempts:   * Selecting a leg or a crew member with a not provided id. * A non lead attendant tries to update their assignment * A user tries to access an assignment that is not theirs * A user tries to access an object that is not theirs, regardless of the class * A user does not specify an id to update * A user tries to update an object with id not compatible (i.e. a string id) | No bugs detected |
| Delete | **SAFE** | A lead attendant can delete any assignments from the specific leg. In case they try to delete their own assignment as lead attendant, they will receive a validation message telling them that they are not as of that moment allowed because there are other assignments for that leg. | No bugs detected |
| **HACK** | The system should provide a 500 Access not authorized exception in the following cases:   * A non lead attendant tries to delete their assignment * A lead attendant tries to delete a published assignment * A user tries to access an assignment that is not theirs * A user tries to access an object that is not theirs, regardless of the class * A user does not specify an id to delete * A user tries to delete an object with an id not compatible (i.e. a string id) | No bugs detected |
| Publish | **SAFE** | A lead attendant can publish any assignment for a leg that is already published. | No bugs detected |
| **HACK** | The system should provide a 500 Access not authorized exception in the following cases:   * A non lead attendant tries to publish their assignment * A lead attendant tries to publish a published assignment * A user tries to access an assignment that is not theirs * A user tries to access an object that is not theirs, regardless of the class * A user does not specify an id to publish * A user tries to publish an object with an id not compatible (i.e. a string id) | No bugs detected |

## Activity Log

|  |  |  |  |
| --- | --- | --- | --- |
| Test case |  | Expected result | Bug detection |
| List | **SAFE** | Show one list with all activity logs registered by the user. | No bugs detected |
| **HACK** | When trying to access the list by another realm, there should be a not authorized exception. | No bugs detected |
| Show | **SAFE** | Show a user’s logs, accessing it through the list specified above. The action buttons must be shown only if the action in question is considered ‘legal’. | No bugs detected |
| **HACK** | The system should provide a 500 Access not authorized exception in the following cases:   * A user tries to access a log that is not theirs * A user tries to access an object that is not theirs, regardless of the class * A user does not specify an id to be shown * A user tries to be shown an object with an id not compatible (i.e. a string id) | No bugs detected |
| Create | **SAFE** | A user can create a log considering the following validation constraints:   * No value can be null * The incident type must be below 50 characters * The description must be below 255 characters * The registration moment must be in the past but after the leg scheduled arrival * The leg given must be from an assignment the user has * The system must find the assignment and link it to the log, considering the leg given and the registered user | Because of the constraint of the legs and how this entity depends on the leg’s scheduled arrival, no logs can be created in this system as of now[[2]](#footnote-2) |
| **HACK** | There should be an invalid value validation message for the following hacking attempts:   * Selecting a leg with a not numeric id.   There should be a 500 Access not authorized message for the following possible hacking attempts:   * A user not registered as a crew member tries to access the create form * Selecting a leg with a not provided id. | No bugs detected |
| Update | **SAFE** | A user registered as a crew member can update their logs with the following validations:   * No value can be null * The incident type must be below 50 characters * The description must be below 255 characters * The registration moment must be in the past but after the leg scheduled arrival * The leg given must be from an assignment the user has * The system must find the assignment and link it to the log, considering the leg given and the registered user | As explained in the create bugs, no log in the system can actually be updated. |
| **HACK** | There should be an invalid value validation message for the following hacking attempts:   * Selecting a leg with a not numeric id.   There should be a 500 Access not authorized message for the following possible hacking attempts:   * Selecting a leg with a not provided id. * The user is trying to update a published log. * A user tries to access a log that is not theirs * A user tries to access an object that is not theirs, regardless of the class * A user does not specify an id to be updated * A user tries to update an object with id not compatible (i.e. a string id) | No bugs detected |
| Delete | **SAFE** | A crew member can delete any of their logs | No bugs detected |
| **HACK** | The system should provide a 500 Access not authorized exception in the following cases:   * A lead attendant tries to delete a published log * A user tries to access a log that is not theirs * A user tries to access an object that is not theirs, regardless of the class * A user does not specify an id to be deleted * A user tries to delete an object with an id not compatible (i.e. a string id) | No bugs detected |
| Publish | **SAFE** | A crew member can publish any of their logs for an assignment that is already published. | No bugs detected |
| **HACK** | The system should provide a 500 Access not authorized exception in the following cases:   * A crew member tries to publish a published log * A user tries to access a log that is not theirs * A user tries to access an object that is not theirs, regardless of the class * A user does not specify an id to publish * A user tries to publish an object with an id not compatible (i.e. a string id) | No bugs detected |

## Coverage

The coverage for the flight assignment features is as follows:

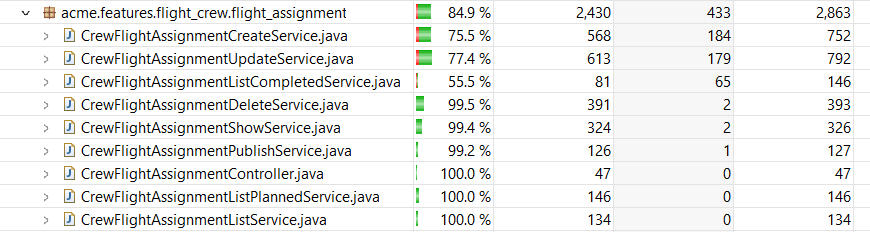


Figure 1. Flight Assignment features coverage

The objective was to manage a 90% of coverage. Although most modules have a coverage of 100%, this is due to the static nature of the project. Because flight assignments have a leg linked and because these legs must be in the future (not a student 3 requirement) all assignments are considered ‘planned’ and none ‘completed’ so the lost service for completed assignments is really low, in part because of its unbinding. If there were at least one completed assignment, this service would be covered as the list service for planned assignments is, that is, fully, and the coverage would go over this desired 90%.

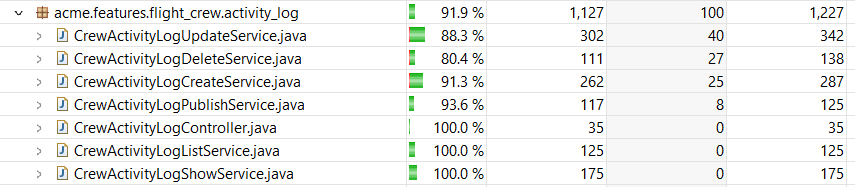


Figure 2. Activity Log feature coverage

In this module, the coverage is better, overcoming this 90% goal.

# Performance testing

Although the annexes specify that the performance must be compared between two computers, I decided to consider two cases to study.

The first case is the difference in performance of the same computer considering if it is charging or if it is not. I consider this case interesting because I generally do not like to work with the computer charging and I did notice the difference while working in this project, so I consider it interesting enough to analyze. The second case is the indexes case, necessary for the completion of the mandatory requirements of the projects.

## First study case: charging vs not charging

For this case there are the following values:

A graph with text on it

AI-generated content may be incorrect.

Figure 3. Average replay times

A graph with blue and white text

AI-generated content may be incorrect.

Figure 4. Average replay times when not charging

As we can see, the difference is not all that notable in the fastest cases, that are the list and show cases, but accumulate with the slower cases, such as update but specially create. The create case is something less than 2 times the time while charging, having an approximate value of 80 and 72 while charging but rising to about 135 and 118 when not.

If we see the means, that are 30.41547744 and 36.27183165 respectively, they are not that different but noticeable enough that while working with the system, the user notices them.

## Second study case: no indexes vs indexes

The study of the usefulness of indexes was mandatory for the deliverable, however there have been some curious values that I would like to discuss.

A graph with text and numbers

AI-generated content may be incorrect.

Figure 5. Average replay times with indexes

The slowest test cases, the create cases, are noticeably lower, managing to reduce almost by 10 the value of assignment/create. However, possibly less noticeable is the rise of update cases.

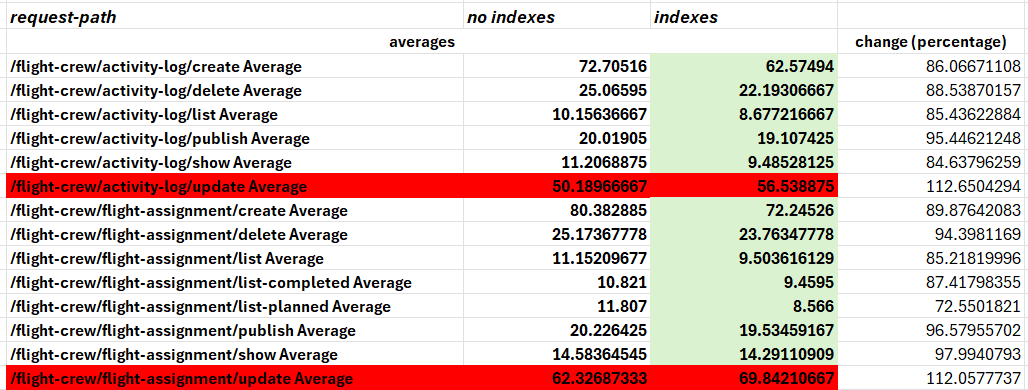


Figure 6. Average times direct comparison

As we can see in (Figure 6), most cases perform better with indexes, around a 12% decrease in time. However, the update cases suffer a 12% uptime, the same percentage but inversed. The mean of all percentages is around 92%, so, overall, the change is positive.

A screenshot of a calculator

AI-generated content may be incorrect.

Figure 7. Z-test study

In our performance evaluation, we have obtained a value of **0.183345503** for an alpha of **0.05**. According to our course methodology, because our p-value is in the (α,1] interval and not in the [0,α) one, the betterment is not significant.

However, due to the reduction of the mean, even if it is minimal, I have decided to keep the indexes as part of the code.

# Conclusions

In this report, we have specified the test cases, their objective and their result and we have analyzed the performance in three distinct cases.

A graph of a graph with text

AI-generated content may be incorrect.

Figure 7. Full performance comparison

By this comparison, we have found that the optimal working conditions for our project and my computer is to run it while charging and with indexes.

1. For more information, see the analysis report [↑](#footnote-ref-1)
2. For more information, check the analysis report [↑](#footnote-ref-2)