Un dibujo con letras

El contenido generado por IA puede ser incorrecto.

July 4th, 2025

Grupo: C2.036

Sevillano Barea, Alejandro - alesevbar@alum.us.es

Naredo Bernardos, Ignacio- ignnarber@alum.us.es

Escudero Aldana, David - davescald@alum.us.es

TESTING REPORT

Repository: https://github.com/C1-036/Acme-ANS-C2

**ÍNDICE**

[*Executive summary 2*](#_heading=h.lhp64ytaurpc)

[*Introduction 2*](#_heading=h.y1gfeebdufeh)

[*Tests on /technician/maintenance-record/... 2*](#_heading=h.9uw3mmnyn03m)

[*Tests on /technician/task/... 3*](#_heading=h.fdtbdyw4bf1u)

[*Tests on /technician/involves/... 5*](#_heading=h.9b1msyfg6bvk)

[*Supplementary tests for coverage 5*](#_heading=h.llbplt330dpq)

[*Code coverage 6*](#_heading=h.2frj70hu1vj4)

[*Performance analysis 6*](#_heading=h.vv3fv3jl8f96)

[*Conclusiones 10*](#_heading=h.i4e05nbbsohk)

[*References 10*](#_heading=h.dorx5wlpv4v3)

# Executive summary

This report compiles the tests performed on the functionalities implemented by Student #5 in the Acme-ANS project, according to the assigned individual requirements. The main objective is to demonstrate that the functionalities are correct, secure, and conform to specifications.

It has been manually tested using .safe (expected functionality) and .hack (robustness and security checks) cases, covering the MaintenanceRecord, Task, and Involves entities. In addition, additional tests have been included to maximize coverage.

Full coverage has been achieved in all services. Both expected functionality and edge cases have been tested. Performance testing is pending and will be completed later.

# Introduction

The objective of this report is to validate through formal tests the correct implementation of the functional requirements associated with the Technician role on the MaintenanceRecord, Task, and Involves entities. The analysis has been structured following the methodology outlined in class, implementing positive tests and negatives using the .safe and .hack files, respectively.

The report is organized into blocks by functionality, detailing the expected behavior, covered cases and coverage assessment code. The tests were launched after performing a full replay of the system.

# Tests on /technician/maintenance-record/...

The features implemented for maintenance records allow technicians to create, view, modify, delete, and publish records of tasks performed on aircraft. Each operation has been verified using .safe (correct flow) and .hack (incorrect access or use) tests.

* **list.safe:** The list of records has been accessed from the authenticated technician view. Only your own records appear, both draft and published. It has been verified that the "Show" and "Delete" buttons only appear for drafts
* **show.safe:** Accessing the details of a specific record from the list. The data and associated tasks are displayed correctly. The file is validated for authorization errors and rendering errors.
* **create.safe:** A new maintenance record has been created by filling out the required fields, and its persistence in the list has been validated. The initial state has been verified as draftMode = true.
* **update.safe:** From the "Show" view, you modify your own draft record. The date, cost, and notes are edited. When saved, the changes are correctly reflected in the database.
* **publish.safe**: A record was created, then a published task was added to it using Involves, and finally "Publish" was pressed. The operation was successful only when there was at least one published task. It was verified that the status changed to draftMode = false.
* **delete.safe:** A draft record is deleted from the list. It has been verified that it no longer appears in the list, and the database has been checked to ensure that it has been completely deleted.
* **.hack:** Access attempts have been attempted via invalid IDs, parameter manipulation, and simulations of improper navigation. The system responds with a "panic" redirect or access denied.

# Tests on /technician/task/...

The Task entity allows you to define and manage specific actions to be performed during aircraft maintenance. Operations are available only to authenticated technicians.

* **list.safe:** From the "List my tasks" menu, the list of your own tasks has been accessed. Both draft and published tasks are displayed. It has been verified that the "Update" button only appears for drafts, and that tasks are sorted by priority.
* **show.safe:** You can access the details of your own task from the list. The description, type, duration, and priority are displayed correctly. The system does not allow access to other tasks.
* **create.safe**: A task linked to a draft maintenance record has been created. In create2, the behavior was checked with data limits (maximum duration, 255-character description, priority 10). It was also confirmed that the task remains as a draft (draftMode = true).
* **update.safe**: Edits a draft task, changing its duration, description, and type. The updated data is verified as being reflected in the list, and editing published tasks is not allowed.
* **publish.safe**: From a task's details, click "Publish." The task is verified to have a draftMode = false status and can no longer be edited. Only tasks with all fields valid were allowed to be published.
* **delete.safe:** A draft task is deleted. It has been confirmed that it disappears from the list and is no longer in the database. It is not possible to delete already published tasks.

**.hack:** The following tests have been carried out to ensure the system's robustness against possible attacks:

* **list.hack:** Attempts to access the full list of published tasks and drafts from other technicians. Although the interface doesn't allow this, direct access via URL has been tested. The system correctly redirects to the error.
* **show.hack:** Forces access to a task using its ID without authorization. The system returns a 403 error and prevents access.
* **create.hack:** A non-existent maintenanceRecordId or one from another technician is submitted. The form is found to be unprocessed. Extreme or invalid values are used in the task description, type, or duration. The system detects errors and displays validation messages.
* **update.hack:** An attempt is made to edit a task that has already been published. The system disables the editing buttons and denies the operation.
* **publish.hack:** Forces the publication of incomplete or manipulated tasks. It is verified that the publication is not completed and that the system retains the task as a draft.
* **delete.hack:** An attempt is made to delete a task that doesn't belong to the technician or that has already been published. Access is correctly denied.

# Tests on /technician/involves/...

The Involves entity represents the relationship between a published task and a draft maintenance record. It has been verified that a technician can only link their own published tasks to their own unpublished records.

* **create.safe:** From the details of a draft MaintenanceRecord, the "Add tasks" view is accessed. We tested selecting our own published tasks from a dropdown and linking them correctly. We verified that the tasks appear listed in the record.
* **delete.safe:** In the MaintenanceRecord detail, a previously linked task has been deleted. It has been verified that it no longer appears linked in either the detail or the database.

.hack: Malicious actions have been simulated to verify that the task association functionality with maintenance records is secure:

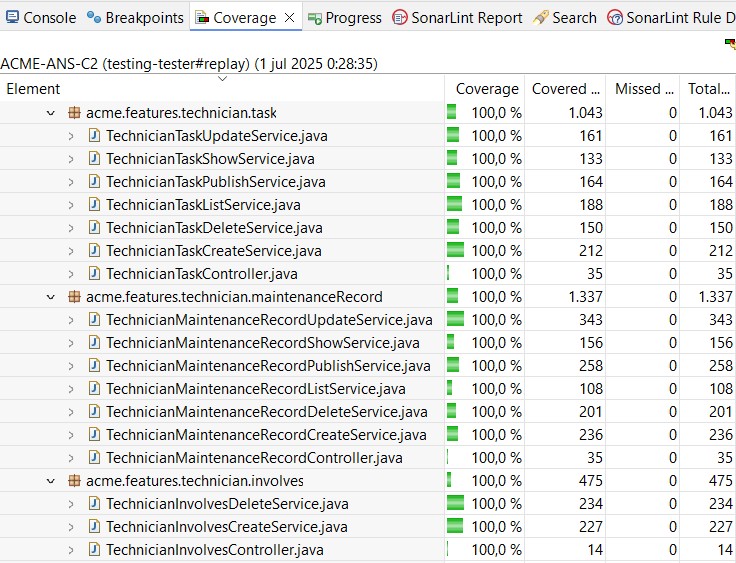
* **create.hack:** An attempt is made to link a published task that doesn't belong to the authenticated technician. The system throws an authorization error. Forces the creation of an Involves on an already published MaintenanceRecord. The system detects that it cannot be modified and blocks the action.
* **delete.hack:** The ID in the URL is manipulated to attempt to delete a relationship between a task and another technician's record. The system denies access and redirects to the error view. Forces the deletion of an Involve with a nonexistent or malformed ID. The system returns a secure response without compromising the data.

# Supplementary tests for coverage

An additional case delete2.hack has been included to ensure the coverage of alternative logical paths in maintenance records.

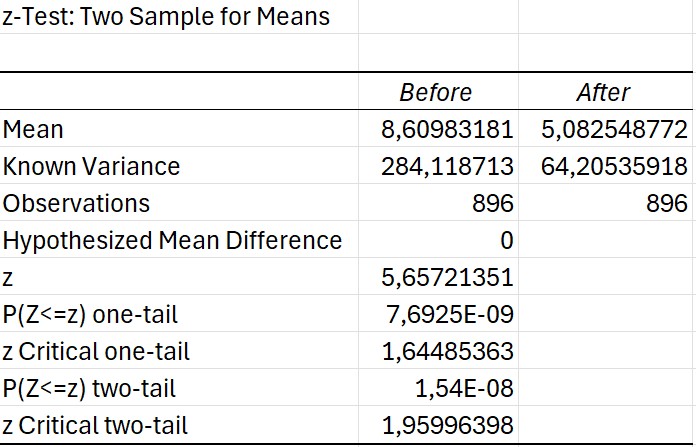
# Code coverage

100 % test coverage has been achieved across all relevant services (MaintenanceRecord, Task, Involves) for the Technician role.All code paths have been fully exercised through both .safe and .hack tests, including all edge conditions and security checks.

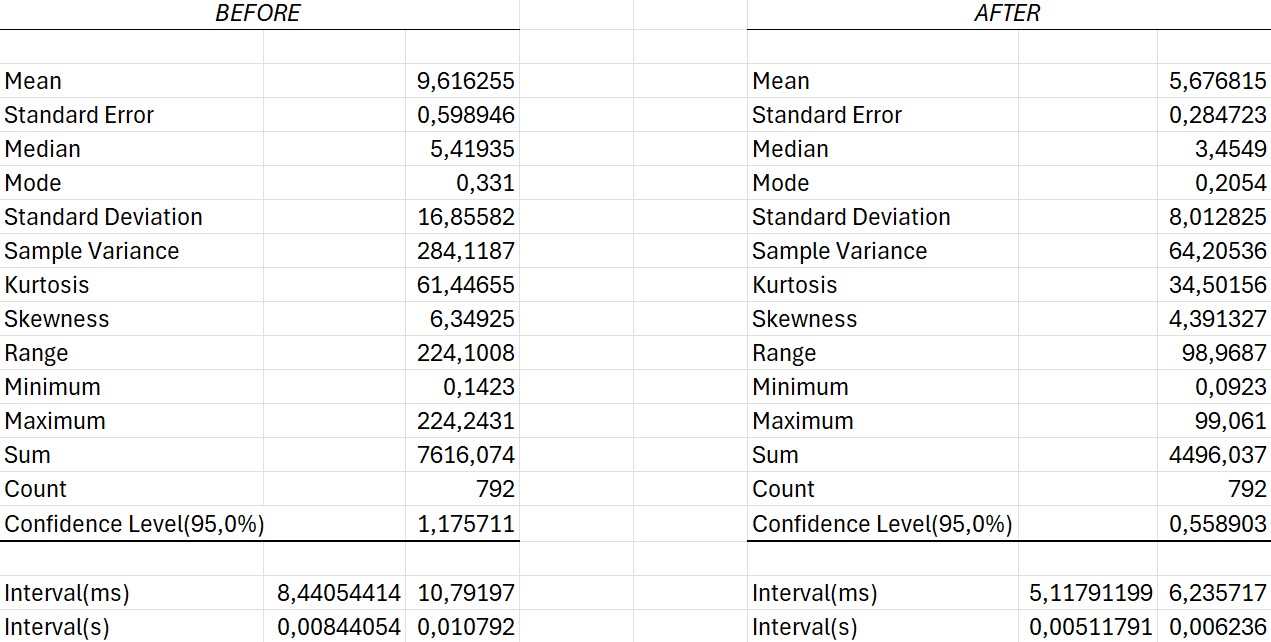


# Performance analysis

First, we'll begin with some basic statistics on the application's performance before and after optimizing the indexes, as well as their respective time/ feature graphs. To compare these times, a hypothesis test was performed using a Z-test, which demonstrates that the changes are indeed statistically significant.



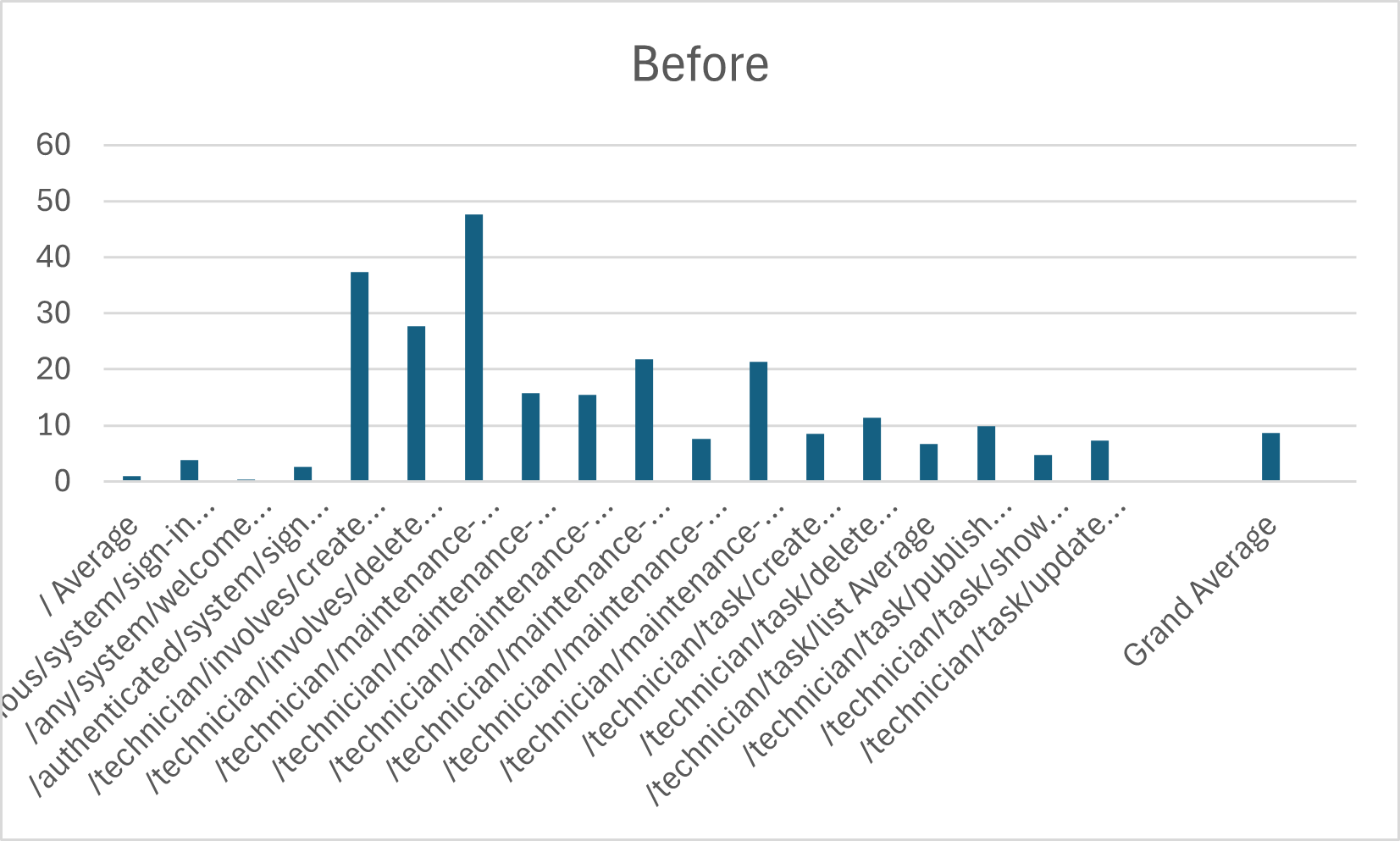
As we can see, the p-value obtained in the Z test (two-tailed) is 1,54E-08(almost 0). Considering a 95% confidence level, the significance level is α = 0,05. Since the p-value is less than α, the null hypothesis is rejected, indicating that the changes made are statistically significant. Furthermore, since the value is very small, this significance is clearly noticeable. Next, we will compare the times before and after applying the changes.



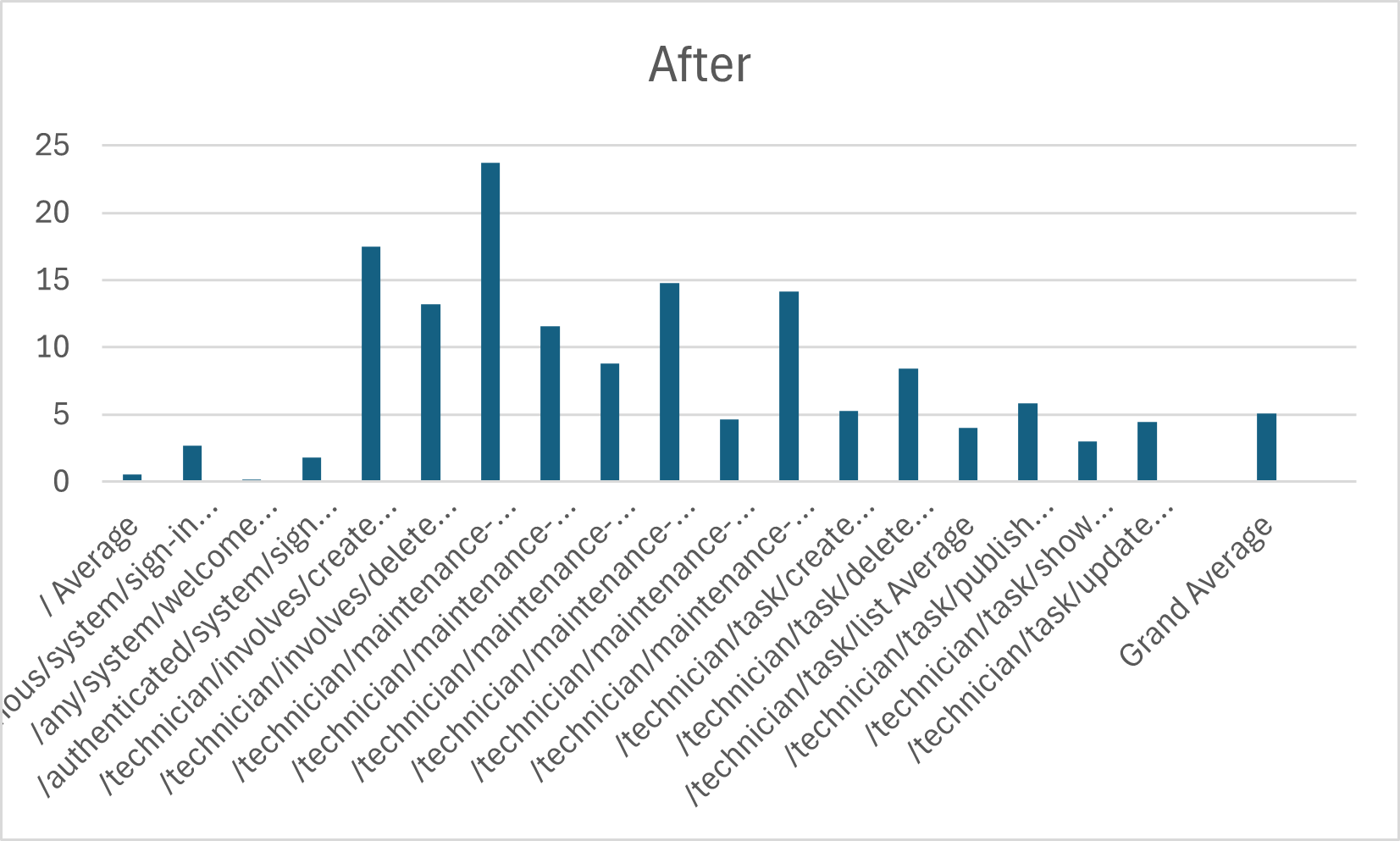
If we look at the confidence interval, before making the changes to the indexes, we could guarantee that any request would be answered in a time less than or equal to 10,79197 ms with 95% confidence. However, after optimizing the database search, this upper bound on the time has decreased to 6,235717 ms, or 4,556253 ms less. This translates into an improvement in application performance of approximately 42,21%. This result is statistically significant, as demonstrated by the p-value obtained, which is considerably lower than the usual significance level of 0,05.

Next, we will proceed to look at the graphs for each one, to see if the MIR response time (most inefficient request) has been mitigated:

**Before graph:**

****

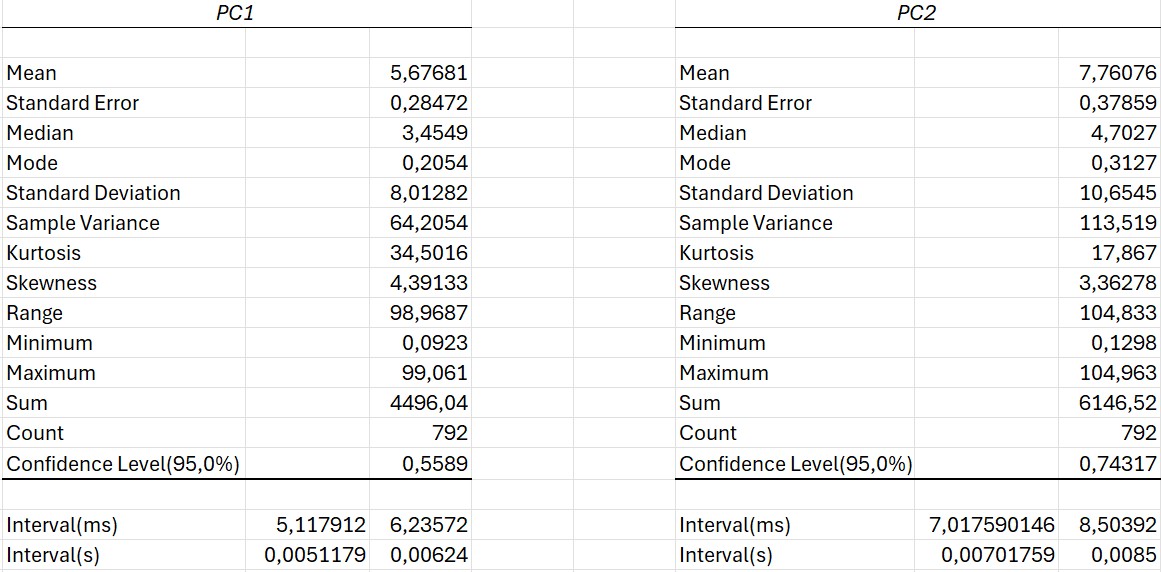
**After graph:**

****

As we can see, the MIR still belongs to the same feature (/technician/maintenance-record/create), but it has been significantly reduced, specifically from approximately 47,64147 ms to 23,74510286 ms. A similar reduction applies to most features except those where the MIR was not as high. Specifically, the second greatest reduction is found in /technician/involves/create, with an improvement of 53.27% (from 37,41973 ms to 17,48398 ms). For a better visualization of these changes, see the graphs shown earlier, which shows the difference in times (after vs. before) in milliseconds.

As we can see, with the exception of the technician/maintenance-record/update and technician/task/update features, which have been worsened, the rest have either remained the same or improved somewhat. What is clear is that this change has turned out for the better, as previously confirmed.

To conclude this section, let's compare two different computers to see how important the hardware is for responding to WIS queries (this comparison is only valid after index optimization). PC\_1 will use the same one used for this report (12th Gen Intel(R) Core(TM) i7-12650H 2.70 GHz processor), while PC\_2 will use a colleague's laptop (13th Gen Intel(R) Core(TM) i5-1335U 1.30 GHz processor):



As we can see, in the case of PC\_2, the confidence interval has an upper bound of 8,50392 ms, which is approximately 1,3639 times greater than the upper bound recorded for PC\_1. Although directly comparing by confidence interval is not always the most accurate method, the results show that PC\_1 performs significantly better, as it guarantees much lower response times.

# Conclusiones

This testing report has demonstrated that the implementation of the functionalities assigned to Student #5 has been rigorous, secure, and aligned with the formal requirements of the project. All features related to maintenance records, tasks, and their relationships were thoroughly tested using both .safe and .hack cases, covering a wide range of positive and negative scenarios.

The system consistently rejected unauthorized access attempts and improper manipulations, ensuring robust security mechanisms. Test coverage reached 100 % in all services, with all execution paths and conditional branches exercised and validated, including edge cases and security-related logic.

Furthermore, the performance analysis has confirmed, through a Z-test and confidence interval calculations, that the introduction of database indexes led to a statistically significant improvement in response times. This optimization ensures a more efficient user experience and reinforces the system’s scalability and reliability.

Overall, this report supports the technical quality and maturity of the solution developed, laying a solid foundation for future enhancements or deployment in production environments.

# References

University of Seville website -<https://ev.us.es>