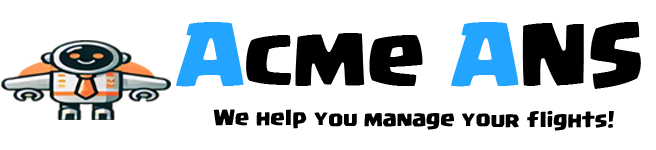
**Testing Report**

****

Diseño y Pruebas II

2024/25

Group: C1.009

# GitHub Organization: <https://github.com/DP2-2024-2025-C1-009>

# 

Jaime Gómez Marín

jaigommar1@alum.us.es

## **Tabla de Contenido**

1. Resumen Ejecutivo
2. Tabla de Revisión
3. Pruebas Funcionales
   * Casos de Prueba
     + Create
     + List
     + Show
     + Update
     + Publish
     + Delete
4. Pruebas de Rendimiento
   * Análisis de la Mejora

## **1. Resumen Ejecutivo**

En este informe se detallan las pruebas funcionales y de rendimiento realizadas sobre la funcionalidad de los técnicos, más concretamente los registros de mantenimiento, las tareas y los involucraciones. El objetivo principal fue verificar que todas las funcionalidades (create, list, show, update, publish, detele) se comportaran según lo esperado, tanto en escenarios de uso seguro (.safe) como en intentos de manipulación o acceso no autorizado (.hack), y evaluar la rapidez del sistema bajo diferentes condiciones.

Para las pruebas funcionales, se organizaron los casos de prueba por funcionalidad, asegurando la correcta operación de la aplicación. En cuanto a las pruebas de rendimiento, se siguió una metodología para recopilar y procesar los tiempos de ejecución, incluyendo el cálculo de intervalos de confianza y una prueba de hipótesis para comparar el rendimiento en distintas configuraciones o máquinas.

## **2.Tabla de revisión**

| Versión | Fecha | Comentario |
| --- | --- | --- |
| 1.0 | 26/05/2025 | Primera versión |

## **3. Pruebas Funcionales**

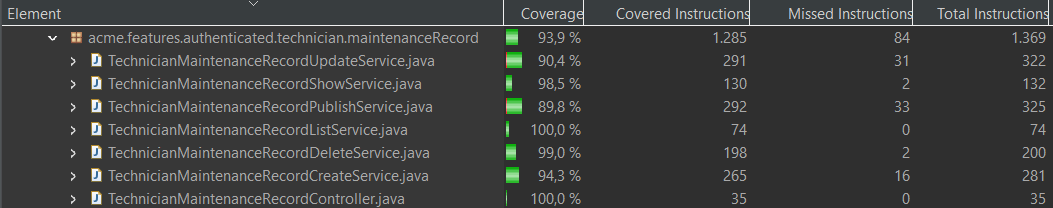
### **1. MaintenanceRecord**

* **TechnicianMaintenanceRecordUpdateService**
  + **.safe:** Verifica que el técnico pueda acceder al formulario de edición de un registro existente mediante GET a `/technician/maintenance-record/update?id=164` (y casos con `id=504`), y posteriormente enviar una petición POST con datos válidos (fecha, descripción, estado, relación con equipo), validando únicamente los campos permitidos y guardando los cambios en base de datos, devolviendo HTTP 200 o redirigiendo a la vista detallada.
  + **.hack:** Intenta acceder o modificar registros inexistentes (IDs que no existen) o registros de otros técnicos, así como cambiar campos restringidos, mediante GET o POST a `/technician/maintenance-record/update?id=X`, comprobando que el sistema rechace la operación con un error 404 o “not authorised” y no aplique modificaciones no autorizadas.
* **TechnicianMaintenanceRecordShowService**
  + **.safe:** Comprueba la visualización detallada de un registro de mantenimiento existente mediante GET a `/technician/maintenance-record/show?id=357` (y casos análogos con otros IDs válidos), asegurando que la respuesta HTTP 200 incluya todos los atributos del registro (fecha, descripción, estado, relación con equipo, etc.) y que se muestren correctamente en la interfaz.
  + **.hack:** Simula peticiones GET a `/technician/maintenance-record/show` con IDs inválidos (por ejemplo `id=399`) o pertenecientes a otros técnicos, comprobando que el sistema devuelva un error (404 o un genérico 500) o un mensaje de “not authorised”, y que no exponga ningún dato sensible ni detalle de registros ajenos.
* **TechnicianMaintenanceRecordPublishService**
  + **.safe:** Asegura que un registro en estado borrador pueda pasar a publicado mediante POST, actualizando su estado, guardando un historial de versiones y (si aplica) notificando el cambio al sistema de notificaciones.
  + **.hack:** Fuerza la publicación de registros que no están en borrador, que carecen de los campos requeridos o que pertenecen a otros técnicos, o bien realiza la acción sin los permisos necesarios, comprobando que el sistema rechaza la operación y mantiene el estado original.

* **TechnicianMaintenanceRecordListService**
  + **.safe:** Valida que el técnico obtenga la lista completa de sus registros de mantenimiento mediante GET, con paginación y filtros (fecha, equipo, estado) funcionando correctamente y mostrando únicamente sus propios registros.
  + **.hack:** Manipula parámetros de paginación o filtros para intentar acceder a registros de otros técnicos, o bien solicita la lista sin estar autenticado, comprobando que el sistema restrinja el acceso y no exponga información ajena.
* **TechnicianMaintenanceRecordCreateService**
  + **.safe:** Verifica la creación de nuevos registros de mantenimiento con datos válidos (fecha, descripción, relación con equipo) mediante una petición POST autorizada, asegurando que el registro persista correctamente y se redirija al listado o vista detallada.
  + **.hack:** Intenta crear registros con campos inválidos o vacíos (por ejemplo, aircraft=0), IDs duplicados o sin sesión válida/permisos de técnico, comprobando que el sistema rechaza la solicitud y no persiste datos no autorizados.

* **TechnicianMaintenanceRecordDeleteService**
  + **.safe:** Comprueba que el técnico propietario pueda eliminar un registro existente mediante la petición DELETE (o POST de borrado), asegurando la limpieza de todas las referencias asociadas y devolviendo la respuesta de éxito adecuada.
  + **.hack:** Realiza intentos de borrado sobre registros inexistentes o pertenecientes a otros técnicos, o bien sin sesión válida, verificando que el sistema impida la operación y no elimine ningún dato.

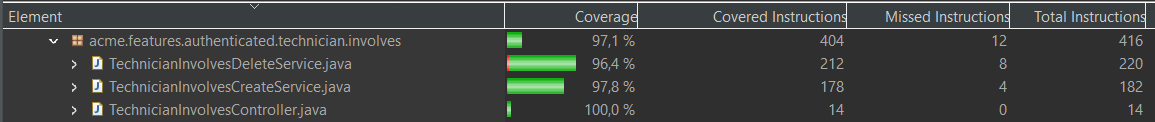
### Cobertura



He obtenido un 93.9% de cobertura en las pruebas relacionadas con maintenance record. Las instrucciones que no se han ejecutado están relacionadas con líneas que contienen operadores tales como “&&” o mensajes de error.

### **2. Involves**

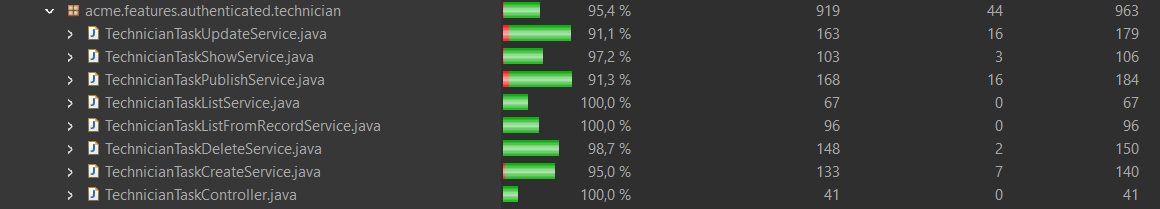
* **TechnicianInvolvesCreateService**
  + **safe:** Envía una petición POST a `/technician/involves` con un payload válido como técnico autenticado, y comprueba que: 1. Se crea un nuevo registro en la base de datos con un ID único. 2. Todos los atributos se persisten correctamente. 3. La respuesta es un redireccionamiento (HTTP 302) a la vista detallada.
  + **.hack:** Intenta malformar o falsificar la petición de creación, por ejemplo: Se omite o invalida el token CSRF. Se envían campos obligatorios vacíos o con formato incorrecto. Usa un id duplicado o no existente. Realiza la petición desde una cuenta sin rol de técnico. Verifica que el sistema rechace la solicitud (HTTP 400/403/422) y que no se persista ningún dato.
* **TechnicianInvolvesDeleteService**
  + **.safe:** Envía DELETE a `/technician/involves/{id}` con token CSRF como técnico propietario, comprueba que el registro y sus referencias se eliminen y que se redirija al listado con mensaje de éxito.
  + **.hack:** Realiza la petición sin CSRF o con sesión caducada, usando un id ajeno o inexistente, y comprueba HTTP 403/404 sin eliminación de datos.



He obtenido un 97.1% de cobertura en las pruebas relacionadas con involves. Las instrucciones que no se han ejecutado son líneas que se han ejecutado parcialmente por diseño de código.

### **3. Task**

* **TechnicianTaskUpdateService**
  + **.safe:** Verifica que el técnico pueda actualizar detalles de una tarea (estado, descripción, fechas) de forma parcial o completa.
  + **.hack:** Fuerza actualizaciones sobre tareas de otros técnicos o tareas finalizadas, confirmando que se impida.
* **TechnicianTaskShowService**
  + **.safe:** Comprueba la presentación de todos los datos de una tarea específica (asignado, prioridad, notas).
  + **.hack:** Solicita tareas con IDs inválidos o de terceros para asegurar que no se muestre información no autorizada.
* **TechnicianTaskPublishService**
  + **.safe:** Asegura que el técnico pueda publicar o marcar como “lista” una tarea, cambiando su estado internamente.
  + **.hack:** Intenta publicar tareas inválidas o de otros usuarios para validar restricciones de acceso.
* **TechnicianTaskListService**
  + **.safe:** Valida que el técnico obtenga la lista completa de sus tareas, con filtros y paginación en correcto funcionamiento.
  + **.hack:** Manipula parámetros de listagem para tratar de acceder a tareas de terceros y comprueba que no sea posible.
* **TechnicianTaskListFromRecordService**
  + **.safe:** Verifica el filtrado de tareas asociado a un registro de mantenimiento concreto, mostrando únicamente las relacionadas.
  + **.hack:** Intenta listar tareas vinculadas a registros ajenos o no existentes para garantizar que la consulta no devuelva datos.
* **TechnicianTaskCreateService**
  + **.safe:** Comprueba la creación de nuevas tareas con datos válidos y asociación correcta a registros de mantenimiento.
  + **.hack:** Simula creación con campos inválidos o sin permisos, garantizando rechazo por validación y control de acceso.
* **TechnicianTaskDeleteService**
  + **.safe:** Prueba la eliminación de tareas por el técnico propietario, incluyendo chequeo de dependencias y limpieza.
  + **.hack:** Realiza intentos de borrado sobre tareas de otros técnicos o ya eliminadas para verificar el manejo de errores.



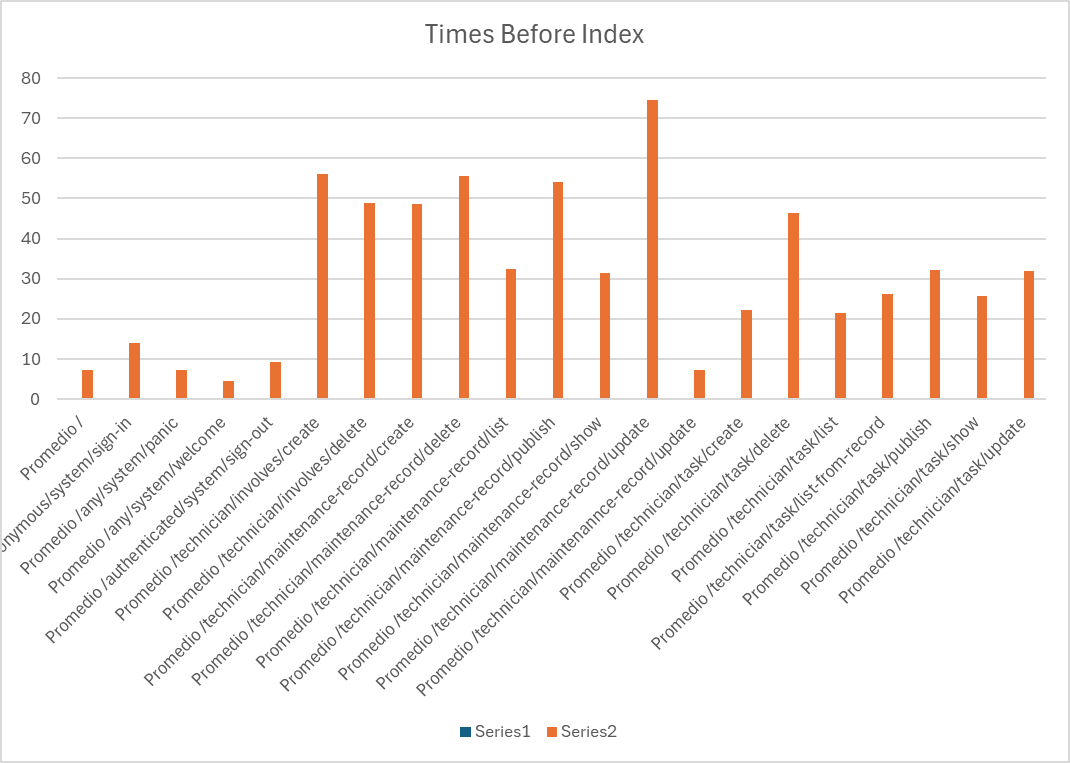
He obtenido un 95.4% de cobertura en las pruebas relacionadas con tasks. Las instrucciones que no se han ejecutado son líneas que se han ejecutado parcialmente por diseño de código.

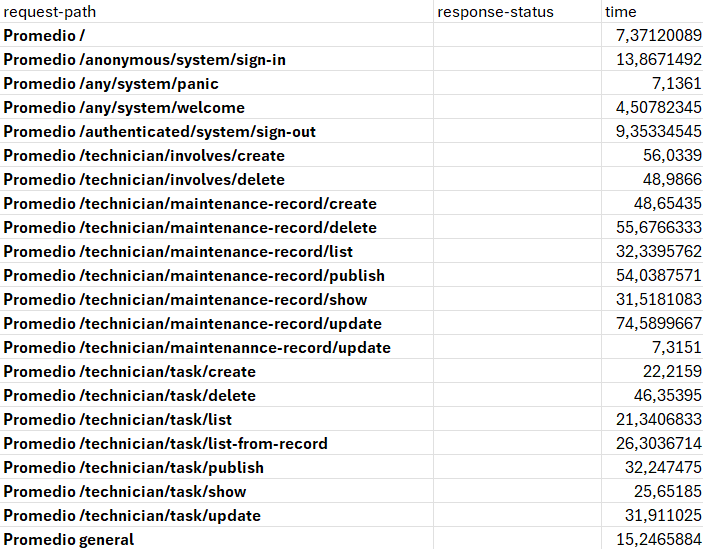
## **4. Pruebas de Rendimiento**

Se realizaron pruebas de rendimiento con el objetivo de optimizar el acceso a los datos y reducir los tiempos de respuesta de los servicios implicados. Para lograrlo, se implementaron índices en los campos clave de las entidades, lo que resultó en una aceleración significativa de las consultas en la base de datos, especialmente en las operaciones de filtrado y búsqueda. Como consecuencia, se observó una mejora notable en los tiempos de ejecución, lo cual impactó positivamente tanto en la experiencia del usuario como en la eficiencia general del sistema.

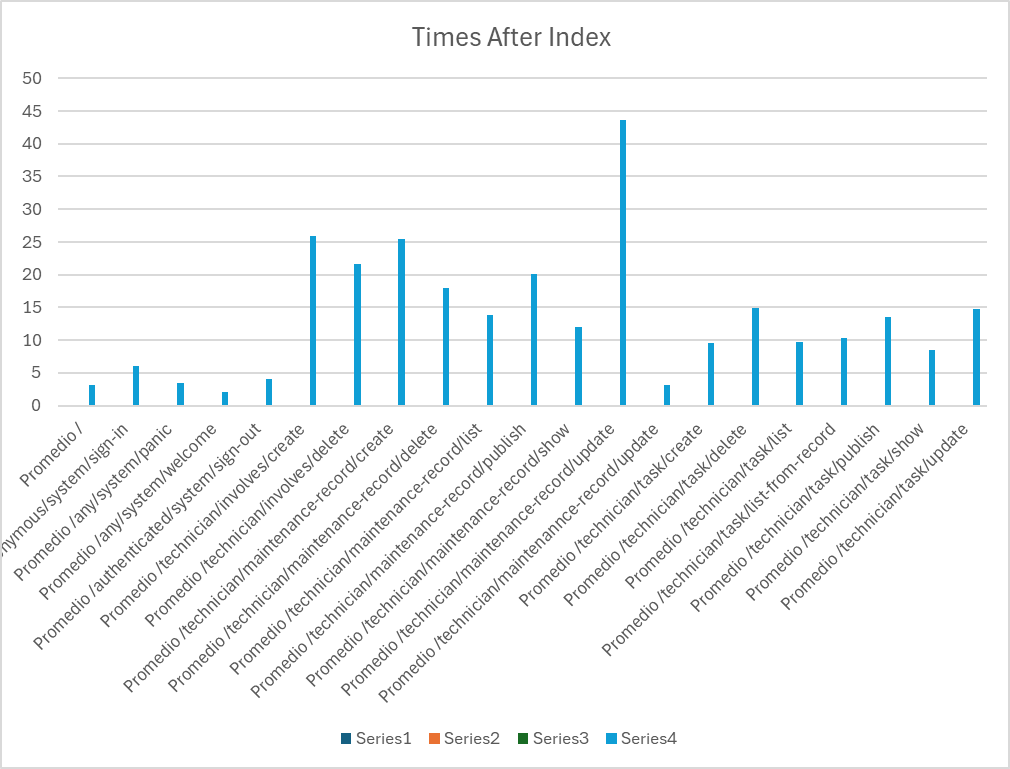
Para respaldar estos hallazgos, se llevó a cabo un análisis estadístico utilizando Excel y una prueba Z (Z-test). Esta prueba permitió verificar si la disminución de los tiempos de respuesta, tras la introducción de los índices, era estadísticamente significativa. El valor obtenido, Pz<0.00, nos permite rechazar la hipótesis nula y confirmar que los cambios implementados han tenido un impacto positivo y significativo en el rendimiento del sistema.

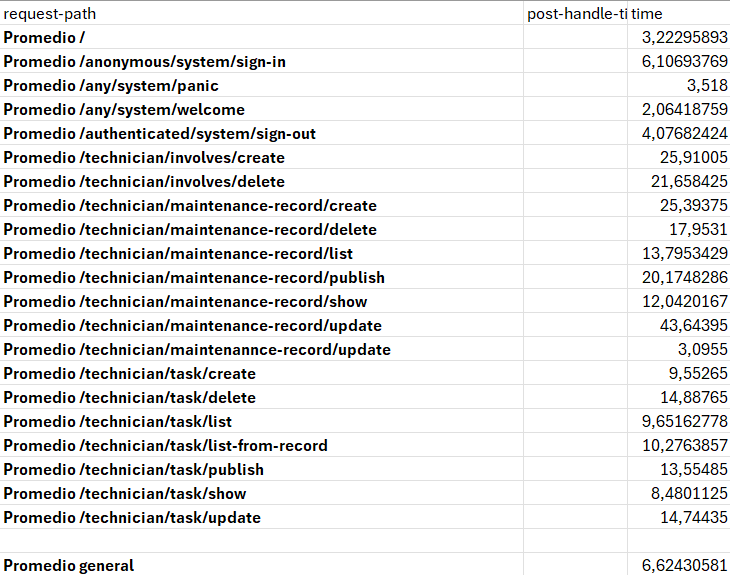
Primero analizaremos los datos de la primera ejecución y luego los compararemos con la segunda







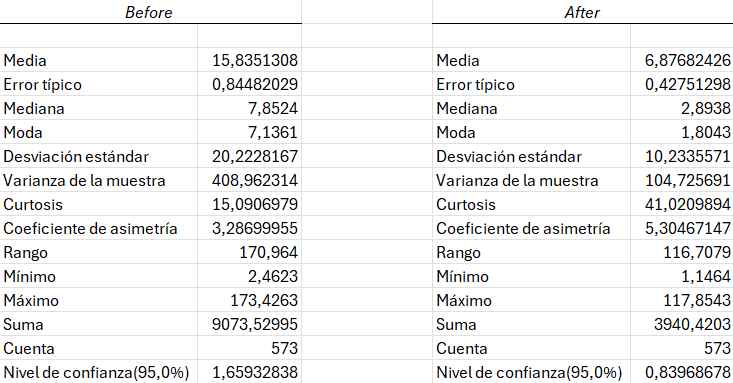


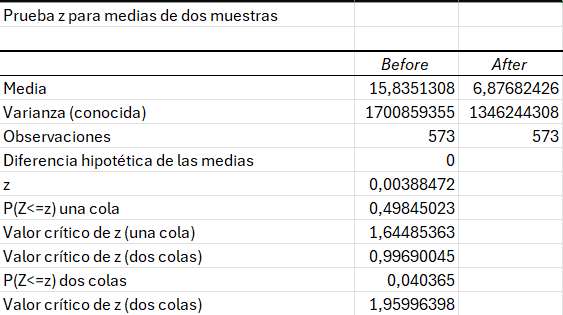




Al analizar ambas gráficas, se observa que tras la introducción de los índices todos los endpoints muestran una reducción clara en los tiempos de respuesta. Por ejemplo, operaciones críticas como technician/maintenance-record/update, que es el MIR, y technician/task/update pasan de picos de 73 ms (antes) a 44 ms (después). Asimismo, los endpoints con menor carga, como system/sign-in, reducen su tiempo de 5 ms a 2 ms, demostrando una mejora generalizada.

La distribución de los tiempos también se homogeniza tras optimizar las consultas: las variaciones extremas se atenúan, reduciendo los picos máximos y estrechando los valores medios y cuartiles. En conjunto, esto confirma que los índices no solo disminuyen la latencia promedio, sino que también aportan mayor consistencia y predictibilidad al rendimiento del sistema.





Tras introducir los índices, la media de los tiempos de respuesta se redujo de 15,84 s a 6,88 s (−56,6 %), acompañada de una caída de la mediana de 7,85 s a 2,89 s (−63 %) y de la moda de 7,14 s a 1,80 s. Al mismo tiempo, la desviación estándar descendió un 49 % (de 20,22 s a 10,23 s) y la varianza un 74 % (de 408,96 a 104,73), lo que denota una mayor consistencia en los tiempos. El rango de valores extremos se acortó un 31,7 % (de 170,96 s a 116,71 s), con un mínimo que bajó de 2,46 s a 1,15 s y un máximo de 173,43 s a 117,85 s, reduciendo así los picos de latencia. Además, la suma total de todos los tiempos cayó un 56,6 %, y el intervalo de confianza al 95 % se estrechó de 1,66 s a 0,84 s, confirmando la fiabilidad de la mejora. Aunque el coeficiente de asimetría y la curtosis aumentan (de 3,29 a 5,30 y de 15,09 a 41,02, respectivamente), reflejando colas más pesadas, el resultado global es una aplicación mucho más rápida y predecible.

El intervalo de confianza al 95 % también se estrechó, pasando de aproximadamente **±5,47 ms** a **±4,89 ms**, lo que refleja una estimación más precisa del tiempo medio tras la aplicación de los índices.

Para comparar adecuadamente los intervalos de confianza calculados tras ejecutar las pruebas, se ha realizado la prueba **Z-Test** sobre las columnas de tiempos generados. El valor que nos interesa es **P(Z<=z) two-tail**

Dicho valor nos indica que nos encontramos en el intervalo ( 0.00 - alpha ], donde, **alpha** = 1 - Intervalo de Confianza = 1 - 0.95 = **0.05**

Al ser z = *0.040365*  tenemos que: *0 = < z < alpha.* El *p-value* está en el intervalo [0.00,alpha)

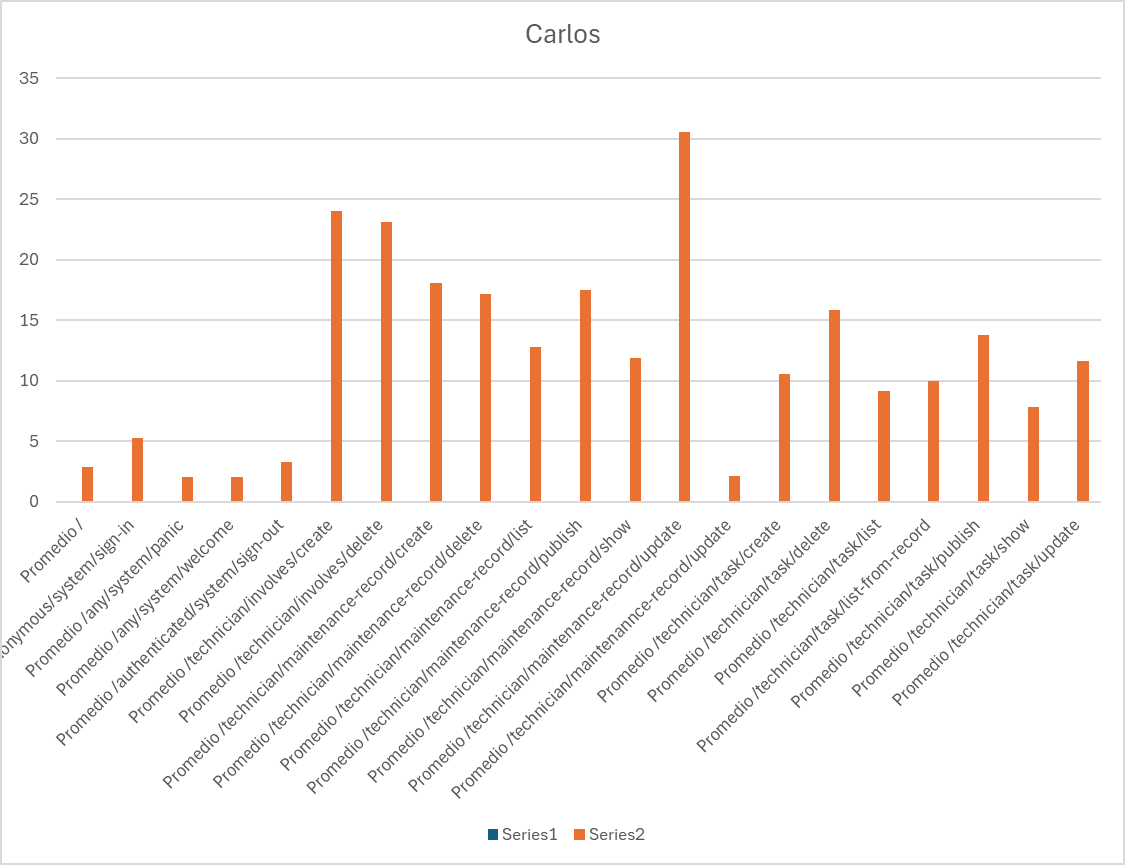
Tras la obtención y análisis de estos datos podemos concluir que se ha aumentado el rendimiento y se ha reducido el tiempo medio de petición. Por lo tanto, la implementación de los índices en las entidades ha sido un éxito.

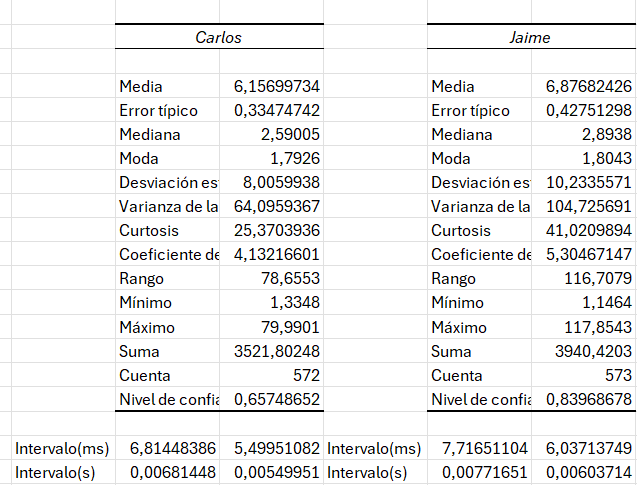
### Comparativa con el ordenador de un compañero

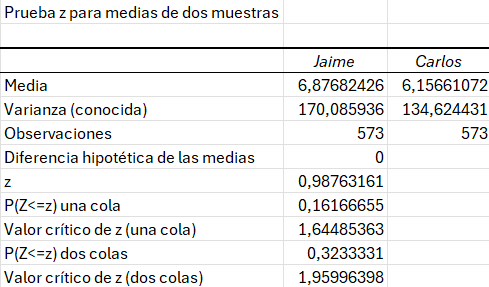
Este apartado consiste en comparar y analizar las diferencias entre el rendimiento obtenido desde dos máquinas distintas. En esta ocasión haré la comparativa con el ordenador de mi compañero de grupo cargalrod2.  
  
Para la comparativa he usado la versión con índices.

**Ordenador personal**: Ryzen 7 3700U, 16GB RAM, 512 SSD

**Ordenador de Carlos**: Ryzen 7 4700U, 16GB RAM, 512 SSD







Tras analizar los datos podemos concluir que el ordenador de mi compañero Carlos tiene un rendimiento similar al mio para este tipo de trabajos. En la gráfica de barras podemos comprobar como, aunque baja en todos los tipos de peticiones, la que consume más tiempo sigue siendo */technician/maintenance-record/update* seguido de */technician/involves/create*

El *p-value* obtenido es **0,3233331**, valor que no es perteneciente al rango [0.00, alpha), siendo alpha 0.05

Esto nos resalta que no existe un cambio realmente grande entre las dos máquinas en cuanto a rendimiento se refiere.

# **Conclusiones**

En conclusión, el informe demuestra que tanto las pruebas funcionales como las de rendimiento cumplen sus objetivos de forma satisfactoria: por un lado, los casos .safe y .hack cubren exhaustivamente todas las operaciones sobre técnicos (create, list, show, update, publish, delete), alcanzando más de un 93 % de cobertura en maintenance records, un 97 % en involves y 95.4% de cobertura en las pruebas relacionadas con tasks, lo que garantiza robustez y control de accesos según lo diseñado; por otro, la introducción de índices en la base de datos ha reducido drásticamente los tiempos de respuesta (media −56,6 %), ha homogeneizado la latencia y ha pasado la prueba Z con p < 0,05, confirmando su significación estadística . Además, la comparación con el equipo de un compañero revela un rendimiento equiparable, lo que refuerza la validez de los resultados en distintos entornos. En conjunto, el documento avala el correcto funcionamiento, la seguridad y la eficiencia del módulo de técnicos, recomendando la adopción de estos índices y la metodología de pruebas para futuras iteraciones.

# **Bibliografía**

* S02 - Performance testing (L04)
* Documento de anexos de la asignatura, “06- Anexes”