Universidad de Sevilla  
Escuela Técnica Superior de Ingeniería Informática

TESTING REPORT INDIVIDUAL - D04



**C1.027**

**Repositorio**: <https://github.com/josrojrom1/DP2-G27-Acme>

**Autor:**

Rojas Romero, José Joaquín

josreimun@alum.us.es

**Fecha:** 19/05/2024

**Tabla de contenidos**

[**Resumen**](#_heading=h.gjdgxs) **2**

[**Tabla de revisión**](#_heading=h.30j0zll) **2**

[**Introducción**](#_heading=h.1fob9te) **2**

[**Contenidos**](#_heading=h.3znysh7) **3**

**Capítulo I: Testing funcional 3**

**Capítulo II: Testing de rendimiento 4**

[**Bibliografía**](#_heading=h.3dy6vkm) **4**

# Resumen

Este documento trata del proceso de testing seguido en los requisitos individuales de entregas anteriores, con el objetivo de llegar a una conclusión de dichos resultados en materia de mejora del código o del mismo proceso de testing. El documento consta de dos capítulos que tratan de los métodos estadísticos realizados para calcular intervalos de confianza y el contraste de hipótesis, alcanzando una valoración final.

# Tabla de revisión

| **Versión** | **Fecha** | **Descripción** |
| --- | --- | --- |
| V1 | 19/05/2024 | Creación y desarrollo del documento |

# Introducción

A continuación veremos separados en dos capítulos, los métodos estadísticos realizados para analizar los resultados tras el testing formal de los requisitos de todas las entregas del proyecto.

# Contenidos

# Capítulo I: Testing funcional

Este capítulo mostrará el listado con los casos de prueba implementados, agrupados por *features*. Para cada caso de prueba, proporcionará una descripción sucinta y una indicación clara sobre la eficacia de la detección de errores.

**Rol Auditor sobre entidad Audit Record (registro de auditoría)**

**auditor/audit-record/create:** Para realizar las pruebas de esta *feature* se ha comenzado haciendo pruebas negativas en el formulario de creación de un registro de auditoría vacío. Para ello se ha enviado el formulario vacío y después uno por uno los campos con valores incorrectos, comprobando que se obtienen por pantalla los correspondientes mensajes de las validaciones implementadas. Finalmente se ha enviado el formulario de creación con valores correctos, realizando la prueba positiva correspondiente con variaciones de resultados positivos. Se ha logrado una cobertura de un 95.4% de código, quedando algunos warnings relacionados con las líneas de código donde se realizan comprobaciones de que un objeto no es nulo, las cuales no pueden ser suprimidas y por tanto se ignoran.

**auditor/audit-record/delete:** Para realizar esta prueba, se ha ingresado a un registro de auditoría aún no publicado y se ha eliminado, completando la prueba positiva de dicha *feature*. Para la prueba de hacking se ha intentado eliminar un registro de auditoría de otro usuario auditor obteniendo el correspondiente error 500.

En cuanto al porcentaje de cobertura, se ha obtenido un valor de 61.4%, quedando líneas marcadas en rojo pertenecientes al método unbind del servicio de borrado (*deleteService*) correspondiente, pues estas en ningún caso se ejecutan. Se ha mantenido el método.

**auditor/audit-record/list:** Para realizar esta prueba, se ha accedido al listado de registros de auditoría de un rol auditor comprobando que se lista adecuadamente, completando la prueba positiva de dicha *feature*. Para la prueba de hacking, se han intentado listar los registros de auditoría de otro rol auditor obteniendo el correspondiente error 500. Se ha logrado un porcentaje de cobertura de 93.8% de los servicios correspondientes, quedando warnings relacionados con la comprobación de si un objeto es nulo a través de *asserts*, los cuales se ignoran.

**auditor/audit-record/publish:** Para realizar esta prueba, se ha accedido a un registro de auditoría de un auditor en estado de *draftMode* o boceto, y se ha publicado satisfactoriamente, completando el test positivo. Para la prueba de hacking, se ha intentado publicar un registro de auditoría aún no publicado de otro auditor, obteniendo el correspondiente error 500. Se ha obtenido un porcentaje de cobertura de 95.8% del código relacionado con el servicio de publicar un registro de auditoría, quedando warnings señalados debido a las comprobaciones de objetos nulos a través de *asserts,* los cuales se ignoran.

**auditor/audit-record/show:** Para realizar esta prueba, se ha accedido al listado de registros de auditoría de un rol auditor y se ha ingresado a una de estas, comprobando que se muestra adecuadamente, completando la prueba positiva de dicha *feature*. Para la prueba de hacking, se han intentado mostrar registros de auditoría de otro rol auditor obteniendo el correspondiente error 500. Se ha conseguido un porcentaje de cobertura de 97.1% del código relacionado con mostrar un registro de auditoría. Solo han quedado algunos warnings marcando comprobaciones de que un objeto no es nulo a través de *asserts*, los cuales se ignoran.

**auditor/audit-record/update:** Para realizar las pruebas de esta *feature* se ha comenzado haciendo pruebas negativas en el formulario de de un registro de auditoría previamente creado. Para ello se ha enviado el formulario vacío y después uno por uno los campos con valores incorrectos, comprobando que se obtienen por pantalla los correspondientes mensajes de las validaciones implementadas. Finalmente se ha enviado el formulario de creación con valores correctos, realizando la prueba positiva correspondiente. Para la prueba de hacking se ha intentado actualizar un registro de auditoría de otro auditor, obteniendo el correspondiente error 500. Se ha conseguido un 95.7% de cobertura del código relacionado con el servicio de actualizar un registro de auditoría. Solo han quedado algunos warnings relacionados con comprobaciones de que un objeto no es nulo a través de *asserts*, los cuales se ignoran.

**Rol Auditor sobre entidad Code Audit (auditoría de código)**

**auditor/code-audit/create:** Para realizar las pruebas de esta *feature* se ha comenzado haciendo pruebas negativas en el formulario de creación de una auditoría de código vacía. Para ello se ha enviado el formulario vacío y después uno por uno los campos con valores incorrectos, comprobando que se obtienen por pantalla los correspondientes mensajes de las validaciones implementadas. Finalmente se ha enviado el formulario de creación con valores correctos, realizando la prueba positiva correspondiente. Se ha alcanzado un porcentaje de cobertura del código de 93.5%. Solo han quedado warnings relacionados con comprobaciones de un objeto nulo a través de *asserts*, los cuales se ignoran.

**auditor/code-audit/delete:** Para realizar esta prueba, se ha ingresado a una auditoría de código aún no publicada y se ha eliminado, completando la prueba positiva de dicha *feature*. Para la prueba de hacking se ha intentado eliminar una auditoría de código de otro usuario auditor obteniendo el correspondiente error 500. Se ha alcanzado un porcentaje de cobertura de 59.7%. Esto se debe a que nunca se prueba el método *unbind* del servicio relacionado al tratarse de un servicio de borrado.

**auditor/code-audit/list-mine:** Para realizar esta prueba, se ha accedido al listado de auditorías de código de un rol auditor comprobando que se lista adecuadamente, completando la prueba positiva de dicha *feature*. Para la prueba de hacking, se han intentado listar las auditorías de código de otro rol auditor obteniendo el correspondiente error 500. Se ha obtenido un porcentaje de cobertura de código de 97.9%. Solo han quedado warnings relacionados con comprobaciones de objetos nulos a través de *asserts*, los cuales se ignoran.

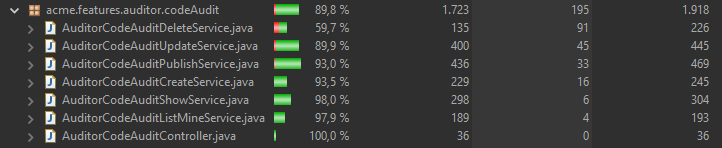
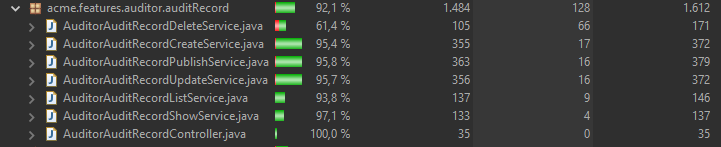
**auditor/code-audit/publish:** Para realizar esta prueba, se ha accedido a una auditoría de código de un auditor en estado de *draftMode* o boceto, y se ha publicado satisfactoriamente, completando el test positivo. Para la prueba de hacking, se ha intentado publicar una auditoría de código aún no publicada de otro auditor, obteniendo el correspondiente error 500. Se ha obtenido un porcentaje de cobertura de 93%. Solo han quedado warnings marcando comprobaciones de un objeto nulo a través de *asserts*, los cuales se ignoran.

**auditor/code-audit/show:** Para realizar esta prueba, se ha accedido al listado de auditorías de código de un rol auditor y se ha mostrado una de estas, comprobando que se muestra adecuadamente, completando la prueba positiva de dicha *feature*. Para la prueba de hacking, se han intentado mostrar auditorías de código de otro rol auditor obteniendo el correspondiente error 500. Se ha alcanzado un porcentaje de cobertura de código del 98%, quedando solo algunos warnings relacionados con comprobaciones de objetos nulos a través de *asserts*, los cuales se ignoran.

**auditor/code-audit/update:** Para realizar las pruebas de esta *feature* se ha comenzado haciendo pruebas negativas en el formulario de de una auditoría de código previamente creada. Para ello se ha enviado el formulario vacío y después uno por uno los campos con valores incorrectos, comprobando que se obtienen por pantalla los correspondientes mensajes de las validaciones implementadas. Finalmente se ha enviado el formulario de creación con valores correctos, realizando la prueba positiva correspondiente. Para la prueba de hacking se ha intentado actualizar una auditoría de código de otro auditor, obteniendo el correspondiente error 500. Se ha alcanzado una cobertura de código de 89.9%, quedando solo warnings relacionados con comprobaciones de objetos nulos a través de *asserts*, los cuales se ignoran.

Para comprobar los detalles sobre el tiempo de ejecución y los resultados de las pruebas realizadas, así como los datos originales de los cuales se ha extraído la información de este capítulo, puede consultar el documento adjunto llamado ***tester-performance-clean.xlsx*** enviado junto a la entrega ***D04: testing performance***.

Por otro lado, las pruebas realizadas en modo ***cobertura*** han mostrado un porcentaje de cobertura adecuado a las pruebas realizadas, el cual quedaría resumido en las siguientes capturas.



Lo único que podría ayudar a mejorar el porcentaje sin añadir más código sería suprimir o reducir las líneas del método *unbind* en ambos servicios de borrado (borrar registro de auditoría y borrar auditoría de código), acción la cual no se ha llevado a cabo porque la diferencia de porcentaje de cobertura sería insignificante.

# Capítulo II: Testing de rendimiento

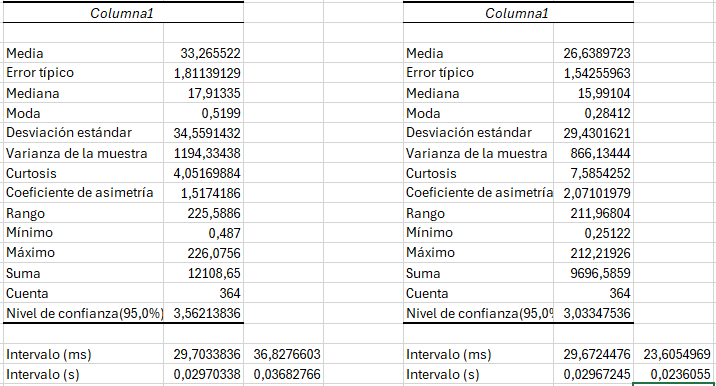
En este capítulo se proporcionarán gráficos adecuados y un intervalo de confianza del 95 % para el tiempo que tarda el proyecto en atender las solicitudes de las pruebas funcionales en dos equipos diferentes, además de un contraste de hipótesis de confianza del 95 % con respecto a cuál es el equipo más potente.

**Intervalo de confianza**

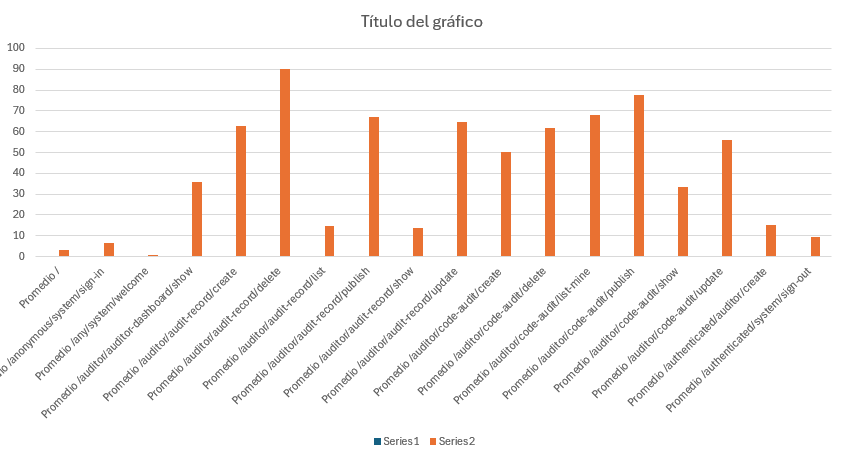
Para este apartado se han recolectado 365 datos de los distintos tests y se ha calculado el **intervalo de confianza con un nivel del 95%** en dos máquinas distintas. Los recursos hardware de dichas máquinas pueden contemplarse en las siguientes tablas.

| COMPONENTES | PC 1 | PC 2 |
| --- | --- | --- |
| CPU | Intel Core i5-8250U | AMD Ryzen 7 2700 Eight-Core |
| RAM | 8 GB DDR3 | 16 GB DDR4 |
| MEMORIA | 120 GB SSD | 500GB M.2 |
| GRÁFICA | Nvidia GeForce MX130 | Sapphire radeon RX 470 |

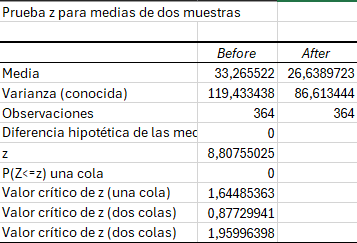
Los primeros tests se ejecutaron en el *PC 1*, obteniendo los resultados antes, y después se volvieron a ejecutar en el *PC 2*. *El PC 2* cuenta con mejores recursos hardware que el *PC 1*, por lo que debería suponer un cambio en cuanto a los tiempos de ejecución de las pruebas respecto al *PC 1*. *El PC 2* añadió índices “*@Table”* para intentar mejorar los tiempos de consulta de las queries.La siguiente captura muestra el resumen del cálculo estadístico de los intervalos de confianza al 95% realizados en ambas máquinas.



Como muestra el intervalo en segundos al final de cada tabla, se ha producido cierta **disminución en los tiempos**, pues en el *PC 2* (segunda tabla) se muestra un intervalo menor de tiempo gracias a la mejora hardware.

Los tests ejecutados en el *PC 1* mostraron que la función: **borrar un registro de auditoría era la que más tardaba en ejecutarse (MIR)**, tal y como muestra el siguiente gráfico del tiempo (en ms) de cada feature probada.

**Contraste de hipótesis**

Para comparar adecuadamente los intervalos de confianza calculados en ambas máquinas tras ejecutar las pruebas, se ha considerado realizar la prueba Z-Test sobre las columnas de tiempos generados por ambas máquinas (antes y después), quedándonos con el valor del campo de la tabla generada llamado ***Valor crítico de z (dos colas)*** ***= 0,87729941.*** 

Dicho valor crítico de z nos indica que nos encontramos en el intervalo *( a - 1.00 ]*, donde, ***a*** *= alpha = 1 - Intervalo de Confianza = 1 - 0.95 =* ***0.05***

Al ser z = 0,8773 tenemos que: *a < z <= 1.00*

Esta información nos está revelando que las segundas pruebas ejecutadas en el *PC 2* **no resultaron en ninguna mejora significativa** con respecto a las pruebas ejecutadas en el *PC 1*. Los tiempos de muestreo son diferentes, pero son globalmente los mismos. Por lo tanto, a pesar de añadir índices a las entidades durante la fase de testing del *PC 2* y de mejorar la potencia de cómputo, los tests terminaron con los mismos resultados. De esta manera podemos garantizar al menos que el proceso de testing realizado refleja los resultados obtenidos, pues el *PC 2* confirma los mismos resultados, aunque con unos tiempos diferentes.

Tras los resultados obtenidos, no ha sido necesario realizar un estudio del rendimiento del PC 1 puesto que ya arrojaba buenos valores en el tiempo de ejecución de las pruebas.

Los cálculos detallados del análisis estadístico de este capítulo puede comprobarse en el documento adjunto a esta entrega llamado ***tester-performance-clean.xlsx***

# Bibliografía

-Documento 08 Annexes, de los contenidos de la plataforma virtual de esta asignatura.

-S02 Performance testing, transparencias del módulo de testing L04 de la asignatura