Universidad de Sevilla  
Escuela Técnica Superior de Ingeniería Informática



Grado en Ingeniería Informática – Ingeniería del Software  
Diseño y Pruebas 2

Curso 2023 – 2024

**Reporte de análisis estudiante 4**

**Nombre completo**

**correo**

**Repositorio:** [https://github.com/DP2-C1-013/Acme-SF-D01-24.1.0](https://github.com/DP2-C1-013/Acme-SF)

| **Fecha** | **Versión** |
| --- | --- |
| 26/05/2024 | v1.0 |

[**1. Resumen 2**](#_4idpezggeose)

[**2. Control de versiones 3**](#_kuytbm87rdrm)

[**3. Introducción 3**](#_7js13dvc08tx)

[**4. Contenidos 3**](#_r1vl5b5325v9)

[4.1 Testing funcional 3](#_ciwqr5jxqhwz)

[4.2 Testing de rendimiento 6](#_f7qrc9ry3jkx)

[**5. Conclusión 9**](#_phy2ya37znhj)

[**6. Bibliografía 9**](#_78o4gtq9n3uv)

# 1. Resumen

En este documento dedicado al testing realizado para el proyecto Acme SF, se dan explicaciones claras y concisas de los tests de cada feature, haciendo especial hincapié en cómo de efectivos han sido a la hora de detectar bugs.

Adicionalmente, se incluye una sección dedicada al testing de rendimiento, donde se analiza el tiempo que las peticiones, de la parte de testing funcional, tardan en realizarse. Para facilitar su lectura, se dan gráficas apropiadas.

# 2. Control de versiones

| Nº de revisión | Fecha | Descripción |
| --- | --- | --- |
| 1 | 26/05/2024 | Descripción de los tests funcionales y análisis de rendimiento |

# 

# 3. Introducción

Como se ha explicado, en este documento vamos a hacer un análisis de todos los tests funcionales, agrupados por feature e indicando, en cada caso, cómo de útil han sido para la detección de bugs: fallos, reglas de negocio que no se comprobaban…

En segundo lugar, tenemos un análisis del rendimiento, usando para ello gráficas que representan el tiempo promedio de cada feature.

# 4. Contenidos

### 4.1 Testing funcional

Es crucial entender la relación entre code audit y audit record antes de analizar los tests. Esta relación ya ha sido explicada en el informe de análisis, pero, en resumidas cuentas consiste en que code audit es la entidad padre, que puede tener ninguno, uno o varios audit records, que es la entidad hija. La direccionalidad es desde audit record a code audit, con un manyToOne

También cabe destacar que, todos los tests que se van a describir a continuación, se han llevado a cabo ejecutando el tester#recorder, con la intención de luego poder repetirlos lanzando el tester#replayer, desde la pestaña de coverage. La cobertura ha sido superior al 90% tanto para code audit, como para audit record.

Por último, es considero oportuno notar que todos los bugs que se mencionan a continuación se han corregido.

Empecemos describiendo los tests realizados para probar cada feature, incluyendo qué bugs se encuentran.

Tests para **code audit:**

* List: el test más simple, que ha consistido de acceder al listado de code audits del auditor 1 y del auditor 2. El .hack ha consistido en, desde la cuenta de un sponsor, intentar acceder a un code-audit/list-mine.
* Show: también bastante más simple que el resto que se describirán a continuación. De nuevo, por cada auditor, 1 y 2, se ha cargado el formulario de show de cada code audit.

Se detecta que no siempre salía correctamente el atributo mark (derivado de sus audit records)

El .hack ha comprobado que no se pueda acceder a los detalles de un code audit desde cualquier cuenta que no sea a la que pertenece dicha entidad.

* Create: este es el primer test en el que se han comprobado validaciones. Desde un auditor cualquiera, se ha abierto el formulario de creación de code audit y se ha probado a mandarlo con todas las posibles combinaciones erróneas en cada campo (procedimiento el cual se ha repetido mucho en el resto de tests). Aquí se han detectado dos bugs importantes: no se comprobaba adecuadamente que el campo code fuese único, es decir, no saltaba ningún error si se intentaba crear un code audit con un código que ya existiese, y, por otro lado, no se comprobaba que la fecha de creación fuese igual o posterior al 2000 (se permitían fechas desde el 1 d.c)

También se descubre que no se ha añadido un código de error al fichero de internacionalización de mensajes.

El .hack ha comprobado que no se puedan crear code audits desde cuentas que no sean auditor (en este caso se ha probado con sponsor1), se descubre un bug que permitía cambiar la propiedad derivada mark haciendo inspeccionar desde el navegador.

* Delete: en este tests se han borrado unos cuantos code audits, y ha servido hallar un bug bastante crucial: cuando se intentaba borrar un code audit que tenía algún audit record, su entidad hija, que sí estaba publicado, se lanzaba un 500. Se ha arreglado para que cuando se de esta casuística, solo se dejen borrar los code audits que no tengan audit records publicados.

En el .hack se ha comprobado que no se puedan borrar code audits que estén publicados, ni siquiera desde el usuario al que pertenecen, y que no se puedan borrar code audits, publicados o no publicados, desde un usuario que no sea el propietario

* Update: en este caso procedemos de forma similar a la descrita en el create, se accede al formulario de un code audit y se intenta actualizar probando para cada atributo todas las posibles combinaciones erróneas. Se han descubierto los mismos errores errores descritos en el create.

Se descubre que no se comprobaba que al actualizar, la nueva fecha de ejecución fuese anterior al primero de sus audit records.

El .hack ha comprobado que no se pueden actualizar code audits publicados por su propietario, y que no se puede actualizar code audits por usuarios que no sean su propietario.

* Publish: de nuevo, seguimos el procedimiento de probar por cada atributo todas las posibles combinaciones erróneas. Adicionalmente, se prueba a publicar un code audit que no tiene audit records publicados. Este test revela los mismos bugs que el create y el update.

El .hack prueba que no se pueda publicar un code audit que no esté publicado o que no tenga audit records publicados desde el usuario correcto, y que un usuario incorrecto no pueda publicar ningún code audit que no sea suyo.

Tests para **audit record:**

* List: se listan, por cada code audit de cada auditor, todos lo audit records que le pertenecen. El .hack prueba que no se pueda acceder a las listas de los audit records de los code audits que no le pertenezcan al usuario logueado. No se detectan bugs.
* Show: en cada una de las listas de audit records que se han mostrado para el test anterior, se entra en los detalles de cada audit record. El .hack comprueba que no se pueda acceder a cada uno de estos audit records desde un usuario que no sea el propietario.
* Create: de nuevo, para los formularios de los audit records también seguimos el procedimiento descrito anteriormente, consistente en, por cada propiedad, probar todas las combinaciones erróneas posibles. Se descubren múltiples fallos en las comprobaciones de las fechas: se estaba comprobando únicamente que la fecha de inicio sea 1 hora anterior a la fecha de fin, pero no se comprobaba que la fecha de inicio fuese posterior o igual a la fecha de ejecución del code audit al que pertenece, ni que fuese al menos una hora anterior al presente. En la fecha de fin, no se comprobaba que fuese al menos una hora posterior a la fecha de ejecución del code audit.

El .hack detecta un bug: se pueden crear audit records para un determinado code audit desde un auditor cualquiera, aunque no sea su propietario. Por otro lado, se comprueba satisfactoriamente que no se pueden crear audit records desde cuentas que no sean auditor.

* Delete: este es más simple al delete de code audit, pues audit record no tiene hijos. Se prueba a borrar varios audit records y no se encuentran bugs.

El .hack prueba que no se puedan borrar audit records publicados desde el propietario, y que no se puedan borrar audit records desde un usuario que no sea el propietario.

* Update: mismo procedimiento que en el create, y se descubren los mismos bugs.

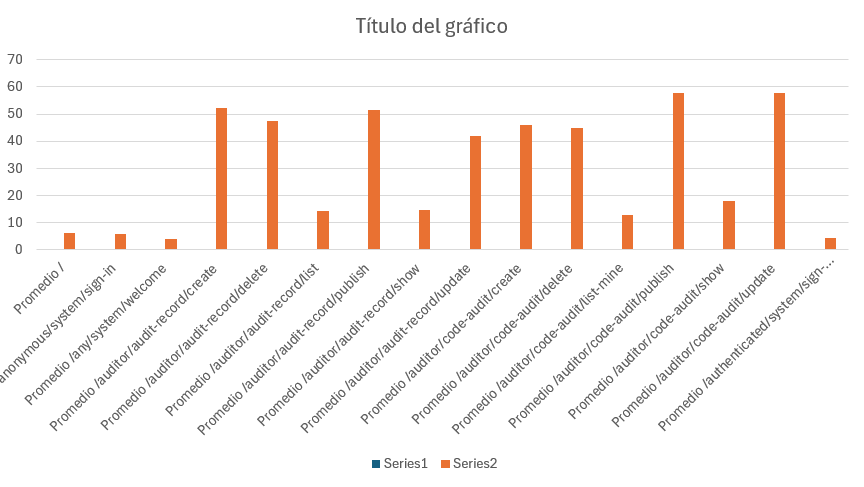
El .hack intenta actualizar audit records ya publicados desde el propietario y audit records no publicados desde usuarios que no son el propietario.

* Publish: de nuevo, mismo procedimiento y mismos descubrimientos de bugs que en el create. El .hack comprueba que no se puede publicar un audit record ya publicado o que incumpla restricciones desde el propietario, y que no se puedan publicar audit records que no están publicados y que no incumplen ninguna restricción desde un usuario que no sea el propietario.

Estos han sido todos los tests, con todos los descubrimientos de bugs para code audit y para audit record. Ahora se procede a analizar el rendimiento de estos tests:

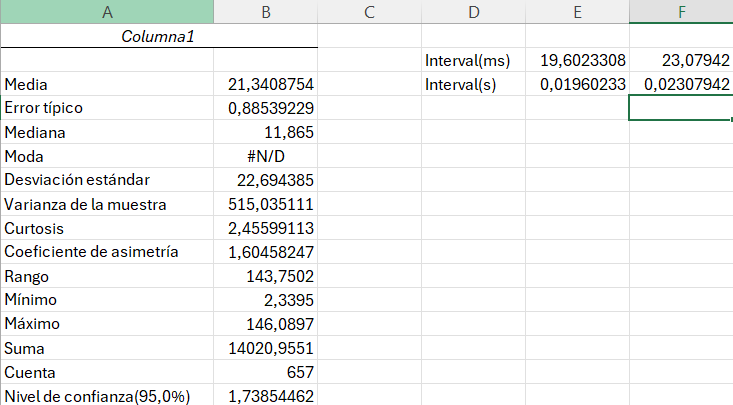
### 4.2 Testing de rendimiento

Tras realizar los tests explicados con el recorder, y luego, repetirlos todos con el replayer, se ha recopilado información clave relacionada con el tiempo sobre ellos:



En el gráfico indicado se puede ver, en el eje y, el tiempo en milisegundos que toma, en promedio, cada tipo de petición, en el eje x.

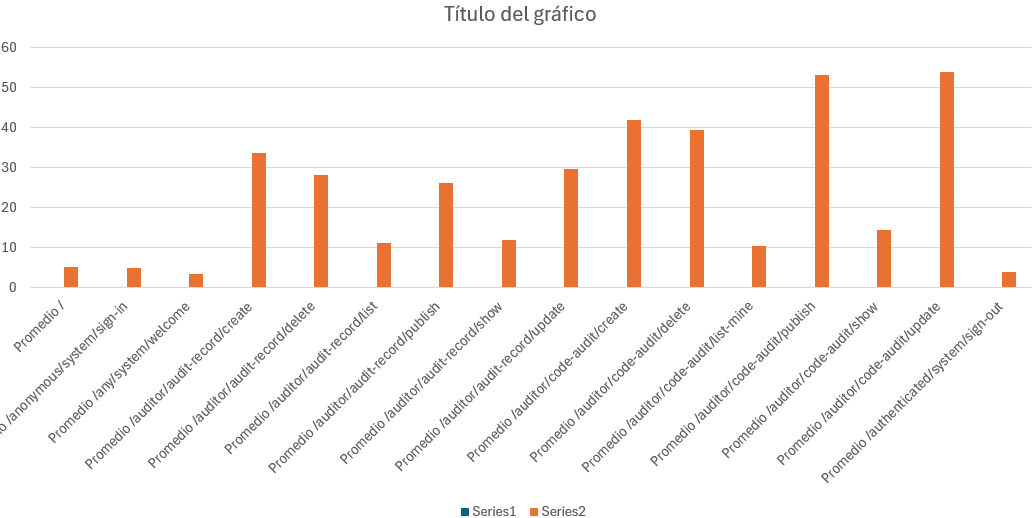
Como vemos, los MIR, Most Inefficient Request, son: audit-record/create, audit-record/publish, code-audit/publish y code-audit/update.



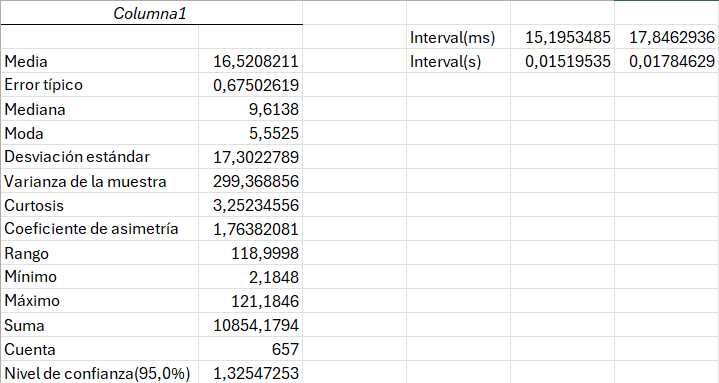
Aquí podemos ver el intervalo de confianza al 95% (19.6023308, 23,07942) y algunos datos estadísticos de los tiempos analizados con los tests.

Ahora, vamos a intentar mejorar el rendimiento para el MIR update de code audit, que es el que peores tiempos muestra.

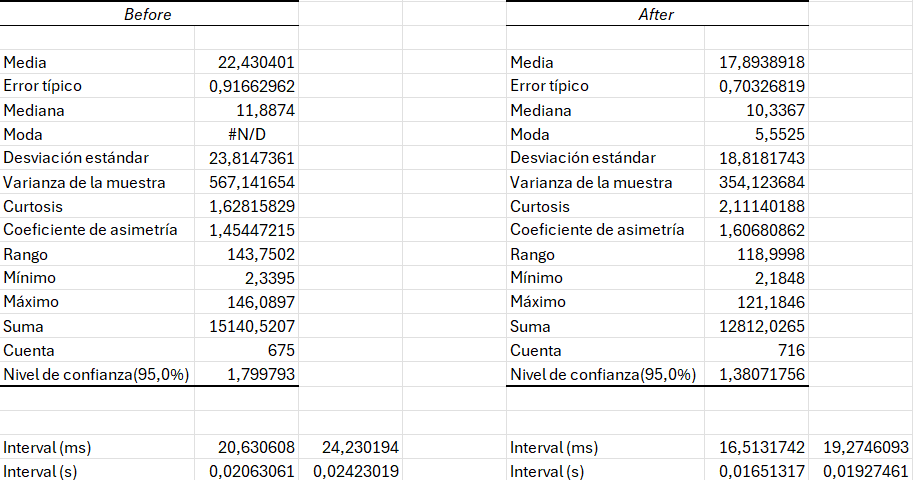
Datos de rendimiento tras implementar la mejora de los índices:



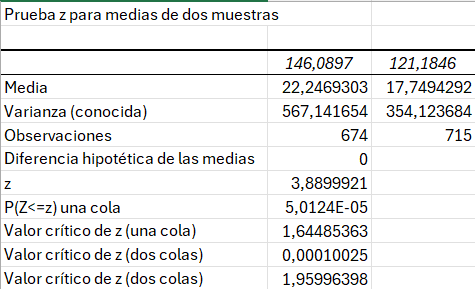
Con su intervalo de confianza:



Como se ve, el intervalo de confianza tras la mejora es (15.1953485 , 17.8462936)



Para tener una mejor perspectiva que nos permita comparar los dos rendimientos de forma más objetiva, vamos a realizar la prueba Z:



Como se ve, el valor P está contenido en el rango [0,0.05], es decir, es menor que alpha (que es 1-0.95, la confianza del intervalo, luego los cambios implementados han influido al rendimiento.

El valor crítico de z, a una y dos colas, nos indica que ha habido una mejora significativa y positiva en el rendimiento.

La media ha mejorado, ha pasado de 22.24 ms a 17.75 ms

La varianza ha disminuido, siendo los tiempos de respuesta más consistentes.

# 5. Conclusión

Tras haber realizado tests para cada feature, se han encontrado una amplia variedad de bugs, lo que ha sido muy importante, porque, en ocasiones, estos bugs suponían un fallo grave para la seguridad del sistema, o no aseguraban el cumplimiento de las reglas de negocio.

Por otro lado, el análisis de rendimiento ha mostrado un resultado muy positivo tras la implementación de los índices. Como se ha visto en la prueba Z, ha habido una mejora sensible.

# 6. Bibliografía

Intencionadamente en blanco.