

Testing report

Tabla de revisiones

Nº de revisión	Fecha	Descripción
1	27/05/24	Primera versión del documento
2	06/07/24	Corrección entrega de Julio

Información general

Fecha: 26/05/24

Grupo: C2.006

Repositorio:

<https://github.com/Danielruizlopezcc/Acme-SF-D04>

Miembro:

Marco Padilla Gómez (
marpadgom1@alum.us.es)

Índice

[Resumen ejecutivo](#)

[Introducción](#)

[Contenidos](#)

[Testing funcional](#)

[Testing de rendimiento](#)

[Corrección de Julio](#)

[Conclusiones](#)

[Bibliografía](#)

Resumen ejecutivo

El proyecto Acme-SF de la asignatura Diseño y Pruebas II, es un proyecto con fines meramente educativos, con el que se busca mejorar las habilidades y trabajar como desarrolladores web. El objetivo es aprender a producir un sistema de información web típico de tamaño pequeño a mediano basándose en una especificación de requisitos informal y métodos y herramientas de potencia industrial.

Introducción

En el contexto del proyecto Acme-SF, se presenta el siguiente documento de testing, que tiene como objetivo presentar el informe de pruebas realizadas a los servicios de las clases Contract y ProgressLog. Se mostrarán todos los resultados obtenidos en el test funcional, donde se incluirán tanto casos de prueba negativos y positivos, así como los resultados obtenidos en la cobertura de código para cada uno de los servicios de estas clases. También se incluirá una sección para mostrar los resultados obtenidos en el test de rendimiento, acompañados de gráficas y tablas que permitirán visualizar de manera más clara todos los resultados.

Contenidos

Testing funcional

- Code audits
 - AuditorCodeAuditsListMineService
 - Safe testing: Para el testing legal de este servicio únicamente fue necesario mostrar las auditorías de código de un auditor, no había forma de realizar un test negativo por como estaba implementado el servicio
 - Hacking:
 - Listado de auditorías de código de un auditor sin estar logueado(Role: Anonymous)
 - Listado de auditorías de código de un auditor sin estar como auditor(Role: Developer)
 - Bugs: No se encontraron bugs en este servicio
 - AuditorCodeAuditsShowService
 - Safe testing: Para el testing legal de este servicio, como caso de prueba positivo se muestra una auditorías de código de un auditor, y no hay forma de realizar un test negativo por como estaba implementado el servicio.
 - Hacking:

- Mostrar auditorías de código de un auditor sin estar logueado(Role: Anonymous)
- Mostrar auditorías de código de un auditor logueado como otro auditor(Role: Auditor)
- Mostrar auditorías de código de un auditor sin estar como auditor(Role: Developer)
- Bugs: No se encontraron bugs en este servicio
- AuditorCodeAuditsCreateService
 - Safe testing:
 - Casos de prueba positivos: Crear auditorías de código con datos correctos.
 - Casos de prueba negativos: Creación de auditorías de código probando todas las restricciones de los campos:
 - Formulario vacío
 - Campos con valores incorrectos, intentando probar todas las restricciones especificadas tanto por las anotaciones como por la lógica del método validate (por ejemplo, códigos duplicados, fechas no permitidas, etc.)
 - Hacking:
 - Crear auditorías de código sin estar logueado(Role: Anonymous)
 - Crear auditorías de código estando logueado con otro role(Role: Developer)
 - Bugs: No se encontraron bugs en este servicio
- AuditorCodeAuditsUpdateService
 - Safe testing:
 - Casos de prueba positivos: Actualizar auditorías de código.
 - Casos de prueba negativos: Actualización de auditorías de código actualizando con datos no permitidos.
 - Hacking:

- Actualizar auditorías de código publicadas por otro auditor(Role: Auditor)
- Actualizar auditorías de código no publicadas por otro auditor(Role: Auditor)
- Actualizar auditorías de código publicadas del auditor que actualiza(Role: Auditor)
- Bugs: No se encontraron bugs en este servicio
- AuditorCodeAuditsDeleteService
 - Safe testing:
 - Casos positivos: Eliminar auditorías de código.
 - Hacking:
 - Eliminar auditorías de código publicadas por otro auditor(Role: Auditor)
 - Eliminar auditorías de código no publicadas por otro auditor(Role: Auditor)
 - Eliminar auditorías de código publicadas del auditor que elimina(Role: Auditor)
 - Bugs: No se encontraron bugs en este servicio
- AuditorCodeAuditsPublishService
 - Safe testing:
 - Casos de prueba positivos: Publicar auditorías de código
 - Casos de prueba negativos: Publicar auditorías de código probando las restricciones de los campos:
 - Formulario vacío
 - Campos con valores incorrectos, intentando probar todas las restricciones especificadas tanto por las anotaciones como por la lógica del método validate (se siguieron los mismos 3 casos que en el servicio de creación de auditorías de código y actualización de auditorías de código)
 - Hacking:
 - Publicar auditorías de código de otro auditor(Role: Auditor)

- Bugs: No se encontraron bugs en este servicio
- Cobertura Code Audits

A continuación se muestra el porcentaje de cobertura de sentencias de todos los servicios de la clase Code Audits:

acme.features.auditor.codeAudits	87,3 %	1.227	178	1.405
> AuditorCodeAuditsDeleteService.java	56,4 %	132	102	234
> AuditorCodeAuditsPublishService.java	91,9 %	316	28	344
> AuditorCodeAuditsUpdateService.java	92,6 %	252	20	272
> AuditorCodeAuditsCreateService.java	92,4 %	206	17	223
> AuditorCodeAuditsShowService.java	96,2 %	150	6	156
> AuditorCodeAuditsListMineService.java	96,4 %	135	5	140
> AuditorCodeAuditsController.java	100,0 %	36	0	36

Siendo la cobertura total del paquete acme.features.auditor.codeAudits un 87,3%

En líneas generales la cobertura de código de los servicios de la clase CodeAudits es bastante buena, las únicas líneas de código en las que no se llega se deben a una restricción quizás excesiva en la autorización de los métodos en las que se contemplan casos que sin usar herramientas externas, como por ejemplo Postman, no se pueden probar, por eso hay ciertas branches que no se llegan a cubrir, al igual que con líneas encargadas de la internacionalización al español. También una línea de código que encontramos en todos los métodos.

```
assert object != null
```

Esta línea de código no se llega a cubrir del todo en ningún método, pero esta presente en todas las plantillas de los servicios proporcionados en la metodología de la asignatura y se consultó con el cliente y no se consideró necesario cubrirla del todo.

- Cobertura Code Audits tras la corrección de Julio

acme.features.auditor.codeAudits	87,6 %	1.219	172	1.391
> AuditorCodeAuditsDeleteService.java	64,1 %	164	92	256
> AuditorCodeAuditsPublishService.java	91,6 %	272	25	297
> AuditorCodeAuditsUpdateService.java	92,1 %	233	20	253
> AuditorCodeAuditsCreateService.java	91,7 %	188	17	205
> MarkMode.java	92,7 %	89	7	96
> AuditorCodeAuditsShowService.java	96,2 %	150	6	156
> AuditorCodeAuditsListMineService.java	94,6 %	87	5	92
> AuditorCodeAuditsController.java	100,0 %	36	0	36

Ha mejorado un poco en líneas generales, donde mas se nota es en el servicio de delete, que ha subido un 10%.

- Audit records
 - AuditorAuditRecordsListService
 - Safe testing: Para el testing legal de este servicio únicamente fue necesario mostrar los registros de auditoría de un auditor, no habia forma de realizar un test negativo por como estaba implementado el servicio
 - Hacking:
 - Listado de registros de auditoría de un auditor sin estar logueado(Role: Anonymous)
 - Listado de registros de auditoría de un auditor sin estar logueado con el auditor al que pertenece la auditoría de código(Role: Auditor)
 - Listado de registros de auditoría de un auditor estando logueado con otro rol(Role: Developer)
 - Bugs: No se encontraron bugs en este servicio
 - AuditorAuditRecordsShowService
 - Safe testing: Para el testing legal de este servicio, como caso de prueba positivo se muestra un registros de auditoría de un auditor, y no hay forma de realizar un test negativo por como estaba implementado el servicio.
 - Hacking:
 - Mostrar registros de auditoría de un auditor sin estar logueado(Role: Anonymous)

- Mostrar registros de auditoría de un auditor sin estar logueado con el auditor al que pertenece la auditoría de código(Role: Auditor)
- Mostrar registros de auditoría de un auditor logueado como otro auditor(Role: Auditor)
- Bugs: No se encontraron bugs en este servicio
- AuditorAuditRecordsCreateService
 - Safe testing:
 - Casos de prueba positivos: Crear registros de auditoría con datos correctos.
 - Casos de prueba negativos: Creación de auditorías de código probando todas las restricciones de los campos:
 - Formulario vacío
 - Campos con valores incorrectos, intentando probar todas las restricciones especificadas tanto por las anotaciones como por la lógica del método validate (por ejemplo, códigos duplicados, fechas no permitidas, etc.)
 - Hacking:
 - Crear registros de auditoría sin estar logueado(Role: Anonymous)
 - Crear registros de auditoría de una auditoría de código de otro auditor(Role: Auditor)
 - Crear registros de auditoría de una auditoría de código logueado con un rol diferente(Role: Developer)
 - Bugs: No se encontraron bugs en este servicio
- AuditorAuditRecordsUpdateService
 - Safe testing:
 - Casos de prueba positivos: Eliminar registros de auditoría
 - Casos de prueba negativos: Actualización de registros de auditoría actualizando con datos no permitidos.
 - Hacking:

- Actualizar registros de auditoría publicados por otro auditor(Role: Auditor)
- Actualizar registros de auditoría no publicados por otro auditor(Role: Auditor)
- Actualizar registros de auditoría publicados del auditor que actualiza(Role: Auditor)
- Bugs: No se encontraron bugs en este servicio
- AuditorAuditRecordsDeleteService
 - Safe testing:
 - Casos positivos: Eliminar registro de auditoría.
 - Hacking:
 - Eliminar registro de auditoría publicados por otro auditor(Role: Auditor)
 - Eliminar registro de auditoría no publicados por otro auditor(Role: Auditor)
 - Eliminar registro de auditoría publicados del auditor que elimina(Role: Auditor)
 - Bugs: No se encontraron bugs en este servicio
- AuditorAuditRecordsPublishService
 - Safe testing:
 - Casos de prueba positivos: Publicar registro de auditoría
 - Casos de prueba negativos: Publicar registro de auditoría probando las restricciones de los campos:
 - Formulario vacío
 - Campos con valores incorrectos, intentando probar todas las restricciones especificadas tanto por las anotaciones como por la lógica del método validate (se siguieron los mismos 3 casos que en el servicio de creación de registro de auditoría y actualización de registro de auditoría)
 - Hacking:
 - Publicar registro de auditoría de otro auditor(Role: Auditor)

- Bugs: No se encontraron bugs en este servicio
- Cobertura auditRecords

A continuación se muestra el porcentaje de cobertura de sentencias de todos los servicios de la clase AuditRecords:

acme.features.auditor.auditRecords	94,6 %	1.342	77	1.419
> AuditorAuditRecordsPublishService.java	92,9 %	263	20	283
> AuditorAuditRecordsDeleteService.java	92,8 %	205	16	221
> AuditorAuditRecordsCreateService.java	94,9 %	262	14	276
> AuditorAuditRecordsUpdateService.java	95,3 %	282	14	296
> AuditorAuditRecordsListService.java	94,7 %	161	9	170
> AuditorAuditRecordsShowService.java	97,1 %	134	4	138
> AuditorAuditRecordsController.java	100,0 %	35	0	35

Siendo la cobertura total del paquete acme.features.auditor.auditRecords un 94.6%

De nuevo la cobertura sigue siendo buena, nos volvemos a encontrar con las mismas líneas de código que no se llegan a cubrir en todos los métodos, y la misma línea de código que no se llega a cubrir del todo en ningún método. Además estos métodos al tener pocas líneas de código gracias a la metodología de la asignatura y del framework, hace que un pequeño cambio en la cobertura la baje significativamente.

- Cobertura de auditRecords tras la corrección de Julio

acme.features.auditor.auditRecords	94,4 %	1.325	79	1.404
> AuditorAuditRecordsUpdateService.java	92,9 %	275	21	296
> AuditorAuditRecordsDeleteService.java	92,2 %	190	16	206
> AuditorAuditRecordsCreateService.java	94,9 %	262	14	276
> AuditorAuditRecordsPublishService.java	95,1 %	269	14	283
> AuditorAuditRecordsListService.java	94,1 %	160	10	170
> AuditorAuditRecordsShowService.java	97,1 %	134	4	138
> AuditorAuditRecordsController.java	100,0 %	35	0	35

La cobertura sigue siendo similar, hay servicios que tienen un porcentaje mayor y otros menor, pero siguen en la misma línea.

Testing de rendimiento

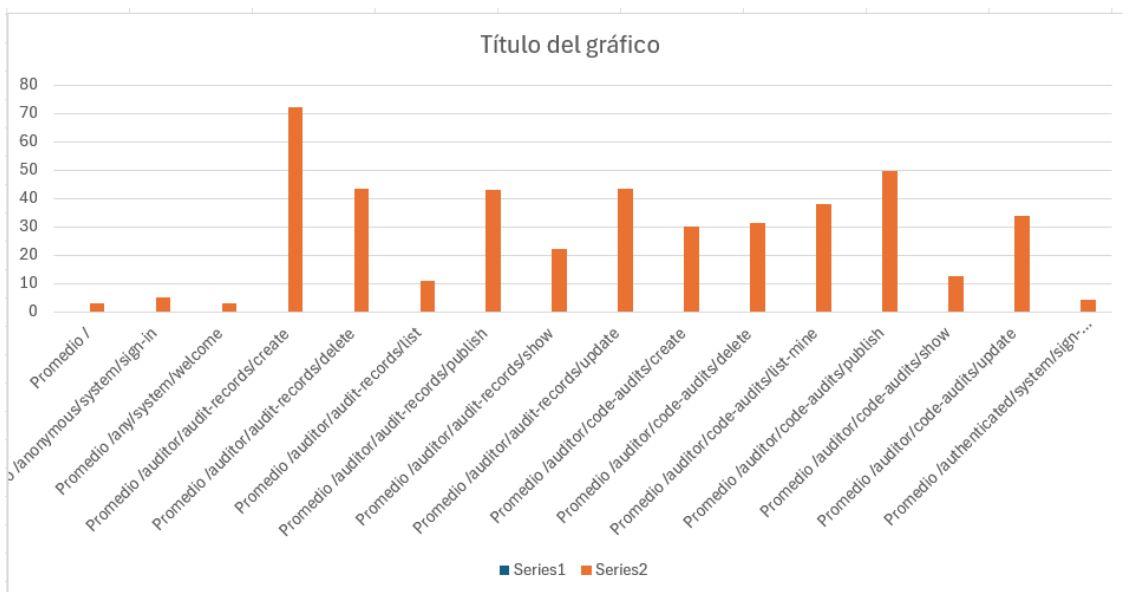
• Análisis de rendimiento

Tras la ejecución de los tests, y con los datos recopilados sobre las peticiones realizadas en los mismos, se llevó a cabo un análisis de los

resultados obtenidos en cuanto al rendimiento de la aplicación, tomando como referencia el tiempo de respuesta de los servicios, centrándonos en los servicios de las

clases CodeAudits y AuditRecords. A continuación se muestran los datos del promedio de cada petición, aunque para mayor claridad también se proporciona una gráfica de barras:

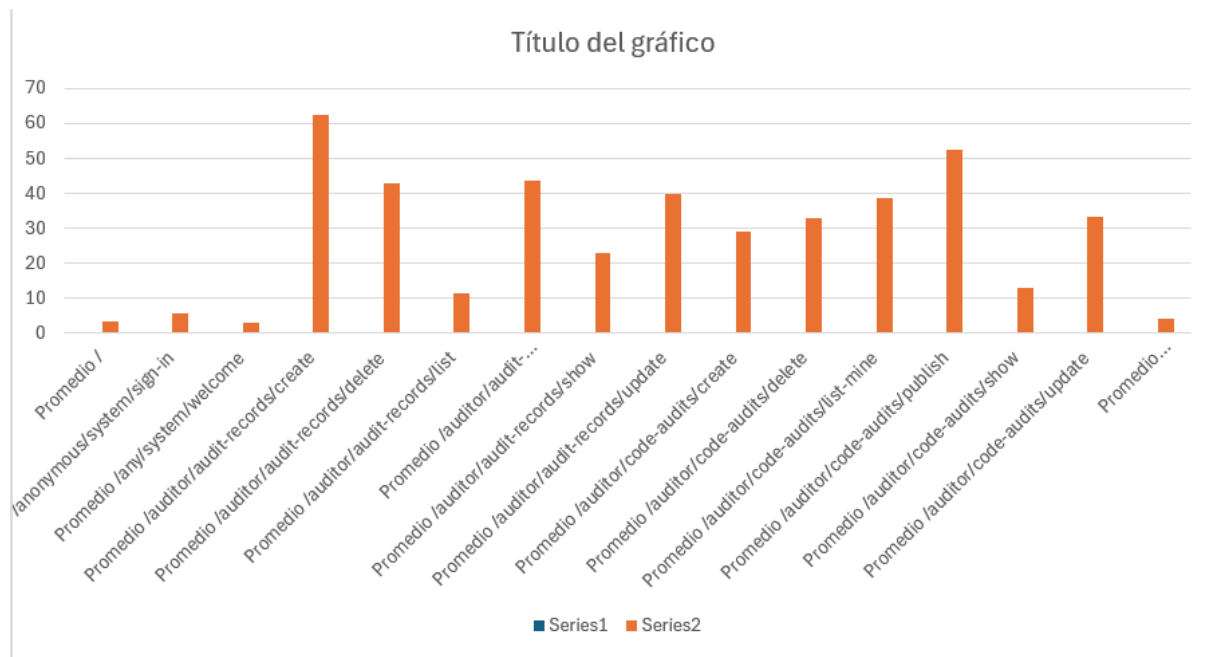
request-path	response-stat time
Promedio /	3.04562407
Promedio /anonymous/system/sign-in	5.32637692
Promedio /any/system/welcome	2.86026176
Promedio /auditor/audit-records/create	72.1438
Promedio /auditor/audit-records/delete	43.2688857
Promedio /auditor/audit-records/list	11.0478086
Promedio /auditor/audit-records/publish	43.0700613
Promedio /auditor/audit-records/show	22.3226636
Promedio /auditor/audit-records/update	43.2723438
Promedio /auditor/code-audits/create	30.3257481
Promedio /auditor/code-audits/delete	31.4274286
Promedio /auditor/code-audits/list-mine	38.186192
Promedio /auditor/code-audits/publish	49.66968
Promedio /auditor/code-audits/show	12.6938149
Promedio /auditor/code-audits/update	34.0798565
Promedio /authenticated/system/sign-out	4.0922
Promedio general	20.7896093



Como vemos en la gráfico, existe grandes diferencia entre los tiempos de respuesta de las peticiones de inicio de sesión o bienvenido, comparado con las features de ambas entidades que estamos analizando. En concreto nos centraremos en los servicios de la clase AuditRecords, que parece tener una de las peticiones más ineficientes y lentas de la aplicación, por ejemplo crear o actualizar una auditoría de código superan los 40 ms de promedio, 10 veces más que por ejemplo el inicio de sesión. El promedio de todas las respuestas es de 20 ms.

Esta identificación de los servicios más lentos fueron el punto de partida para optimizar la aplicación, y se llevaron a cabo una serie de cambios en el código, en concreto se añadieron índices a las tablas de la base de datos de la entidad AuditRecords, con el objetivo de mejorar el rendimiento de las peticiones. A continuación se muestra una gráfica de barras con los resultados obtenidos tras la optimización y los nuevos promedios de tiempo de respuesta:

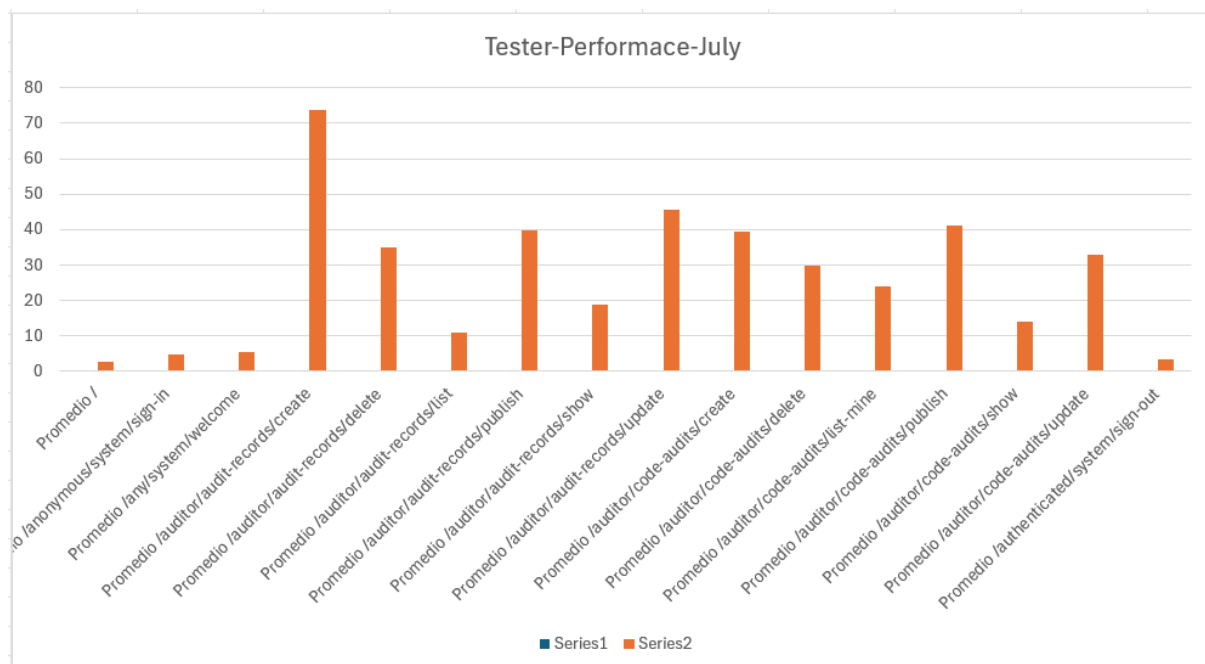
request-path	response-stat time
Promedio /	3.2358537
Promedio /anonymous/system/sign-in	5.76508462
Promedio /any/system/welcome	2.943475
Promedio /auditor/audit-records/create	62.3810875
Promedio /auditor/audit-records/delete	42.9030571
Promedio /auditor/audit-records/list	11.3789
Promedio /auditor/audit-records/publish	43.7217839
Promedio /auditor/audit-records/show	23.0183364
Promedio /auditor/audit-records/update	39.80275
Promedio /auditor/code-audits/create	29.1047593
Promedio /auditor/code-audits/delete	32.8183571
Promedio /auditor/code-audits/list-mine	38.7661333
Promedio /auditor/code-audits/publish	52.45499
Promedio /auditor/code-audits/show	13.0912223
Promedio /auditor/code-audits/update	33.3992217
Promedio /authenticated/system/sign-out	4.14963846
Promedio general	20.8159345



Observamos una infima diferencia en los tiempos de respueesta en prácticamente todas las peticiones, de hecho el promedio general ha subido, pero no es significativo.

Corrección de Julio

Promedio /	2.7157
Promedio /anonymous/system/sign-in	4.934785
Promedio /any/system/welcome	5.450074
Promedio /auditor/audit-records/create	73.85571
Promedio /auditor/audit-records/delete	34.8593
Promedio /auditor/audit-records/list	11.11974
Promedio /auditor/audit-records/publish	39.74373
Promedio /auditor/audit-records/show	18.91719
Promedio /auditor/audit-records/update	45.65469
Promedio /auditor/code-audits/create	39.5995
Promedio /auditor/code-audits/delete	29.69788
Promedio /auditor/code-audits/list-mine	24.07284
Promedio /auditor/code-audits/publish	41.2937
Promedio /auditor/code-audits/show	14.00064
Promedio /auditor/code-audits/update	33.00084
Promedio /authenticated/system/sign-out	3.432456
Promedio general	15.51984



Como observamos el promedio general se ha reduciode bastante, teniendo en cuenta que no se ha desoptimizado para volverlo a optimizar y compararlos, estamos comparandolo con la versión optimizada de la última entrega

- **Intervalo de confianza:**

Se analizó el intervalo de confianza de los tiempos de respuesta de las peticiones de la aplicación, con el objetivo de comprobar si la aplicación cumple con el requisito del rendimiento establecido: que el tiempo de respuesta de las peticiones no supere el segundo de media . Para ello se realizó un análisis de los datos obtenidos en los tests, y se calculó el intervalo de confianza de los tiempos de respuesta de las peticiones, con un nivel de confianza del 95%. La herramienta utilizada para el análisis ha sido excel, que proporciona un complemento llamado Herramientas para el análisis. A continuación se muestra el resultado obtenido:

Before	After								
5.4576	5.0184								
6.3196	5.7032								
4.6475	4.9664								
5.4022	3.4619								
5.0349	5.2789								
3.2099	3.1875								
4.4834	4.9165								
3.2588	2.9365								
4.0028	4.4748								
3.2554	3.702								
4.1768	5.9797								
3.1112	3.841								
3.2249	3.056								
4.1282	4.4639								
2.9513	3.4427								
4.5022	4.6324								
2.694	3.1362								
2.5927	3.043								
2.5617	3.2693								
2.8224	3.3314								
2.4627	2.7753								
2.49	2.6224								
2.8814	2.8053								
2.6135	3.4113								
		BEFORE				AFTER			
		Media	20.9600556			Media	20.9739639		
		Error típico	0.85779952			Error típico	0.78130884		
		Mediana	12.76915			Mediana	13.68935		
		Moda	2.9567			Moda	#N/D		
		Desviación es	22.2035901			Desviación es	20.2236779		
		Varianza de la	492.999415			Varianza de la	408.99715		
		Curtosis	71.4707381			Curtosis	31.092877		
		Coefficiente de	5.56167229			Coefficiente de	3.41606096		
		Rango	346.9827			Rango	260.9157		
		Mínimo	1.9918			Mínimo	2.0438		
		Máximo	348.9745			Máximo	262.9595		
		Suma	14043.2373			Suma	14052.5558		
		Cuenta	670			Cuenta	670		
		Nivel de confi	1.68430334			Nivel de confi	1.53411264		
							</		

- Intervalo de confianza tras la corrección de Julio

July					
Media	15.8463532	Intervalo(ms)	13.9212871	17.7714194	
Error típico	0.97996014	Intervalo(s)	0.01392129	0.01777142	
Mediana	8.4795				
Moda	2.3645				
Desviación es	22.6241367				
Varianza de la	511.85156				
Curtosis	46.772677				
Coefficiente de	5.2740556				
Rango	276.4641				
Mínimo	1.8293				
Máximo	278.2934				
Suma	8446.10628				
Cuenta	533				
Nivel de confia	1.92506617				

- **Hypothesis contrast**

Como se comentó anteriormente se ha llevado a cabo una refactorización mediante el uso de índices para aumentar el rendimiento de las queries de los repositorios asociados a las entidades CodeAudits y AuditRecords. Para comprobar si esta refactorización ha tenido un impacto en el rendimiento de la aplicación, se ha llevado a cabo un contraste de hipótesis, mas concretamente, se ha realizado nuevamente en excel con el complemento Herramientas para el análisis, un Z-test (con $\alpha = 0.05$) para evaluar si los nuevos tiempos de respuesta son mejores o peores y en que medida. Para poder realizar el Z-test, se ha debido de recopilar nuevamente los tiempos de respuesta tras la refactorización y obtener los datos estadísticos. A continuación se muestran los datos del análisis post-refactorización, así como los resultados del Z-test.

Prueba z para medias de dos muestras		
	<i>Before</i>	<i>After</i>
Media	20.9600556	20.97396386
Varianza (conocida)	492.999415	408.99715
Observaciones	670	670
Diferencia hipotética de las medias	0	
z	-0.011986919	
P(Z<=z) una cola	0.495218026	
Valor crítico de z (una cola)	1.644853627	
Valor crítico de z (dos colas)	0.990436052	
Valor crítico de z (dos colas)	1.959963985	

Como podemos observar, el valor crítico de z (dos colas) es 0.00, para saber si los cambios han sido significativos, hemos de comparar el valor z con α . En este caso, nuestro z se encuentra entre el intervalo $[0.00, \alpha)$, por lo que podemos afirmar que los cambios realizados no han mejorado mucho el rendimiento. Igualmente, esto se veía bastante claro en las imágenes de las gráficas de barras o en los promedios, la refactorización no ha servido de mucho.

Conclusiones

A lo largo del documento se ha llevado a cabo un análisis de los servicios de las clases CodeAudits y AuditRecords, en el que se han realizado tests funcionales y de rendimiento, con el objetivo de comprobar el correcto funcionamiento de los servicios y el rendimiento de la aplicación. En cuanto a los tests funcionales, se han realizado pruebas tanto positivas como negativas, y se ha comprobado que los servicios funcionan correctamente y no presentan bugs. En cuanto a los tests de rendimiento, se ha llevado a cabo un análisis de los tiempos de respuesta de las peticiones, y se ha comprobado que la aplicación cumple con el requisito de rendimiento establecido. Además, se ha llevado a cabo una refactorización de los servicios de las clases CodeAudits y AuditRecords, añadiendo índices a las tablas de la base de datos, con el objetivo de mejorar el rendimiento de la aplicación. Tras la refactorización, se ha llevado a cabo un contraste de hipótesis, y se ha comprobado que los cambios realizados no han mejorado el rendimiento de la aplicación.

En resumen, considero que he adquirido valiosas lecciones sobre el testing de aplicaciones web usando el Acme-Framework, el cual ha facilitado todo este proceso, también aprendí a utilizar la herramienta excel para un análisis más profundo de los datos obtenidos en los tests, cosa que no había hecho antes.

- Tras las correcciones de Julio se observan mejoras a nivel de rendimiento y de cobertura de código.

Bibliografía

- 08 Annexes.