Test Report-Student 5- D04



Integrantes del grupo C1.026:

* Ignacio Blanquero Blanco ([ignblabla@alum.us.es](mailto:ignblabla@alum.us.es))
* Adrián Cabello Martín ([adrcabmar@alum.us.es](mailto:adrcabmar@alum.us.es))
* María de la Salud Carrera Talaverón ([marcartal1@alum.us.es](mailto:marcartal1@alum.us.es))
* Joaquín González Ganfornina ([joagongan@alum.us.es](mailto:joagongan@alum.us.es))
* Natalia Olmo Villegas ([natolmvil@alum.us.es](mailto:natolmvil@alum.us.es))

Fecha: Sevilla, 27 de Mayo 2024

Contenido

[Resumen del Informe 3](#_Toc167744953)

[Historial de Versiones 4](#_Toc167744954)

[Introducción 5](#_Toc167744955)

[Testing funcional 6](#_Toc167744956)

[AuditorCodeAuditList.java 7](#_Toc167744957)

[AuditorCodeAuditShowService.java 8](#_Toc167744958)

[AuditorCodeAuditCreateService.java 9](#_Toc167744959)

[9](#_Toc167744960)

[AuditorCodeAuditUpdateService.java 10](#_Toc167744961)

[AuditorCodeAuditDeleteService.java 13](#_Toc167744962)

[AuditorCodeAuditController.java 14](#_Toc167744963)

[AuditorAuditRecordList.java 15](#_Toc167744964)

[AuditorAuditRecordShowService.java 16](#_Toc167744965)

[AuditorAuditRecordCreateService.java 17](#_Toc167744966)

[17](#_Toc167744967)

[AuditorAuditRecordUpdateService.java 18](#_Toc167744968)

[AuditorAuditRecordDeleteService.java 19](#_Toc167744969)

[AuditorAuditRecordController.java 20](#_Toc167744970)

[Rendimiento del testing 20](#_Toc167744971)

[Performance-testing sin índices 21](#_Toc167744972)

[Performance-testing con índices 22](#_Toc167744973)

[Comparación sin índices - con índices 22](#_Toc167744974)

[Conclusiones 23](#_Toc167744975)

[Bibliografía 24](#_Toc167744976)

# Resumen del Informe

En este documento se detalla la metodología empleada para llevar a cabo la tarea individual obligatoria, que consistió en realizar pruebas para los requisitos de los auditores.

El documento analiza la cobertura de cada clase involucrada para cumplir con los requisitos propuestos. En este informe, se presentarán los porcentajes de cobertura de las pruebas y se explicarán las razones de dichos porcentajes, apoyándose en capturas de pantalla de las clases analizadas.

Además, se realizará un análisis estadístico utilizando las herramientas aprendidas en la asignatura, evaluando el rendimiento de las pruebas. Este análisis permitirá valorar la calidad del trabajo y comparar los resultados de las pruebas antes y después de implementar algunas mejoras.

# Historial de Versiones

|  |  |  |  |
| --- | --- | --- | --- |
| **Versión** | **Contenidos** | **Fecha** | **Contribuyente** |
| V0.1 | Estructura principal | 26/05/2024 | Natalia Olmo Villegas |
| V1.1 | Análisis cobertura testing funcional | 27/05/2024 | Joaquín González Ganfornina |
|  |  |  |  |

# Introducción

En este documento, se explicará la metodología usada para resolver la tarea grupal obligatoria que consistía en realizar pruebas para la entidad Banner. Para resolver esta tarea, cada integrante del grupo participó activamente, con el objetivo de completarla sin mayores complicaciones.

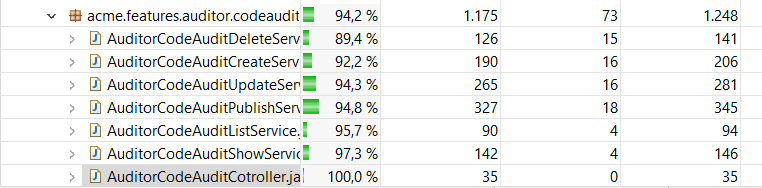
A continuación, se analizará la cobertura de cada clase que se complementa para satisfacer el requisito propuesto para esta entidad. Para ello, se mostrará el porcentaje de cobertura de las pruebas y además, apoyándose en capturas de pantallas de las clases analizadas, se explicará la razón de dichos porcentajes.

Además, con las herramientas vistas en la asignatura, se realizará un análisis estadístico que valora el rendimiento de las pruebas. Dicho análisis valorará la calidad del trabajo y comprará los resultados de las pruebas antes y después de realizar algunas mejoras.

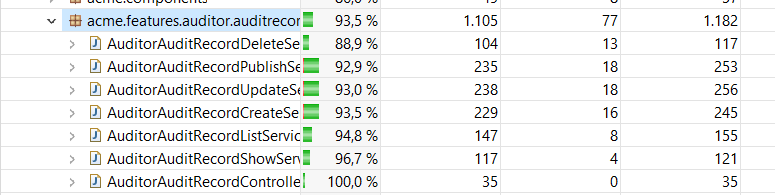
Este documento tiene una estructura simple. Primeramente, tenemos el índice, un breve resumen del contenido de dicho documento y un historial de versiones. Más adelante, tenemos esta introducción. Posteriormente, se analizarán los resultados de las pruebas realizadas en las clases. En el punto siguiente, se muestra el análisis estadístico de las pruebas realizadas antes y después de las mejoras. Finalmente, se realiza una pequeña conclusión y se muestra la bibliografía.

# Testing funcional

En el proceso de pruebas de la entidad *“CodeAudit.java”*, se ha logrado alcanzar un 94.2% de cobertura. Este indicador es muy positivo, pues destaca la exhaustividad de las pruebas realizadas. Dicha cobertura se traduce en que, de un total de 1248 instrucciones, 1175 han sido cubiertas y 73 no lo han sido.



Parael proceso de pruebas de la entidad *“AuditRecordt.java”*, se ha logrado alcanzar un porcentaje similar, de 93.5% de cobertura. Dicha cobertura se traduce en que, de un total de 1182 instrucciones, 1105 han sido cubiertas y 73 no lo han sido.



Es importante destacar que todos los tests incluyen líneas de código resaltadas en amarillo correspondientes a la instrucción *“assert object != null”*. Esta situación es completamente normal y esperada, ya que la cobertura de esta línea indicaría que el sistema ha encontrado un fallo irreversible durante los casos de prueba. Estas líneas están presentes como medida preventiva; sin embargo, es afortunado que no hayan sido ejecutadas durante las pruebas.

Es importante señalar que, en este proyecto, solo se han considerado posibles intentos de hacking que implican la inserción de URLs en la barra de búsqueda. Esto se debe a que no se disponía del conocimiento necesario para implementar otros tipos de pruebas de hacking en el contexto específico de este proyecto.

A continuación, se muestra más detallado la cobertura de test de cada feature implementada para esta entidad.

## AuditorCodeAuditList.java



Como se puede apreciar en las imágenes, la cobertura del listado de los code audits es casi completa. Lo único que destaca una línea subrayada en amarillo, aunque esto no es preocupante. Esta línea se corresponde con la instrucción “assert object !=null;” que ya se explicó previamente que es completamente normal.

## AuditorCodeAuditShowService.java



En esta clase se alcanzó una cobertura del 97,7%, destacando nuevamente la instrucción “assert object !=null;”.

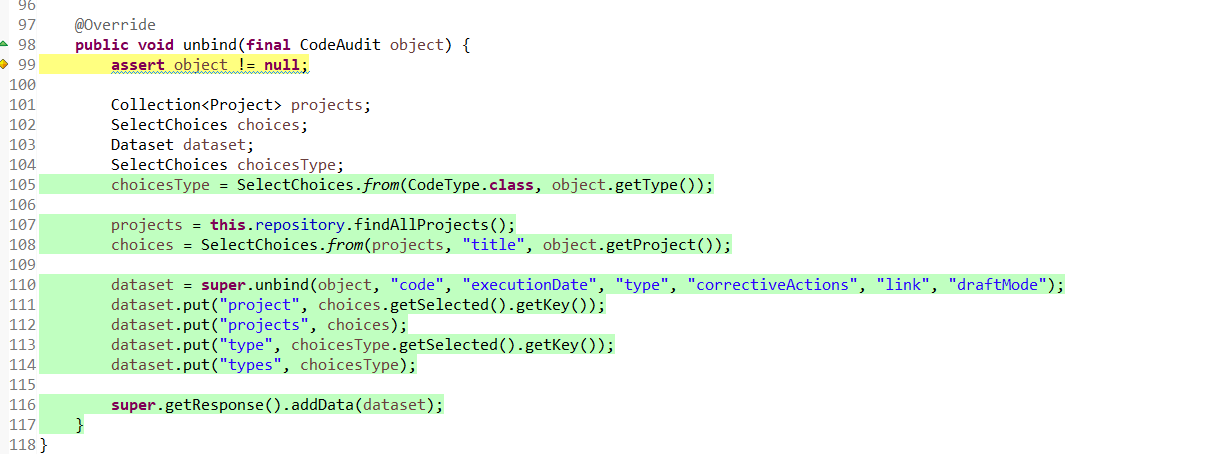
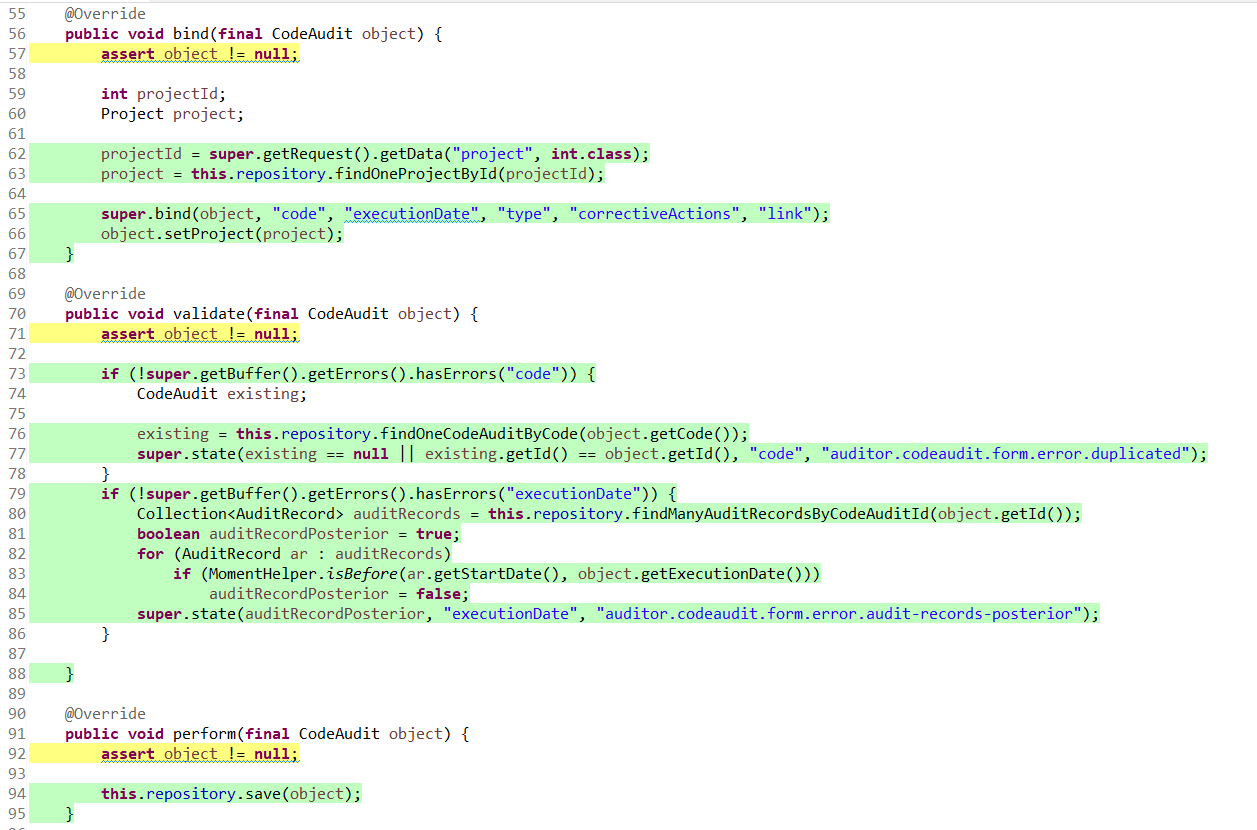
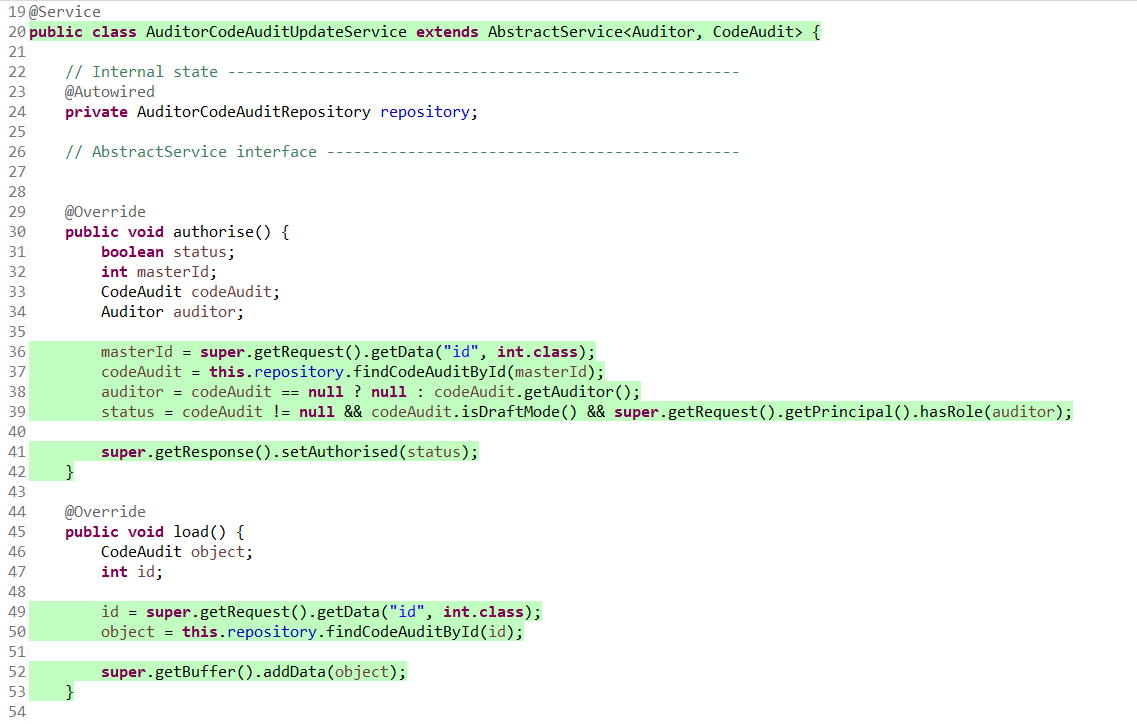
## AuditorCodeAuditCreateService.java

### 

Al igual que en las entidades anteriores, las únicas línas que podrían tener conflicto son las líneas relacionadas con la instrucción de “asser object!=null”.

### 

## AuditorCodeAuditUpdateService.java



En esta entidad se pueden ver más líneas amarillas a parte de las que se corresponden a la instrucción mencionada previamente. Las primeras líneas amarillas corresponden a la validación del usuario. Esto es probablemente debido a que no se realizaron las pruebas necesarias para comprobar la validación del usuario. Más adelante, vemos una instrucción que calcula la moda de las notas de los Audit records relacionados a un Code audit. Esto se puede deber a que no se comprobó alguno de los casos de la estructura condicional. Posteriormente, hay una instrucción que comprueba si la moda es distinta de null y si la moda es null se lanza una excepción. Puede que esta línea esté en amarillo porque no se llegó a comprobar esta instrucción. Finalmente vemos dos condicionales “if” que también están subrayados de amrillo. Esto puede deberse a que no se exploraron todos los resultados de dicho condicional.

## AuditorCodeAuditDeleteService.java



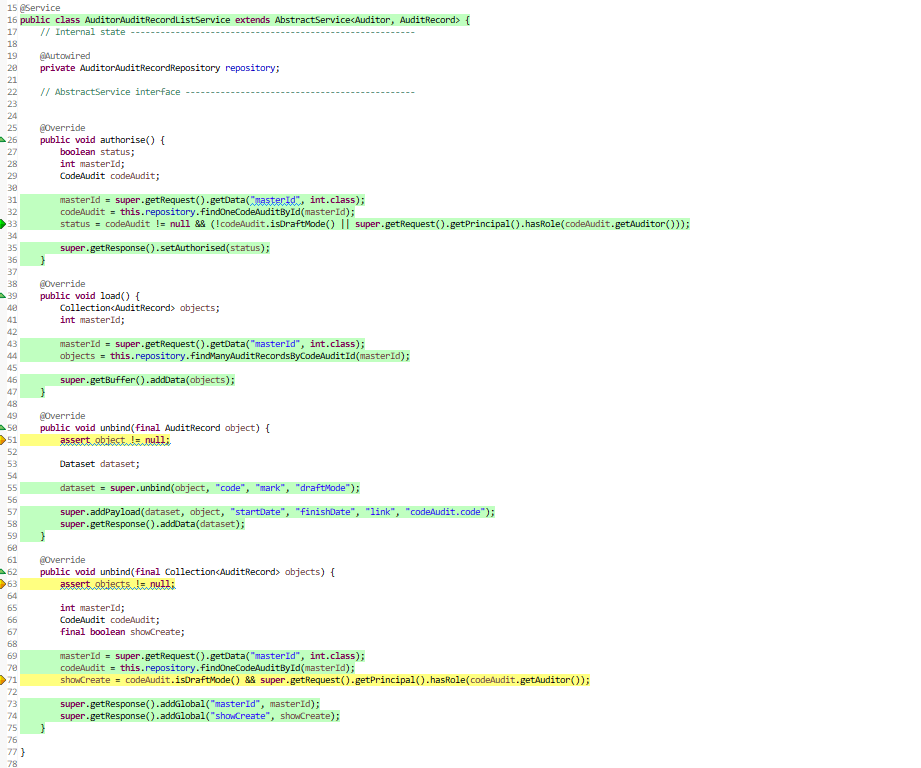
En esta clase podemos apreciar nuevamente líneas subrayadas de amarillo necesarias para la autorización. De nuevo, no se comprueban todos los posibles resultados de la estructura condicional.

## AuditorCodeAuditController.java



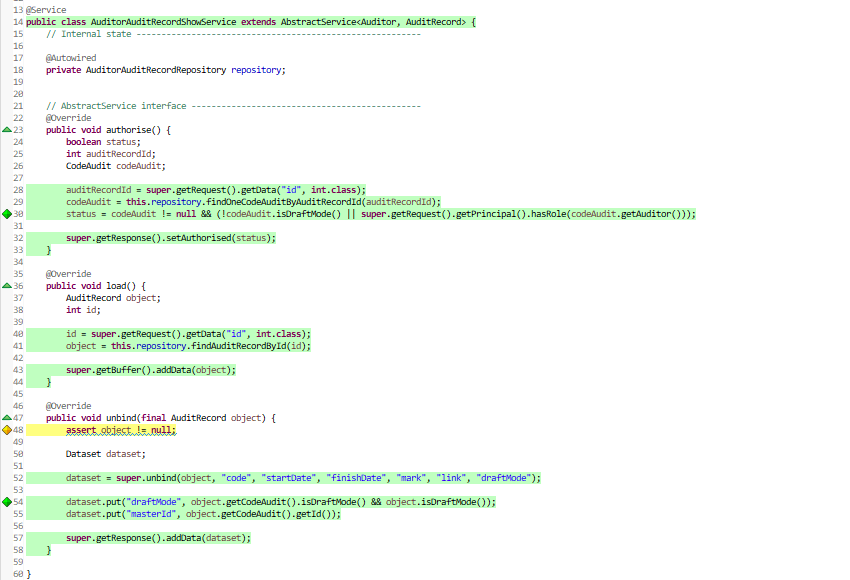
El controlador muestra una cobertura del 100%, ya que se usan todos los servicios de este.

## AuditorAuditRecordList.java



En esta clase de nuevo destaca es la línea de la instrucción “assert object !=null”. Además, hay una línea en el unbind también subrayada en amarillo.

## AuditorAuditRecordShowService.java



Al igual que en los casos anteriores, solo destaca la línea de la instrucción “assert object !=null”.

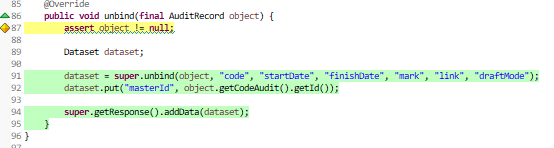
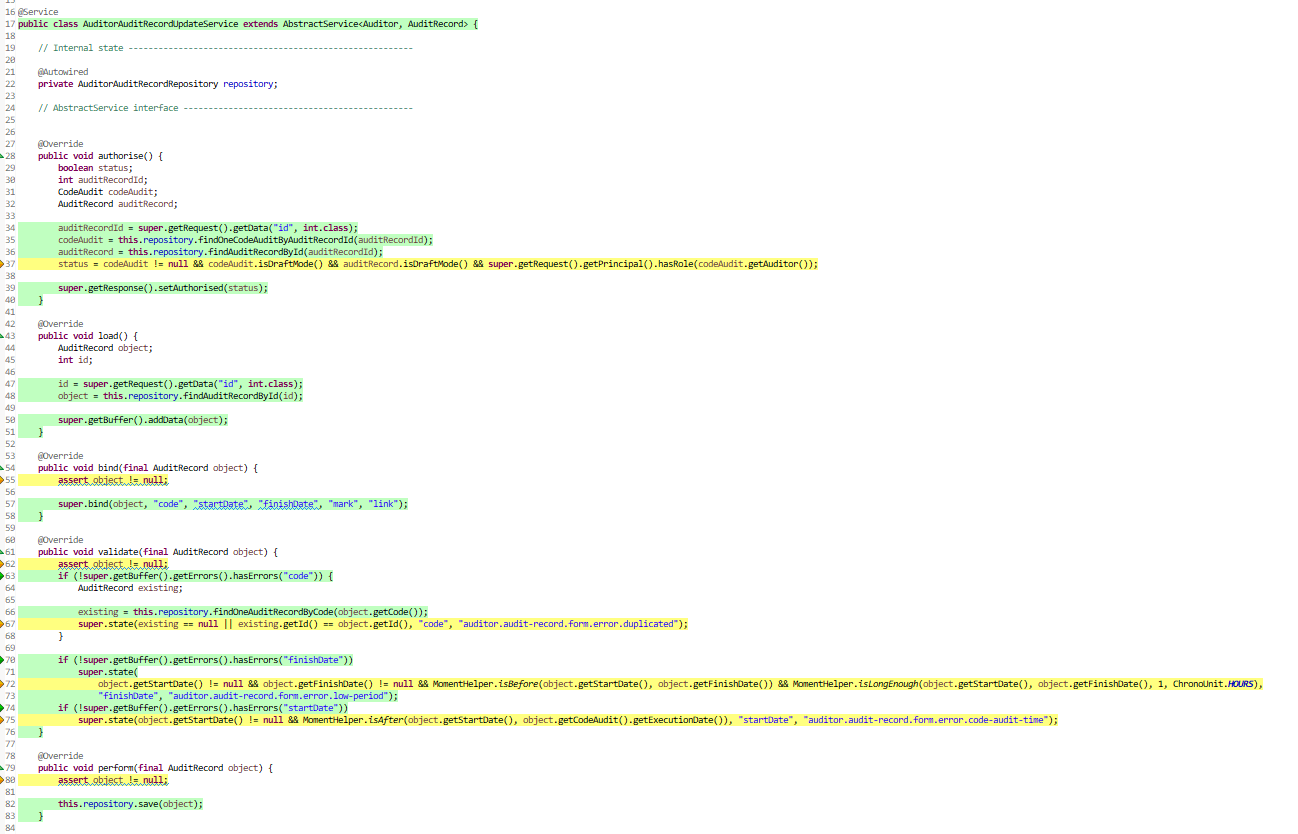
## AuditorAuditRecordCreateService.java

### 

En esta clase hay una línea subrayada en amarillo que corresponde a una línea que comprueba si la fecha de inicio es previa a la final. Puede que esté en amarillo debido a que no se realizó esta comprobación.

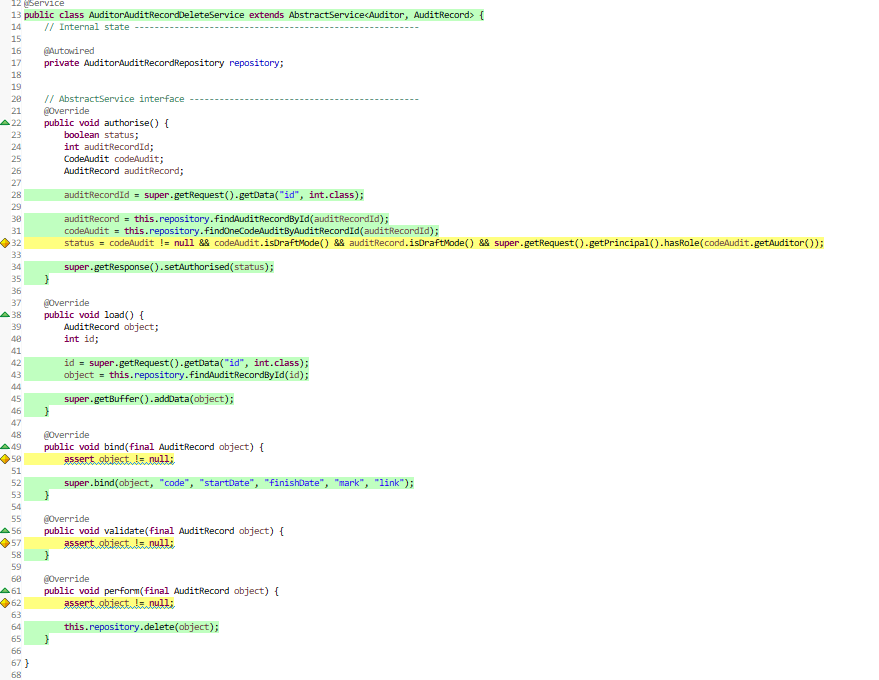
### 

## AuditorAuditRecordUpdateService.java



Hay una línea en el authorise() de que de nuevo está subrayada en amarillo. Además, en el bind vemos varias líneas subrayadas, correspondientes a comprobaciones. Como mencioné previamente, es posible que no se comprobaran todos los casos al hacer los tests.

## AuditorAuditRecordDeleteService.java



La cobertura de esta clase no llega al 90%. Esto es debido a que hay pocas líneas de código y varios “assert object!=null;” lo que hace que el porcentaje de cobertura baje. Además, volvemos a ver otra instrucción relacionada con la autorización en amarillo. Los tipos test que cubrirían completamente esta línea serían: “delete-right-role-wrong-user”, “delete-right-role-right-user-wrong-action” y “delete-wrong-role".

## AuditorAuditRecordController.java



El controlador muestra una cobertura del 100%, ya que se usan todos los servicios de este.

# Rendimiento del testing

Para evaluar el rendimeinto del testing solo se utilizó un ordenador, pero primero se realizaron las pruebas sin utilizar índices en las tablas de las entidades y después si se usó esta funcionalidad.

A continuación se muestra una gráfica de los datos antes de poner los íncides.

## Performance-testing sin índices

Como podemos observar, las peticiones más ineficientes son las de “delete”, “update”, “publish” y “list”. En el caso de los listados, se puede deber a que para probas todas las demás características de la aplicación se debe acceder primero a las listas de los objetos, lo cual va acumulando en ella una gran cantidad de peticiones. En el caso de “update” y “publish” se puede deber a la gran cantidad de validaciones que hay que realizar a la hora de invocar estos servicios.

## Performance-testing con índices

Aunque han variado los resultados con respecto a las pruebas sin los índices, ambas gráficas son bastante parecidas, por lo que no se puede destacar nada en particular.

## Comparación sin índices - con índices

A continuación, se muestra una comparación de los tests antes y después de utilizar los índices.

|  |  |  |
| --- | --- | --- |
| Prueba z para medias de dos muestras |  |  |
|  |  |  |
|  | *Before* | *After* |
| Media | 10,97332423 | 11,08003145 |
| Varianza (conocida) | 156,0384946 | 207,756568 |
| Observaciones | 1329 | 1307 |
| Diferencia hipotética de las medias | 0 |  |
| z | -0,202978704 |  |
| P(Z<=z) una cola | 0,419575839 |  |
| Valor crítico de z (una cola) | 1,644853627 |  |
| Valor crítico de z (dos colas) | 0,839151678 |  |
| Valor crítico de z (dos colas) | 1,959963985 |  |

Para analizar esta prueba es necesario fijarse en el valor de la celda “Valor crítico de z(dos colas)”. Como podemos comprobar, este valor (0,839151678) está entre [0.00, Alpha], lo que indica que se puede hacer una comparación de los tiempos medios. Sin embargo, el tiempo medio ha crecido levemente, por lo que, aunque no es nada grave, la introducción de los índices ha empeorado un poco el rendimiento.

# Conclusiones

Después de analizar los tests realizados, se destaca que la cobertura de las entidades implementadas por el estudiante 5 (Joaquín González Ganfornina) ha sido bastante alta. Además, se ha establecido un índice del 95% para el conjunto de casos de prueba sin los índices recomendados, así como para el conjunto que se han utilizado los índices recomendados.

# Bibliografía

En blanco a propóstio.