Universidad de Sevilla  
Escuela Técnica Superior de Ingeniería Informática

TESTING REPORT INDIVIDUAL - D04



**C1.027**

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

**Autor:**

Maureira Flores, Benjamín Ignacio

benmauflo@alum.us.es

**Fecha:** 27/05/2024

**Tabla de contenidos**

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

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

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

[**Capítulo I: Testing funcional 3**](#_heading=h.3k3mzb7frzkc)

[**Capítulo II: Testing de rendimiento 9**](#_heading=h.vr2c6jw2oh94)

[**Bibliografía 11**](#_heading=h.3dy6vkm)

# 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 | 27/05/2024 | 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.

# 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 Client sobre entidad Contract**

**client/contract/create:** Para realizar las pruebas de esta *feature* se ha comenzado haciendo pruebas negativas en el formulario de creación de un contrato 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 95.0% 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 ya que se nos demanda que sigamos esa práctica y por tanto, se ignoran. Para la prueba de hacking se ha intentado mostrar el formulario de creación con un rol incorrecto, lo que deriva en un error 500.

**client/contract/delete:** Para realizar esta prueba, se ha ingresado a un contrato no publicado y se ha eliminado, completando la prueba positiva de dicha *feature*.

Se han realizado 5 pruebas de hacking:

* Delete de un contrato que no me pertenece
* Delete de un contrato con id = null
* Delete de un contrato publicado
* Delete de un contrato con rol incorrecto
* Delete de un contrato con usuario incorrecto

Todos derivando en sus correspondientes errores 500.

En cuanto al porcentaje de cobertura, se ha obtenido un valor de 65.1%, quedando líneas marcadas en rojo pertenecientes al método unbind, pues estas en ningún caso se ejecutan, ya que no hay ninguna validación implementada que haga que el formulario se tenga que volver a mostrar, por lo que se ha mantenido el método para cumplir con las características del framework. Hay dos warnings a destacar:

* Línea 32: si el id = null, como en la segunda prueba de hacking, el método del repositorio devolverá null y en esta línea se devolverá null inmediatamente, sin llegar a evaluar contract.getClient().
* Línea 33: en esta línea se evalúan todas las condiciones posibles de autorización para el POST, por tanto al hacer las pruebas de hacking, en varias ocasiones hay código que no llega a evaluarse ya que habrá algún boolean a false.

**client/contract/list:** Para realizar esta prueba, se ha accedido al listado de contratos de un cliente comprobando que se lista adecuadamente, completando la prueba positiva de dicha *feature*. Se ha logrado un porcentaje de cobertura de 94.7% del servicio correspondiente, quedando warnings relacionados con la comprobación de si un objeto es nulo a través de *asserts*, los cuales se ignoran.

**client/contract/publish:** Para realizar esta prueba, se ha accedido a un contrato no publicado de un cliente, y se ha publicado vacío para comprobar todos los campos, tras hacer todos los casos de prueba negativos y comprobar las validaciones correspondientes, se ha completado el test positivo. Se han realizado las mismas pruebas de hacking que en el create, por lo que las conclusiones para los warnings en el authorise son las mismas. Hay dos warnings a destacar, ignorando los assert:

* Línea 70: no llego a ninguna conclusión lógica de por qué se trata de un warning, ya que es una operación OR, por lo que debería comprobar los estados a los dos lados de la operación, en ese caso se ejecutaría el código completo.
* Línea 83: este condicional ha sido traído directamente desde el create, de manera que se comprobara de que no daba errores 500 al enviar el formulario sin un proyecto. Entonces, al publicar siempre envía el proyecto y éste no se puede modificar, por lo que object.getProject() nunca será null.

Se ha obtenido un porcentaje de cobertura de 95.3% del código relacionado con el servicio.

**client/contract/show:** Para realizar esta prueba, se ha accedido al listado de contratos de un cliente y se ha ingresado a uno de ellos, comprobando que se muestra adecuadamente, completando la prueba positiva de dicha *feature*. Para la prueba de hacking, se ha intentado mostrar un contrato de otro cliente, obteniendo el correspondiente error 500.

Se ha conseguido un porcentaje de cobertura de 94.9% del código relacionado con el servicio. Solo han quedado algunos warnings marcando comprobaciones de que un objeto no es nulo a través de *asserts*, los cuales se ignoran.

**client/contract/update:** Para realizar las pruebas de esta *feature* se ha comenzado haciendo pruebas negativas en el formulario de de un contrato previamente creado y no publicado. 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 han realizado las mismas que en publish y create, llegando a las mismas conclusiones sobre los warnings que ya se han comentado, sin añadir nuevos e ignorando los assert.

Se ha conseguido un 94.5% de cobertura del código relacionado con el servicio correspondiente.

**Rol Client sobre entidad Progress Log**

**client/progress-log/create:** Para realizar las pruebas de esta *feature* se ha comenzado haciendo pruebas negativas en el formulario de creación de un informe de progreso 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. Para la prueba de hacking se ha intentado mostrar el formulario de creación con un rol incorrecto, lo que deriva en un error 500.

Se ha logrado una cobertura de un 93.1% de código, quedando algunos warnings a destacar, ignorando los assert:

* Línea 32: debería ejecutarse completamente ya que no se puede mostrar el formulario de creación sin el masterId, por lo que siempre encontrará el proyecto asociado en la base de datos.
* Línea 33: en el caso de hacking, se intenta mostrar el formulario sin ser cliente, por lo que el boolean que evalúa el rol del usuario principal es false y no evalúa el resto de la instrucción al ser una operación AND.
* Línea 67: debería ejecutarse siempre, ya que se debe comprobar que el código no se duplica.
* Línea 73: no llego a ninguna conclusión lógica de por qué se trata de un warning, ya que es una operación OR, por lo que debería comprobar los estados a los dos lados de la operación, en ese caso se ejecutaría el código completo.
* Línea 80: este warning se debe a que el informe de progreso que se ha creado no es el primero de un contrato, por lo que siempre habrá un informe de progreso *maxCompleteness*, y nunca será null. En el caso de que sí fuera el primer informe de progreso de un contract, *maxCompleteness* sería null y no se entraría en el bloque condicional.
* Línea 90: este warning se ha traído directamente del servicio publish, se explicará ahí más detalladamente.

**client/progress-log/delete:** Para realizar esta prueba, se ha ingresado a un informe de progreso no publicado y se ha eliminado, completando la prueba positiva de dicha *feature*. Se han realizado 5 pruebas de hacking:

* Delete de un informe de progreso que no me pertenece
* Delete de un informe de progreso con id = null
* Delete de un informe de progreso publicado
* Delete de un informe de progreso con rol incorrecto
* Delete de un informe de progreso con usuario incorrecto

Todos derivando en sus correspondientes errores 500.

En cuanto al porcentaje de cobertura, se ha obtenido un valor de 71.0%, quedando líneas marcadas en rojo pertenecientes al método unbind, pues estas en ningún caso se ejecutan, ya que no hay ninguna validación implementada que haga que el formulario se tenga que volver a mostrar, por lo que se ha mantenido el método para cumplir con las características del framework. Hay tres warnings a destacar:

* Línea 32: si el id = null, como en la segunda prueba de hacking, el método del repositorio devolverá null y en esta línea se devolverá null inmediatamente, sin llegar a evaluar contract.getClient().
* Línea 34: en esta línea se evalúan todas las condiciones posibles de autorización para el POST, por tanto al hacer las pruebas de hacking, en varias ocasiones hay código que no llega a evaluarse ya que habrá algún boolean a false.
* Línea 66: no tiene sentido esta validación, ya que se comprueba en el authorise que el informe de progreso no está publicado, por lo que la validación se puede eliminar sin problemas.

**client/progress-log/list:** Para realizar esta prueba, se ha accedido al listado de informes de progreso asociados a un contrato publicado de un cliente, comprobando que se lista adecuadamente, completando la prueba positiva de dicha *feature*. Se ha logrado un porcentaje de cobertura de 95.5% del servicio correspondiente, quedando warnings relacionados con la comprobación de si un objeto es nulo a través de *asserts*, los cuales se ignoran.

**client/progress-log/publish:** Para realizar esta prueba, se ha accedido a un informe de progreso no publicado asociado a un contrato publicado de un cliente, y se ha publicado vacío para comprobar todos los campos, tras hacer todos los casos de prueba negativos y comprobar las validaciones correspondientes, se ha completado el test positivo. Se han realizado las mismas pruebas de hacking que en el create, por lo que las conclusiones para los warnings en el authorise son las mismas. Hay varios warnings a destacar, ignorando los assert:

* Línea 61: debería ejecutarse siempre, ya que se debe comprobar que el código no se duplica.
* Línea 67: no llego a ninguna conclusión lógica de por qué se trata de un warning, ya que es una operación OR, por lo que debería comprobar los estados a los dos lados de la operación, en ese caso se ejecutaría el código completo.
* Línea 76: este warning se debe a que el informe de progreso que se va a publicar no es el primero de un contrato, por lo que siempre habrá un informe de progreso *maxCompleteness*, y nunca será null. En el caso de que sí fuera el primer informe de progreso de un contract, *maxCompleteness* sería null y no se entraría en el bloque condicional.
* Línea 88: este warning se implementa con el objetivo de impedir que se publique un informe de progreso con una fecha anterior al último publicado, pero sucede igualmente que en el anterior, si el informe de progreso no es el primero de un contrato, *maxCompleteness* nunca será null.
* Líneas 92 y 96: esta validación se podría eliminar, ya que se comprueba en el authorise que el informe de progreso no está publicado.

Se ha obtenido un porcentaje de cobertura de 93.5% del código relacionado con el servicio.

**client/progress-log/show:** Para realizar esta prueba, se ha accedido al listado de informes de progreso asociados a un contrato publicado de un cliente y se ha ingresado a uno de ellos, comprobando que se muestra adecuadamente, completando la prueba positiva de dicha *feature*. Para la prueba de hacking, se ha intentado mostrar un informe de progreso de otro cliente, obteniendo el correspondiente error 500.

Se ha conseguido un porcentaje de cobertura de 95.5% del código relacionado con el servicio. Solo han quedado algunos warnings marcando comprobaciones de que un objeto no es nulo a través de *asserts*, los cuales se ignoran. Sólo se destaca el warning de la línea 32, que no se ejecuta completamente debido a que en el hacking se habrá usado un id de un contrato no publicado, por lo que no evalúa el resto de la línea.

**client/progress-log/update:** Para realizar las pruebas de esta *feature* se ha comenzado haciendo pruebas negativas en el formulario de de un informe de progreso asociado a un contrato publicado previamente creado y no publicado. 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 han realizado las mismas que en publish y create, llegando a las mismas conclusiones sobre los warnings que ya se han comentado, sin añadir nuevos e ignorando los assert.

Se ha conseguido un 94.0% de cobertura del código relacionado con el servicio correspondiente.

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 siguiente captura.



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 contrato y borrar informe de progreso).

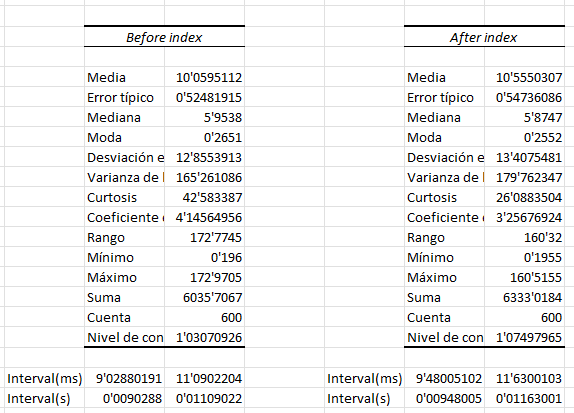
# 

# 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 los tests antes y después de añadir índices.

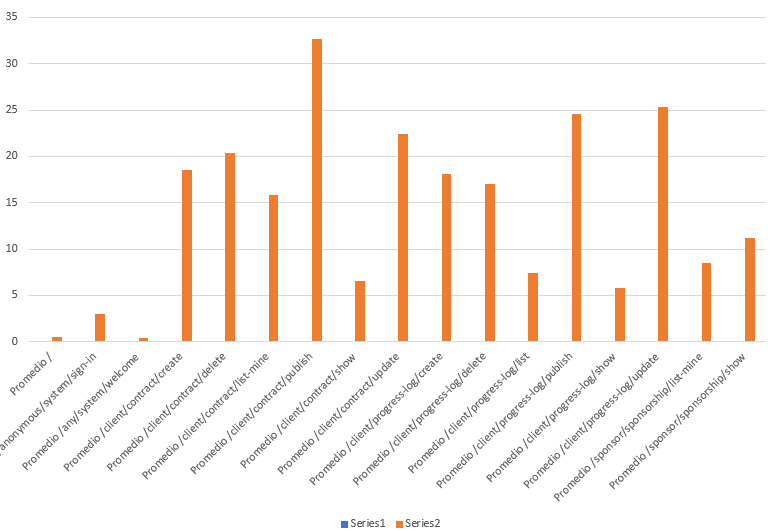
**Intervalo de confianza**

Para este apartado se han recolectado 601 datos de los distintos tests y se ha calculado el **intervalo de confianza con un nivel del 95%** en la misma máquina, primero sin índices en las tablas de la base de datos y luego con ellos.



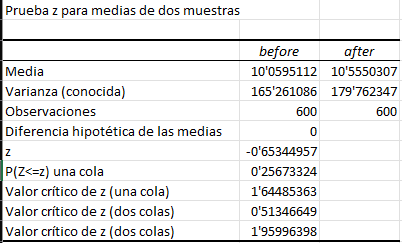
Como muestra el intervalo en segundos al final de cada tabla, se ha producido un **aumento en los tiempos**, pese a que en la segunda tabla se muestran los resultados de ejecutar los tests usando índices, que en teoría debería disminuir los tiempos.

Los tests ejecutados antes de añadir los índices mostraron que la función: **publicar un contrato era la que más tarda 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 pruebas tras ejecutar los tests, se ha considerado realizar la prueba Z-Test sobre las columnas de tiempos generados por ambas pruebas (antes y después de índices), quedándonos con el valor del campo de la tabla generada llamado ***Valor crítico de z (dos colas)*** ***= 0'513466488525277.***



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 = 0'513466488525277 tenemos que: *a < z <= 1.00*

Esta información nos está revelando que las segundas pruebas ejecutadas tras añadir índices **no resultaron en ninguna mejora significativa** con respecto a las pruebas ejecutadas anteriormente. 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, 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 la segunda prueba con índices confirma los mismos resultados, aunque con unos tiempos diferentes.

Los cálculos detallados del análisis estadístico de este capítulo puede comprobarse en los documentos adjuntos a esta entrega:

* ***tester-performance-clean.xlsx***
* ***tester-index-performance-clean.xlsx***
* ***z-test.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