

Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

## D04 – Testing Report



Grado en Ingeniería Informática – Ingeniería del Software

Diseño y Pruebas II

Curso 2023 – 2024

Fecha	Versión
27/05/2024	v1r1


Grupo de prácticas: C1.002	
Autores	Correo Corporativo
Rubén Pérez Garrido	rubpergar@alum.us.es

Link repositorio: <https://github.com/Joserra24/Acme-SF-D04-24.4.0>



## ***Índice***

<b>1. Tabla de versiones .....</b>	<b>3</b>
<b>2. Resumen ejecutivo .....</b>	<b>4</b>
<b>3. Introducción .....</b>	<b>5</b>
<b>4. Functional testing.....</b>	<b>6</b>
<b>5. Performance testing.....</b>	<b>10</b>
<b>6. Conclusiones .....</b>	<b>15</b>
<b>7. Bibliografía .....</b>	<b>16</b>

	<div>Diseño y Pruebas II</div> <div>C1.002</div> <div>Testing report - Rubén Pérez Garrido</div>
---	--

## 1. *Tabla de versiones*

Fecha	Versión	Descripción
26/05/2024	v1r0	Creación del documento
27/05/2024	v1r1	Entrega 1



## **2. Resumen ejecutivo**

Para este documento de testing se han desarrollado y explicado cada una de las implementaciones presentes en los requisitos obligatorios de la entrega D04.

Se han evaluado tanto el rendimiento del sistema como el desempeño funcional de todas y cada una de las funciones solicitadas, en este caso, las relativas a “contract” y a “progress log”. Para ello se siguió la metodología proporcionada en “S01 - Formal testing” y “S02 - Performance testing”.

El sistema muestra un comportamiento robusto en términos de funcionalidad pese a haber áreas que podrían precisar de una ligera toma de atención. En términos de rendimiento, la computadora 2 es más adecuada para manejar las cargas de trabajo del sistema.



### **3. Introducción**

Este documento se divide en dos secciones distintas:

1. Functional testing: se presentará un listado con los casos de prueba implementados, agrupados por funcionalidad. Por cada uno se dará una descripción y una indicación de cuan efectivo es detectando errores. Para la efectividad, se usará el coverage del código para comprobar que se han probado todas las decisiones posibles durante la ejecución del programa y así evitar la existencia de errores.
2. Performance testing: se proporcionarán los gráficos adecuados y un intervalo de confianza del 95% para el tiempo tomado para las solicitudes en las pruebas en dos ordenadores distintos. Además, tras las pruebas en los diferentes ordenadores, se indicará cual de estos es el más potente y ofrece mejor rendimiento.



#### 4. Functional testing

Casos de pruebas relativos a contratos:

- Create contract
  - Descripción: Se prueban las restricciones de todos los campos del formulario de creación de un contrato con valores relativos a casos positivos, negativos y de hacking.
  - Coverage: 93,4%
  - Efectividad: Entendemos que un coverage alrededor del 95% es un valor alto de efectividad. Vemos que lo que baja el coverage son las líneas de “assert object != null” que están en amarillo. Esto nos indica que, en ningún caso al realizar la función en la que se encuentra esta línea de código, se ha obtenido un objeto nulo.
- Delete contract
  - Descripción: Se prueba la eliminación de un contrato teniendo en cuenta restricciones como que no se podrá eliminar un contrato si tiene asociado algún registro de progreso que esté publicado. Se valida que un cliente que no es propietario de un contrato ajeno no tenga acceso a esta funcionalidad.
  - Coverage: 92%
  - Efectividad: Alta. Misma situación con “assert object != null”. Junto a esto, en la autorización, observamos en amarillo la línea “client = object == null ? null : object.getClient()”, la cual se encuentra en este estado porque en las pruebas siempre se obtiene que el objeto cliente es distinto de nulo. También encontramos que la línea “status = contract != null && ...” está en amarillo. Esto se debe a que en las pruebas siempre se obtiene un contrato, por lo que nunca es nula esta variable y por ende esta primera condición siempre es verdadera.
- List my contracts
  - Descripción: Se prueba el listado de contratos pertenecientes al cliente que ha iniciado sesión en el sistema. Se valida que un cliente que no es propietario de un contrato ajeno no tenga acceso a esta funcionalidad.
  - Coverage: 94,6%
  - Efectividad: Alta. Misma situación con “assert object != null”.



- Publish contract
  - Descripción: Se prueba la publicación de un contrato teniendo en cuenta restricciones como que no se podrá publicar un contrato si tiene al menos un registro de progreso sin publicar o si no tiene registros de progreso asociados. Se valida que un cliente que no es propietario de un contrato ajeno no tenga acceso a esta funcionalidad.
  - Coverage: 94%
  - Efectividad: Alta. Misma situación con “assert object != null”, “client = object == null ? null : object.getClient()” y “status = contract != null && ...”. Además de esto, el if de la línea de la validación que comprueba que no existan registros de progreso sin publicar está en amarillo. Esto puede ser debido a un problema de Eclipse, ya que están probados tanto los casos en los que se quiere publicar un contrato, tanto con registros de progreso sin publicar, como con y sin ellos.
- Show contract details
  - Descripción: Se prueba la muestra de detalles de un contrato del que el cliente es propietario. Se valida que un cliente que no es propietario de un contrato ajeno no tenga acceso a esta funcionalidad.
  - Coverage: 95,5%
  - Efectividad: Alta. Misma situación con “assert object != null”, “client = object == null ? null : object.getClient()” y “status = contract != null && ...”.
- Update contract
  - Descripción: Se prueban las restricciones de todos los campos del formulario de actualización de un contrato con valores relativos a casos positivos, negativos y de hacking. Se valida que un cliente que no es propietario de un contrato ajeno no tenga acceso a esta funcionalidad.
  - Coverage: 92%
  - Efectividad: Alta. Misma situación con “assert object != null”. y alguna condición del status y client que siempre será igual y no termina de ser verde y el error del código duplicado me pone en amarillo el super.state

Acme-SF-D04 (tester#replayer) (27 may 2024 0:34:12)				
Element	Coverage	Verified Instructions	Unverified Instructions	Total Instructions
> acme.features.authenticated.risk	15,1 %	22	124	146
> acme.features.developer.dashboard	10,0 %	12	108	120
▼ acme.features.client.contract	93,4 %	1.169	82	1.251
> ClientContractUpdateService.java	92,0 %	243	21	264
> ClientContractDeleteService.java	92,0 %	229	20	249
> ClientContractCreateService.java	93,4 %	228	16	244
> ClientContractPublishService.java	94,0 %	236	15	251
> ClientContractShowService.java	95,5 %	128	6	134
> ClientContractListMineService.java	94,6 %	70	4	74
> ClientContractController.java	100,0 %	35	0	35



Casos de pruebas relativos a registros de progreso:

- Create progress log
  - Descripción: Se prueban las restricciones todos los campos del formulario de creación de un registro de progreso con valores relativos a casos positivos, negativos y de hacking.
  - Coverage: 90,3%
  - Efectividad: Alta. Misma situación con “assert object != null”. Además de esto, la línea del código de error en la validación de la duplicación del recordId está en amarillo. Esto puede ser debido a un problema de Eclipse, ya que están probados tanto los casos en los que se quiere crear un registro de progreso, tanto con un recordId válido, duplicado y erróneo.
- Delete progress log
  - Descripción: Se prueba la eliminación de un registro de progreso. Se valida que un cliente que no es propietario de un registro de progreso ajeno no tenga acceso a esta funcionalidad.
  - Coverage: 90,5%
  - Efectividad: Alta. Misma situación con “assert object != null” y “status = contract != null && ...”.
- List my progress logs
  - Descripción: Se prueba el listado de registros de progreso pertenecientes a un contrato que pertenece a su vez al cliente que ha iniciado sesión en el sistema. Se valida que un cliente que no es propietario de un registro de progreso ajeno no tenga acceso a esta funcionalidad.
  - Coverage: 96,3%
  - Efectividad: Alta. Misma situación con “assert object != null” y “status = contract != null && ...”.
- Publish progress log
  - Descripción: Se prueba la publicación de un registro de progreso. Se valida que un cliente que no es propietario de un registro de progreso ajeno no tenga acceso a esta funcionalidad.
  - Coverage: 90,8%
  - Efectividad: Alta. Misma situación con “assert object != null” y “status = contract != null && ...”.





- Show progress log details
  - Descripción: Se prueba la muestra de detalles de un registro de progreso del que el cliente es propietario. Se valida que un cliente que no es propietario de un registro de progreso ajeno no tenga acceso a esta funcionalidad.
  - Coverage: 95,3%
  - Efectividad: Alta. Misma situación con “assert object != null” y “status = contract != null && ...”.
- Update progress log
  - Descripción: Se prueban las restricciones de todos los campos del formulario de actualización de un registro de progreso con valores relativos a casos positivos, negativos y de hacking. Se valida que un cliente que no es propietario de un registro de progreso ajeno no tenga acceso a esta funcionalidad.
  - Coverage: 91,1%
  - Efectividad: Alta. Misma situación con “assert object != null” y “status = contract != null && ...”. Además de esto, la línea del código de error en la validación de la duplicación del recordId está en amarillo. Esto puede ser debido a un problema de Eclipse, ya que están probados tanto los casos en los que se quiere crear un registro de progreso, tanto con un recordId válido, duplicado y erróneo.

Element	Coverage	covered Instructions	missed Instructions	Total Instructions
> ClientContractController.java	100,0 %	35	0	35
> acme.roles	61,6 %	125	78	203
▼ acme.features.client.progressLog	92,2 %	913	77	990
> ClientProgressLogUpdateService.java	91,1 %	184	18	202
> ClientProgressLogCreateService.java	90,3 %	158	17	175
> ClientProgressLogDeleteService.java	90,5 %	153	16	169
> ClientProgressLogPublishService.java	90,8 %	157	16	173
> ClientProgressLogShowService.java	95,3 %	121	6	127
> ClientProgressLogListMineService.java	96,3 %	104	4	108
> ClientProgressLogController.java	100,0 %	36	0	36

En resumen, no hay nada en rojo en el coverage, es decir, que no hay nada que no haya sido probado íntegramente. Por otra parte, las líneas en amarillo ya se han explicado el motivo de su causa. Por todo esto y por tener un coverage del 93,4% de media en contract y de 92,2% de media en progress log, se considera que la existencia de potenciales fallos o bugs es ínfima.

## 5. Performance testing

Tras realizar el conjunto de tests para las funcionalidades oportunas, se han realizado todos los pasos que se muestran en “S02 - Performance testing”, obteniendo los siguientes resultados:

request-path	time
Promedio /	9.370808
Promedio /anonymous/system/sign-in	8.719893878
Promedio /any/system/welcome	4.847741509
Promedio /authenticated/system/sign-out	6.1436
Promedio /client/contract/create	103.9346241
Promedio /client/contract/delete	106.021875
Promedio /client/contract/list	23.278148
Promedio /client/contract/publish	114.312275
Promedio /client/contract/show	36.511
Promedio /client/contract/update	98.91798269
Promedio /client/progress-log/create	52.73856981
Promedio /client/progress-log/delete	49.3879
Promedio /client/progress-log/list-mine	20.03825294
Promedio /client/progress-log/publish	52.88755
Promedio /client/progress-log/show	37.36146
Promedio /client/progress-log/update	68.49123673
<b>Promedio general</b>	<b>47.57473991</b>

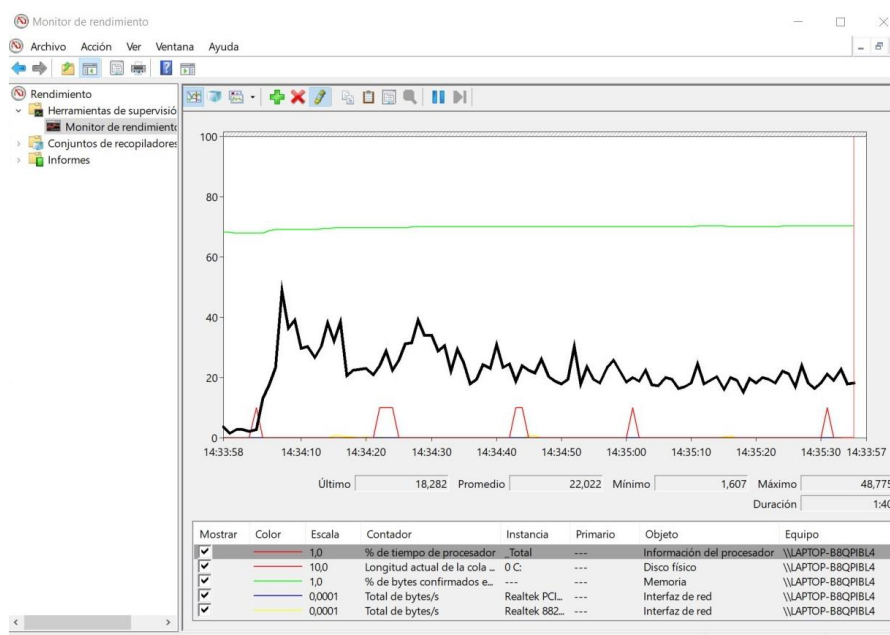
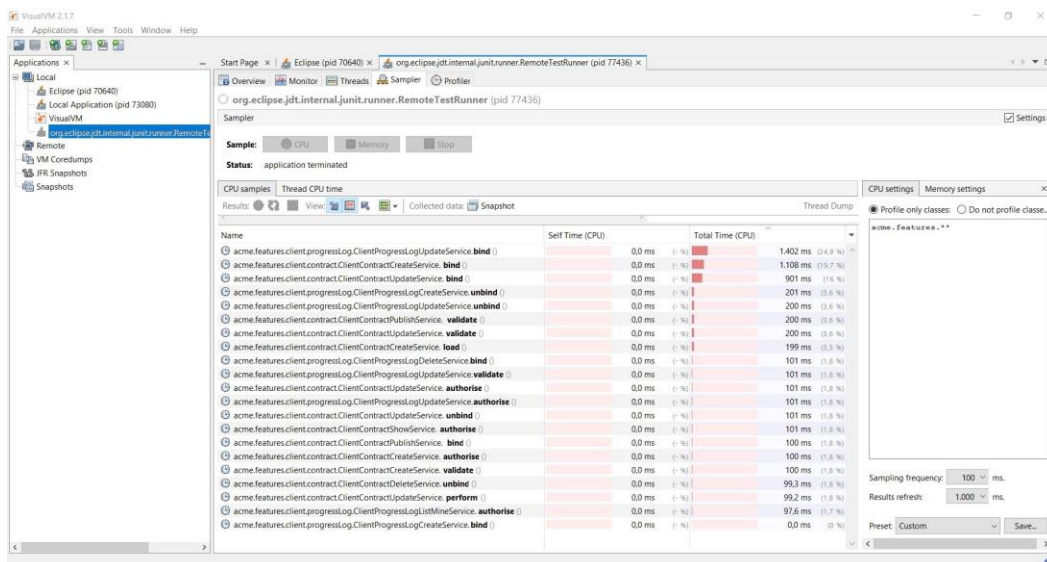




Como se puede observar en las imágenes superiores, el tiempo promedio que tarda el sistema en realizar una petición es de unos 47,6 ms, es decir, 0,047 segundos, bastante rápido.

También se puede comprobar en el gráfico de barras de forma más sencilla, que las peticiones que más tiempo tardan son aquellas que manejan mayor cantidad de datos y validaciones; las relativas a la creación, edición, eliminación y publicación de un contrato. Esto puede ser en parte debido a que se han de comprobar distintas validaciones tanto de la propia clase contract, como de la clase asociada progress log.

Posteriormente se realizó un examen del estado de la computadora donde se estaban realizando las pruebas y estos fueron los resultados:



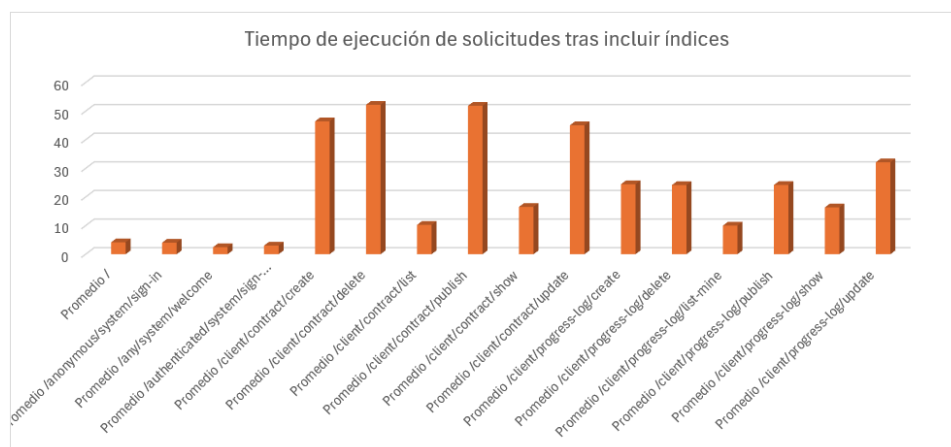


En la primera imagen se puede observar que las funciones que más tiempo están en CPU para poder completarse son aquellas relacionadas con el bind, concretamente las de actualizar el registro de progreso, seguido por la creación y actualización de un contrato. Hay que destacar que, gracias al Self Time que tiene un valor de 0,0 ms, podemos saber que lo que consume tiempo no es el método en sí, sino los métodos internos a los que hace referencia e invoca.

Por otro lado, en la segunda imagen se puede ver el rendimiento de la computadora durante el tiempo de ejecución de las pruebas. No se presenta cuello de botella, ya que ninguno de los componentes medidos alcanza el 100% de su capacidad.

Posteriormente, se añadieron los índices necesarios para mejorar el rendimiento de las consultas SQL y estos fueron los resultados obtenidos tras las nuevas pruebas:

request-path	time
<b>Promedio /</b>	<b>4.16927</b>
<b>Promedio /anonymous/system/sign-in</b>	<b>4.044753061</b>
<b>Promedio /any/system/welcome</b>	<b>2.462396226</b>
<b>Promedio /authenticated/system/sign-out</b>	<b>3.0255</b>
<b>Promedio /client/contract/create</b>	<b>46.4410537</b>
<b>Promedio /client/contract/delete</b>	<b>52.216</b>
<b>Promedio /client/contract/list</b>	<b>10.255588</b>
<b>Promedio /client/contract/publish</b>	<b>51.85715</b>
<b>Promedio /client/contract/show</b>	<b>16.53687727</b>
<b>Promedio /client/contract/update</b>	<b>45.06773462</b>
<b>Promedio /client/progress-log/create</b>	<b>24.46059057</b>
<b>Promedio /client/progress-log/delete</b>	<b>24.147475</b>
<b>Promedio /client/progress-log/list-mine</b>	<b>9.999970588</b>
<b>Promedio /client/progress-log/publish</b>	<b>24.193525</b>
<b>Promedio /client/progress-log/show</b>	<b>16.36315</b>
<b>Promedio /client/progress-log/update</b>	<b>32.13359592</b>
<b>Promedio general</b>	<b>21.78708226</b>





Se puede observar que ha habido una mejora sustancial de rendimiento, pues el tiempo medio de ejecución de solicitudes ha bajado un 50% tras la adición de los índices. Una vez sabido esto se realizan las comparativas oportunas de los distintos valores obtenidos en cada prueba y obtenemos:

<i>Before</i>			<i>After</i>		
Media	47.57473991		Media	21.78708226	
Error típico	2.073797405		Error típico	0.902146949	
Mediana	45.4349		Mediana	21.215	
Moda	#N/D		Moda	#N/D	
Desviación estándar	44.04073899		Desviación estándar	19.15867876	
Varianza de la muestra	1939.586691		Varianza de la muestra	367.054972	
Curtosis	12.70920906		Curtosis	3.876994454	
Coeficiente de asimetría	1.982464018		Coeficiente de asimetría	1.175314689	
Rango	436.1373		Rango	151.4443	
Mínimo	2.7245		Mínimo	1.5946	
Máximo	438.8618		Máximo	153.0389	
Suma	21456.2077		Suma	9825.9741	
Cuenta	451		Cuenta	451	
Nivel de confianza(95.0%)	4.075529656		Nivel de confianza(95.0%)	1.772943989	
Interval (ms)	43.49921026	51.6502696	Interval (ms)	20.01413827	23.5600263
Interval (s)	0.04349921	0.05165027	Interval (s)	0.020014138	0.02356003

Para determinar que los promedios de los tiempos antes y después de los cambios puedan ser considerados los mismos o no, se ha realizado un z-test con los siguientes resultados:

	<i>Before</i>	<i>After</i>
Media	47.57473991	21.78708226
Varianza (conocida)	1940	367
Observaciones	451	451
Diferencia hipotética de las medias	0	
z	11.40187747	
P(Z<=z) una cola	0	
Valor crítico de z (una cola)	1.644853627	
Valor crítico de z (dos colas)	0	
Valor crítico de z (dos colas)	1.959963985	

Ante este resultado del two-tail p-value (0), podemos concluir que, al estar en el intervalo que se encuentra entre [0, 0.05), los cambios realizados han sido fructíferos y han ayudado a mejorar el rendimiento.



Ahora se comparará el rendimiento del sistema en dos computadoras distintas. La primera computadora será en la que se han realizado todas las pruebas anteriores y la segunda será la computadora de otro miembro del equipo. Estos son los resultados:

<i>Computadora 1</i>			<i>Computadora 2</i>		
Media	21.78708226		Media	22.12824368	
Error típico	0.902146949		Error típico	0.981633762	
Mediana	21.215		Mediana	17.5146	
Moda	#N/D		Moda	2.7645	
Desviación estándar	19.15867876		Desviación estándar	20.84672118	
Varianza de la muestra	367.054972		Varianza de la muestra	434.5857839	
Curtosis	3.876994454		Curtosis	5.013711279	
Coefficiente de asimetría	1.175314689		Coefficiente de asimetría	1.329444602	
Rango	151.4443		Rango	172.3919	
Mínimo	1.5946		Mínimo	1.2826	
Máximo	153.0389		Máximo	173.6745	
Suma	9825.9741		Suma	9979.8379	
Cuenta	451		Cuenta	451	
Nivel de confianza(95.0%)	1.772943989		Nivel de confianza(95.0%)	1.929155421	
Interval (ms)	20.01413827	23.5600263	Interval (ms)	20.19908826	24.0573991
Interval (s)	0.020014138	0.02356003	Interval (s)	0.020199088	0.0240574

Como se puede comprobar, ambos resultados son muy parecidos, por lo que se puede deducir que las características y rendimiento de ambas computadoras son muy similares. Además de esto, con el z-test podemos comprobar que si el valor del two-tail p-value es mayor que alpha (0.05), los resultados se pueden considerar los mismos.

	<i>Computadora 1</i>	<i>Computadora 2</i>
Media	21.78708226	22.12824368
Varianza (conocida)	367	435
Observaciones	451	451
Diferencia hipotética de las medias	0	
z	-0.255835613	
P(Z<=z) una cola	0.399038887	
Valor crítico de z (una cola)	1.644853627	
Valor crítico de z (dos colas)	0.798077774	
Valor crítico de z (dos colas)	1.959963985	

Como se ve, el valor del two-tail p-value es 0.798, un valor muy alejado de alpha, por lo que se puede afirmar que ambos computadores son muy similares. Sin embargo, pese a la leve diferencia, se puede afirmar que, en función del promedio de tiempo de ejecución de solicitudes, el computador 1 es ligeramente superior al computador 2.



## **6. Conclusiones**

Después de elaborar este documento sobre testing, se ha concluido que esta fase del ciclo de vida de un proyecto es crucial. Validar que todas las funciones desarrolladas operen correctamente y sean revisadas minuciosamente para minimizar errores o bugs, además de asegurar que el rendimiento esté optimizado al máximo, son factores fundamentales de cara al cliente. Un sistema bien probado permite que el usuario final lo utilice de manera rápida e intuitiva, evitando problemas que puedan afectar negativamente su experiencia. Además, un proceso de testing riguroso contribuye a la satisfacción del cliente y a la reputación del producto, garantizando que las expectativas de calidad y eficiencia sean cumplidas de manera consistente.



## **7. Bibliografía**

- 08 Annexes – Material proporcionado en la asignatura *Diseño y Pruebas II* por la Universidad de Sevilla.
- L04 - S01 - Formal testing - Material proporcionado en la asignatura *Diseño y Pruebas II* por la Universidad de Sevilla.
- L04 - S02 - Performance testing - Material proporcionado en la asignatura *Diseño y Pruebas II* por la Universidad de Sevilla.