

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
08/07/2024	v1r2

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

Link repositorio: <https://github.com/SoniaRM/Acme-SF-D04-24.5.0>



Índice

1. Tabla de versiones.....	3
2. Resumen ejecutivo	4
3. Introducción.....	5
4. Functional testing	6
5. Performance testing previo a la revisión del cliente	11
6. Performance testing tras la revisión del cliente	16
7. Conclusiones.....	19
8. Bibliografía.....	20



1. Tabla de versiones

Fecha	Versión	Descripción
26/05/2024	v1r0	Creación del documento
27/05/2024	v1r1	Entrega 1
01/07/2024	v1r2	Modificación del contenido de acuerdo con los cambios realizados para subsanar los errores encontrados por el cliente



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”.

Además de esto, se presentan los resultados del sistema tras los cambios realizados en el mismo para subsanar los errores presentes encontrados por el cliente.

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. Esto se presentará para el sistema antes y después de su revisión por parte del propio cliente.



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. Se valida que un usuario no registrado como cliente no tenga acceso a esta funcionalidad.
 - Coverage: 93,8%
 - Efectividad: Entendemos que una cobertura alrededor del 95% es un valor alto de efectividad. Observamos que lo que reduce la cobertura son las líneas de “assert object != null” que están en amarillo. Esto nos indica que, al ejecutar la función en la que se encuentra esta línea de código, nunca se ha obtenido un objeto nulo, por lo que este caso no se ha podido verificar. Por otra parte, dentro del método de validación, vemos que la línea relacionada con el mensaje de error que aparece al ingresar una cantidad superior al límite establecido está en amarillo. Esto puede deberse a un fallo interno de Eclipse, ya que durante la grabación de las pruebas se utilizaron todos los valores que podrían comprometer este campo y, por tanto, se probó introducir un valor superior al límite.
- Delete contract
 - Descripción: Se prueba la eliminación de un contrato. Se valida que un cliente u otro usuario no registrado que no es propietario de un contrato ajeno no tenga acceso a esta funcionalidad.
 - Coverage: 78,7%
 - Efectividad: Media. Tenemos la misma situación con la línea “assert object != null”. Además, 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 no es nulo. También encontramos que la línea “status = object != null && object.isDraftMode() && super.getRequest().getPrincipal().hasRole(client)” 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. El bajo nivel de cobertura en esta funcionalidad se debe al método unbind, que no es llamado durante el proceso de ejecución. Como unbind se encarga de transformar la información cargada en tuplas para poder procesar las vistas y, en este caso, estamos eliminando un contrato, esto no será necesario. Además, unbind también ayuda al manejo de errores, pero en este caso, al no tener que guardar ningún valor antes de realizar la acción, tampoco será necesario llamar a este método.



- List my contracts
 - Descripción: Se prueba el listado de contratos pertenecientes al cliente que ha iniciado sesión en el sistema. Se valida que un usuario no registrado no tenga acceso a esta funcionalidad y que otro cliente solo pueda visionar sus contratos.
 - Coverage: 94,6%
 - Efectividad: Alta. Misma situación con “assert object != null”.
- Publish contract
 - Descripción: Se prueba la publicación de un contrato. Se valida que un cliente u otro usuario no registrado que no es propietario de un contrato ajeno no tenga acceso a esta funcionalidad.
 - Coverage: 77,5%
 - Efectividad: Media. Misma situación con “assert object != null”, “client = object == null ? null : object.getClient()”, “s status = object != null && object.isDraftMode() && super.getRequest().getPrincipal().hasRole(client)” y con el unbind.
- Show contract details
 - Descripción: Se prueba la muestra de detalles de un contrato del que el cliente es propietario. Se valida que un cliente u otro usuario no registrado que no es propietario de un contrato ajeno no tenga acceso a esta funcionalidad.
 - Coverage: 95,4%
 - Efectividad: Alta. Misma situación con “assert object != null”, “client = object == null ? null : object.getClient()” y “status = object != null && super.getRequest().getPrincipal().hasRole(client)”.
- 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 u otro usuario no registrado que no es propietario de un contrato ajeno no tenga acceso a esta funcionalidad.
 - Coverage: 92,4%
 - Efectividad: Alta. Tenemos la misma situación con las líneas “assert object != null”, “client = object == null ? null : object.getClient()” y “status = object != null && object.isDraftMode() && super.getRequest().getPrincipal().hasRole(client)”. Además, ocurre una incidencia similar a la vista en la creación del contrato: obtenemos elementos en amarillo para los mensajes de error relativos a la introducción de un código duplicado y un valor del presupuesto por encima del límite. Sin embargo, ambos casos fueron contemplados durante la grabación de las pruebas.



Acme-SF-D04 (tester#replayer) (2 jul 2024 13:36:26)

Element	Coverage	Verified Instructions	Missed Instructions	Total Instructions
acme.features.client.contract	90,6 %	918	95	1.013
> ClientContractDeleteService.java	78,7 %	85	23	108
> ClientContractPublishService.java	77,5 %	79	23	102
> ClientContractUpdateService.java	92,4 %	268	22	290
> ClientContractCreateService.java	93,8 %	256	17	273
> ClientContractShowService.java	95,4 %	125	6	131
> 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. Se valida que un cliente u otro usuario no registrado que no es propietario de un contrato ajeno no tenga acceso a esta funcionalidad.
 - Coverage: 92,0%
 - Efectividad: Alta. Misma situación con “assert object != null” y “status = object != null && object.isDraftMode() && super.getRequest().getPrincipal().hasRole(client)”. 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 u otro usuario no registrado que no es propietario de un contrato ajeno no tenga acceso a esta funcionalidad.
 - Coverage: 81,6%
 - Efectividad: Media. Misma situación con “assert object != null”, “status = object != null && object.isDraftMode() && super.getRequest().getPrincipal().hasRole(client)” y con el método unbind.
- List 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 u otro usuario no registrado que no es propietario de un contrato ajeno no tenga acceso a esta funcionalidad.
 - Coverage: 93,7%
 - Efectividad: Alta. Misma situación con “assert object != null”, “client = object == null ? null : object.getClient()” y “status = object != null && super.getRequest().getPrincipal().hasRole(client)”.



- Publish progress log
 - Descripción: Se prueba la publicación de un registro de progreso. Se valida que un cliente u otro usuario no registrado que no es propietario de un contrato ajeno no tenga acceso a esta funcionalidad.
 - Coverage: 77,4%
 - Efectividad: Media. Misma situación con “assert object != null”, “status = object != null && object.isDraftMode() && super.getRequest().getPrincipal().hasRole(client)” y con el método unbind.
- 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 u otro usuario no registrado que no es propietario de un contrato ajeno no tenga acceso a esta funcionalidad.
 - Coverage: 93,7%
 - Efectividad: Alta. Misma situación con “assert object != null”, contract= object == null ? null : object.getContract(), “client = object == null ? null : object.getClient()” y “status = object != null && super.getRequest().getPrincipal().hasRole(client)”
- 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 u otro usuario no registrado que no es propietario de un contrato ajeno no tenga acceso a esta funcionalidad.
 - Coverage: 90,5%
 - Efectividad: Alta. Misma situación con “assert object != null” y “status = contract != null && object.isDraftMode() && super.getRequest().getPrincipal().hasRole(contract.getClient())”. 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.

Acme-SF-D04 (tester#replayer) (2 jul 2024 13:36:26)				
Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
acme.features.client.progressLog	89,7 %	830	95	925
> ClientProgressLogDeleteService.java	81,6 %	93	21	114
> ClientProgressLogPublishService.java	77,4 %	72	21	93
> ClientProgressLogUpdateService.java	90,5 %	180	19	199
> ClientProgressLogCreateService.java	92,0 %	184	16	200
> ClientProgressLogListService.java	93,7 %	148	10	158
> ClientProgressLogShowService.java	93,7 %	118	8	126
> ClientProgressLogController.java	100,0 %	35	0	35

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

En resumen, las únicas líneas en rojo del código son las relativas al método unbind en las funcionalidades de borrado y publicación tanto de contratos como de registros de progreso. La causa de esto ya ha sido justificada anteriormente, al igual que las líneas que están en amarillo. Por todo esto, y debido a que tenemos una cobertura del 90.6% en contratos y del 89.7% en registros de progreso, se considera que la existencia de potenciales fallos o bugs es muy baja.

5. Performance testing previo a la revisión del cliente

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
Promedio general	47.57473991



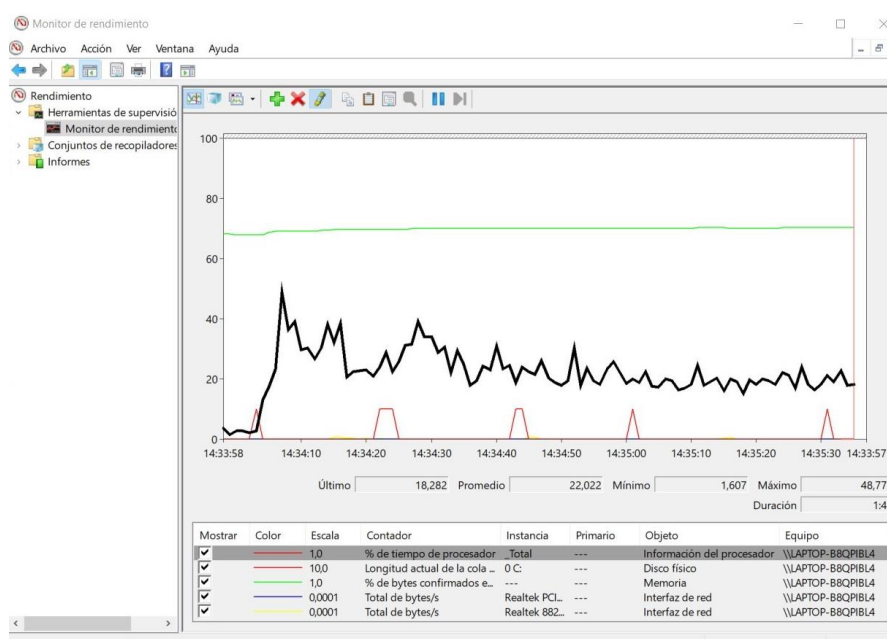
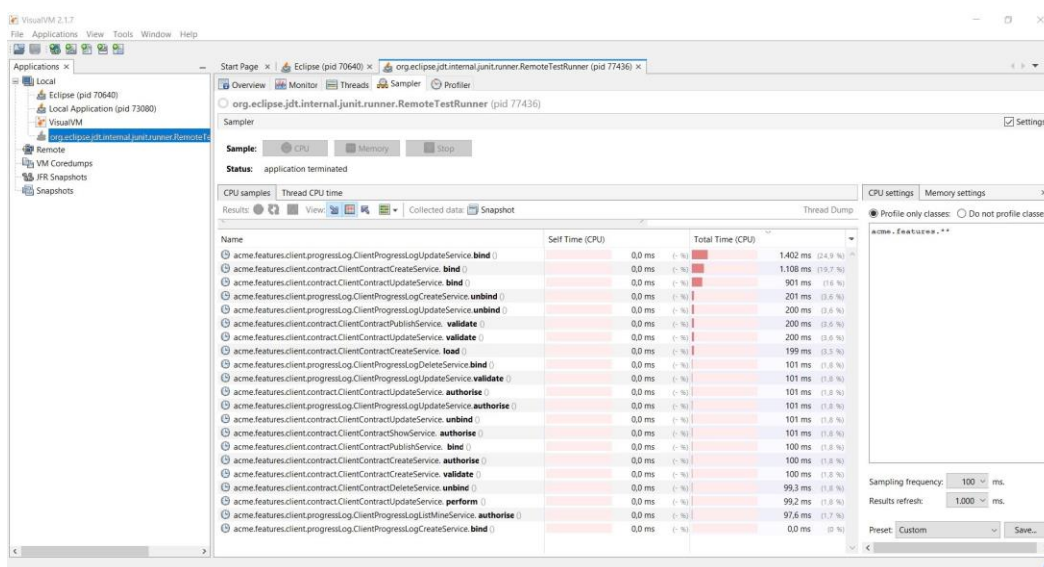
Como se puede observar en las imágenes superiores, el tiempo promedio que tarda el sistema en realizar una petición es de unos 47,6 ms, es decir, 0,047 segundos, bastante rápido.



Testing report - Rubén Pérez Garrido

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:



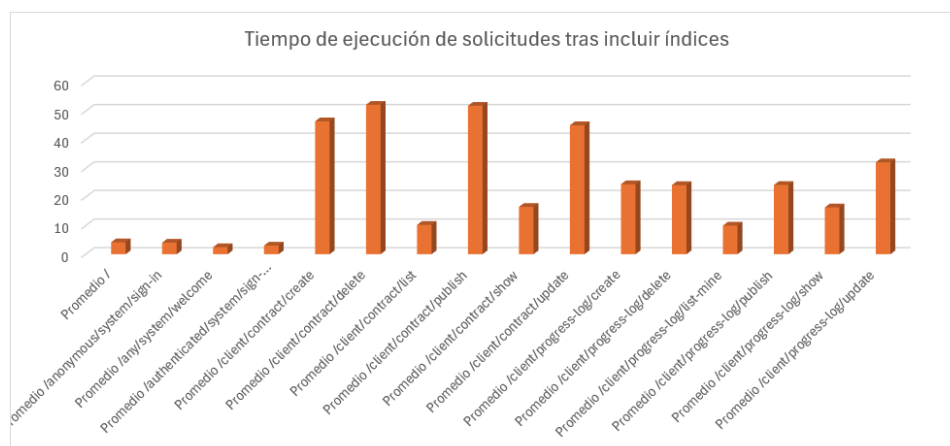
En la primera imagen se puede observar que las funciones que más tiempo están en CPU para poder completarse son aquellas relacionadas con el bind, concretamente las de actualizar el registro de progreso, seguido por la creación y actualización de un contrato. Hay que destacar que, gracias al Self Time 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.

Testing report - Rubén Pérez Garrido

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



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:



Testing report - Rubén Pérez Garrido

<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:



Testing report - Rubén Pérez Garrido

Computadora 1			Computadora 2		
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.

	Computadora 1	Computadora 2
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	

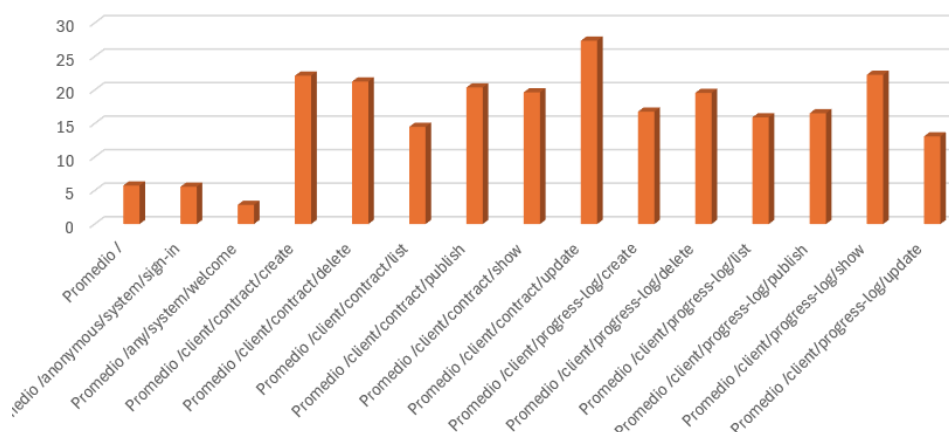
Como se ve, el valor del two-tail p-value es 0.798, un valor muy alejado de alpha, por lo que se puede afirmar que ambos computadores son muy similares. Sin embargo, pese a la leve diferencia, se puede afirmar que, en función del promedio de tiempo de ejecución de solicitudes, el computador 1 es ligeramente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

6. Performance testing tras la revisión del cliente

Tras realizar los cambios pertinentes a las clases relativas de las entidades “contract” y “progress log”, se realiza una nueva prueba de rendimiento para ver si algo ha variado. Estos han sido los resultados:

request-path	response-status	time
Promedio /		5.751914
Promedio /anonymous/system/sign-in		5.564418
Promedio /any/system/welcome		2.863402
Promedio /client/contract/create		22.12810345
Promedio /client/contract/delete		21.26115
Promedio /client/contract/list		14.5029087
Promedio /client/contract/publish		20.380625
Promedio /client/contract/show		19.64503448
Promedio /client/contract/update		27.35587119
Promedio /client/progress-log/create		16.77700444
Promedio /client/progress-log/delete		19.57166
Promedio /client/progress-log/list		15.92327059
Promedio /client/progress-log/publish		16.52836667
Promedio /client/progress-log/show		22.27628947
Promedio /client/progress-log/update		13.08806222
Promedio general		14.91164557

Tiempo de ejecución tras cambios solicitados por el cliente

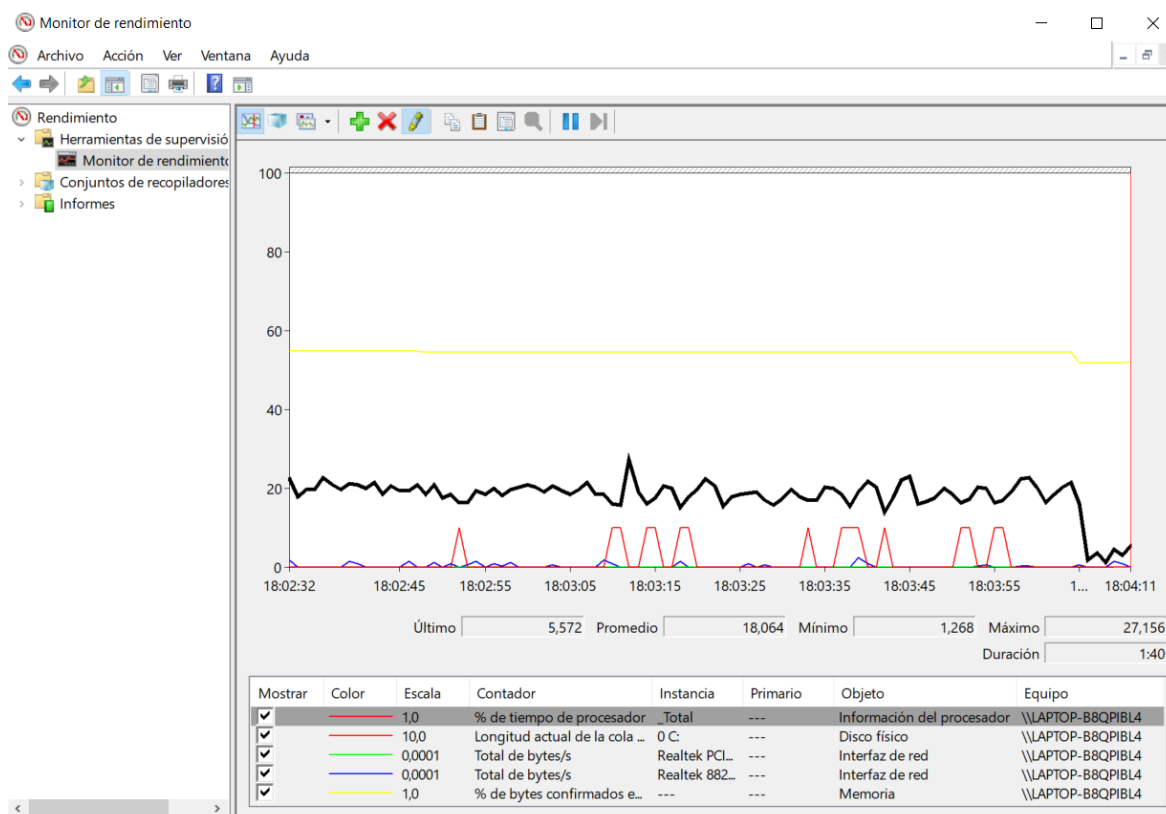
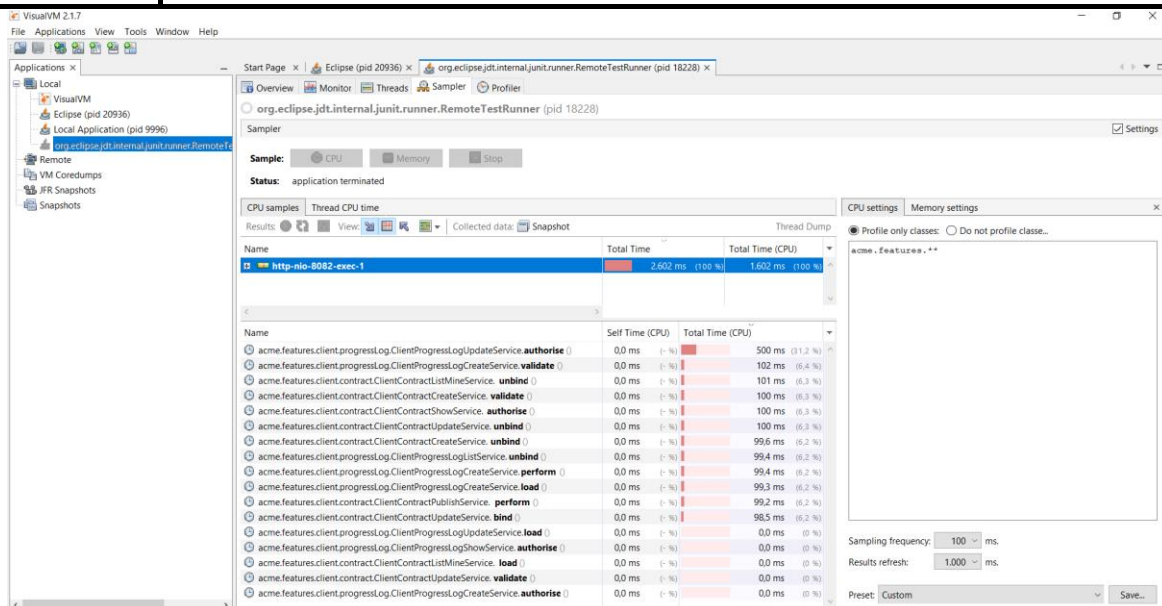


A simple vista se puede observar que el promedio de tiempo general ha disminuido de 21.78 ms a 14.91 ms, lo que supone una reducción de 31%.

Ahora se muestran los resultados obtenidos del rendimiento de la computadora principal donde se ha desarrollado el proyecto durante la ejecución de los casos de prueba:



Testing report - Rubén Pérez Garrido



En la primera imagen correspondiente al “profiler” se puede observar que las funciones que, gracias al Self Time que tiene un valor de 0,0 ms, podemos saber que lo que consume tiempo no son los métodos en sí, sino los métodos internos a los que hace referencia e invoca cada uno.

Por otro lado, en la segunda imagen se puede ver el rendimiento de la computadora durante el tiempo de ejecución de las pruebas finales, ya que al acabar estas, la línea negra tiende a ir a un número más próximo a 0. No se presenta cuello de botella, ya que ninguno de los componentes medidos alcanza el 100% de su capacidad.

Por todo esto, podemos afirmar que la computadora sigue siendo óptima para la ejecución del



sistema.

Finalmente, se vuelve a comparar 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>	
Media	14.91164557
Error típico	0.606214352
Mediana	12.794
Moda	27.2448
Desviación estándar	13.04417779
Varianza de la muestra	170.1505743
Curtosis	27.74425285
Coefficiente de asimetría	3.537428169
Rango	131.5785
Mínimo	1.8818
Máximo	133.4603
Suma	6904.0919
Cuenta	463
Nivel de confianza(95.0%)	1.19127911

<i>Computadora 2</i>	
Media	14.95060216
Error típico	0.91782866
Mediana	13.4418
Moda	1.3424
Desviación estándar	19.74931834
Varianza de la muestra	390.0355748
Curtosis	59.63463332
Coefficiente de asimetría	6.211256578
Rango	253.2216
Mínimo	0.7808
Máximo	254.0024
Suma	6922.1288
Cuenta	463
Nivel de confianza(95.0%)	1.803636132

Interval (ms)	16.10292468	13.72036646
Interval (s)	0.016102925	0.013720366

Interval (ms)	16.75423829	13.146966
Interval (s)	0.016754238	0.01314697

Como se puede comprobar, ambos resultados son muy parecidos como en el apartado anterior, pues ambas computadoras siguen siendo las mismas para ambos casos. Con el z-test podemos comprobar que si el valor del two-tail p-value es mayor que alpha (0.05), los resultados son:

	<i>Computadora 1</i>	<i>Computadora 2</i>
Media	14.91164557	14.95060216
Varianza (conocida)	170	390
Observaciones	463	463
Diferencia hipotética de las medias	0	
z	-0.035422352	
P(Z<=z) una cola	0.485871481	
Valor crítico de z (una cola)	1.644853627	
Valor crítico de z (dos colas)	0.971742962	
Valor crítico de z (dos colas)	1.959963985	

Como se observa, el valor del two-tail p-value es 0.971, cuyo valor está rozando el 1 y muy alejado de alpha, por ende, nuevamente, se puede afirmar que ambas computadoras son muy similares y los resultados de las pruebas son muy similares. Sin embargo, se podría tomar como mejor computadora por muy poco margen, la número 1, por tener una media de ejecución de solicitudes levemente inferior.



7. 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.



8. 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.



Diseño y Pruebas II

C1.002

Testing report - Rubén Pérez Garrido



Diseño y Pruebas II

C1.002

Testing report - Rubén Pérez Garrido