

# Informe de Testing

## INFORMACIÓN DE GRUPO

Group: C1.043

Repository: <https://github.com/DaniFdezCab/DP2-2324-C1-043.git>

### Student #1

UVUS: danfercab

Contact: [danfercab@alum.us.es](mailto:danfercab@alum.us.es)

### Student #2

UVUS: alvmarmun1

Contact: [alvmarmun1@alum.us.es](mailto:alvmarmun1@alum.us.es)

### Student #3

UVUS: fracapgar1

Contact: [fracapgar1@alum.us.es](mailto:fracapgar1@alum.us.es)

### Student #4

UVUS: alepingar

Contact: [alepingar@alum.us.es](mailto:alepingar@alum.us.es)

### Student #5

UVUS: pabberima

Contact: [pabberima@alum.us.es](mailto:pabberima@alum.us.es)

Date: Sevilla Mayo 27, 2024

## TABLA DE CONTENIDOS

Informe de Testing .....	1
INFORMACIÓN DE GRUPO .....	1
TABLA DE CONTENIDOS .....	2
RESUMEN EJECUTIVO .....	3
TABLA DE REVISIONES.....	3
INTRODUCCIÓN.....	3
CONTENIDOS.....	3
CONCLUSION.....	12
BIBLIOGRAFIA.....	12

## RESUMEN EJECUTIVO

El presente informe documenta el proceso de pruebas de testeo del proyecto Acme-SF-D04. Se han llevado a cabo pruebas exhaustivas de los servicios para comprobar la cobertura del código. Adicionalmente, se realizaron intentos de hackeos para evaluar la seguridad del código de las entidades Contract y ProgressLog. Estos esfuerzos garantizan que el sistema no solo funcione correctamente, sino que también sea seguro contra posibles vulnerabilidades.

## TABLA DE REVISIONES

Número de Revisión	Fecha	Descripción
1	23/05/2024	Realización de los .hack y .safe.
2	24/05/2024	Realización del Coverage.
3	25/05/2024	Pruebas de rendimiento.
4	27/05/2024	Realización del documento.

## INTRODUCCIÓN

Este informe documenta las pruebas exhaustivas realizadas en el proyecto Acme-SF-D04 para asegurar la calidad y seguridad del sistema. Se han implementado y verificado varios requisitos críticos, centrados en la cobertura del código y la seguridad de las entidades Contract y ProgressLog.

Las pruebas de cobertura del código se llevaron a cabo mediante un riguroso análisis de los servicios, asegurando que todas las rutas y funciones del código estén adecuadamente cubiertas y probadas. Esto permite garantizar que el sistema funcione correctamente bajo diversas condiciones y que cualquier posible error sea identificado y corregido de manera oportuna.

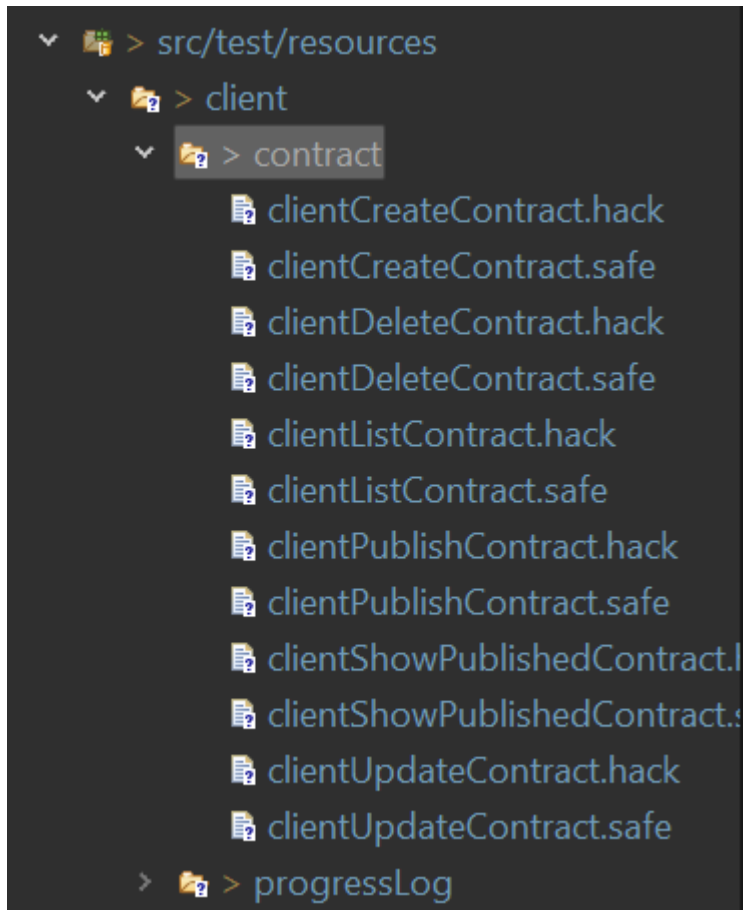
Además, se realizaron intentos de hackeo específicos para evaluar la seguridad de las entidades Contract y ProgressLog. Estas pruebas incluyeron técnicas de penetración para descubrir vulnerabilidades, como inyecciones SQL y problemas de autenticación y autorización.

## CONTENIDOS

### **PRUEBAS FUNCIONALES**

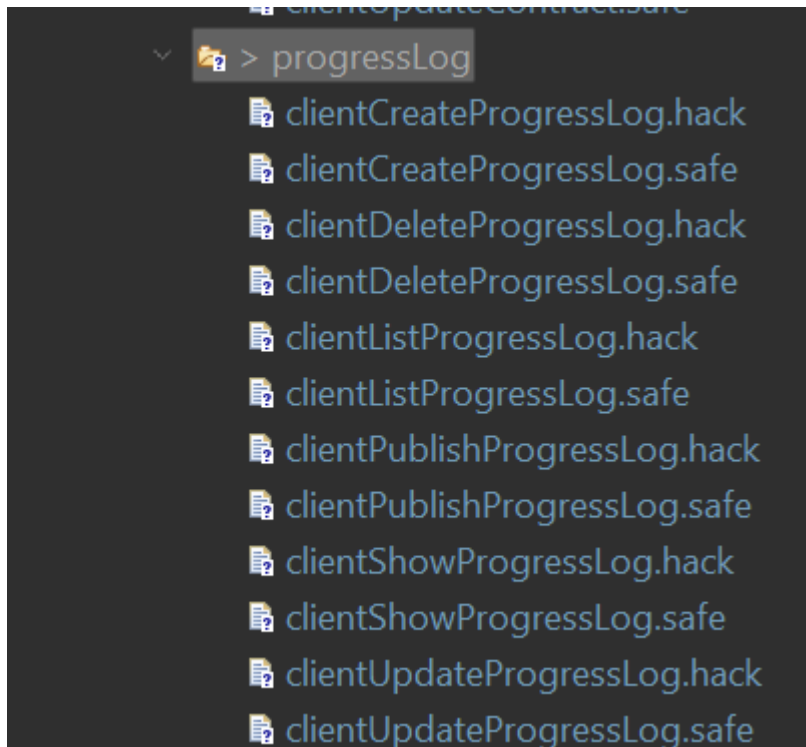
Se realizaron pruebas funcionales .safe y .hack para las dos entidades mencionadas anteriormente (Contract y ProgressLog). Solo se van a explicar detalladamente los tester.trace de Contract ya que el procedimiento es el mismo que para ProgressLog.

Estos son todos los tester.trace que se realizaron para Contract:



Un hack y un safe para cada servicio.

Estos son los de ProgressLog:



Primeramente se van a explicar los .safe:

**ClientCreateContract.safe:** Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos crear un contrato con todos los campos válidos. Una vez se crea el contrato, nos deslogueamos y paramos la traza.

**ClientDeleteContract.safe:** Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos eliminar un contrato que no este publicado. Una vez se elimina el contrato, nos deslogueamos y paramos la traza.

**ClientListContract.safe:** Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos listar todos los contratos de un cliente. Una vez se listan los contratos, nos deslogueamos y paramos la traza.

**ClientPublishContract.safe:** Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos publicar un contrato de un cliente. Una vez se publica, nos deslogueamos y paramos la traza.

**ClientShowContract.safe:** Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos ver un contrato de un cliente. Una vez se muestra, nos deslogueamos y paramos la traza.

**ClientUpdateContract.safe:** Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos actualizar un contrato de un cliente. Una vez se actualiza, nos deslogueamos y paramos la traza.

A continuación se explican los .hack:

**ClientCreateContract.hack:** Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos crear un contrato vacío (dará error). Se ponen todos los campos del contrato correctos y se empieza a probar cada campo con datos que deberían de dar error. Una vez han saltado todas las restricciones de cada campo de contrato, nos deslogueamos y paramos la traza.

**ClientDeleteContract.hack:** Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos eliminar un contrato del cliente2. Una vez nos sale la pantalla de error, nos deslogueamos y nos logueamos como otro rol, en mi caso manager1. Se vuelve a intentar borrar un contrato del cliente2. Cuando se muestra el error nos deslogueamos y paramos la traza.

**ClientListContract.hack:** Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos listar los contratos del cliente2. Una vez nos sale la pantalla de error, nos deslogueamos y nos logueamos como otro rol, en mi caso manager1. Se vuelve a intentar listar los contratos del cliente2. Cuando se muestra el error nos deslogueamos y paramos la traza.

**ClientPublishContract.hack:** Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos publicar un contrato del cliente2. Una vez nos sale la pantalla de error, nos deslogueamos y nos logueamos como otro rol, en mi caso manager1. Se vuelve a intentar publicar un contrato del cliente2. Cuando se muestra el error nos deslogueamos y paramos la traza.

**ClientShowContract.hack:** Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos ver un contrato del cliente2. Una vez nos sale la pantalla de error, nos deslogueamos y nos logueamos como otro rol, en mi caso manager1. Se vuelve a intentar ver un contrato del cliente2. Cuando se muestra el error nos deslogueamos y paramos la traza.

**ClientUpdateContract.hack:** Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos updatear un contrato vacío (dará error). Se ponen todos los campos del contrato correctos y se empieza a probar cada campo con datos que deberían de dar error. Una vez han saltado todas las restricciones de cada campo de contrato, nos deslogueamos. Nos logueamos como cliente1 y intentamos updatear un contrato del cliente2. Una vez nos sale la pantalla de error, nos deslogueamos y nos logueamos como otro rol, en mi caso manager1. Se vuelve a intentar updatear un contrato del cliente2. Cuando se muestra el error nos deslogueamos y paramos la traza.

**SE REALIZA MISMO PROCEDIMIENTO DE LOS .SAFE Y .HACK CON PROGRESSLOG.**

**Una vez se han realizado todos los .safe y .hack runeamos tester#replayer y vemos el coverage de cada servicio:**

acme.features.client.contract	81,7 %	997	223	1.220
> ClientContractPublishService.java	68,6 %	212	97	309
> ClientContractDeleteService.java	62,7 %	131	78	209
> ClientContractUpdateService.java	90,7 %	206	21	227
> ClientContractCreateService.java	91,3 %	179	17	196
> ClientContractShowService.java	96,4 %	163	6	169
> ClientContractListMineService.java	94,6 %	70	4	74
> ClientContractController.java	100,0 %	36	0	36

Como se puede comprobar, los tester.trace realizados anteriormente cubren el 90% de todos los servicios excepto en el Publish y Delete por que no se utilizan el unbind de su código en las trazas que se han grabado.

```

@Override
public void unbind(final Contract object) {
    assert object != null;

    Collection<Project> projects;
    SelectChoices choices;

    projects = this.clientContractRepository.findAllProjects();
    choices = SelectChoices.from(projects, "code", object.getProject());

    Dataset dataset;

    dataset = super.unbind(object, "code", "instantiationMoment", "providerName", "customerName", "goals", "budget");
    dataset.put("project", choices.getSelected().getKey());
    dataset.put("projects", choices);

    super.getResponse().addData(dataset);
}

```

En ProgressLog sucede lo mismo:

acme.features.client.progressLog	82,7 %	925	193	1.118
> ClientProgressLogDeleteService.java	68,1 %	145	68	213
> ClientProgressLogPublishService.java	68,7 %	149	68	217
> ClientProgressLogUpdateService.java	91,1 %	195	19	214
> ClientProgressLogCreateService.java	91,1 %	184	18	202
> ClientProgressLogShowService.java	88,7 %	94	12	106
> ClientProgressLogListService.java	93,9 %	123	8	131
> ClientProgressLogController.java	100,0 %	35	0	35

```

@Override
public void unbind(final ProgressLog object) {
    assert object != null;

    Dataset dataset;

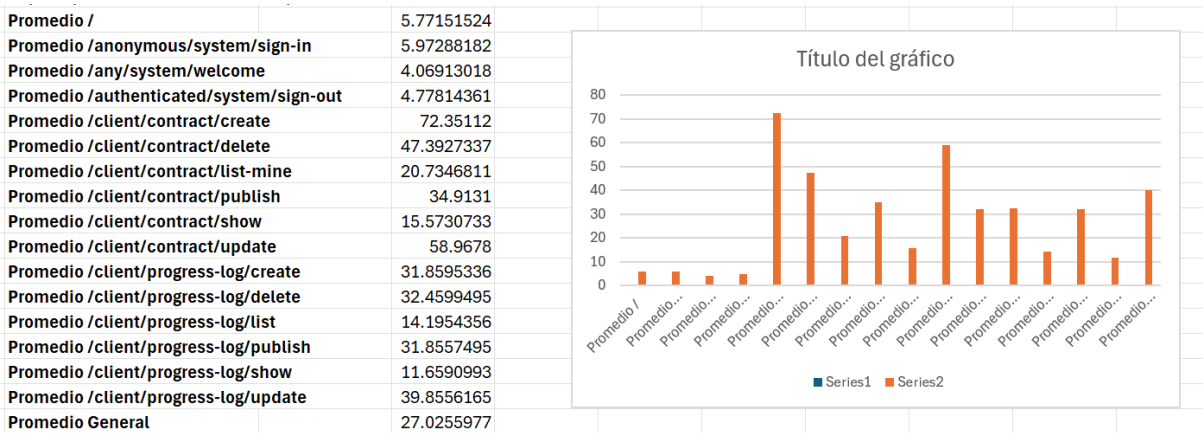
    dataset = super.unbind(object, "recordId", "completeness", "comment", "registrationMoment", "responsiblePerson");
    dataset.put("masterId", object.getContract().getId());
    dataset.put("draftMode", object.isDraftMode());

    super.getResponse().addData(dataset);
}

```

# PRUEBAS DE RENDIMIENTO

Con el seguimiento de las diapositivas obtenemos el promedio que tarda en obtener los resultados de cada ruta.



Tras instalar herramientas de para análisis y usar la estadística descriptiva con un nivel de confianza del 95%, obtenemos:

Columna1				
		Interval(ms)	10.9096689	15.9029039
		Interval(s)	0.01090967	0.0159029
Media	13.4062864			
Error típico	1.26854622			
Mediana	5.5706			
Moda	#N/D			
Desviación estándar	21.7510366			
Varianza de la muestra	473.107595			
Curtosis	53.8852125			
Coefficiente de asimetría	5.78251919			
Rango	254.380699			
Mínimo	1.4672			
Máximo	255.847899			
Suma	3941.4482			
Cuenta	294			
Nivel de confianza(95.0%)	2.49661751			

Añadimos índices a las entidades que estamos testeando:

(Contract)

```
@Table(indexes = {
    @Index(columnList = "client_id"), @Index(columnList = "project_id"), @Index(columnList = "code")
})
```

(ProgressLog)

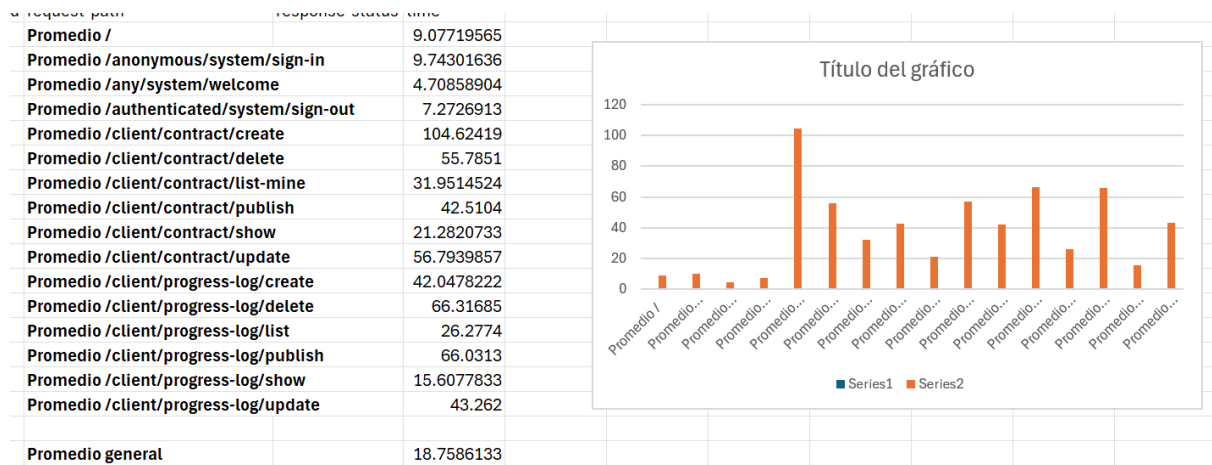


```

9@Setter
10@Table(indexes = {
11    @Index(columnList = "contract_id")
12})

```

Obtenemos todos las graficas anteriores tras la implementación de los índices:



Columna1				
		Interval(ms)	14.7808602	22.7363663
Media	18.7586133	Interval(s)	0.01478086	0.02273637
Error típico	2.02112			
Mediana	7.8262			
Moda	5.5596			
Desviación es	34.6549889			
Varianza de la	1200.96826			
Curtosis	105.081105			
Coeficiente d	8.53964064			
Rango	474.6586			
Mínimo	2.3385			
Máximo	476.9971			
Suma	5515.0323			
Cuenta	294			
Nivel de confi	3.97775303			

Vamos a compararlas:

Before	After									
87.794499	141.5444									
33.290599	21.7135									
12.035299	26.0613									
22.240199	47.9436									
5.5638	6.3737									
67.1194	134.0963									
36.565001	77.2704									
255.847899	476.9971									
43.4829	56.8068									
80.9867	91.5674									
62.2918	60.8523									
37.9151	66.0507									
84.590301	79.1886									
49.0589	45.6554									
7.2016	7.3024									
16.547799	8.2948									
4.6622	6.0302									
7.389301	8.8									
4.3361	8.0881									
6.0852	5.3995									
45.1959	41.3652	Interval(ms)		10.9096689	15.9029039	Interval(ms)		14.78086023	22.7363663	
4.5433	7.2923	Interval(s)		0.010909669	0.0159029	Interval(s)		0.01478086	0.02273637	

La mejor forma para compararlo es realizando un Z-Test:

Prueba z para medias de dos muestras		
	Before	After
Media	13.1524017	18.3395491
Varianza (conocida)	473	1200
Observaciones	293	293
Diferencia hipotética de las medias	0	
z	-2.1707731	
P(Z<=z) una cola	0.01497416	
Valor crítico de z (una cola)	1.64485363	
Valor crítico de z (dos colas)	0.02994833	
Valor crítico de z (dos colas)	1.95996398	

Nos debemos de fijar en la fila que esta marcada.

Cuanto mas cerca a cero tenga este valor, mayor habrá sido la mejora de rendimiento. Sin embargo, cuanto mas cerca de 1 sea, menos significativo serán los cambios realizados en cuanto a rendimiento.

En nuestro caso como este valor es 0,0299. Por lo tanto, ha habido una notoria mejora de rendimiento tras añadir los índices.

## Utilización de VisualVm

Con el uso de VisualVm podemos ver el tiempo dedicado a cada método mientras se ha realizado el tester#replayer. De esta forma podemos ver los métodos que mas tiempo han tardado en ejecutarse. Esto nos proporciona una idea de los métodos que limitan a otros en cuanto a tiempos de uso.

Results: View: | Collected data: Snapshot Thread Dump

Name	Total Time	Total Time (CPU)
http-nio-8082-exec-1	2.355 ms (100 %)	2.113 ms (100 %)
main	172 ms (100 %)	172 ms (100 %)

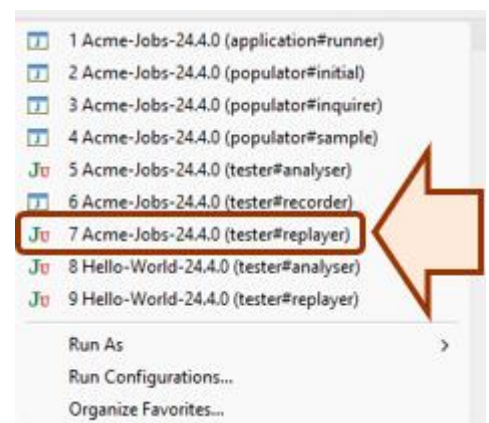
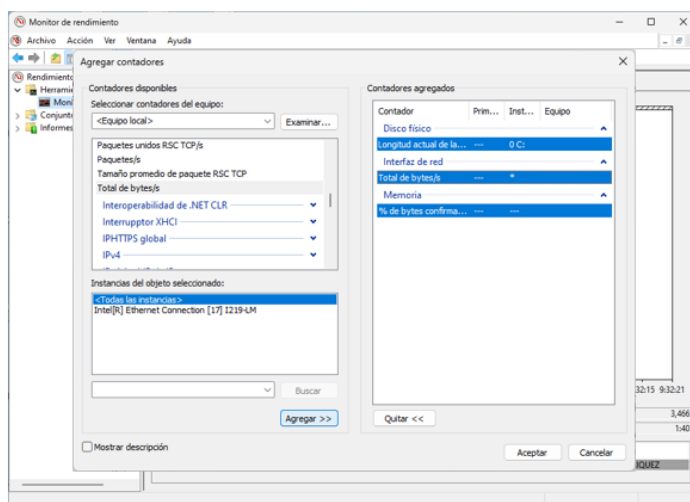
Name	Self Time (CPU)	Total Time (CPU)
acme.features.client.contract.ClientContractCreateService. bind ()	0,0 ms (0 %)	512 ms (22,1 %)
acme.features.client.contract.ClientContractUpdateService. bind ()	0,0 ms (0 %)	299 ms (12,9 %)
acme.features.client.progressLog.ClientProgressLogCreateService. bind ()	0,0 ms (0 %)	274 ms (11,9 %)
acme.features.client.progressLog.ClientProgressLogUpdateService. bind ()	0,0 ms (0 %)	197 ms (8,5 %)
acme.features.client.contract.ClientContractListMineService. load ()	0,0 ms (0 %)	148 ms (6,4 %)
acme.features.client.contract.ClientContractCreateService. unbind ()	0,0 ms (0 %)	103 ms (4,4 %)
acme.features.client.contract.ClientContractUpdateService. unbind ()	0,0 ms (0 %)	51,4 ms (2,2 %)
acme.features.client.contract.ClientContractShowService. authorise ()	0,0 ms (0 %)	49,2 ms (2,1 %)
acme.features.client.contract.ClientContractCreateService. validate ()	0,0 ms (0 %)	48,8 ms (2,1 %)
acme.features.client.progressLog.ClientProgressLogUpdateService. authorise ()	0,0 ms (0 %)	43,8 ms (1,9 %)
acme.features.administrator.banner.AdministratorBannerController. <init> ()	0,0 ms (0 %)	35,6 ms (1,5 %)
acme.features.client.progressLog.ClientProgressLogShowService. authorise ()	0,0 ms (0 %)	35,0 ms (1,5 %)
acme.features.client.progressLog.ClientProgressLogDeleteService. bind ()	0,0 ms (0 %)	30,2 ms (1,3 %)
acme.features.client.contract.ClientContractDeleteService. bind ()	0,0 ms (0 %)	28,7 ms (1,2 %)
acme.features.client.contract.ClientContractDeleteService. perform ()	0,0 ms (0 %)	22,8 ms (1 %)

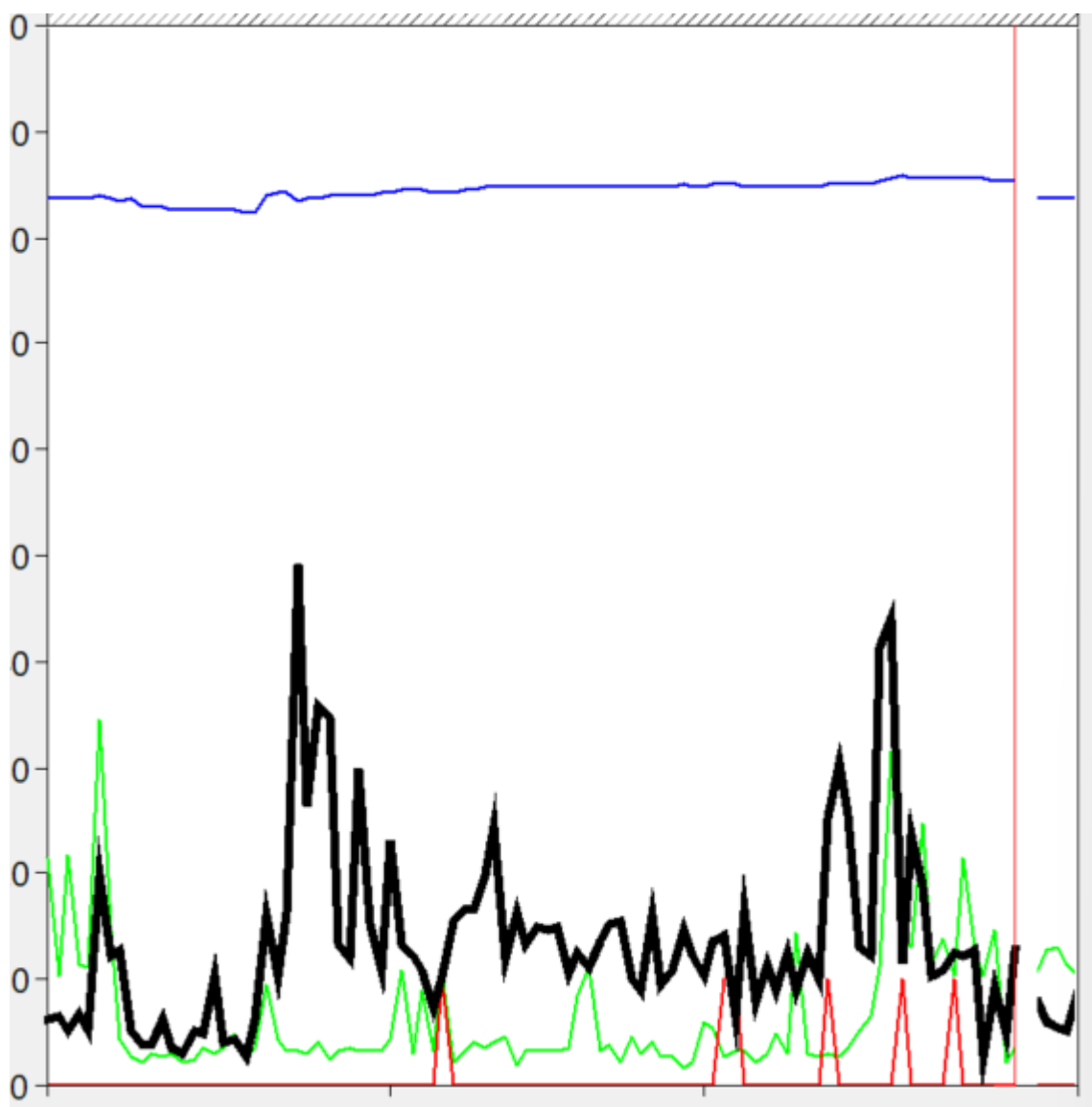
Como se puede ver, el método bind() es el que mas tarda en realizarse con diferencia. Sobre todo el del create con 512ms. El resto de métodos (quitando los bind()) no superan los 50ms.

Hay que puntualizar que el Self Time es de cero milisegundos. Eso significa que no es el método el que está consumiendo demasiado tiempo, sino los métodos que invoca.

## MONITOR DE RENDIMIENTO

Iniciamos el monitor de rendimiento y lo configuramos:





Se puede apreciar el rendimiento de la CPU mientras se ejecuta el recorder#replayer. La red de este ordenador está siendo utilizada moderadamente, pero no es un claro cuello de botella ya que no se está usando al 100% del tiempo. La CPU y la memoria están poco utilizadas, por lo que están lejos de ser un cuello de botella.

## CONCLUSION

En conclusión, la implementación y el proceso de pruebas del proyecto Acme-SF-D04 han sido exitosos, cumpliendo con los criterios de calidad y seguridad establecidos. Las pruebas exhaustivas de cobertura del código y los intentos de hackeo realizados aseguran que el sistema funcione

correctamente y esté protegido contra posibles vulnerabilidades. Las entidades clave, como Contract y ProgressLog, fueron sometidas a rigurosas evaluaciones tanto funcionales como de seguridad, garantizando su robustez y fiabilidad.

Los resultados de las pruebas funcionales, tanto .safe como .hack, demostraron una cobertura del 90% de los servicios, con algunas excepciones en los métodos Publish y Delete debido a la falta de uso del unbind en las trazas grabadas. Las pruebas de rendimiento mostraron mejoras significativas tras la implementación de índices, evidenciadas por un valor de 0,0299 en el Z-Test, lo cual indica una notable mejora en el rendimiento del sistema.

El análisis con VisualVM reveló que los métodos bind() son los que más tiempo de ejecución requieren, mientras que otros métodos se ejecutan en menos de 50ms. Es importante destacar que el Self Time es de cero milisegundos, indicando que el tiempo de ejecución prolongado se debe a los métodos invocados, no al propio método en sí.

El monitoreo de rendimiento indicó que la red está siendo utilizada moderadamente y no representa un cuello de botella, mientras que la CPU y la memoria están subutilizadas. Esto sugiere que el sistema tiene capacidad adicional para manejar cargas de trabajo más intensas sin comprometer el rendimiento.

## BIBLIOGRAFIA

Intencionalmente en blanco.