

Universidad de Sevilla
Escuela Técnica Superior de Ingeniería Informática
DP2-Informe de Testing Student 5 D4



Grado en Ingeniería Informática – Ingeniería del Software
Diseño y Pruebas II
Curso 2023 – 2024

<u>Group:</u>	C1.016
<u>Repository:</u>	https://github.com/luchersol/Acme-SF-D04
<u>Student #5</u>	
	
UVUS:	dangalcan
Name:	Galván Cancio, Daniel
Email:	dangalcan@alum.us.es
Date:	Sevilla Mayo 04, 2024

Índice de contenido

1. Versiones	2
2. Testing funcional	2
3. Testing de rendimiento	2
4. Bibliografía	2

1. Versiones

Versión	Fecha	Autor
1.0	16/05/2024	Daniel Galván Cancio
1.1	22/05/2024	Daniel Galván Cancio
1.2	26/05/2024	Daniel Galván Cancio

2. Testing funcional

- Para la Task 6:

- listar las codeAudits de un auditor:

El caso de uso bajo prueba consiste en listar las auditorías de un auditor. En este caso de prueba lo que hice fue iniciar sesión con cada auditor (auditor1, auditor2 y auditor3) y listar los codeAudits de dichos auditores pulsando el botón de “My Code Audits”. Así tengo registrados los casos de más de un codeAudit, un solo codeAudit, y ningún codeAudit.

Con respecto al hacking probé a listar todas las code audits sin ser auditor.

- mostrar los detalles de un codeAudit:

El caso de uso bajo prueba consiste en mostrar los valores de los atributos de una auditoría de código. Lo que hice fue iniciar sesión como auditor1, y pulsar en “My Code Audits”. Una vez allí, fue cuestión de clickar en cada codeAudit para ver sus detalles. Así tengo registrados la mayor cantidad de combinaciones válidas de los valores de los atributos posible.

En el caso del hacking lo que hice fue tratar de visualizar un codeAudit que no existe, otro que no es mío (siendo auditor 2 intentar ver codeAudits del auditor1) e intentar ver un codeAudit sin ser auditor (con otro rol y sin estar autenticado en el sistema).

- crear un codeAudit:

El caso de uso bajo prueba consiste en crear una auditoría de código. Aquí estuve siguiendo los consejos estudiados en teoría. Primero inicié sesión como auditor2, y entré al apartado de My Code Audits. Tras esto, le di al botón de crear, e intenté enviar el formulario vacío. Una vez da error, voy probando atributo por atributo los valores incorrectos. Cuando sucede que pruebo todo lo incorrecto, comienzo a crear entidades con valores en los límites de los permitidos según el negocio (la máxima longitud en las corrective actions, fechas de ejecución el 29 de febrero, el fin de año, el inicio de año, etc). Además, varío los valores de los enumerados para dar un poco más de variedad.

Para el tema del hacking probé a intentar crear un codeAudit sin ser auditor y sin estar autenticado.

- eliminar un codeAudit:

El caso de uso bajo prueba consiste en eliminar una auditoría de código. Aquí simplemente inicié sesión como auditor1 y eliminé auditorías de código sin registros y con registros.

Para el hacking lo que hice fue, como auditor1, tratar de eliminar un codeAudit que no existe, uno que esté publicado y otro que no sea mío. Luego, probé a hacer lo mismo pero con otro rol y sin estar autenticado.

- editar un codeAudit:

El caso de uso bajo prueba consiste en editar uno o más valores de los atributos de una auditoría de código. Hice algo parecido con lo indicado en el create pero añadí un par de detalles extras. También edité un codeAudit sin alterar ninguno de sus valores, edité un codeAudit sin registros, y otro con varios registros. En este caso de prueba me dí cuenta que debí implementar la restricción de que la nueva fecha de ejecución que quiera establecer debe ser anterior a todas las fechas de inicio de los registros de auditoría, lo cuál fue bastante lógico pero no caí en su momento cuando estuve implementando las entregas 2 y 3. Obviamente tuve que repetir el test, pero mereció la pena a mi parecer, ya que así pude adaptar los cambios al publish antes de testearlo, lo cuál me daba más seguridad de que estaba haciendo las cosas bien.

Para el hacking lo que hice fue, como auditor1, tratar de editar un codeAudit que no existe, uno que esté publicado y otro que no sea mío. Luego, probé a hacer lo mismo pero con otro rol y sin estar autenticado.

- publicar un codeAudit:

El caso de uso bajo prueba consiste en publicar una auditoría de código. Fue exactamente idéntico al testeо de la edición de un codeAudit, solo que tuve que añadir el detalle de probar que no podía publicar un codeAudit que tuviera registros sin publicar.

Para el hacking lo que hice fue, como auditor1, tratar de publicar un codeAudit que no existe, uno que esté publicado y otro que no sea mío. Luego, probé a hacer lo mismo pero con otro rol y sin estar autenticado.

- listar los auditRecords de un codeAudit:

El caso de uso bajo prueba consiste en listar los registros de auditoría de una auditoría de código. Para probarlo que hice fue iniciar sesión como auditor1 y clickar en “My Code Audits”. Una vez allí clické sobre todos los codeAudits y fuí dándole al botón de “show audit records” en cada uno de ellos. Así me aseguraba de tener ejemplos de todas las combinaciones de valores válidos posibles.

Para el hack lo que hice fue tratar de listar los audit records de un codeAudit que no esté publicado, mientras no estoy autenticado y estando autenticado como developer1. Además de eso, traté de listar los auditRecords de un codeAudit que no me pertenece (siendo auditor2). Por último también probé a intentar listar los auditRecords de un codeAudit que no existe.

- Para la Task 7:

- mostrar los detalles de un auditRecord:

El caso de uso bajo prueba consiste en mostrar los valores de los atributos de un registro de auditoría. Lo que hice fue iniciar sesión como auditor1, e inspeccionar todos y cada uno de los auditRecords que tiene disponible este usuario para ver sus detalles.

Para el hack, lo que hice fué intentar visualizar los detalles de un auditRecord que no existe, uno que pertenece a un codeAudit que no es mío, y luego, cerré sesión. Después de esto, traté de visualizar auditRecords de codeAudits no publicados siendo developer (habiendo iniciado previamente sesión como developer1) y sin estar autenticado.

- crear un AuditRecord:

El caso de uso bajo prueba consiste en crear un nuevo registro de auditoría de código. Para este caso de prueba inicié sesión como auditor y seguí la metodología vista en clase para probar todas las restricciones en las propiedades de los auditRecords. Estuve creando también auditRecords con parámetros válidos para tener ejemplos variados de los posibles valores de los atributos de los auditRecords.

Para el hacking intenté crear un auditRecord en un codeAudit que no sea mío, en uno que esté publicado e intentar lo mismo pero con otro rol y sin estar autenticado.

- eliminar un AuditRecord:

El caso de uso bajo prueba consiste en eliminar un registro de auditoría. Fué un testeo sencillo, ya que solo me ceñí a iniciar sesión como auditor1 y eliminar unos cuantos auditRecords.

Para el hacking lo que hice fue, como auditor1, tratar de eliminar un auditRecord que no existe y uno que esté publicado. Como auditor2 probé lo mismo pero con otros que no sean míos. Luego, probé a hacer lo mismo pero con otro rol y sin estar autenticado.

- editar un AuditRecord:

El caso de uso bajo prueba consiste en alterar los valores de uno o más atributos de un registro de auditoría. Aquí seguí lo mismo que mencioné anteriormente pero en esta ocasión con la edición de los auditRecords.

Para el hacking lo que hice fue, como auditor1, tratar de editar un auditRecord que no existe, uno que esté publicado. Con auditor2 probé lo mismo pero además con otro que no sea mío. Luego, probé a hacer lo mismo pero con otro rol y sin estar autenticado.

- publicar un AuditRecord:

El caso de uso bajo prueba consiste en publicar un registro de auditoría. En este caso seguí los mismos pasos que los descritos en los otros casos, apoyándome además en la metodología propuesta en las transparencias.

Para el hacking lo que hice fue, como auditor1, tratar de publicar un auditRecord que no existe, uno que esté publicado y con el auditor2 otro que no sea mío. Luego, probé a hacer lo mismo pero con otro rol y sin estar autenticado.

-Instrucciones "no testeadas":

A lo largo de este documento hemos comentado las funcionalidades testeadas así como un resumen de las pruebas realizadas. No obstante, la cobertura de pruebas no es del 100% debido a ciertas líneas de código que no se pudieron cubrir. A lo largo de esta sección hablaremos de dichas líneas y mostraremos la cobertura obtenida. La mayoría de instrucciones que aparecen en amarillo son double checks (para asegurar un buen funcionamiento ante hackeo) que son estrictamente necesarios. No obstante, también encontramos como instrucciones no cubiertas los “assert object != null”, que simplemente no pueden ponerse en verde porque dichas validaciones son por seguridad y por recomendación a la hora de implementar nuestros métodos en el contexto tecnológico de la asignatura.

```
    @Override
    public void authorise() {
        boolean status;
        int masterId;
        CodeAudit codeAudit;
        masterId = super.getRequest().getData("masterId", int.class);
        codeAudit = this.repository.findCodeAuditById(masterId);
        status = codeAudit != null && codeAudit.getDraftMode() && super.getRequest().getPrincipal().hasRole(codeAudit.getAuditor());
        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        int masterId;
        CodeAudit codeAudit;
        masterId = super.getRequest().getData("masterId", int.class);
        codeAudit = this.repository.findCodeAuditById(masterId);
        AuditRecord object;
        object = new AuditRecord();
        object.setCodeAudit(codeAudit);
        object.setDraftMode(true);
        super.getBuffer().addData(object);
    }

    @Override
    public void bind(final AuditRecord object) {
        assert object != null;
        int masterId;
        CodeAudit codeAudit;
        masterId = super.getRequest().getData("masterId", int.class);
        codeAudit = this.repository.findCodeAuditById(masterId);
        super.bind(object, "code", "startDate", "endDate", "link", "mark");
        object.setCodeAudit(codeAudit);
    }

    @Override
    public void validate(final AuditRecord object) {
        assert object != null;
```

Además, también salen en amarillo condiciones que no cubren todas las posibilidades, como el caso del ejemplo. Lo que no “sabe” eclipse es que por ejemplo en la mostrada en la imagen, hay una condición que no es posible validar. En este caso del ejemplo sería la condición de un codeAudit nulo, con su draftMode en verdadero pero con el rol sin ser auditor o siendolo. Esto no es posible porque en el momento en el que el codeAudit es nulo, no puede tener draftMode, por lo que esta condición no se puede cubrir.

En la siguiente imagen podemos observar cosas parecidas pero menos evidentes. Comencemos con los boolean “notNull” de las validaciones. A priori puede ser cierto que sean innecesarios por el tema de que el framework se asegura de que los valores no sean nulos, pero hay un caso de prueba que causa un panic que no se contempla si no los ponemos. Cuando introducimos textos maliciosos del estilo “ ” o caracteres invisibles. En esas situaciones se le hace un .trim() y entra como null a los validate. Es necesario también verificarlo porque para entrar en el if solo comprueba el atributo de la condición, y no tiene en cuenta los demás. Por eso deberemos poner notNull. De lo contrario los atributos inválidos se meten dentro del MomentHelper y su assert genera un panic. Por eso validamos eso tanto en el publish, como en el create, como en el update de los auditRecord. Con respecto a la validación del codeAudit, es para prevenir ataques de hacking, ya que debemos comprobar que un auditor no cree registros de una auditoría que no es suya. De manera gráfica en los tests no hemos podido comprobar eso, pero lo tenemos así para prevenir hackeos con postman. Además, también aseguramos de paso que la auditoría esté en modo borrador, para evitar también que se puedan modificar los datos de una auditoría ya publicada.

En conclusión, aparece en amarillo por seguridad, pero si existe alguna duda con respecto a que no probé las combinaciones adecuadas, pueden observarse las trazas y/o el analyzer.

```

@Override
public void validate(final AuditRecord object) {
    assert object != null;
    if (!super.getBuffer().getErrors().hasErrors("code")) {
        AuditRecord ar = this.repository.findAuditRecordByCode(object.getCode());
        super.state(ar == null, "code", "auditor.auditRecord.form.error.duplicated");
    }
    if (!super.getBuffer().getErrors().hasErrors("startDate")) {
        boolean notNull = object.getCodeAudit().getExecution() != null;
        Boolean timeConcordance = notNull && MomentHelper.isAfter(object.getStartDate(), object.getCodeAudit().getExecution());
        super.state(timeConcordance, "startDate", "auditor.auditRecord.form.error.badStartDate");
    }
    if (!super.getBuffer().getErrors().hasErrors("endDate")) {
        boolean notNull = object.getStartDate() != null;
        Boolean timeConcordance = notNull && MomentHelper.isAfter(object.getEndDate(), object.getStartDate());
        super.state(timeConcordance, "endDate", "auditor.auditRecord.form.error.badTimeConcordance");
    }
    if (!super.getBuffer().getErrors().hasErrors("endDate")) {
        boolean notNull = object.getStartDate() != null;
        Boolean goodDuration = notNull && MomentHelper.isLongEnough(object.getEndDate(), object.getStartDate(), 1, ChronoUnit.HOURS);
        super.state(goodDuration, "endDate", "auditor.auditRecord.form.error.notEnoughDuration");
    }
    if (!super.getBuffer().getErrors().hasErrors("codeAudit")) {
        CodeAudit ca = object.getCodeAudit();
        Auditor a = this.repository.findOneAuditorById(super.getRequest().getPrincipal().getActiveRoleId());
        boolean codeAuditIsYours = ca.getAuditor().getId() == a.getId();
        super.state(codeAuditIsYours && ca.getDraftMode(), "codeAudit", "auditor.auditRecord.form.error.codeAudit");
    }
    super.validateSpam(object);
}

```

Aquí vemos algo parecido: Este tipo de condiciones no pueden cubrirse en su totalidad ya que la casuística de la condición no puede cubrirse de forma total. En este caso es porque si para que haya un auditor, el auditRecord debe ser distinto de nulo, la posibilidad de que el auditor no sea nulo pero el auditRecord sí no es algo factible, por la dependencia entre ambas variables. Este mismo caso sucede también con los codeAudits.

```
@Override  
public void authorise() {  
    boolean status;  
    int masterId;  
    AuditRecord auditRecord;  
    Auditor auditor;  
    masterId = super.getRequest().getData("id", int.class);  
    auditRecord = this.repository.findAuditRecordById(masterId);  
    auditor = auditRecord == null ? null : auditRecord.getCodeAudit().getAuditor();  
    status = super.getRequest().getPrincipal().hasRole(auditor) && auditRecord != null;  
    super.getResponse().setAuthorised(status);  
}
```

Continuemos estudiando otros casos:

En el caso del codeAudit, tenemos varios double checks en sus validaciones. Por una parte validamos que la nota sea válida, ya que aunque se gestiona sola por la aplicación para evitar ataques de hacking verificamos que no sea ni F_MINUS ni F en el servicio del publish, y por otro lado validamos que todos los auditRecords hayan sido publicados (esta validación si está cubierta).

Con la validación de la fecha de ejecución, no está del todo comprobada porque nuevamente, no puede pasar que la fecha máxima sea nula y no se cumpla la condición del isAfter, ya que en tal caso, la fecha máxima debería no ser nula.

```

5     @Override
6     public void validate(final CodeAudit object) {
7         assert object != null;
8         CodeAudit ca = this.repository.findCodeAuditWithCode(object.getCode());
9
10        if (!super.getBuffer().getErrors().hasErrors("code"))
11            super.state(ca == null || ca.getId() == object.getId(), "code", "auditor.codeAudit.form.error.duplicated");
12
13        if (!super.getBuffer().getErrors().hasErrors("mark")) {
14            Mark mark = this.repository.findCodeAuditMark(object.getId()).get(0);
15            Boolean isPossibleMark = mark != null && mark.toString() != "F_MINUS" && mark.toString() != "F";
16            super.state(isPossibleMark, "mark", "auditor.codeAudit.form.error.invalidMarkForPublish");
17            object.setMark(mark);
18        }
19
20        if (!super.getBuffer().getErrors().hasErrors("mark")) {
21            Integer notPublishedAuditRecord = this.repository.countNotPublishedAuditRecordsOfCodeAudit(object.getId());
22            super.state(notPublishedAuditRecord == 0, "mark", "auditor.codeAudit.form.error.notAllAuditRecordArePublished");
23        }
24
25        if (!super.getBuffer().getErrors().hasErrors("execution")) {
26            Date maximumDate = this.repository.findMaximumValidExecutionDate(object.getId());
27            Boolean validExecution = maximumDate == null || MomentHelper.isAfter(maximumDate, object.getExecution());
28            super.state(validExecution, "execution", "auditor.codeAudit.form.error.badExecution");
29        }
30
31        if (!super.getBuffer().getErrors().hasErrors("project")) {
32            Boolean isDraftMode = this.repository.projectIsDraftMode(object.getProject().getId());
33            super.state(isDraftMode != null && !isDraftMode, "project", "auditor.codeAudit.form.error.notPublishedProject");
34        }
35
36        super.validateSpam(object);
37    }

```

Para concluir, mencionaremos 2 detalles importantes:

Por un lado, la validación del proyecto no se cubre de manera total porque no puede ocurrir que el proyecto sea nulo y el draftMode tenga algún valor (tanto true como false). Es importante tener en cuenta que el draftMode no sea nulo porque en caso de que el proyecto que se haya seleccionado mediante hacking sea uno que no exista, el repositorio devolvería null, y por eso hay que tener ese caso en cuenta.

```

if (!super.getBuffer().getErrors().hasErrors("project")) {
    Boolean isDraftMode = this.repository.projectIsDraftMode(object.getProject().getId());
    super.state(isDraftMode != null && !isDraftMode, "project", "auditor.codeAudit.form.error.notPublishedProject");
}

```

Por otro lado, tenemos el caso de esta condición, que también es imposible cubrir del todo, ya que para que el auditor exista el codeAudit debe ser distinto de nulo, entonces no puede ocurrir que el codeAudit sea null y que el auditor exista.

```

2@Override
3public void authorise() {
4    boolean status;
5    int masterId;
6    CodeAudit codeAudit;
7
8    masterId = super.getRequest().getData("masterId", int.class);
9    codeAudit = this.repository.findCodeAuditById(masterId);
10   status = codeAudit != null && super.getRequest().getPrincipal().hasRole(codeAudit.getAuditor());
11
12   super.getResponse().setAuthorised(status);
13}
14
15@Override

```

-Cobertura alcanzada:

De manera conjunta conseguí los siguientes valores de cobertura:

> acme.features.auditor.auditRecord	95,0 %	1.666	88	1.754
> acme.features.auditor.codeAudit	94,5 %	1.364	80	1.444

De manera desglosada estos son los porcentajes:

- Para el codeAudit:

> acme.features.auditor.codeAudit	94,5 %	1.364	80	1.444
> AuditorCodeAuditPublishService.java	94,5 %	381	22	403
> AuditorCodeAuditCreateService.java	92,8 %	220	17	237
> AuditorCodeAuditUpdateService.java	94,8 %	308	17	325
> AuditorCodeAuditDeleteService.java	93,0 %	212	16	228
> AuditorCodeAuditListMineService.java	94,7 %	71	4	75
> AuditorCodeAuditShowService.java	97,1 %	136	4	140
> AuditorCodeAuditController.java	100,0 %	36	0	36

- Para el auditRecord:

> acme.features.auditor.auditRecord	95,0 %	1.666	88	1.754
> AuditorAuditRecordCreateService.java	94,8 %	366	20	386
> AuditorAuditRecordPublishService.java	95,2 %	401	20	421
> AuditorAuditRecordUpdateService.java	94,9 %	372	20	392
> AuditorAuditRecordDeleteService.java	92,3 %	193	16	209
> AuditorAuditRecordListService.java	95,3 %	162	8	170
> AuditorAuditRecordShowService.java	97,2 %	137	4	141
> AuditorAuditRecordController.java	100,0 %	35	0	35

3. Testing de rendimiento

En esta sección hablaremos del rendimiento de nuestro código, analizando los métodos más costosos así como haciendo una comparativa de su ejecución en dos ordenadores diferentes

Aquí podemos observar unas estadísticas generales del resultado de haber ejecutado los tests:

Columna1				
Media	15,97023561	Interval (ms)	14,3766971	17,5637741
Error típico	0,811512005	Interval (s)	0,0143767	0,01756377
Mediana	9,3327			
Moda	20,8644			
Desviación estándar	20,57787076			
Varianza de la muestra	423,4487648			
Curtosis	17,9825545			
Coeficiente de asimetría	3,466474635			
Rango	205,571			
Mínimo	1,6773			
Máximo	207,2483			
Suma	10268,8615			
Cuenta	643			
Nivel de confianza(95,0%)	1,593538507			

A priori aún no nos dicen mucho estos números, así que analicemos los datos un poco mejor:

request-meth	request-path	response-status	time
	Promedio /		5,00449565
	Promedio /anonymous/system/sign-in		4,70408621
	Promedio /any/system/welcome		2,91354554
	Promedio /auditor/audit-record/create		62,2466963
	Promedio /auditor/audit-record/delete		24,9715571
	Promedio /auditor/audit-record/list		12,3779243
	Promedio /auditor/audit-record/publish		49,6833773
	Promedio /auditor/audit-record/show		14,3714277
	Promedio /auditor/audit-record/update		49,5426941
	Promedio /auditor/code-audit/create		27,857976
	Promedio /auditor/code-audit/delete		20,9536571
	Promedio /auditor/code-audit/list-mine		9,95762055
	Promedio /auditor/code-audit/publish		26,7272298
	Promedio /auditor/code-audit/show		11,9373598
	Promedio /auditor/code-audit/update		28,3937038
	Promedio /authenticated/system/sign-out		3,5404814
	Promedio general		15,4735974

Adjuntamos además un gráfico con los tiempos:



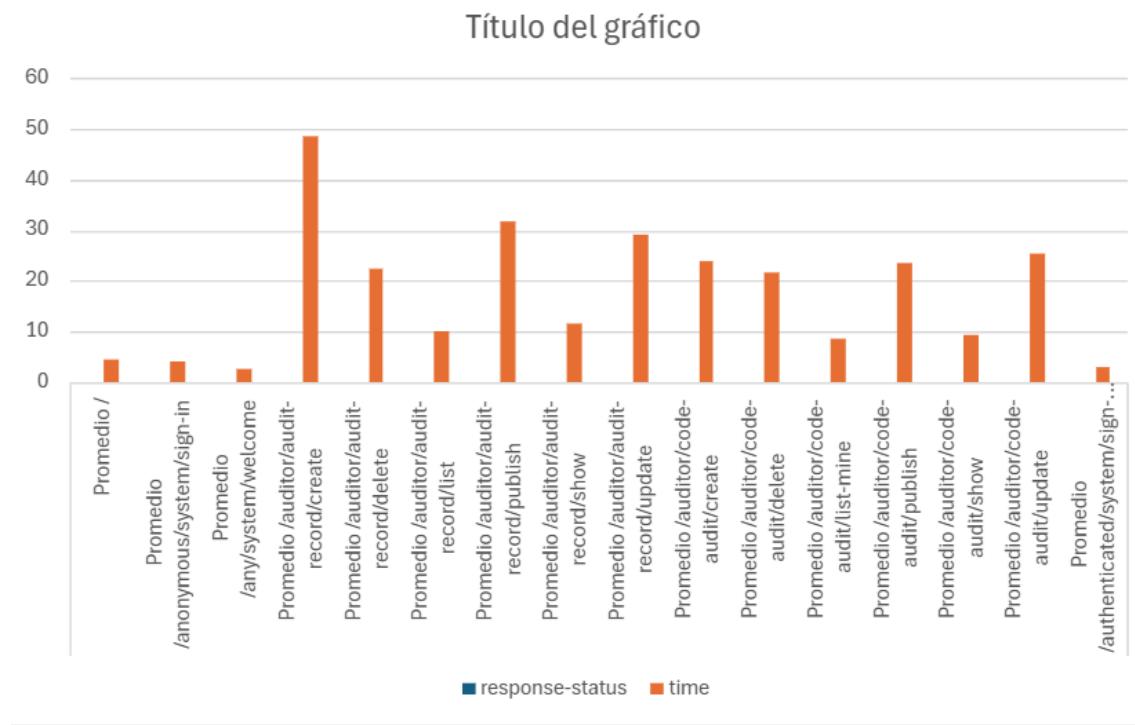
Ahora sí, podemos comentar los resultados de las estadísticas. Comenzamos diciendo que hemos superado el objetivo de tener un promedio inferior a 1 segundo, ya que obtuvimos una media de 15 milisegundos.

Por otra parte podemos ver que las funcionalidades que más tiempo requieren son la creación del auditRecord (por todas las comprobaciones que hay que hacer y el hecho de tener que añadir una nueva fila en base de datos, lo cuál tiene efectos en los índices que se generan para la búsqueda óptima). Los otros 2 más costosos son la edición y publicación del AuditRecord, los cuales tienen muchas validaciones por eso suponemos que tardan más. El resto de features a nuestro parecer dan buenos tiempos, muy ayudados de los índices.

No obstante, también hemos analizado los tiempos en el PC de otro miembro del grupo, en nuestro caso de Lucas Manuel Herencia Solís. Veamos los resultados:

request-method	request-path	response-status	time
	Promedio /		4,43509385
	Promedio /anonymous/system/sign-in		3,98486049
	Promedio /any/system/welcome		2,53686633
	Promedio /auditor/audit-record/create		48,6829
	Promedio /auditor/audit-record/delete		22,4051429
	Promedio /auditor/audit-record/list		10,1716255
	Promedio /auditor/audit-record/publish		31,9391667
	Promedio /auditor/audit-record/show		11,7099625
	Promedio /auditor/audit-record/update		29,0855824
	Promedio /auditor/code-audit/create		23,857956
	Promedio /auditor/code-audit/delete		21,5980875
	Promedio /auditor/code-audit/list-mine		8,56349524
	Promedio /auditor/code-audit/publish		23,6037222
	Promedio /auditor/code-audit/show		9,40472407
	Promedio /auditor/code-audit/update		25,3232115
	Promedio /authenticated/system/sign-out		3,12256364
	Promedio general		11,1136681

Este es el gráfico:



Sinceramente los números son muy parecidos a los obtenidos anteriormente, aunque se pueden encontrar ligeras diferencias. En este caso la media obtenida es de 11 milisegundos.

<i>Lucas</i>					
Media	11,11366812	Interval (ms)	10,1912701	12,0360662	
Error típico	0,469732664	Interval (s)	0,01019127	0,01203607	
Mediana	6,6561				
Moda	2,2467				
Desviación estándar	11,91122003				
Varianza de la muestra	141,8771626				
Curtosis	11,24438999				
Coeficiente de asimetría	2,561686773				
Rango	105,306				
Mínimo	1,4826				
Máximo	106,7886				
Suma	7146,0886				
Cuenta	643				
Nivel de confianza(95,0%)	0,922398046				

A pesar de estos parecidos, no podemos determinar que los datos obtenidos sean comparables. Sigamos a delante:

Prueba z para medias de dos muestras		
	<i>Dani</i>	<i>Lucas</i>
Media	15,4653378	11,1136681
Varianza (conocida)	423,448765	141,877163
Observaciones	798	643
Diferencia hipotética de las medias	0	
z	5,02057183	
P(Z<=z) una cola	2,5759E-07	
Valor crítico de z (una cola)	1,64485363	
Valor crítico de z (dos colas)	5,15E-07	
Valor crítico de z (dos colas)	1,95996398	

Estos son los resultados de hacer la prueba Z para ver si las dos medias son comparables entre sí.

Ahora, una vez hecho esto, se puede afirmar que las medias sí son comparables, ya que el valor P (P value) es menor que 0.05

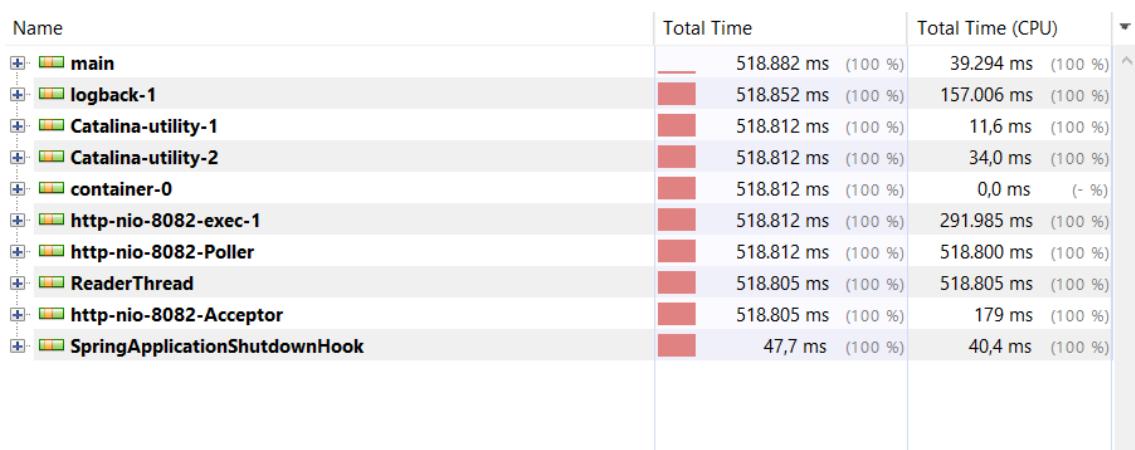
<i>Dani</i>		<i>Lucas</i>	
Media	15,9702356	Media	11,1136681
Error típico	0,811512	Error típico	0,46973266
Mediana	9,3327	Mediana	6,6561
Moda	20,8644	Moda	2,2467
Desviación estándar	20,5778708	Desviación estándar	11,91122
Varianza de la muestra	423,448765	Varianza de la muestra	141,877163
Curtosis	17,9825545	Curtosis	11,24439
Coeficiente de asimetría	3,46647463	Coeficiente de asimetría	2,56168677
Rango	205,571	Rango	105,306
Mínimo	1,6773	Mínimo	1,4826
Máximo	207,2483	Máximo	106,7886
Suma	10268,8615	Suma	7146,0886
Cuenta	643	Cuenta	643
Nivel de confianza(95,0%)	1,59353851	Nivel de confianza(95,0%)	0,92239805
Interval (ms)	14,3766971	Interval (ms)	10,1912701
Interval (s)	0,0143767	Interval (s)	0,01019127
	17,5637741		12,0360662
	0,01756377		0,01203607

Por tanto, podemos deducir que el ordenador de Lucas es ligeramente mejor que el Daniel, ya que Lucas obtiene una mejor media ($11.11 < 15.9$)

Aún así, ambos ordenadores verificaron que los métodos tardan menos de un segundo en ejecutarse.

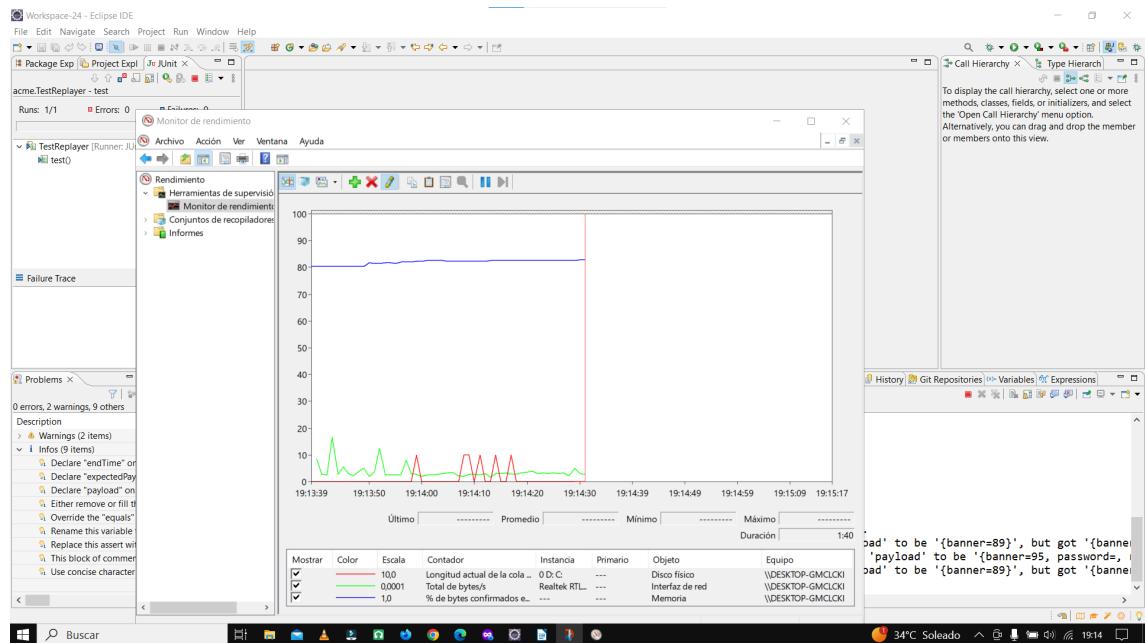
En cuanto a las pruebas con el VisualVM, seguimos los pasos de las diapositivas y usamos los parámetros de muestreo de las mismas (10 y 500 ms).

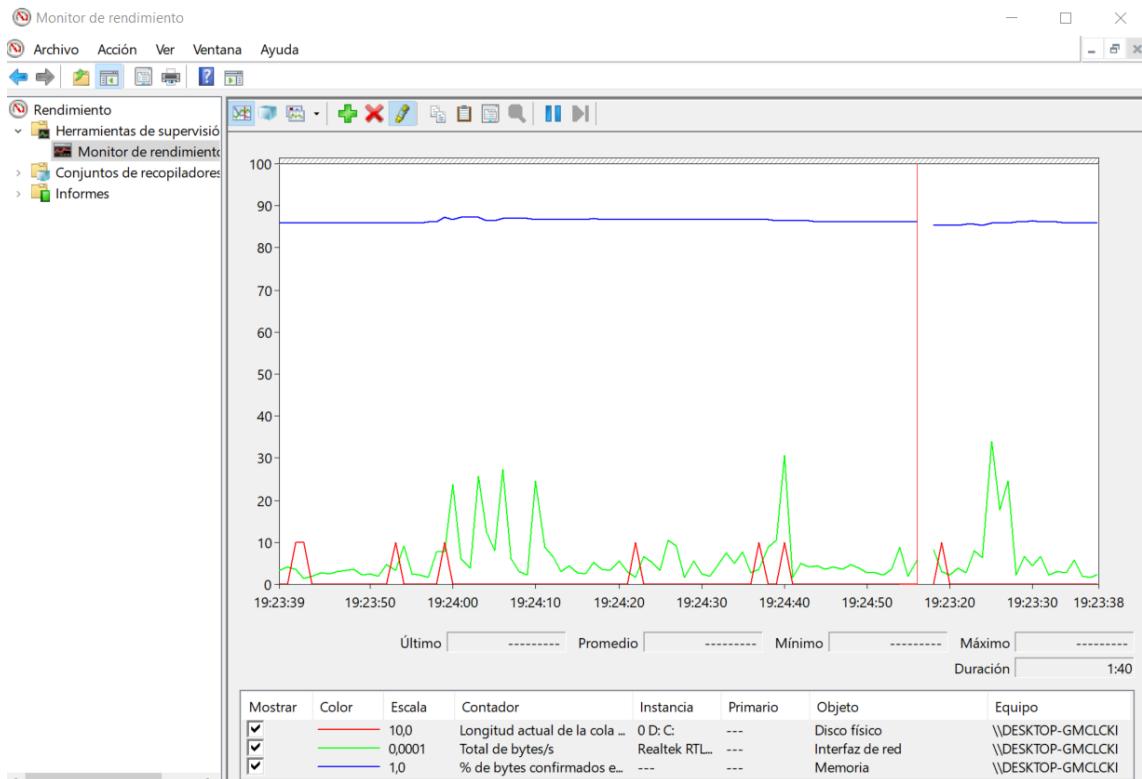
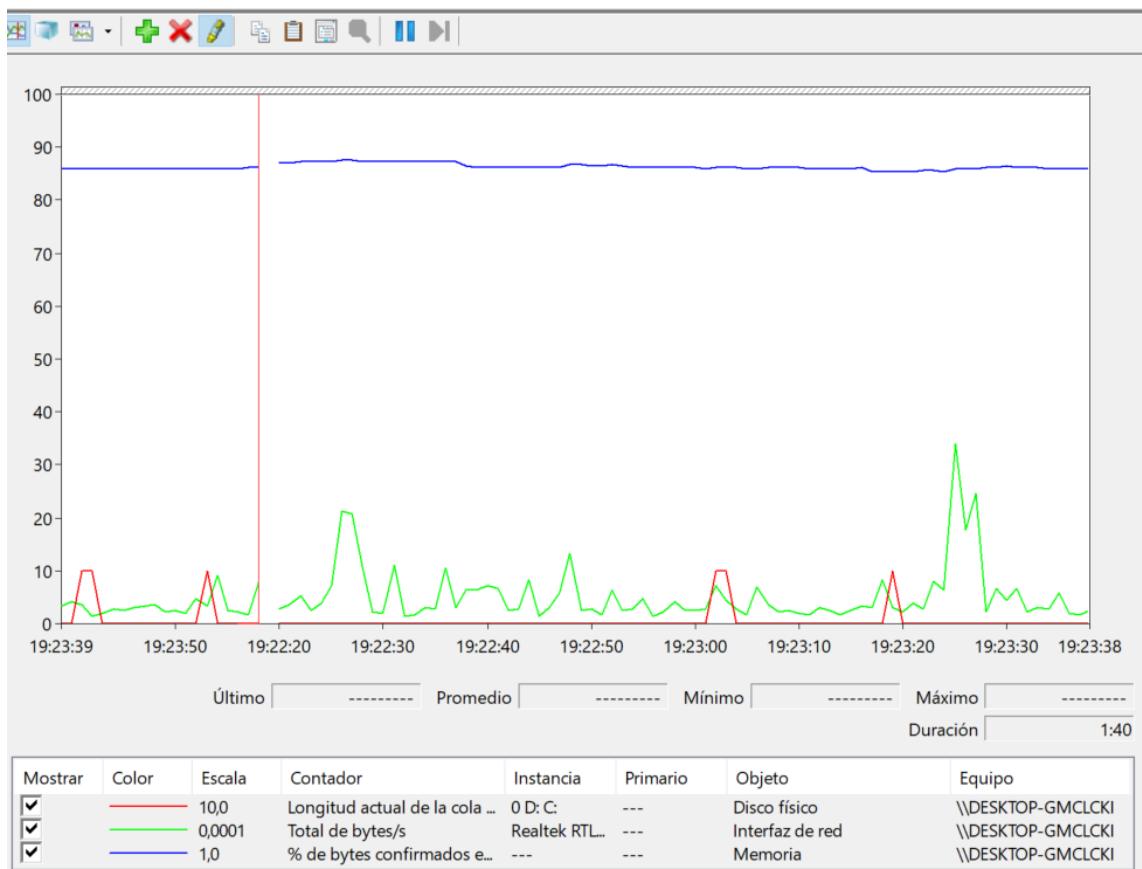
Aquí el resultado de haber corrido todos los tests visualizando con visualVM



Se adjuntarán los archivos exportados en la carpeta student5 de la entrega 4 (D4) dentro de la carpeta docs del proyecto.

Por último utilizamos el monitor de rendimiento, con el que obtuvimos los siguientes resultados:





Observamos unos picos al inicio y al final pero por lo demás no se observó nada fuera de lo común.

4. Bibliografía

No hay bibliografía presente para esta entrega.