Testing report

Diseño y pruebas II

Sprint 4

Versión 1.0

Fecha 27/05/2024

Preparado por:
Fernando José de Celis Hurtado

Índice

1.) Introducción **3**
2.) Contenido **3**
2.1) Functional testing 3
2.1.1) CodeAudit 3
2.1.1.1) List 3
2.1.1.2) Show 5
2.1.1.3) Create 7
2.1.1.4) Update 10
2.1.1.5) Publish 14
2.1.1.6) Delete 19
2.1.2) AuditRecord 21
2.1.2.1) List 21
2.1.2.2) Show 23
2.1.2.3) Create 25
2.1.2.4) Update 29
2.1.2.5) Publish 32
2.1.2.6) Delete 36
2.2) Performance testing 38
3.) Conclusiones **38**

1.) Introducción

En este documento se va a hablar sobre los casos de prueba implementados y el rendimiento de los mismos del student 5. Se va a dividir en dos puntos principales:

- Functional testing: En este apartado se va a comentar cómo se han hecho las pruebas y como se ven reflejadas en el código gracias al coverage. Aparecen lineas en amarillo en todas las clases "`assert object != null;`", esta estructura es heredada de los proyectos de ejemplo y supone una situación que no es posible controlar por nosotros por que pertenece al funcionamiento del framework. Es por ello que no se tendrán en cuenta en el análisis del código.

- Performance testing: Y en el segundo apartado se incluye información sobre el rendimiento de la aplicación. Gráficas e intervalos de confianza 95% tomados en dos ordenadores distintos, además de un contraste de hipótesis de confianza respecto a qué ordenador es más potente.

2.) Contenido

2.1) Functional testing

2.1.1) CodeAudit

2.1.1.1) List

list.safe = Para hacer esta prueba, me he metido en las listas de code audits de todos los usuarios de mi rol creados (auditor1, 2 y 3).

list.hack = Para hacer esta prueba he iniciado la aplicación y sin registrarme en ningún usuario, he intentado acceder a la siguiente url: http://localhost:8082/acme-sf-d04/auditor/code-audit/list dandome un error 500 de "access is not authorised".

Debido a que se había realizado un análisis en profundidad de los distintos requisitos realizados antes de comenzar a realizar tests, no se han encontrado bugs durante la realización de estos

Clase AuditorCodeAuditListService:

```
@Service
public class AuditorCodeAuditListService extends AbstractService<Auditor,
CodeAudit> {
```

```
    // Internal state
-------------------------------------------------------------

    @Autowired
    private AuditorCodeAuditRepository repository;

    // AbstractService interface
-----------------------------------------------


    @Override
    public void authorise() {
        super.getResponse().setAuthorised(true);
    }

    @Override
    public void load() {
        Collection<CodeAudit> objects;
        int auditorId;

        auditorId = super.getRequest().getPrincipal().getActiveRoleId();

        objects = this.repository.findCodeAuditsByAuditorId(auditorId);

        super.getBuffer().addData(objects);
    }

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

        Dataset dataset;
        List<Mark> marks;

        marks =
this.repository.findMarksByCodeAuditId(object.getId()).stream().toList();

        dataset = super.unbind(object, "code", "execution", "type",
"proposedCorrectiveActions", "link");
        dataset.put("mark", this.repository.averageMark(marks));
        dataset.put("project", object.getProject().getCode());

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

Línea amarilla	Explicación de por qué ocurre
Intencionalmente en blanco	Intencionalmente en blanco

2.1.1.2) Show

show.safe = Para hacer esta prueba, me he metido en las listas de code audits de los usuarios de mi rol creados (auditor1 y 2) y me he metido en todos y cada uno de los code audits.

show.hack = Para hacer esta prueba he iniciado la aplicación y sin registrarme en ningún usuario, he intentado acceder a la siguiente url que pertenece a un code audit del auditor1: http://localhost:8082/acme-sf-d04/auditor/code-audit/show?id=70 dandome un error 500 de “access is not authorised”. Y también me he registrado como auditor2 intentando acceder a ese code audit con la misma url dándome el mismo error.

Debido a que se había realizado un análisis en profundidad de los distintos requisitos realizados antes de comenzar a realizar tests, no se han encontrado bugs durante la realización de estos

Clase AuditorAuditRecordShowService:

```
@Service
public class AuditorCodeAuditShowService extends AbstractService<Auditor,
CodeAudit> {

    // Internal state
----------------------------------------------------------

    @Autowired
    private AuditorCodeAuditRepository repository;

    // AbstractService interface
-----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int id;
        int auditorId;
        CodeAudit codeAudit;

        id = super.getRequest().getData("id", int.class);
        codeAudit = this.repository.findOneCodeAuditById(id);

        auditorId = super.getRequest().getPrincipal().getActiveRoleId();

        status = auditorId == codeAudit.getAuditor().getId();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
```

```
        CodeAudit object;
        int id;

        id = super.getRequest().getData("id", int.class);
        object = this.repository.findOneCodeAuditById(id);

        super.getBuffer().addData(object);
    }

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

        Dataset dataset;
        SelectChoices choices;
        Collection<Project> projects =
this.repository.findPublishedProjects();
        SelectChoices choices2;
        List<Mark> marks;
        String projectCode;

        projectCode = object.getProject() != null ?
object.getProject().getCode() : null;

        Project project = object.getProject() != null ? object.getProject() :
(Project) projects.toArray()[0];

        marks =
this.repository.findMarksByCodeAuditId(object.getId()).stream().toList();

        choices = SelectChoices.from(CodeAuditType.class, object.getType());
        choices2 = SelectChoices.from(projects, "code", project);

        dataset = super.unbind(object, "code", "execution", "type",
"proposedCorrectiveActions", "link", "project", "draftMode");
        dataset.put("mark", this.repository.averageMark(marks));
        dataset.put("types", choices);
        dataset.put("projects", choices2);
        dataset.put("project", projectCode);

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

}
```

Línea amarilla	Explicación de por qué ocurre

`projectCode = object.getProject() != null ? object.getProject().getCode() : null;`
 	
`Project project = object.getProject() != null ? object.getProject() : (Project) projects.toArray()[0];`
	Estas dos lineas están en amarillo puesto que para poder hacerse el show de un code audit, este debe estar al menos creado. Y para que un code audit esté creado, obligatoriamente debe estar asociado a un proyecto, por tanto nunca va a dar nulo. Está creado para poder evitar el posible hacking.

2.1.1.3) Create

create.safe = Para hacer esta prueba, he probado a crear un code audit con todos los posibles valores negativos empezando por todos los valores a nulos y a continuación yendo atributo por atributo con sus respectivos casos. Tras probar los escenarios negativos creé diversas entidades code audits con los casos positivos yendo atributo por atributo, estos casos positivos son el extremo inferior, el extremo superior, un valor intermedio y valores de diferentes tipos de hacking respetando los rangos de cada atributo.

create.hack = Para hacer esta prueba he iniciado la aplicación y sin registrarme en ningún usuario, he intentado acceder a la siguiente url: http://localhost:8082/acme-sf-d04/auditor/code-audit/create dandome un error 500 de “access is not authorised”.

Debido a que se había realizado un análisis en profundidad de los distintos requisitos realizados antes de comenzar a realizar tests, no se han encontrado bugs durante la realización de estos

Clase AuditorCodeAuditCreateService:

```
@Service
public class AuditorCodeAuditCreateService extends AbstractService<Auditor,
CodeAudit> {

    // Internal state
----------------------------------------------------------

    @Autowired
    private AuditorCodeAuditRepository repository;

    // AbstractService interface
-----------------------------------------------



    @Override
    public void authorise() {
        super.getResponse().setAuthorised(true);
    }
```

```
    @Override
    public void load() {
        CodeAudit object;

        object = new CodeAudit();
        Integer auditorId =
super.getRequest().getPrincipal().getActiveRoleId();
        Auditor auditor = this.repository.findOneAuditorById(auditorId);
        object.setAuditor(auditor);
        object.setDraftMode(true);

        super.getBuffer().addData(object);
    }

    @Override
    public void bind(final CodeAudit object) {
        assert object != null;

        super.bind(object, "code", "execution", "type",
"proposedCorrectiveActions", "link", "project");
    }

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

        if (!super.getBuffer().getErrors().hasErrors("code")) {

            CodeAudit codeAuditSameCode =
this.repository.findOneCodeAuditByCode(object.getCode());

            super.state(codeAuditSameCode == null, "code",
"auditor.code-audit.form.error.code");
        }

        if (!super.getBuffer().getErrors().hasErrors("project"))
            super.state(!object.getProject().isDraftMode(), "project",
"auditor.code-audit.form.error.project");

        if (!super.getBuffer().getErrors().hasErrors("execution")) {

            Date codeAuditDate = object.getExecution();
            Date minimumDate = MomentHelper.parse("1999-12-31 23:59",
"yyyy-MM-dd HH:mm");

            Boolean isAfter = codeAuditDate.after(minimumDate);
            super.state(isAfter, "execution",
"auditor.code-audit.form.error.execution");
        }

    }
```

```
    @Override
    public void perform(final CodeAudit object) {
        assert object != null;
        object.setId(0);
        this.repository.save(object);
    }

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

        Dataset dataset;
        SelectChoices choices;
        Collection<Project> projects =
this.repository.findPublishedProjects();
        SelectChoices choices2;
        List<Mark> marks;
        String projectCode;

        projectCode = object.getProject() != null ?
object.getProject().getCode() : null;

        Project project = projects.stream().toList().isEmpty() ? null :
projects.stream().toList().get(0);

        marks =
this.repository.findMarksByCodeAuditId(object.getId()).stream().toList();

        choices = SelectChoices.from(CodeAuditType.class, object.getType());
        choices2 = SelectChoices.from(projects, "code", project);

        dataset = super.unbind(object, "code", "execution", "type",
"proposedCorrectiveActions", "link", "project", "draftMode");
        dataset.put("mark", this.repository.averageMark(marks));
        dataset.put("types", choices);
        dataset.put("projects", choices2);
        dataset.put("project", projectCode);

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

}
```

Línea amarilla	Explicación de por qué ocurre

`super.state(!object.getProject().isDraftMode(), "project", "auditor.code-audit.form.error.project");`	Este código supone una validación realizada con el fin de evitar un posible hackeo en el campo de proyecto, el usuario común nunca va a encontrar posibilidad de llegar a este estado (seleccionar un project que no esté publicado) usando el sistema de forma correcta.
`Project project = projects.stream().toList().isEmpty() ? null : projects.stream().toList().get(0);`	Este código está pensado para el caso en el que se intente crear una code audit sin existir proyectos. Dado el procedimiento de testing, este caso nunca se podrá probar de forma legal.

2.1.1.4) Update

update.safe = Para hacer esta prueba, he actualizado un code audit probando todos los posibles valores negativos empezando por todos los valores a nulos y a continuación yendo atributo por atributo con sus respectivos casos. Tras probar los escenarios negativos actualicé el mismo code audit con los casos positivos yendo atributo por atributo, estos casos positivos son el extremo inferior, el extremo superior, un valor intermedio y valores de diferentes tipos de hacking respetando los rangos de cada atributo.

update.hack = Para hacer esta prueba he iniciado la aplicación y me he registrado con el usuario auditor1, tras iniciada la sesión me he metido en un code audit con id:70 y dándole a inspeccionar en la pestaña, he cambiado ese valor de id por 123 (code audit perteneciente al auditor2) y le di al botón de update, tras ello me volví a meter en la misma code audit y cambié la id a 71 (code audit publicada de auditor1) y volviendo a pulsar en el botón de update. En ambos procesos me salió un error 500 de “access is not authorised”.

Gracias al intentar este tipo de test, me percaté de un error en el código relacionado con las validaciones entre las entidades padre (code audit) y las hijas (audit records), ya que las fechas de las entidades hijas deben ser siempre superior a la de la entidad padre.

Clase AuditorCodeAuditUpdateService:

```
@Service
public class AuditorCodeAuditUpdateService extends AbstractService<Auditor,
CodeAudit> {

    // Internal state
-----------------------------------------------------------

    @Autowired
    private AuditorCodeAuditRepository repository;

    // AbstractService interface
-----------------------------------------------
```

```
    @Override
    public void authorise() {
        boolean status;
        int id;
        int auditorId;
        CodeAudit codeAudit;

        id = super.getRequest().getData("id", int.class);
        codeAudit = this.repository.findOneCodeAuditById(id);

        auditorId = super.getRequest().getPrincipal().getActiveRoleId();

        status = auditorId == codeAudit.getAuditor().getId() &&
codeAudit.isDraftMode();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        CodeAudit object;
        Integer id;

        id = super.getRequest().getData("id", int.class);
        object = this.repository.findOneCodeAuditById(id);

        super.getBuffer().addData(object);
    }

    @Override
    public void bind(final CodeAudit object) {
        assert object != null;

        Integer auditorId =
super.getRequest().getPrincipal().getActiveRoleId();
        Auditor auditor = this.repository.findOneAuditorById(auditorId);
        object.setAuditor(auditor);
        super.bind(object, "code", "execution", "type",
"proposedCorrectiveActions", "link", "project");
    }

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

        if (!super.getBuffer().getErrors().hasErrors("code")) {

            CodeAudit codeAuditSameCode =
this.repository.findOneCodeAuditByCode(object.getCode());

            if (codeAuditSameCode != null)
```

```
                super.state(codeAuditSameCode.getId() == object.getId(),
"code", "auditor.code-audit.form.error.code");
        }

        if (!super.getBuffer().getErrors().hasErrors("project"))
            super.state(!object.getProject().isDraftMode(), "project",
"auditor.code-audit.form.error.project");

        if (!super.getBuffer().getErrors().hasErrors("execution")) {

            Date codeAuditDate = object.getExecution();
            Date minimumDate = MomentHelper.parse("1999-12-31 23:59",
"yyyy-MM-dd HH:mm");

            Boolean isAfter = codeAuditDate.after(minimumDate);
            super.state(isAfter, "execution",
"auditor.code-audit.form.error.execution");
        }

        if (!super.getBuffer().getErrors().hasErrors("execution")) {
            AuditRecord earliestAuditRecord;
            Boolean validExecution;
            Date execution = object.getExecution();

            earliestAuditRecord =
this.repository.findAuditRecordWithEarliestDateByCodeAuditId(object.getId()).
stream().findFirst().orElse(null);

            if (earliestAuditRecord != null) {
                validExecution =
execution.before(earliestAuditRecord.getAuditStartTime());
                super.state(validExecution, "execution",
"auditor.code-audit.form.error.execution-ar");
            }
        }
    }

    @Override
    public void perform(final CodeAudit object) {
        assert object != null;

        this.repository.save(object);
    }

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

        Dataset dataset;
        SelectChoices choices;
        Collection<Project> projects =
this.repository.findPublishedProjects();
        SelectChoices choices2;
```

```
        List<Mark> marks;
        String projectCode;

        projectCode = object.getProject() != null ?
object.getProject().getCode() : null;

        Project project = object.getProject() != null ? object.getProject() :
(Project) projects.toArray()[0];

        marks =
this.repository.findMarksByCodeAuditId(object.getId()).stream().toList();

        choices = SelectChoices.from(CodeAuditType.class, object.getType());
        choices2 = SelectChoices.from(projects, "code", project);

        dataset = super.unbind(object, "code", "execution", "type",
"proposedCorrectiveActions", "link", "project", "draftMode");
        dataset.put("mark", this.repository.averageMark(marks));
        dataset.put("types", choices);
        dataset.put("projects", choices2);
        dataset.put("project", projectCode);

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

}
```

Línea amarilla	Explicación de por qué ocurre
`if (codeAuditSameCode != null)`	Este código no debería salir en amarillo ya que se han probado todos los casos posibles (que tenga el mismo código y que no). Se desconoce el motivo de que salga amarillo, pero haciendo el tester#analyser, se podrá comprobar esta validación correctamente probada
`super.state(!object.getProject().isDraftMode(), "project", "auditor.code-audit.form.error.project");`	Este código supone una validación realizada con el fin de evitar un posible hackeo en el campo de proyecto, el usuario común nunca va a encontrar posibilidad de llegar a este estado (seleccionar un project que no esté publicado) usando el sistema de forma correcta.

La línea amarilla de “`project`” se comprobó en la siguiente petición:

** POST auditor/code-audit/update (RESPONSE)

banner	code	code$error
181		Must match pattern "[A-Z]{1,3}-[0-9]{3}". Must not be blank.
181		Must match pattern "[A-Z]{1,3}-[0-9]{3}". Must not be blank.
181	Aa-000	Must match pattern "[A-Z]{1,3}-[0-9]{3}".
171	A-008	This code is already in use.
181	A-002	

2.1.1.5) Publish

publish.safe = Para hacer esta prueba, he publicado diversos code audits probando todos los posibles valores negativos empezando por todos los valores a nulos y a continuación yendo atributo por atributo con sus respectivos casos. Tras probar los escenarios negativos publiqué los code audits con los casos positivos yendo atributo por atributo, estos casos positivos son el extremo inferior, el extremo superior, un valor intermedio y valores de diferentes tipos de hacking respetando los rangos de cada atributo.

publish.hack = Para hacer esta prueba he iniciado la aplicación y me he registrado con el usuario auditor1, tras iniciada la sesión me he metido en un code audit con id:70 y dándole a inspeccionar en la pestaña, he cambiado ese valor de id por 123 (code audit perteneciente al auditor2) y le di al botón de publish, tras ello me volví a meter en la misma code audit y cambié la id a 71 (code audit publicada de auditor1) y volviendo a pulsar en el botón de publish. En ambos procesos me salió un error 500 de “access is not authorised”.

Gracias al intentar este tipo de test, me percaté de un error en el código relacionado con las validaciones entre las entidades padre (code audit) y las hijas (audit records), ya que las fechas de las entidades hijas deben ser siempre superior a la de la entidad padre.

También se probaron las validaciones específicas que tiene el publish de esta entidad.

Clase AuditorCodeAuditPublishService:

```
@Service
public class AuditorCodeAuditPublishService extends AbstractService<Auditor,
CodeAudit> {

    // Internal state
----------------------------------------------------------

    @Autowired
    private AuditorCodeAuditRepository repository;

    // AbstractService interface
-------------------------------------------


    @Override
```

```
    public void authorise() {
        boolean status;
        int id;
        int auditorId;
        CodeAudit codeAudit;

        id = super.getRequest().getData("id", int.class);
        codeAudit = this.repository.findOneCodeAuditById(id);

        auditorId = super.getRequest().getPrincipal().getActiveRoleId();

        status = auditorId == codeAudit.getAuditor().getId() &&
codeAudit.isDraftMode();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        CodeAudit object;
        int id;

        id = super.getRequest().getData("id", int.class);
        object = this.repository.findOneCodeAuditById(id);

        super.getBuffer().addData(object);
    }

    @Override
    public void bind(final CodeAudit object) {
        assert object != null;

        super.bind(object, "code", "execution", "type",
"proposedCorrectiveActions", "link", "project");
    }

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

        if (!super.getBuffer().getErrors().hasErrors("project"))
            super.state(!object.getProject().isDraftMode(), "project",
"auditor.code-audit.form.error.project");

        if (!super.getBuffer().getErrors().hasErrors("code")) {

            CodeAudit codeAuditSameCode =
this.repository.findOneCodeAuditByCode(object.getCode());

            if (codeAuditSameCode != null)
                super.state(codeAuditSameCode.getId() == object.getId(),
"code", "auditor.code-audit.form.error.code");
        }
```

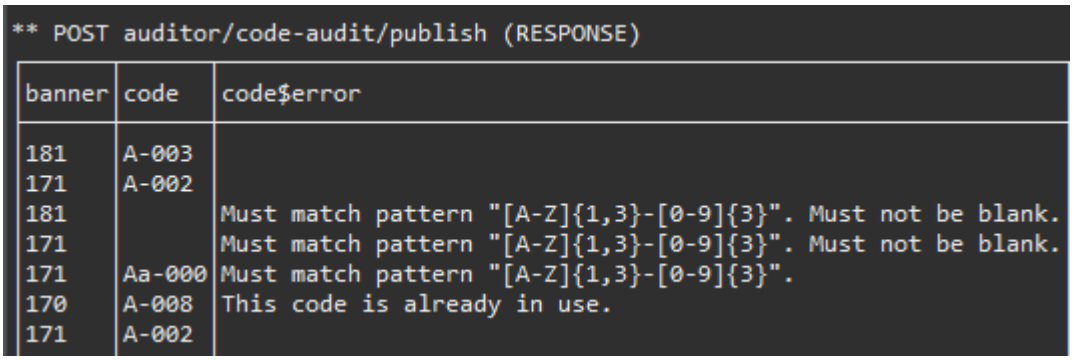
```
        if (!super.getBuffer().getErrors().hasErrors("mark")) {
            List<Mark> marks;

            marks =
this.repository.findMarksByCodeAuditId(object.getId()).stream().toList();
            Mark nota = this.repository.averageMark(marks);

            super.state(nota == Mark.C || nota == Mark.B || nota == Mark.A ||
nota == Mark.AA, "mark", "auditor.code-audit.form.error.mark");
        }

        if
(!super.getBuffer().getErrors().hasErrors("unpublishedAuditRecords")) {

            Collection<AuditRecord> unpublishedAuditRecords;

            unpublishedAuditRecords =
this.repository.findUnpublishedAuditRecordsByCodeAuditId(object.getId());

            super.state(unpublishedAuditRecords.isEmpty(), "*",
"auditor.code-audit.form.error.unpublished-audit-records");
        }

        if (!super.getBuffer().getErrors().hasErrors("execution")) {

            Date codeAuditDate = object.getExecution();
            Date minimumDate = MomentHelper.parse("1999-12-31 23:59",
"yyyy-MM-dd HH:mm");

            Boolean isAfter = codeAuditDate.after(minimumDate);
            super.state(isAfter, "execution",
"auditor.code-audit.form.error.execution");
        }

        if (!super.getBuffer().getErrors().hasErrors("execution")) {
            AuditRecord earliestAuditRecord;
            Boolean validExecution;
            Date execution = object.getExecution();

            earliestAuditRecord =
this.repository.findAuditRecordWithEarliestDateByCodeAuditId(object.getId()).
stream().findFirst().orElse(null);

            if (earliestAuditRecord != null) {
                validExecution =
execution.before(earliestAuditRecord.getAuditStartTime());
                super.state(validExecution, "execution",
"auditor.code-audit.form.error.execution-ar");
            }
        }
    }
```

```
    @Override
    public void perform(final CodeAudit object) {
        assert object != null;

        object.setDraftMode(false);
        this.repository.save(object);
    }

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

        Dataset dataset;
        SelectChoices choices;
        Collection<Project> projects =
this.repository.findPublishedProjects();
        SelectChoices choices2;
        List<Mark> marks;
        String projectCode;

        projectCode = object.getProject() != null ?
object.getProject().getCode() : null;

        Project project = object.getProject() != null ? object.getProject() :
(Project) projects.toArray()[0];

        marks =
this.repository.findMarksByCodeAuditId(object.getId()).stream().toList();

        choices = SelectChoices.from(CodeAuditType.class, object.getType());
        choices2 = SelectChoices.from(projects, "code", project);

        dataset = super.unbind(object, "code", "execution", "type",
"proposedCorrectiveActions", "link", "project", "draftMode");
        dataset.put("mark", this.repository.averageMark(marks));
        dataset.put("types", choices);
        dataset.put("projects", choices2);
        dataset.put("project", projectCode);

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

Línea amarilla	Explicación de por qué ocurre
`if (codeAuditSameCode != null)`	Este código no debería salir en amarillo ya que se han probado todos los casos posibles (que tenga el mismo código y que no). Se desconoce el motivo de que salga amarillo, pero haciendo el tester#analyser, se podrá comprobar esta validación correctamente probada

`super.state(!object.getProject().isDraftMode(), "project", "auditor.code-audit.form.error.project");`	Este código supone una validación realizada con el fin de evitar un posible hackeo en el campo de proyecto, el usuario común nunca va a encontrar posibilidad de llegar a este estado (seleccionar un project que no esté publicado) usando el sistema de forma correcta.
`if (!super.getBuffer().getErrors().hasErrors("mark")) {`	Este código no debería salir en amarillo ya que se han probado todos los casos posibles (que tenga menor nota que C y que no). Se desconoce el motivo de que salga amarillo, pero haciendo el tester#analyser, se podrá comprobar esta validación correctamente probada
`if (!super.getBuffer().getErrors().hasErrors("unpublishedAuditRecords")) {`	Este código no debería salir en amarillo ya que se han probado todos los casos posibles (que tenga audit records sin publicar y publicados). Se desconoce el motivo de que salga amarillo, pero haciendo el tester#analyser, se podrá comprobar esta validación correctamente probada

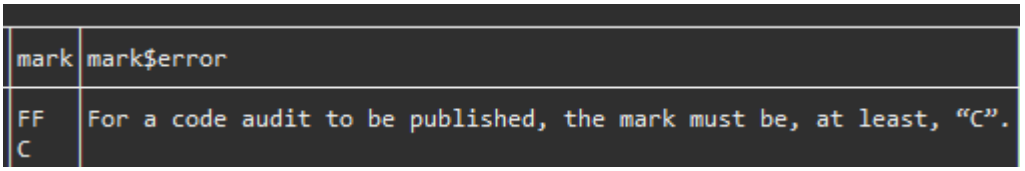
La línea amarilla de "`project`" se comprobó en la siguiente petición:



** POST auditor/code-audit/publish (RESPONSE)

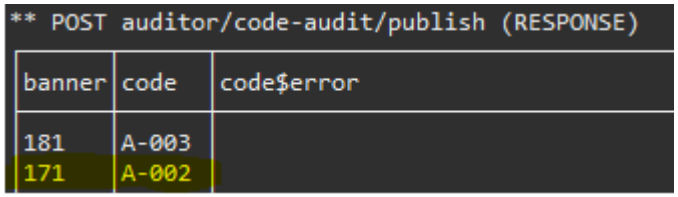
banner	code	code$error
181	A-003	
171	A-002	
181		Must match pattern "[A-Z]{1,3}-[0-9]{3}". Must not be blank.
171		Must match pattern "[A-Z]{1,3}-[0-9]{3}". Must not be blank.
171	Aa-000	Must match pattern "[A-Z]{1,3}-[0-9]{3}".
170	A-008	This code is already in use.
171	A-002	

La línea amarilla de "`mark`" se comprobó en la siguiente petición:



mark	mark$error
FF C	For a code audit to be published, the mark must be, at least, "C".

La línea amarilla de "`unpublishedAuditRecords`" se comprobó en la siguiente petición:



** POST auditor/code-audit/publish (RESPONSE)

banner	code	code$error
181	A-003	
171	A-002	

2.1.1.6) Delete

delete.safe = Para hacer esta prueba,he intentado borrar una code audit con entidades secundarias publicadas (caso negativo) y borrar una sin entidades secundarias secundarias (caso positivo).

delete.hack = Para hacer esta prueba he iniciado la aplicación y me he registrado con el usuario auditor1, tras iniciada la sesión me he metido en un code audit con id:70 y dándole a inspeccionar en la pestaña, he cambiado ese valor de id por 123 (code audit perteneciente al auditor2) y le di al botón de delete, tras ello me volví a meter en la misma code audit y cambié la id a 71 (code audit publicada de auditor1) y volviendo a pulsar en el botón de delete. En ambos procesos me salió un error 500 de “access is not authorised”.

Debido a que se había realizado un análisis en profundidad de los distintos requisitos realizados antes de comenzar a realizar tests, no se han encontrado bugs durante la realización de estos

Clase AuditorCodeAuditDeleteService:

```
@Service
public class AuditorCodeAuditDeleteService extends AbstractService<Auditor,
CodeAudit> {

    // Internal state
----------------------------------------------------------

    @Autowired
    private AuditorCodeAuditRepository repository;

    // AbstractService interface
-----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int id;
        int auditorId;
        CodeAudit codeAudit;

        id = super.getRequest().getData("id", int.class);
        codeAudit = this.repository.findOneCodeAuditById(id);

        auditorId = super.getRequest().getPrincipal().getActiveRoleId();

        status = auditorId == codeAudit.getAuditor().getId() &&
codeAudit.isDraftMode();

        super.getResponse().setAuthorised(status);
    }
```

```
    @Override
    public void load() {
        CodeAudit object;
        int id;

        id = super.getRequest().getData("id", int.class);
        object = this.repository.findOneCodeAuditById(id);

        super.getBuffer().addData(object);
    }

    @Override
    public void bind(final CodeAudit object) {
        assert object != null;

        super.bind(object, "code", "execution", "type",
"proposedCorrectiveActions", "link", "project");
    }

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

        Collection<AuditRecord> publishedAuditRecord;

        publishedAuditRecord =
this.repository.findPublishedAuditRecordsByCodeAuditId(object.getId());
        super.state(publishedAuditRecord.isEmpty(), "*",
"auditor.code-audit.form.error.published-audit-records");

    }

    @Override
    public void perform(final CodeAudit object) {
        assert object != null;

        Collection<AuditRecord> relations =
this.repository.findRelationsByCodeAuditId(object.getId());

        this.repository.deleteAll(relations);

        this.repository.delete(object);
    }

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

        Dataset dataset;
        SelectChoices choices;
        Collection<Project> projects =
this.repository.findPublishedProjects();
        SelectChoices choices2;
```

```
        List<Mark> marks;
        String projectCode;

        projectCode = object.getProject() != null ?
object.getProject().getCode() : null;

        Project project = object.getProject() != null ? object.getProject() :
(Project) projects.toArray()[0];

        marks =
this.repository.findMarksByCodeAuditId(object.getId()).stream().toList();

        choices = SelectChoices.from(CodeAuditType.class, object.getType());
        choices2 = SelectChoices.from(projects, "code", project);

        dataset = super.unbind(object, "code", "execution", "type",
"proposedCorrectiveActions", "link", "project", "draftMode");
        dataset.put("mark", this.repository.averageMark(marks));
        dataset.put("types", choices);
        dataset.put("projects", choices2);
        dataset.put("project", projectCode);

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

Línea amarilla	Explicación de por qué ocurre

`projectCode = object.getProject() != null ? object.getProject().getCode() : null;`
 	
`Project project = object.getProject() != null ? object.getProject() : (Project) projects.toArray()[0];`
	Estas dos lineas están en amarillo puesto que para poder hacerse el show de un code audit, este debe estar al menos creado. Y para que un code audit esté creado, obligatoriamente debe estar asociado a un proyecto, por tanto nunca va a dar nulo. Está creado para poder evitar el posible hacking.

2.1.2) AuditRecord

2.1.2.1) List

list.safe = Para hacer esta prueba, me he metido en las listas de code audits de los usuarios de mi rol creados (auditor1 y 2) y he listado los audit records de cada code audit.

list.hack = Para hacer esta prueba he iniciado la aplicación y sin registrarme en ningún usuario, he intentado acceder a la siguiente url que pertenece a un code audit del auditor1: http://localhost:8082/acme-sf-d04/auditor/audit-record/list?codeAuditId=70 dandome un error 500 de “access is not authorised”. Y también me he registrado como auditor2 intentando acceder a ese code audit con la misma url dándome el mismo error.

Debido a que se había realizado un análisis en profundidad de los distintos requisitos realizados antes de comenzar a realizar tests, no se han encontrado bugs durante la realización de estos

Clase AuditorAuditRecordListService:

```
@Service
public class AuditorAuditRecordListService extends AbstractService<Auditor,
AuditRecord> {

    // Internal state
----------------------------------------------------------

    @Autowired
    private AuditorAuditRecordRepository    repository;

    @Autowired
    private AuditorCodeAuditRepository      codeAuditRepository;

    // AbstractService interface
-----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int codeAuditId;
        int auditorId;
        CodeAudit codeAudit;

        codeAuditId = super.getRequest().getData("codeAuditId", int.class);
        codeAudit =
this.codeAuditRepository.findOneCodeAuditById(codeAuditId);

        auditorId = super.getRequest().getPrincipal().getActiveRoleId();

        status = auditorId == codeAudit.getAuditor().getId();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        Collection<AuditRecord> objects;
        int codeAuditId;

        codeAuditId = super.getRequest().getData("codeAuditId", int.class);

        objects = this.repository.findAuditRecordsByCodeAuditId(codeAuditId);

        super.getBuffer().addData(objects);
    }
```

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

        Dataset dataset;

        dataset = super.unbind(object, "code", "auditStartTime",
"auditEndTime", "mark", "link");
        super.getResponse().addData(dataset);
    }

    @Override
    public void unbind(final Collection<AuditRecord> objects) {
        assert objects != null;

        int codeAuditId;
        CodeAudit codeAudit;

        codeAuditId = super.getRequest().getData("codeAuditId", int.class);
        codeAudit = this.repository.findOneCodeAuditById(codeAuditId);

        super.getResponse().addGlobal("codeAuditId", codeAuditId);
        super.getResponse().addGlobal("canCreate", codeAudit.isDraftMode());
    }
}
```

Línea amarilla	Explicación de por qué ocurre
Intencionalmente en blanco	Intencionalmente en blanco

2.1.2.2) Show

show.safe = Para hacer esta prueba, me he metido en las listas de code audits los usuarios de mi rol creados (auditor1 y 2), he listado los audit records de cada code audit y por último me he metido en todos y cada uno de los audit records existentes.

show.hack = Para hacer esta prueba he iniciado la aplicación y sin registrarme en ningún usuario, he intentado acceder a la siguiente url que pertenece a un audit record del auditor1: http://localhost:8082/acme-sf-d04/auditor/audit-record/show?id=124 dandome un error 500 de “access is not authorised”. Y también me he registrado como auditor2 intentando acceder a ese audit record con la misma url dándome el mismo error.

Debido a que se había realizado un análisis en profundidad de los distintos requisitos realizados antes de comenzar a realizar tests, no se han encontrado bugs durante la realización de estos

Clase AuditorAuditRecordShowService:

```java
@Service
public class AuditorAuditRecordShowService extends AbstractService<Auditor,
AuditRecord> {

    // Internal state
----------------------------------------------------------

    @Autowired
    private AuditorAuditRecordRepository repository;

    // AbstractService interface
-----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int id;
        int auditorId;
        AuditRecord auditRecord;

        id = super.getRequest().getData("id", int.class);
        auditRecord = this.repository.findOneAuditRecordById(id);

        auditorId = super.getRequest().getPrincipal().getActiveRoleId();

        status = auditorId == auditRecord.getAuditor().getId();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        AuditRecord object;
        int id;

        id = super.getRequest().getData("id", int.class);
        object = this.repository.findOneAuditRecordById(id);

        super.getBuffer().addData(object);
    }

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

        Dataset dataset;
        SelectChoices choices;

        choices = SelectChoices.from(Mark.class, object.getMark());

        dataset = super.unbind(object, "code", "auditStartTime",
"auditEndTime", "mark", "link", "draftMode");
```

```
        dataset.put("marks", choices);

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

}
```

Línea amarilla	Explicación de por qué ocurre
Intencionalmente en blanco	Intencionalmente en blanco

2.1.2.3) Create

create.safe = Para hacer esta prueba, he probado a crear un audit record con todos los posibles valores negativos empezando por todos los valores a nulos y a continuación yendo atributo por atributo con sus respectivos casos. Tras probar los escenarios negativos creé diversas entidades audit records con los casos positivos yendo atributo por atributo, estos casos positivos son el extremo inferior, el extremo superior, un valor intermedio y valores de diferentes tipos de hacking respetando los rangos de cada atributo.

create.hack = Para hacer esta prueba he iniciado la aplicación y sin registrarme en ningún usuario, he intentado acceder a la siguiente url que pertenece a un audit record de un code audit del auditor1: http://localhost:8082/acme-sf-d04/auditor/audit-record/create?codeAuditId=70 dandome un error 500 de “access is not authorised”. Y también me he registrado como auditor2 intentando acceder a ese audit record con la misma url dándome el mismo error.

Gracias al intentar este tipo de test, me percaté de un error en el código relacionado con las validaciones entre las entidades padre (code audit) y las hijas (audit records), ya que las fechas de las entidades hijas deben ser siempre superior a la de la entidad padre.

Clase AuditorAuditRecordCreateService:

```
@Service
public class AuditorAuditRecordCreateService extends
AbstractService<Auditor, AuditRecord> {

    // Internal state
-----------------------------------------------------------

    @Autowired
    private AuditorAuditRecordRepository repository;

    // AbstractService interface
-----------------------------------------------


    @Override
    public void authorise() {
```

```
        boolean status;
        int codeAuditId;
        int auditorId;
        CodeAudit codeAudit;

        codeAuditId = super.getRequest().getData("codeAuditId", int.class);
        codeAudit = this.repository.findOneCodeAuditById(codeAuditId);

        auditorId = super.getRequest().getPrincipal().getActiveRoleId();

        status = auditorId == codeAudit.getAuditor().getId();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        AuditRecord object;
        Integer codeAuditId;
        CodeAudit codeAudit;

        object = new AuditRecord();
        Integer auditorId =
super.getRequest().getPrincipal().getActiveRoleId();
        Auditor auditor = this.repository.findOneAuditorById(auditorId);

        codeAuditId = super.getRequest().getData("codeAuditId", int.class);
        codeAudit = this.repository.findOneCodeAuditById(codeAuditId);

        object.setAuditor(auditor);
        object.setCodeAudit(codeAudit);
        object.setDraftMode(true);

        super.getBuffer().addData(object);

    }

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

        super.bind(object, "code", "auditStartTime", "auditEndTime", "mark",
"link");
    }

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

        if (!super.getBuffer().getErrors().hasErrors("code")) {

            AuditRecord auditRecordSameCode =
this.repository.findOneAuditRecordByCode(object.getCode());
```

```
            if (auditRecordSameCode != null)
                super.state(auditRecordSameCode.getId() == object.getId(),
"code", "auditor.audit-record.form.error.code");
        }

        if (!super.getBuffer().getErrors().hasErrors("auditEndTime")) {
            Date auditStartTime;
            Date auditEndTime;

            auditStartTime = object.getAuditStartTime();
            auditEndTime = object.getAuditEndTime();

            if (auditStartTime != null && auditEndTime != null)
                super.state(MomentHelper.isLongEnough(auditStartTime,
auditEndTime, 1, ChronoUnit.HOURS) && auditEndTime.after(auditStartTime),
"auditEndTime", "auditor.audit-record.form.error.audit-end-time");
        }

        if (!super.getBuffer().getErrors().hasErrors("publishedCodeAudit"))
{
            Integer codeAuditId;
            CodeAudit codeAudit;

            codeAuditId = super.getRequest().getData("codeAuditId",
int.class);
            codeAudit = this.repository.findOneCodeAuditById(codeAuditId);

            super.state(codeAudit.isDraftMode(), "*",
"auditor.audit-record.form.error.published-code-audit");
        }

        if (!super.getBuffer().getErrors().hasErrors("auditStartTime")) {

            Date auditRecordDate = object.getAuditStartTime();
            Date minimumDate = object.getCodeAudit().getExecution();

            Boolean isAfter = auditRecordDate.after(minimumDate);
            super.state(isAfter, "auditStartTime",
"auditor.audit-record.form.error.auditStartTime");
        }
    }

    @Override
    public void perform(final AuditRecord object) {
        assert object != null;
        object.setId(0);
        this.repository.save(object);
    }

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

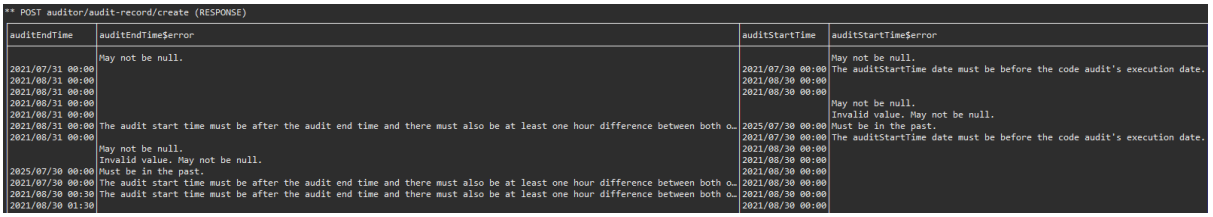
```
        Dataset dataset;
        SelectChoices choices;

        choices = SelectChoices.from(Mark.class, object.getMark());

        dataset = super.unbind(object, "code", "auditStartTime",
"auditEndTime", "mark", "link", "draftMode");
        dataset.put("marks", choices);
        dataset.put("codeAuditId", super.getRequest().getData("codeAuditId",
int.class));

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

}
```

Línea amarilla	Explicación de por qué ocurre
`super.state(auditRecordSameCode.getId() == object.getId(), "code", "auditor.audit-record.form.error.code");`	Este código no debería salir en amarillo ya que se han probado todos los casos posibles (que tenga el mismo código y que no). Se desconoce el motivo de que salga amarillo, pero haciendo el tester#analyser, se podrá comprobar esta validación.
`if (auditStartTime != null && auditEndTime != null)`	Este código no debería salir en amarillo ya que se han probado todos los casos posibles (las dos a null, una null otra no, una no otra null y las dos si). Se desconoce el motivo de que salga amarillo, pero haciendo el tester#analyser, se podrá comprobar esta validación.
`if (!super.getBuffer().getErrors().hasErrors("publishedCodeAudit"))`	Este código supone una validación realizada con el fin de evitar un posible hackeo en el campo de code audit, el usuario común nunca va a poder crear un audit record en un code audit publicado usando el sistema de forma correcta.

La línea amarilla de “`code`” se comprobó en la siguiente petición:

code	code$error
	Must match pattern "AU-[0-9]{4}-[0-9]{3}". Must not be blank.
AUa-0000-008	Must match pattern "AU-[0-9]{4}-[0-9]{3}".
AUa-0000-008	Must match pattern "AU-[0-9]{4}-[0-9]{3}".
AU-0000-008	This code is already in use.
AU-0001-000	

La línea amarilla de “`auditStartTime != null && auditEndTime != null`” se comprobó en la siguiente petición:



** POST auditor/audit-record/create (RESPONSE)

auditEndTime	auditEndTime$error	auditStartTime	auditStartTime$error
	May not be null.		May not be null.
2021/07/31 00:00		2021/07/30 00:00	The auditStartTime date must be before the code audit's execution date.
2021/08/31 00:00		2021/08/30 00:00	
2021/08/31 00:00		2021/08/30 00:00	
2021/08/31 00:00			May not be null.
2021/08/31 00:00			Invalid value. May not be null.
2021/08/31 00:00	The audit start time must be after the audit end time and there must also be at least one hour difference between both o…	2025/07/30 00:00	Must be in the past.
2021/08/31 00:00		2021/07/30 00:00	The auditStartTime date must be before the code audit's execution date.
	May not be null.	2021/08/30 00:00	
	Invalid value. May not be null.	2021/08/30 00:00	
2025/07/30 00:00	Must be in the past.	2021/08/30 00:00	
2021/07/30 00:00	The audit start time must be after the audit end time and there must also be at least one hour difference between both o…	2021/08/30 00:00	
2021/08/30 00:30	The audit start time must be after the audit end time and there must also be at least one hour difference between both o…	2021/08/30 00:00	
2021/08/30 01:30		2021/08/30 00:00	

2.1.2.4) Update

update.safe = Para hacer esta prueba, he probado a actualizar un audit record con todos los posibles valores negativos empezando por todos los valores a nulos y a continuación yendo atributo por atributo con sus respectivos casos. Tras probar los escenarios negativos actualicé el mismo audit record con los casos positivos yendo atributo por atributo, estos casos positivos son el extremo inferior, el extremo superior, un valor intermedio y valores de diferentes tipos de hacking respetando los rangos de cada atributo.

update.hack = Para hacer esta prueba he iniciado la aplicación y me he registrado con el usuario auditor1, tras iniciada la sesión me he metido en un audit record con id:124 y dándole a inspeccionar en la pestaña, he cambiado ese valor de id por 164 (audit record perteneciente al auditor2) y le di al botón de update, tras ello me volví a meter en el mismo audit record y cambié la id a 125 (audit record publicado de auditor1) y volviendo a pulsar en el botón de update. En ambos procesos me salió un error 500 de “access is not authorised”.

Gracias al intentar este tipo de test, me percaté de un error en el código relacionado con las validaciones entre las entidades padre (code audit) y las hijas (audit records), ya que las fechas de las entidades hijas deben ser siempre superior a la de la entidad padre.

Clase AuditorAuditRecordUpdateService:

```
@Service
public class AuditorAuditRecordUpdateService extends
AbstractService<Auditor, AuditRecord> {

    // Internal state
-----------------------------------------------------------

    @Autowired
    private AuditorAuditRecordRepository repository;

    // AbstractService interface
-----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int id;
        int auditorId;
        AuditRecord auditRecord;

        id = super.getRequest().getData("id", int.class);
```

```
        auditRecord = this.repository.findOneAuditRecordById(id);

        auditorId = super.getRequest().getPrincipal().getActiveRoleId();

        status = auditorId == auditRecord.getAuditor().getId() &&
auditRecord.isDraftMode();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        AuditRecord object;
        int id;

        id = super.getRequest().getData("id", int.class);

        object = this.repository.findOneAuditRecordById(id);

        super.getBuffer().addData(object);
    }

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

        Integer auditorId =
super.getRequest().getPrincipal().getActiveRoleId();
        Auditor auditor = this.repository.findOneAuditorById(auditorId);
        object.setAuditor(auditor);
        super.bind(object, "code", "auditStartTime", "auditEndTime", "mark",
"link");
    }

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

        if (!super.getBuffer().getErrors().hasErrors("code")) {

            AuditRecord auditRecordSameCode =
this.repository.findOneAuditRecordByCode(object.getCode());

            if (auditRecordSameCode != null)
                super.state(auditRecordSameCode.getId() == object.getId(),
"code", "auditor.audit-record.form.error.code");
        }

        if (!super.getBuffer().getErrors().hasErrors("auditEndTime")) {
            Date auditStartTime;
            Date auditEndTime;

            auditStartTime = object.getAuditStartTime();
```

```
            auditEndTime = object.getAuditEndTime();

            if (auditStartTime != null && auditEndTime != null)
                super.state(MomentHelper.isLongEnough(auditStartTime,
auditEndTime, 1, ChronoUnit.HOURS) && auditEndTime.after(auditStartTime),
"auditEndTime", "auditor.audit-record.form.error.audit-end-time");
        }

        if (!super.getBuffer().getErrors().hasErrors("publishedCodeAudit"))
            super.state(object.getCodeAudit().isDraftMode(), "*",
"auditor.audit-record.form.error.published-code-audit");

        if (!super.getBuffer().getErrors().hasErrors("auditStartTime")) {

            Date auditRecordDate = object.getAuditStartTime();
            Date minimumDate = object.getCodeAudit().getExecution();

            Boolean isAfter = auditRecordDate.after(minimumDate);
            super.state(isAfter, "auditStartTime",
"auditor.audit-record.form.error.auditStartTime");
        }
    }

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

        this.repository.save(object);
    }

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

        Dataset dataset;
        SelectChoices choices;

        choices = SelectChoices.from(Mark.class, object.getMark());

        dataset = super.unbind(object, "code", "auditStartTime",
"auditEndTime", "mark", "link", "draftMode");
        dataset.put("marks", choices);

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

}
```

Línea amarilla	Explicación de por qué ocurre

`if (auditRecordSameCode != null)`	Este código no debería salir en amarillo ya que se han probado todos los casos posibles (que tenga el mismo código y que no). Se desconoce el motivo de que salga amarillo, pero haciendo el tester#analyser, se podrá comprobar esta validación.
`if (auditStartTime != null && auditEndTime != null)`	Este código no debería salir en amarillo ya que se han probado todos los casos posibles (las dos a null, una null otra no, una no otra null y las dos si). Se desconoce el motivo de que salga amarillo, pero haciendo el tester#analyser, se podrá comprobar esta validación.
`if (!super.getBuffer().getErrors().hasErrors("publishedCodeAudit"))`	Este código supone una validación realizada con el fin de evitar un posible hackeo en el campo de code audit, el usuario común nunca va a poder crear un audit record en un code audit publicado usando el sistema de forma correcta.

La línea amarilla de "`code`" se comprobó en la siguiente petición:

code	code$error
	Must match pattern "AU-[0-9]{4}-[0-9]{3}". Must not be blank.
	Must match pattern "AU-[0-9]{4}-[0-9]{3}". Must not be blank.
AUa-0000-008	Must match pattern "AU-[0-9]{4}-[0-9]{3}".
AU-0000-008	This code is already in use.
AU-0000-036	

La línea amarilla de "`auditStartTime != null && auditEndTime != null`" se comprobó en la siguiente petición:

** POST auditor/audit-record/update (RESPONSE)

auditEndTime	auditEndTime$error	auditStartTime	auditStartTime$error
	May not be null.		May not be null.
2021/07/31 00:00		2021/07/30 00:00	
2021/07/31 00:00		2021/07/30 00:00	
2021/07/31 00:00		2021/07/30 00:00	
2021/07/31 00:00			May not be null.
2021/07/31 00:00		2021/07/30 00:00	Invalid value.
2021/07/31 00:00	The audit start time must be after the audit end time and there must also be at least one hour difference between both o…	2025/07/30 00:00	Must be in the past.
2021/07/31 00:00		2000/01/01 00:00	The auditStartTime date must be before the code audit's execution date.
2021/07/31 00:00		1999/01/01 00:00	The auditStartTime date must be before the code audit's execution date.
	May not be null.	2021/07/30 00:00	
2021/07/31 00:00	Invalid value.	2021/07/30 00:00	
2025/07/30 00:00	Must be in the past.	2021/07/30 00:00	
2021/07/29 23:59	The audit start time must be after the audit end time and there must also be at least one hour difference between both o…	2021/07/30 00:00	
2021/07/30 00:30	The audit start time must be after the audit end time and there must also be at least one hour difference between both o…	2021/07/30 00:00	
2021/07/31 00:00		2021/07/30 00:00	

2.1.2.5) Publish

publish.safe = Para hacer esta prueba, he probado a publicar un audit record con todos los posibles valores negativos empezando por todos los valores a nulos y a continuación yendo atributo por atributo con sus respectivos casos. Tras probar los escenarios negativos publiqué diversos audit records con los casos positivos yendo atributo por atributo, estos casos positivos son el extremo inferior, el extremo superior, un valor intermedio y valores de diferentes tipos de hacking respetando los rangos de cada atributo.

publish.hack = Para hacer esta prueba he iniciado la aplicación y me he registrado con el usuario auditor1, tras iniciada la sesión me he metido en un audit record con id:124 y dándole a inspeccionar en la pestaña, he cambiado ese valor de id por 164 (audit record perteneciente al auditor2) y le di al botón de publish, tras ello me volví a meter en el mismo

audit record y cambié la id a 125 (audit record publicado de auditor1) y volviendo a pulsar en el botón de publish. En ambos procesos me salió un error 500 de “access is not authorised”.

Gracias al intentar este tipo de test, me percaté de un error en el código relacionado con las validaciones entre las entidades padre (code audit) y las hijas (audit records), ya que las fechas de las entidades hijas deben ser siempre superior a la de la entidad padre.

Clase AuditorAuditRecordPublishService:

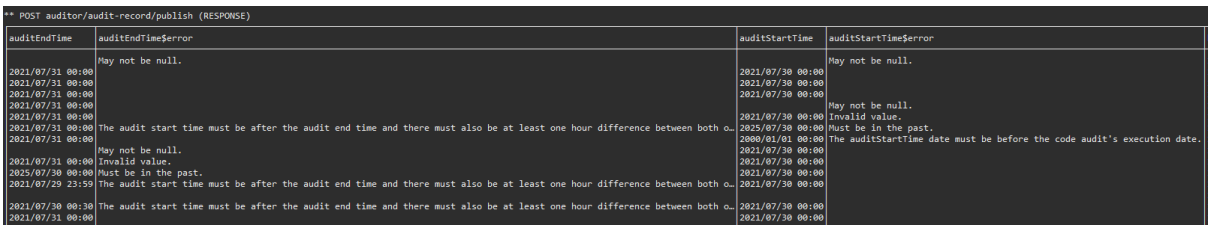
```
@Service
public class AuditorAuditRecordPublishService extends
AbstractService<Auditor, AuditRecord> {

    // Internal state
----------------------------------------------------------

    @Autowired
    private AuditorAuditRecordRepository repository;

    // AbstractService interface
-----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int id;
        int auditorId;
        AuditRecord auditRecord;

        id = super.getRequest().getData("id", int.class);
        auditRecord = this.repository.findOneAuditRecordById(id);

        auditorId = super.getRequest().getPrincipal().getActiveRoleId();

        status = auditorId == auditRecord.getAuditor().getId() &&
auditRecord.isDraftMode();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        AuditRecord object;
        int id;

        id = super.getRequest().getData("id", int.class);
        object = this.repository.findOneAuditRecordById(id);

        super.getBuffer().addData(object);
    }

    @Override
```

```
    public void bind(final AuditRecord object) {
        assert object != null;

        super.bind(object, "code", "auditStartTime", "auditEndTime", "mark",
"link", "project");
    }

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

        if (!super.getBuffer().getErrors().hasErrors("code")) {

            AuditRecord auditRecordSameCode =
this.repository.findOneAuditRecordByCode(object.getCode());

            if (auditRecordSameCode != null)
                super.state(auditRecordSameCode.getId() == object.getId(),
"code", "auditor.audit-record.form.error.code");
        }

        if (!super.getBuffer().getErrors().hasErrors("auditEndTime")) {
            Date auditStartTime;
            Date auditEndTime;

            auditStartTime = object.getAuditStartTime();
            auditEndTime = object.getAuditEndTime();

            if (auditStartTime != null && auditEndTime != null)
                super.state(MomentHelper.isLongEnough(auditStartTime,
auditEndTime, 1, ChronoUnit.HOURS) && auditEndTime.after(auditStartTime),
"auditEndTime", "auditor.audit-record.form.error.audit-end-time");
        }

        if (!super.getBuffer().getErrors().hasErrors("publishedCodeAudit"))
            super.state(object.getCodeAudit().isDraftMode(), "*",
"auditor.audit-record.form.error.published-code-audit");

        if (!super.getBuffer().getErrors().hasErrors("auditStartTime")) {

            Date auditRecordDate = object.getAuditStartTime();
            Date minimumDate = object.getCodeAudit().getExecution();

            Boolean isAfter = auditRecordDate.after(minimumDate);
            super.state(isAfter, "auditStartTime",
"auditor.audit-record.form.error.auditStartTime");
        }
    }

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

```
        object.setDraftMode(false);
        this.repository.save(object);
    }

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

        Dataset dataset;
        SelectChoices choices;

        choices = SelectChoices.from(Mark.class, object.getMark());

        dataset = super.unbind(object, "code", "auditStartTime",
"auditEndTime", "mark", "link", "draftMode");
        dataset.put("marks", choices);

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

Línea amarilla	Explicación de por qué ocurre
`if (auditStartTime != null && auditEndTime != null)`	Este código no debería salir en amarillo ya que se han probado todos los casos posibles (las dos a null, una null otra no, una no otra null y las dos si). Se desconoce el motivo de que salga amarillo, pero haciendo el tester#analyser, se podrá comprobar esta validación.
`if (!super.getBuffer().getErrors().hasErrors("publishedCodeAudit"))`	Este código supone una validación realizada con el fin de evitar un posible hackeo en el campo de code audit, el usuario común nunca va a poder crear un audit record en un code audit publicado usando el sistema de forma correcta.

La línea amarilla de “`auditStartTime != null && auditEndTime != null`” se comprobó en la siguiente petición:



** POST auditor/audit-record/publish (RESPONSE)

auditEndTime	auditEndTime$error	auditStartTime	auditStartTime$error
	May not be null.		May not be null.
2021/07/31 00:00		2021/07/30 00:00	
2021/07/31 00:00		2021/07/30 00:00	
2021/07/31 00:00		2021/07/30 00:00	
2021/07/31 00:00			May not be null.
2021/07/31 00:00		2021/07/30 00:00	Invalid value.
2021/07/31 00:00	The audit start time must be after the audit end time and there must also be at least one hour difference between both o…	2025/07/30 00:00	Must be in the past.
2021/07/31 00:00		2000/01/01 00:00	The auditStartTime date must be before the code audit's execution date.
	May not be null.	2021/07/30 00:00	
2021/07/31 00:00	Invalid value.	2021/07/30 00:00	
2025/07/30 00:00	Must be in the past.	2021/07/30 00:00	
2021/07/29 23:59	The audit start time must be after the audit end time and there must also be at least one hour difference between both o…	2021/07/30 00:00	
2021/07/30 00:30	The audit start time must be after the audit end time and there must also be at least one hour difference between both o…	2021/07/30 00:00	
2021/07/31 00:00		2021/07/30 00:00	

2.1.2.6) Delete

delete.safe = Para hacer esta prueba,he intentado borrar un audit record en dos code audits distintas ya que no existe caso negativo legal.

delete.hack = Para hacer esta prueba he iniciado la aplicación y me he registrado con el usuario auditor1, tras iniciada la sesión me he metido en un audit record con id:124 y dándole a inspeccionar en la pestaña, he cambiado ese valor de id por 164 (audit record perteneciente al auditor2) y le di al botón de delete, tras ello me volví a meter en el mismo audit record y cambié la id a 125 (audit record publicado de auditor1) y volviendo a pulsar en el botón de delete. En ambos procesos me salió un error 500 de “access is not authorised”.

Debido a que se había realizado un análisis en profundidad de los distintos requisitos realizados antes de comenzar a realizar tests, no se han encontrado bugs durante la realización de estos

Clase AuditorAuditRecordDeleteService:

```
@Service
public class AuditorAuditRecordDeleteService extends
AbstractService<Auditor, AuditRecord> {

    // Internal state
----------------------------------------------------------

    @Autowired
    private AuditorAuditRecordRepository repository;

    // AbstractService interface
-----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int id;
        int auditorId;
        AuditRecord auditRecord;

        id = super.getRequest().getData("id", int.class);
        auditRecord = this.repository.findOneAuditRecordById(id);

        auditorId = super.getRequest().getPrincipal().getActiveRoleId();

        status = auditorId == auditRecord.getAuditor().getId() &&
auditRecord.isDraftMode();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
```

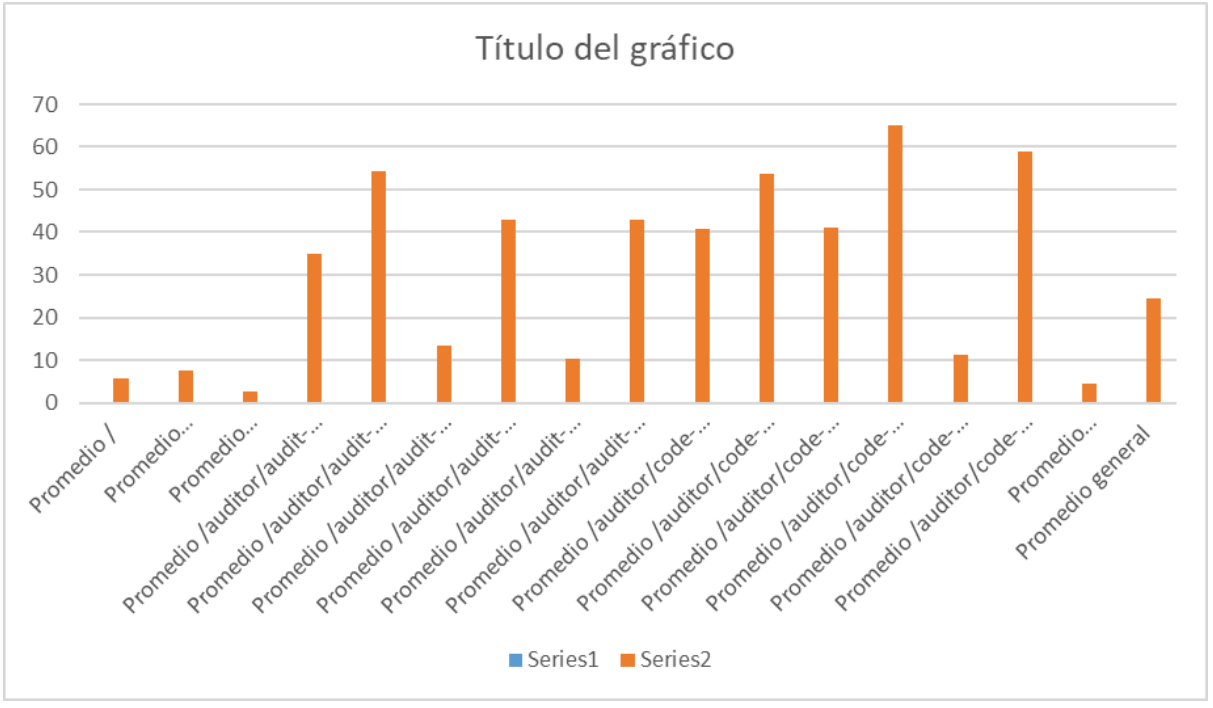
```
        AuditRecord object;
        int id;

        id = super.getRequest().getData("id", int.class);

        object = this.repository.findOneAuditRecordById(id);

        super.getBuffer().addData(object);
    }

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

        super.bind(object, "code", "auditStartTime", "auditEndTime", "mark",
"link");
    }

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

        if (!super.getBuffer().getErrors().hasErrors("publishedCodeAudit"))
            super.state(object.getCodeAudit().isDraftMode(), "*",
"auditor.audit-record.form.error.published-code-audit");
    }

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

        this.repository.delete(object);
    }

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

        Dataset dataset;
        SelectChoices choices;

        choices = SelectChoices.from(Mark.class, object.getMark());

        dataset = super.unbind(object, "code", "auditStartTime",
"auditEndTime", "mark", "link", "draftMode");
        dataset.put("marks", choices);

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

}
```

Línea amarilla	Explicación de por qué ocurre
`if (!super.getBuffer().getErrors().hasErrors("publishedCodeAudit"))`	Este código no debería salir en amarillo ya que se han probado todos los casos posibles (que el code audit esté publicado y no). Se desconoce el motivo de que salga amarillo, pero haciendo el tester#analyser, se podrá comprobar esta validación. El caso negativo solo se puede hacer en el .hack y está explicado anteriormente.

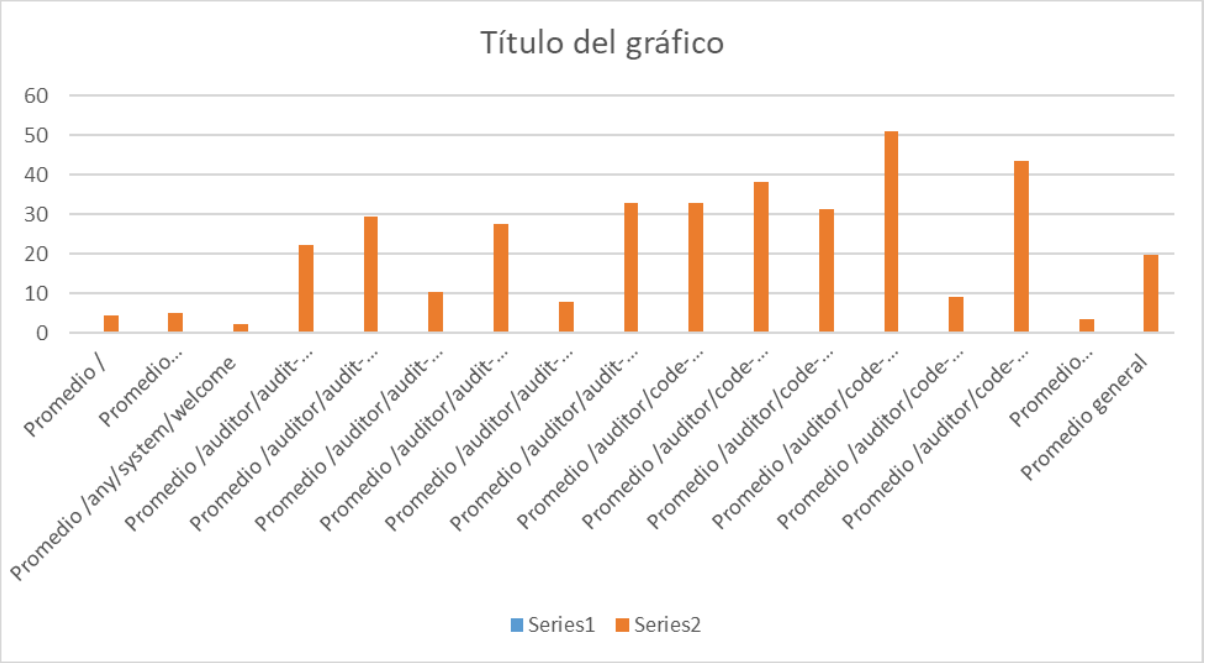
2.2) Performance testing

2.2.1) Comparativa entre dos pcs

PC_A



PC_B



Podemos observar que el PC_B ha tardado bastante menos tiempo que el PC_A.

PC_A				*PC_B*		
Media	25,384622			Media	20,6789791	
Error típico	0,65124347			Error típico	0,60155608	
Mediana	13,8624			Mediana	13,96475	
Moda	7,0386			Moda	6,908	
Desviación estándar	22,0559156			Desviación estándar	17,2259446	
Varianza de la muestra	486,463414			Varianza de la muestra	296,733168	
Curtosis	2,3455081			Curtosis	0,84404171	
Coeficiente de asimetría	1,30095116			Coeficiente de asimetría	1,02590851	
Rango	164,694			Rango	111,0586	
Mínimo	1,455			Mínimo	0,9578	
Máximo	166,149			Máximo	112,0164	
Suma	29116,1614			Suma	16956,7629	
Cuenta	1147			Cuenta	820	
Nivel de confianza(95,0%)	1,27776325			Nivel de confianza(95,0%)	1,18077322	
Interval (ms)	24,1068588	26,6623853		Interval (ms)	19,4982059	21,8597523
Interval (s)	0,02410686	0,02666239		Interval (s)	0,01949821	0,02185975

El PC_A tiene intervalo de confianza 95% (24’10, 26’66) y el PC_B tiene intervalo (19’50, 21’86). Como se aclaró anteriormente, el PC_B tiene más rendimiento que el PC_A

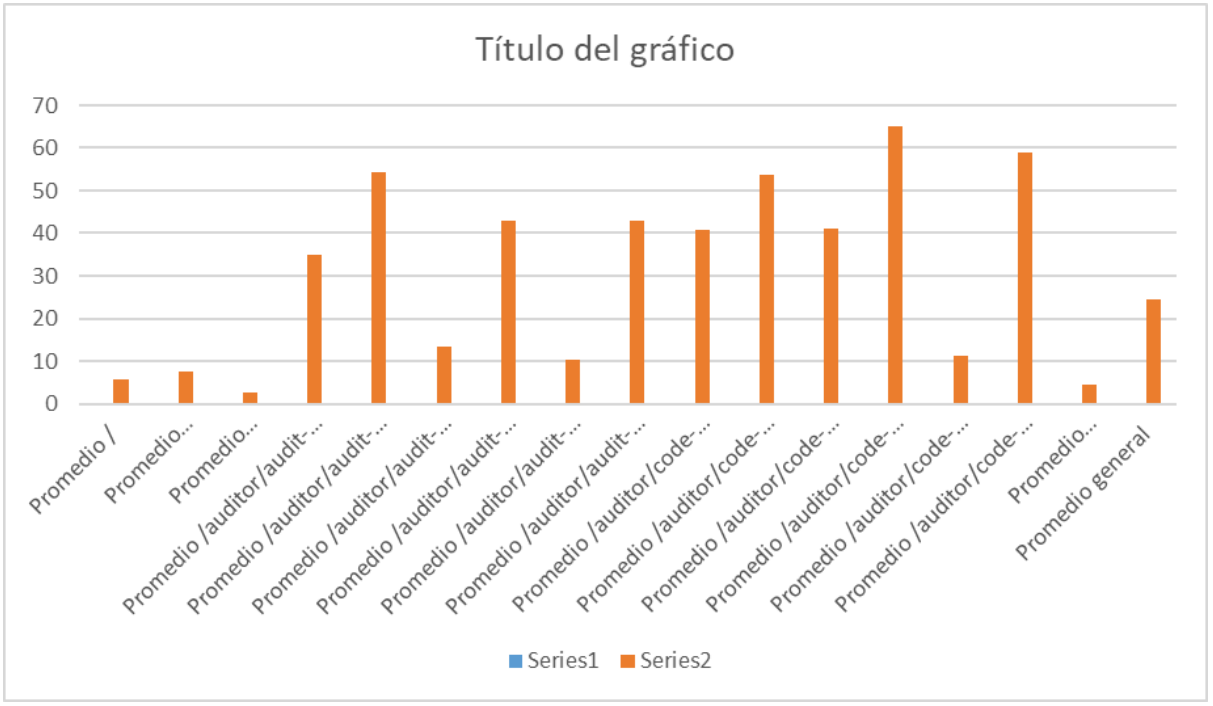
A continuación, calculamos la hipótesis de contraste con 95% confianza para intentar averiguar qué ordenador es más potente:

	PC_A	PC_B
Media	24,54069636	19,7565527
Varianza (conocida)	486,463414	296,733168
Observaciones	1198	869
Diferencia hipotética de las media	0	
z	5,533379651	
P(Z<=z) una cola	1,57059E-08	
Valor crítico de z (una cola)	1,644853627	
Valor crítico de z (dos colas)	3,14119E-08	
Valor crítico de z (dos colas)	1,959963985	

Como podemos observar, el valor P (Valor crítico de z (dos colas)) es menor que 0.05, que es nuestro porcentaje de confianza. Por esta razón sabemos que comparar las medias de los tiempos es una buena manera de averiguar qué ordenador es más potente. En este caso PC_B es mejor al tener una media de tiempos menor.

2.2.2) Comparativa con y sin índices

Sin índices



Con índices

Podemos observar que las gráficas son bastante similares, pero el tiempo sin índices ha tardado bastante menor que con índices.

Sin índices				*Con índices*		
Media	25,384622			Media	26,9943106	
Error típico	0,65124347			Error típico	0,68769499	
Mediana	13,8624			Mediana	13,8132	
Moda	7,0386			Moda	47,5988	
Desviación estándar	22,0559156			Desviación estándar	23,2904332	
Varianza de la muestra	486,463414			Varianza de la muestra	542,444279	
Curtosis	2,3455081			Curtosis	5,75347929	
Coeficiente de asimetría	1,30095116			Coeficiente de asimetría	1,5577445	
Rango	164,694			Rango	230,4546	
Mínimo	1,455			Mínimo	1,4262	
Máximo	166,149			Máximo	231,8808	
Suma	29116,1614			Suma	30962,4742	
Cuenta	1147			Cuenta	1147	
Nivel de confianza(95,0%)	1,27776325			Nivel de confianza(95,0%)	1,34928244	
Interval (ms)	24,1068588	26,6623853		Interval (ms)	25,6450281	28,343593
Interval (s)	0,02410686	0,02666239		Interval (s)	0,02564503	0,02834359

El análisis sin índices tiene intervalo de confianza 95% (24’10, 26’66) y el análisis con índices tiene intervalo (25’64, 28’34). Como se aclaró anteriormente, sin índices tiene más rendimiento que con índices.

A continuación, calculamos la hipótesis de contraste con 95% confianza para intentar averiguar qué ordenador es más potente:

	Sin índices	*Con índices*
Media	24,54069636	26,08487423
Varianza (conocida)	486,463414	542,444279
Observaciones	1198	1198
Diferencia hipotética de las medias	0	
z	-1,666239887	
P(Z<=z) una cola	0,047832822	
Valor crítico de z (una cola)	1,644853627	
Valor crítico de z (dos colas)	0,095665644	
Valor crítico de z (dos colas)	1,959963985	

Como podemos observar, el valor P (Valor crítico de z (dos colas)) es mayor que 0.05, que es nuestro porcentaje de confianza. Por esta razón sabemos que los cambios no dieron como resultado ninguna mejora significativa, los tiempos de muestreo son diferentes, pero son globalmente iguales.

3.) Conclusiones

En conclusión, la cobertura abarcada en el testing ha sido bastante satisfactoria puesto que las líneas amarillas que salen, se han argumentado con el motivo de su aparición. Y con respecto al rendimiento de las pruebas, se ha comprobado que según el pc en el que se haga, saldrán más eficientes o no, y al aplicar los índices, resulta que el tiempo es mayor en muy poca medida, aumenta el tiempo tan poco que según el análisis del z-test lo considera que no varía lo suficiente como para comparar los dos resultados.