Testing report

Diseño y pruebas II

D04

Versión 2.0

https://github.com/DP2-c2-028/Acme-SF-D04

Fecha 22/06/2024

Preparado por:
Gonzalo Navas Remmers
gonnavrem@alum.us.es
C2.028

1.Introducción **3**
2. Functional testing **4**
2.1. Testeo de funcionalidades de TrainingModule 4
2.1.1. List 4
2.1.1.1. list.safe 4
2.1.1.2. list.hack 4
2.1.1.3. Cobertura 4
2.1.2. Show 5
2.1.2.1. show.safe 5
2.1.2.2. show.hack 5
2.1.2.3 Cobertura 5
2.1.3. Create 7
2.1.3.1. create.safe 7
2.1.3.2. create.hack 7
2.1.3.3. Cobertura 7
2.1.4. Update 11
2.1.4.1. update.safe 11
2.1.4.2. update.hack 11
2.1.4.3. Cobertura 11
2.1.5. Publish 15
2.1.5.1. publish.safe 15
2.1.5.2. publish.hack 16
2.1.5.3. Cobertura 16
2.1.6. Delete 21
2.1.6.1. delete.safe 21
2.1.6.2. delete.hack 21
2.1.6.3. Cobertura 21
2.2. Testeo de funcionalidades de TrainingSession 23
2.2.1. List 23
2.2.1.1. list.safe 23
2.2.1.2. list.hack 23
2.2.1.3. Cobertura 23
2.2.2. Show 25
2.2.2.1. show.safe 25
2.2.2.2. show.hack 25
2.2.2.3 Cobertura 25
2.2.3. Create 26
2.2.3.1. create.safe 26
2.2.3.2. create.hack 27
2.2.3.3. Cobertura 27
2.2.4. Update 30
2.2.4.1. update.safe 30
2.2.4.2. update.hack 30

2.2.4.3. Cobertura 31
2.2.5. Publish 34
2.2.5.1. publish.safe 34
2.2.5.2. publish.hack 35
2.2.5.3. Cobertura 35
2.2.6. Delete 38
2.2.6.1. delete.safe 38
2.2.6.2. delete-extra.safe 38
2.2.6.3. delete.hack 38
2.2.6.4. Cobertura 39
3. Performance testing 40
3.1. Contraste de hipótesis 40
3.2. Conclusión del performance testing 42
4. Conclusiones generales 43

1.Introducción

En este documento se analizará el trabajo realizado con el objetivo de desarrollar una suite de tests funcionales para nuestro proyecto, así como un análisis del rendimiento en el que se hace un contraste de hipótesis.

Este informe tiene como objetivo principal proporcionar una visión comprensiva de los procesos de prueba ejecutados, los resultados obtenidos y las conclusiones derivadas de dichos resultados. Durante el ciclo de pruebas, se evaluaron las distintas funcionalidades para asegurar su correcto funcionamiento, rendimiento, y cumplimiento con los requisitos especificados. A continuación, se detallan las metodologías de prueba empleadas, los casos de prueba ejecutados y los defectos identificados durante el desarrollo de las pruebas.

Cabe destacar que en los análisis de cobertura se omitirá explicar los fragmentos `assert object != null;` dado que son un caso inevitable de cobertura incompleta producidos por el propio funcionamiento del framework.

2. Functional testing

2.1. Testeo de funcionalidades de TrainingModule

2.1.1. List

2.1.1.1. list.safe

Esta prueba se llevó a cabo realizando un listado de los training module de cada uno de los developers, cubriendo todos los datos de ejemplo. Funcionó como se esperaba y no se encontró ningún bug.

2.1.1.2. list.hack

Esta prueba se llevó a cabo intentando realizar un listado de los training module de developer1 desde un rol distinto al de developer cambiando la url del navegador. Provocó el error esperado (500 “access is not authorised”) y no se encontró ningún bug.

2.1.1.3. Cobertura

```
public class DeveloperTrainingModuleListService extends
AbstractService<Developer, TrainingModule> {

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

    @Autowired
    private DeveloperTrainingModuleRepository repository;

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


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

    @Override
    public void load() {
        Collection<TrainingModule> objects;
        int developerId;

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

        objects =
this.repository.findTrainingModulesByDeveloperId(developerId);

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

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

        Dataset dataset;

        dataset = super.unbind(object, "code", "creationMoment",
"updateMoment", "difficulty", "details", "totalTime", "link", "published",
"project");

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

}
```

La cobertura es completa y tal y como se esperaba.

2.1.2. Show

2.1.2.1. show.safe

Esta prueba se llevó a cabo accediendo a la mayoría de los training module a través del listado. Gracias a esta prueba se detectó un fallo en los sample data en el que un training module estaba asociado a un project no publicado, produciendo un error 500.

2.1.2.2. show.hack

Esta prueba se llevó a cabo intentando acceder a los detalles de training modules en los siguientes dos casos:

- Acceder a un training module desde un rol que no fuese developer.
- Acceder a un training module desde el developer incorrecto.

Provocaron el error esperado (500 “access is not authorised”) y no se encontró ningún bug.

2.1.2.3 Cobertura

```
public class DeveloperTrainingModuleShowService extends
AbstractService<Developer, TrainingModule> {

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

    @Autowired
    private DeveloperTrainingModuleRepository repository;

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


    @Override
```

```
    public void authorise() {
        boolean status;
        int id;
        int developerId;
        TrainingModule trainingModule;

        id = super.getRequest().getData("id", int.class);
        trainingModule = this.repository.findOneTrainingModuleById(id);

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

        status = developerId == trainingModule.getDeveloper().getId();

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

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

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

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

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

        SelectChoices difficultyChoices;
        SelectChoices projectChoices;

        difficultyChoices =
SelectChoices.from(TrainingModuleDifficulty.class, object.getDifficulty());
        projectChoices =
SelectChoices.from(this.repository.findPublishedProjects(), "title",
object.getProject());

        Dataset dataset;

        dataset = super.unbind(object, "code", "creationMoment",
"updateMoment", "difficulty", "details", "totalTime", "link", "published",
"project");
        dataset.put("difficulties", difficultyChoices);
        dataset.put("projects", projectChoices);

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

La cobertura es completa y tal y como se esperaba.

2.1.3. Create

2.1.3.1. create.safe

Esta prueba se llevó a cabo realizando gran cantidad de submits en el formulario de creación de training modules. Se introdujeron datos que comprobasen el correcto funcionamiento de todas las validaciones en las que es posible realizar dicha comprobación a través del formulario, así como un formulario vacío. Además, se introdujeron bastantes datos válidos para comprobar el correcto funcionamiento de la funcionalidad en casos normales. Todo funcionó según lo esperado y no se encontraron bugs.

2.1.3.2. create.hack

Esta prueba se llevó a cabo intentando acceder al formulario de creación de training modules desde un rol que no fuese developer. Provocó el error esperado (500 “access is not authorised”) y no se encontró ningún bug.

2.1.3.3. Cobertura

```
public class DeveloperTrainingModuleCreateService extends
AbstractService<Developer, TrainingModule> {
    // Internal state
-----------------------------------------------------------

    @Autowired
    private DeveloperTrainingModuleRepository repository;

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


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

    @Override
    public void load() {
        TrainingModule object;

        object = new TrainingModule();
        Integer developerId =
super.getRequest().getPrincipal().getActiveRoleId();
        Developer developer =
this.repository.findOneDeveloperById(developerId);
        object.setDeveloper(developer);

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

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

        super.bind(object, "code", "creationMoment", "updateMoment",
"difficulty", "details", "totalTime", "link", "project");

        Date currentMoment;
        Date creationMoment;

        currentMoment = MomentHelper.getCurrentMoment();
        creationMoment = new Date(currentMoment.getTime() - 2000);
//Substracts two seconds to ensure the moment is in the past and before the
update moment
        object.setCreationMoment(creationMoment);

        object.setPublished(false);
    }

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

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

            TrainingModule trainingModuleSameCode =
this.repository.findOneTrainingModuleByCode(object.getCode());

            super.state(trainingModuleSameCode == null, "code",
"developer.training-module.form.error.code");
        }

        if (!super.getBuffer().getErrors().hasErrors("updateMoment")) {
            Date creationMoment;
            Date updateMoment;

            creationMoment = object.getCreationMoment();
            updateMoment = object.getUpdateMoment();

            if (updateMoment != null)
                super.state(updateMoment.after(creationMoment),
"updateMoment", "developer.training-module.form.error.update-moment");
        }

        if (!super.getBuffer().getErrors().hasErrors("project"))
            super.state(!object.getProject().isDraftMode(), "project",
"developer.training-module.form.error.project");

        Date minimumDate;
        Date maximumDate;
```

```
        minimumDate = MomentHelper.parse("2000-01-01 00:00", "yyyy-MM-dd
HH:mm");
        maximumDate = MomentHelper.parse("2200-12-31 23:59", "yyyy-MM-dd
HH:mm");

        if (!super.getBuffer().getErrors().hasErrors("creationMoment"))

super.state(MomentHelper.isAfterOrEqual(object.getCreationMoment(),
minimumDate), "creationMoment",
"developer.training-module.form.error.before-min-date");

        if (!super.getBuffer().getErrors().hasErrors("creationMoment"))

super.state(MomentHelper.isBeforeOrEqual(object.getCreationMoment(),
maximumDate), "creationMoment",
"developer.training-module.form.error.after-max-date");

        if (!super.getBuffer().getErrors().hasErrors("creationMoment"))

super.state(MomentHelper.isBeforeOrEqual(object.getCreationMoment(),
MomentHelper.deltaFromMoment(maximumDate, -14, ChronoUnit.DAYS)),
"creationMoment", "developer.training-module.form.error.no-room-for-period");

        Date updateMoment = object.getUpdateMoment();
        if (updateMoment != null) {
            if (!super.getBuffer().getErrors().hasErrors("updateMoment"))

super.state(MomentHelper.isAfterOrEqual(object.getUpdateMoment(),
minimumDate), "updateMoment",
"developer.training-module.form.error.before-min-date");

            if (!super.getBuffer().getErrors().hasErrors("updateMoment"))

super.state(MomentHelper.isBeforeOrEqual(object.getUpdateMoment(),
maximumDate), "updateMoment",
"developer.training-module.form.error.after-max-date");
        }
    }

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

        object.setId(0);
        this.repository.save(object);
    }

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

        SelectChoices difficultyChoices;
        SelectChoices projectChoices;
```

```
        difficultyChoices =
SelectChoices.from(TrainingModuleDifficulty.class, object.getDifficulty());
        projectChoices =
SelectChoices.from(this.repository.findPublishedProjects(), "title", null);

        Dataset dataset;

        dataset = super.unbind(object, "code", "creationMoment",
"updateMoment", "difficulty", "details", "totalTime", "link", "published",
"project");
        dataset.put("difficulties", difficultyChoices);
        dataset.put("projects", projectChoices);

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

A pesar de que la cobertura no es completa, es la correcta. Se comentarán los casos especiales a continuación:

Fragmento	Justificación de cobertura

`if (updateMoment != null)`
 	
`super.state(updateMoment.after(creationMoment), "updateMoment", "developer.training-module.form.error.update-moment");`
	Las fechas de training module se manejan de manera automática. En la creación, updateMoment no recibe valor y nunca se entra al if en los casos que podemos probar, pero se valida igualmente para evitar POST hacking.

`if (updateMoment != null) {`
`if (!super.getBuffer().getErrors().hasErrors("updateMoment"))`
 	
`super.state(MomentHelper.isAfterOrEqual(object.getUpdateMoment(), minimumDate), "updateMoment", "developer.training-module.form.error.before-min-date");`
 	
`if (!super.getBuffer().getErrors().hasErrors("updateMoment"))`
 	
`super.state(MomentHelper.isBeforeOrEqual(object.getUpdateMoment(), maximumDate), "updateMoment", "developer.training-module.form.error.after-max-date");`
	Misma explicación que en el caso anterior.

`super.state(!object.getProject().isDraftMode(), "project", "developer.training-module.form.error.project");`	Mediante el formulario solo se pueden asignar projects publicados, así solo podríamos cubrir esto completamente mediante POST hacking

2.1.4. Update

2.1.4.1. update.safe

Esta prueba se llevó a cabo accediendo a los detalles de algunos training module e introduciendo los mismos datos que en la prueba anterior para probar el correcto funcionamiento de la funcionalidad. Todo funcionó según lo esperado y no se encontraron bugs.

2.1.4.2. update.hack

Se realizaron los siguientes casos de hacking modificando el valor de la id del formulario:

- Intentar hacer un update a un training module sin publicar desde un developer incorrecto.
- Intentar hacer un update a un training module publicado desde un developer incorrecto.
- Intentar hacer un update a un training module publicado desde un developer correcto.

Provocaron el error esperado (500 “access is not authorised”) y no se encontró ningún bug. Durante el replay de este test salen algunos FAILED por consola debido a cómo el framework recoge las peticiones en algunos casos de hacking en los que se inspecciona el formulario para cambiar valores que no deberían poder cambiarse, pero no suponen problema alguno y los test comprueban los casos de forma correcta.

2.1.4.3. Cobertura

```
public class DeveloperTrainingModuleUpdateService extends
AbstractService<Developer, TrainingModule> {
    // Internal state
-----------------------------------------------------------

    @Autowired
    private DeveloperTrainingModuleRepository repository;

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


    @Override
    public void authorise() {
        boolean status;
        int id;
        int developerId;
```

```
        TrainingModule trainingModule;

        id = super.getRequest().getData("id", int.class);
        trainingModule = this.repository.findOneTrainingModuleById(id);

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

        status = developerId == trainingModule.getDeveloper().getId() &&
!trainingModule.isPublished();

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

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

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

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

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

        Integer developerId =
super.getRequest().getPrincipal().getActiveRoleId();
        Developer developer =
this.repository.findOneDeveloperById(developerId);
        object.setDeveloper(developer);
        super.bind(object, "code", "difficulty", "details", "totalTime",
"link", "project");

        Date currentMoment;
        Date updateMoment;

        currentMoment = MomentHelper.getCurrentMoment();
        updateMoment = new Date(currentMoment.getTime() - 1000); //Substracts
one second to ensure the moment is in the past and after the creation moment
        object.setUpdateMoment(updateMoment);
    }

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

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

            TrainingModule trainingModuleSameCode =
this.repository.findOneTrainingModuleByCode(object.getCode());
```

```
            if (trainingModuleSameCode != null)
                super.state(trainingModuleSameCode.getId() == object.getId(),
"code", "developer.training-module.form.error.code");
        }

        if (!super.getBuffer().getErrors().hasErrors("updateMoment")) {
            Date creationMoment;
            Date updateMoment;

            //Creation moment is retrieved from the db because the data from
the frontend omits seconds and misleads the validation to an unwanted trigger
            creationMoment =
this.repository.findOneTrainingModuleById(object.getId()).getCreationMoment()
;
            updateMoment = object.getUpdateMoment();

            if (updateMoment != null)
                super.state(updateMoment.after(creationMoment),
"updateMoment", "developer.training-module.form.error.update-moment");
        }

        if (!super.getBuffer().getErrors().hasErrors("creationMoment")) {
            TrainingSession earliestTrainingSession;
            Boolean validCreationMoment;
            Date creationMoment =
this.repository.findOneTrainingModuleById(object.getId()).getCreationMoment()
;

            earliestTrainingSession =
this.repository.findTrainingSessionsWithEarliestDateByTrainingModuleId(object
.getId()).stream().findFirst().orElse(null);

            if (earliestTrainingSession != null) {
                validCreationMoment =
creationMoment.before(earliestTrainingSession.getPeriodStart()) &&
MomentHelper.isLongEnough(creationMoment,
earliestTrainingSession.getPeriodStart(), 1, ChronoUnit.WEEKS);
                super.state(validCreationMoment, "creationMoment",
"developer.training-module.form.error.creation-moment");
            }
        }

        if (!super.getBuffer().getErrors().hasErrors("project"))
            super.state(!object.getProject().isDraftMode(), "project",
"developer.training-module.form.error.project");

        Date MIN_DATE;
        Date MAX_DATE;

        MIN_DATE = MomentHelper.parse("2000-01-01 00:00", "yyyy-MM-dd
HH:mm");
```

```
        MAX_DATE = MomentHelper.parse("2200-12-31 23:59", "yyyy-MM-dd
HH:mm");

        if (!super.getBuffer().getErrors().hasErrors("creationMoment"))

super.state(MomentHelper.isAfterOrEqual(object.getCreationMoment(),
MIN_DATE), "creationMoment",
"developer.training-module.form.error.before-min-date");

        if (!super.getBuffer().getErrors().hasErrors("creationMoment"))

super.state(MomentHelper.isBeforeOrEqual(object.getCreationMoment(),
MAX_DATE), "creationMoment",
"developer.training-module.form.error.after-max-date");

        if (!super.getBuffer().getErrors().hasErrors("creationMoment"))

super.state(MomentHelper.isBeforeOrEqual(object.getCreationMoment(),
MomentHelper.deltaFromMoment(MAX_DATE, -14, ChronoUnit.DAYS)),
"creationMoment", "developer.training-module.form.error.no-room-for-period");

        if (!super.getBuffer().getErrors().hasErrors("updateMoment"))
            super.state(MomentHelper.isAfterOrEqual(object.getUpdateMoment(),
MIN_DATE), "updateMoment",
"developer.training-module.form.error.before-min-date");

        if (!super.getBuffer().getErrors().hasErrors("updateMoment"))

super.state(MomentHelper.isBeforeOrEqual(object.getUpdateMoment(), MAX_DATE),
"updateMoment", "developer.training-module.form.error.after-max-date");

    }

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

        this.repository.save(object);
    }

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

        SelectChoices difficultyChoices;
        SelectChoices projectChoices;

        difficultyChoices =
SelectChoices.from(TrainingModuleDifficulty.class, object.getDifficulty());
        projectChoices =
SelectChoices.from(this.repository.findPublishedProjects(), "title",
object.getProject());
```

```
        Dataset dataset;

        dataset = super.unbind(object, "code", "creationMoment",
"updateMoment", "difficulty", "details", "totalTime", "link", "published",
"project");
        dataset.put("difficulties", difficultyChoices);
        dataset.put("projects", projectChoices);

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

}
```

La cobertura no es completa pero sí es la esperada. Se explica a continuación:

Fragmento	Justificación de cobertura
`validCreationMoment = creationMoment.before(earliestTrainingSession.getPeriodStart()) && MomentHelper.isLongEnough(creationMoment, earliestTrainingSession.getPeriodStart(), 1, ChronoUnit.WEEKS);`	Las fechas de training module se manejan de manera automática, por lo que no podemos probar todas las ramas. En la modificación no se puede modificar el creationMoment, pero se implementa esta validación para implementar que mediante hacking se pueda modificar de manera que cause una inconsistencia con el inicio del periodo más temprano de sus training sessions.
`super.state(!object.getProject().isDraftMode(), "project", "developer.training-module.form.error.project");`	Mediante el formulario solo se pueden asignar projects publicados, así solo podríamos cubrir esto completamente mediante POST hacking

2.1.5. Publish

2.1.5.1. publish.safe

Esta prueba se ha llevado a cabo de manera similar a las dos anteriores. Primero se probaron que todas las validaciones funcionaban correctamente intentando publicar datos inválidos así como un formulario vacío. Acto seguido se pasó a publicar diversos training module con distintos rangos de valores válidos. Todo funcionó según lo esperado y no se encontró ningún bug.

2.1.5.2. publish.hack

Se realizaron los siguientes casos de hacking modificando el valor de la id del formulario:

- Intentar hacer un publish a un training module sin publicar desde un developer incorrecto.
- Intentar hacer un publish a un training module publicado desde un developer incorrecto.

- Intentar hacer un publish a un training module publicado desde un developer correcto.

Provocaron el error esperado (500 “access is not authorised”) y no se encontró ningún bug. Durante el replay de este test salen algunos FAILED por consola debido a cómo el framework recoge las peticiones en algunos casos de hacking en los que se inspecciona el formulario para cambiar valores que no deberían poder cambiarse, pero no suponen problema alguno y los test comprueban los casos de forma correcta.

2.1.5.3. Cobertura

```
public class DeveloperTrainingModulePublishService extends
AbstractService<Developer, TrainingModule> {
    // Internal state
-----------------------------------------------------------

    @Autowired
    private DeveloperTrainingModuleRepository repository;

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


    @Override
    public void authorise() {
        boolean status;
        int id;
        int developerId;
        TrainingModule trainingModule;

        id = super.getRequest().getData("id", int.class);
        trainingModule = this.repository.findOneTrainingModuleById(id);

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

        status = developerId == trainingModule.getDeveloper().getId() &&
!trainingModule.isPublished();

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

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

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

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

    @Override
```

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

        Integer developerId =
super.getRequest().getPrincipal().getActiveRoleId();
        Developer developer =
this.repository.findOneDeveloperById(developerId);
        object.setDeveloper(developer);
        super.bind(object, "code", "difficulty", "details", "totalTime",
"link", "project");

        Date currentMoment;
        Date updateMoment;

        currentMoment = MomentHelper.getCurrentMoment();
        updateMoment = new Date(currentMoment.getTime() - 1000); //Substracts
one second to ensure the moment is in the past and after the creation moment
        object.setUpdateMoment(updateMoment);
    }

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

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

            TrainingModule trainingModuleSameCode =
this.repository.findOneTrainingModuleByCode(object.getCode());

            if (trainingModuleSameCode != null)
                super.state(trainingModuleSameCode.getId() == object.getId(),
"code", "developer.training-module.form.error.code");
        }

        if (!super.getBuffer().getErrors().hasErrors("updateMoment")) {
            Date creationMoment;
            Date updateMoment;

            //Creation moment is retrieved from the db because the data from
the frontend omits seconds and misleads the validation to an unwanted trigger
            creationMoment =
this.repository.findOneTrainingModuleById(object.getId()).getCreationMoment()
;
            updateMoment = object.getUpdateMoment();

            if (updateMoment != null)
                super.state(updateMoment.after(creationMoment),
"updateMoment", "developer.training-module.form.error.update-moment");
        }

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

            boolean publishable;
```

```
            publishable =
!this.repository.findTrainingSessionsByTrainingModuleId(object.getId()).isEmp
ty();

            super.state(publishable, "*",
"developer.training-module.form.error.unpublishable");
        }

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

            Collection<TrainingSession> unpublishedSessions;

            unpublishedSessions =
this.repository.findUnpublishedTrainingSessionsByTrainingModuleId(object.getI
d());

            super.state(unpublishedSessions.isEmpty(), "*",
"developer.training-module.form.error.unpublished-sessions");
        }

        if (!super.getBuffer().getErrors().hasErrors("creationMoment")) {
            TrainingSession earliestTrainingSession;
            Boolean validCreationMoment;
            Date creationMoment = object.getCreationMoment();

            earliestTrainingSession =
this.repository.findTrainingSessionsWithEarliestDateByTrainingModuleId(object
.getId()).stream().findFirst().orElse(null);

            if (earliestTrainingSession != null) {
                validCreationMoment =
creationMoment.before(earliestTrainingSession.getPeriodStart()) &&
MomentHelper.isLongEnough(creationMoment,
earliestTrainingSession.getPeriodStart(), 1, ChronoUnit.WEEKS);
                super.state(validCreationMoment, "creationMoment",
"developer.training-module.form.error.creation-moment");
            }
        }

        if (!super.getBuffer().getErrors().hasErrors("project"))
            super.state(!object.getProject().isDraftMode(), "project",
"developer.training-module.form.error.project");

        Date MIN_DATE;
        Date MAX_DATE;

        MIN_DATE = MomentHelper.parse("2000-01-01 00:00", "yyyy-MM-dd
HH:mm");
        MAX_DATE = MomentHelper.parse("2200-12-31 23:59", "yyyy-MM-dd
HH:mm");
```

```
        if (!super.getBuffer().getErrors().hasErrors("creationMoment"))

super.state(MomentHelper.isAfterOrEqual(object.getCreationMoment(),
MIN_DATE), "creationMoment",
"developer.training-module.form.error.before-min-date");

        if (!super.getBuffer().getErrors().hasErrors("creationMoment"))

super.state(MomentHelper.isBeforeOrEqual(object.getCreationMoment(),
MAX_DATE), "creationMoment",
"developer.training-module.form.error.after-max-date");

        if (!super.getBuffer().getErrors().hasErrors("creationMoment"))

super.state(MomentHelper.isBeforeOrEqual(object.getCreationMoment(),
MomentHelper.deltaFromMoment(MAX_DATE, -14, ChronoUnit.DAYS)),
"creationMoment", "developer.training-module.form.error.no-room-for-period");

        if (!super.getBuffer().getErrors().hasErrors("updateMoment"))
            super.state(MomentHelper.isAfterOrEqual(object.getUpdateMoment(),
MIN_DATE), "updateMoment",
"developer.training-module.form.error.before-min-date");

        if (!super.getBuffer().getErrors().hasErrors("updateMoment"))

super.state(MomentHelper.isBeforeOrEqual(object.getUpdateMoment(), MAX_DATE),
"updateMoment", "developer.training-module.form.error.after-max-date");

    }

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

        object.setPublished(true);

        this.repository.save(object);
    }

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

        SelectChoices difficultyChoices;
        SelectChoices projectChoices;

        difficultyChoices =
SelectChoices.from(TrainingModuleDifficulty.class, object.getDifficulty());
        projectChoices =
SelectChoices.from(this.repository.findPublishedProjects(), "title",
object.getProject());

        Dataset dataset;
```

```
        dataset = super.unbind(object, "code", "creationMoment",
"updateMoment", "difficulty", "details", "totalTime", "link", "published",
"project");
        dataset.put("difficulties", difficultyChoices);
        dataset.put("projects", projectChoices);

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

}
```

La cobertura no es completa pero sí es la esperada. Se explica a continuación:

Fragmento	Justificación de cobertura
`validCreationMoment = creationMoment.before(earliestTrainingSession.getPeriodStart()) && MomentHelper.isLongEnough(creationMoment, earliestTrainingSession.getPeriodStart(), 1, ChronoUnit.WEEKS);`	Las fechas de training module se manejan de manera automática, por lo que no podemos probar todas las ramas. En la publicación no se puede modificar el creationMoment, pero se implementa esta validación para implementar que mediante hacking se pueda modificar de manera que cause una inconsistencia con el inicio del periodo más temprano de sus training sessions.
`super.state(!object.getProject().isDraftMode(), "project", "developer.training-module.form.error.project");`	Mediante el formulario solo se pueden asignar projects publicados, así solo podríamos cubrir esto completamente mediante POST hacking

2.1.6. Delete

2.1.6.1. delete.safe

Esta prueba se llevó a cabo probando en primer lugar a borrar un training module con training sessions publicadas de manera que el sistema no debe permitir al usuario que se borre, devolviendo un mensaje en el formulario que le informe del caso. Acto seguido se pasó a probar un caso en el que se realiza un borrado en cascada y a borrar la mayoría de training modules de developer1. Todo funcionó según lo esperado y no se encontró ningún bug.

2.1.6.2. delete.hack

Se realizaron los siguientes casos de hacking modificando el valor de la id del formulario:

- Intentar hacer un delete a un training module sin publicar desde un developer incorrecto.
- Intentar hacer un delete a un training module publicado desde un developer incorrecto.
- Intentar hacer un delete a un training module publicado desde un developer correcto.

Provocaron el error esperado (500 “access is not authorised”) y no se encontró ningún bug. Durante el replay de este test salen algunos FAILED por consola debido a cómo el framework recoge las peticiones en algunos casos de hacking en los que se inspecciona el formulario para cambiar valores que no deberían poder cambiarse, pero no suponen problema alguno y los test comprueban los casos de forma correcta.

2.1.6.3. Cobertura

```
public class DeveloperTrainingModuleDeleteService extends
AbstractService<Developer, TrainingModule> {

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

    @Autowired
    private DeveloperTrainingModuleRepository repository;

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


    @Override
    public void authorise() {
        boolean status;
        int id;
        int developerId;
        TrainingModule trainingModule;

        id = super.getRequest().getData("id", int.class);
        trainingModule = this.repository.findOneTrainingModuleById(id);

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

        status = developerId == trainingModule.getDeveloper().getId() &&
!trainingModule.isPublished();

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

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

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

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

    @Override
    public void bind(final TrainingModule object) {
```

```
        assert object != null;

        super.bind(object, "code", "creationMoment", "updateMoment",
"difficulty", "details", "totalTime", "link", "published", "project");
    }

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

        Collection<TrainingSession> publishedSessions;

        publishedSessions =
this.repository.findPublishedTrainingSessionsByTrainingModuleId(object.getId(
));
        super.state(publishedSessions.isEmpty(), "*",
"developer.training-module.form.error.published-sessions");
    }

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

        Collection<TrainingSession> trainingSessions =
this.repository.findTrainingSessionsByTrainingModuleId(object.getId());

        this.repository.deleteAll(trainingSessions);

        this.repository.delete(object);
    }

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

        SelectChoices difficultyChoices;
        SelectChoices projectChoices;

        difficultyChoices =
SelectChoices.from(TrainingModuleDifficulty.class, object.getDifficulty());
        projectChoices =
SelectChoices.from(this.repository.findPublishedProjects(), "title",
object.getProject());

        Dataset dataset;

        dataset = super.unbind(object, "code", "creationMoment",
"updateMoment", "difficulty", "details", "totalTime", "link", "published",
"project");
        dataset.put("difficulties", difficultyChoices);
        dataset.put("projects", projectChoices);

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

```
    }
}
```

La cobertura es completa y tal y como se esperaba.

2.2. Testeo de funcionalidades de TrainingSession

2.2.1. List

2.2.1.1. list.safe

Esta prueba se llevó a cabo realizando un listado de los training session de la mayoría de training module que los poseen. Funcionó como se esperaba y no se encontró ningún bug.

2.2.1.2. list.hack

Esta prueba se llevó a cabo intentando realizar un listado de los training session de un training module de developer2 desde un rol distinto al de developer y acto seguido desde developer1, cambiando en ambos la url del navegador. Provocaron el error esperado (500 “access is not authorised”) y no se encontró ningún bug.

2.2.1.3. Cobertura

```
public class DeveloperTrainingSessionListService extends
AbstractService<Developer, TrainingSession> {

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

    @Autowired
    private DeveloperTrainingSessionRepository repository;

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


    @Override
    public void authorise() {
        boolean status;
        int trainingModuleId;
        int developerId;
        TrainingModule trainingModule;

        trainingModuleId = super.getRequest().getData("trainingModuleId",
int.class);
        trainingModule =
this.repository.findOneTrainingModuleById(trainingModuleId);

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

        status = developerId == trainingModule.getDeveloper().getId();
```

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

    @Override
    public void load() {
        Collection<TrainingSession> objects;
        int trainingModuleId;

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

        objects =
this.repository.findTrainingSessionsByTrainingModuleId(trainingModuleId);

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

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

        Dataset dataset;

        dataset = super.unbind(object, "code", "periodStart", "periodEnd",
"location", "instructor", "contactEmail", "link", "published",
"trainingModule");

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

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

        int trainingModuleId;
        TrainingModule trainingModule;

        trainingModuleId = super.getRequest().getData("trainingModuleId",
int.class);
        trainingModule =
this.repository.findOneTrainingModuleById(trainingModuleId);

        super.getResponse().addGlobal("trainingModuleId", trainingModuleId);
        super.getResponse().addGlobal("trainingModulePublished",
trainingModule.isPublished());
    }
}
```

La cobertura es completa y tal y como se esperaba.

2.2.2. Show

2.2.2.1. show.safe

Esta prueba se llevó a cabo accediendo a la mayoría de los training session a través del listado del training module A-001. Funcionó como se esperaba y no se encontró ningún bug.

2.2.2.2. show.hack

Esta prueba se llevó a cabo intentando acceder a los detalles de training session en los siguientes dos casos:

- Acceder a una training session desde un rol que no fuese developer.
- Acceder a una training session desde el developer incorrecto.

Provocaron el error esperado (500 “access is not authorised”) y no se encontró ningún bug.

2.2.2.3 Cobertura

```
public class DeveloperTrainingSessionShowService extends
AbstractService<Developer, TrainingSession> {

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

    @Autowired
    private DeveloperTrainingSessionRepository repository;

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


    @Override
    public void authorise() {
        boolean status;
        int id;
        int DeveloperId;
        TrainingSession TrainingSession;

        id = super.getRequest().getData("id", int.class);
        TrainingSession = this.repository.findTrainingSessionById(id);

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

        status = DeveloperId ==
TrainingSession.getTrainingModule().getDeveloper().getId();

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

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

```
        id = super.getRequest().getData("id", int.class);
        object = this.repository.findTrainingSessionById(id);
        super.getBuffer().addData(object);
    }

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

        Dataset dataset;

        dataset = super.unbind(object, "code", "periodStart", "periodEnd",
"location", "instructor", "contactEmail", "link", "published",
"trainingModule");

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

El coverage es completo y tal como se esperaba.

2.2.3. Create

2.2.3.1. create.safe

Esta prueba se llevó a cabo realizando gran cantidad de submits en el formulario de creación de training sessions. Se introdujeron datos que comprobasen el correcto funcionamiento de todas las validaciones en las que es posible realizar dicha comprobación a través del formulario, así como un formulario vacío. Además, se introdujeron bastantes datos válidos para comprobar el correcto funcionamiento de la funcionalidad en casos normales. Se encontró un error en el que se producía un error 500 al introducir un comienzo o final de periodo vacío.

2.2.3.2. create.hack

Esta prueba se llevó a cabo intentando acceder al formulario de creación de training modules desde un rol que no fuese developer. Provocó el error esperado (500 “access is not authorised”) y no se encontró ningún bug.

2.2.3.3. Cobertura

```
public class DeveloperTrainingSessionCreateService extends
AbstractService<Developer, TrainingSession> {

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

    @Autowired
    private DeveloperTrainingSessionRepository repository;
```

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


    @Override
    public void authorise() {
        int developerId;
        int trainingModuleId;
        TrainingModule trainingModule;
        Boolean status;

        developerId = super.getRequest().getPrincipal().getActiveRoleId();
        trainingModuleId = super.getRequest().getData("trainingModuleId",
int.class);
        trainingModule =
this.repository.findOneTrainingModuleById(trainingModuleId);

        status = developerId == trainingModule.getDeveloper().getId();
        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        TrainingSession object;
        Integer trainingModuleId;
        TrainingModule trainingModule;

        object = new TrainingSession();

        trainingModuleId = super.getRequest().getData("trainingModuleId",
int.class);
        trainingModule =
this.repository.findOneTrainingModuleById(trainingModuleId);

        object.setTrainingModule(trainingModule);

        super.getBuffer().addData(object);

    }

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

        super.bind(object, "code", "periodStart", "periodEnd", "location",
"instructor", "contactEmail", "link");
    }

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

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

```
            TrainingSession trainingSessionSameCode =
this.repository.findOneTrainingSessionByCode(object.getCode());

            super.state(trainingSessionSameCode == null, "code",
"developer.training-session.form.error.code");
        }

        if (!super.getBuffer().getErrors().hasErrors("periodStart")) {
            Date periodStart;
            Date trainingModuleCreationMoment;
            Boolean periodStartIsValid;

            periodStart = object.getPeriodStart();
            trainingModuleCreationMoment =
object.getTrainingModule().getCreationMoment();

            if (periodStart != null) {
                periodStartIsValid =
MomentHelper.isLongEnough(trainingModuleCreationMoment, periodStart, 1,
ChronoUnit.WEEKS) && periodStart.after(trainingModuleCreationMoment);
                super.state(periodStartIsValid, "periodStart",
"developer.training-session.form.error.period-start");
            }
        }

        if (!super.getBuffer().getErrors().hasErrors("periodEnd")) {
            Date periodStart;
            Date periodEnd;
            Boolean periodEndIsValid;

            periodStart = object.getPeriodStart();
            periodEnd = object.getPeriodEnd();

            if (periodStart != null) {
                periodEndIsValid = MomentHelper.isLongEnough(periodStart,
periodEnd, 1, ChronoUnit.WEEKS) && periodEnd.after(periodStart);
                super.state(periodEndIsValid, "periodEnd",
"developer.training-session.form.error.period-end");
            }
        }

        if
(!super.getBuffer().getErrors().hasErrors("publishedTrainingModule")) {
            Integer trainingModuleId;
            TrainingModule trainingModule;

            trainingModuleId = super.getRequest().getData("trainingModuleId",
int.class);
            trainingModule =
this.repository.findOneTrainingModuleById(trainingModuleId);
```

```
            super.state(!trainingModule.isPublished(), "*",
"developer.training-session.form.error.published-training-module");
        }

        Date MIN_DATE;
        Date MAX_DATE;

        MIN_DATE = MomentHelper.parse("2000-01-01 00:00", "yyyy-MM-dd
HH:mm");
        MAX_DATE = MomentHelper.parse("2200-12-31 23:59", "yyyy-MM-dd
HH:mm");

        if (!super.getBuffer().getErrors().hasErrors("periodStart"))
            super.state(MomentHelper.isAfterOrEqual(object.getPeriodStart(),
MIN_DATE), "periodStart",
"developer.training-session.form.error.before-min-date");

        if (!super.getBuffer().getErrors().hasErrors("periodStart"))
            super.state(MomentHelper.isBeforeOrEqual(object.getPeriodStart(),
MAX_DATE), "periodStart",
"developer.training-session.form.error.after-max-date");

        if (!super.getBuffer().getErrors().hasErrors("periodStart"))
            super.state(MomentHelper.isBeforeOrEqual(object.getPeriodStart(),
MomentHelper.deltaFromMoment(MAX_DATE, -7, ChronoUnit.DAYS)), "periodStart",
"developer.training-session.form.error.no-room-for-min-period-duration");

        if (!super.getBuffer().getErrors().hasErrors("periodEnd"))
            super.state(MomentHelper.isAfterOrEqual(object.getPeriodEnd(),
MIN_DATE), "periodEnd",
"developer.training-session.form.error.before-min-date");

        if (!super.getBuffer().getErrors().hasErrors("periodEnd"))
            super.state(MomentHelper.isBeforeOrEqual(object.getPeriodEnd(),
MAX_DATE), "periodEnd",
"developer.training-session.form.error.after-max-date");

        if (!super.getBuffer().getErrors().hasErrors("periodEnd"))
            super.state(MomentHelper.isAfterOrEqual(object.getPeriodEnd(),
MomentHelper.deltaFromMoment(MIN_DATE, 7, ChronoUnit.DAYS)), "periodEnd",
"developer.training-session.form.error.no-room-for-min-period-duration");

    }

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

        object.setId(0);
        this.repository.save(object);
    }

    @Override
```

```
    public void unbind(final TrainingSession object) {
        assert object != null;

        Dataset dataset;

        dataset = super.unbind(object, "code", "periodStart", "periodEnd",
"location", "instructor", "contactEmail", "link", "published",
"trainingModule");
        dataset.put("trainingModuleId",
super.getRequest().getData("trainingModuleId", int.class));

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

}
```

La cobertura es completa y tal y como se esperaba.

2.2.4. Update

2.2.4.1. update.safe

Esta prueba se llevó a cabo accediendo a los detalles de algunos training session e introduciendo los mismos datos que en la prueba anterior para probar el correcto funcionamiento de la funcionalidad. Todo funcionó según lo esperado y no se encontraron bugs.

2.2.4.2. update.hack

Se realizaron los siguientes casos de hacking modificando el valor de la id del formulario:

- Intentar hacer un update a una training session sin publicar desde un developer incorrecto.
- Intentar hacer un update a una training session publicada desde un developer incorrecto.
- Intentar hacer un update a una training session publicada desde un developer correcto.

Provocaron el error esperado (500 “access is not authorised”) y no se encontró ningún bug. Durante el replay de este test salen algunos FAILED por consola debido a cómo el framework recoge las peticiones en algunos casos de hacking en los que se inspecciona el formulario para cambiar valores que no deberían poder cambiarse, pero no suponen problema alguno y los test comprueban los casos de forma correcta.

2.2.4.3. Cobertura

```
public class DeveloperTrainingSessionUpdateService extends
AbstractService<Developer, TrainingSession> {

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

```
    @Autowired
    private DeveloperTrainingSessionRepository repository;

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


    @Override
    public void authorise() {
        boolean status;
        int id;
        int developerId;
        TrainingSession trainingSession;

        id = super.getRequest().getData("id", int.class);
        trainingSession = this.repository.findTrainingSessionById(id);

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

        status = developerId ==
trainingSession.getTrainingModule().getDeveloper().getId() &&
!trainingSession.isPublished();

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

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

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

        object = this.repository.findTrainingSessionById(id);

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

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

        super.bind(object, "code", "periodStart", "periodEnd", "location",
"instructor", "contactEmail", "link");
    }

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

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

```
            TrainingSession trainingSessionSameCode =
this.repository.findOneTrainingSessionByCode(object.getCode());

            if (trainingSessionSameCode != null)
                super.state(trainingSessionSameCode.getId() ==
object.getId(), "code", "developer.training-session.form.error.code");
        }

        if (!super.getBuffer().getErrors().hasErrors("periodStart")) {
            Date periodStart;
            Date trainingModuleCreationMoment;
            Boolean periodStartIsValid;

            periodStart = object.getPeriodStart();
            trainingModuleCreationMoment =
object.getTrainingModule().getCreationMoment();

            if (periodStart != null) {
                periodStartIsValid =
MomentHelper.isLongEnough(trainingModuleCreationMoment, periodStart, 1,
ChronoUnit.WEEKS) && periodStart.after(trainingModuleCreationMoment);
                super.state(periodStartIsValid, "periodStart",
"developer.training-session.form.error.period-start");
            }
        }

        if (!super.getBuffer().getErrors().hasErrors("periodEnd")) {
            Date periodStart;
            Date periodEnd;
            Boolean periodEndIsValid;

            periodStart = object.getPeriodStart();
            periodEnd = object.getPeriodEnd();

            if (periodStart != null) {
                periodEndIsValid = MomentHelper.isLongEnough(periodStart,
periodEnd, 1, ChronoUnit.WEEKS) && periodEnd.after(periodStart);
                super.state(periodEndIsValid, "periodEnd",
"developer.training-session.form.error.period-end");
            }
        }

        if
(!super.getBuffer().getErrors().hasErrors("publishedTrainingModule"))
            super.state(!object.getTrainingModule().isPublished(), "*",
"developer.training-session.form.error.published-training-module");

        Date MIN_DATE;
        Date MAX_DATE;

        MIN_DATE = MomentHelper.parse("2000-01-01 00:00", "yyyy-MM-dd
HH:mm");
```

```
        MAX_DATE = MomentHelper.parse("2200-12-31 23:59", "yyyy-MM-dd
HH:mm");

        if (!super.getBuffer().getErrors().hasErrors("periodStart"))
            super.state(MomentHelper.isAfterOrEqual(object.getPeriodStart(),
MIN_DATE), "periodStart",
"developer.training-session.form.error.before-min-date");

        if (!super.getBuffer().getErrors().hasErrors("periodStart"))
            super.state(MomentHelper.isBeforeOrEqual(object.getPeriodStart(),
MAX_DATE), "periodStart",
"developer.training-session.form.error.after-max-date");

        if (!super.getBuffer().getErrors().hasErrors("periodStart"))
            super.state(MomentHelper.isBeforeOrEqual(object.getPeriodStart(),
MomentHelper.deltaFromMoment(MAX_DATE, -7, ChronoUnit.DAYS)), "periodStart",
"developer.training-session.form.error.no-room-for-min-period-duration");

        if (!super.getBuffer().getErrors().hasErrors("periodEnd"))
            super.state(MomentHelper.isAfterOrEqual(object.getPeriodEnd(),
MIN_DATE), "periodEnd",
"developer.training-session.form.error.before-min-date");

        if (!super.getBuffer().getErrors().hasErrors("periodEnd"))
            super.state(MomentHelper.isBeforeOrEqual(object.getPeriodEnd(),
MAX_DATE), "periodEnd",
"developer.training-session.form.error.after-max-date");

        if (!super.getBuffer().getErrors().hasErrors("periodEnd"))
            super.state(MomentHelper.isAfterOrEqual(object.getPeriodEnd(),
MomentHelper.deltaFromMoment(MIN_DATE, 7, ChronoUnit.DAYS)), "periodEnd",
"developer.training-session.form.error.no-room-for-min-period-duration");

    }

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

        this.repository.save(object);
    }

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

        Dataset dataset;

        dataset = super.unbind(object, "code", "periodStart", "periodEnd",
"location", "instructor", "contactEmail", "link", "published");

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

```
}
```

La cobertura no es completa pero sí la esperada. Se explica a continuación:

Fragmento	Justificación de cobertura
`super.state(!object.getTrainingModule().isPublished(), "*", "developer.training-session.form.error.published-training-module");`	A través de la aplicación no se puede realizar esta acción sobre una training session publicada, pero se introduce esta validación para prevenir hacking aunque no podemos probar todas sus ramas.

2.2.5. Publish

2.2.5.1. publish.safe

Esta prueba se ha llevado a cabo de manera similar a las dos anteriores. Primero se probaron que todas las validaciones funcionaban correctamente intentando publicar datos inválidos así como un formulario vacío. Acto seguido se pasó a publicar todas las training session del training module A-001 con distintos rangos de valores válidos. Todo funcionó según lo esperado y no se encontró ningún bug.

Cabe destacar que tras la implementación de los índices, este test devuelve algunos FAILED por consola durante su replay. Esto se debe a que si bien las peticiones devuelven la misma información, la implementación de un índice (training_module_id, published) en la entidad training session cambia el orden en que se devuelven dichas entidades para las queries afectadas. Es por esto que a pesar de aparecer fallos en la consola, estos no suponen ningún problema.

2.2.5.2. publish.hack

Se realizaron los siguientes casos de hacking modificando el valor de la id del formulario:

- Intentar hacer un publish a una training session sin publicar desde un developer incorrecto.
- Intentar hacer un publish a una training session publicada desde un developer incorrecto.
- Intentar hacer un publish a una training session publicada desde un developer correcto.

Provocaron el error esperado (500 “access is not authorised”) y no se encontró ningún bug. Durante el replay de este test salen algunos FAILED por consola debido a cómo el framework recoge las peticiones en algunos casos de hacking en los que se inspecciona el formulario para cambiar valores que no deberían poder cambiarse, pero no suponen problema alguno y los test comprueban los casos de forma correcta.

2.2.5.3. Cobertura

```
public class DeveloperTrainingSessionPublishService extends
AbstractService<Developer, TrainingSession> {
```

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

    @Autowired
    private DeveloperTrainingSessionRepository repository;

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


    @Override
    public void authorise() {
        boolean status;
        int id;
        int developerId;
        TrainingSession trainingSession;

        id = super.getRequest().getData("id", int.class);
        trainingSession = this.repository.findTrainingSessionById(id);

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

        status = developerId ==
trainingSession.getTrainingModule().getDeveloper().getId() &&
!trainingSession.isPublished();

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

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

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

        object = this.repository.findTrainingSessionById(id);

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

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

        super.bind(object, "code", "periodStart", "periodEnd", "location",
"instructor", "contactEmail", "link");
    }

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

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

            TrainingSession trainingSessionSameCode =
this.repository.findOneTrainingSessionByCode(object.getCode());

            if (trainingSessionSameCode != null)
                super.state(trainingSessionSameCode.getId() ==
object.getId(), "code", "developer.training-session.form.error.code");
        }

        if (!super.getBuffer().getErrors().hasErrors("periodStart")) {
            Date periodStart;
            Date trainingModuleCreationMoment;
            Boolean periodStartIsValid;

            periodStart = object.getPeriodStart();
            trainingModuleCreationMoment =
object.getTrainingModule().getCreationMoment();

            if (periodStart != null) {
                periodStartIsValid =
MomentHelper.isLongEnough(trainingModuleCreationMoment, periodStart, 1,
ChronoUnit.WEEKS) && periodStart.after(trainingModuleCreationMoment);
                super.state(periodStartIsValid, "periodStart",
"developer.training-session.form.error.period-start");
            }
        }

        if (!super.getBuffer().getErrors().hasErrors("periodEnd")) {
            Date periodStart;
            Date periodEnd;
            Boolean periodEndIsValid;

            periodStart = object.getPeriodStart();
            periodEnd = object.getPeriodEnd();

            if (periodStart != null) {
                periodEndIsValid = MomentHelper.isLongEnough(periodStart,
periodEnd, 1, ChronoUnit.WEEKS) && periodEnd.after(periodStart);
                super.state(periodEndIsValid, "periodEnd",
"developer.training-session.form.error.period-end");
            }
        }

        Date MIN_DATE;
        Date MAX_DATE;

        MIN_DATE = MomentHelper.parse("2000-01-01 00:00", "yyyy-MM-dd
HH:mm");
        MAX_DATE = MomentHelper.parse("2200-12-31 23:59", "yyyy-MM-dd
HH:mm");
```

```
        if (!super.getBuffer().getErrors().hasErrors("periodStart"))
            super.state(MomentHelper.isAfterOrEqual(object.getPeriodStart(),
MIN_DATE), "periodStart",
"developer.training-session.form.error.before-min-date");

        if (!super.getBuffer().getErrors().hasErrors("periodStart"))
            super.state(MomentHelper.isBeforeOrEqual(object.getPeriodStart(),
MAX_DATE), "periodStart",
"developer.training-session.form.error.after-max-date");

        if (!super.getBuffer().getErrors().hasErrors("periodStart"))
            super.state(MomentHelper.isBeforeOrEqual(object.getPeriodStart(),
MomentHelper.deltaFromMoment(MAX_DATE, -7, ChronoUnit.DAYS)), "periodStart",
"developer.training-session.form.error.no-room-for-min-period-duration");

        if (!super.getBuffer().getErrors().hasErrors("periodEnd"))
            super.state(MomentHelper.isAfterOrEqual(object.getPeriodEnd(),
MIN_DATE), "periodEnd",
"developer.training-session.form.error.before-min-date");

        if (!super.getBuffer().getErrors().hasErrors("periodEnd"))
            super.state(MomentHelper.isBeforeOrEqual(object.getPeriodEnd(),
MAX_DATE), "periodEnd",
"developer.training-session.form.error.after-max-date");

        if (!super.getBuffer().getErrors().hasErrors("periodEnd"))
            super.state(MomentHelper.isAfterOrEqual(object.getPeriodEnd(),
MomentHelper.deltaFromMoment(MIN_DATE, 7, ChronoUnit.DAYS)), "periodEnd",
"developer.training-session.form.error.no-room-for-min-period-duration");

    }

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

        object.setPublished(true);

        this.repository.save(object);
    }

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

        Dataset dataset;

        dataset = super.unbind(object, "code", "periodStart", "periodEnd",
"location", "instructor", "contactEmail", "link", "published");

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

```
}
```

La cobertura es completa y tal y como se esperaba.

2.2.6. Delete

2.2.6.1. delete.safe

Esta prueba se llevó a cabo eliminando todas las training session del training module A-001. Todo funcionó según lo esperado y no se encontró ningún bug.

2.2.6.2. delete-extra.safe

Esta prueba se llevó a cabo para añadir un caso no probado.

2.2.6.3. delete.hack

Se realizaron los siguientes casos de hacking modificando el valor de la id del formulario:

- Intentar hacer un delete a un training module sin publicar desde un developer incorrecto.
- Intentar hacer un delete a un training module publicado desde un developer incorrecto.
- Intentar hacer un delete a un training module publicado desde un developer correcto.

Provocaron el error esperado (500 “access is not authorised”) y no se encontró ningún bug. Durante el replay de este test salen algunos FAILED por consola debido a cómo el framework recoge las peticiones en algunos casos de hacking en los que se inspecciona el formulario para cambiar valores que no deberían poder cambiarse, pero no suponen problema alguno y los test comprueban los casos de forma correcta.

2.2.6.4. Cobertura

```
public class DeveloperTrainingSessionDeleteService extends
AbstractService<Developer, TrainingSession> {

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

    @Autowired
    private DeveloperTrainingSessionRepository repository;

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


    @Override
    public void authorise() {
        boolean status;
        int id;
        int developerId;
        TrainingSession trainingSession;
```

```
        id = super.getRequest().getData("id", int.class);
        trainingSession = this.repository.findTrainingSessionById(id);

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

        status = developerId ==
trainingSession.getTrainingModule().getDeveloper().getId() &&
!trainingSession.isPublished();

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

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

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

        object = this.repository.findTrainingSessionById(id);

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

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

        super.bind(object, "code", "periodStart", "periodEnd", "location",
"instructor", "contactEmail", "link");
    }

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

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

        this.repository.delete(object);
    }

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

        Dataset dataset;

        dataset = super.unbind(object, "code", "periodStart", "periodEnd",
"location", "instructor", "contactEmail", "link", "published");
```

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

}
```

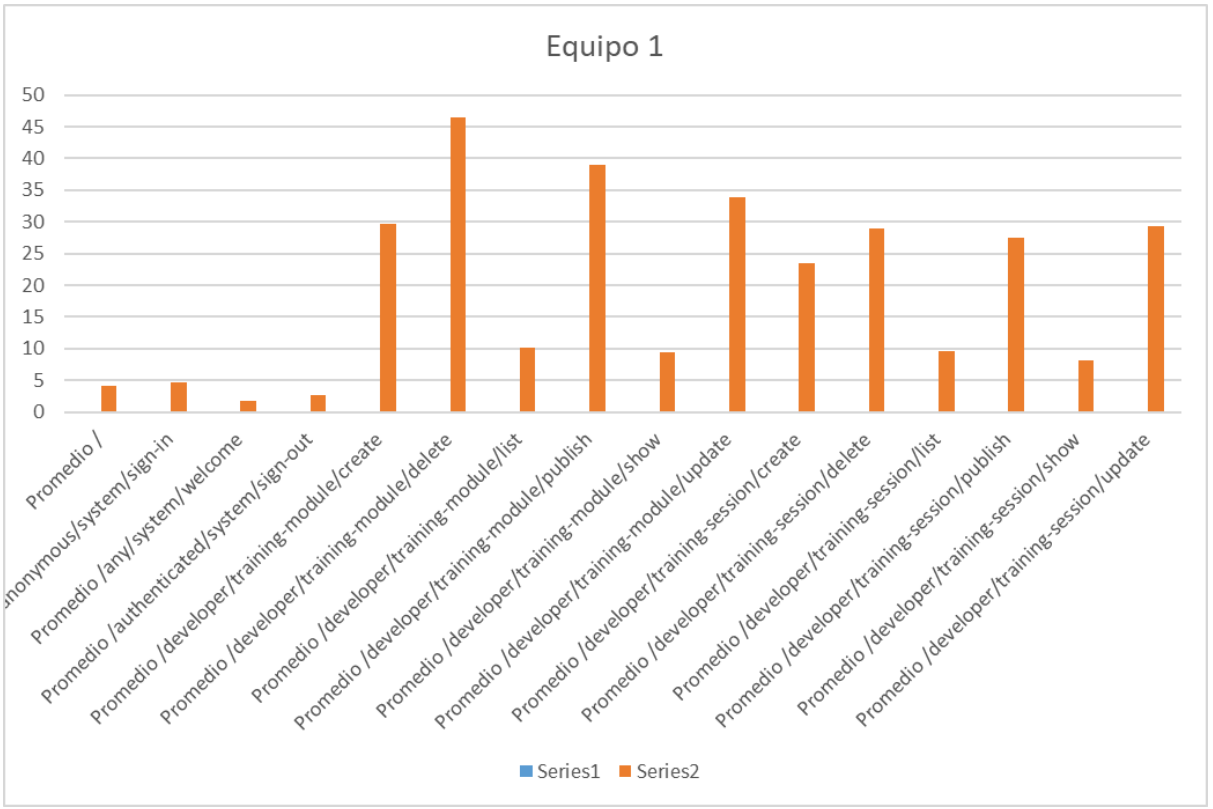
La cobertura es completa y tal y como se esperaba.

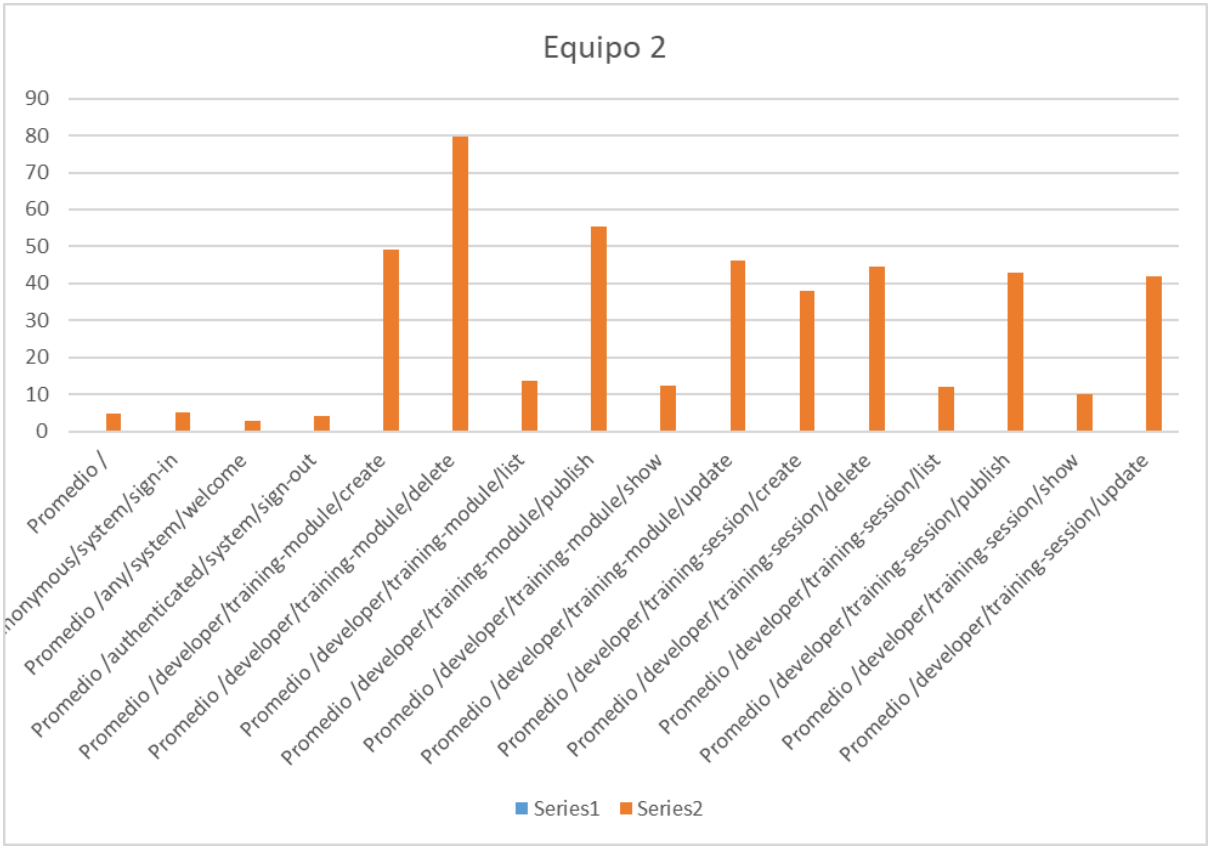
3. Performance testing

3.1. Contraste de hipótesis

Se ha ejecutado la misma suite de tests en dos equipos distintos. El equipo 1 corresponde a mi propio ordenador y el equipo 2 al de Pablo Jesús Castellanos Compaña, estudiante 4 de este mismo grupo.

Se muestran a continuación gráficos con el promedio de los tiempos para cada request path:





Observamos que el aspecto de las gráficas es similar aunque la escala del equipo 2 es mayor (tardan más de media).

A continuación se muestran las estadísticas descriptivas de los análisis para cada equipo:

Equipo 1				*Equipo 2*		
Media	15.34808952			Media	22.69702146	
Error típico	0.402003183			Error típico	0.625080309	
Mediana	9.29275			Mediana	12.2737	
Moda	28.9814			Moda	11.3709	
Desviación estándar	13.93739877			Desviación estándar	22.41590629	
Varianza de la muestra	194.2510846			Varianza de la muestra	502.4728549	
Curtosis	4.575910459			Curtosis	2.038091479	
Coeficiente de asimetría	1.682045491			Coeficiente de asimetría	1.503019097	
Rango	130.1944			Rango	168.2307	
Mínimo	0.7408			Mínimo	1.0641	
Máximo	130.9352			Máximo	169.2948	
Suma	18448.4036			Suma	29188.3696	
Cuenta	1202			Cuenta	1286	
Nivel de confianza(95.0%)	0.788706604			Nivel de confianza(95.0%)	1.226289939	
Interval (ms)	14.55938291	16.13679612		Interval (ms)	21.47073152	23.9233114
Interval (s)	0.014559383	0.016136796		Interval (s)	0.021470732	0.02392331

Si revisamos los intervalos comprobamos que en ambos equipos son buenos tiempos. Por último realizamos el Z-test para clarificar definitivamente si ha habido un cambio significativo en el rendimiento

	Equipo 1	*Equipo 2*
Media	15.34808952	22.6970215
Varianza (conocida)	194.2510846	502.472855
Observaciones	1202	1286
Diferencia hipotética de las medias	0	
z	-9.888357098	
P(Z<=z) una cola	0	
Valor crítico de z (una cola)	1.644853627	
Valor crítico de z (dos colas)	0	
Valor crítico de z (dos colas)	1.959963985	

Como se puede comprobar el p-value es 0, lo que nos dice claramente que la diferencia de rendimiento es significativa, aunque en este caso es a peor. Vemos que el equipo 2, cuyas especificaciones de hardware son superiores al del equipo 1, ha resultado desempeñar peor con una notable diferencia.

3.2. Conclusión del performance testing

Tras realizar este procedimiento hemos podido comprobar que, en nuestro contexto, una mejora en cuanto a potencia hardware no se ha traducido directamente en una mejora de rendimiento, llegando incluso a ser peor en un equipo con mejores características en cuanto a potencia de procesamiento.

A juicio personal he de decir que este proceso me ha resultado tremendamente útil y una manera muy interesante de plasmar de manera objetiva el rendimiento de un sistema y compararlo tras realizar cambios en las variables que pueden influir en ello.

4. Conclusiones generales

En conclusión, tras realizar todo el proceso de testing durante este Sprint se ha podido comprobar que lo desarrollado durante el proyecto cumple en términos generales con lo requerido, rindiendo bien y respondiendo correctamente ante casos tanto válidos como inválidos. Ha servido además para solucionar algunos errores críticos que no habían sido detectados durante el testing informal.