Testing report

Diseño y pruebas II

Sprint 4

Versión 1.0

Fecha 26/05/2024

Preparado por:
Pablo Jesús Castellanos Compaña

1) Introducción

En este documento se va a hablar sobre los casos de prueba implementados y el rendimiento de los mismos del student 4. 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 reflejan en el código gracias al coverage. La siguiente línea aparece 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: 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) Sponsorship

2.1.1.1) List

- list.safe: para hacer esta prueba, me he metido en las listas de sponsorships de todos los usuarios de mi rol (sponsor 1, 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/sponsor/sponsorship/list dando 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 los mismos.

- class SponsorSponsorshipListService:

```
@Service
public class SponsorSponsorshipListService extends AbstractService<Sponsor, Sponsorship> {

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

    @Autowired
    private SponsorSponsorshipRepository repository;

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


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

    @Override
    public void load() {
        Collection<Sponsorship> objects;
        int sponsorId;

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

        objects = this.repository.findSponsorshipBySponsorId(sponsorId);

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

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

        Dataset dataset;

        dataset = super.unbind(object, "code", "moment", "durationStartTime", "durationEndTime", "amount", "type", "email", "link");
        dataset.put("project", object.getProject().getCode());

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

}
```

Línea amarilla (no completamente probada)	Explicación
Intencionalmente en blanco	Intencionalmente en blanco

2.1.1.2) Show

- show.safe: para hacer esta prueba, me he metido en las listas de sponsorships de los usuarios de mi rol (sponsor 1 y 2) y me he metido en todas las sponsorships.

- 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 sponsorship del sponsor1: http://localhost:8082/acme-sf-d04/sponsor/sponsorship/show?id=274 dandome un error 500 de “access is not authorised”. Y también me he registrado como sponsor2 intentando acceder a ese sponsorship 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 los mismos.

- class SponsorSponsorshipShowService:

```
1.  @Service
2.  public class SponsorSponsorshipShowService extends
    AbstractService<Sponsor, Sponsorship> {
3.
4.      // Internal state
    ----------------------------------------------------------
5.
6.      @Autowired
7.      private SponsorSponsorshipRepository repository;
8.
9.      // AbstractService interface
    ----------------------------------------------
10.
11.
12.     @Override
13.     public void authorise() {
14.         boolean status;
15.         int id;
16.         int sponsorId;
17.         Sponsorship sponsorship;
18.
19.         id = super.getRequest().getData("id", int.class);
20.         sponsorship = this.repository.findOneSponsorshipById(id);
21.
22.         sponsorId =
    super.getRequest().getPrincipal().getActiveRoleId();
23.
24.         status = sponsorId == sponsorship.getSponsor().getId();
25.
26.         super.getResponse().setAuthorised(status);
```

```
27.     }
28.
29.     @Override
30.     public void load() {
31.         Sponsorship object;
32.         int id;
33.
34.         id = super.getRequest().getData("id", int.class);
35.         object = this.repository.findOneSponsorshipById(id);
36.
37.         super.getBuffer().addData(object);
38.     }
39.
40.     @Override
41.     public void unbind(final Sponsorship object) {
42.         assert object != null;
43.
44.         SelectChoices choices;
45.         Collection<Project> projects = this.repository.findProjects();
46.         SelectChoices choices2;
47.         Dataset dataset;
48.
49.         choices = SelectChoices.from(SponsorshipType.class,
    object.getType());
50.
51.         choices2 = SelectChoices.from(projects, "code", (Project)
    projects.toArray()[0]);
52.
53.         dataset = super.unbind(object, "code", "moment",
    "durationStartTime", "durationEndTime", "amount", "type", "email",
    "link", "draftMode");
54.         dataset.put("types", choices);
55.         dataset.put("projects", choices2);
56.
57.         super.getResponse().addData(dataset);
58.     }
59. }
```

Línea amarilla (no completamente probada)	Explicación
Intencionalmente en blanco	Intencionalmente en blanco

2.1.1.3) Create

- create.safe: para hacer esta prueba, he probado a crear un sponsorship 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 sponsorships con los casos positivos yendo atributo por atributo, estos casos positivos son los extremos tanto superiores como inferiores y también valores intermedios; 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/sponsor/sponsorship/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 los mismos.

- class SponsorSponsorshipCreateService:

```
@Service
public class SponsorSponsorshipCreateService extends
AbstractService<Sponsor, Sponsorship> {

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

    @Autowired
    private SponsorSponsorshipRepository    repository;

    @Autowired
    private SystemConfigurationRepository
systemConfigurationRepository;

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


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

    @Override
    public void load() {
        Sponsorship object;

        object = new Sponsorship();
        Integer sponsorId =
super.getRequest().getPrincipal().getActiveRoleId();
        Sponsor sponsor =
this.repository.findOneSponsorById(sponsorId);
        object.setSponsor(sponsor);
        object.setDraftMode(true);

```

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

    }

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

        super.bind(object, "code", "moment", "durationStartTime",
"durationEndTime", "amount", "type", "email", "link", "project");
    }

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

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

            Sponsorship projectSameCode =
this.repository.findOneSponsorshipByCode(object.getCode());

            super.state(projectSameCode == null, "code",
"sponsor.sponsorship.form.error.code");
        }

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

            Date sponsorshipDate = object.getMoment();
            Date minimumDate = MomentHelper.parse("1969-12-31 0:00",
"yyyy-MM-dd HH:mm");
            Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
"yyyy-MM-dd HH:mm");

            if (sponsorshipDate != null) {
                Boolean isAfter = sponsorshipDate.after(minimumDate)
&& sponsorshipDate.before(maximumDate);
                super.state(isAfter, "moment",
"sponsor.sponsorship.form.error.moment");
            }
        }

        if
(!super.getBuffer().getErrors().hasErrors("durationStartTime")) {
            Date durationStartTime;
            Date moment;
            durationStartTime = object.getDurationStartTime();
            moment = object.getMoment();
            Date minimumDate = MomentHelper.parse("1969-12-31 0:00",
"yyyy-MM-dd HH:mm");
            Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
"yyyy-MM-dd HH:mm");

            if (moment != null)
```

```
                super.state(durationStartTime.after(moment) &&
durationStartTime.after(minimumDate) &&
durationStartTime.before(maximumDate), "durationStartTime",
"sponsor.sponsorship.form.error.durationStartTime");
            }

            if
(!super.getBuffer().getErrors().hasErrors("durationEndTime")) {
                Date durationStartTime;
                Date durationEndTime;

                durationStartTime = object.getDurationStartTime();
                durationEndTime = object.getDurationEndTime();
                Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
"yyyy-MM-dd HH:mm");

                if (durationStartTime != null && durationEndTime != null)

super.state(MomentHelper.isLongEnough(durationStartTime,
durationEndTime, 1, ChronoUnit.MONTHS) &&
durationEndTime.after(durationStartTime) &&
durationEndTime.before(maximumDate), "durationEndTime",
                        "sponsor.sponsorship.form.error.durationEndTime");
            }

            if (!super.getBuffer().getErrors().hasErrors("amount") &&
this.systemConfigurationRepository.existsCurrency(object.getAmount().g
etCurrency()))
                super.state(object.getAmount().getAmount() >= 0 &&
object.getAmount().getAmount() <= 1000000, "amount",
"sponsor.sponsorship.form.error.amount");

            if (!super.getBuffer().getErrors().hasErrors("amount")) {
                String symbol = object.getAmount().getCurrency();
                boolean existsCurrency =
this.systemConfigurationRepository.existsCurrency(symbol);
                super.state(existsCurrency, "amount",
"sponsor.sponsorship.form.error.acceptedCurrency");
            }

            if (!super.getBuffer().getErrors().hasErrors("project"))
                super.state(!object.getProject().isDraftMode(), "project",
"sponsor.sponsorship.form.error.project-not-published");

        }

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

            this.repository.save(object);
        }

```

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

        SelectChoices choices;

        Collection<Project> projects = this.repository.findProjects();
        SelectChoices choices2;
        Dataset dataset;
        String projectCode;

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

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

        dataset = super.unbind(object, "code", "moment",
"durationStartTime", "durationEndTime", "amount", "type", "email",
"link", "project", "draftMode");
        dataset.put("types", choices);
        dataset.put("projects", choices2);
        dataset.put("project", projectCode);

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

}
```

Línea amarilla (no completamente probada)	Explicación
`if (sponsorshipDate != null) { Boolean isAfter = sponsorshipDate.after(minimumDate) && sponsorshipDate.before(maximumDate);`	Esta validación comprueba que el atributo moment de la sponsorship no se pase ni del límite inferior ni del superior; la única opción que no se puede probar es que la fecha incumpla ambos límites a la vez, por lo tanto, la validación está cubierta.
`super.state(durationStartTime.after(moment) && durationStartTime.after(minimumDate) && durationStartTime.before(maximumDate), "durationStartTime", "sponsor.sponsorship.form.error.durationStartTime");`	Esta validación comprueba que el atributo start time de la sponsorship no se pase ni del límite inferior ni del superior y que esté antes del atributo moment ; la única opción que no se puede probar es que la fecha esté antes del moment y por encima del límite superior, por lo tanto, la validación está cubierta.

`if (durationStartTime != null && durationEndTime != null)`	En este if nunca se va a dar el caso de que ambos estén vacíos, y el resto han sido correctamente probados.
`super.state(!object.getProject().isDraftMode(), "project", "sponsor.sponsorship.form.error.project-not-published");`	Este código comprueba una validación realizada para 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.

2.1.1.4) Update

- update.safe: para hacer esta prueba, he probado a crear un sponsorship 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 sponsorships con los casos positivos yendo atributo por atributo, estos casos positivos son los extremos tanto superiores como inferiores y también valores intermedios; respetando los rangos de cada atributo.

- update.hack: para hacer esta prueba he iniciado la aplicación y me he registrado con el usuario sponsor1, tras iniciada la sesión me he metido en un sponsorship y dándole a inspeccionar en la pestaña, he cambiado ese valor de id por 307 (sponsorship perteneciente al sponsor2) y le di al botón de update, tras ello me a metí en otra sponsorship y cambié la id a 275 (sponsorship publicada de sponsor1) 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 (sponsorship) y las hijas (invoice), ya que la fecha creación de las entidades hijas debe ser siempre superior a la de la entidad padre.

- class SponsorSponsorshipUpdateService:

```
1.  @Service
2.  public class SponsorSponsorshipUpdateService extends
    AbstractService<Sponsor, Sponsorship> {
3.
4.      // Internal state
    ----------------------------------------------------------
5.
6.      @Autowired
7.      private SponsorSponsorshipRepository    repository;
8.
9.      @Autowired
10.     private SystemConfigurationRepository
    systemConfigurationRepository;
11.
```

```
12.     // AbstractService interface
    ----------------------------------------------
13.
14.
15.     @Override
16.     public void authorise() {
17.         boolean status;
18.         int id;
19.         int sponsorId;
20.         Sponsorship sponsorship;
21.
22.         id = super.getRequest().getData("id", int.class);
23.         sponsorship = this.repository.findOneSponsorshipById(id);
24.
25.         sponsorId =
    super.getRequest().getPrincipal().getActiveRoleId();
26.
27.         status = sponsorId == sponsorship.getSponsor().getId() &&
    sponsorship.isDraftMode();
28.
29.         super.getResponse().setAuthorised(status);
30.     }
31.
32.     @Override
33.     public void load() {
34.         Sponsorship object;
35.
36.         int sponsorId;
37.
38.         sponsorId = super.getRequest().getData("id", int.class);
39.
40.         object = this.repository.findOneSponsorshipById(sponsorId);
41.
42.         super.getBuffer().addData(object);
43.     }
44.
45.     @Override
46.     public void bind(final Sponsorship object) {
47.         assert object != null;
48.
49.         super.bind(object, "code", "moment", "durationStartTime",
    "durationEndTime", "amount", "type", "email", "link", "project");
50.     }
51.
52.     @Override
53.     public void validate(final Sponsorship object) {
54.         assert object != null;
55.
56.         if (!super.getBuffer().getErrors().hasErrors("code")) {
57.
58.             Sponsorship projectSameCode =
    this.repository.findOneSponsorshipByCode(object.getCode());
59.
60.             if (projectSameCode != null)
61.                 super.state(projectSameCode.getId() == object.getId(),
    "code", "sponsor.sponsorship.form.error.code");
62.         }
63.
64.         if (!super.getBuffer().getErrors().hasErrors("moment")) {
65.
66.             Date sponsorshipDate = object.getMoment();
```

```
67.             Date minimumDate = MomentHelper.parse("1969-12-31 0:00",
    "yyyy-MM-dd HH:mm");
68.             Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
    "yyyy-MM-dd HH:mm");
69.
70.             if (sponsorshipDate != null) {
71.                 Boolean isAfter = sponsorshipDate.after(minimumDate)
    && sponsorshipDate.before(maximumDate);
72.                 super.state(isAfter, "moment",
    "sponsor.sponsorship.form.error.moment");
73.             }
74.         }
75.
76.         if (!super.getBuffer().getErrors().hasErrors("moment")) {
77.             Invoice earliestInvoice;
78.             Boolean validMoment;
79.             Date moment = object.getMoment();
80.
81.             earliestInvoice =
    this.repository.findInvoiceWithEarliestDateBySponsorshipId(object.getI
    d()).stream().findFirst().orElse(null);
82.             //System.out.println(earliestInvoice);
83.
84.             if (earliestInvoice != null) {
85.                 //System.out.println(earliestInvoice);
86.                 validMoment =
    moment.before(earliestInvoice.getRegistrationTime());
87.                 //System.out.println(validMoment);
88.                 super.state(validMoment, "moment",
    "sponsor.sponsorship.form.error.creation-moment");
89.             }
90.         }
91.
92.         if
    (!super.getBuffer().getErrors().hasErrors("durationStartTime")) {
93.             Date durationStartTime;
94.             Date moment;
95.             durationStartTime = object.getDurationStartTime();
96.             moment = object.getMoment();
97.             Date minimumDate = MomentHelper.parse("1969-12-31 0:00",
    "yyyy-MM-dd HH:mm");
98.             Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
    "yyyy-MM-dd HH:mm");
99.
100.                if (moment != null)
101.                    super.state(durationStartTime.after(moment) &&
    durationStartTime.after(minimumDate) &&
    durationStartTime.before(maximumDate), "durationStartTime",
    "sponsor.sponsorship.form.error.durationStartTime");
102.            }
103.
104.        if
    (!super.getBuffer().getErrors().hasErrors("durationEndTime")) {
105.                Date durationStartTime;
106.                Date durationEndTime;
107.
108.                durationStartTime = object.getDurationStartTime();
109.                durationEndTime = object.getDurationEndTime();
110.                Date maximumDate = MomentHelper.parse("2200-12-31
    23:59", "yyyy-MM-dd HH:mm");
111.
```

```
112.             if (durationStartTime != null && durationEndTime != null)
113.                 super.state(MomentHelper.isLongEnough(durationStartTime, durationEndTime, 1, ChronoUnit.MONTHS) && durationEndTime.after(durationStartTime) && durationEndTime.before(maximumDate), "durationEndTime", "sponsor.sponsorship.form.error.durationEndTime");
114.             }
115. 
116.             if (!super.getBuffer().getErrors().hasErrors("amount") && this.systemConfigurationRepository.existsCurrency(object.getAmount().getCurrency()))
117.                 super.state(object.getAmount().getAmount() >= 0 && object.getAmount().getAmount() <= 1000000, "amount", "sponsor.sponsorship.form.error.amount");
118. 
119.             if (!super.getBuffer().getErrors().hasErrors("amount")) {
120.                 String symbol = object.getAmount().getCurrency();
121.                 boolean existsCurrency = this.systemConfigurationRepository.existsCurrency(symbol);
122.                 super.state(existsCurrency, "amount", "sponsor.sponsorship.form.error.acceptedCurrency");
123.             }
124. 
125.             if (!super.getBuffer().getErrors().hasErrors("project"))
126.                 super.state(!object.getProject().isDraftMode(), "project", "sponsor.sponsorship.form.error.project-not-published");
127. 
128.         }
129. 
130.         @Override
131.         public void perform(final Sponsorship object) {
132.             assert object != null;
133. 
134.             this.repository.save(object);
135.         }
136. 
137.         @Override
138.         public void unbind(final Sponsorship object) {
139.             assert object != null;
140. 
141.             SelectChoices choices;
142. 
143.             Collection<Project> projects = this.repository.findProjects();
144.             SelectChoices choices2;
145.             Dataset dataset;
146.             String projectCode;
147.             int sponsorId;
148. 
149.             sponsorId = super.getRequest().getData("id", int.class);
150. 
151.             Sponsorship s = this.repository.findOneSponsorshipById(sponsorId);
152. 
153.             projectCode = s.getProject() != null ? s.getProject().getCode() : null;
154. 
155.             choices = SelectChoices.from(SponsorshipType.class,
156.             
```

```
object.getType());
157.            choices2 = SelectChoices.from(projects, "code",
s.getProject());
158.
159.            dataset = super.unbind(object, "code", "moment",
"durationStartTime", "durationEndTime", "amount", "type", "email",
"link", "project", "draftMode");
160.            dataset.put("types", choices);
161.            dataset.put("projects", choices2);
162.            dataset.put("project", projectCode);
163.
164.            super.getResponse().addData(dataset);
165.        }
166.
167.    }
```

Línea amarilla (no completamente probada)	Explicación
`if (sponsorshipDate != null) { Boolean isAfter = sponsorshipDate.after(minimumDate) && sponsorshipDate.before(maximumDate);`	Esta validación comprueba que el atributo moment de la sponsorship no se pase ni del límite inferior ni del superior; la única opción que no se puede probar es que la fecha incumpla ambos límites a la vez, por lo tanto, la validación está cubierta.
`super.state(durationStartTime.after(moment) && durationStartTime.after(minimumDate) && durationStartTime.before(maximumDate), "durationStartTime", "sponsor.sponsorship.form.error.durationStartTime");`	Esta validación comprueba que el atributo start time de la sponsorship no se pase ni del límite inferior ni del superior y que esté antes del atributo moment ; la única opción que no se puede probar es que la fecha esté antes del moment y por encima del límite superior, por lo tanto, la validación está cubierta.
`if (durationStartTime != null && durationEndTime != null)`	En este if nunca se va a dar el caso de que ambos estén vacíos, y el resto han sido correctamente probados.
`super.state(!object.getProject().isDraftMode(), "project", "sponsor.sponsorship.form.error.project-not-published");`	Este código comprueba una validación realizada para 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.

2.1.1.5) Publish

- publish.safe: para hacer esta prueba, he probado a crear un sponsorship 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 sponsorships con los casos positivos yendo atributo por atributo, estos casos positivos son los extremos tanto superiores como inferiores y también valores intermedios; respetando los rangos de cada atributo.

- publish.hack: para hacer esta prueba he iniciado la aplicación y me he registrado con el usuario sponsor1, tras iniciada la sesión me he metido en un sponsorship y dándole a inspeccionar en la pestaña, he cambiado ese valor de id por 307 (sponsorship perteneciente al sponsor2) y le di al botón de publish, tras ello me a metí en otra sponsorship y cambié la id a 275 (sponsorship publicada de sponsor1) 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 (sponsorship) y las hijas (invoice), ya que la fecha creación de las entidades hijas debe ser siempre superior a la de la entidad padre.

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

- class SponsorSponsorshipPublishService:

```
@Service
public class SponsorSponsorshipPublishService extends
AbstractService<Sponsor, Sponsorship> {

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

    @Autowired
    private SponsorSponsorshipRepository    repository;

    @Autowired
    private SystemConfigurationRepository
systemConfigurationRepository;

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


    @Override
    public void authorise() {
        boolean status;
        int id;
        int sponsorId;
        Sponsorship sponsorship;

        id = super.getRequest().getData("id", int.class);
        sponsorship = this.repository.findOneSponsorshipById(id);

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

        status = sponsorId == sponsorship.getSponsor().getId() &&
sponsorship.isDraftMode();
```

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

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

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

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

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

        super.bind(object, "code", "moment", "durationStartTime",
"durationEndTime", "amount", "type", "email", "link", "project");
    }

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

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

            Sponsorship projectSameCode =
this.repository.findOneSponsorshipByCode(object.getCode());

            if (projectSameCode != null)
                super.state(projectSameCode.getId() == object.getId(),
"code", "sponsor.sponsorship.form.error.code");
        }

        if (!super.getBuffer().getErrors().hasErrors("totalAmount")) {
            Collection<Invoice> invoices =
this.repository.findInvoicesOfASponsorship(object.getId());

            double invoiceTotAmount = invoices.stream().mapToDouble(i
-> this.currencyTransformerUsd(i.getQuantity(),
i.totalAmount().getAmount()).getAmount()).sum();
            if (object.getAmount() != null)
                super.state(invoiceTotAmount ==
this.currencyTransformerUsd(object.getAmount(),
object.getAmount().getAmount()).getAmount(), "*",
"sponsor.sponsorship.form.error.invalidTotalAmount");
        }

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

            Date sponsorshipDate = object.getMoment();
            Date minimumDate = MomentHelper.parse("1969-12-31 0:00",
"yyyy-MM-dd HH:mm");
            Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
"yyyy-MM-dd HH:mm");

            if (sponsorshipDate != null) {
                Boolean isAfter = sponsorshipDate.after(minimumDate)
```

```
&& sponsorshipDate.before(maximumDate);
                super.state(isAfter, "moment",
"sponsor.sponsorship.form.error.moment");
            }
        }

        if (!super.getBuffer().getErrors().hasErrors("moment")) {
            Invoice earliestInvoice;
            Boolean validMoment;
            Date moment = object.getMoment();

            earliestInvoice =
this.repository.findInvoiceWithEarliestDateBySponsorshipId(object.getI
d()).stream().findFirst().orElse(null);
            //System.out.println(earliestInvoice);

            if (earliestInvoice != null) {
                //System.out.println(earliestInvoice);
                validMoment =
moment.before(earliestInvoice.getRegistrationTime());
                //System.out.println(validMoment);
                super.state(validMoment, "moment",
"sponsor.sponsorship.form.error.creation-moment");
            }
        }

        if
(!super.getBuffer().getErrors().hasErrors("durationStartTime")) {
            Date durationStartTime;
            Date moment;
            durationStartTime = object.getDurationStartTime();
            moment = object.getMoment();
            Date minimumDate = MomentHelper.parse("1969-12-31 0:00",
"yyyy-MM-dd HH:mm");
            Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
"yyyy-MM-dd HH:mm");

            if (moment != null)
                super.state(durationStartTime.after(moment) &&
durationStartTime.after(minimumDate) &&
durationStartTime.before(maximumDate), "durationStartTime",
"sponsor.sponsorship.form.error.durationStartTime");
        }

        if
(!super.getBuffer().getErrors().hasErrors("durationEndTime")) {
            Date durationStartTime;
            Date durationEndTime;

            durationStartTime = object.getDurationStartTime();
            durationEndTime = object.getDurationEndTime();
            Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
"yyyy-MM-dd HH:mm");

            if (durationStartTime != null && durationEndTime != null)

super.state(MomentHelper.isLongEnough(durationStartTime,
durationEndTime, 1, ChronoUnit.MONTHS) &&
durationEndTime.after(durationStartTime) &&
durationEndTime.before(maximumDate), "durationEndTime",
                    "sponsor.sponsorship.form.error.durationEndTime");
```

```
        }

        if (!super.getBuffer().getErrors().hasErrors("amount") &&
this.systemConfigurationRepository.existsCurrency(object.getAmount().g
etCurrency()))
            super.state(object.getAmount().getAmount() >= 0 &&
object.getAmount().getAmount() <= 1000000, "amount",
"sponsor.sponsorship.form.error.amount");

        if (!super.getBuffer().getErrors().hasErrors("amount")) {
            String symbol = object.getAmount().getCurrency();
            boolean existsCurrency =
this.systemConfigurationRepository.existsCurrency(symbol);
            super.state(existsCurrency, "amount",
"sponsor.sponsorship.form.error.acceptedCurrency");
        }

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

            Collection<Invoice> unpublishedInvoices;

            unpublishedInvoices =
this.repository.findUnpublishedInvoicesBySponsorshipId(object.getId())
;

            super.state(unpublishedInvoices.isEmpty(), "*",
"sponsor.sponsorship.form.error.unpublished-invoices");
        }

        if (!super.getBuffer().getErrors().hasErrors("project"))
            super.state(!object.getProject().isDraftMode(), "project",
"sponsor.sponsorship.form.error.project-not-published");

    }

    private Money currencyTransformerUsd(final Money currency, final
Double amount) {
        Money res = new Money();
        res.setCurrency("USD");

        if (currency.getCurrency().equals("USD"))
            res.setAmount(amount);

        else if (currency.getCurrency().equals("EUR"))
            res.setAmount(amount * 1.07);

        else
            res.setAmount(amount * 1.25);

        return res;
    }

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

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

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

        SelectChoices choices;

        Collection<Project> projects = this.repository.findProjects();
        SelectChoices choices2;
        Dataset dataset;
        String projectCode;
        int sponsorId;

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

        Sponsorship s =
this.repository.findOneSponsorshipById(sponsorId);

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

        choices = SelectChoices.from(SponsorshipType.class,
object.getType());
        choices2 = SelectChoices.from(projects, "code",
s.getProject());

        dataset = super.unbind(object, "code", "moment",
"durationStartTime", "durationEndTime", "amount", "type", "email",
"link", "project", "draftMode");
        dataset.put("types", choices);
        dataset.put("projects", choices2);
        dataset.put("project", projectCode);

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

}
```

Línea amarilla (no completamente probada)	Explicación
`if (!super.getBuffer().getErrors().hasErrors("totalAmount")) {`	En este if nunca se va a dar el caso de que total amount falle, por lo tanto todo se ha probado correctamente.
`if (sponsorshipDate != null) {` `Boolean isAfter = sponsorshipDate.after(minimumDate) && sponsorshipDate.before(maximumDate);`	Esta validación comprueba que el atributo moment de la sponsorship no se pase ni del límite inferior ni del superior; la única opción que no se puede probar es que la fecha incumpla ambos límites a la vez, por lo tanto, la validación está cubierta.
`super.state(durationStartTime.after(moment) && durationStartTime.after(minimu`	Esta validación comprueba que el atributo start time de la sponsorship no se pase ni del límite inferior ni del superior y que esté

`mDate) && durationStartTime.before(maximumDate), "durationStartTime", "sponsor.sponsorship.form.error.durationStartTime");`	antes del atributo moment ; la única opción que no se puede probar es que la fecha esté antes del moment y por encima del límite superior, por lo tanto, la validación está cubierta.
`- if (durationStartTime != null && durationEndTime != null)`	En este if nunca se va a dar el caso de que ambos estén vacíos, y el resto han sido correctamente probados.
`if (!super.getBuffer().getErrors().hasErrors("unpublishedInvoices")) {`	Este código no debería salir en amarillo ya que se han probado todos los casos posibles (que tenga invoices sin publicar y publicados). Se desconoce el motivo de que salga amarillo.
`- super.state(!object.getProject().isDraftMode(), "project", "sponsor.sponsorship.form.error.project-not-published");`	Este código comprueba una validación realizada para 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.

2.1.1.6) Delete

- delete.safe = para hacer esta prueba,he intentado borrar una sponsorship 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 sponsor1, tras iniciada la sesión me he metido en un sponsorship y dándole a inspeccionar en la pestaña, he cambiado ese valor de id por 307 (sponsorship perteneciente al sponsor2) y le di al botón de delete, tras ello me a metí en otra sponsorship y cambié la id a 275 (sponsorship publicada de sponsor1) 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 los mismos.

- class SponsorSponsorshipDeleteService:

```
1. @Service
2. public class SponsorSponsorshipDeleteService extends
   AbstractService<Sponsor, Sponsorship> {
3.
```

```
4.      // Internal state
    ---------------------------------------------------------
5.
6.      @Autowired
7.      private SponsorSponsorshipRepository repository;
8.
9.      // AbstractService interface
    ------------------------------------------------
10.
11.
12.     @Override
13.     public void authorise() {
14.         boolean status;
15.         int id;
16.         int sponsorId;
17.         Sponsorship sponsorship;
18.
19.         id = super.getRequest().getData("id", int.class);
20.         sponsorship = this.repository.findOneSponsorshipById(id);
21.
22.         sponsorId =
    super.getRequest().getPrincipal().getActiveRoleId();
23.
24.         status = sponsorId == sponsorship.getSponsor().getId() &&
    sponsorship.isDraftMode();
25.
26.         super.getResponse().setAuthorised(status);
27.     }
28.
29.     @Override
30.     public void load() {
31.         Sponsorship object;
32.         int id;
33.
34.         id = super.getRequest().getData("id", int.class);
35.
36.         object = this.repository.findOneSponsorshipById(id);
37.
38.         super.getBuffer().addData(object);
39.     }
40.
41.     @Override
42.     public void bind(final Sponsorship object) {
43.         assert object != null;
44.
45.         super.bind(object, "code", "moment", "durationStartTime",
    "durationEndTime", "amount", "type", "email", "link");
46.     }
47.
48.     @Override
49.     public void validate(final Sponsorship object) {
50.         assert object != null;
51.
```

```
52.         Collection<Invoice> publishedInvoices;
53.
54.         publishedInvoices =
    this.repository.findPublishedInvoicesBySponsorshipId(object.getId());
55.         super.state(publishedInvoices.isEmpty(), "*",
    "sponsor.sponsorship.form.error.published-invoices");
56.     }
57.
58.     @Override
59.     public void perform(final Sponsorship object) {
60.         assert object != null;
61.
62.         Collection<Invoice> relations =
    this.repository.findInvoicesOfASponsorship(object.getId());
63.
64.         this.repository.deleteAll(relations);
65.
66.         this.repository.delete(object);
67.     }
68.
69.     @Override
70.     public void unbind(final Sponsorship object) {
71.         assert object != null;
72.
73.         SelectChoices choices;
74.
75.         Collection<Project> projects = this.repository.findProjects();
76.         SelectChoices choices2;
77.         Dataset dataset;
78.
79.         choices = SelectChoices.from(SponsorshipType.class,
    object.getType());
80.         choices2 = SelectChoices.from(projects, "code", (Project)
    projects.toArray()[0]);
81.
82.         dataset = super.unbind(object, "code", "moment",
    "durationStartTime", "durationEndTime", "amount", "type", "email",
    "link", "project", "draftMode");
83.         dataset.put("types", choices);
84.         dataset.put("projects", choices2);
85.
86.         super.getResponse().addData(dataset);
87.     }
88.
89. }
```

Línea amarilla (no completamente probada)	Explicación
Intencionalmente en blanco	Intencionalmente en blanco

2.1.2) Invoice

2.1.2.1) List

- list.safe: para hacer esta prueba, me he metido en la lista de sponsorships de los usuarios de mis roles (sponsor 1 y 2) y he listado los invoices de cada uno.

- 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 las invoices de una sponsorship del sponsor1: http://localhost:8082/acme-sf-d04/sponsor/invoice/list?sponsorshipId=274 dandome un error 500 de “access is not authorised”. Y también me he registrado como sponsor2 intentando acceder a ese invoice 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 los mismos.

- class SponsorInvoiceListService:

```
1. @Service
2. public class SponsorInvoiceListService extends
   AbstractService<Sponsor, Invoice> {
3.
4.     // Internal state
   ----------------------------------------------------------
5.
6.     @Autowired
7.     private SponsorInvoiceRepository repository;
8.
9.     // AbstractService interface
   ----------------------------------------------
10.
11.
12.    @Override
13.    public void authorise() {
14.        boolean status;
15.        int sponsorshipId;
16.        int sponsorId;
17.        Sponsorship sponsorship;
18.
19.        sponsorshipId = super.getRequest().getData("sponsorshipId",
   int.class);
20.        sponsorship =
   this.repository.findOneSponsorshipById(sponsorshipId);
21.
22.        sponsorId =
   super.getRequest().getPrincipal().getActiveRoleId();
23.
24.        status = sponsorId == sponsorship.getSponsor().getId();
25.
26.        super.getResponse().setAuthorised(status);
```

```
27.     }
28.
29.     @Override
30.     public void load() {
31.         Collection<Invoice> objects;
32.         int sponsorshipId;
33.
34.         sponsorshipId = super.getRequest().getData("sponsorshipId",
    int.class);
35.
36.         objects =
    this.repository.findAllInvoicesBySponsorshipId(sponsorshipId);
37.
38.         super.getBuffer().addData(objects);
39.     }
40.
41.     @Override
42.     public void unbind(final Invoice object) {
43.         assert object != null;
44.
45.         Dataset dataset;
46.
47.         dataset = super.unbind(object, "code", "registrationTime",
    "dueDate", "quantity", "tax", "link", "draftMode");
48.         dataset.put("totalAmount", object.totalAmount());
49.         super.getResponse().addData(dataset);
50.     }
51.
52.     @Override
53.     public void unbind(final Collection<Invoice> objects) {
54.         assert objects != null;
55.
56.         int sponsorshipId;
57.         Sponsorship sponsorship;
58.
59.         sponsorshipId = super.getRequest().getData("sponsorshipId",
    int.classç);
60.         sponsorship =
    this.repository.findOneSponsorshipById(sponsorshipId);
61.
62.         super.getResponse().addGlobal("sponsorshipId", sponsorshipId);
63.         super.getResponse().addGlobal("canCreate",
    sponsorship.isDraftMode());
64.     }
65.
66. }
```

Línea amarilla (no completamente probada)	Explicación
Intencionalmente en blanco	Intencionalmente en blanco

2.1.2.2) Show

- show.safe: para hacer esta prueba, me he metido en las listas de las sponsorships de los usuarios de mis roles (sponsor 1 y 2), he listado las invoices de cada sponsorship, por último me he metido en todos y cada uno de las invoices 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 invoice del sponsor1: http://localhost:8082/acme-sf-d04/sponsor/invoice/show?id=317 dandome un error 500 de “access is not authorised”. Y también me he registrado como sponsor2 intentando acceder a ese invoice 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 los mismos.

- class SponsorInvoiceShowService:

```
1.  @Service
2.  public class SponsorInvoiceShowService extends
    AbstractService<Sponsor, Invoice> {
3.
4.      // Internal state
    ----------------------------------------------------------
5.
6.      @Autowired
7.      private SponsorInvoiceRepository repository;
8.
9.      // AbstractService interface
    ----------------------------------------------
10.
11.
12.     @Override
13.     public void authorise() {
14.         boolean status;
15.         int id;
16.         int sponsorId;
17.         Invoice invoice;
18.
19.         id = super.getRequest().getData("id", int.class);
20.         invoice = this.repository.findOneInvoiceById(id);
21.
22.         sponsorId =
    super.getRequest().getPrincipal().getActiveRoleId();
23.
24.         status = sponsorId == invoice.getSponsor().getId();
25.
26.         super.getResponse().setAuthorised(status);
27.     }
28.
29.     @Override
30.     public void load() {
31.         Invoice object;
32.         int id;
33.
34.         id = super.getRequest().getData("id", int.class);
35.         object = this.repository.findOneInvoiceById(id);
```

```
36.
37.         super.getBuffer().addData(object);
38.     }
39.
40.     @Override
41.     public void unbind(final Invoice object) {
42.         assert object != null;
43.
44.         Dataset dataset;
45.
46.         dataset = super.unbind(object, "code", "registrationTime",
    "dueDate", "quantity", "tax", "link", "draftMode");
47.         dataset.put("totalAmount", object.totalAmount());
48.
49.         super.getResponse().addData(dataset);
50.     }
51.
52. }
```

Línea amarilla (no completamente probada)	Explicación
Intencionalmente en blanco	Intencionalmente en blanco

2.1.2.3) Create

- create.safe: para hacer esta prueba, he probado a crear una invoice 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 invoice con los casos positivos yendo atributo por atributo, estos casos positivos son los extremos tanto superiores como inferiores y también valores intermedios; 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 una invoice del sponsor1: http://localhost:8082/acme-sf-d04/sponsor/invoice/create?sponsorshipId=274 dandome un error 500 de “access is not authorised”. Y también me he registrado como sponsor2 intentando acceder a esa invoice 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 los mismos.

- class SponsorInvoiceCreateService:

```
@Service
public class SponsorInvoiceCreateService extends
AbstractService<Sponsor, Invoice> {

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

    @Autowired
    private SponsorInvoiceRepository        repository;

    @Autowired
    private SystemConfigurationRepository
systemConfigurationRepository;

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


    @Override
    public void authorise() {
        boolean status;
        int sponsorshipId;
        int sponsorId;
        Sponsorship sponsorship;

        sponsorshipId = super.getRequest().getData("sponsorshipId",
int.class);
        sponsorship =
this.repository.findOneSponsorshipById(sponsorshipId);

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

        status = sponsorId == sponsorship.getSponsor().getId();

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

    @Override
    public void load() {
        Invoice object;
        Integer sponsorshipId;
        Sponsorship sponsorship;

        object = new Invoice();
        Integer sponsorId =
super.getRequest().getPrincipal().getActiveRoleId();
        Sponsor sponsor =
this.repository.findOneSponsorById(sponsorId);

        sponsorshipId = super.getRequest().getData("sponsorshipId",
int.class);
        sponsorship =
this.repository.findOneSponsorshipById(sponsorshipId);

        object.setSponsor(sponsor);
        object.setSponsorship(sponsorship);
        object.setDraftMode(true);
```

```
48.
49.         super.getBuffer().addData(object);
50.
51.     }
52.
53.     @Override
54.     public void bind(final Invoice object) {
55.         assert object != null;
56.
57.         super.bind(object, "code", "registrationTime", "dueDate",
    "quantity", "tax", "link");
58.     }
59.
60. @Override
61.     public void validate(final Invoice object) {
62.         assert object != null;
63.
64.         if (!super.getBuffer().getErrors().hasErrors("code")) {
65.
66.             Invoice projectSameCode =
    this.repository.findOneInvoiceByCode(object.getCode());
67.
68.             super.state(projectSameCode == null, "code",
    "sponsor.invoice.form.error.code");
69.         }
70.
71.         if
    (!super.getBuffer().getErrors().hasErrors("registrationTime")) {
72.
73.
74.             Date registrationTime = object.getRegistrationTime();
75.             Date minimumDate = MomentHelper.parse("1969-12-31 23:59",
    "yyyy-MM-dd HH:mm");
76.             Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
    "yyyy-MM-dd HH:mm");
77.
78.             if (registrationTime != null) {
79.                 Boolean isAfter = registrationTime.after(minimumDate)
    && registrationTime.before(maximumDate);
80.                 super.state(isAfter, "registrationTime",
    "sponsor.invoice.form.error.registration-time");
81.             }
82.         }
83.
84.         if
    (!super.getBuffer().getErrors().hasErrors("registrationTime")) {
85.             Date registrationTime;
86.             Date moment;
87.
88.             registrationTime = object.getRegistrationTime();
89.             moment = object.getSponsorship().getMoment();
90.
91.             if (registrationTime != null)
92.                 super.state(registrationTime.after(moment),
    "registrationTime",
    "sponsor.invoice.form.error.registration-time-bis");
93.         }
94.
95.         if (!super.getBuffer().getErrors().hasErrors("dueDate")) {
96.             Date registrationTime;
97.             Date dueDate;
```

```
98.
99.             registrationTime = object.getRegistrationTime();
100.                dueDate = object.getDueDate();
101.                Date minimumDate = MomentHelper.parse("1969-12-31
    23:59", "yyyy-MM-dd HH:mm");
102.                Date maximumDate = MomentHelper.parse("2200-12-31
    23:59", "yyyy-MM-dd HH:mm");
103.
104.                if (registrationTime != null && dueDate != null)
105.
    super.state(MomentHelper.isLongEnough(registrationTime, dueDate, 1,
    ChronoUnit.MONTHS) && dueDate.after(registrationTime) &&
    dueDate.after(minimumDate) && dueDate.before(maximumDate), "dueDate",
    "sponsor.invoice.form.error.dueDate");
106.            }
107.
108.            if (!super.getBuffer().getErrors().hasErrors("quantity") &&
    this.systemConfigurationRepository.existsCurrency(object.getQuantity()
    .getCurrency()))
109.                super.state(object.getQuantity().getAmount() >= 0 &&
    object.getQuantity().getAmount() <= 1000000, "quantity",
    "sponsor.invoice.form.error.quantity");
110.
111.            if (!super.getBuffer().getErrors().hasErrors("quantity")) {
112.                String symbol = object.getQuantity().getCurrency();
113.                boolean existsCurrency =
    this.systemConfigurationRepository.existsCurrency(symbol);
114.                super.state(existsCurrency, "quantity",
    "sponsor.invoice.form.error.acceptedCurrency");
115.            }
116.
117.            if
    (!super.getBuffer().getErrors().hasErrors("publishedSponsorship")) {
118.                Integer sponsorshipId;
119.                Sponsorship sponsorship;
120.
121.                sponsorshipId =
    super.getRequest().getData("sponsorshipId", int.class);
122.                sponsorship =
    this.repository.findOneSponsorshipById(sponsorshipId);
123.
124.                super.state(sponsorship.isDraftMode(), "*",
    "sponsor.invoice.form.error.published-sponsorship");
125.            }
126.
127.        }
128.
129.        @Override
130.        public void perform(final Invoice object) {
131.            assert object != null;
132.
133.            if (object.getTax() == null)
134.                object.setTax(0.0);
135.
136.            this.repository.save(object);
137.        }
138.
139.        @Override
140.        public void unbind(final Invoice object) {
141.            assert object != null;
142.
```

```
143.            Dataset dataset;
144.
145.            dataset = super.unbind(object, "code", "registrationTime",
    "dueDate", "quantity", "tax", "link", "draftMode");
146.            dataset.put("sponsorshipId",
    super.getRequest().getData("sponsorshipId", int.class));
147.
148.            super.getResponse().addData(dataset);
149.        }
150.
151.    }
152.
```

Línea amarilla (no completamente probada)	Explicación
`if (registrationTime != null) { Boolean isAfter = registrationTime.after(minimumDate) && registrationTime.before(maximumDate) ;`	Esta validación comprueba que el atributo registration time de la invoice no se pase ni del límite inferior ni del superior; la única opción que no se puede probar es que la fecha incumpla ambos límites a la vez, por lo tanto, la validación está cubierta.
`if (registrationTime != null)`	Este código no debería salir en amarillo ya que se han probado tanto a poner una registration time vacía como con valores. Se desconoce el motivo de que salga amarillo.

`if (registrationTime != null && dueDate != null)`
 	
`super.state(MomentHelper.isLongEnough(registrationTime, dueDate, 1, ChronoUnit.MONTHS) && dueDate.after(registrationTime) && dueDate.after(minimumDate) && dueDate.before(maximumDate), "dueDate", "sponsor.invoice.form.error.dueDate" );`
	Esta validación comprueba que el atributo registration time y el due date de la invoice no se pasen ni del límite inferior ni del superior y que el due date esté después del registration time. También que entre las dos fechas haya mínimo un mes de diferencia ; la única opción que no se puede probar es que la due date se pase del límite superior y que esté a la vez antes de la fecha de registro, por lo tanto, la validación está cubierta.

`if (!super.getBuffer().getErrors().hasErrors("publishedSponsorship")) {`	Este código supone una validación realizada con el fin de evitar un posible hackeo en el campo de sponsorship, el usuario común nunca va a poder crear un audit record en un code audit publicado usando el sistema de forma correcta.

2.1.2.4) Update

- update.safe: para hacer esta prueba, he probado a actualizar una invoice 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 diversas entidades invoice con los casos positivos yendo atributo por atributo, estos casos positivos son los extremos tanto superiores como inferiores y también valores intermedios; respetando los rangos de cada atributo.

- update.hack: para hacer esta prueba he iniciado la aplicación y me he registrado con el usuario sponsor1, tras iniciada la sesión me he metido en una invoice y dándole a inspeccionar en la pestaña, he cambiado ese valor de id por 343 (invoice perteneciente al sponsor2) y le di al botón de update, tras ello me volví a meter en el mismo invoice y cambié la id a 342 (audit record publicado de sponsor1) y volviendo a pulsar en el botón de update. 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 los mismos.

- class SponsorInvoiceUpdateService:

```
@Service
public class SponsorInvoiceUpdateService extends AbstractService<Sponsor, Invoice> {

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

    @Autowired
    private SponsorInvoiceRepository        repository;

    @Autowired
    private SystemConfigurationRepository systemConfigurationRepository;

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


    @Override
    public void authorise() {
        boolean status;
        int id;
        int sponsorId;
        Invoice invoice;

        id = super.getRequest().getData("id", int.class);
        invoice = this.repository.findOneInvoiceById(id);

```

```
24.         sponsorId =
    super.getRequest().getPrincipal().getActiveRoleId();
25.
26.         status = sponsorId == invoice.getSponsor().getId() &&
    invoice.isDraftMode();
27.
28.         super.getResponse().setAuthorised(status);
29.     }
30.
31.     @Override
32.     public void load() {
33.         Invoice object;
34.         int id;
35.
36.         id = super.getRequest().getData("id", int.class);
37.
38.         object = this.repository.findOneInvoiceById(id);
39.
40.         super.getBuffer().addData(object);
41.     }
42.
43.     @Override
44.     public void bind(final Invoice object) {
45.         assert object != null;
46.
47.         Integer sponsorId =
    super.getRequest().getPrincipal().getActiveRoleId();
48.         Sponsor sponsor =
    this.repository.findOneSponsorById(sponsorId);
49.         object.setSponsor(sponsor);
50.         super.bind(object, "code", "registrationTime", "dueDate",
    "quantity", "tax", "link");
51.     }
52.
53.     @Override
54.     public void validate(final Invoice object) {
55.         assert object != null;
56.
57.         if (!super.getBuffer().getErrors().hasErrors("code")) {
58.
59.             Invoice projectSameCode =
    this.repository.findOneInvoiceByCode(object.getCode());
60.
61.             if (projectSameCode != null)
62.                 super.state(projectSameCode.getId() == object.getId(),
    "code", "sponsor.invoice.form.error.code");
63.         }
64.
65.         if
    (!super.getBuffer().getErrors().hasErrors("registrationTime")) {
66.
67.             Date registrationTime = object.getRegistrationTime();
```

```
68.             Date minimumDate = MomentHelper.parse("1969-12-31 23:59",
    "yyyy-MM-dd HH:mm");
69.             Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
    "yyyy-MM-dd HH:mm");
70.
71.             if (registrationTime != null) {
72.                 Boolean isAfter = registrationTime.after(minimumDate)
    && registrationTime.before(maximumDate);
73.                 super.state(isAfter, "registrationTime",
    "sponsor.invoice.form.error.registration-time");
74.             }
75.         }
76.
77.         if
    (!super.getBuffer().getErrors().hasErrors("registrationTime")) {
78.             Date registrationTime;
79.             Date moment;
80.
81.             registrationTime = object.getRegistrationTime();
82.             moment = object.getSponsorship().getMoment();
83.
84.             if (registrationTime != null)
85.                 super.state(registrationTime.after(moment),
    "registrationTime",
    "sponsor.invoice.form.error.registration-time-bis");
86.         }
87.
88.         if (!super.getBuffer().getErrors().hasErrors("dueDate")) {
89.             Date registrationTime;
90.             Date dueDate;
91.
92.             registrationTime = object.getRegistrationTime();
93.             dueDate = object.getDueDate();
94.             Date minimumDate = MomentHelper.parse("1969-12-31 23:59",
    "yyyy-MM-dd HH:mm");
95.             Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
    "yyyy-MM-dd HH:mm");
96.
97.             if (registrationTime != null && dueDate != null)
98.
    super.state(MomentHelper.isLongEnough(registrationTime, dueDate, 1,
    ChronoUnit.MONTHS) && dueDate.after(registrationTime) &&
    dueDate.after(minimumDate) && dueDate.before(maximumDate), "dueDate",
    "sponsor.invoice.form.error.dueDate");
99.         }
100.
101.            if (!super.getBuffer().getErrors().hasErrors("quantity") &&
    this.systemConfigurationRepository.existsCurrency(object.getQuantity()
    .getCurrency()))
102.                super.state(object.getQuantity().getAmount() >= 0 &&
    object.getQuantity().getAmount() <= 1000000, "quantity",
    "sponsor.invoice.form.error.quantity");
103.
```

```
104.         if (!super.getBuffer().getErrors().hasErrors("quantity")) {
105.             String symbol = object.getQuantity().getCurrency();
106.             boolean existsCurrency =
    this.systemConfigurationRepository.existsCurrency(symbol);
107.             super.state(existsCurrency, "quantity",
    "sponsor.invoice.form.error.acceptedCurrency");
108.         }
109.
110.         if
    (!super.getBuffer().getErrors().hasErrors("publishedSponsorship"))
111.             super.state(object.getSponsorship().isDraftMode(), "*",
    "sponsor.invoice.form.error.published-sponsorship");
112.
113.     }
114.
115.     @Override
116.     public void perform(final Invoice object) {
117.         assert object != null;
118.
119.         if (object.getTax() == null)
120.             object.setTax(0.0);
121.
122.         this.repository.save(object);
123.     }
124.
125.     @Override
126.     public void unbind(final Invoice object) {
127.         assert object != null;
128.
129.         Dataset dataset;
130.
131.         dataset = super.unbind(object, "code", "registrationTime",
    "dueDate", "quantity", "tax", "link", "draftMode");
132.         dataset.put("totalAmount", object.totalAmount());
133.
134.         super.getResponse().addData(dataset);
135.     }
136.
137. }
```

Línea amarilla (no completamente probada)	Explicación
`if (registrationTime != null) { Boolean isAfter = registrationTime.after(minimumDate) && registrationTime.before(maximumDate) ;`	Esta validación comprueba que el atributo registration time de la invoice no se pase ni del límite inferior ni del superior; la única opción que no se puede probar es que la fecha incumpla ambos límites a la vez, por lo tanto, la validación está cubierta.
`if (registrationTime != null)`	Este código no debería salir en amarillo ya que se han probado tanto a poner una registration time vacía como con valores. Se desconoce el motivo de que salga

	amarillo.

`if (registrationTime != null && dueDate != null)`
 	
`super.state(MomentHelper.isLongEnough(registrationTime, dueDate, 1, ChronoUnit.MONTHS) && dueDate.after(registrationTime) && dueDate.after(minimumDate) && dueDate.before(maximumDate), "dueDate", "sponsor.invoice.form.error.dueDate");`
	Esta validación comprueba que el atributo registration time y el due date de la invoice no se pasen ni del límite inferior ni del superior y que el due date esté después del registration time. También que entre las dos fechas haya mínimo un mes de diferencia ; la única opción que no se puede probar es que la due date se pase del límite superior y que esté a la vez antes de la fecha de registro, por lo tanto, la validación está cubierta.

`if (!super.getBuffer().getErrors().hasErrors("publishedSponsorship")) {`	Este código supone una validación realizada con el fin de evitar un posible hackeo en el campo de sponsorship, el usuario común nunca va a poder crear un audit record en un code audit publicado usando el sistema de forma correcta.

2.1.2.5) Publish

- publish.safe: para hacer esta prueba, he probado a publicar una invoice 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 diversas entidades invoice con los casos positivos yendo atributo por atributo, estos casos positivos son los extremos tanto superiores como inferiores y también valores intermedios; respetando los rangos de cada atributo.

- publish.hack: para hacer esta prueba he iniciado la aplicación y me he registrado con el usuario sponsor1, tras iniciada la sesión me he metido en una invoice y dándole a inspeccionar en la pestaña, he cambiado ese valor de id por 343 (invoice perteneciente al sponsor2) y le di al botón de publish, tras ello me volví a meter en el mismo invoice y cambié la id a 342 (audit record publicado de sponsor1) y volviendo a pulsar en el botón de publish. 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 los mismos.

- class SponsorInvoicePublishService:

```
@Service
public class SponsorInvoicePublishService extends AbstractService<Sponsor, Invoice> {

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

```
4.
5.      @Autowired
6.      private SponsorInvoiceRepository        repository;
7.
8.      @Autowired
9.      private SystemConfigurationRepository
   systemConfigurationRepository;
10.
11.     // AbstractService interface
   ----------------------------------------------
12.
13.
14.     @Override
15.     public void authorise() {
16.         boolean status;
17.         int id;
18.         int sponsorId;
19.         Invoice invoice;
20.
21.         id = super.getRequest().getData("id", int.class);
22.         invoice = this.repository.findOneInvoiceById(id);
23.
24.         sponsorId =
   super.getRequest().getPrincipal().getActiveRoleId();
25.
26.         status = sponsorId == invoice.getSponsor().getId() &&
   invoice.isDraftMode();
27.
28.         super.getResponse().setAuthorised(status);
29.     }
30.
31.     @Override
32.     public void load() {
33.         Invoice object;
34.         int id;
35.
36.         id = super.getRequest().getData("id", int.class);
37.         object = this.repository.findOneInvoiceById(id);
38.
39.         super.getBuffer().addData(object);
40.     }
41.
42.     @Override
43.     public void bind(final Invoice object) {
44.         assert object != null;
45.
46.         super.bind(object, "code", "registrationTime", "dueDate",
   "quantity", "tax", "link");
47.     }
48.
49.     @Override
50.     public void validate(final Invoice object) {
51.         assert object != null;
52.
53.         if (!super.getBuffer().getErrors().hasErrors("code")) {
54.
55.             Invoice projectSameCode =
   this.repository.findOneInvoiceByCode(object.getCode());
56.
57.             if (projectSameCode != null)
58.                 super.state(projectSameCode.getId() == object.getId(),
```

```
"code", "sponsor.invoice.form.error.code");
59.         }
60.
61.         if
(!super.getBuffer().getErrors().hasErrors("registrationTime")) {
62.
63.             Date registrationTime = object.getRegistrationTime();
64.             Date minimumDate = MomentHelper.parse("1969-12-31 23:59",
"yyyy-MM-dd HH:mm");
65.             Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
"yyyy-MM-dd HH:mm");
66.
67.             if (registrationTime != null) {
68.                 Boolean isAfter = registrationTime.after(minimumDate)
&& registrationTime.before(maximumDate);
69.                 super.state(isAfter, "registrationTime",
"sponsor.invoice.form.error.registration-time");
70.             }
71.         }
72.
73.         if
(!super.getBuffer().getErrors().hasErrors("registrationTime")) {
74.             Date registrationTime;
75.             Date moment;
76.
77.             registrationTime = object.getRegistrationTime();
78.             moment = object.getSponsorship().getMoment();
79.
80.             if (registrationTime != null)
81.                 super.state(registrationTime.after(moment),
"registrationTime",
"sponsor.invoice.form.error.registration-time-bis");
82.         }
83.
84.         if (!super.getBuffer().getErrors().hasErrors("dueDate")) {
85.             Date registrationTime;
86.             Date dueDate;
87.
88.             registrationTime = object.getRegistrationTime();
89.             dueDate = object.getDueDate();
90.             Date minimumDate = MomentHelper.parse("1969-12-31 23:59",
"yyyy-MM-dd HH:mm");
91.             Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
"yyyy-MM-dd HH:mm");
92.
93.             if (registrationTime != null && dueDate != null)
94.
super.state(MomentHelper.isLongEnough(registrationTime, dueDate, 1,
ChronoUnit.MONTHS) && dueDate.after(registrationTime) &&
dueDate.after(minimumDate) && dueDate.before(maximumDate), "dueDate",
"sponsor.invoice.form.error.dueDate");
95.         }
96.
97.         if (!super.getBuffer().getErrors().hasErrors("quantity") &&
this.systemConfigurationRepository.existsCurrency(object.getQuantity()
.getCurrency()))
98.             super.state(object.getQuantity().getAmount() >= 0 &&
object.getQuantity().getAmount() <= 1000000, "quantity",
"sponsor.invoice.form.error.quantity");
99.
100.        if (!super.getBuffer().getErrors().hasErrors("quantity")) {
```

```
101.                String symbol = object.getQuantity().getCurrency();
102.                boolean existsCurrency =
    this.systemConfigurationRepository.existsCurrency(symbol);
103.                super.state(existsCurrency, "quantity",
    "sponsor.invoice.form.error.acceptedCurrency");
104.            }
105.
106.        }
107.
108.        @Override
109.        public void perform(final Invoice object) {
110.            assert object != null;
111.
112.            if (object.getTax() == null)
113.                object.setTax(0.0);
114.
115.            object.setDraftMode(false);
116.            this.repository.save(object);
117.        }
118.
119.        @Override
120.        public void unbind(final Invoice object) {
121.            assert object != null;
122.
123.            Dataset dataset;
124.
125.            dataset = super.unbind(object, "code", "registrationTime",
    "dueDate", "quantity", "tax", "link", "draftMode");
126.            dataset.put("totalAmount", object.totalAmount());
127.
128.            super.getResponse().addData(dataset);
129.        }
130.
131.    }
```

Línea amarilla (no completamente probada)	Explicación
`if (registrationTime != null) { Boolean isAfter = registrationTime.after(minimumDate) && registrationTime.before(maximumDate) ;`	Esta validación comprueba que el atributo registration time de la invoice no se pase ni del límite inferior ni del superior; la única opción que no se puede probar es que la fecha incumpla ambos límites a la vez, por lo tanto, la validación está cubierta.
`if (registrationTime != null)`	Este código no debería salir en amarillo ya que se han probado tanto a poner una registration time vacía como con valores. Se desconoce el motivo de que salga amarillo.

`if (registrationTime != null && dueDate != null)`
 	
`super.state(MomentHelper.isLongEnough(registrationTime, dueDate, 1, ChronoUnit.MONTHS) &&`
	Esta validación comprueba que el atributo registration time y el due date de la invoice no se pasen ni del límite inferior ni del superior y que el due date esté después del registration time. También que entre las dos

`dueDate.after(registrationTime) && dueDate.after(minimumDate) && dueDate.before(maximumDate), "dueDate", "sponsor.invoice.form.error.dueDate" );`	fechas haya mínimo un mes de diferencia ; la única opción que no se puede probar es que la due date se pase del límite superior y que esté a la vez antes de la fecha de registro, por lo tanto, la validación está cubierta.

2.1.2.6) Delete

- delete.safe: para hacer esta prueba, he intentado borrar una invoice en dos sponsorships 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 sponsor1, tras iniciada la sesión me he metido en una invoice y dándole a inspeccionar en la pestaña, he cambiado ese valor de id por 343 (invoice perteneciente al sponsor2) y le di al botón de delete, tras ello me volví a meter en el mismo invoice y cambié la id a 342 (audit record publicado de sponsor1) 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 los mismos.

- class SponsorInvoiceDeleteService:

```
1.  @Service
2.  public class SponsorInvoiceDeleteService extends
    AbstractService<Sponsor, Invoice> {
3.
4.      // Internal state
    ----------------------------------------------------------
5.
6.      @Autowired
7.      private SponsorInvoiceRepository repository;
8.
9.      // AbstractService interface
    ----------------------------------------------
10.
11.
12.     @Override
13.     public void authorise() {
14.         boolean status;
15.         int id;
16.         int sponsorId;
17.         Invoice invoice;
18.
19.         id = super.getRequest().getData("id", int.class);
20.         invoice = this.repository.findOneInvoiceById(id);
21.
22.         sponsorId =
    super.getRequest().getPrincipal().getActiveRoleId();
23.
```

```
24.         status = sponsorId == invoice.getSponsor().getId() &&
    invoice.isDraftMode();
25.
26.         super.getResponse().setAuthorised(status);
27.     }
28.
29.     @Override
30.     public void load() {
31.         Invoice object;
32.         int id;
33.
34.         id = super.getRequest().getData("id", int.class);
35.
36.         object = this.repository.findOneInvoiceById(id);
37.
38.         super.getBuffer().addData(object);
39.     }
40.
41.     @Override
42.     public void bind(final Invoice object) {
43.         assert object != null;
44.
45.         super.bind(object, "code", "registrationTime", "dueDate",
    "quantity", "tax", "link");
46.     }
47.
48.     @Override
49.     public void validate(final Invoice object) {
50.         assert object != null;
51.
52.         if
    (!super.getBuffer().getErrors().hasErrors("publishedSponsorship"))
53.             super.state(object.getSponsorship().isDraftMode(), "*",
    "sponsor.invoice.form.error.published-sponsorship");
54.     }
55.
56.     @Override
57.     public void perform(final Invoice object) {
58.         assert object != null;
59.
60.         this.repository.delete(object);
61.     }
62.
63. @Override
64.     public void unbind(final Invoice object) {
65.         assert object != null;
66.
67.         Dataset dataset;
68.
69.         dataset = super.unbind(object, "code", "registrationTime",
    "dueDate", "quantity", "tax", "link", "draftMode");
70.
71.         super.getResponse().addData(dataset);
72.     }
73. }
```

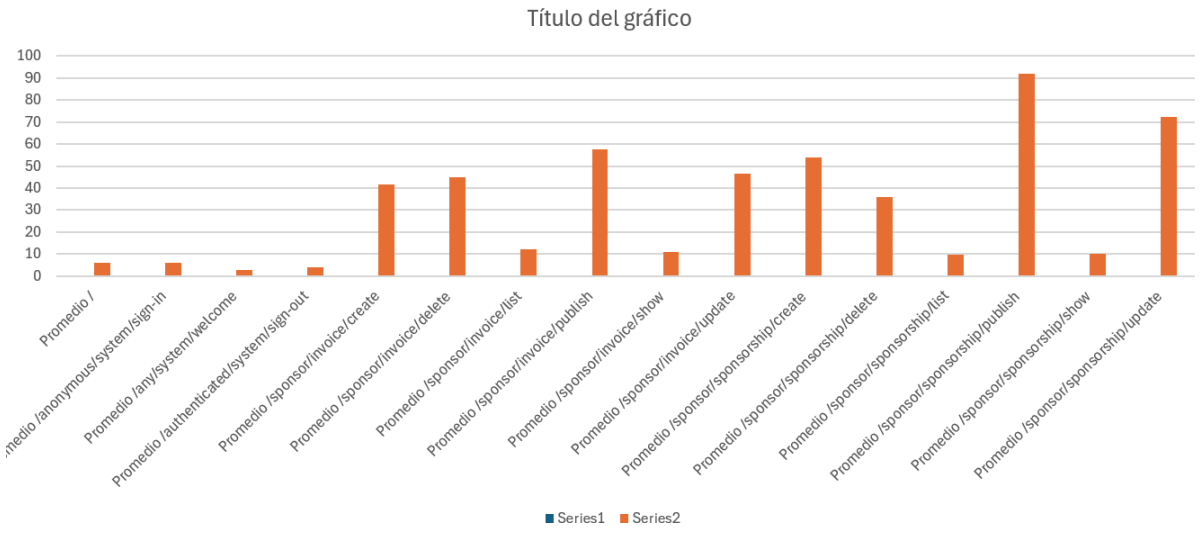
Línea amarilla (no completamente probada)	Explicación

`if (!super.getBuffer().getErrors().hasErrors("publishedSponsorship"))`	Este código no debería salir en amarillo ya que se han probado todos los casos posibles. Se desconoce el motivo de que salga amarillo. El caso negativo solo se puede hacer en el .hack y está explicado anteriormente.

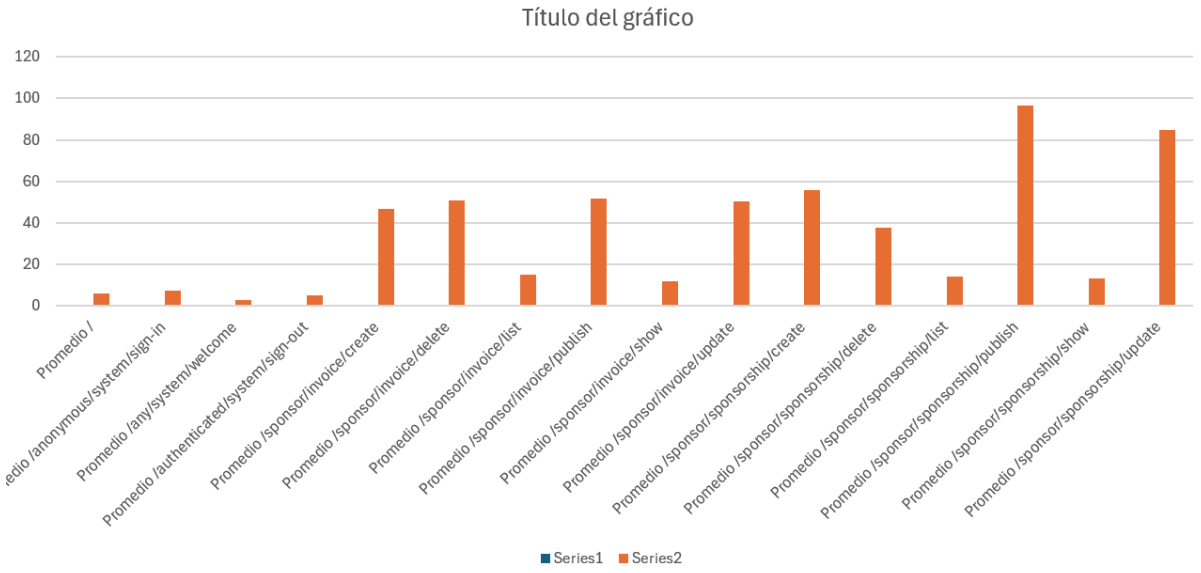
2.2) Performance testing

2.2.1) Comparativa entre dos PCs

PC_A



PC_B



Se puede ver que el PC_B tiene peor rendimiento que el PC_A, pero realmente están muy parejos.

PC_A				*PC_B*		
Media	25,8434206			Media	28,8110435	
Error típico	0,8971192			Error típico	0,90548054	
Mediana	10,46185			Mediana	13,689501	
Moda	1,5848			Moda	11,56	
Desviación estándar	30,0501457			Desviación estándar	30,2083292	
Varianza de la muestra	903,011257			Varianza de la muestra	912,54315	
Curtosis	1,99939985			Curtosis	1,34220355	
Coeficiente de asimetría	1,5942637			Coeficiente de asimetría	1,48025828	
Rango	164,8007			Rango	145,9682	
Mínimo	1,414			Mínimo	1,631	
Máximo	166,2147			Máximo	147,5992	
Suma	28996,3179			Suma	32066,6914	
Cuenta	1122			Cuenta	1113	
Nivel de confianza(95,0%)	1,76022184			Nivel de confianza(95,0%)	1,77664302	
Interval (ms)	24,0831987	27,6036424		Interval (ms)	27,0344004	30,5876865
Interval (s)	0,0240832	0,02760364		Interval (s)	0,0270344	0,03058769

El PC_A tiene un intervalo de confianza 95% (24,08; 27,60) y el PC_B tiene el intervalo (27,03; 30,59). Son intervalos más que aceptables.

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

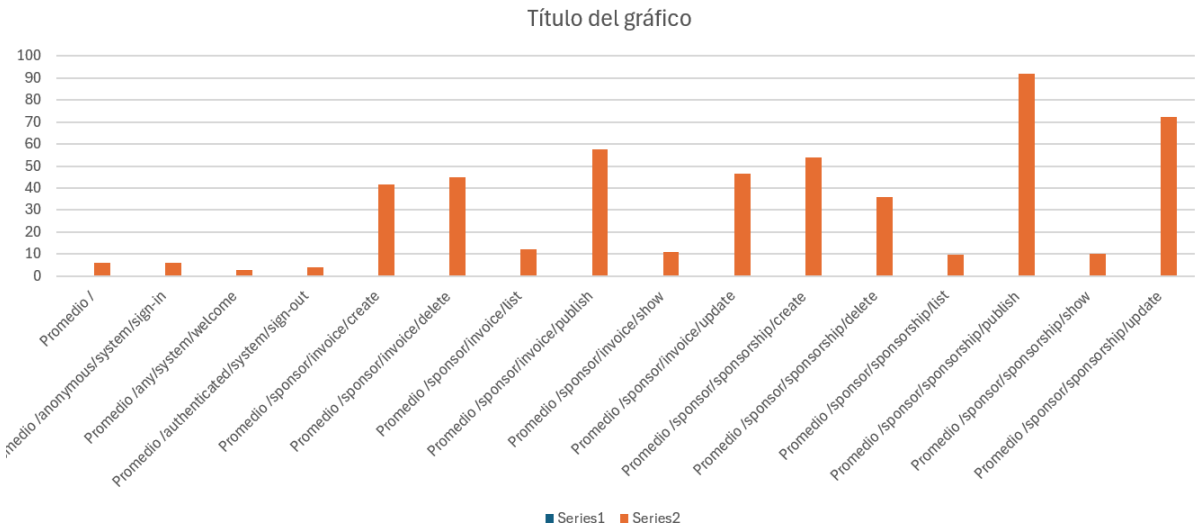
Prueba z para medias de dos muestras		
	PC_A	*PC_B*
Media	25,84342058	28,81104346
Varianza (conocida)	903,011257	912,54315
Observaciones	1122	1113
Diferencia hipotética de las medias	0	
z	-2,328197035	
P(Z<=z) una cola	0,009950822	
Valor crítico de z (una cola)	1,644853627	
Valor crítico de z (dos colas)	0,019901644	
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

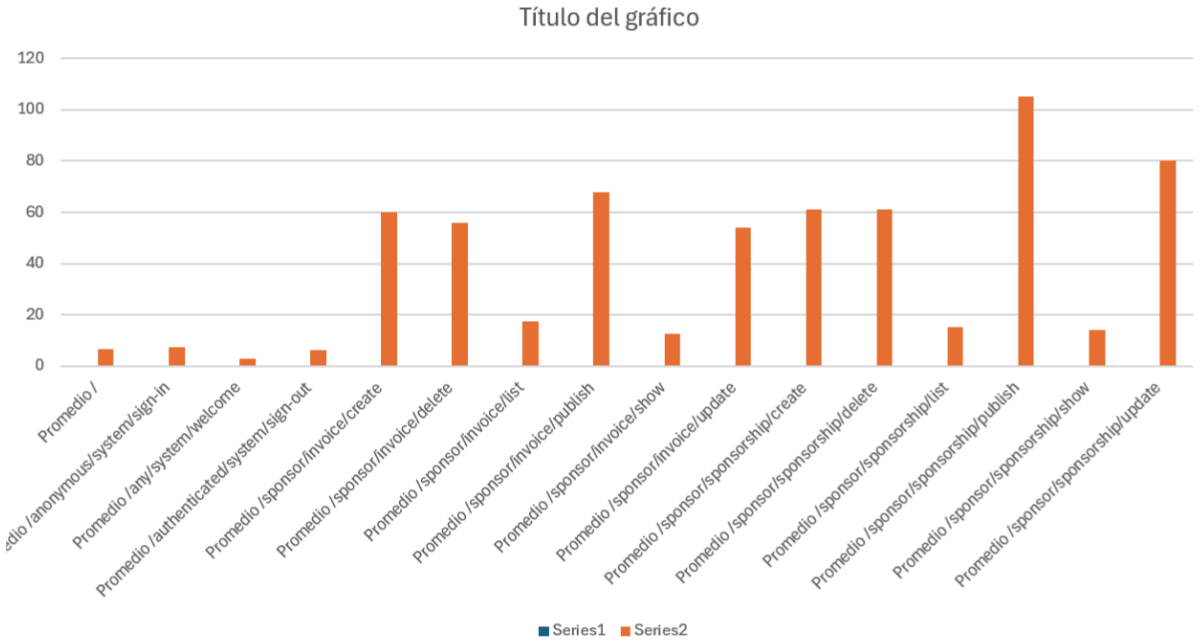
medias de los tiempos es una buena manera de averiguar qué ordenador es más potente. En este caso PC_A es mejor al tener una media de tiempos menor.

2.2.2) Comparativa con y sin índices

SIN ÍNDICES



CON ÍNDICES



Se puede ver que sin índices tiene mejor rendimiento que con índices, pero prácticamente, se compensan unos con otros.

Columna1				*Columna1*		
Media	25,8434206			Media	31,8775239	
Error típico	0,8971192			Error típico	1,04202146	
Mediana	10,46185			Mediana	14,5351	
Moda	1,5848			Moda	12,511	
Desviación estándar	30,0501457			Desviación estándar	34,9193813	
Varianza de la muestra	903,011257			Varianza de la muestra	1219,36319	
Curtosis	1,99939985			Curtosis	2,6605361	
Coeficiente de asimetría	1,5942637			Coeficiente de asimetría	1,67103226	
Rango	164,8007			Rango	214,2125	
Mínimo	1,414			Mínimo	1,4369	
Máximo	166,2147			Máximo	215,6494	
Suma	28996,3179			Suma	35798,4593	
Cuenta	1122			Cuenta	1123	
Nivel de confianza(95,0%)	1,76022184			Nivel de confianza(95,0%)	2,04453004	
Interval (ms)	24,0831987	27,6036424		Interval (ms)	29,8329939	33,9220539
Interval (s)	0,0240832	0,02760364		Interval (s)	0,02983299	0,03392205

Previo a los índices tiene un intervalo de confianza 95% (24,08; 27,60) y posterior a los índices tiene el intervalo (29,83; 33,92). Son intervalos más que aceptables.

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

Prueba z para medias de dos muestras		
	after	*before*
Media	25,75863882	31,77134855
Varianza (conocida)	903,011257	1219,36319
Observaciones	1105	1106
Diferencia hipotética de las medias	0	
z	-4,339634996	
P(Z<=z) una cola	7,13598E-06	
Valor crítico de z (una cola)	1,644853627	
Valor crítico de z (dos colas)	1,4272E-05	
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 con y sin índices es una buena manera de averiguar qué ordenador es más potente. En este caso, antes de los índices es mejor al tener una media de tiempos menor.

3) Conclusión

En conclusión, tras analizar tanto los tests como los tiempos de los mismos, podemos determinar que estos han sido llevados a cabo correctamente y que posteriormente, su análisis nos ha permitido determinar su eficacia respecto a otro ordenador de mi compañero así como pre y post índices.