Testing report

Diseño y pruebas II

Sprint 4

Versión 2.0

Fecha 23/06/2024

Preparado por:
Pablo Jesús Castellanos Compaña (C2.028)

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-extra.safe: se añadió un caso de prueba no probado previamente en el update.safe. Se realizó para no repetir el test entero ya que es de grandes dimensiones, por ello se complementó con este.

- 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:

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

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

    @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.         Collection<Invoice> allInvoices;
57.
58.         allInvoices =
this.repository.findInvoicesOfASponsorship(object.getId());
59.
60.         if (!super.getBuffer().getErrors().hasErrors("code")) {
61.
```

```
62.             Sponsorship projectSameCode =
this.repository.findOneSponsorshipByCode(object.getCode());
63.
64.             if (projectSameCode != null)
65.                 super.state(projectSameCode.getId() == object.getId(),
"code", "sponsor.sponsorship.form.error.code");
66.         }
67.
68.         if (!super.getBuffer().getErrors().hasErrors("moment")) {
69.
70.             Date sponsorshipDate = object.getMoment();
71.             Date minimumDate = MomentHelper.parse("1969-12-31 0:00",
"yyyy-MM-dd HH:mm");
72.             Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
"yyyy-MM-dd HH:mm");
73.
74.             if (sponsorshipDate != null) {
75.                 Boolean isAfter = sponsorshipDate.after(minimumDate)
&& sponsorshipDate.before(maximumDate);
76.                 super.state(isAfter, "moment",
"sponsor.sponsorship.form.error.moment");
77.             }
78.         }
79.
80.         if (!super.getBuffer().getErrors().hasErrors("moment")) {
81.             Invoice earliestInvoice;
82.             Boolean validMoment;
83.             Date moment = object.getMoment();
84.
85.             earliestInvoice =
this.repository.findInvoiceWithEarliestDateBySponsorshipId(object.getI
d()).stream().findFirst().orElse(null);
86.             //System.out.println(earliestInvoice);
87.
88.             if (earliestInvoice != null) {
89.                 //System.out.println(earliestInvoice);
90.                 validMoment =
moment.before(earliestInvoice.getRegistrationTime());
91.                 //System.out.println(validMoment);
92.                 super.state(validMoment, "moment",
"sponsor.sponsorship.form.error.creation-moment");
93.             }
94.         }
95.
96.         if
(!super.getBuffer().getErrors().hasErrors("durationStartTime")) {
97.             Date durationStartTime;
98.             Date moment;
99.             durationStartTime = object.getDurationStartTime();
100.                moment = object.getMoment();
101.                Date minimumDate = MomentHelper.parse("1969-12-31
0:00", "yyyy-MM-dd HH:mm");
102.                Date maximumDate = MomentHelper.parse("2200-12-31
23:59", "yyyy-MM-dd HH:mm");
103.
104.                if (moment != null)
105.                    super.state(durationStartTime.after(moment) &&
durationStartTime.after(minimumDate) &&
durationStartTime.before(maximumDate), "durationStartTime",
"sponsor.sponsorship.form.error.durationStartTime");
106.            }
```

```
107.
108.         if
(!super.getBuffer().getErrors().hasErrors("durationEndTime")) {
109.             Date durationStartTime;
110.             Date durationEndTime;
111.
112.             durationStartTime = object.getDurationStartTime();
113.             durationEndTime = object.getDurationEndTime();
114.             Date maximumDate = MomentHelper.parse("2200-12-31
23:59", "yyyy-MM-dd HH:mm");
115.
116.             if (durationStartTime != null && durationEndTime !=
null)
117.
super.state(MomentHelper.isLongEnough(durationStartTime,
durationEndTime, 1, ChronoUnit.MONTHS) &&
durationEndTime.after(durationStartTime) &&
durationEndTime.before(maximumDate), "durationEndTime",
118.
"sponsor.sponsorship.form.error.durationEndTime");
119.         }
120.
121.         if (!super.getBuffer().getErrors().hasErrors("amount") &&
this.systemConfigurationRepository.existsCurrency(object.getAmount().g
etCurrency()))
122.             super.state(object.getAmount().getAmount() >= 0 &&
object.getAmount().getAmount() <= 1000000, "amount",
"sponsor.sponsorship.form.error.amount");
123.
124.         if (!super.getBuffer().getErrors().hasErrors("amount")) {
125.             String symbol = object.getAmount().getCurrency();
126.             boolean existsCurrency =
this.systemConfigurationRepository.existsCurrency(symbol);
127.             super.state(existsCurrency, "amount",
"sponsor.sponsorship.form.error.acceptedCurrency");
128.         }
129.
130.         if (!super.getBuffer().getErrors().hasErrors("project"))
131.             super.state(!object.getProject().isDraftMode(),
"project", "sponsor.sponsorship.form.error.project-not-published");
132.
133.         if (object.getAmount() != null) {
134.             double sumaAmount = 0.0;
135.             for (Invoice i : allInvoices)
136.                 sumaAmount +=
this.systemConfigurationRepository.convertToUsd(i.totalAmount()).getAm
ount();
137.
138.             if
(this.systemConfigurationRepository.existsCurrency(object.getAmount().
getCurrency()))
139.
super.state(this.systemConfigurationRepository.convertToUsd(object.get
Amount()).getAmount() >= sumaAmount, "amount",
"sponsor.sponsorship.form.error.amount-less-sum-invoices");
140.         }
141.
142.     }
143.
144.     @Override
145.     public void perform(final Sponsorship object) {
```

```
146.         assert object != null;
147.
148.         this.repository.save(object);
149.     }
150.
151.     @Override
152.     public void unbind(final Sponsorship object) {
153.         assert object != null;
154.
155.         SelectChoices choices;
156.
157.         Collection<Project> projects =
    this.repository.findProjects();
158.         SelectChoices choices2;
159.         Dataset dataset;
160.         String projectCode;
161.         int sponsorId;
162.
163.         sponsorId = super.getRequest().getData("id", int.class);
164.
165.         Sponsorship s =
    this.repository.findOneSponsorshipById(sponsorId);
166.
167.         projectCode = s.getProject() != null ?
    s.getProject().getCode() : null;
168.
169.         choices = SelectChoices.from(SponsorshipType.class,
    object.getType());
170.         choices2 = SelectChoices.from(projects, "code",
    s.getProject());
171.
172.         dataset = super.unbind(object, "code", "moment",
    "durationStartTime", "durationEndTime", "amount", "type", "email",
    "link", "project", "draftMode");
173.         dataset.put("types", choices);
174.         dataset.put("projects", choices2);
175.         dataset.put("project", projectCode);
176.
177.         super.getResponse().addData(dataset);
178.     }
179.
180. }
181.
```

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(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.
`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:

```
1. @Service
2. public class SponsorSponsorshipPublishService 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.         int id;
36.
37.         id = super.getRequest().getData("id", int.class);
38.         object = this.repository.findOneSponsorshipById(id);
39.
40.         super.getBuffer().addData(object);
41.     }
42.
43.     @Override
44.     public void bind(final Sponsorship object) {
45.         assert object != null;
46.
47.         super.bind(object, "code", "moment", "durationStartTime",
    "durationEndTime", "amount", "type", "email", "link", "project");
48.     }
49.
50.     @Override
51.     public void validate(final Sponsorship object) {
52.         assert object != null;
53.
54.         Collection<Invoice> allInvoices;
55.
56.         allInvoices =
    this.repository.findInvoicesOfASponsorship(object.getId());
57.
```

```
58.         if (!super.getBuffer().getErrors().hasErrors("code")) {
59.
60.             Sponsorship projectSameCode =
this.repository.findOneSponsorshipByCode(object.getCode());
61.
62.             if (projectSameCode != null)
63.                 super.state(projectSameCode.getId() == object.getId(),
"code", "sponsor.sponsorship.form.error.code");
64.         }
65.
66.         if (!super.getBuffer().getErrors().hasErrors("totalAmount") &&
object.getAmount() != null &&
this.systemConfigurationRepository.existsCurrency(object.getAmount().g
etCurrency())) {
67.             double sumaAmount = 0.0;
68.             for (Invoice i : allInvoices)
69.                 sumaAmount +=
this.systemConfigurationRepository.convertToUsd(i.totalAmount()).getAm
ount();
70.
71.             //if (object.getAmount() != null)
72.
73.             super.state(sumaAmount ==
this.systemConfigurationRepository.convertToUsd(object.getAmount()).ge
tAmount(), "*", "sponsor.sponsorship.form.error.invalidTotalAmount");
74.         }
75.
76.         if (!super.getBuffer().getErrors().hasErrors("moment")) {
77.
78.             Date sponsorshipDate = object.getMoment();
79.             Date minimumDate = MomentHelper.parse("1969-12-31 0:00",
"yyyy-MM-dd HH:mm");
80.             Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
"yyyy-MM-dd HH:mm");
81.
82.             if (sponsorshipDate != null) {
83.                 Boolean isAfter = sponsorshipDate.after(minimumDate)
&& sponsorshipDate.before(maximumDate);
84.                 super.state(isAfter, "moment",
"sponsor.sponsorship.form.error.moment");
85.             }
86.         }
87.
88.         if (!super.getBuffer().getErrors().hasErrors("moment")) {
89.             Invoice earliestInvoice;
90.             Boolean validMoment;
91.             Date moment = object.getMoment();
92.
93.             earliestInvoice =
this.repository.findInvoiceWithEarliestDateBySponsorshipId(object.getI
d()).stream().findFirst().orElse(null);
94.             //System.out.println(earliestInvoice);
95.
96.             if (earliestInvoice != null) {
97.                 //System.out.println(earliestInvoice);
98.                 validMoment =
moment.before(earliestInvoice.getRegistrationTime());
99.                 //System.out.println(validMoment);
100.                    super.state(validMoment, "moment",
"sponsor.sponsorship.form.error.creation-moment");
101.            }
```

```
102.            }
103.
104.            if
    (!super.getBuffer().getErrors().hasErrors("durationStartTime")) {
105.                Date durationStartTime;
106.                Date moment;
107.                durationStartTime = object.getDurationStartTime();
108.                moment = object.getMoment();
109.                Date minimumDate = MomentHelper.parse("1969-12-31
    0:00", "yyyy-MM-dd HH:mm");
110.                Date maximumDate = MomentHelper.parse("2200-12-31
    23:59", "yyyy-MM-dd HH:mm");
111.
112.                if (moment != null)
113.                    super.state(durationStartTime.after(moment) &&
    durationStartTime.after(minimumDate) &&
    durationStartTime.before(maximumDate), "durationStartTime",
    "sponsor.sponsorship.form.error.durationStartTime");
114.            }
115.
116.            if
    (!super.getBuffer().getErrors().hasErrors("durationEndTime")) {
117.                Date durationStartTime;
118.                Date durationEndTime;
119.
120.                durationStartTime = object.getDurationStartTime();
121.                durationEndTime = object.getDurationEndTime();
122.                Date maximumDate = MomentHelper.parse("2200-12-31
    23:59", "yyyy-MM-dd HH:mm");
123.
124.                if (durationStartTime != null && durationEndTime !=
    null)
125.
    super.state(MomentHelper.isLongEnough(durationStartTime,
    durationEndTime, 1, ChronoUnit.MONTHS) &&
    durationEndTime.after(durationStartTime) &&
    durationEndTime.before(maximumDate), "durationEndTime",
126.
    "sponsor.sponsorship.form.error.durationEndTime");
127.            }
128.
129.            if (!super.getBuffer().getErrors().hasErrors("amount") &&
    this.systemConfigurationRepository.existsCurrency(object.getAmount().g
    etCurrency()))
130.                super.state(object.getAmount().getAmount() >= 0 &&
    object.getAmount().getAmount() <= 1000000, "amount",
    "sponsor.sponsorship.form.error.amount");
131.
132.            if (!super.getBuffer().getErrors().hasErrors("amount")) {
133.                String symbol = object.getAmount().getCurrency();
134.                boolean existsCurrency =
    this.systemConfigurationRepository.existsCurrency(symbol);
135.                super.state(existsCurrency, "amount",
    "sponsor.sponsorship.form.error.acceptedCurrency");
136.            }
137.
138.            if
    (!super.getBuffer().getErrors().hasErrors("unpublishedInvoices")) {
139.
140.                Collection<Invoice> unpublishedInvoices;
141.
```

```
142.                 unpublishedInvoices =
    this.repository.findUnpublishedInvoicesBySponsorshipId(object.getId())
    ;
143.
144.                 super.state(unpublishedInvoices.isEmpty(), "*",
    "sponsor.sponsorship.form.error.unpublished-invoices");
145.             }
146.
147.             if (!super.getBuffer().getErrors().hasErrors("project"))
148.                 super.state(!object.getProject().isDraftMode(),
    "project", "sponsor.sponsorship.form.error.project-not-published");
149.
150.             if (object.getAmount() != null) {
151.                 double sumaAmount = 0.0;
152.                 for (Invoice i : allInvoices)
153.                     sumaAmount +=
    this.systemConfigurationRepository.convertToUsd(i.totalAmount()).getAm
    ount();
154.
155.                 if
    (this.systemConfigurationRepository.existsCurrency(object.getAmount().
    getCurrency()))
156.
    super.state(this.systemConfigurationRepository.convertToUsd(object.get
    Amount()).getAmount() >= sumaAmount, "amount",
    "sponsor.sponsorship.form.error.amount-less-sum-invoices");
157.             }
158.
159.         }
160.
161.         @Override
162.         public void perform(final Sponsorship object) {
163.             assert object != null;
164.
165.             object.setDraftMode(false);
166.             this.repository.save(object);
167.         }
168.
169.         @Override
170.         public void unbind(final Sponsorship object) {
171.             assert object != null;
172.
173.             SelectChoices choices;
174.
175.             Collection<Project> projects =
    this.repository.findProjects();
176.             SelectChoices choices2;
177.             Dataset dataset;
178.             String projectCode;
179.             int sponsorId;
180.
181.             sponsorId = super.getRequest().getData("id", int.class);
182.
183.             Sponsorship s =
    this.repository.findOneSponsorshipById(sponsorId);
184.
185.             projectCode = s.getProject() != null ?
    s.getProject().getCode() : null;
186.
187.             choices = SelectChoices.from(SponsorshipType.class,
    object.getType());
```

```
188.             choices2 = SelectChoices.from(projects, "code",
    s.getProject());
189.
190.             dataset = super.unbind(object, "code", "moment",
    "durationStartTime", "durationEndTime", "amount", "type", "email",
    "link", "project", "draftMode");
191.             dataset.put("types", choices);
192.             dataset.put("projects", choices2);
193.             dataset.put("project", projectCode);
194.
195.             super.getResponse().addData(dataset);
196.         }
197.
198.     }
```

Línea amarilla (no completamente probada)	Explicación
`if (!super.getBuffer().getErrors().hasErrors("totalAmount") && object.getAmount() != null && this.systemConfigurationRepository.existsCurrency(object.getAmount().getCurrency())) {`	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(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.
`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",`	Este código comprueba una validación realizada para evitar un posible hackeo en

`"sponsor.sponsorship.form.error.project-not-published");`	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:

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

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

    @Autowired
    private SponsorInvoiceRepository repository;

    // 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() {
        Collection<Invoice> objects;
        int sponsorshipId;

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

        objects =
this.repository.findAllInvoicesBySponsorshipId(sponsorshipId);

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

    @Override
    public void unbind(final Invoice object) {
        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.class);
60.         sponsorship =
this.repository.findOneSponsorshipById(sponsorshipId);
61.
62.         super.getResponse().addGlobal("sponsorshipId", sponsorshipId);
63.         super.getResponse().addGlobal("canCreate",
sponsorship.isDraftMode());
64.     }
65.
66. }
67.
```

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:

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

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

    @Autowired
    private SponsorInvoiceRepository repository;

    // 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);

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

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

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

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

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

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

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

        Dataset dataset;

        dataset = super.unbind(object, "code", "registrationTime", "dueDate", "quantity", "tax", "link", "draftMode");
        dataset.put("totalAmount", object.totalAmount());

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

}

```

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:

```
1. @Service
2. public class SponsorInvoiceCreateService extends
   AbstractService<Sponsor, Invoice> {
3.
4.     // Internal state
   ---------------------------------------------------------
5.
6.     @Autowired
7.     private SponsorInvoiceRepository        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 sponsorshipId;
19.         int sponsorId;
20.         Sponsorship sponsorship;
21.
22.         sponsorshipId = super.getRequest().getData("sponsorshipId",
    int.class);
23.         sponsorship =
    this.repository.findOneSponsorshipById(sponsorshipId);
24.
25.         sponsorId =
    super.getRequest().getPrincipal().getActiveRoleId();
26.
27.         status = sponsorId == sponsorship.getSponsor().getId();
28.
29.         super.getResponse().setAuthorised(status);
30.     }
31.
32.     @Override
33.     public void load() {
34.         Invoice object;
35.         Integer sponsorshipId;
36.         Sponsorship sponsorship;
37.
38.         object = new Invoice();
39.         Integer sponsorId =
    super.getRequest().getPrincipal().getActiveRoleId();
40.         Sponsor sponsor =
    this.repository.findOneSponsorById(sponsorId);
41.
42.         sponsorshipId = super.getRequest().getData("sponsorshipId",
    int.class);
43.         sponsorship =
    this.repository.findOneSponsorshipById(sponsorshipId);
44.
45.         object.setSponsor(sponsor);
46.         object.setSponsorship(sponsorship);
47.         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.         Collection<Invoice> allInvoices;
65.
66.         allInvoices =
    this.repository.findAllInvoicesBySponsorshipId(object.getSponsorship()
    .getId());
67.
68.         if (!super.getBuffer().getErrors().hasErrors("code")) {
```

```
69.
70.             Invoice projectSameCode =
    this.repository.findOneInvoiceByCode(object.getCode());
71.
72.             super.state(projectSameCode == null, "code",
    "sponsor.invoice.form.error.code");
73.         }
74.
75.         if
    (!super.getBuffer().getErrors().hasErrors("registrationTime")) {
76.
77.             Date registrationTime = object.getRegistrationTime();
78.             Date minimumDate = MomentHelper.parse("1969-12-31 23:59",
    "yyyy-MM-dd HH:mm");
79.             Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
    "yyyy-MM-dd HH:mm");
80.
81.             if (registrationTime != null) {
82.                 Boolean isAfter = registrationTime.after(minimumDate)
    && registrationTime.before(maximumDate);
83.                 super.state(isAfter, "registrationTime",
    "sponsor.invoice.form.error.registration-time");
84.             }
85.         }
86.
87.         if
    (!super.getBuffer().getErrors().hasErrors("registrationTime")) {
88.             Date registrationTime;
89.             Date moment;
90.
91.             registrationTime = object.getRegistrationTime();
92.             moment = object.getSponsorship().getMoment();
93.
94.             if (registrationTime != null)
95.                 super.state(registrationTime.after(moment),
    "registrationTime",
    "sponsor.invoice.form.error.registration-time-bis");
96.         }
97.
98.         if (!super.getBuffer().getErrors().hasErrors("dueDate")) {
99.             Date registrationTime;
100.                Date dueDate;
101.
102.                registrationTime = object.getRegistrationTime();
103.                dueDate = object.getDueDate();
104.                Date minimumDate = MomentHelper.parse("1969-12-31
    23:59", "yyyy-MM-dd HH:mm");
105.                Date maximumDate = MomentHelper.parse("2200-12-31
    23:59", "yyyy-MM-dd HH:mm");
106.
107.                if (registrationTime != null && dueDate != null)
108.
    super.state(MomentHelper.isLongEnough(registrationTime, dueDate, 1,
    ChronoUnit.MONTHS) && dueDate.after(registrationTime) &&
    dueDate.after(minimumDate) && dueDate.before(maximumDate), "dueDate",
    "sponsor.invoice.form.error.dueDate");
109.            }
110.
111.        if (!super.getBuffer().getErrors().hasErrors("quantity") &&
    this.systemConfigurationRepository.existsCurrency(object.getQuantity()
    .getCurrency()))
```

```
112.                 super.state(object.getQuantity().getAmount() >= 0 &&
    object.getQuantity().getAmount() <= 1000000, "quantity",
    "sponsor.invoice.form.error.quantity");
113.
114.             if (!super.getBuffer().getErrors().hasErrors("quantity")) {
115.                 String symbol = object.getQuantity().getCurrency();
116.                 boolean existsCurrency =
    this.systemConfigurationRepository.existsCurrency(symbol);
117.                 super.state(existsCurrency, "quantity",
    "sponsor.invoice.form.error.acceptedCurrency");
118.             }
119.
120.             if
    (!super.getBuffer().getErrors().hasErrors("publishedSponsorship")) {
121.                 Integer sponsorshipId;
122.                 Sponsorship sponsorship;
123.
124.                 sponsorshipId =
    super.getRequest().getData("sponsorshipId", int.class);
125.                 sponsorship =
    this.repository.findOneSponsorshipById(sponsorshipId);
126.
127.                 super.state(sponsorship.isDraftMode(), "*",
    "sponsor.invoice.form.error.published-sponsorship");
128.             }
129.
130.             if (!super.getBuffer().getErrors().hasErrors()) {
131.                 double sumaNueva =
    this.systemConfigurationRepository.convertToUsd(object.totalAmount()).
    getAmount();
132.                 for (Invoice i : allInvoices)
133.                     sumaNueva +=
    this.systemConfigurationRepository.convertToUsd(i.totalAmount()).getAm
    ount();
134.
135.                 if
    (this.systemConfigurationRepository.existsCurrency(object.getQuantity(
    ).getCurrency()))
136.                     super.state(sumaNueva <=
    this.systemConfigurationRepository.convertToUsd(object.getSponsorship(
    ).getAmount()).getAmount(), "*",
    "sponsor.invoice.form.error.incorrect-sum");
137.             }
138.
139.         }
140.
141.         @Override
142.         public void perform(final Invoice object) {
143.             assert object != null;
144.
145.             if (object.getTax() == null)
146.                 object.setTax(0.0);
147.
148.             this.repository.save(object);
149.         }
150.
151.         @Override
152.         public void unbind(final Invoice object) {
153.             assert object != null;
154.
155.             Dataset dataset;
```

```
156.
157.            dataset = super.unbind(object, "code", "registrationTime",
    "dueDate", "quantity", "tax", "link", "draftMode");
158.            dataset.put("sponsorshipId",
    super.getRequest().getData("sponsorshipId", int.class));
159.
160.            super.getResponse().addData(dataset);
161.        }
162.
163.    }
164.
165.
```

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 invoice en un sponsorship publicado usando el sistema de forma correcta.
`if (this.systemConfigurationRepository.existsCurrency(object.getQuantity().getCurrency()))`	Aquí se comprueba si la divisa que se le pasa al formulario existe o no, ambas posibilidades han sido contempladas, por lo tanto, no debería salir en amarillo.

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 (invoice 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;
```

```
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.
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.         Collection<Invoice> allInvoices;
58.
59.         allInvoices =
    this.repository.findAllInvoicesBySponsorshipId(object.getSponsorship()
    .getId());
60.
61.         if (!super.getBuffer().getErrors().hasErrors("code")) {
62.
```

```
63.             Invoice projectSameCode =
    this.repository.findOneInvoiceByCode(object.getCode());
64.
65.             if (projectSameCode != null)
66.                 super.state(projectSameCode.getId() == object.getId(),
    "code", "sponsor.invoice.form.error.code");
67.         }
68.
69.         if
    (!super.getBuffer().getErrors().hasErrors("registrationTime")) {
70.
71.             Date registrationTime = object.getRegistrationTime();
72.             Date minimumDate = MomentHelper.parse("1969-12-31 23:59",
    "yyyy-MM-dd HH:mm");
73.             Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
    "yyyy-MM-dd HH:mm");
74.
75.             if (registrationTime != null) {
76.                 Boolean isAfter = registrationTime.after(minimumDate)
    && registrationTime.before(maximumDate);
77.                 super.state(isAfter, "registrationTime",
    "sponsor.invoice.form.error.registration-time");
78.             }
79.         }
80.
81.         if
    (!super.getBuffer().getErrors().hasErrors("registrationTime")) {
82.             Date registrationTime;
83.             Date moment;
84.
85.             registrationTime = object.getRegistrationTime();
86.             moment = object.getSponsorship().getMoment();
87.
88.             if (registrationTime != null)
89.                 super.state(registrationTime.after(moment),
    "registrationTime",
    "sponsor.invoice.form.error.registration-time-bis");
90.         }
91.
92.         if (!super.getBuffer().getErrors().hasErrors("dueDate")) {
93.             Date registrationTime;
94.             Date dueDate;
95.
96.             registrationTime = object.getRegistrationTime();
97.             dueDate = object.getDueDate();
98.             Date minimumDate = MomentHelper.parse("1969-12-31 23:59",
    "yyyy-MM-dd HH:mm");
99.             Date maximumDate = MomentHelper.parse("2200-12-31 23:59",
    "yyyy-MM-dd HH:mm");
100.
101.                if (registrationTime != null && dueDate != null)
102.
    super.state(MomentHelper.isLongEnough(registrationTime, dueDate, 1,
```

```
ChronoUnit.MONTHS) && dueDate.after(registrationTime) &&
dueDate.after(minimumDate) && dueDate.before(maximumDate), "dueDate",
"sponsor.invoice.form.error.dueDate");
            }

            if (!super.getBuffer().getErrors().hasErrors("quantity") &&
this.systemConfigurationRepository.existsCurrency(object.getQuantity()
.getCurrency()))
                super.state(object.getQuantity().getAmount() >= 0 &&
object.getQuantity().getAmount() <= 1000000, "quantity",
"sponsor.invoice.form.error.quantity");

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

            if
(!super.getBuffer().getErrors().hasErrors("publishedSponsorship"))
                super.state(object.getSponsorship().isDraftMode(), "*",
"sponsor.invoice.form.error.published-sponsorship");

            if (!super.getBuffer().getErrors().hasErrors()) {
                double valorAntiguo =
this.systemConfigurationRepository.convertToUsd(this.repository.findOn
eInvoiceById(object.getId()).totalAmount()).getAmount();
                double sumaNueva =
this.systemConfigurationRepository.convertToUsd(object.totalAmount()).
getAmount() - valorAntiguo;
                for (Invoice i : allInvoices)
                    sumaNueva +=
this.systemConfigurationRepository.convertToUsd(i.totalAmount()).getAm
ount();

                if
(this.systemConfigurationRepository.existsCurrency(object.getQuantity(
).getCurrency()))
                    super.state(sumaNueva <=
this.systemConfigurationRepository.convertToUsd(object.getSponsorship(
).getAmount()).getAmount(), "*",
"sponsor.invoice.form.error.incorrect-sum");
            }

        }

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

            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("totalAmount", object.totalAmount());
147.
148.             super.getResponse().addData(dataset);
149.         }
150.
151.     }
```

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 invoice en un sponsorship publicado

	usando el sistema de forma correcta.
`if (this.systemConfigurationRepository.existsCurrency(object.getQuantity().getCurrency()))`	Aquí se comprueba si la divisa que se le pasa al formulario existe o no, ambas posibilidades han sido contempladas, por lo tanto, no debería salir en amarillo.

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 (invoice 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 ---------------------------------------------------------

    @Autowired
    private SponsorInvoiceRepository        repository;

    @Autowired
    private SystemConfigurationRepository systemConfigurationRepository;

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


    @Override
    public void authorise() {
        boolean status;
        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.         Collection<Invoice> allInvoices;
54.
55.         allInvoices =
    this.repository.findAllInvoicesBySponsorshipId(object.getSponsorship()
    .getId());
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");

            if (registrationTime != null) {
                Boolean isAfter = registrationTime.after(minimumDate) && registrationTime.before(maximumDate);
                super.state(isAfter, "registrationTime", "sponsor.invoice.form.error.registration-time");
            }
        }

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

            registrationTime = object.getRegistrationTime();
            moment = object.getSponsorship().getMoment();

            if (registrationTime != null)
                super.state(registrationTime.after(moment), "registrationTime", "sponsor.invoice.form.error.registration-time-bis");
        }

        if (!super.getBuffer().getErrors().hasErrors("dueDate")) {
            Date registrationTime;
            Date dueDate;

            registrationTime = object.getRegistrationTime();
            dueDate = object.getDueDate();
            Date minimumDate = MomentHelper.parse("1969-12-31 23:59", "yyyy-MM-dd HH:mm");
            Date maximumDate = MomentHelper.parse("2200-12-31 23:59", "yyyy-MM-dd HH:mm");

            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");
        }

        if (!super.getBuffer().getErrors().hasErrors("quantity") && this.systemConfigurationRepository.existsCurrency(object.getQuantity().getCurrency()))
            super.state(object.getQuantity().getAmount() >= 0 && object.getQuantity().getAmount() <= 1000000, "quantity", "sponsor.invoice.form.error.quantity");

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

        if (!super.getBuffer().getErrors().hasErrors()) {
            double valorAntiguo = this.systemConfigurationRepository.convertToUsd(this.repository.findOn
```

```
eInvoiceById(object.getId()).totalAmount()).getAmount();
                double sumaNueva =
this.systemConfigurationRepository.convertToUsd(object.totalAmount()).
getAmount() - valorAntiguo;
                for (Invoice i : allInvoices)
                    sumaNueva +=
this.systemConfigurationRepository.convertToUsd(i.totalAmount()).getAm
ount();

                if
(this.systemConfigurationRepository.existsCurrency(object.getQuantity(
).getCurrency()))
                    super.state(sumaNueva <=
this.systemConfigurationRepository.convertToUsd(object.getSponsorship(
).getAmount()).getAmount(), "*",
"sponsor.invoice.form.error.incorrect-sum");
            }

        }

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

            if (object.getTax() == null)
                object.setTax(0.0);

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

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

            Dataset dataset;

            dataset = super.unbind(object, "code", "registrationTime",
"dueDate", "quantity", "tax", "link", "draftMode");
            dataset.put("totalAmount", object.totalAmount());

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

    }
```

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 (this.systemConfigurationRepository.existsCurrency(object.getQuantity().getCurrency()))`	Aquí se comprueba si la divisa que se le pasa al formulario existe o no, ambas posibilidades han sido contempladas, por lo tanto, no debería salir en amarillo.

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 (invoice 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
```

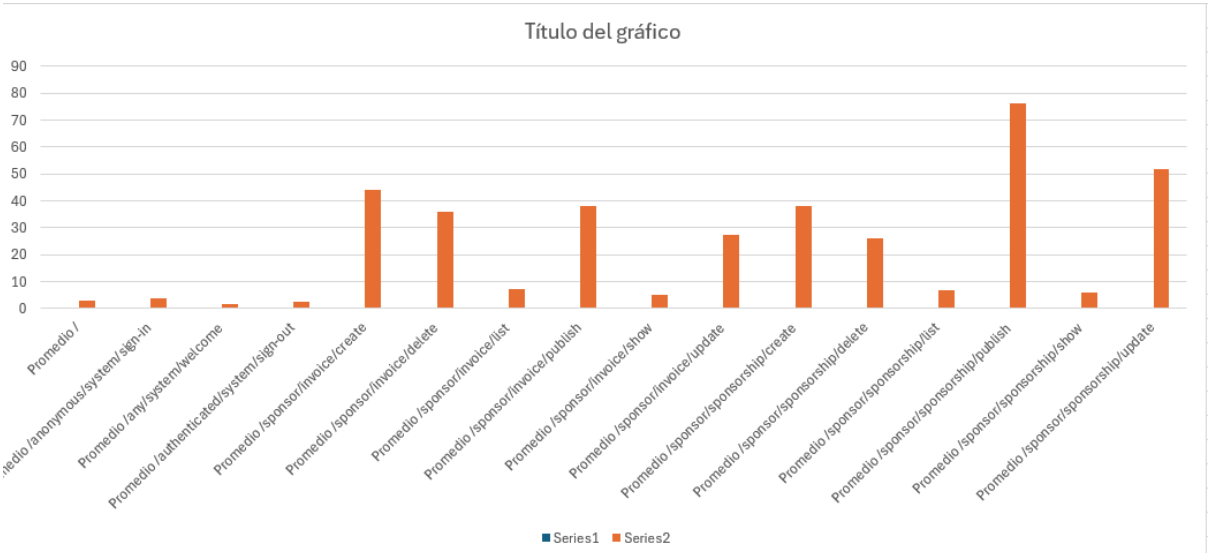
```
64.     public void unbind(final Invoice object) {
65.         assert object != null;
66.
67.         Dataset dataset;
68.
69.         dataset = super.unbind(object, "code", "registrationTime",
    "dueDate", "quantity", "tax", "link", "draftMode");
70.
71.         super.getResponse().addData(dataset);
72.     }
73.
74. }
75.
```

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.

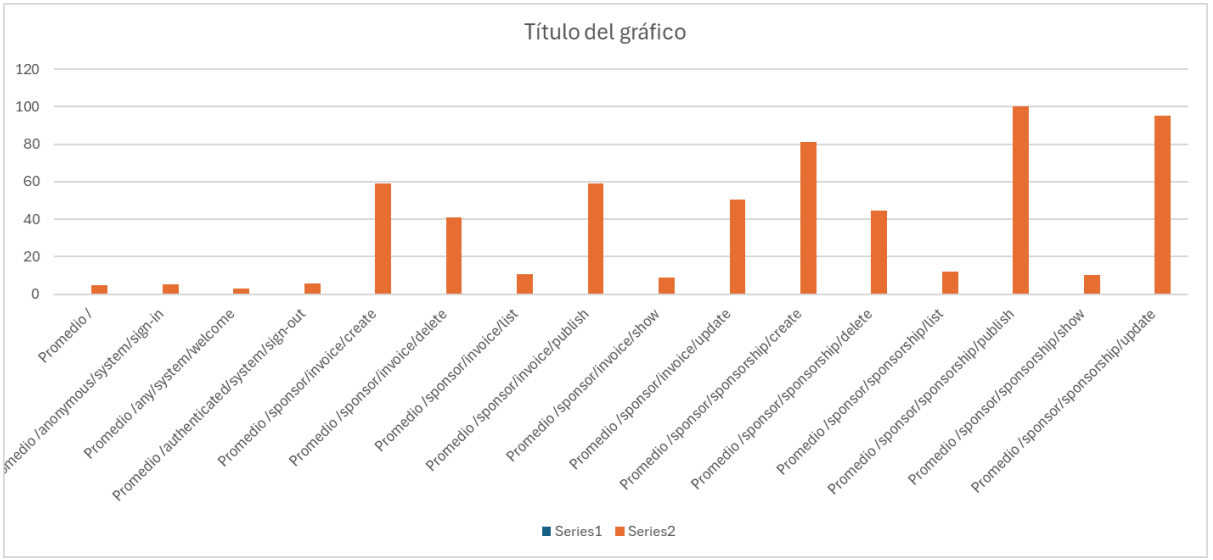
2.2) Performance testing

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 están relativamente parejos.

PC_A				*PC_B*		
Media	18,30815516			Media	29,93779295	
Error típico	0,674313589			Error típico	1,096640122	
Mediana	5,8147			Mediana	9,4020505	
Moda	2,6135			Moda	2,058	
Desviación estándar	22,98608327			Desviación estándar	37,38240129	
Varianza de la muestra	528,3600239			Varianza de la muestra	1397,443926	
Curtosis	1,823018017			Curtosis	1,67832335	
Coeficiente de asimetría	1,621425349			Coeficiente de asimetría	1,597663636	
Rango	114,1959			Rango	213,2746	
Mínimo	0,6062			Mínimo	1,1688	
Máximo	114,8021			Máximo	214,4434	
Suma	21274,0763			Suma	34787,71541	
Cuenta	1162			Cuenta	1162	
Nivel de confianza(95,0%)	1,323009586			Nivel de confianza(95,0%)	2,151618206	
Interval (ms)	16,98514558	19,631165		Interval (ms)	27,78617474	32,089411
Interval (s)	0,016985146	0,0196312		Interval (s)	0,027786175	0,0320894

El PC_A tiene un intervalo de confianza 95% (16,99; 16,63) en milisegundos y el PC_B tiene el intervalo (27,79; 32,09) en milisegundos también. 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	18,30815516	29,93779295
Varianza (conocida)	528,3600239	1397,443926
Observaciones	1162	1162
Diferencia hipotética de las medias	0	
z	-9,033649079	
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 podemos observar, el valor P (valor crítico de z (dos colas)) es 0, denotando que hay una diferencia significativa. 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.

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 al ordenador de mi compañero.