Reporte de Testing

Logotipo

Descripción generada automáticamente

**Grupo:** C3.039

**Repositorio:** <https://github.com/pabalcber/C3.039-Acme-SF>

**Integrantes:**

|  |  |  |
| --- | --- | --- |
| Nombre | Apellidos | Correo Corporativo |
| Jun | Yao | junyao@alum.us.es |
|  |  |  |

Tabla de versiones:

|  |  |  |  |
| --- | --- | --- | --- |
| Fecha | Versión | Descripción de los cambios | Sprint |
| 24/05/2024 | 1.0 | Creación del documento y redacción de sus contenidos | 4 |
| 26/08/2024 | 2.0 | Terminar el documento | 4 |
| 15/10/2024 | 3.0 | Revisión final | 4 |
|  |  |  |  |

1. [Introducción 3](#_TOC_250019)
2. [Contenido 3](#_TOC_250018)
   1. [Functional testing 3](#_TOC_250017)
      1. [Sponsorship 3](#_TOC_250016)
         1. [List 3](#_TOC_250015)
         2. [Show 5](#_TOC_250014)
         3. [Create 7](#_TOC_250013)
         4. [Update 11](#_TOC_250012)
         5. [Publish 16](#_TOC_250011)
         6. [Delete 22](#_TOC_250010)
      2. [Invoice 25](#_TOC_250009)
         1. [List 25](#_TOC_250008)
         2. [Show 26](#_TOC_250007)
         3. [Create 28](#_TOC_250006)
         4. [Update 33](#_TOC_250005)
         5. [Publish 38](#_TOC_250004)
         6. [Delete 42](#_TOC_250003)
   2. [Performance testing 45](#_TOC_250002)
      1. [Comparativa entre dos PCs 45](#_TOC_250001)
3. [Conclusión 47](#_TOC_250000)

# 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.

# Contenido

## Functional testing

### Sponsorship

#### 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:

1. @Service
2. **public class** SponsorSponsorshipListService **extends** AbstractService<Sponsor, Sponsorship> {

3.

4. // Internal state

5.

1. @Autowired
2. **private** SponsorSponsorshipRepository repository; 8.

9. // AbstractService interface

10.

11.

1. @Override
2. **public void** authorise() {
3. **super**.getResponse().setAuthorised(**true**); 15. }

16.

1. @Override
2. **public void** load() {
3. Collection<Sponsorship> objects;
4. **int** sponsorId; 21.

22. sponsorId =

**super**.getRequest().getPrincipal().getActiveRoleId();

23.

24. objects =

**this**.repository.findSponsorshipBySponsorId(sponsorId);

25.

26. **super**.getBuffer().addData(objects); 27. }

28.

1. @Override
2. **public void** unbind(**final** Sponsorship object) { 31. assert object != **null**;

32.

33. Dataset dataset; 34.

1. dataset = **super**.unbind(object, "code", "moment", "durationStartTime", "durationEndTime", "amount", "type", "email", "link");
2. dataset.put("project", object.getProject().getCode()); 37.

38. **super**.getResponse().addData(dataset); 39. }

40.

41.}

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

#### 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.

1. @Autowired
2. **private** SponsorSponsorshipRepository repository; 8.

9. // AbstractService interface

10.

11.

1. @Override
2. **public void** authorise() {
3. **boolean** status;
4. **int** id;
5. **int** sponsorId;
6. Sponsorship sponsorship; 18.
7. id = **super**.getRequest().getData("id", **int**.**class**);
8. 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.

1. @Override
2. **public void** load() {
3. Sponsorship object;
4. **int** id; 33.

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

34.

35.

36.

37. **super**.getBuffer().addData(object); 38. }

39.

1. @Override
2. **public void** unbind(**final** Sponsorship object) { 42. assert object != **null**;

43.

1. SelectChoices choices;
2. Collection<Project> projects = **this**.repository.findProjects();
3. SelectChoices choices2;
4. Dataset dataset; 48.

49. choices = SelectChoices.from(SponsorshipType.**class**, object.getType());

50.

51. choices2 = SelectChoices.from(projects, "code", (Project) projects.toArray()[0]);

52.

1. dataset = **super**.unbind(object, "code", "moment", "durationStartTime", "durationEndTime", "amount", "type", "email", "link", "draftMode");
2. 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 |

#### 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(minimumD ate) && sponsorshipDate.before(maximum Date); | 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 mDate) && durationStartTime.before(maxim umDate), "durationStartTime", "sponsor.sponsorship.form.erro r.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.erro r.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. |

#### 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:

1. @Service
2. **public class** SponsorSponsorshipUpdateService **extends** AbstractService<Sponsor, Sponsorship> {

3.

4. // Internal state

5.

1. @Autowired
2. **private** SponsorSponsorshipRepository repository; 8.
3. @Autowired
4. **private** SystemConfigurationRepository systemConfigurationRepository;
5. // AbstractService interface
6. @Override
7. **public void** authorise() {
8. **boolean** status;
9. **int** id;
10. **int** sponsorId;
11. Sponsorship sponsorship; 21.
12. id = **super**.getRequest().getData("id", **int**.**class**);
13. 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.

1. @Override
2. **public void** load() {
3. 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.

1. @Override
2. **public void** bind(**final** Sponsorship object) { 47. assert object != **null**;

48.

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

49.

50. }

51.

1. @Override
2. **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.

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

66. }

67.

68. **if** (!**super**.getBuffer().getErrors().hasErrors("moment")) { 69.

1. Date sponsorshipDate = object.getMoment();
2. Date minimumDate = MomentHelper.parse("1969-12-31 0:00", "yyyy-MM-dd HH:mm");
3. Date maximumDate = MomentHelper.parse("2200-12-31 23:59", "yyyy-MM-dd HH:mm");
4. **if** (sponsorshipDate != **null**) {
5. Boolean isAfter = sponsorshipDate.after(minimumDate) && sponsorshipDate.before(maximumDate);
6. **super**.state(isAfter, "moment", "sponsor.sponsorship.form.error.moment");

77. }

78. }

79.

1. **if** (!**super**.getBuffer().getErrors().hasErrors("moment")) {
2. Invoice earliestInvoice;
3. Boolean validMoment;
4. Date moment = object.getMoment(); 84.
5. earliestInvoice = **this**.repository.findInvoiceWithEarliestDateBySponsorshipId(object.getI d()).stream().findFirst().orElse(**null**);
6. //System.out.println(earliestInvoice); 87.
7. **if** (earliestInvoice != **null**) {
8. //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.

1. **if** (!**super**.getBuffer().getErrors().hasErrors("durationStartTime")) {
2. Date durationStartTime;
3. Date moment;
4. durationStartTime = object.getDurationStartTime();
5. moment = object.getMoment();
6. Date minimumDate = MomentHelper.parse("1969-12-31 0:00", "yyyy-MM-dd HH:mm");
7. Date maximumDate = MomentHelper.parse("2200-12-31 23:59", "yyyy-MM-dd HH:mm");
8. **if** (moment != **null**)
9. **super**.state(durationStartTime.after(moment) && durationStartTime.after(minimumDate) && durationStartTime.before(maximumDate), "durationStartTime", "sponsor.sponsorship.form.error.durationStartTime");

106. }

107.

1. **if**

(!**super**.getBuffer().getErrors().hasErrors("durationEndTime")) {

1. Date durationStartTime;
2. Date durationEndTime; 111.
3. durationStartTime = object.getDurationStartTime();
4. durationEndTime = object.getDurationEndTime();
5. Date maximumDate = MomentHelper.parse("2200-12-31 23:59", "yyyy-MM-dd HH:mm");
6. **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.

1. **if** (!**super**.getBuffer().getErrors().hasErrors("amount") && **this**.systemConfigurationRepository.existsCurrency(object.getAmount().g etCurrency()))
2. **super**.state(object.getAmount().getAmount() >= 0 && object.getAmount().getAmount() <= 1000000, "amount", "sponsor.sponsorship.form.error.amount");
3. **if** (!**super**.getBuffer().getErrors().hasErrors("amount")) {
4. String symbol = object.getAmount().getCurrency();
5. **boolean** existsCurrency =

**this**.systemConfigurationRepository.existsCurrency(symbol);

1. **super**.state(existsCurrency, "amount", "sponsor.sponsorship.form.error.acceptedCurrency");

128. }

129.

1. **if** (!**super**.getBuffer().getErrors().hasErrors("project"))
2. **super**.state(!object.getProject().isDraftMode(), "project", "sponsor.sponsorship.form.error.project-not-published");
3. **if** (object.getAmount() != **null**) {
4. **double** sumaAmount = 0.0;
5. **for** (Invoice i : allInvoices)
6. sumaAmount += **this**.systemConfigurationRepository.convertToUsd(i.totalAmount()).getAm ount();
7. **if** (**this**.systemConfigurationRepository.existsCurrency(object.getAmount(). getCurrency()))

**super**.state(**this**.systemConfigurationRepository.convertToUsd(object.get Amount()).getAmount() >= sumaAmount, "amount", "sponsor.sponsorship.form.error.amount-less-sum-invoices");

140. }

141.

142. } 143.

1. @Override
2. **public void** perform(**final** Sponsorship object) {
3. assert object != **null**; 147.

148. **this**.repository.save(object);

149. } 150.

1. @Override
2. **public void** unbind(**final** Sponsorship object) {
3. assert object != **null**; 154.

155. SelectChoices choices; 156.

1. Collection<Project> projects =

**this**.repository.findProjects();

1. SelectChoices choices2;
2. Dataset dataset;
3. String projectCode;
4. **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.

1. choices = SelectChoices.from(SponsorshipType.**class**, object.getType());
2. choices2 = SelectChoices.from(projects, "code", s.getProject());
3. dataset = **super**.unbind(object, "code", "moment", "durationStartTime", "durationEndTime", "amount", "type", "email", "link", "project", "draftMode");
4. dataset.put("types", choices);
5. dataset.put("projects", choices2);
6. 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(minimumD ate) && sponsorshipDate.before(maximum Date); | 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(maxim umDate), "durationStartTime", "sponsor.sponsorship.form.erro r.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.erro r.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. |

#### 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.

1. @Autowired
2. **private** SponsorSponsorshipRepository repository; 8.
3. @Autowired
4. **private** SystemConfigurationRepository systemConfigurationRepository;
5. // AbstractService interface
6. @Override
7. **public void** authorise() {
8. **boolean** status;
9. **int** id;
10. **int** sponsorId;
11. Sponsorship sponsorship; 21.
12. id = **super**.getRequest().getData("id", **int**.**class**);
13. 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.

1. @Override
2. **public void** load() {
3. Sponsorship object;
4. **int** id; 36.

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

37.

38.

39.

40. **super**.getBuffer().addData(object); 41. }

42.

1. @Override
2. **public void** bind(**final** Sponsorship object) { 45. assert object != **null**;

46.

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

47.

48. }

49.

1. @Override
2. **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.

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

64. }

65.

1. **if** (!**super**.getBuffer().getErrors().hasErrors("totalAmount") && object.getAmount() != **null** && **this**.systemConfigurationRepository.existsCurrency(object.getAmount().g etCurrency())) {
2. **double** sumaAmount = 0.0;
3. **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.

1. Date sponsorshipDate = object.getMoment();
2. Date minimumDate = MomentHelper.parse("1969-12-31 0:00", "yyyy-MM-dd HH:mm");
3. Date maximumDate = MomentHelper.parse("2200-12-31 23:59", "yyyy-MM-dd HH:mm");
4. **if** (sponsorshipDate != **null**) {
5. Boolean isAfter = sponsorshipDate.after(minimumDate) && sponsorshipDate.before(maximumDate);
6. **super**.state(isAfter, "moment", "sponsor.sponsorship.form.error.moment");

85. }

86. }

87.

1. **if** (!**super**.getBuffer().getErrors().hasErrors("moment")) {
2. Invoice earliestInvoice;
3. Boolean validMoment;
4. Date moment = object.getMoment(); 92.
5. earliestInvoice = **this**.repository.findInvoiceWithEarliestDateBySponsorshipId(object.getI d()).stream().findFirst().orElse(**null**);
6. //System.out.println(earliestInvoice); 95.
7. **if** (earliestInvoice != **null**) {
8. //System.out.println(earliestInvoice); 98. validMoment =

moment.before(earliestInvoice.getRegistrationTime());

1. //System.out.println(validMoment);
2. **super**.state(validMoment, "moment", "sponsor.sponsorship.form.error.creation-moment");

101. }

102. }

103.

1. **if**

(!**super**.getBuffer().getErrors().hasErrors("durationStartTime")) {

1. Date durationStartTime;
2. Date moment;
3. durationStartTime = object.getDurationStartTime();
4. moment = object.getMoment();
5. Date minimumDate = MomentHelper.parse("1969-12-31 0:00", "yyyy-MM-dd HH:mm");
6. Date maximumDate = MomentHelper.parse("2200-12-31 23:59", "yyyy-MM-dd HH:mm");
7. **if** (moment != **null**)
8. **super**.state(durationStartTime.after(moment) && durationStartTime.after(minimumDate) && durationStartTime.before(maximumDate), "durationStartTime", "sponsor.sponsorship.form.error.durationStartTime");

114. }

115.

1. **if**

(!**super**.getBuffer().getErrors().hasErrors("durationEndTime")) {

1. Date durationStartTime;
2. Date durationEndTime; 119.
3. durationStartTime = object.getDurationStartTime();
4. durationEndTime = object.getDurationEndTime();
5. Date maximumDate = MomentHelper.parse("2200-12-31 23:59", "yyyy-MM-dd HH:mm");
6. **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.

1. **if** (!**super**.getBuffer().getErrors().hasErrors("amount") && **this**.systemConfigurationRepository.existsCurrency(object.getAmount().g etCurrency()))
2. **super**.state(object.getAmount().getAmount() >= 0 && object.getAmount().getAmount() <= 1000000, "amount", "sponsor.sponsorship.form.error.amount");
3. **if** (!**super**.getBuffer().getErrors().hasErrors("amount")) {
4. String symbol = object.getAmount().getCurrency();
5. **boolean** existsCurrency =

**this**.systemConfigurationRepository.existsCurrency(symbol);

1. **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.

1. **if** (!**super**.getBuffer().getErrors().hasErrors("project"))
2. **super**.state(!object.getProject().isDraftMode(), "project", "sponsor.sponsorship.form.error.project-not-published");
3. **if** (object.getAmount() != **null**) {
4. **double** sumaAmount = 0.0;
5. **for** (Invoice i : allInvoices)
6. sumaAmount += **this**.systemConfigurationRepository.convertToUsd(i.totalAmount()).getAm ount();
7. **if** (**this**.systemConfigurationRepository.existsCurrency(object.getAmount(). getCurrency()))

**super**.state(**this**.systemConfigurationRepository.convertToUsd(object.get Amount()).getAmount() >= sumaAmount, "amount", "sponsor.sponsorship.form.error.amount-less-sum-invoices");

157. }

158.

159. } 160.

1. @Override
2. **public void** perform(**final** Sponsorship object) {
3. assert object != **null**; 164.
4. object.setDraftMode(**false**);
5. **this**.repository.save(object);

167. } 168.

1. @Override
2. **public void** unbind(**final** Sponsorship object) {
3. assert object != **null**; 172.

173. SelectChoices choices; 174.

1. Collection<Project> projects =

**this**.repository.findProjects();

1. SelectChoices choices2;
2. Dataset dataset;
3. String projectCode;
4. **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.

1. choices = SelectChoices.from(SponsorshipType.**class**, object.getType());
2. choices2 = SelectChoices.from(projects, "code", s.getProject());
3. dataset = **super**.unbind(object, "code", "moment", "durationStartTime", "durationEndTime", "amount", "type", "email", "link", "project", "draftMode");
4. dataset.put("types", choices);
5. dataset.put("projects", choices2);
6. dataset.put("project", projectCode); 194.

195. **super**.getResponse().addData(dataset);

196. } 197.

198. }

|  |  |
| --- | --- |
|  | |
| Línea amarilla (no completamente probada) | Explicación |
| **if** (!**super**.getBuffer().getErrors().hasE rrors("totalAmount") && object.getAmount() != **null** && **this**.systemConfigurationRepository.e xistsCurrency(object.getAmount().get Currency())) { | 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(minimumD ate) && sponsorshipDate.before(maximum Date); | 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 mDate) && durationStartTime.before(maxim umDate), "durationStartTime", "sponsor.sponsorship.form.erro r.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("unpublishedInvoic es")) { | 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.erro r.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. |

#### 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.

1. @Autowired
2. **private** SponsorSponsorshipRepository repository; 8.

9. // AbstractService interface

10.

11.

1. @Override
2. **public void** authorise() {
3. **boolean** status;
4. **int** id;
5. **int** sponsorId;
6. Sponsorship sponsorship; 18.
7. id = **super**.getRequest().getData("id", **int**.**class**);
8. 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.

1. @Override
2. **public void** load() {
3. Sponsorship object;
4. **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.

1. @Override
2. **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.

1. @Override
2. **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.

1. @Override
2. **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.

1. @Override
2. **public void** unbind(**final** Sponsorship object) { 71. assert object != **null**;

72.

73. SelectChoices choices; 74.

1. Collection<Project> projects = **this**.repository.findProjects();
2. SelectChoices choices2;
3. Dataset dataset; 78.
4. choices = SelectChoices.from(SponsorshipType.**class**, object.getType());
5. choices2 = SelectChoices.from(projects, "code", (Project) projects.toArray()[0]);
6. dataset = **super**.unbind(object, "code", "moment", "durationStartTime", "durationEndTime", "amount", "type", "email", "link", "project", "draftMode");
7. 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 |

### Invoice

#### 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.

1. @Autowired
2. **private** SponsorInvoiceRepository repository; 8.

9. // AbstractService interface

10.

11.

1. @Override
2. **public void** authorise() {
3. **boolean** status;
4. **int** sponsorshipId;
5. **int** sponsorId;
6. Sponsorship sponsorship; 18.
7. sponsorshipId = **super**.getRequest().getData("sponsorshipId",

**int**.**class**);

1. sponsorship =

**this**.repository.findOneSponsorshipById(sponsorshipId);

1. sponsorId =

**super**.getRequest().getPrincipal().getActiveRoleId();

1. status = sponsorId == sponsorship.getSponsor().getId(); 25.

26. **super**.getResponse().setAuthorised(status); 27. }

28.

1. @Override
2. **public void** load() {
3. Collection<Invoice> objects;
4. **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.

1. @Override
2. **public void** unbind(**final** Invoice object) { 43. assert object != **null**;

44.

45. Dataset dataset; 46.

1. dataset = **super**.unbind(object, "code", "registrationTime", "dueDate", "quantity", "tax", "link", "draftMode");
2. dataset.put("totalAmount", object.totalAmount()); 49. **super**.getResponse().addData(dataset);

50. }

51.

1. @Override
2. **public void** unbind(**final** Collection<Invoice> objects) { 54. assert objects != **null**;

55.

1. **int** sponsorshipId;
2. Sponsorship sponsorship; 58.
3. sponsorshipId = **super**.getRequest().getData("sponsorshipId",

**int**.**class**);

1. sponsorship =

**this**.repository.findOneSponsorshipById(sponsorshipId);

1. **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 |

#### 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.

1. @Autowired
2. **private** SponsorInvoiceRepository repository; 8.

9. // AbstractService interface

10.

11.

1. @Override
2. **public void** authorise() {
3. **boolean** status;
4. **int** id;
5. **int** sponsorId;
6. Invoice invoice; 18.

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

19.

20.

21.

22. sponsorId =

**super**.getRequest().getPrincipal().getActiveRoleId();

23.

24. status = sponsorId == invoice.getSponsor().getId(); 25.

26. **super**.getResponse().setAuthorised(status); 27. }

28.

1. @Override
2. **public void** load() {
3. Invoice object;
4. **int** id; 33.

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

34.

35.

36.

37. **super**.getBuffer().addData(object); 38. }

39.

1. @Override
2. **public void** unbind(**final** Invoice object) { 42. assert object != **null**;

43.

44. Dataset dataset; 45.

1. dataset = **super**.unbind(object, "code", "registrationTime", "dueDate", "quantity", "tax", "link", "draftMode");
2. dataset.put("totalAmount", object.totalAmount()); 48.

49. **super**.getResponse().addData(dataset); 50. }

51.

52.}

53.

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

#### 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.

1. @Autowired
2. **private** SponsorInvoiceRepository repository; 8.
3. @Autowired
4. **private** SystemConfigurationRepository systemConfigurationRepository;
5. // AbstractService interface
6. @Override
7. **public void** authorise() {
8. **boolean** status;
9. **int** sponsorshipId;
10. **int** sponsorId;
11. Sponsorship sponsorship; 21.
12. sponsorshipId = **super**.getRequest().getData("sponsorshipId",

**int**.**class**);

1. sponsorship =

**this**.repository.findOneSponsorshipById(sponsorshipId);

1. sponsorId =

**super**.getRequest().getPrincipal().getActiveRoleId();

1. status = sponsorId == sponsorship.getSponsor().getId(); 28.

29. **super**.getResponse().setAuthorised(status); 30. }

31.

1. @Override
2. **public void** load() {
3. Invoice object;
4. Integer sponsorshipId;
5. Sponsorship sponsorship; 37.

38. object = **new** Invoice(); 39. Integer sponsorId =

**super**.getRequest().getPrincipal().getActiveRoleId(); 40. Sponsor sponsor =

**this**.repository.findOneSponsorById(sponsorId);

41.

1. sponsorshipId = **super**.getRequest().getData("sponsorshipId",

**int**.**class**);

1. sponsorship =

**this**.repository.findOneSponsorshipById(sponsorshipId);

1. object.setSponsor(sponsor);
2. object.setSponsorship(sponsorship); 47. object.setDraftMode(**true**);

48.

49. **super**.getBuffer().addData(object); 50.

51. }

52.

1. @Override
2. **public void** bind(**final** Invoice object) { 55. assert object != **null**;

56.

57. **super**.bind(object, "code", "registrationTime", "dueDate", "quantity", "tax", "link");

58. }

59.

1. @Override
2. **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.

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

84. }

85. }

86.

1. **if** (!**super**.getBuffer().getErrors().hasErrors("registrationTime")) {
2. Date registrationTime;
3. Date moment; 90.

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

91.

92.

93.

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

96. }

97.

1. **if** (!**super**.getBuffer().getErrors().hasErrors("dueDate")) {
2. Date registrationTime;
3. Date dueDate; 101.
4. registrationTime = object.getRegistrationTime();
5. dueDate = object.getDueDate();
6. Date minimumDate = MomentHelper.parse("1969-12-31 23:59", "yyyy-MM-dd HH:mm");
7. Date maximumDate = MomentHelper.parse("2200-12-31 23:59", "yyyy-MM-dd HH:mm");
8. **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.

1. **if** (!**super**.getBuffer().getErrors().hasErrors("quantity") &&

**this**.systemConfigurationRepository.existsCurrency(object.getQuantity()

.getCurrency()))

1. **super**.state(object.getQuantity().getAmount() >= 0 && object.getQuantity().getAmount() <= 1000000, "quantity", "sponsor.invoice.form.error.quantity");
2. **if** (!**super**.getBuffer().getErrors().hasErrors("quantity")) {
3. String symbol = object.getQuantity().getCurrency();
4. **boolean** existsCurrency =

**this**.systemConfigurationRepository.existsCurrency(symbol);

1. **super**.state(existsCurrency, "quantity", "sponsor.invoice.form.error.acceptedCurrency");

118. }

119.

1. **if**

(!**super**.getBuffer().getErrors().hasErrors("publishedSponsorship")) {

1. Integer sponsorshipId;
2. Sponsorship sponsorship; 123.
3. sponsorshipId =

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

1. sponsorship =

**this**.repository.findOneSponsorshipById(sponsorshipId); 126.

127. **super**.state(sponsorship.isDraftMode(), "\*", "sponsor.invoice.form.error.published-sponsorship");

128. }

129.

1. **if** (!**super**.getBuffer().getErrors().hasErrors()) {
2. **double** sumaNueva = **this**.systemConfigurationRepository.convertToUsd(object.totalAmount()). getAmount();
3. **for** (Invoice i : allInvoices)
4. sumaNueva += **this**.systemConfigurationRepository.convertToUsd(i.totalAmount()).getAm ount();
5. **if**

(**this**.systemConfigurationRepository.existsCurrency(object.getQuantity(

).getCurrency()))

1. **super**.state(sumaNueva <=

**this**.systemConfigurationRepository.convertToUsd(object.getSponsorship(

).getAmount()).getAmount(), "\*", "sponsor.invoice.form.error.incorrect-sum");

137. }

138.

139. } 140.

1. @Override
2. **public void** perform(**final** Invoice object) {
3. assert object != **null**; 144.
4. **if** (object.getTax() == **null**)
5. object.setTax(0.0); 147.

148. **this**.repository.save(object);

149. } 150.

1. @Override
2. **public void** unbind(**final** Invoice object) {
3. assert object != **null**; 154.

155. Dataset dataset;

156.

1. dataset = **super**.unbind(object, "code", "registrationTime", "dueDate", "quantity", "tax", "link", "draftMode");
2. 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.isLongEnoug h(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().hasE rrors("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. |

#### 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

1. **public class** SponsorInvoiceUpdateService **extends** AbstractService<Sponsor, Invoice> {

2.

3. // Internal state

4.

1. @Autowired
2. **private** SponsorInvoiceRepository repository; 7.
3. @Autowired
4. **private** SystemConfigurationRepository systemConfigurationRepository;
5. // AbstractService interface
6. @Override
7. **public void** authorise() {
8. **boolean** status;
9. **int** id;
10. **int** sponsorId;
11. Invoice invoice; 20.

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

21.

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.

1. @Override
2. **public void** load() {
3. Invoice object;
4. **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.

1. @Override
2. **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.

1. @Override
2. **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.

64.

**this**.repository.findOneInvoiceByCode(object.getCode());

Invoice projectSameCode =

1. **if** (projectSameCode != **null**)
2. **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**) {

1. Boolean isAfter = registrationTime.after(minimumDate) && registrationTime.before(maximumDate);
2. **super**.state(isAfter, "registrationTime", "sponsor.invoice.form.error.registration-time");

78. }

79. }

80.

1. **if** (!**super**.getBuffer().getErrors().hasErrors("registrationTime")) {
2. Date registrationTime;
3. Date moment; 84.

registrationTime = object.getRegistrationTime();

85.

86. moment = object.getSponsorship().getMoment(); 87.

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

90. }

91.

1. **if** (!**super**.getBuffer().getErrors().hasErrors("dueDate")) {
2. Date registrationTime;
3. Date dueDate; 95.

96. registrationTime = object.getRegistrationTime(); 97. dueDate = object.getDueDate();

1. Date minimumDate = MomentHelper.parse("1969-12-31 23:59", "yyyy-MM-dd HH:mm");
2. Date maximumDate = MomentHelper.parse("2200-12-31 23:59", "yyyy-MM-dd HH:mm");
3. **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");

103. }

104.

105.

**if** (!**super**.getBuffer().getErrors().hasErrors("quantity") &&

**this**.systemConfigurationRepository.existsCurrency(object.getQuantity()

.getCurrency()))

106. **super**.state(object.getQuantity().getAmount() >= 0 && object.getQuantity().getAmount() <= 1000000, "quantity", "sponsor.invoice.form.error.quantity");

107.

1. **if** (!**super**.getBuffer().getErrors().hasErrors("quantity")) {
2. String symbol = object.getQuantity().getCurrency();
3. **boolean** existsCurrency =

**this**.systemConfigurationRepository.existsCurrency(symbol);

1. **super**.state(existsCurrency, "quantity", "sponsor.invoice.form.error.acceptedCurrency");

112. }

113.

1. **if**

(!**super**.getBuffer().getErrors().hasErrors("publishedSponsorship"))

1. **super**.state(object.getSponsorship().isDraftMode(), "\*", "sponsor.invoice.form.error.published-sponsorship");
2. **if** (!**super**.getBuffer().getErrors().hasErrors()) {
3. **double** valorAntiguo = **this**.systemConfigurationRepository.convertToUsd(**this**.repository.findOn eInvoiceById(object.getId()).totalAmount()).getAmount();
4. **double** sumaNueva = **this**.systemConfigurationRepository.convertToUsd(object.totalAmount()). getAmount() - valorAntiguo;
5. **for** (Invoice i : allInvoices)
6. sumaNueva += **this**.systemConfigurationRepository.convertToUsd(i.totalAmount()).getAm ount();
7. **if**

(**this**.systemConfigurationRepository.existsCurrency(object.getQuantity(

).getCurrency()))

1. **super**.state(sumaNueva <=

**this**.systemConfigurationRepository.convertToUsd(object.getSponsorship(

).getAmount()).getAmount(), "\*", "sponsor.invoice.form.error.incorrect-sum");

125. }

126.

127. } 128.

1. @Override
2. **public void** perform(**final** Invoice object) {
3. assert object != **null**; 132.
4. **if** (object.getTax() == **null**)
5. object.setTax(0.0); 135.

136. **this**.repository.save(object);

137. } 138.

1. @Override
2. **public void** unbind(**final** Invoice object) {
3. assert object != **null**; 142.

143. Dataset dataset; 144.

1. dataset = **super**.unbind(object, "code", "registrationTime", "dueDate", "quantity", "tax", "link", "draftMode");
2. 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.isLongEnoug h(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().hasE rrors("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. |

#### 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

1. **public class** SponsorInvoicePublishService **extends** AbstractService<Sponsor, Invoice> {

2.

3. // Internal state

4.

1. @Autowired
2. **private** SponsorInvoiceRepository repository; 7.
3. @Autowired
4. **private** SystemConfigurationRepository systemConfigurationRepository;
5. // AbstractService interface
6. @Override
7. **public void** authorise() {
8. **boolean** status;
9. **int** id;
10. **int** sponsorId;
11. Invoice invoice; 20.

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

21.

22.

23.

24. sponsorId =

**super**.getRequest().getPrincipal().getActiveRoleId();

25.

26. status = sponsorId == invoice.getSponsor().getId() && invoice.isDraftMode();

27.

28. **super**.getResponse().setAuthorised(status); 29. }

30.

1. @Override
2. **public void** load() {
3. Invoice object;
4. **int** id; 35.

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

36.

37.

38.

39. **super**.getBuffer().addData(object); 40. }

41.

1. @Override
2. **public void** bind(**final** Invoice object) { 44. assert object != **null**;

45.

46. **super**.bind(object, "code", "registrationTime", "dueDate", "quantity", "tax", "link");

47. }

48.

1. @Override
2. **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.

1. **if** (projectSameCode != **null**)
2. **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.

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

74. }

75. }

76.

1. **if** (!**super**.getBuffer().getErrors().hasErrors("registrationTime")) {
2. Date registrationTime;
3. Date moment; 80.

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

81.

82.

83.

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

86. }

87.

1. **if** (!**super**.getBuffer().getErrors().hasErrors("dueDate")) {
2. Date registrationTime;
3. Date dueDate; 91.

92. registrationTime = object.getRegistrationTime(); 93. dueDate = object.getDueDate();

1. Date minimumDate = MomentHelper.parse("1969-12-31 23:59", "yyyy-MM-dd HH:mm");
2. Date maximumDate = MomentHelper.parse("2200-12-31 23:59", "yyyy-MM-dd HH:mm");
3. **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.

1. **if** (!**super**.getBuffer().getErrors().hasErrors("quantity") &&

**this**.systemConfigurationRepository.existsCurrency(object.getQuantity()

.getCurrency()))

1. **super**.state(object.getQuantity().getAmount() >= 0 && object.getQuantity().getAmount() <= 1000000, "quantity", "sponsor.invoice.form.error.quantity");
2. **if** (!**super**.getBuffer().getErrors().hasErrors("quantity")) {
3. String symbol = object.getQuantity().getCurrency();
4. **boolean** existsCurrency =

**this**.systemConfigurationRepository.existsCurrency(symbol);

1. **super**.state(existsCurrency, "quantity", "sponsor.invoice.form.error.acceptedCurrency");

108. }

109.

1. **if** (!**super**.getBuffer().getErrors().hasErrors()) {
2. **double** valorAntiguo =

**this**.systemConfigurationRepository.convertToUsd(**this**.repository.findOn

eInvoiceById(object.getId()).totalAmount()).getAmount();

1. **double** sumaNueva = **this**.systemConfigurationRepository.convertToUsd(object.totalAmount()). getAmount() - valorAntiguo;
2. **for** (Invoice i : allInvoices)
3. sumaNueva += **this**.systemConfigurationRepository.convertToUsd(i.totalAmount()).getAm ount();
4. **if**

(**this**.systemConfigurationRepository.existsCurrency(object.getQuantity(

).getCurrency()))

1. **super**.state(sumaNueva <=

**this**.systemConfigurationRepository.convertToUsd(object.getSponsorship(

).getAmount()).getAmount(), "\*", "sponsor.invoice.form.error.incorrect-sum");

118. }

119.

120. } 121.

1. @Override
2. **public void** perform(**final** Invoice object) {
3. assert object != **null**; 125.
4. **if** (object.getTax() == **null**)
5. object.setTax(0.0); 128.
6. object.setDraftMode(**false**);
7. **this**.repository.save(object);

131. } 132.

1. @Override
2. **public void** unbind(**final** Invoice object) {
3. assert object != **null**; 136.

137. Dataset dataset; 138.

1. dataset = **super**.unbind(object, "code", "registrationTime", "dueDate", "quantity", "tax", "link", "draftMode");
2. dataset.put("totalAmount", object.totalAmount()); 141.

142. **super**.getResponse().addData(dataset);

143. } 144.

145. }

|  |  |
| --- | --- |
| 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.isLongEnoug h(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. |

#### 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.

1. @Autowired
2. **private** SponsorInvoiceRepository repository; 8.

9. // AbstractService interface

10.

11.

1. @Override
2. **public void** authorise() {
3. **boolean** status;
4. **int** id;
5. **int** sponsorId;
6. Invoice invoice; 18.

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

19.

20.

21.

22. sponsorId =

**super**.getRequest().getPrincipal().getActiveRoleId();

23.

24. status = sponsorId == invoice.getSponsor().getId() && invoice.isDraftMode();

25.

26. **super**.getResponse().setAuthorised(status); 27. }

28.

1. @Override
2. **public void** load() {
3. Invoice object;
4. **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.

1. @Override
2. **public void** bind(**final** Invoice object) { 43. assert object != **null**;

44.

45. **super**.bind(object, "code", "registrationTime", "dueDate", "quantity", "tax", "link");

46. }

47.

1. @Override
2. **public void** validate(**final** Invoice object) { 50. assert object != **null**;

51.

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

54. }

55.

1. @Override
2. **public void** perform(**final** Invoice object) { 58. assert object != **null**;

59.

60. **this**.repository.**delete**(object); 61. }

62.

1. @Override
2. **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.}

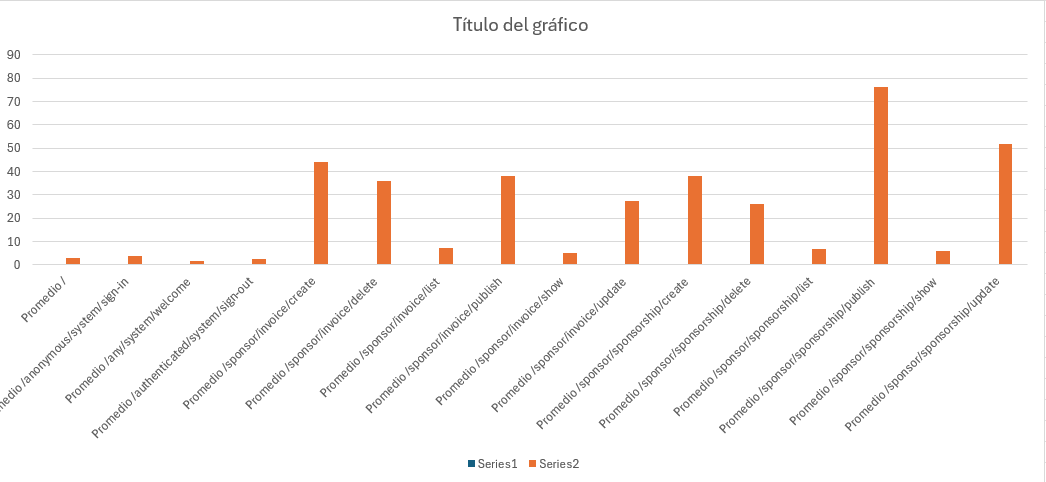
75.

|  |  |
| --- | --- |
| Línea amarilla (no completamente probada) | Explicación |
| **if** (!**super**.getBuffer().getErrors().hasE rrors("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. |

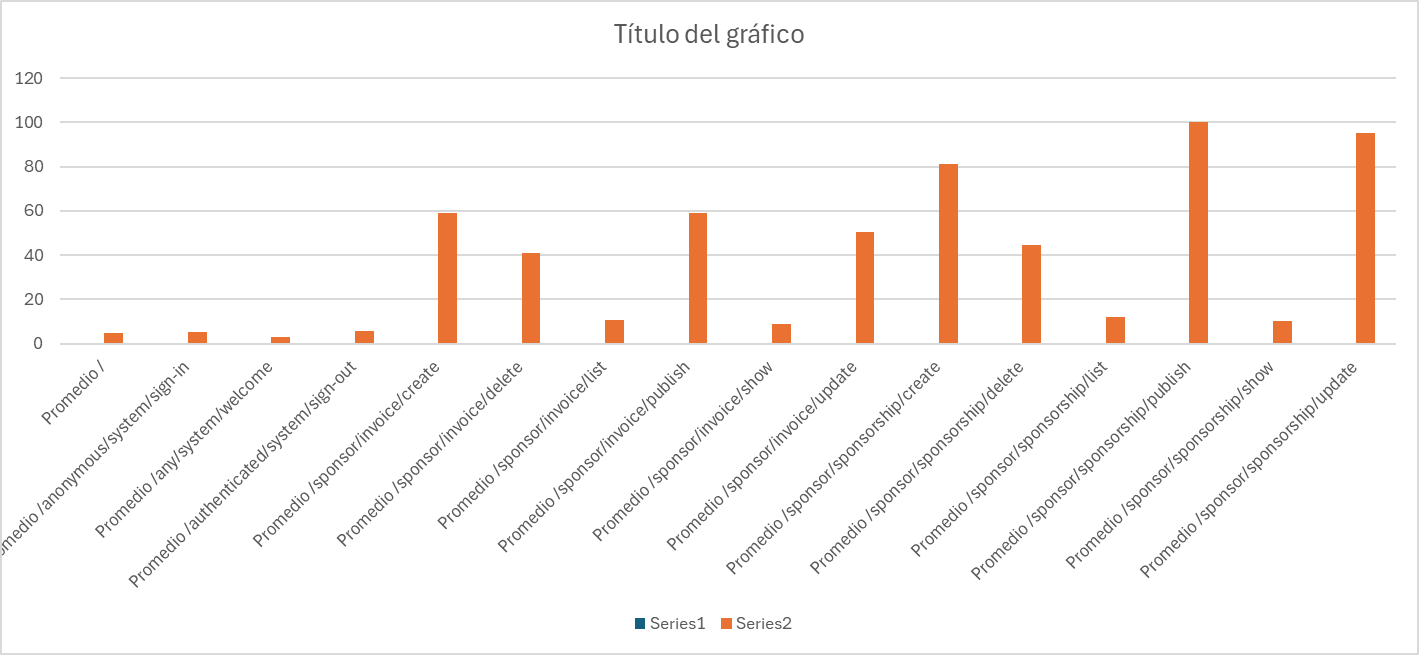
## Performance testing

### Comparativa entre dos PCs

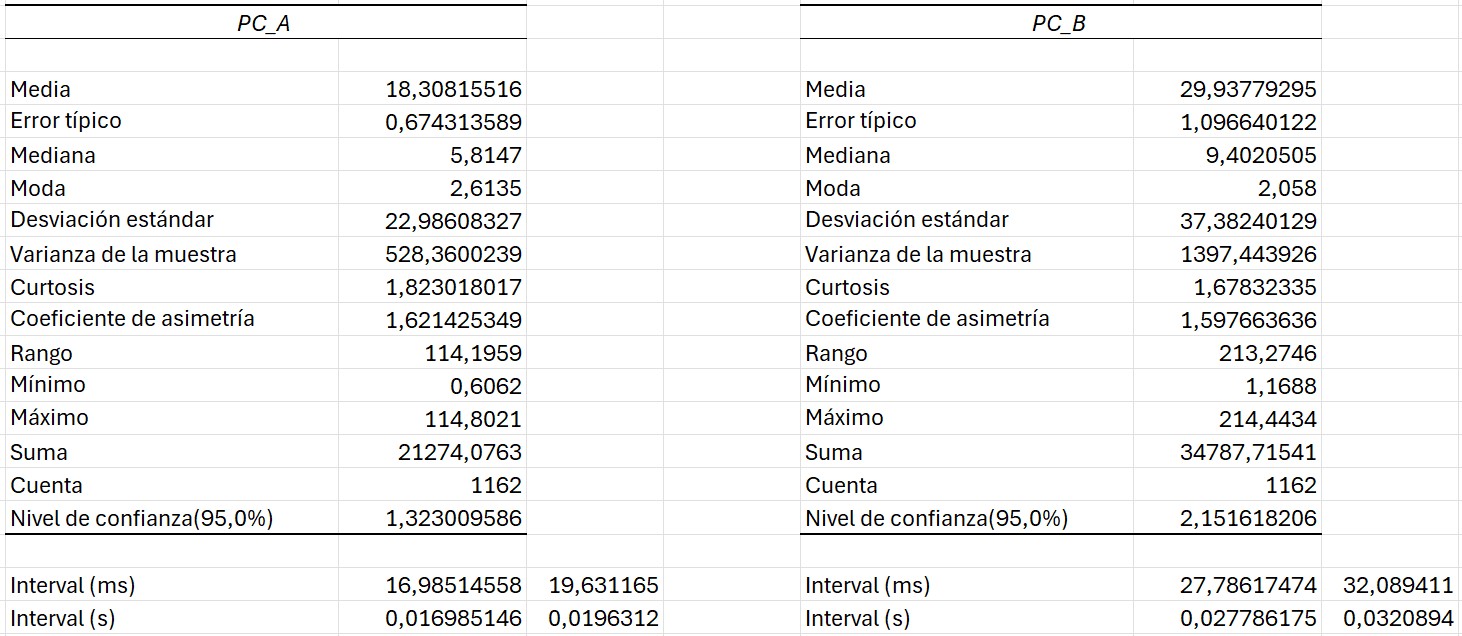
###### PC\_A



###### PC\_B

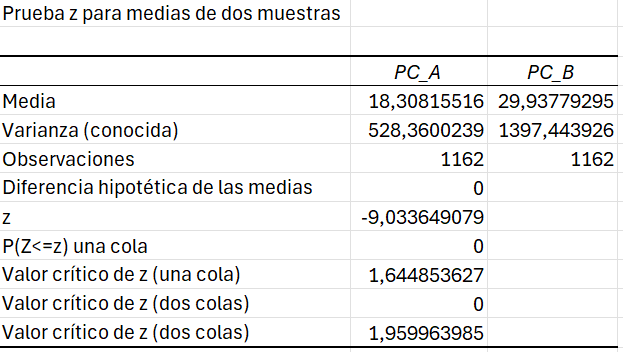


###### Se puede ver que el PC\_B tiene peor rendimiento que el PC\_A, pero están relativamente parejos.



###### 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.



###### 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.

# 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.