Testing report

Diseño y pruebas II

D04

Versión 1.0

https://github.com/DP2-c1-028/Acme-SF-D04

Fecha 22/06/2024

Preparado por:
C2.028

1.Introducción

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

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

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

2. Functional testing

2.1. Testeo de funcionalidades de Banner

2.1.1. List

2.1.1.1. list.safe

Esta prueba se llevó a cabo realizando un listado de los banner. Funcionó como se esperaba y no se encontró ningún bug.

2.1.1.2. list.hack

Para hacer esta prueba se ha iniciado la aplicación y sin registrarse en ningún usuario, se ha intentado acceder a la siguiente url: http://localhost:8082/acme-sf-d04/administrator/banner/list.
Provocó el error esperado (500 “access is not authorised”) y no se encontró ningún bug.

2.1.1.3. Cobertura

```
1. @Service
2. public class AdministratorBannerListService extends
   AbstractService<Administrator, Banner> {
3. 
4.     // Internal state
   ----------------------------------------------------------
5. 
6.     @Autowired
7.     private AdministratorBannerRepository repository;
8. 
9.     // AbstractService interface
   -------------------------------------------
10. 
11. 
12.     @Override
13.     public void authorise() {
14.         super.getResponse().setAuthorised(true);
15.     }
16. 
17.     @Override
18.     public void load() {
19.         Collection<Banner> objects;
20. 
21.         objects = this.repository.findAllBanners();
22. 
23.         super.getBuffer().addData(objects);
24.     }
25. 
26.     @Override
27.     public void unbind(final Banner object) {
28.         assert object != null;
```

```
29.
30.         Dataset dataset;
31.         dataset = super.unbind(object, "instantiationMoment",
    "bannerStartTime", "bannerEndTime", "picture", "slogan", "link");
32.
33.         super.getResponse().addData(dataset);
34.     }
35.
36.}
```

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

2.1.2. Show

2.1.2.1. show.safe

Esta prueba se llevó a cabo accediendo a todos los banners existentes. Funcionó como se esperaba y no se encontró ningún bug.

2.1.2.2. show.hack

Para hacer esta prueba se ha iniciado la aplicación y sin registrarse en ningún usuario, se ha intentado acceder a la siguiente url: http://localhost:8082/acme-sf-d04/administrator/banner/show?id=167.
Provocó el error esperado (500 “access is not authorised”) y no se encontró ningún bug.

2.1.2.3 Cobertura

```
1. @Service
2. public class AdministratorBannerShowService extends
   AbstractService<Administrator, Banner> {
3.
4.     // Internal state
   ---------------------------------------------------------
5.
6.     @Autowired
7.     private AdministratorBannerRepository repository;
8.
9.     // AbstractService interface
   ---------------------------------------------
10.
11.
12.    @Override
13.    public void authorise() {
14.        super.getResponse().setAuthorised(true);
15.    }
16.
```

```
17.     @Override
18.     public void load() {
19.         Banner object;
20.         int id;
21.
22.         id = super.getRequest().getData("id", int.class);
23.         object = this.repository.findOneBannerById(id);
24.
25.         super.getBuffer().addData(object);
26.     }
27.
28.     @Override
29.     public void unbind(final Banner object) {
30.         assert object != null;
31.
32.         Dataset dataset;
33.         dataset = super.unbind(object, "instantiationMoment",
    "bannerStartTime", "bannerEndTime", "picture", "slogan", "link");
34.         super.getResponse().addData(dataset);
35.     }
36.
37. }
```

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

2.1.3. Create

2.1.3.1. create.safe

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

2.1.3.2. create.hack

Para hacer esta prueba se ha iniciado la aplicación y sin registrarse en ningún usuario, se ha intentado acceder a la siguiente url: http://localhost:8082/acme-sf-d04/administrator/banner/create.
Provocó el error esperado (500 “access is not authorised”) y no se encontró ningún bug.

2.1.3.3. Cobertura

```
1. @Service
2. public class AdministratorBannerCreateService extends
   AbstractService<Administrator, Banner> {
3. 
4.     // Internal state
   ----------------------------------------------------------
5. 
6.     @Autowired
7.     private AdministratorBannerRepository repository;
8. 
9.     // AbstractService interface
   ---------------------------------------------
10. 
11. 
12.     @Override
13.     public void authorise() {
14.         super.getResponse().setAuthorised(true);
15.     }
16. 
17.     @Override
18.     public void load() {
19.         Banner object;
20.         Date instantiationMoment;
21.         Date currentMoment;
22. 
23.         currentMoment = MomentHelper.getCurrentMoment();
24.         instantiationMoment = new Date(currentMoment.getTime() -
   1000);
25. 
26.         object = new Banner();
27.         object.setInstantiationMoment(instantiationMoment);
28. 
29.         super.getBuffer().addData(object);
30.     }
31. 
32.     @Override
33.     public void bind(final Banner object) {
34.         assert object != null;
35. 
36.         super.bind(object, "bannerStartTime", "bannerEndTime",
   "picture", "slogan", "link");
37.     }
38. 
39.     @Override
40.     public void validate(final Banner object) {
41.         assert object != null;
42. 
43.         if
   (!super.getBuffer().getErrors().hasErrors("instantiationMoment")) {
44. 
45.             Date instantiationMoment =
   object.getInstantiationMoment();
```

```
46.             Date maximumDate = MomentHelper.parse("2100-01-01 00:00",
    "yyyy-MM-dd HH:mm");
47.             Date minimumDate = MomentHelper.parse("1969-12-31 23:59",
    "yyyy-MM-dd HH:mm");
48.
49.             if (instantiationMoment != null) {
50.                 Boolean isAfter =
    instantiationMoment.after(minimumDate) &&
    instantiationMoment.before(maximumDate);
51.                 super.state(isAfter, "instantiationMoment",
    "administrator.banner.form.error.instantiation-moment");
52.             }
53.         }
54.
55.         if
    (!super.getBuffer().getErrors().hasErrors("bannerStartTime") &&
    object.getBannerStartTime() != null) {
56.             Date bannerStartTime;
57.             Date instantiationMoment;
58.             bannerStartTime = object.getBannerStartTime();
59.             instantiationMoment = object.getInstantiationMoment();
60.
61.             super.state(bannerStartTime.after(instantiationMoment),
    "bannerStartTime",
    "administrator.banner.form.error.banner-start-time");
62.         }
63.
64.         if (!super.getBuffer().getErrors().hasErrors("bannerEndTime")
    && object.getBannerEndTime() != null) {
65.             Date bannerStartTime;
66.             Date bannerEndTime;
67.
68.             bannerStartTime = object.getBannerStartTime();
69.             bannerEndTime = object.getBannerEndTime();
70.             Date maximumDate = MomentHelper.parse("2100-01-01 00:00",
    "yyyy-MM-dd HH:mm");
71.
72.             if (bannerStartTime != null && bannerEndTime != null)
73.                 super.state(MomentHelper.isLongEnough(bannerStartTime,
    bannerEndTime, 1, ChronoUnit.WEEKS) &&
    bannerEndTime.after(bannerStartTime) &&
    bannerEndTime.before(maximumDate), "bannerEndTime",
    "administrator.banner.form.error.banner-end-time");
74.         }
75.
76.         if
    (!super.getBuffer().getErrors().hasErrors("bannerStartTime")) {
77.
78.             Date bannerDate = object.getBannerStartTime();
79.             Date maximumDate = MomentHelper.parse("2100-01-01 00:00",
    "yyyy-MM-dd HH:mm");
80.             Date minimumDate = MomentHelper.parse("1969-12-31 23:59",
    "yyyy-MM-dd HH:mm");
```

```
81.
82.             Boolean isBefore = bannerDate.after(minimumDate) &&
    bannerDate.before(maximumDate);
83.             super.state(isBefore, "bannerStartTime",
    "administrator.banner.form.error.banner-start-time-maximum");
84.         }
85.     }
86.
87.     @Override
88.     public void perform(final Banner object) {
89.         assert object != null;
90.
91.         this.repository.save(object);
92.     }
93.
94.     @Override
95.     public void unbind(final Banner object) {
96.         assert object != null;
97.
98.         Dataset dataset;
99.
100.            dataset = super.unbind(object, "instantiationMoment",
    "bannerStartTime", "bannerEndTime", "picture", "slogan", "link");
101.
102.            super.getResponse().addData(dataset);
103.        }
104.
105.    }
```

Línea amarilla (no completamente probada)	Explicación
`if (!super.getBuffer().getErrors().hasErrors("instantiationMoment")) {`	instantiationMoment se maneja de manera automática. En la creación, instantiationMoment no recibe valor y nunca se entra al if en los casos que podemos probar, pero se valida igualmente para evitar POST hacking.
`if (instantiationMoment != null) {` `Boolean isAfter = instantiationMoment.after(minimumDate) && instantiationMoment.before(maximumDate);`	Misma explicación que en el caso anterior.
`if (!super.getBuffer().getErrors().hasErrors("bannerStartTime") &&`	En este if nunca se va a dar el caso de que ambos estén vacíos, y el resto han sido correctamente probados.

`object.getBannerStartTime() != null) {`	
`if (!super.getBuffer().getErrors().hasErrors("bannerEndTime") && object.getBannerEndTime() != null) {`	Misma explicación que en el caso anterior.
`if (bannerStartTime != null && bannerEndTime != null)`	Misma explicación que en el caso anterior.
`Boolean isBefore = bannerDate.after(minimumDate) && bannerDate.before(maximumDate);`	El único caso que no se puede comprobar que la fecha esté tanto por encima como por debajo de los límites establecidos. El resto de casos han sido correctamente probados.

2.1.4. Update

2.1.4.1. update.safe

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

2.1.4.3. Cobertura

```
1.  @Service
2.  public class AdministratorBannerUpdateService extends
    AbstractService<Administrator, Banner> {
3.
4.      // Internal state
    ----------------------------------------------------------
5.
6.      @Autowired
7.      private AdministratorBannerRepository repository;
8.
9.      // AbstractService interface
    ---------------------------------------------
10.
11.
12.     @Override
13.     public void authorise() {
14.         super.getResponse().setAuthorised(true);
15.     }
16.
17.     @Override
18.     public void load() {
19.         Banner object;
20.         int id;
```

```
21.
22.         id = super.getRequest().getData("id", int.class);
23.         object = this.repository.findOneBannerById(id);
24.
25.         Date instantiationMoment;
26.         instantiationMoment = MomentHelper.getCurrentMoment();
27.         object.setInstantiationMoment(instantiationMoment);
28.
29.         super.getBuffer().addData(object);
30.     }
31.
32.     @Override
33.     public void bind(final Banner object) {
34.         assert object != null;
35.
36.         super.bind(object, "bannerStartTime", "bannerEndTime",
    "picture", "slogan", "link");
37.         Date instantiationMoment;
38.         Date currentMoment;
39.
40.         currentMoment = MomentHelper.getCurrentMoment();
41.         instantiationMoment = new Date(currentMoment.getTime() -
    1000);
42.         object.setInstantiationMoment(instantiationMoment);
43.     }
44.
45.     @Override
46.     public void validate(final Banner object) {
47.         assert object != null;
48.
49.         if
    (!super.getBuffer().getErrors().hasErrors("instantiationMoment")) {
50.
51.             Date instantiationMoment =
    object.getInstantiationMoment();
52.             Date maximumDate = MomentHelper.parse("2100-01-01 00:00",
    "yyyy-MM-dd HH:mm");
53.             Date minimumDate = MomentHelper.parse("1969-12-31 23:59",
    "yyyy-MM-dd HH:mm");
54.
55.             if (instantiationMoment != null) {
56.                 Boolean isAfter =
    instantiationMoment.after(minimumDate) &&
    instantiationMoment.before(maximumDate);
57.                 super.state(isAfter, "instantiationMoment",
    "administrator.banner.form.error.instantiation-moment");
58.             }
59.         }
60.
61.         if
    (!super.getBuffer().getErrors().hasErrors("bannerStartTime") &&
    object.getBannerStartTime() != null) {
62.             Date bannerStartTime;
```

```
63.             Date instantiationMoment;
64.             bannerStartTime = object.getBannerStartTime();
65.             instantiationMoment = object.getInstantiationMoment();
66.
67.             super.state(bannerStartTime.after(instantiationMoment),
    "bannerStartTime",
    "administrator.banner.form.error.banner-start-time");
68.         }
69.
70.         if (!super.getBuffer().getErrors().hasErrors("bannerEndTime")
    && object.getBannerEndTime() != null) {
71.             Date bannerStartTime;
72.             Date bannerEndTime;
73.
74.             bannerStartTime = object.getBannerStartTime();
75.             bannerEndTime = object.getBannerEndTime();
76.             Date maximumDate = MomentHelper.parse("2100-01-01 00:00",
    "yyyy-MM-dd HH:mm");
77.
78.             if (bannerStartTime != null && bannerEndTime != null)
79.                 super.state(MomentHelper.isLongEnough(bannerStartTime,
    bannerEndTime, 1, ChronoUnit.WEEKS) &&
    bannerEndTime.after(bannerStartTime) &&
    bannerEndTime.before(maximumDate), "bannerEndTime",
    "administrator.banner.form.error.banner-end-time");
80.         }
81.
82.         if
    (!super.getBuffer().getErrors().hasErrors("bannerStartTime")) {
83.
84.             Date bannerDate = object.getBannerStartTime();
85.             Date maximumDate = MomentHelper.parse("2100-01-01 00:00",
    "yyyy-MM-dd HH:mm");
86.             Date minimumDate = MomentHelper.parse("1969-12-31 23:59",
    "yyyy-MM-dd HH:mm");
87.
88.             Boolean isBefore = bannerDate.after(minimumDate) &&
    bannerDate.before(maximumDate);
89.             super.state(isBefore, "bannerStartTime",
    "administrator.banner.form.error.banner-start-time-maximum");
90.         }
91.     }
92.
93.     @Override
94.     public void perform(final Banner object) {
95.         assert object != null;
96.         this.repository.save(object);
97.     }
98.
99.     @Override
100.        public void unbind(final Banner object) {
101.            assert object != null;
102.            Dataset dataset;
```

```
103.            dataset = super.unbind(object, "bannerStartTime",
    "bannerEndTime", "picture", "slogan", "link");
104.            dataset.put("instantiationMoment",
    MomentHelper.getCurrentMoment());
105.
106.            super.getResponse().addData(dataset);
107.        }
108.
109.    }
```

Línea amarilla (no completamente probada)	Explicación
`if (!super.getBuffer().getErrors().hasErrors("instantiationMoment")) {`	instantiationMoment se maneja de manera automática. En la creación, instantiationMoment no recibe valor y nunca se entra al if en los casos que podemos probar, pero se valida igualmente para evitar POST hacking.
`3. if (instantiationMoment != null) {` `4. Boolean isAfter = instantiationMoment.after(minimumDate) && instantiationMoment.before(maximumDate);`	Misma explicación que en el caso anterior.
`if (!super.getBuffer().getErrors().hasErrors("bannerStartTime") && object.getBannerStartTime() != 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("bannerEndTime") && object.getBannerEndTime() != null) {`	Misma explicación que en el caso anterior.
`if (bannerStartTime != null && bannerEndTime != null)`	Misma explicación que en el caso anterior.
`Boolean isBefore = bannerDate.after(minimumDate) && bannerDate.before(maximumDate);`	El único caso que no se puede comprobar que la fecha esté tanto por encima como por debajo de los límites establecidos. El resto de casos han sido correctamente probados.

2.1.5. Delete

2.1.5.1. delete.safe

Esta prueba se llevó a cabo probando a borrar algunos de los banners existentes. Todo funcionó según lo esperado y no se encontró ningún bug.

2.1.5.3. Cobertura

```
1. @Service
2. public class AdministratorBannerDeleteService extends
   AbstractService<Administrator, Banner> {
3. 
4.     // Internal state
   ----------------------------------------------------------
5. 
6.     @Autowired
7.     private AdministratorBannerRepository repository;
8. 
9.     // AbstractService interface
   ---------------------------------------------
10. 
11. 
12.     @Override
13.     public void authorise() {
14.         super.getResponse().setAuthorised(true);
15.     }
16. 
17.     @Override
18.     public void load() {
19.         Banner object;
20.         int id;
21. 
22.         id = super.getRequest().getData("id", int.class);
23.         object = this.repository.findOneBannerById(id);
24.         super.getBuffer().addData(object);
25.     }
26. 
27.     @Override
28.     public void bind(final Banner object) {
29.         assert object != null;
30. 
31.         super.bind(object, "instantiationMoment", "bannerStartTime",
   "bannerEndTime", "picture", "slogan", "link");
32.     }
33. 
34.     @Override
35.     public void validate(final Banner object) {
36.         assert object != null;
37.     }
38. 
39.     @Override
40.     public void perform(final Banner object) {
```

```
41.         assert object != null;
42.         this.repository.delete(object);
43.     }
44.
45.     @Override
46.     public void unbind(final Banner object) {
47.         assert object != null;
48.         Dataset dataset;
49.         dataset = super.unbind(object, "instantiationMoment",
    "bannerStartTime", "bannerEndTime", "picture", "slogan", "link");
50.         super.getResponse().addData(dataset);
51.     }
52.
53. }
```

La cobertura es completa y tal y como se esperaba.

3. Performance testing

3.1. Contraste de hipótesis

Se ha ejecutado la misma suite de tests en dos equipos distintos. El equipo 1 corresponde al de Gonzalo Navas Remmers y el equipo 2 al de Pablo Jesús Castellanos Compaña, miembros de este grupo.

Se muestran a continuación las estadísticas descriptivas de los análisis para cada equipo:

Equipo 1				*Equipo 2*		
Media	14.81090638			Media	21.56217039	
Error típico	0.387933072			Error típico	0.593826293	
Mediana	8.96750375			Mediana	11.660015	
Moda	27.967051			Moda	10.802355	
Desviación estándar	13.44958982			Desviación estándar	21.29511098	
Varianza de la muestra	180.8914662			Varianza de la muestra	453.4817515	
Curtosis	4.575910459			Curtosis	2.038091479	
Coeficiente de asimetría	1.682045491			Coeficiente de asimetría	1.503019097	
Rango	125.637596			Rango	159.819165	
Mínimo	0.714872			Mínimo	1.010895	
Máximo	126.352468			Máximo	160.83006	
Suma	17802.70947			Suma	27728.95112	
Cuenta	1202			Cuenta	1286	
Nivel de confianza(95.0%)	0.761101873			Nivel de confianza(95.0%)	1.164975442	
Interval (ms)	14.04980451	15.57200826		Interval (ms)	20.39719495	22.72714583
Interval (s)	0.014049805	0.015572008		Interval (s)	0.020397195	0.022727146

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

	Equipo 1	*Equipo 2*
Media	14.81090638	21.56217039
Varianza (conocida)	180.8914662	453.4817515
Observaciones	1202	1286
Diferencia hipotética de las medias	0	
z	-9.518062481	
P(Z<=z) una cola	0	
Valor crítico de z (una cola)	1.644853627	
Valor crítico de z (dos colas)	0	
Valor crítico de z (dos colas)	1.959963985	

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

3.2. Conclusión del performance testing

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

Nos ha resultado un proceso tremendamente útil y una manera muy interesante de plasmar de manera objetiva el rendimiento de un sistema y compararlo tras realizar cambios en las variables que pueden influir en ello.

4. Conclusiones generales

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