



TESTING REPORT

[GROUP] Grupo C2.052

<https://github.com/BVP2455/Acme-ANS-C2>

José Luis Cegrí Marcos

joscegmar@alum.us.es

Tabla de contenidos

1. Pruebas funcionales
2. Pruebas de rendimiento

Resumen ejecutivo

En este Testing Report, presentamos las pruebas realizadas para las funcionalidades grupales, tanto funcionales como de rendimiento, para las funcionalidades de Aeropuertos (Airport) correspondientes a las tareas del administrador (Administrator) del proyecto Acme-ANS-C2.

Tabla de revisión

Revisión	Fecha	Descripción corta
V1.0	02/07/2025	Primera versión del proyecto con los datos completados

Índice

1.	Pruebas funcionales	4
1.1	Introducción	4
1.2	Cobertura de las funcionalidades de los aeropuertos (Airport)	6
1.2.1	AdministratorAirportListService	6
1.2.2	AdministratorAirportShowService	7
1.2.3	AdministratorAirportCreateService	8
1.2.4	AdministratorAirportUpdateService	9
2.	Pruebas de rendimiento	10
3.	Conclusión	13

1. Pruebas funcionales

1.1 Introducción








En las pruebas funcionales, hemos utilizado la herramienta record que nos proporciona el framework Acme-Framework-25.6.0 para probar cada una de la funcionalidad de Airport. La herramienta carga el proyecto en la dirección <http://localhost:8082/Acme-ANS-C2?debug=true&locale=en> y registra cada una de las peticiones que realizamos al servidor.

Aprovechando esto, hemos realizado un testing exhaustivo a cada una de las funcionalidades que se implementan en Airport: listado de aeropuertos (AdministratorAirportListService.java), mostrado de aeropuertos (AdministratorAirportShowService.java), creación de aeropuertos (AdministratorAirportCreateService.java), actualización de aeropuertos (AdministratorAirportUpdateService.java).













Las pruebas formales a las funcionalidades que han probado los casos positivos (que devolvían el resultado esperado) y negativos (que devolvían un error) se almacenan en ficheros de extensión “.safe” (ver figura 1). En esta entidad, los casos de hacking son cubiertos por el framework. Una vez realizadas todas las pruebas a las funcionalidades, utilizamos la herramienta replay proporcionada por el framework, que ejecutaba todos los ficheros de pruebas y, en caso de no devolver el resultado esperado devolvía un error. Una vez terminada la ejecución, nos devolvía la cobertura de las funcionalidades que se probaban en los ficheros.

Las pruebas han dado resultado a una cobertura del 99.6% (véase figura 2)

A continuación, en la sección de contenidos mostraremos la cobertura de cada fichero, centrándonos principalmente en las líneas subrayadas en amarillo, que indican que ha habido condiciones dentro de estructuras de bucles o de condición que no se han cumplido en todas sus posibles ramas durante la ejecución de las pruebas.

 airport-replay.trace	02/07/2025 19:25	TRACE File	530 KB
 create.safe	02/07/2025 19:25	SAFE File	230 KB
 create2.safe	02/07/2025 19:58	SAFE File	29 KB
 list.safe	02/07/2025 19:25	SAFE File	17 KB
 show.safe	02/07/2025 19:25	SAFE File	20 KB
 update.safe	02/07/2025 19:25	SAFE File	266 KB
 update2.safe	02/07/2025 19:55	SAFE File	29 KB

1. Archivos .safe

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
▼  acme.features.administrator.airport	 99,6 %	560	2	562
>  AdministratorAirportController.java	 100,0 %	24	0	24
>  AdministratorAirportListService.java	 100,0 %	46	0	46
>  AdministratorAirportShowService.java	 100,0 %	85	0	85
>  AdministratorAirportUpdateService.java	 99,5 %	207	1	208
>  AdministratorAirportCreateService.java	 99,5 %	198	1	199

2. Cobertura de la funcionalidad de aeropuertos

1.2 Cobertura de las funcionalidades de los aeropuertos (Airport)

1.2.1 AdministratorAirportListService

Para probar la funcionalidad inicio sesión como administrador y voy pasando por los índices. Después, apago el sistema mediante shut system down. La cobertura final es del 100%.

```
package acme.features.administrator.airport;

import java.util.Collection;

import org.springframework.beans.factory.annotation.Autowired;

import acme.client.components.models.Dataset;
import acme.client.components.principals.Administrator;
import acme.client.services.AbstractGuiService;
import acme.client.services.GuiService;
import acme.entities.airport.Airport;

@GuiService
public class AdministratorAirportListService extends AbstractGuiService<Administrator, Airport> {

    @Autowired
    private AdministratorAirportRepository repository;

    @Override
    public void authorise() {
        boolean status = super.getRequest().getPrincipal().hasRealmOfType(Administrator.class);
        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        Collection<Airport> airports;
        airports = this.repository.findAllAirports();
        super.getBuffer().addData(airports);
    }

    @Override
    public void unbind(final Airport airport) {
        Dataset dataset;
        dataset = super.unbindObject(airport, "iataCode", "name", "city");
        super.getResponse().addData(dataset);
    }
}
```

1.2.2 AdministratorAirportShowService

- show.safe: Inicio sesión como administrador, y visualizo varios aeropuertos con atributos diferentes. Después apago el sistema (shut system down).

La cobertura final es del 100%.

```
package acme.features.administrator.airport;

import org.springframework.beans.factory.annotation.Autowired;

import acme.client.components.models.Dataset;
import acme.client.components.principals.Administrator;
import acme.client.components.views.SelectChoices;
import acme.client.services.AbstractGuiService;
import acme.client.services.GuiService;
import acme.entities.airport.Airport;
import acme.entities.airport.OperationalType;

@GuiService
public class AdministratorAirportShowService extends AbstractGuiService<Administrator, Airport> {

    @Autowired
    private AdministratorAirportRepository repository;

    @Override
    public void authorise() {
        boolean status = super.getRequest().getPrincipal().hasRealmOfType(Administrator.class);
        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        Airport airport;
        int id;

        id = super.getRequest().getData("id", int.class);
        airport = this.repository.findAirportById(id);

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

    @Override
    public void unbind(final Airport airport) {
        Dataset dataset;
        SelectChoices choices;

        choices = SelectChoices.from(OperationalType.class, airport.getOperationalScope());
        dataset = super.unbindObject(airport, "name", "iataCode", "operationalScope", "country", "city", "website", "email", "phoneNumber");
        dataset.put("operationalScopes", choices);
        super.getResponse().addData(dataset);
    }
}
```

1.2.3 AdministratorAirportCreateService

- create.safe: Pruebo con cada uno de los atributos del airport. En el caso de los atributos de texto, también he probado con caracteres no latinos e injections. Además, he probado con todos los valores del desplegable del atributo enumerado.
- create2.safe: había faltado por probar el perform que guarda los datos en el repositorio.

En la línea 24, 30 y 35 han faltado por recorrerse una de las dos ramas, en la línea 42 han faltado por recorrerse dos de las cuatro ramas. La cobertura final es del 99.5%.

```
package acme.features.administrator.airport;

import org.springframework.beans.factory.annotation.Autowired;

import acme.client.components.models.Dataset;
import acme.client.components.principals.Administrator;
import acme.client.components.views.SelectChoices;
import acme.client.controllers.GuiController;
import acme.client.services.AbstractGuiService;
import acme.entities.airport.Airport;
import acme.entities.airport.OperationalType;

@GuiController
public class AdministratorAirportCreateService extends AbstractGuiService<Administrator, Airport> {

    @Autowired
    private AdministratorAirportRepository repository;

    @Override
    public void authorise() {
        boolean status = false;
        if (super.getRequest().getPrincipal().hasRealmOfType(Administrator.class)) {
            status = true;
            if (super.getRequest().getMethod().equals("POST")) {
                String operationalScopeInput = super.getRequest().getData("operationalScope", String.class);
                boolean operationalScopeValid = false;

                if (operationalScopeInput != null) {
                    String trimmedInput = operationalScopeInput.trim();
                    if (trimmedInput.equals("0"))
                        operationalScopeValid = true;
                    else
                        for (OperationalType ot : OperationalType.values())
                            if (ot.name().equalsIgnoreCase(trimmedInput)) {
                                operationalScopeValid = true;
                                break;
                            }
                }

                status = status && operationalScopeValid;
            }
        }
        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        Airport airport;

        airport = new Airport();

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

    @Override
    public void bind(final Airport airport) {
        super.bindObject(airport, "name", "iataCode", "operationalScope", "country", "city", "website", "email", "phoneNumber");
    }

    @Override
    public void validate(final Airport airport) {
        boolean confirmation;
        confirmation = super.getRequest().getData("confirmation", boolean.class);
        super.state(confirmation, "confirmation", "acme.validation.confirmation.message");
    }

    @Override
    public void perform(final Airport airport) {
        this.repository.save(airport);
    }

    @Override
    public void unbind(final Airport airport) {
        Dataset dataset;
        SelectChoices choices;

        choices = SelectChoices.from(OperationalType.class, airport.getOperationalScope());
        dataset = super.unbindObject(airport, "name", "iataCode", "operationalScope", "country", "city", "website", "email", "phoneNumber");
        dataset.put("operationalScopes", choices);
        super.getResponse().addData(dataset);
    }
}
```


1.2.4 AdministratorAirportUpdateService

- update.safe: Pruebo con cada uno de los atributos del airport. En el caso de los atributos de texto, también he probado con caracteres no latinos e injections. Además, he probado con todos los valores del desplegable del atributo enumerado.
- update2.safe: había faltado por probar el perform que actualiza los datos en el repositorio

En la línea 24, 26, 30 y 35 han faltado por recorrerse una de las dos ramas, en la línea 42 han faltado por recorrerse dos de las cuatro ramas. La cobertura final es del 99.5%.

```
|
package acme.features.administrator.airport;

import org.springframework.beans.factory.annotation.Autowired;

import acme.client.components.models.Dataset;
import acme.client.components.principals.Administrator;
import acme.client.components.views.SelectChoices;
import acme.client.services.AbstractGuiService;
import acme.client.services.GuiService;
import acme.entities.airport.Airport;
import acme.entities.airport.OperationalType;

@GuiService
public class AdministratorAirportUpdateService extends AbstractGuiService<Administrator, Airport> {

    @Autowired
    private AdministratorAirportRepository repository;

    @Override
    public void authorise() {
        boolean status = false;
        if (super.getRequest().getPrincipal().hasRealmOfType(Administrator.class)) {
            status = true;
            if (super.getRequest().getMethod().equals("POST")) {
                String operationalScopeInput = super.getRequest().getData("operationalScope", String.class);
                boolean operationalScopeValid = false;

                if (operationalScopeInput != null) {
                    String trimmedInput = operationalScopeInput.trim();
                    if (trimmedInput.equals("0"))
                        operationalScopeValid = true;
                    else
                        for (OperationalType ot : OperationalType.values())
                            if (ot.name().equalsIgnoreCase(trimmedInput)) {
                                operationalScopeValid = true;
                                break;
                            }
                }

                status = status && operationalScopeValid;
            }
            super.getResponse().setAuthorised(status);
        }

        @Override
        public void load() {
            Airport airport;
            int id;
            id = super.getRequest().getData("id", int.class);

            airport = this.repository.findAirportById(id);

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

        @Override
        public void bind(final Airport airport) {
            super.bindObject(airport, "name", "iataCode", "operationalScope", "country", "city", "website", "email", "phoneNumber");
        }

        @Override
        public void validate(final Airport airport) {
            boolean confirmation;
            confirmation = super.getRequest().getData("confirmation", boolean.class);
            super.state(confirmation, "confirmation", "acme.validation.confirmation.message");
        }

        @Override
        public void perform(final Airport airport) {
            this.repository.save(airport);
        }

        @Override
        public void unbind(final Airport airport) {
            Dataset dataset;
            SelectChoices choices;

            choices = SelectChoices.from(OperationalType.class, airport.getOperationalScope());
            dataset = super.unbindObject(airport, "name", "iataCode", "operationalScope", "country", "city", "website", "email", "phoneNumber");
            dataset.put("operationalScopes", choices);
            super.getResponse().addData(dataset);
        }
    }
}
```

2. Pruebas de rendimiento

Para realizar las pruebas de rendimiento, nos hemos valido de la traza final que devuelve la herramienta replay proporcionada por el framework. Al devolver el fichero “.trace” y convertirlo a CSV extraemos los datos de tiempo de cada una de las peticiones. Vamos a analizar los rendimientos antes y después de optimizar los índices.

Especificaciones de la computadora en la que se han realizado las primeras pruebas:

- Procesador
 - Ryzen 4800H
 - 8 núcleos (16 hilos)
 - 4.2 GHz
- Memoria
 - 32 GiB de RAM
 - DDR4 @ 3200 MHz
 - Dual Channel (2x16GiB)
- Almacenamiento
 - 512 GiB de capacidad
 - NVMe @ Lectura 1600MB/s, Escritura 800MB/s
 - SSD

Se ha realizado un contraste de hipótesis mediante Z-Test que demuestra que los cambios han provocado mejoras en el rendimiento.

z-Test: Two Sample for Means		
	<i>before</i>	<i>after</i>
Mean	15,46143388	13,98950103
Known Variance	107,8203	75,3713
Observations	183	183
Hypothesized Mean Difference	0	
z	1,471162899	
P(Z<=z) one-tail	0,070623534	
z Critical one-tail	1,644853627	
P(Z<=z) two-tail	0,141247068	
z Critical two-tail	1,959963985	

El dato relevante en el que nos tenemos que fijar es en valor P de dos colas, en el que estar bastante alejado de cero (el intervalo es [0, 1]) no se considera evidencia estadísticamente fuerte y no hemos mejorado el rendimiento.

En las siguientes tablas, se observa el rendimiento antes y después de aplicar los cambios:

<i>before</i>				<i>after</i>		
Mean	15,46143			Mean	13,9895	
Standard Error	0,767582			Standard Error	0,641767	
Median	16,4895			Median	15,1445	
Mode	#N/A			Mode	#N/A	
Standard Deviation	10,38366			Standard Deviation	8,681664	
Sample Variance	107,8203			Sample Variance	75,3713	
Kurtosis	34,7118			Kurtosis	33,3535	
Skewness	4,619022			Skewness	4,329943	
Range	94,9461			Range	82,6895	
Minimum	2,0069			Minimum	2,1696	
Maximum	96,953			Maximum	84,8591	
Sum	2829,442			Sum	2560,079	
Count	183			Count	183	
Confidence Level(95,0%)	1,514504			Confidence Level(95,0%)	1,26626	
Interval (ms)	13,94693	16,97594		Interval (ms)	12,72324	15,25576
Interval (s)	0,013947	0,016976		Interval (s)	0,012723	0,015256

Al fijarnos en los intervalos de abajo, vemos que la diferencia apenas se llega a percibir.

He instalado el espacio de trabajo en otra computadora de familia ligeramente peor en especificaciones:

- Procesador
 - Intel Core 5 8250U, 8va generación
 - 4 núcleos (8 hilos)
 - 3.4 GHz
- Memoria
 - 16 GiB de RAM
 - DDR4 @ 2400 MHz
 - Dual Channel (2x8GiB)
- Almacenamiento
 - 240 GiB de capacidad
 - SATA-III @ Lectura 500MB/s, Escritura 450MB/s
 - SSD

z-Test: Two Sample for Means		
	<i>before</i>	<i>after</i>
Mean	23,1085459	22,00713115
Known Variance	1221,668275	549,8743138
Observations	183	183
Hypothesized Mean Difference	0	
z	0,353997859	
P(Z<=z) one-tail	0,361670244	
z Critical one-tail	1,644853627	
P(Z<=z) two-tail	0,723340487	
z Critical two-tail	1,959963985	

El valor P de dos colas vuelve a estar muy alejado de cero, en este caso es un valor gigante lo que nos da entender que no ha habido ninguna mejora al implementar los índices.

En las siguientes tablas, se observa el rendimiento antes y después de aplicar los cambios:

<i>before</i>				<i>after</i>		
Mean	23,1085459			Mean	22,00713115	
Standard Error	2,58375365			Standard Error	1,733429454	
Median	19,6288			Median	18,6872	
Mode	#N/A			Mode	#N/A	
Standard Deviation	34,95237152			Standard Deviation	23,44939901	
Sample Variance	1221,668275			Sample Variance	549,8743138	
Kurtosis	74,1348893			Kurtosis	20,3861419	
Skewness	7,999047719			Skewness	4,151305047	
Range	382,8791			Range	170,663	
Minimum	1,7333			Minimum	1,669	
Maximum	384,6124			Maximum	172,332	
Sum	4228,8639			Sum	4027,305	
Count	183			Count	183	
Confidence Level(95,0%)	5,097963186			Confidence Level(95,0%)	3,420202054	
Interval (ms)	18,01058272	28,20650909		Interval (ms)	18,58692909	25,42733
Interval (s)	0,018010583	0,028206509		Interval (s)	0,018586929	0,025427

3. Conclusión

Este informe evidencia que el proceso de testing formal implementado en el proyecto ha sido riguroso y orientado a la detección y corrección de posibles errores. Las pruebas realizadas cubren una amplia variedad de escenarios, en especial aquellos relacionados con el requisito 11 grupal, lo cual permite minimizar significativamente la probabilidad de fallos en el sistema.

En cuanto al análisis de rendimiento, se ha llevado a cabo Z-Test con un nivel de confianza del 95%. Los resultados muestran que, pese a las optimizaciones aplicadas para acelerar las consultas a la base de datos, no se ha logrado una mejora apreciable en los tiempos de respuesta. Esto ha permitido detectar ciertos métodos que, a pesar de haber sido revisados e incluso refactorizados, no han producido el impacto esperado en el rendimiento.

En términos generales, se puede afirmar que el sistema presenta una funcionalidad sólida y estable gracias al alto grado de cobertura de pruebas con un 100% en la mayoría de estas, aunque aún existen aspectos relacionados con el rendimiento que podrían afectar a usuarios con dispositivos o entornos menos potentes.