

# Testing Report

- Número de grupo: C1.047
- Repositorio:  
<https://github.com/JoaquinBorjaLeon/C1-047-Acme-ANS-D04>
- Miembros y emails corporativos:  
  
Joaquín Borja León: [joaborleo@alum.us.es](mailto:joaborleo@alum.us.es)  
  
Ariel Escobar Capilla: [ariesccap@alum.us.es](mailto:ariesccap@alum.us.es)  
  
Héctor Guerra Prada: [hecguepra@alum.us.es](mailto:hecguepra@alum.us.es)  
  
Juan Carlos León Madroñal: [jualeomad@alum.us.es](mailto:jualeomad@alum.us.es)  
  
José Ángel Rodríguez Durán: [josroddur@alum.us.es](mailto:josroddur@alum.us.es)
- Fecha: 26/05/2025

## 1. Resumen Ejecutivo

En este informe se pueden apreciar las pruebas funcionales y de rendimiento que llevé a cabo sobre la tarea grupal. El objetivo fue asegurar que todas las funcionalidades se comportan según lo esperado y evaluar la rapidez con la que responde el sistema en condiciones normales.

Para las pruebas funcionales, organicé los casos de prueba por funcionalidad. Cada caso se centra en una función específica y fue esencial para verificar que la aplicación se comporta correctamente.

Para las pruebas de rendimiento, seguí la metodología indicada en la guía de la sesión: recopilé los tiempos de ejecución a partir de los archivos `.trace` y procesé los datos utilizando Excel. Generé gráficos y calculé intervalos de confianza del 95 % para evaluar si los tiempos de respuesta del sistema se mantenían dentro de los límites aceptables. Las pruebas se ejecutaron en dos portátiles y se adjuntan las más rápidas.

En resumen, este informe refleja las pruebas que he realizado sobre las funcionalidades clave de la aplicación, respaldadas por datos de rendimiento que ofrecen una comprensión sólida del comportamiento del sistema en condiciones reales.

## 2. Introducción

El propósito de este informe es documentar las pruebas que realicé sobre mi proyecto, abarcando tanto la funcionalidad como el rendimiento. Mi objetivo fue asegurar que cada

funcionalidad funcionara correctamente y que el sistema pudiera gestionar las peticiones de forma eficiente.

Para ello, en primer lugar, ejecuté un conjunto de pruebas funcionales agrupadas por característica, verificando que la aplicación se comportaba según lo esperado. A continuación, me centré en las pruebas de rendimiento, analizando cuánto tiempo tardaba el sistema en responder a distintas peticiones. Esto me permitió identificar posibles cuellos de botella y comparar el rendimiento entre diferentes configuraciones.

En conjunto, esta fase de pruebas me proporcionó una visión más clara del comportamiento del sistema en condiciones reales.

### 3. Content













#### Pruebas funcionales realizadas para las funcionalidades de Airport:

- **List.safe:** Verifica que los aeropuertos registrados se muestren correctamente. Asegura que se recuperen y presenten todos los registros disponibles en el sistema según el rol del usuario.
  - **Show.safe:** Comprueba que los detalles de un aeropuerto específico se visualicen correctamente, incluyendo todos los campos opcionales y obligatorios.
  - **Create.safe:** Evalúa la creación de nuevos aeropuertos con datos válidos. Se realizan pruebas de validación con los siguientes escenarios:
    - Campos obligatorios vacíos.
    - Códigos IATA duplicados o inválidos.
    - Cadenas de texto fuera del rango (menos de 1 o más de 50 caracteres en name, city, country).
    - Valores opcionales mal formateados, como URLs no válidas, correos electrónicos inválidos o números de teléfono mal estructurados.
  - **Update.safe:** Verifica que los datos de un aeropuerto puedan actualizarse correctamente, respetando las mismas restricciones y validaciones al igual que en create.safe. Se prueban tanto entradas válidas como inválidas.
-

#### 4. Pruebas de seguridad realizadas (hack) para las funcionalidades de Airport:

- **Show.hack:** Simula el acceso a detalles de aeropuertos que no existen o que no deberían ser accesibles según el rol del usuario, manipulando el ID en la URL.
- **Create.hack:** Intenta crear aeropuertos usando herramientas del navegador para introducir códigos IATA ya registrados, valores fuera del formato permitido en `operationalScope`
- **Update.hack:** Evalúa intentos de modificar aeropuertos que no pertenecen al usuario autorizado, o alterar campos como el `operationalScope` usando inyecciones en el cuerpo de la solicitud.
- **List.hack:** Se manipulan filtros o parámetros de consulta para intentar acceder a información de aeropuertos restringidos o probar exposiciones indebidas de datos sensibles a través del listado.

La cobertura de test alcanzada de la entidad `airport` ha sido del 99.3%

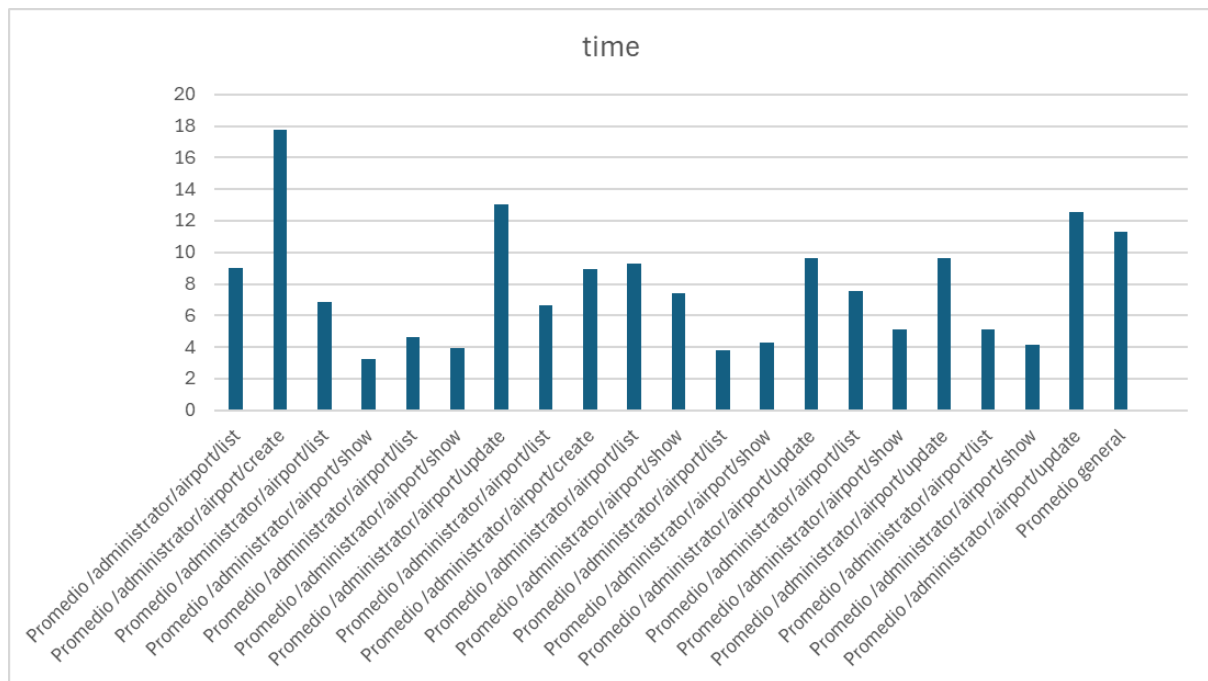
✓  <code>acme.features.administrator.airport</code>		99,3 %	561	4	565
>  <code>AdministratorAirportCreateService.java</code>		97,8 %	181	4	185
>  <code>AdministratorAirportController.java</code>		100,0 %	24	0	24
>  <code>AdministratorAirportListService.java</code>		100,0 %	48	0	48
>  <code>AdministratorAirportShowService.java</code>		100,0 %	106	0	106
>  <code>AdministratorAirportUpdateService.java</code>		100,0 %	202	0	202

#### 5. Performance Testing

Se han realizado pruebas de rendimiento sobre el módulo de *airport* con el objetivo de evaluar los tiempos de respuesta y garantizar un acceso eficiente a los datos. Durante este análisis, se comprobó que las operaciones realizadas sobre esta entidad son mínimas y se limitan al uso del método `findById` en el repositorio.

Dado que no se realizan consultas complejas ni filtrados sobre campos específicos, no ha sido necesario incorporar índices adicionales en la base de datos. El acceso por ID ya está optimizado por defecto en la mayoría de los sistemas de gestión de bases de datos, por lo que el rendimiento observado ha sido satisfactorio sin requerir ajustes adicionales.

Como resultado, el sistema presenta tiempos de respuesta adecuados en las operaciones relacionadas con *airport*, manteniendo así una buena experiencia de usuario y eficiencia en el acceso a esta entidad.



Columna1				
Media	10,2766263			
Error típico	0,68672604	Interval(ms)	8,90699437	11,6462582
Mediana	9,6464	Interval(s)	0,00890699	0,01164626
Moda	8,9945			
Desviación e	5,78645646			
Varianza de	33,4830784			
Curtosis	0,64006885			
Coeficiente d	0,98436542			
Rango	23,7608			
Mínimo	3,2727			
Máximo	27,0335			
Suma	729,640465			
Cuenta	71			
Nivel de confi	1,3696319			

## 6. Software Profiling – Airport

La entidad **Airport** presenta un comportamiento eficiente en operaciones de lectura (**list** y **show**), ya que se trata de consultas simples sin relaciones complejas. Las operaciones de escritura (**create** y **update**) requieren más procesamiento debido a múltiples validaciones, como formatos de texto, correos electrónicos, URLs, teléfonos y la unicidad del código IATA. Esto puede aumentar el uso de CPU y memoria cuando se procesan entradas inválidas o muy extensas. Las validaciones se realizan en tiempo de ejecución usando anotaciones como `@ValidIataCode`, lo que puede impactar ligeramente el rendimiento si se reciben muchos datos simultáneamente.

Name	Self Time (CPU)	Total Time (CPU)
acme.features.administrator.airport.AdministratorAirportUpda	0,0 ms (- %)	312 ms (43,6 %)
acme.features.administrator.airport.AdministratorAirportCreateService.validate () s	0,0 ms (- %)	298 ms (41,6 %)
acme.features.administrator.airport.AdministratorAirportCreat	0,0 ms (- %)	105 ms (14,8 %)

## 7. Hardware Profiling – Airport

A nivel de hardware, la entidad **Airport** tiene un impacto bajo. Las operaciones de lectura son ligeras en CPU, memoria y E/S de disco. Las operaciones de escritura (crear/actualizar) consumen un poco más de recursos debido a la validación y escritura en la base de datos, especialmente si hay concurrencia o validaciones fallidas frecuentes. No se requiere hardware especializado; un entorno estándar con CPU moderna y SSD ofrece un rendimiento óptimo.

