INFORME DE TESTING GRUPAL

Grupo 1-C1.020 | https://github.com/PDJ6975/Acme-ANS-D04-25.5.0

Nombre	Correo Corporativo
Antonio Rodríguez Calderón	antrodcal@alum.us.es
Adrián Ramírez Gil	adrramgil@alum.us.es
Jianwu Hu	jiahu3@alum.us.es
Pablo Castrillón Mora	pabcasmor1@alum.us.es
Pablo Olivencia Moreno	pabolimor@alum.us.es

25 DE MAYO DE 2025

ANTONIO RODRÍGUEZ | STUDENT #3

Tabla de Contenido

1. Testing Funcional

1.1 Introducción

En esta sección del documento se pretende describir un listado con los casos de prueba realizados, agrupados por características, de todas las funcionalidades de la entidad "Airport" accesibles desde los administradores.

Para cada caso de prueba, se va a proporcionar una descripción y cuál es su eficacia para detectar errores.

1.2 Metodología

Para probar los servicios mencionados, se han aplicado pruebas **".safe"** y **".hack"**. Para las primeras se ha probado que las entidades se listen, muestren y permitan un CRUD válido y funcional, permitiendo todo tipo de valores válidos y rechazando aquellos que no cumplen con la validación.

Por otro lado, para las segundas, se ha buscado comprobar la seguridad de las "url" para el acceso a las funcionalidades y la integridad de los enumerados y otros campos ante valores ilegales.

En lo que respecta a la cobertura global alcanzada en el paquete **"administrator.airport"**, se ha conseguido un 98% de cobertura, de forma que se cubren la gran mayoría de líneas de código del paquete.

Por otro lado, en lo que respecta a cada servicio específico, nos encontramos con:

Servicio	Cobertura alcanzada
AdministratorAirportCreateService	97,5%
AdministratorAirportUpdateService	97,5%
AdministratorAirportShowService	99%
AdministratorAirportListService	100%

Para el controlador "AdministratorAirportController" se alcanza un 100% de la cobertura, lo que significa que se prueban todas las líneas de código, es decir, que se realiza el acceso a todos los servicios del mismo.

1.3 Casos de prueba por característica

Comencemos especificando los casos de pruebas para la característica **“administrator/airport/list”**:

Caso de prueba	Tipo	Descripción	Eficacia
list.safe	Renderizado	Se prueba que el listado de aeropuertos se renderice correctamente	No se detectó ningún bug oculto
list.hack	Hack	Se prueba que: -Miembros sin rol privilegiado no puedan ver el listado de aeropuertos (deben acceder con su rol por url)	El servicio valida correctamente las acciones ilegales

Pasemos a **“administrator/airport/show?id=xxx”:**

Caso de prueba	Tipo	Descripción	Eficacia
show.safe	Renderizado	Se prueba que los detalles de un aeropuerto se rendericen correctamente sin error	No ha detectado ningún bug oculto
show.hack	Hack	Permite probar: -Miembros sin rol privilegiado no puedan ver los detalles del aeropuerto (deben acceder desde su rol directo) -Se maneje correctamente la llegada de aeropuertos nulos	El servicio valida correctamente las acciones ilegales

En lo que respecta a **“administrator/airport/create” (GET y POST):**

Caso de prueba	Tipo	Descripción	Eficacia
create.safe	Casos positivos y negativos del formulario	Se prueba que los campos del formulario estén correctamente validados y que la confirmación funcione correctamente	Se detectó la necesidad de activar “remote=true” para validar las “url”
create.hack	Hack	Permite probar: -Miembros sin rol privilegiado no puedan acceder al formulario de creación - Atributos de selección no puedan ser modificados con valores ilegales.	El servicio valida correctamente las acciones ilegales

En lo que respecta a **“administrator/airport/update?id=xxx” (GET y POST):**

Caso de prueba	Tipo	Descripción	Eficacia
update.safe	Casos positivos y negativos del formulario	Se prueba que todos los campos del formulario estén correctamente validados, rechazando los valores no admitidos por el modelo.	No se detectó ningún bug oculto
update.hack	Hack	Permite probar: -Miembros sin rol privilegiado no puedan acceder a los detalles de un aeropuerto por medio del “update” (deben acceder por su rol y solo por show). - Atributos de selección no puedan ser modificados con valores ilegales.	El servicio valida correctamente las acciones ilegales

1.4 Conclusiones finales

En definitiva, se han desarrollado un conjunto de pruebas que cubre la gran mayoría de líneas de código de la entidad grupal.

2. Testing de Rendimiento

2.1 Entorno y protocolo

Para el desarrollo de esta sección se van a emplear dos equipos con las siguientes características:

Modelo	Tipo	RAM	CPU	GPU	Disco	SO
Rog Strix G513RM	Rápido	16.0 GB	AMD Ryzen 7 6800H 3.20 GHz	Nvidia RTX 3060 6 GB	954 GB	Windows 11 Pro
HP Pavilion 15-cs0008ns	Lento	16.0 GB	Intel core i7-8550u 1.80GHz	Nvidia MX150 2GB	1.82 TB	Windows 10 Home

Para este estudio se va a emplear únicamente los casos de prueba realizados para la entidad grupal del requisito once, empleando la versión 25.5.0 del proyecto y del framework. El lanzamiento se realizará directamente con los índices ya establecidos, debido a que solo era necesario un índice en la entidad "Service" grupal, que ni siquiera va a influir directamente en el desarrollo de los casos de prueba.

Así, se va a realizar una comparativa entre uno de los equipos más rápidos del equipo y el más lento, con el objetivo de asegurar la confianza en el rendimiento del sistema bajo cualquier circunstancia.

2.2 Recogida de datos

Para la obtención de datos, se han seguido los pasos explicados en la teoría para ambos equipos, analizando la información del ".trace" generado por la aplicación y filtrándola para obtener un fichero limpio del que poder obtener información útil como gráficos.

2.3 Estadísticas Descriptivas

Una vez filtradas las peticiones realizadas en los casos de prueba eliminando las irrelevantes, podemos afirmar que, para el estudio, se utiliza un total de 224 filas de datos. La media, desviación y el resto de datos difieren bastante entre ambos equipos, pero se mantienen con mucha holgura con respecto al requisito impuesto:

Estadística Descriptiva	
Media	18,35873616
Error típico	1,136184581
Mediana	18,5862
Moda	-
Desviación estándar	17,00485373
Varianza de la muestra	289,1650503
Curtosis	1,23786038
Coeficiente de asimetría	1,129188296
Rango	81,1736
Mínimo	1,2081
Máximo	82,3817
Suma	4112,3569
Cuenta	224
Nivel de confianza(95,0%)	2,239032326

interval(ms)	16,1197038	20,5977685
interval(s)	0,0161197	0,02059777

Ilustración 1: Datos estadísticos obtenidos para el equipo rápido

Estadística Descriptiva	
Media	50,82398928
Error típico	3,154191643
Mediana	56,565
Moda	-
Desviación estándar	47,20761784
Varianza de la muestra	2228,559182
Curtosis	1,075048108
Coeficiente de asimetría	0,968706021
Rango	254,5108
Mínimo	2,6696
Máximo	257,1804
Suma	11384,5736
Cuenta	224
Nivel de confianza(95,0%)	6,215836025

interval(ms)	44,6081533	57,0398253
interval(s)	0,04460815	0,05703983

Ilustración 2: Datos estadísticos obtenidos para el equipo más lento.

Como podemos observar, para el equipo más rápido se obtiene una media cercana a ≈18 ms, con desviación ≈17 ms. Sin embargo, para el equipo más lento obtenemos una media ≈50 ms, con desviación ≈47 ms. Así, se puede observar una brecha bastante marcada entre ambos equipos, pero siguen siendo datos muy pequeños en las unidades de tiempo.

Esto se puede observar también en los intervalos de confianza, donde el límite superior del primer equipo ronda los ≈0,02 s, a diferencia del segundo, que ronda los ≈0,05 s. Así, si bien el margen del PC rápido es ~3 veces menor, ambos están

muy por debajo del límite de 1 segundo impuesto. Así, el objetivo de rendimiento se cumple con un margen muy amplio.

2.4 Gráficos de eficiencia por característica

Uno de los aspectos que más nos interesa del rendimiento, es comprobar cuál es la petición más ineficiente (MIR):

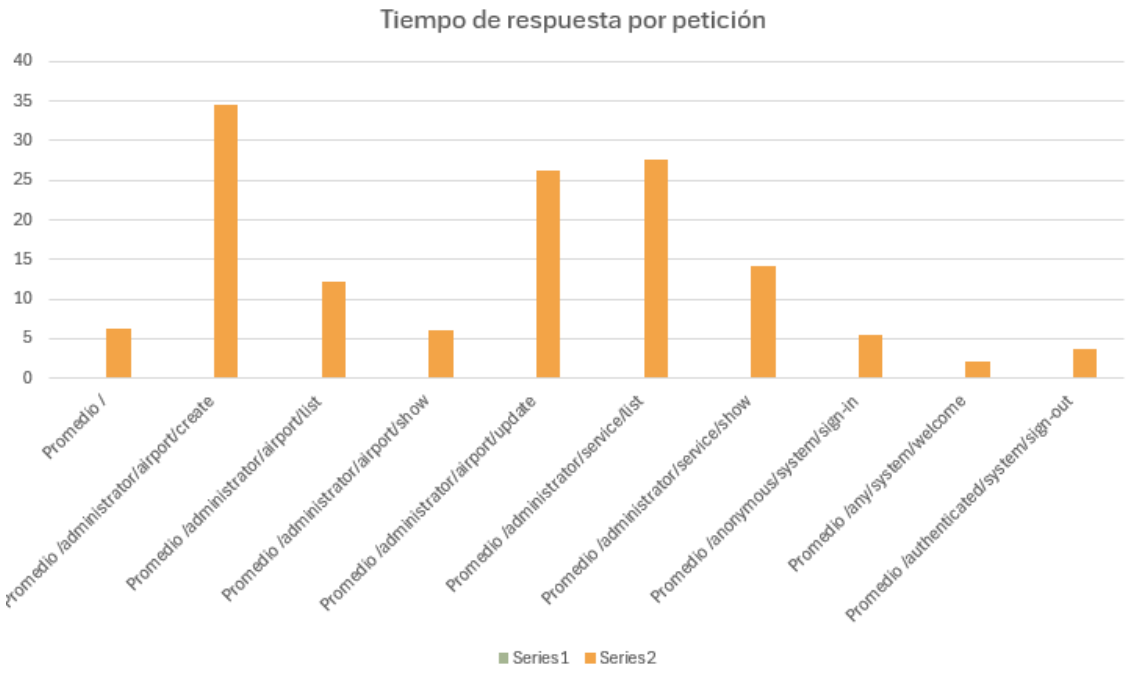

Ilustración 3: Gráfico de eficiencia para el equipo rápido

Ilustración 4: Gráfico de eficiencia para el equipo lento

Se observa claramente que la petición más ineficiente es "administrator/airport/create", puesto que es la que más datos debe recuperar para poder llevarse a cabo.

Es importante observar cómo en el equipo más lento la petición "administrator/créate/update" conlleva mucho más tiempo que en el equipo rápido de manera proporcional al resto de peticiones.

Así, efectivamente, comparando los gráficos con el estudio estadístico obtenido anteriormente, se puede corroborar que el equipo rápido es generalmente tres veces más rápido para procesar cada petición.

En definitiva, si bien no era necesario analizar el MIR porque los límites superiores de los intervalos de confianza de ambos equipos estaban muy por debajo del requisito impuesto, nos ha ayudado a perfilar un poco más las diferencias de rendimiento.

2.5 Hipótesis y conclusión

Para finalizar con las pruebas de rendimiento, vamos a concluir este informe con una decisión firme sobre los resultados de las dos trazas generadas. Tras comparar el tiempo de las peticiones realizadas en ambas, hemos obtenido los siguientes datos:

	Fast	*Slow*
Media	18,3587362	50,8239893
Varianza (conocida)	2891650	222855918
Observaciones	224	224
Diferencia hipotética de las medias	0	
z	-0,03233935	
P(Z<=z) una cola	0,48710072	
Valor crítico de z (una cola)	1,64485363	
P(Z<=z) dos cola	0,97420143	
Valor crítico de z (dos colas)	1,95996398	

Ilustración 5: Prueba z para muestras de dos medias

El tiempo que realmente nos interesa de estos datos es el **valor "p" de dos colas (≈ 0,97420143)**. Sabiendo que **"α = 1 – nivel de confianza"**, nos queda que este valor "**p > α > 1 – 0.95 -> p > 0.05"**. Como este valor se sitúa a la derecha del intervalo [0, α], podemos afirmar que el contraste no evidencia diferencias estadísticas significativas.

En ambos casos el tiempo medio queda muy por debajo del segundo impuesto, de modo que el sistema satisface el requisito de rendimiento incluso en el equipo más modesto. Por tanto, operativamente no sería necesario afinar más el estudio.

3. Testing de Mutaciones

4. Tabla de Revisión

Versión	Fecha	Descripción de los cambios
1.0	25/05/2025	Creación inicial del documento