ACME AirNav Solutions

TESTING REPORT - Group

Grupo: C1.050
Miembros: Cristina Fernández Chica (criferchi@alum.us.es), Ángel Amo Sánchez (angamosan@alum.us.es), Candela Jazmín Gutiérrez González (cangutgon@alum.us.es), Marta Aguilar Morcillo (maragumor@alum.us.es) y Luis Emmanuel Chávez Malavé (luichamal@alum.us.es)
Repositorio: https://github.com/Cristinafernandezchica/Acme-ANS
Planning dashboard: https://github.com/users/Cristinafernandezchica/projects/1/views/1

Sevilla 25 mayo, 2025

TABLA DE CONTENIDOS

Resumen Ejecutivo

En este documento se presentan los diferentes procedimientos llevados a cabo a la hora de realizar el testing formal del proyecto llevado a cabo. Esto pasa por la generación de múltiples casos de prueba que traten las distintas posibilidades, siguiendo por el análisis de la funcionalidad del código en base a las mismas, y el análisis del rendimiento entre dos ordenadores (en mi caso no ha existido la posibilidad de realizarlo con dos ordenadores distintos, por lo que se ha añadido o restado aleatoriamente un 10% al tiempo medio de solicitud obtenido en mi ordenador) y en distintas condiciones, que para mi caso será la adición de índices en la base de datos.

Tabla de Revisiones

Número de revisión	Fecha	Descripción de revisión	Autor
1.0	25/05/2025	Primera versión del documento. Añadido el apartado completo de “Pruebas funcionales”	Cristina Fernández Chica
2.0	26/05/2025	Finalización del documento.	Cristina Fernández Chica

Introducción

En el informe se explicará cómo se ha realizado el testing funcional y el análisis del desempeño, indicando los posibles errores encontrados en el proceso. Eso se aplicará al requisito funcional 11 de los requisitos grupales.

Para la redacción del mismo se seguirán los apartados indicados en los anexos proporcionados en la asignatura. Contará con dos apartados principales, las pruebas funcionales, donde indicaremos las pruebas realizadas por funcionalidad junto con la efectividad de las mismas, es decir, en qué medida nos han ayudado a detectar errores en el código, y las pruebas de rendimiento, con un intervalo de confianza del 95% al realizar pruebas antes y después de optimizar la búsqueda en base de datos con índices. Además, en este segundo apartado, se incluirán gráficos de tiempo empleado por funcionalidad, y un contraste de hipótesis con el nivel de confianza del 95%, esto se realizará mediante un Z-Test pudiendo comprobar si se han producido mejoras significativas al introducir los índices. Por último, se incluirá una conclusión en base a lo analizado en los apartados anteriores.

Pruebas

Pruebas funcionales

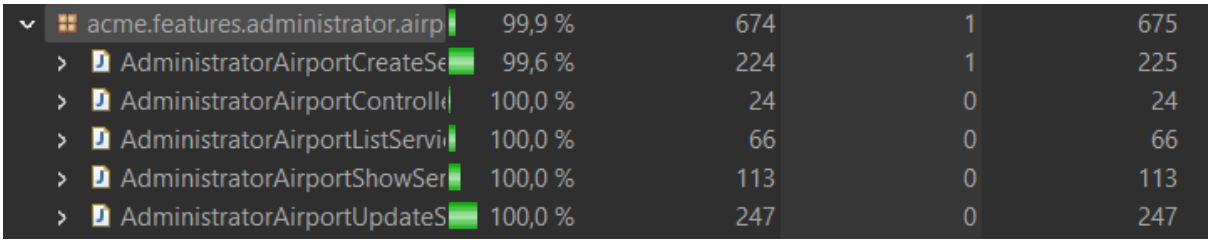
Para cada funcionalidad se han realizado pruebas positivas y negativas (archivos .safe) e intentos de hacking (archivos .hack). Se han seguido los pasos explicados en clase para la realización de las pruebas.

administrator/airline		
__.safe	**Descripción**	**Bugs**
list	Se listan los aeropuertos en los usuarios administrator y administrator1.	No se detectaron.
show	Se mostraron varios aeropuertos en administrator y administrator1.	No se detectaron.
create	Se probaron todas las variaciones de datos válidos e inválidos en cada campo verificando las validaciones hasta la creación de un aeropuerto con todos los campos correctos.	No se detectaron.
update	Se probaron todas las variaciones de datos válidos e inválidos en cada campo verificando las validaciones hasta la actualización de algún vuelo con datos correctos.	No se detectaron.
__.hack	**Descripción**	**Bugs**
list	Se intentaron listar los aeropuertos desde un usuario no autenticado y desde un usuario con rol distinto a administrator.	No se detectaron.
show	En administrator: → Aeropuerto con id=500 (administrator/airport/show?id=500) → Aeropuerto con id= (administrator/airport/show?id=) → Aeropuerto sin id (administrator/airport/show) En usuario no autenticado y rol distinto a administrator: → Todas las anteriores. → Aeropuerto que no deben poder ver (administrator/airport/show?id=156)	Se detectó que al introducir un id nulo no saltaba excepción.
create	En administrator: → Hackeo de campo enumerado. → Intento de actualización con create (cambiando la url del botón por administrator/airport/create?id=156). En usuario sin autenticar o con rol distinto: → Meter la dirección de creación de aeropuertos.	En un principio el segundo hackeo mencionado podía realizarse.

	(administrator/airport/create)	
update	En administrator: → Aeropuerto con id=500 (inexistente) → Aeropuerto con id= (vacío) → Entrar a la url de actualización justo al entrar sin hacerlo desde un show previo. → Hackeo de campo enumerado. Desde otro rol y usuario no autenticado: → Todo lo anterior → Intento de actualizar un aeropuerto.	En un inicio se podían meter IDs de aeropuertos no existentes.

Recubrimiento del código

Las pruebas se pueden considerar lo suficientemente buenas pues la cobertura de las funcionalidades de airport es casi del 100%. Por tanto, podemos decir que el código se ha probado en profundidad intentando buscar todos los casos posibles en cada una de las funcionalidades.



Análisis de desempeño

A continuación, vamos a analizar las estadísticas básicas sobre el rendimiento de la aplicación antes y después de optimizar los índices, utilizando también sus respectivas gráficas de tiempo por funcionalidad. Para ello, se ha realizado un contraste de hipótesis mediante Z-test que demuestra que los cambios no tienen impacto estadístico.

	Before	*After*
Media	7,348852105	7,176584737
Varianza (conocida)	55,23946888	55,20172287
Observaciones	190	190
Diferencia hipotética de las medias	0	
z	0,225951005	
P(Z<=z) una cola	0,410619764	
Valor crítico de z (una cola)	1,644853627	
Valor crítico de z (dos colas)	0,821239527	
Valor crítico de z (dos colas)	1,959963985	

Podemos observar que el p-valor obtenido al realizar la prueba Z (dos colas) es de 0.821239527.Habiendo considerado un nivel de confianza del 95%, el nivel de significancia es 0.05. Observamos que el p-valor no es menor que la significancia, lo que quiere decir que tenemos una hipótesis nula, pues no existe una diferencia significativa en los tiempos de ejecución antes y después de la adición de los índices. En las siguientes tablas, se observa los tiempos antes y después de aplicar los cambios:

Before	
Media	7,348852105
Error típico	0,539197595
Mediana	5,27315
Moda	1,6695
Desviación estándar	7,43232594
Varianza de la muestra	55,23946888
Curtosis	13,37157644
Coeficiente de asimetría	2,926409708
Rango	55,9492
Mínimo	0,9897
Máximo	56,9389
Suma	1396,2819
Cuenta	190
Nivel de confianza(95,0%)	1,063618523

After	
Media	7,176584737
Error típico	0,539013343
Mediana	5,0932
Moda	#N/D
Desviación estándar	7,429786193
Varianza de la muestra	55,20172287
Curtosis	16,17509243
Coeficiente de asimetría	3,19359111
Rango	58,2776
Mínimo	1,082
Máximo	59,3596
Suma	1363,5511
Cuenta	190
Nivel de confianza(95,0%)	1,063255067

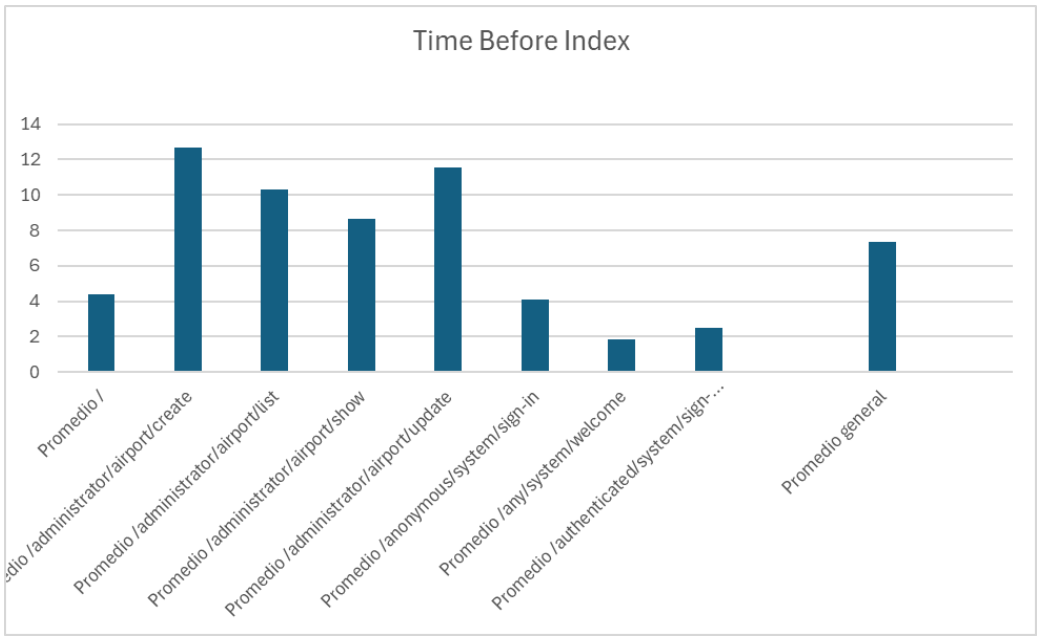
Interval (ms)	6,285233582	8,41247063
Interval (s)	0,006285234	0,00841247

Interval (ms)	6,11332967	8,2398398
Interval (s)	0,00611333	0,00823984

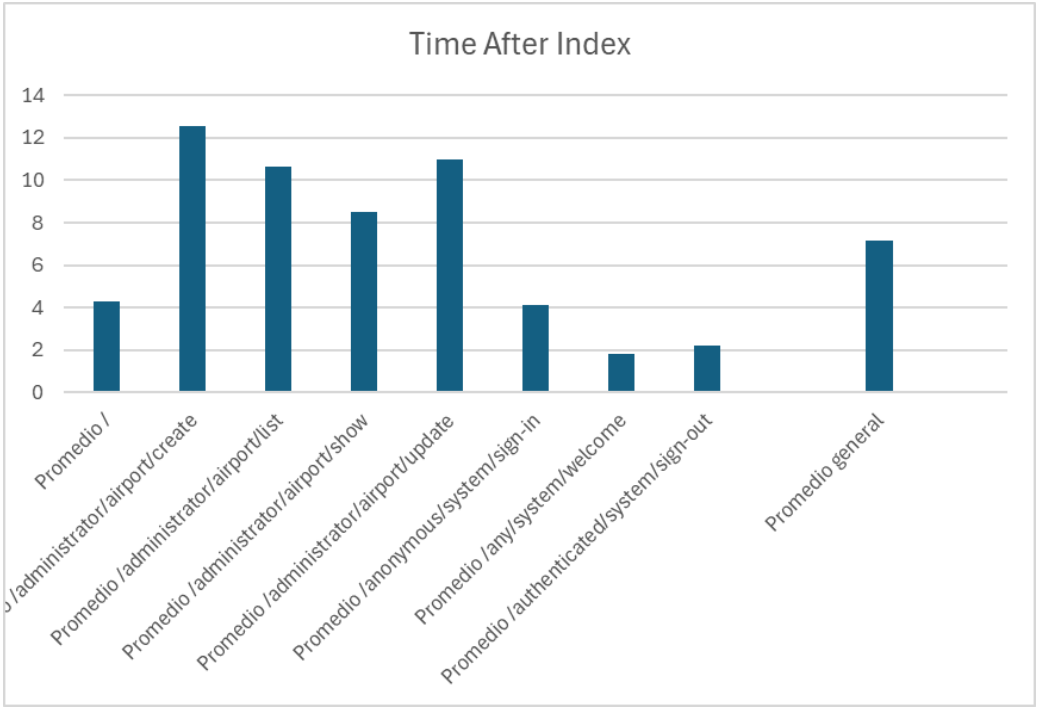
Fijándonos en el intervalo de confianza, antes de añadir los índices, podíamos responder en un tiempo inferior o igual a 8.41247063 ms. Mientras que, tras la adición de los índices optimizando la búsqueda en base de datos, la cota superior pasa a ser 8.2398398 ms, reduciéndose en 0.1726083. Este resultado no tiene significancia, como lo podemos corroborar con el p-valor (0.821239527), que es bastante mayor al nivel de significancia habitual (0.05).

Tras esto, vamos a observar como quedan los gráficos de cada uno, para ver cuales son los MIR que no se han conseguido reducir a pesar de la adición de índices:

Antes



Después



Se puede comprobar, que los MIR más claros provienen de /administrator/airport/create y administrator/airport/update y administrator/airport/list, y aunque update se ha reducido un poco, en general se han quedado al mismo nivel. Esto se debe a que los servicios no hacen demasiadas consultas que requieran índices más allá de los id de las entidades.

Los MIR no eran demasiado altos, pero al observar que trás la adición de los índices no se consiguió una mejora notable, se procedió a analizar la ejecución de pruebas con VisualVM. Los resultados no fueron concluyentes, pues los peores tiempos de ejecución se generaban en al método bind y validate de la actualización. Esto se revisó pero los bind constan, en su mayoría, de una sola línea para la carga de atributos, y en el validate se utiliza una consulta un poco complicada que no hemos podido cambiar

Name	Self Time (CPU)	Total Time (CPU)
acme.features.administrator.airport.AdministratorAirportUpdateService.**validate** ()	0,0 ms (- %)	101 ms (67,6 %)
acme.features.administrator.airport.AdministratorAirportUpdateService.**bind** ()	0,0 ms (- %)	48,6 ms (32,4 %)

Además, todos los tiempos Self Time (CPU) salían a 0.0 ms, lo que indicaba que no era el método en sí, si no lo que se llamaba dentro. Como se ha comentado se intentó refactorizar para comprobar si se conseguía una mejora, pero tras volver a analizar los datos tras la refactorización, no había una diferencia que destacase.

Para terminar con este análisis, vamos a ver una comparativa entre dos ordenadores distintos, comprobando de esta manera la relevancia del hardware a la hora de responder peticiones. La comparativa es únicamente tras la optimización de los índices.
El PC_1 es el mismo utilizado para el reporte, un HP Victus 16-d1001ns con procesador Intel Core i7 12700H y tarjeta gráfica dedicada (GeForce RTX 3050 Ti), y para el PC_2 se ha utilizado el portátil de uno de mis compañeros del grupo, con un Intel Core i7, sin tarjeta gráfica dedicada.

PC_1	
Media	7,176584737
Error típico	0,539013343
Mediana	5,0932
Moda	#N/D
Desviación estándar	7,429786193
Varianza de la muestra	55,20172287
Curtosis	16,17509243
Coeficiente de asimetría	3,19359111
Rango	58,2776
Mínimo	1,082
Máximo	59,3596
Suma	1363,5511
Cuenta	190
Nivel de confianza(95,0%)	1,063255067

Interval (ms)	6,11332967	8,239839804
Interval (s)	0,00611333	0,00823984

PC_2	
Media	18,64038947
Error típico	1,55594994
Mediana	9,50975
Moda	#N/D
Desviación estándar	21,44728983
Varianza de la muestra	459,986241
Curtosis	10,13258889
Coeficiente de asimetría	2,698554166
Rango	134,2064
Mínimo	2,0589
Máximo	136,2653
Suma	3541,674
Cuenta	190
Nivel de confianza(95,0%)	3,069259195

Interval (ms)	15,57113028	21,70964867
Interval (s)	0,01557113	0,021709649

Se puede observar que, para el PC_2, el intervalo de confianza tiene una cota superior de 21.70964867 ms, lo que es 2.635 veces mayor que la cota superior del PC_1. Esto deja en evidencia que el PC_1 tiene un desempeño mejor, ya que garantiza tiempos de respuestas considerablemente menores a los del PC_2.

Conclusión

En este informe se demuestra que el proceso de testing formal llevado a cabo en el proyecto ha sido completo y con el objetivo de encontrar todos los errores posibles para así corregirlos. Las pruebas realizadas indican que se han probado múltiples escenarios posibles para el requisito 11 grupal, pudiendo de esta manera reducir la posibilidad de fallo del sistema al mínimo.

Por otra parte, el análisis de rendimiento ha descubierto, mediante un Z-test con significancia estadística del 95%, que las optimizaciones aplicadas para una mayor rapidez en la búsqueda en base de datos, no han sido suficientes para mejorar el tiempo de respuesta del sistema, lo que nos ha llevado a identificar ciertos problemas con algunos métodos, que a pesar de ser revisados e incluso refactorizados, no se ha tenido éxito en la mejora del rendimiento.

Si lo miramos en conjunto, podemos decir que es un sistema con una calidad funcional bastante robusta, en base a los test realizados, pero con un rendimiento que puede llegar a dar ciertos problemas en dispositivos con sistemas menos avanzados.

De esto hemos aprendido que debemos revisar la eficiencia desde un inicio a la hora de implementar un proyecto, pues al querer refactorizar puede dar bastantes problemas y resultar en un sistema demasiado lento.

Bibliografía

Material proporcionado por el profesorado a través de la Enseñanza Virtual.