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

  Ariel Escobar Capilla: ariesccap@alum.us.es

  Héctor Guerra Prada: hecguepra@alum.us.es

  Juan Carlos León Madroñal: jualeomad@alum.us.es

  José Ángel Rodríguez Durán: josroddur@alum.us.es

- **Fecha**: 26/05/2025

Índice:

1. Resumen ejecutivo

En este informe presento las pruebas funcionales y de rendimiento que llevé a cabo sobre mi estudiante. El objetivo fue asegurar que todas las funcionalidades se comportaran 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 configuraciones distintas: una utilizando la base de datos sin ningún índice adicional, y otra con los índices relevantes aplicados. Posteriormente, realicé una comparación estadística entre ambas configuraciones para determinar el impacto del uso de índices en el rendimiento.

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. Historial de versiones

Número de versión	Fecha	Descripción
1.0	[26/05/2025]	Versión Inicial

3. 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.

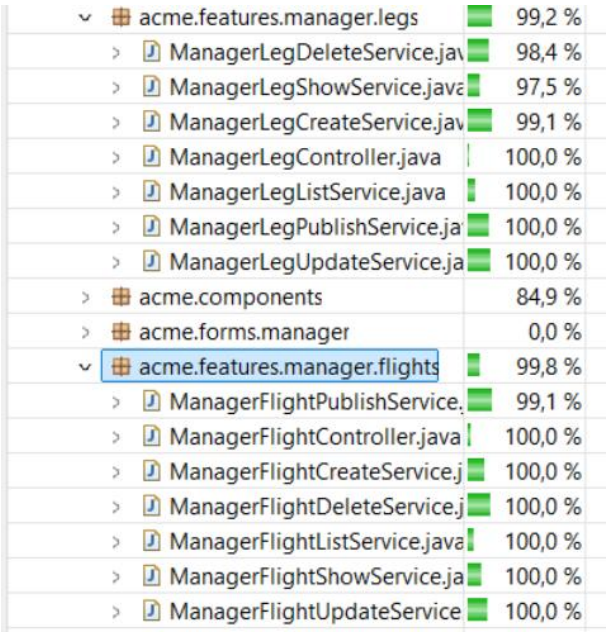
En conjunto, esta fase de pruebas me proporcionó una visión más clara del comportamiento del sistema en condiciones reales.

4. Contenido

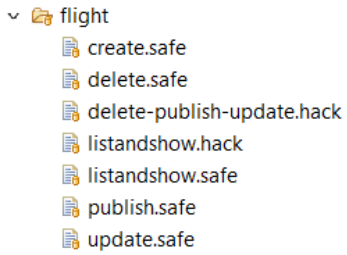
- **<u>Pruebas Funcionales:</u>**

Para estas pruebas, evalué las funcionalidades que mi proyecto, Acme-ANS, ofrece a los managers, concretamente en lo relativo a sus *flights* y *legs*. Las pruebas están documentadas mediante archivos de seguimiento ubicados en los directorios /src/test/resources/manager/flight y /src/test/resources/manager/legs. Los casos de prueba alcanzaron una cobertura del 99,2% para legs y un 99,8% para flights, como se muestra en la imagen a continuación.

Estos son los archivos de seguimiento (trace files) que demuestran la cobertura del código, incluidos justo debajo y creados siguiendo las recomendaciones proporcionadas en el curso. Realicé pruebas tanto con trazas.safe que cubren casos positivos y negativos como con trazas .hack, que simulan intentos de vulnerar la aplicación. Todos los intentos de hacking fueron gestionados correctamente y dieron como resultado una respuesta de "Access Unauthorised".



En las siguientes secciones se especifican los casos de prueba concretos y cómo se abordó cada uno de ellos.



Pruebas para flight:

- Create.safe Prueba la creación de vuelos utilizando datos de entrada válidos. Se evalúan los mecanismos de validación del sistema intentando crear un vuelo con campos vacíos, esperando que se generen los mensajes de error correspondientes. Además, se realizan pruebas con datos parcialmente válidos combinados con entradas no válidas, como cadenas excesivamente largas o valores de precio incorrectos.
- Delete.safe: Confirma que manager0 puede eliminar sus vuelos y que las entidades asociadas también se eliminan. Se prueba que el sistema impide la eliminación de vuelos que contiene legs que ya han sido publicadas.
- Delete-publish-update.hack: Agrupa una serie de pruebas destinadas a simular ataques mediante el uso indebido de herramientas del navegador y la manipulación de solicitudes. Se alteran manualmente IDs para simular actualizaciones no autorizadas de vuelos ya publicados o que no pertenecen al usuario. También se modifican *payloads* y URLs en peticiones POST para intentar actualizar, eliminar o volver a publicar vuelos no existentes, ya publicados o pertenecientes a otros *managers*. El sistema responde correctamente en todos los casos, denegando el acceso con mensajes de "Access Unauthorised".
- Listandshow.hack: Este conjunto de pruebas verifica la protección del sistema frente a accesos no autorizados. En el caso de show, se inyectan IDs inválidos o no autorizados en la URL para intentar acceder a vuelos que no existen, pertenecen a otros *managers* o no han sido publicados, todo ello estando autenticado como un *manager* diferente. En el caso de list, se simulan intentos de acceder a la lista de vuelos de otro *manager*, comprobando que el sistema impide la visualización de información no autorizada y responde adecuadamente con "Access Unauthorised".
- Listandshow.safe: Este conjunto de pruebas verifica que un *manager* pueda listar correctamente sus vuelos y visualizar los detalles de cada uno de ellos. En la parte de list, se comprueba que todos los vuelos asociados a manager0 se recuperan y muestran correctamente. En la parte de show, se asegura que la información individual de cada vuelo perteneciente a manager0 se renderiza de forma adecuada.
- Update.safe: Se actualizan datos de varios vuelos pertenecientes a manager0 utilizando tanto entradas válidas aceptadas por el sistema como entradas no válidas, de forma similar a las pruebas realizadas en create.safe.
- Publish.safe: Evalúa la capacidad de publicar vuelos. Se publican correctamente vuelos válidos y se verifica que aquellos que no tienen legs o que contienen legs no publicados no puedan ser publicados, mostrando los mensajes de error correspondientes

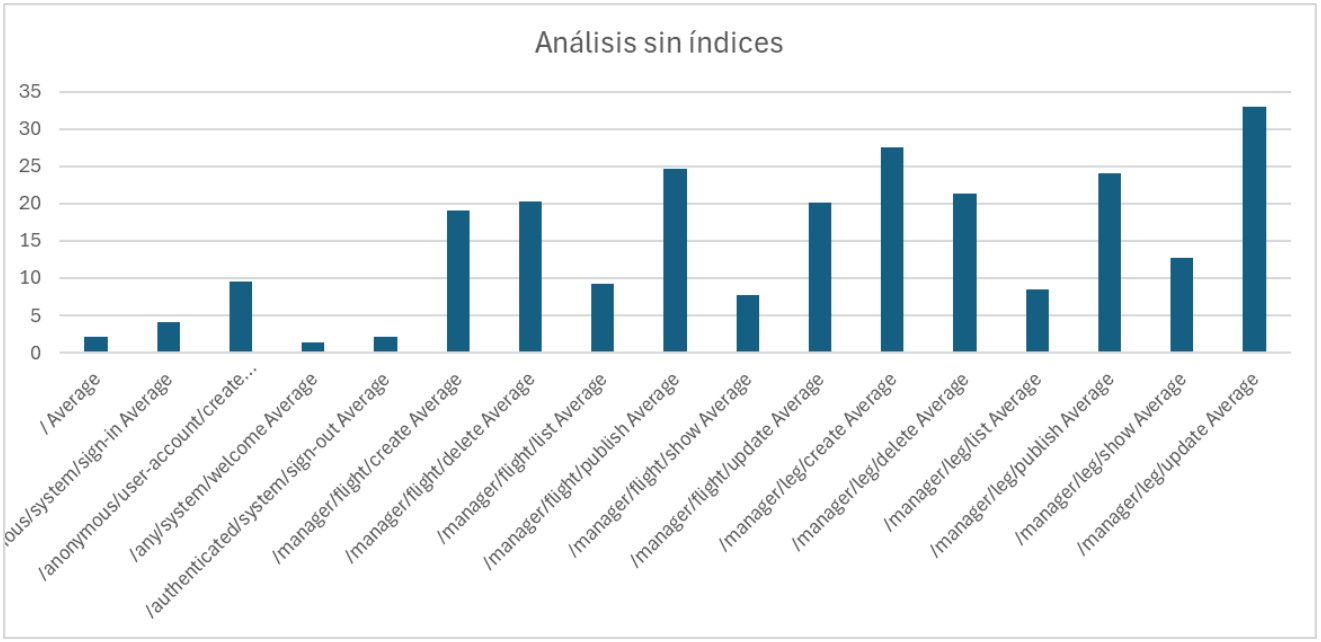
Pruebas para leg:

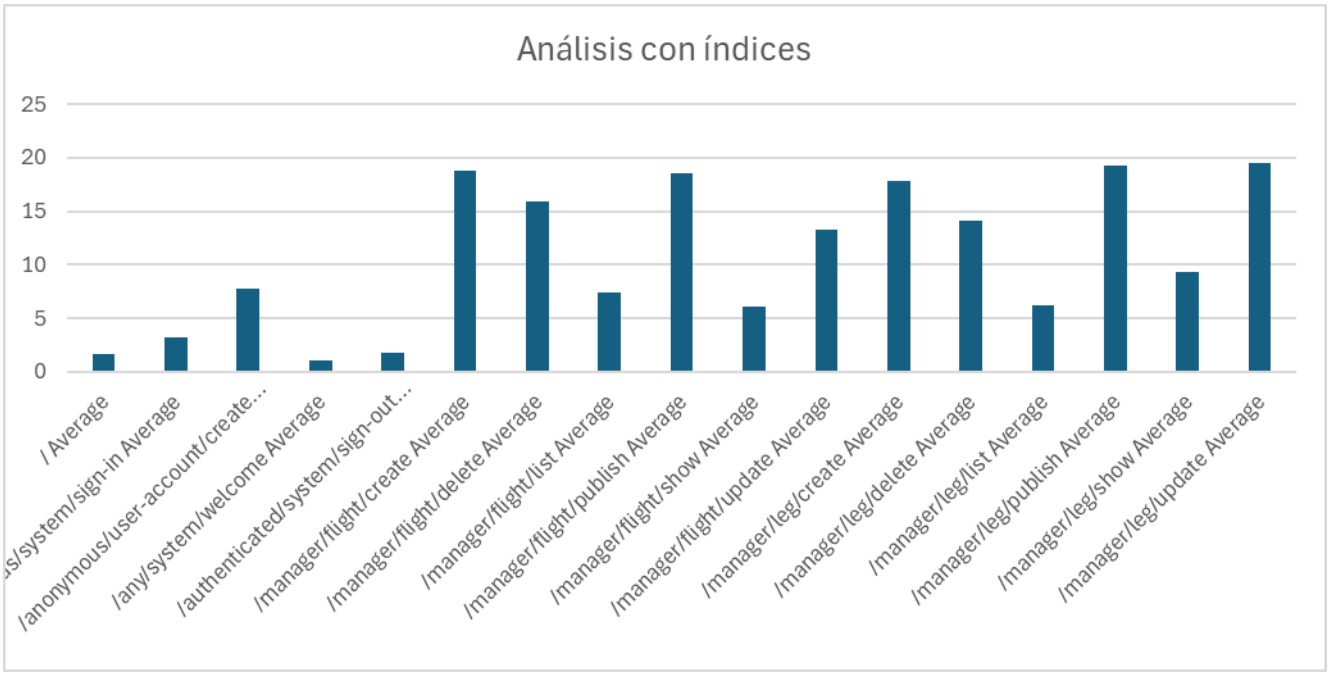
- Create.hack: Simula manipulaciones en el lado del cliente utilizando herramientas de desarrollador del navegador (por ejemplo, F12) para alterar manualmente los IDs de relaciones como *aircraft* o *airports* durante el proceso de creación de un *leg*. Estas pruebas confirman que el sistema detecta y bloquea dichos intentos ilegítimos, devolviendo los mensajes de error correspondientes. Además, se verifica que un usuario distinto o no autenticado no pueda crear *legs* accediendo directamente a la URL.
- Create.safe: Prueba la creación de *legs* utilizando datos de entrada válidos aceptados por el sistema. También incluye intentos de creación sin introducir datos, con el objetivo de verificar que se generan los mensajes de error de validación correspondientes. Además, se evalúan escenarios con entradas no válidas, como el uso de un formato de fecha incorrecto o el intento de registrar un número de vuelo duplicado. En todos estos casos, se espera que el sistema devuelva mensajes de error claros y adecuados.
- Delete.hack: Simula intentos de eliminación no autorizada de registros de *leg* mediante la manipulación de parámetros de la solicitud o de IDs relacionadas. Las pruebas se realizan estando autenticado como un usuario no autorizado o sin haber iniciado sesión. El sistema debe bloquear correctamente este tipo de acciones y evitar la eliminación indebida de datos.
- Delete.safe: Confirma que manager0 puede eliminar correctamente los registros de *leg* seleccionados. Verifica que las entradas correspondientes se eliminan de forma adecuada y sin errores.
- List.hack: Se realizan pruebas de GET hacking accediendo a URLs vinculadas a otros managers, tanto desde usuarios no autenticados como desde cuentas sin los permisos adecuados. Además, se intenta acceder a listas de legs pertenecientes a vuelos de otros managers mientras se está conectado como un usuario distinto. En todos los casos, el sistema actúa correctamente, bloqueando el acceso no autorizado y devolviendo el mensaje "Access Unauthorised".
- Listandshow.safe: Este conjunto de pruebas verifica que un *manager* pueda recuperar y listar correctamente todos sus registros de *leg*. Se confirma que los *legs* asociados a manager0 se obtienen y muestran de forma adecuada. Asimismo, se comprueba que la información individual de cada *leg* perteneciente a manager0 se visualiza correctamente.

- Publish.hack: Prueba la publicación de *legs* utilizando herramientas de desarrollador (F12), inspeccionando los campos y modificando los valores de los enumerados, lo que genera respuestas de "Invalid data". En el resto de los campos, se alteran manualmente los valores de las IDs por otros inválidos, lo que provoca respuestas de "Access Unauthorised". Además, se verifica que un usuario distinto o no autenticado no pueda publicar *legs* accediendo directamente a la URL.
- Publish.safe: Prueba la publicación de *legs* que anteriormente no estaban publicados. Verifica que los *legs* elegibles, es decir, aquellos que cumplen todas las condiciones requeridas, se transfieren correctamente al estado de publicados.
- Show.hack: Simula manipulaciones en el lado del cliente utilizando herramientas de desarrollador del navegador (por ejemplo, F12) para alterar manualmente los IDs de relaciones, como *aircraft* o *airports*, durante la creación de un *leg*. Estas pruebas confirman que el sistema detecta y bloquea los intentos ilegítimos, devolviendo los mensajes de error correspondientes. Además, se verifica que ni un *manager* distinto del creador ni un usuario no autenticado puedan acceder al proceso de creación.
- Update.hack: Prueba la actualización de legs utilizando herramientas de desarrollador (F12), inspeccionando los campos y modificando los valores de los enumerados, lo que genera respuestas de "Invalid data". En el resto de los campos, se alteran manualmente los valores de las IDs por otros inválidos, lo que da lugar a respuestas de "Access Unauthorised". Además, se verifica que ni un usuario distinto ni un cliente no autenticado puedan actualizar legs accediendo directamente a la URL.
- Update.safe: Se actualizan datos de varios *legs* pertenecientes a manager0, utilizando tanto entradas válidas aceptadas por el sistema como entradas no válidas, siguiendo un enfoque similar al empleado en las pruebas de create.safe.

- **<u>Análisis de rendimiento</u>**

Las pruebas de rendimiento se llevaron a cabo en el mismo ordenador. Para ello, se ejecutaron dos veces todas las trazas mostradas previamente: primero sin índices en la base de datos y, en una segunda ejecución, con los índices aplicados, con el objetivo de mejorar el rendimiento reduciendo el tiempo de resolución de las consultas. A continuación, se muestran las tablas de tiempos obtenidas en ambos análisis:





Los gráficos comparativos muestran de forma clara que, al aplicar los índices, todos los tiempos medios de resolución de consulta disminuyen. Por ejemplo, las operaciones más pesadas pasan de un rango de 20–30 ms sin índices a 12–20 ms con ellos. Esta reducción global se refleja también en una media general que baja de ≈ 9,5 ms a ≈ 6,7 ms.

Para evaluar el impacto de los índices, se ejecutaron las mismas mediciones sin índices y mediciones con índices, siguiendo los pasos de limpieza y cálculo de “time” en Excel descritos en S02 – Performance testing. A continuación, se obtuvieron estadísticas descriptivas y los intervalos de confianza al 95%:

Sin índices	
Mean	10,5088
Standard Error	0,63659
Median	5,7711
Mode	#N/A
Standard Deviation	13,6237
Sample Variance	185,604
Kurtosis	9,37082
Skewness	2,6817
Range	88,8905
Minimum	0,7067
Maximum	89,5972
Sum	4813,04
Count	458
Confidence Level(95,0%)	1,25101

Interval (ms)	9,25779956	11,7598205
Interval (s)	0,0092578	0,01175982

Con índices	
Mean	7,526807202
Standard Error	0,410932059
Median	4,63925
Mode	1,4141
Standard Deviation	9,059164769
Sample Variance	82,0684663
Kurtosis	18,91747246
Skewness	3,120510848
Range	93,6291
Minimum	0,6367
Maximum	94,2658
Sum	3658,0283
Count	486
Confidence Level(95,0%)	0,80742696

Interval (ms)	8,334234161	8,334234161
Interval (s)	0,008334234	0,008334234

Observamos que el intervalo de confianza con índices no se solapa con el de “sin índices” y la media de respuesta desciende de 10,51 ms a 7,53 ms.

Para confirmar que esta reducción es significativa, se realizó un Z-Test de dos muestras ($\alpha = 0,05$):

z-Test: Two Sample for Means		
	before	*after*
Mean	9,394900379	6,705802827
Known Variance	185,6041965	82,0684663
Observations	528	566
Hypothesized Mean Difference	0	
z	3,816260512	
P(Z<=z) one-tail	6,77447E-05	
z Critical one-tail	1,644853627	
P(Z<=z) two-tail	0,000135489	
z Critical two-tail	1,959963985	

Podemos observar que:

- $p\text{-}\alpha < 0.05$

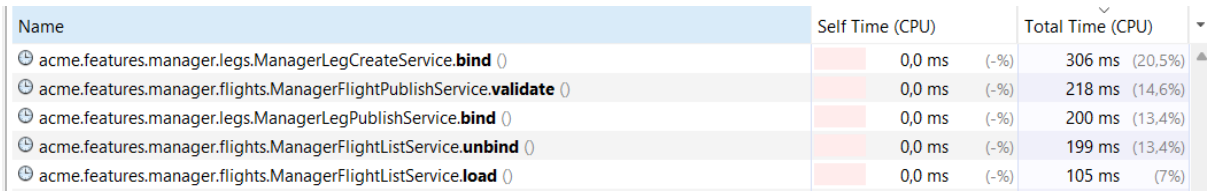
Por tanto, rechazamos la hipótesis nula de igualdad de medias y concluimos que la media "con índices" es significativamente menor que la "sin índices".

Por lo que podríamos concluir que la introducción de índices adecuados sobre las columnas críticas ha logrado mejorar de forma consistente el rendimiento de las consultas, reduciendo el tiempo medio de respuesta en un 28 % aproximadamente, una mejora que queda validada estadísticamente.

5. Software Profiling

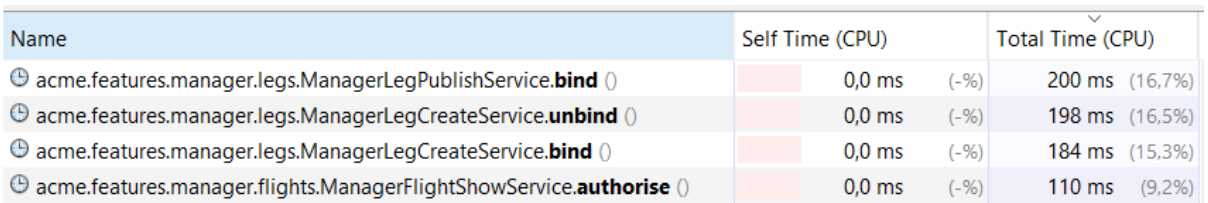
Además de las pruebas funcionales y de rendimiento, se llevó a cabo una prueba de Software Profiling para medir el consumo de CPU de los servicios clave en los módulos ManagerLeg y ManagerFlight. Con él identificamos los puntos más costosos y comprobamos cómo varían tras aplicar los índices en la base de datos:

Sin índices:



Name	Self Time (CPU)		Total Time (CPU)	
acme.features.manager.legs.ManagerLegCreateService.**bind** ()	0,0 ms	(-%)	306 ms	(20,5%)
acme.features.manager.flights.ManagerFlightPublishService.**validate** ()	0,0 ms	(-%)	218 ms	(14,6%)
acme.features.manager.legs.ManagerLegPublishService.**bind** ()	0,0 ms	(-%)	200 ms	(13,4%)
acme.features.manager.flights.ManagerFlightListService.**unbind** ()	0,0 ms	(-%)	199 ms	(13,4%)
acme.features.manager.flights.ManagerFlightListService.**load** ()	0,0 ms	(-%)	105 ms	(7%)

Con índices:



Name	Self Time (CPU)		Total Time (CPU)	
acme.features.manager.legs.ManagerLegPublishService.**bind** ()	0,0 ms	(-%)	200 ms	(16,7%)
acme.features.manager.legs.ManagerLegCreateService.**unbind** ()	0,0 ms	(-%)	198 ms	(16,5%)
acme.features.manager.legs.ManagerLegCreateService.**bind** ()	0,0 ms	(-%)	184 ms	(15,3%)
acme.features.manager.flights.ManagerFlightShowService.**authorise** ()	0,0 ms	(-%)	110 ms	(9,2%)

En concreto, ManagerLegCreateService.bind() reduce su tiempo de CPU en un 40 % (de 306 ms a 184 ms), y los servicios de Flight dejan de aparecer en los primeros puestos o lo hacen con valores muy inferiores.

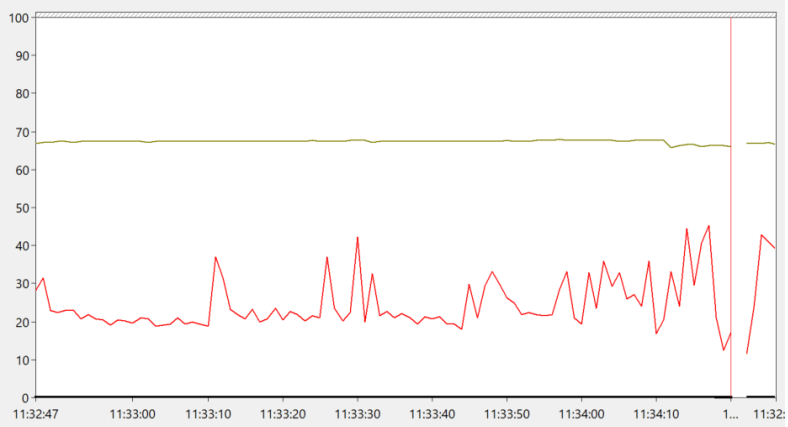
Por lo que vemos que la introducción de índices no solo acelera las consultas, sino que alivia la carga de CPU en los métodos más críticos, mejorando la eficiencia global del sistema y desplazando los hotspots a niveles de consumo mucho menores.

6. Hardware Profiling

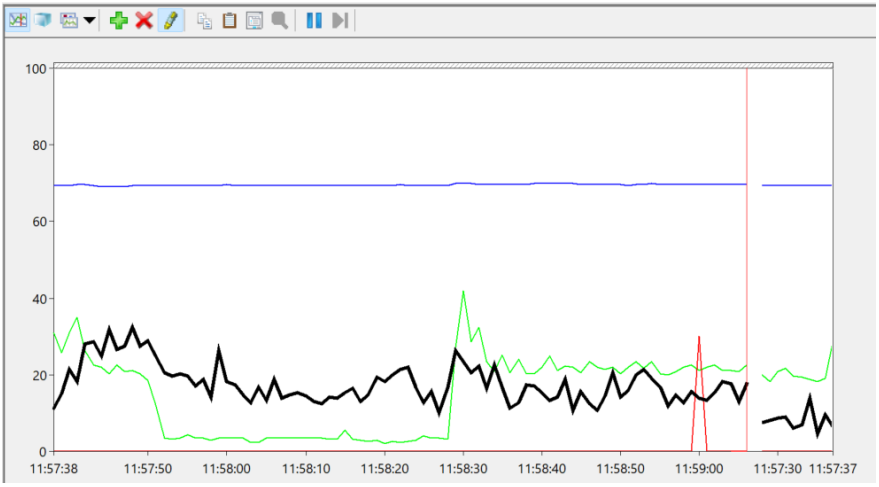
En los dos gráficos de Performance Monitor –el primero sin índices y el segundo con índices– hemos seguido cuatro contadores clave:

1. Longitud actual de la cola de disco (verde)
2. Bytes totales/s (azul)
3. % Committed Bytes In Use (rosa)
4. % Processor Time (rojo)

Sin índices:



Con índices:



Conclusión de monitorización de recursos:
Al comparar ambos gráficos –Sin índices (arriba) y Con índices (abajo)– sobre los contadores clave, obtenemos:

- Longitud de la cola de disco
  - *Sin índices:* picos frecuentes que alcanzan valores de 30–40, indicando contención de I/O.
  - *Con índices:* la cola se aplanó y los picos son menos numerosos y de menor magnitud, señal de menor bloqueo de disco.
- Throughput (Bytes totales/s)
  - *Sin índices:* oscilaciones pronunciadas, sobre todo en los momentos de grabación de trazas.
  - *Con índices:* curva más estable y con menos variabilidad, fruto de consultas más eficientes.
- Memoria (% Committed Bytes In Use)
  - En ambos casos se mantiene prácticamente constante (~67 %), sin fugas ni sobrecargas, lo que refleja estabilidad en el uso de RAM.
- CPU (% Processor Time)
  - *Sin índices:* oscilaciones amplias entre 20 % y 40 %, con picos irregulares que pueden alargar la latencia en operaciones pesadas.
  - *Con índices:* perfil más homogéneo, con picos reducidos (aprox. 10 %–30 %) y menos variabilidad.

En conjunto, la aplicación de índices reduce la presión sobre el subsistema de E/S (menos contención de disco) y alivia la carga de CPU (picos más bajos y uniformes), sin afectar negativamente al uso de memoria. Todo ello se traduce en una respuesta más rápida y consistente del sistema bajo carga.

7. Bibliography

intentionally blank.