**INFORME DE PRUEBAS S02**

* **Grupo**: C1.047
* **Repositorio**: <https://github.com/JoaquinBorjaLeon/C1-047-Acme-ANSD04>
* **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)
* **Tabla de versiones**:

| Fecha | Versión | Autor |
| --- | --- | --- |
| 26/05/2025 | 1.0.0 | Ariel Escobar Capilla |

* **Índice**:

[Resumen ejecutivo](#_o9f227ha6hr)

[Pruebas funcionales](#_tg6oaib3caym)

[Pruebas de rendimiento](#_57jtsv3mnsl0)

[Conclusión](#_vpcsxfp1nwap)

[Bibliografía](#_5livvrfre5p0)

## **Resumen ejecutivo**

En este informe presento las pruebas funcionales y de rendimiento que llevé a cabo para los requisitos #8 y #9 del Student-02, mi estudiante.

El objetivo fue asegurar que todas las funcionalidades se comportarán 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.

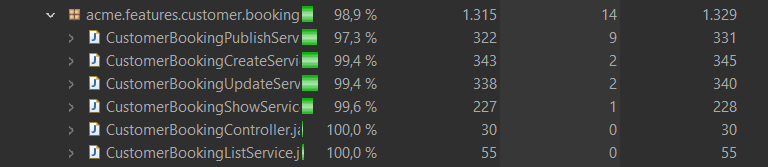
# **Pruebas funcionales**

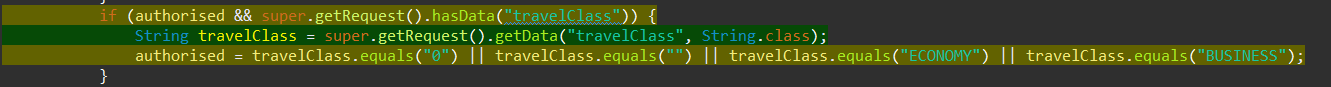
Para cada característica se han realizado tanto casos de prueba positivos y negativos (funcionalidad.safe) como intentos de hacking (funcionalidad.hack). Para la realización de dichas pruebas, se ha seguido las recomendaciones dictadas en el documento “L04-S01 – Formal testing”:

**Entidad Booking**:

| **Test** | **Descripción** |
| --- | --- |
| list.safe | Simplemente se accede al listado de Bookings para comprobar que funciona correctamente. |
| list.hack | Se intenta hacer GET Hacking para acceder al listado de Bookings tanto con la sesión cerrada como con un usuario de rol Administrator y así comprobar que no están autorizados. |
| show.list | Simplemente se accede a un Booking para comprobar que el formulario con su información funciona correctamente. |
| show.hack | Se intenta hacer GET Hacking cambiando en la URI el id del Booking correspondiente por los valores 0 y -1, se quita el campo id de la URI dejando solo “show” y finalmente se intenta acceder a la información de ese Booking con la sesión cerrada, con un Customer diferente y con un usuario de rol Administrator. Todo ello para comprobar que son peticiones no autorizadas. |
| create.safe | Se accede a un formulario de creación de Booking probando todas las validaciones posibles para ver si funcionan correctamente, tanto en campos desplegables como de escritura manual. Finalmente, se crea un Booking válido. |
| create.hack | Se intenta hacer POST Hacking en los desplegables del formulario de creación poniendo el valor -1. También se intenta hacer GET Hacking accediendo al formulario con la sesión cerrada y con un usuario del rol Administrator. Se comprueba que todas las peticiones son desautorizadas. |
| update.safe | Se accede a un formulario de actualización de un Booking concreto, probando todas las validaciones posibles para ver si funcionan correctamente, tanto en campos desplegables como de escritura manual. Finalmente, se actualiza un Booking con datos válidos. |
| update.hack | Se intenta hacer GET Hacking de dos formas: primero intentando actualizar el Booking a través de la URI, comprobando que no sucede nada. Después, intentando actualizar un Booking ya publicado; luego quitando la id de la URI, sustituyéndola por los valores 0 y -1 y, por último, intentando actualizar ese Booking con la sesión cerrada, con un Customer diferente y con un usuario de rol Administrator. En todos los casos se devuelve un error de petición no autorizada.  También se intenta hacer POST Hacking en los campos de solo lectura para comprobar que no ocurre nada, así como poner el valor -1 en los desplegables para comprobar que se desautoriza la petición. |
| publish.safe | Se comprueban todos los casos en los que un Booking no se podría publicar, viendo que el botón no aparece, y después se publica correctamente. También se intenta, con el botón visible, vaciar el campo de la tarjeta de crédito (necesario para publicarse) y darle a publicar, para que salte el mensaje de validación correspondiente. |
| publish.hack | Se intenta hacer GET Hacking de dos formas: primero intentando publicar un Booking a través de la URI, comprobando que no sucede nada. Después, intentando publicar un Booking ya publicado y uno que no cumple las condiciones necesarias; luego quitando la id de la URI, sustituyéndola por los valores 0 y -1 y, por último, intentando publicar ese Booking con la sesión cerrada, con un Customer diferente y con un usuario de rol Administrator. En todos los casos se devuelve un error de petición no autorizada.  También se intenta hacer POST Hacking en los campos de solo lectura para comprobar que no ocurre nada, así como poner el valor -1 en los desplegables para comprobar que se desautoriza la petición. |

En esta entidad, la cobertura es la siguiente:



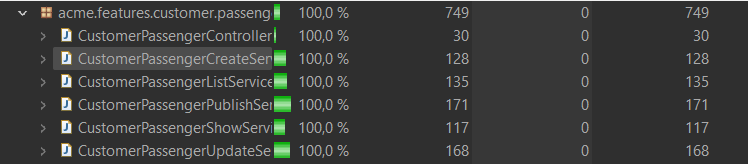
El pequeño porcentaje restante se debe composiciones lógicas de AND y/o OR en los que nunca se podrá cubrir todas las ramas, ya sea porque cumplir una condición implica que la otra no lo haga, o porque es una condición que siempre se va a cumplir pero por seguridad es mejor contemplar su negación. Por ejemplo:  
  




**Entidad Passenger**:

| **Test** | **Descripción** |
| --- | --- |
| list.safe | Simplemente se accede al listado de Passengers, asignados o no a un Booking, para comprobar que funciona correctamente. |
| list.hack | Se intenta hacer GET Hacking para acceder al listado de Passengers tanto con la sesión cerrada como con un usuario de rol Administrator, además de acceder a un listado de Passengers asignados a un Booking con otro Customer y así comprobar que no están autorizados. |
| show.list | Simplemente se accede a un Passenger para comprobar que el formulario con su información funciona correctamente. |
| show.hack | Se intenta hacer GET Hacking cambiando en la URI el id del Passenger correspondiente por los valores 0 y -1, se quita el campo id de la URI dejando solo “show” y finalmente se intenta acceder a la información de ese Passenger con la sesión cerrada, con un Customer diferente y con un usuario de rol Administrator. Todo ello para comprobar que son peticiones no autorizadas. |
| create.safe | Se accede a un formulario de creación de Passenger probando todas las validaciones posibles para ver si funcionan correctamente. Finalmente, se crea un Passenger válido. |
| create.hack | Se intenta hacer GET Hacking accediendo al formulario con la sesión cerrada y con un usuario del rol Administrator. Se comprueba que todas las peticiones son desautorizadas. |
| update.safe | Se accede a un formulario de actualización de un Passenger concreto, probando todas las validaciones posibles para ver si funcionan correctamente. Finalmente, se actualiza un Passenger con datos válidos. |
| update.hack | Se intenta hacer GET Hacking de dos formas: primero intentando actualizar el Passenger a través de la URI, comprobando que no sucede nada. Después, intentando actualizar un Passenger ya publicado; luego quitando la id de la URI, sustituyéndola por los valores 0 y -1 y, por último, intentando actualizar ese Passenger con la sesión cerrada, con un Customer diferente y con un usuario de rol Administrator. En todos los casos se devuelve un error de petición no autorizada. |
| publish.safe | Se intenta modificar algún campo del formulario con una valor inválido y luego pulsar el botón de publicar, para comprobar que los mensajes de validación correspondientes. Después, se publica correctamente un Passenger. |
| publish.hack | Se intenta hacer GET Hacking de dos formas: primero intentando publicar un Passenger a través de la URI, comprobando que no sucede nada. Después, intentando publicar un Passenger ya publicad; luego quitando la id de la URI, sustituyéndola por los valores 0 y -1 y, por último, intentando publicar ese Passenger con la sesión cerrada, con un Customer diferente y con un usuario de rol Administrator. En todos los casos se devuelve un error de petición no autorizada. |

En esta entidad, la cobertura es la siguiente:



En este caso, todas las líneas están completamente cubiertas.

**Entidad BookingRecord**:

| **Test** | **Descripción** |
| --- | --- |
| create.safe | Se accede a un formulario de creación de BookingRecord, probando a enviar el único desplegable que hay sin ningún valor para ver que las validaciones funcionan correctamente. Finalmente, se crea un BookingRecord válido. |
| create.hack | Se intenta hacer GET Hacking accediendo al formulario con la sesión cerrada, con un Customer que no sea dueño del Booking asociado y con un usuario del rol Administrator. Se comprueba que todas las peticiones son desautorizadas.  También se intenta hacer POST Hacking intentando añadir un Passenger válido, como por ejemplo uno cuyo id sea -1, otro cuyo id pertenezca a otro Customer u otro que ya fue asociado previamente al Booking. |

En esta entidad, la cobertura es la siguiente:



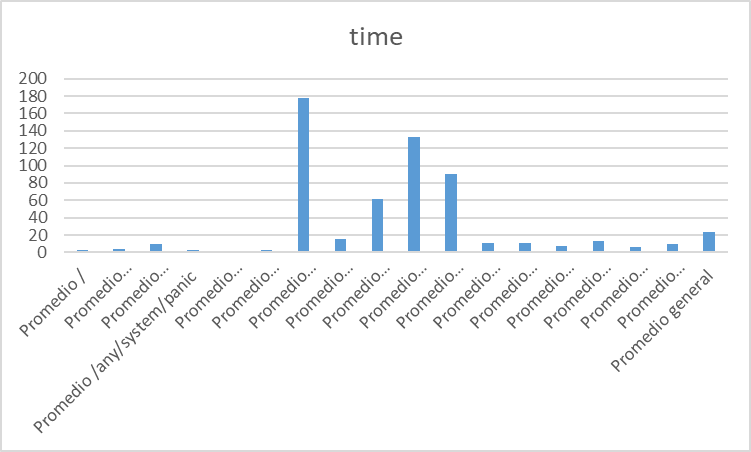
En este caso, todas las líneas están completamente cubiertas.

# **Pruebas de rendimiento**

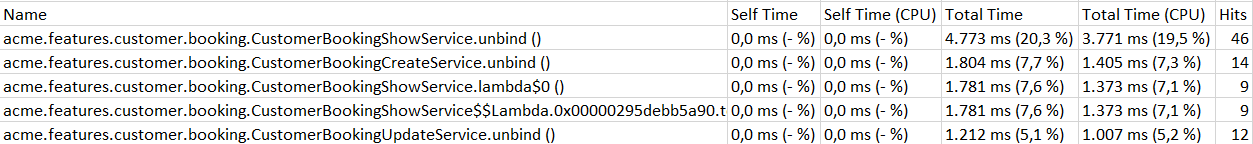
Posteriormente, se han llevado a cabo pruebas de rendimiento con el objetivo de optimizar el acceso a los datos y mejorar los tiempos de respuesta de los distintos servicios involucrados. Para ello, se han introducido índices en los campos clave de las entidades (en este caso se han añadido a Booking, BookingRecord y Passenger, y se han usado los índices creados por otros compañeros en Flight y Leg), acelerando de forma significativa las consultas a la base de datos, especialmente en las consultas de filtrado y búsqueda. Como resultado, se ha conseguido una mejora notable en los tiempos de ejecución, lo que impacta positivamente en la experiencia del usuario y en la eficiencia general del sistema.

**Análisis de la mejora:**

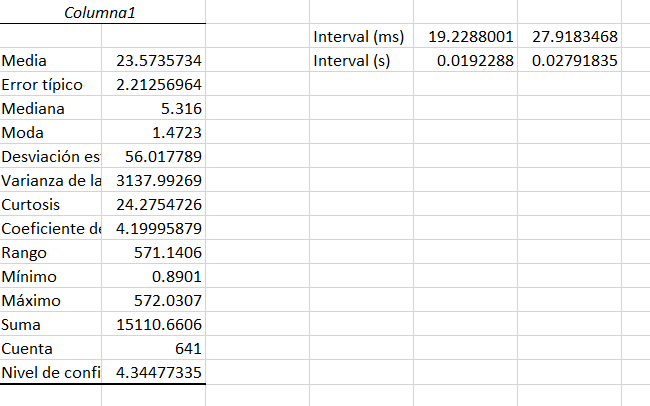
A continuación se mostrarán los datos antes de poner los índices:



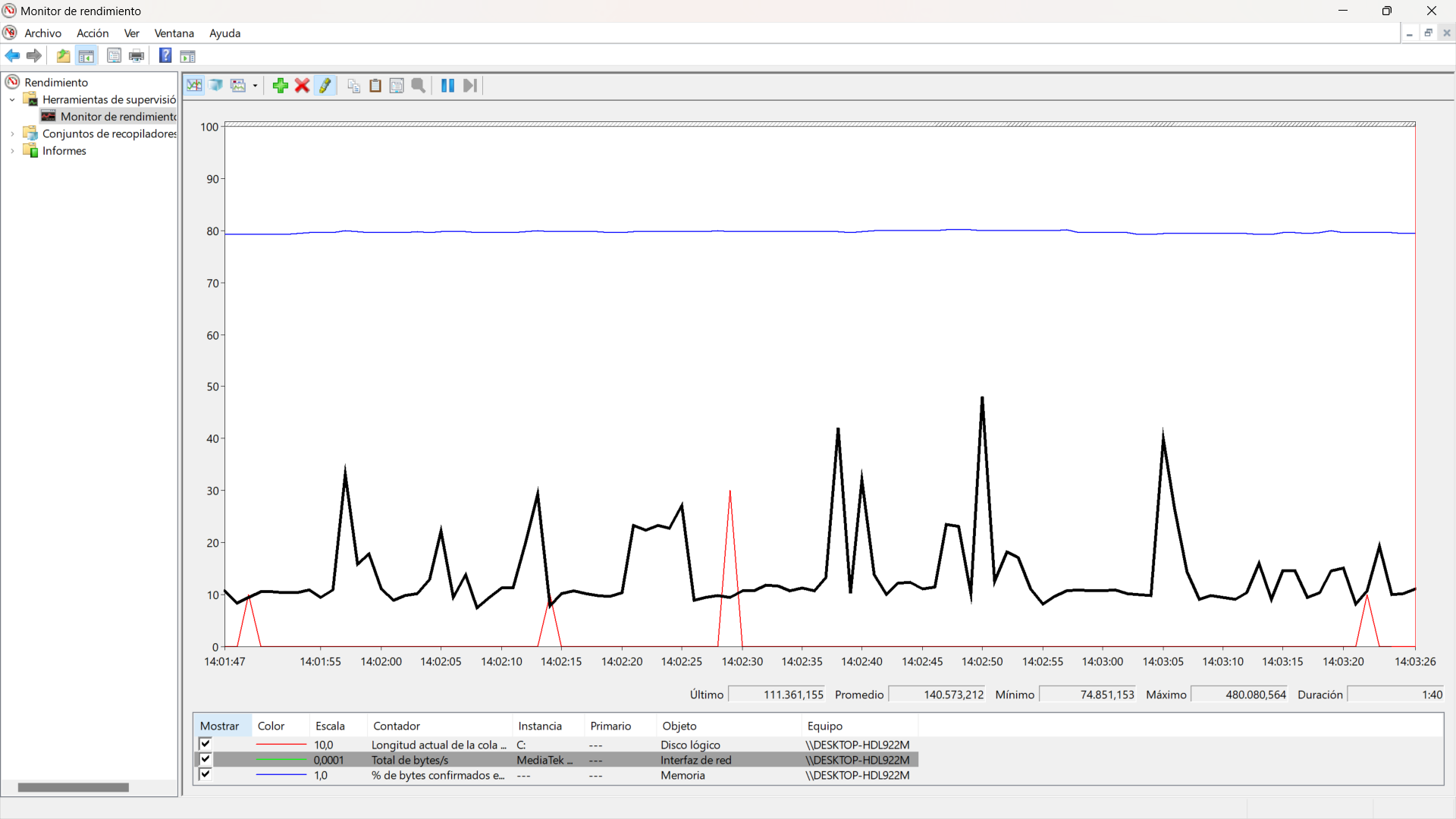
Tiempo por método en ms.



Métodos que más tardaron en ejecutarse.

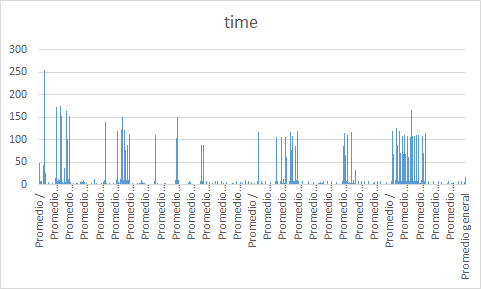


Estadística descriptiva.

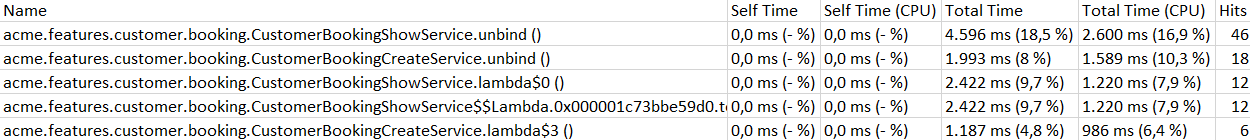


Rendimiento.

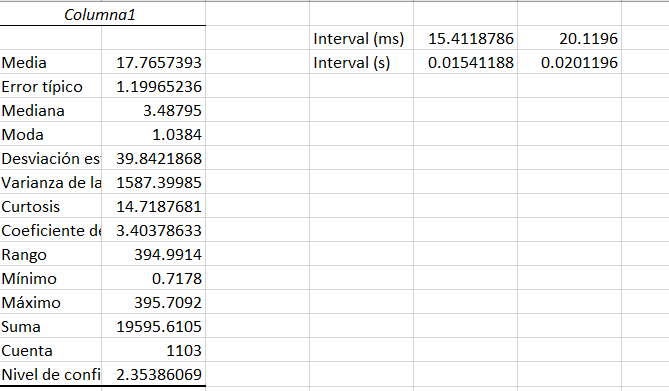
Ahora veremos las mismas medidas, pero después de poner los índices:



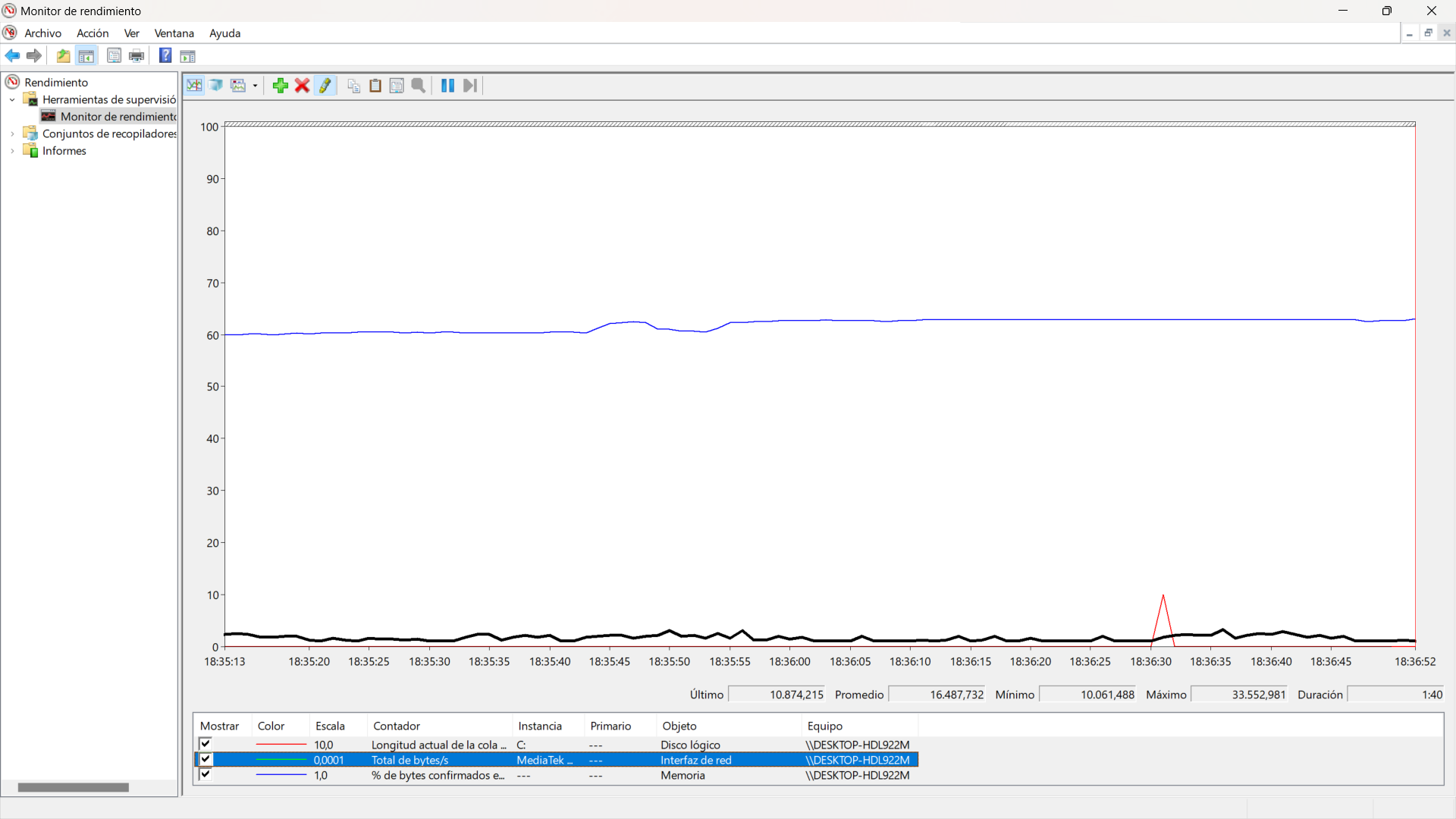
Tiempo por método en ms.



Métodos que más tardaron en ejecutarse.



Estadística descriptiva.



Rendimiento.

A simple vista, con estos datos podemos comprobar que la inclusión de los índices ha sido muy efectiva.

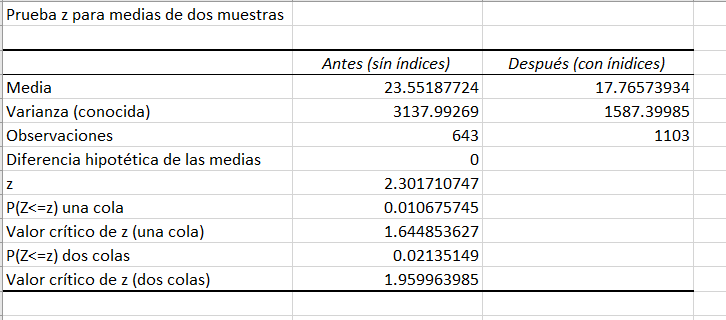
A nivel de tiempo de ejecución, se aprecia que aunque se hacen más peticiones el tiempo total que tardan en ejecutarse acaba siendo menor. Por ejemplo, el método *unbind()* del servicio *Show* de *Booking* fue en ambas ocasiones el más lento, pero su tiempo de ejecución por la CPU bajó de 3771 ms a 2600 ms.

En cuanto a la estadística descriptiva, nos fijamos en que el nivel de confianza disminuye considerablemente, de 4.34 a 2.33.

Por último, en los gráficos del motor de rendimiento vemos quizás las diferencias más considerables, con un uso de memoria promedio de ~134MB, varias colas de espera en el disco, y sobre todo unos picos de memoria muy altos y frecuentes. En general, una gráfica inestable.

En cambio, al introducir los índices se aprecia como todos los valores se estabilizan, con un uso de memoria promedio de ~15MB, picos de memoria muy bajos y solo un pico puntual de cola de espera en el disco.

Ahora haremos la prueba Z-Test con la que veremos si el *p-value* con dos colas está dentro del intervalo [0, a], siendo a 1-0.95=0.05:



Como vemos, su valor es de 0.021, que se encuentra en la mitad inferior del intervalo, por lo que podemos confirmar definitivamente que la mejora del rendimiento ha sido muy significativa.

# **Conclusión**

Después de añadir los índices a las entidades principales del sistema, se ha observado una mejora notable en el rendimiento. Antes de incluirlos, el uso de memoria era más elevado y variable, y se producían accesos al disco de forma constante. Esto generaba tiempos de respuesta más largos y un comportamiento menos estable durante la ejecución.

Tras introducir los índices adecuados en campos clave las consultas se han vuelto más rápidas y el sistema más eficiente. Los gráficos obtenidos durante las pruebas muestran que el consumo de memoria se estabiliza y se reduce considerablemente, y que las operaciones de acceso a disco casi desaparecen.

Esta mejora se debe a que los índices permiten acceder a los datos directamente, sin necesidad de recorrer toda la base de datos. Además, algunos índices ayudan también a evitar registros duplicados en algunas combinaciones.

En resumen, aplicar los índices ha sido una mejora importante que reduce la carga del sistema, acelera las operaciones y mejora la experiencia de uso de la aplicación.

# **Bibliografía**

* Enseñanza Virtual de la Universidad de Sevilla: [ev.us.es](http://ev.us.es)