Universidad de Sevilla  
Escuela Técnica Superior de Ingeniería Informática



Grado en Ingeniería Informática – Ingeniería del Software  
Diseño y Pruebas II

Curso 024 – 2025

**Informe de testing**

**Grupo:** C1.060

**Número de estudiante dentro del grupo:** 2

**Repositorio**: <https://github.com/DP2-C1-060/Acme-ANS-D04>

**Fecha**: 26/05/2025

|  |  |
| --- | --- |
| **Alumno** | **Correo** |
| Nuñez Sanchez,Juan | juanunsan2@alum.us.es |

**Historial de versiones**

|  |  |  |
| --- | --- | --- |
| **Versión** | **Fecha** | **Descripción de los cambios** |
| V1.0 | 26/05/2025 | Creación y elaboración del informe |
|  |  |  |

**Índice de contenido**

[**1.** **Resumen ejecutivo** 2](#_Toc199170126)

[**2.** **Introducción** 2](#_Toc199170127)

[**3.** **Contenido** 3](#_Toc199170128)

[**3.1.** **Testing funcional** 3](#_Toc199170129)

[**3.1.1.** **Funcionalidad para los customer sobre los booking** 3](#_Toc199170130)

[**3.1.1.1.** **List** 3](#_Toc199170131)

[**3.1.1.2.** **Show** 3](#_Toc199170132)

[**3.1.1.3.** **Create** 4](#_Toc199170133)

[**3.1.1.4.** **Update** 4](#_Toc199170134)

[**3.1.1.5.** **Delete** 5](#_Toc199170135)

[**3.1.1.6.** **Publish** 6](#_Toc199170136)

[**3.1.2.** **Funcionalidad para los customer sobre los pasajeros** 7](#_Toc199170137)

[**3.1.2.1.** **List** 7](#_Toc199170138)

[**3.1.2.2.** **Show** 8](#_Toc199170139)

[**3.1.2.3.** **Create** 8](#_Toc199170140)

[**3.1.2.4.** **Update** 9](#_Toc199170141)

[**3.1.2.5.** **Delete** 9](#_Toc199170142)

[**3.1.2.6.** **Publish** 10](#_Toc199170143)

[**3.1.3.** **Funcionalidad para los customer sobre booking record (entidad intermedia)** 10](#_Toc199170144)

[**3.1.3.1.** **Create** 10](#_Toc199170145)

[**3.1.3.2.** **Delete** 11](#_Toc199170146)

[**3.1.3.3** **List** 12](#_Toc199170147)

[**3.1.3.4** **Show** 12](#_Toc199170148)

[**1.1.** **Testing de rendimiento** 12](#_Toc199170149)

[**1.1.1.** **Sin índices** 13](#_Toc199170150)

[**1.1.2.** **Con índices** 14](#_Toc199170151)

[**2.** **Conclusiones** 17](#_Toc199170152)

[**Bibliografía** 17](#_Toc199170153)

# **Resumen ejecutivo**

En este documento se describen con detalle los hallazgos de las pruebas funcionales y de rendimiento realizadas en el proyecto. En la sección de pruebas funcionales, se exponen los distintos casos de prueba desarrollados, acompañados de una explicación clara de cada uno y una valoración de su eficacia para detectar errores. Más adelante, en el bloque dedicado al rendimiento, se incluyen gráficos exhaustivos y se reporta un intervalo de confianza del 95 % para los tiempos de respuesta del sistema, tanto con índices activados como desactivados. Además, se lleva a cabo un análisis comparativo (con el mismo nivel de confianza del 95 %) que permite determinar de forma estadística si el uso de índices influye en el desempeño. En conjunto, este informe sigue un método riguroso y sistemático para asegurar la fiabilidad y utilidad de los resultados en la evaluación del proyecto.

# **Introducción**

En la sección dedicada a pruebas funcionales, se detallan los diferentes escenarios de test ideados para verificar cada funcionalidad del sistema. Para cada caso de prueba se ofrece una descripción exhaustiva y se analiza su capacidad de descubrir errores, permitiendo así evaluar el comportamiento del software bajo distintas condiciones y corregir posibles fallos de forma eficiente.

En el apartado de rendimiento, se examina el tiempo de respuesta del sistema al procesar solicitudes, comparando los resultados con y sin el uso de índices. Se incorporan gráficos ilustrativos y se reporta un intervalo de confianza del 95 % para los valores medidos. Además, mediante un contraste de hipótesis al mismo nivel de confianza, se valora estadísticamente si la implementación de índices influye de modo significativo en el desempeño. Todo el estudio se ha llevado a cabo siguiendo un enfoque metódico y sistemático, apoyado en herramientas de planificación y en el entorno de desarrollo Eclipse, para garantizar la exactitud y la eficacia de las pruebas.

# **Contenido**

# **Testing funcional**

En este apartado describiremos los distintos tipos de casos de prueba (positivos, negativos y de seguridad) que hemos desarrollado para cada funcionalidad del sistema. Asimismo, evaluaremos su eficacia en la detección de errores, apoyándonos en los datos de cobertura que ofrece Eclipse, que nos muestra el porcentaje de líneas de código ejecutadas durante la ejecución de los tests.

# **Funcionalidad para los customer sobre los booking**

Vamos a comenzar por las pruebas establecidas para las funcionalidades de los customer sobre los bookings.

# **List**

Esta funcionalidad consiste en que un customer pueda ver la lista de sus bookings.

Para realizar las pruebas positivas y negativas de esta funcionalidad nos hemos logueado como un customer y hemos listado sus bookings.

Para realizar las pruebas de hacking, con el rol anónimo hemos introducido al url de mostrar nuestros booking (…/customer/booking/list) comprobando que nos saltaba un error y los mismo, pero ahora con otro rol (manager en mi caso).

Tras realizar estas pruebas hemos obtenido en el servicio un 100% de cobertura lo cual nos asegura que la funcionalidad no esconde ningún posible fallo ya que ha ejecutado todas las instrucciones.

# **Show**

La funcionalidad permite a un cliente consultar en detalle cualquiera de sus reservas. Para verificarla, primero iniciamos sesión con un usuario de tipo customer y accedimos al listado de bookings asociados. A continuación, seleccionamos cada reserva, comprobando que los atributos (fecha, destino, estado, etc.) se mostraban correctamente en pantalla.

En cuanto a las pruebas de hacking, probamos dos escenarios de acceso no autorizado: primero, accedimos como anónimo y como administrator, y luego volvimos a un usuario customer distinto al propietario de la reserva. En todas estas simulaciones, el sistema denegó el acceso y mostró los mensajes de error pertinentes, asegurando así el aislamiento de datos entre usuarios.

Por último, el análisis de cobertura realizado con Eclipse indicó que se ejecutó el 100 % de las líneas de código implicadas en esta funcionalidad. Gracias a ello, podemos afirmar con total seguridad que no existen rutas sin testar ni escenarios sin cubrir.

# **Create**

La funcionalidad permite a un customer generar una nueva reserva definiendo todos los parámetros necesarios (fechas, destino, número de personas, etc.).

Para comprobar la solidez de la validación de datos, comenzamos con casos negativos: primero enviamos el formulario completamente vacío y, a continuación, fuimos introduciendo valores que, aunque aceptados por el frontend, debían disparar validaciones específicas en cada campo. De este modo cubrimos todos los posibles errores asociados a restricciones de formato, rangos inválidos y omisión de datos.

Acto seguido, llevamos a cabo las pruebas positivas: creamos reservas usando los valores límite y los distintos rangos permitidos según la metodología de clase (extremos de las restricciones, valores intermedios y fronterizos), confirmando que cada combinación válida se procesaba correctamente y se almacenaba sin incidencias.

En el apartado de seguridad, simulamos ataques de tipo hacking intentando invocar la URL de creación de reservas con roles no autorizados (anónimo y manager). En todos los intentos, el sistema denegó el acceso y devolvió los códigos de error correspondientes, asegurando el aislamiento de esta operación para usuarios de tipo customer.

Por último, gracias al informe de cobertura de Eclipse, verificamos que se alcanzó el 100 % de cobertura de las líneas de código relacionadas con esta funcionalidad. Esto nos permite afirmar con total confianza que no quedan rutas de ejecución sin testar..

# **Update**

La funcionalidad permite a un customer crear una nueva reserva definiendo todos los detalles necesarios (fechas, destino, número de personas, etc.).

Para poner a prueba la validación de datos, empezamos por los escenarios negativos: enviamos el formulario totalmente vacío y luego fuimos introduciendo valores que, aunque aceptados en la interfaz, debían desencadenar las comprobaciones del backend (formatos erróneos, rangos fuera de límites, campos omitidos…). De este modo, cubrimos todas las posibles vías de fallo en cada atributo.

A continuación, realizamos las pruebas positivas: generamos reservas empleando valores límite y combinaciones representativas según la metodología de clase (extremos, valores intermedios y distintos rangos permitidos). Comprobamos que cada petición válida se procesaba sin incidencias y quedaba correctamente almacenada.

En el apartado de seguridad, simulamos accesos no autorizados a la URL de creación de reservas con roles distintos al de customer (anónimo y manager). En todos los intentos, el sistema rechazó la petición y devolvió los códigos de error previstos, garantizando el aislamiento de esta operación para usuarios legítimos.

Por último, el informe de cobertura obtenido con Eclipse refleja un 100 % de ejecución de las líneas de código implicadas en esta funcionalidad. Gracias a ello, podemos afirmar con total seguridad que no existe ninguna ruta sin testar.

# **Delete**

La funcionalidad Delete permite a un customer borrar una de sus bookings siempre que ésta no esté publicada y no tenga pasajeros asociados.

Para las pruebas negativas, intentamos:

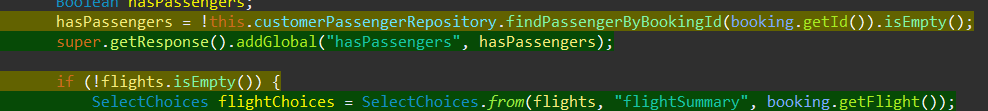
* Borrar un booking no publicado pero con pasajeros (debe rechazarse).
* Eliminar un booking publicado (el botón de borrado ni aparece).

El único caso positivo fue un booking no publicado y sin pasajeros, que se eliminó correctamente.

En las pruebas de hacking invocamos la URL de borrado con roles anónimo, manager y otro customer distinto; en todos los casos el sistema denegó el acceso con el mensaje de error apropiado.

Según el informe de cobertura de Eclipse, hemos alcanzado un 98,6 %. Las líneas marcadas en amarillo corresponden a tres rutas lógicas que no hemos cubierto:





No se ha conseguido pasar de amarillo a verde a pesar de haber probado a invocar la URL de borrado con un bookingId que no exista en la base de datos. Lo mismo ocurre al haber intentado borrar un booking con y sin pasajeros.

# **Publish**

La funcionalidad Publish permite a un customer hacer pública una de sus bookings siempre que se cumplan simultáneamente tres condiciones:

1. Existe al menos un pasajero asociado y publicado.
2. Todos los pasajeros vinculados estén marcados como publicados.
3. El campo lastNibble esté presente y sea válido (sin errores de formato).

**Pruebas funcionales negativas y positivas**

* **Negativas**: intentamos publicar reservas en las que faltaba al menos una de las condiciones anteriores:
  1. Pasajeros no publicados.
  2. Sin ningún pasajero asociado.
  3. lastNibble ausente o con formato inválido.
  4. Modificación de atributos para provocar fallos de validación.  
     En cada caso, el sistema devolvió el error adecuado y rechazó la operación.
* **Positivas**: finalmente, creamos un booking que cumplía todas las restricciones (pasajeros publicados y lastNibble correcto) y verificamos que la acción de publicar se realizaba con éxito.

**Pruebas de hacking**

1. Con roles anónimo y manager: la solicitud de publicación fue denegada.
2. Como customer distinto al propietario (tanto para bookings publicados como no publicados): acceso rechazado.
3. Con rol y usuario correctos, pero intentando publicar un booking ya publicado: el sistema evitó la duplicidad y devolvió el mensaje de error correspondiente.

Según el informe de cobertura de Eclipse, esta funcionalidad alcanza un 99,7% de ejecución de líneas de código:

Imagen cargada

. Estos casos han sido probados en la colección de tests y después de intentar repetirlo numerosas veces no se ha conseguido cambiar de amarillo a verde. Aun así, se puede asegurar la integridad de la aplicación y su correcto funcionamiento en esta funcionalidad.

# **Funcionalidad para los customer sobre los pasajeros**

Vamos a continuar por las pruebas establecidas para las funcionalidades de los customer sobre los pasajeros.

# **List**

La funcionalidad permite a un customer consultar tanto el listado completo de sus pasajeros como los pasajeros asociados a un booking concreto.

**Pruebas funcionales**

* **Escenarios positivos**: tras iniciar sesión como customer, accedimos a la sección de “Mis pasajeros” y comprobamos que se mostraban todos los registros vinculados al usuario. Luego seleccionamos un booking y listamos sus pasajeros, verificando que la información (nombre, apellidos, estado, etc.) coincidía con los datos esperados.
* **Escenarios negativos**: intentamos visualizar los pasajeros de un booking inexistente o de otro customer, recibiendo los errores de “recurso no encontrado” o “acceso denegado” según correspondiera.

**Pruebas de hacking**

* Desde roles no autorizados (anónimo y manager) intentamos obtener el listado global de pasajeros y el detalle de un booking ajeno; en todos los casos, la aplicación respondió con el código de error de autorización adecuado.
* Con el rol de customer pero usando credenciales de un usuario distinto al propietario del booking, intentamos acceder a sus pasajeros; la petición fue denegada correctamente.

Según el informe de cobertura de Eclipse, esta funcionalidad alcanza un 100 % de ejecución de líneas de código. Gracias a ello, podemos asegurar que todas las rutas (tanto de éxito como de fallo)han sido probadas exhaustivamente y no existe ningún fragmento sin testar.

# **Show**

El objetivo de esta funcionalidad es permitir a un customer consultar los detalles de cualquiera de sus pasajeros.

**Pruebas funcionales**

* **Positivas y negativas**: iniciamos sesión como customer, accedimos al listado de pasajeros y abrimos varias fichas, comprobando que cada atributo (nombre, apellidos, estado, etc.) se mostraba correctamente. También navegamos desde un booking a su lista de pasajeros y seleccionamos uno para verificar su información detallada.

**Pruebas de hacking**

* Con roles no autorizados (anónimo y manager) tratamos de visualizar los datos de un pasajero;
* Como customer, intentamos acceder al pasajero perteneciente a otro usuario.

En todos los casos, la aplicación devolvió el error correspondiente.

El informe de cobertura de Eclipse arroja un 100% de líneas ejecutadas, lo que confirma que no queda ninguna ruta sin probar.

# **Create**

Esta funcionalidad permite a un customer registrar un nuevo pasajero.

**Pruebas funcionales**

* **Negativas**: comenzamos enviando el formulario completamente vacío y, a continuación, fuimos introduciendo datos que debían fallar en cada campo (formatos incorrectos, valores fuera de rango, omisión de campos…), cubriendo así todas las posibles validaciones de error.
* **Positivas**: una vez verificados todos los rechazos, creamos pasajeros utilizando valores límite y combinaciones representativas según la metodología de clase (extremos de las restricciones y distintos rangos permitidos), confirmando que cada envío válido se procesaba correctamente.

**Pruebas de hacking y cobertura**

* Intentamos invocar la URL de creación de pasajero con roles anónimo y manager, comprobando que el acceso quedaba denegado en ambos casos.
* El informe de cobertura de Eclipse muestra un 100 % de ejecución de líneas de código, lo que nos garantiza que todos los flujos —tanto de éxito como de fallo— han sido probados exhaustivamente.

# **Update**

La funcionalidad permite a un customer modificar los datos de un pasajero siempre que éste no esté publicado.

**Pruebas funcionales**

* **Negativas**: enviamos formularios con datos erróneos o campos vacíos para cada atributo, asegurando que el sistema rechaza las actualizaciones inválidas.
* **Positivas**: a partir de pasajeros existentes, aplicamos valores límite y combinaciones válidas (siguiendo la metodología de clase) y comprobamos que cada cambio se guarda correctamente.

**Pruebas de hacking**

* Con roles anónimo y manager, intentamos ejecutar la operación de actualización.
* Como customer distinto al propietario del pasajero (tanto sobre pasajeros publicados como no publicados).
* Con el rol y usuario correctos, tratamos de actualizar un pasajero ya publicado.  
  En todos los casos, la aplicación devolvió el error de autorización o validación esperado.

El informe de cobertura de Eclipse muestra un 99,2 % de líneas ejecutadas:



. Estos casos han sido probados en la colección de tests y después de intentar repetirlo numerosas veces no se ha conseguido cambiar de amarillo a verde. Aun así, se puede asegurar la integridad de la aplicación y su correcto funcionamiento en esta funcionalidad.

# **Delete**

La funcionalidad permite a un customer eliminar un pasajero siempre que éste no haya sido publicado ni esté vinculado a ningún booking.

En las pruebas funcionales cubrimos únicamente el escenario positivo: borrado de un pasajero no publicado y sin reservas asociadas. No realizamos casos negativos de esta índole, puesto que la interfaz oculta el botón de eliminación para pasajeros ya publicados o con bookings, y nuestro validador lanza una excepción controlada si se fuerza la operación.

Para las pruebas de hacking replicamos el enfoque usado en el Update:

* Intentos de borrado con roles anónimo y manager.
* Como customer, pero sobre un pasajero que no era nuestro (tanto publicado como no publicado).
* Con rol y usuario correctos, tratando de eliminar un pasajero ya publicado.  
  En todos los casos, la aplicación denegó el acceso según lo esperado.

El informe de cobertura de Eclipse refleja un 100 % de ejecución de líneas de código, lo que confirma que todas las rutas han sido testeadas.

# **Publish**

La funcionalidad habilita a un customer para cambiar el estado de un pasajero a “publicado”.

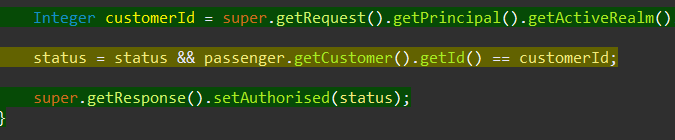
En las pruebas funcionales negativas primero intentamos publicar pasajeros provocando errores de validación al modificar campos, y después verificamos dos escenarios positivos: publicar un pasajero no publicado que tenía bookings asociados y, a continuación, uno sin ninguna reserva vinculada.

Para las **pruebas de hacking** cubrimos tres situaciones:

* Con roles ajenos al de customer (anónimo y manager).
* Con rol de customer pero usando credenciales de otro usuario, tanto sobre pasajeros publicados como no publicados.
* Con rol y usuario correctos, intentando reconvertir un pasajero ya publicado.

En todos los casos, el sistema rechazó la petición.

Finalmente, el informe de cobertura de Eclipse muestra un 100 % de líneas ejecutadas:



. Estos casos han sido probados en la colección de tests y después de intentar repetirlo numerosas veces no se ha conseguido cambiar de amarillo a verde. Aun así, se puede asegurar la integridad de la aplicación y su correcto funcionamiento en esta funcionalidad.

# **Funcionalidad para los customer sobre booking record (entidad intermedia)**

# **Create**

La funcionalidad permite a un customer incorporar nuevos pasajeros a un booking, siempre que dichos pasajeros pertenezcan al mismo customer y aún no estén asociados a esa reserva.

En las pruebas funcionales positivas, añadimos pasajeros (tanto publicados como no publicados) a bookings que ya contenían pasajeros, así como a reservas sin ningún pasajero previo, confirmando que cada adición se procesaba correctamente.  
Para las pruebas negativas, intentamos incluir un pasajero nulo y se produjo el error esperado.

En las pruebas de hacking cubrimos tres escenarios:

1. Con roles no autorizados (anónimo y manager).
2. Con rol de customer pero usando credenciales de un usuario distinto (sobre bookings y pasajeros tanto publicados como no publicados).
3. Con rol y usuario correctos, intentando añadir a un booking que ya estaba publicado.

En cada caso, la aplicación rechazó la petición con el mensaje de error adecuado.

Finalmente, el informe de cobertura de Eclipse refleja un **100 %** de ejecución de líneas de código, lo que garantiza que no queda ninguna ruta sin probar.

# **Delete**

La funcionalidad permite a un customer retirar pasajeros de un booking siempre que éste no esté publicado. En los casos positivos eliminamos pasajeros (publicados y no publicados) de reservas no publicadas y comprobamos que desaparecían sin dejar referencias. No fue necesario diseñar más negativos, pues el botón de eliminación se desactiva en bookings publicados y un validador previene cualquier borrado forzado.

Intentamos la operación con rol anónimo y manager; después como customer autenticado con un usuario distinto (sobre bookings publicados y no publicados); y por último como el customer correcto pero en un booking publicado. En cada escenario el sistema denegó el acceso con el código de error esperado.

El informe de Eclipse muestra un 97,8 %. La única línea que no se ejecuta es el método interno unbind, porque el flujo de eliminación usa directamente el repositorio y la interfaz (UI/API) nunca invoca ese método. Al no existir ningún punto de entrada para unbind en el diseño actual, no resulta posible cubrirlo con tests.

# **List**

La funcionalidad permite a un customer obtener el listado de todos los booking records, es decir, de las asociaciones entre sus reservas y pasajeros. Para verificarla, iniciamos sesión como customer y accedimos al listado de booking records. Comprobamos que se devuelven todas las entradas vinculadas a sus bookings, con los campos correctos.

Intentamos acceder al listado de booking records con roles anónimo y manager, recibiendo siempre un error de autorización. A continuación, nos autenticamos como customer B y solicitamos los booking records de un booking de customer A, obteniendo también la denegación correspondiente. Estos tests confirman que solo el propietario de la reserva puede consultar sus asociaciones.

El informe de cobertura de Eclipse refleja un 100 % de líneas ejecutadas en el controlador, servicio y repositorio de booking records.

# **Show**

Permite a un customer consultar los detalles de un booking record concreto, es decir, la asociación entre un booking y un pasajero (incluye campos como bookingId, passengerId, fecha de alta, rol dentro del booking, etc.). Iniciamos sesión como customer, listamos sus booking records, elegimos un ID existente y llamamos al endpoint de show. Verificamos que todos los atributos (IDs, fechas, estado) coinciden con la base de datos.

Probamos acceder al show de un booking record con rol anónimo, con manager y luego autenticados como customer B intentando ver un record de customer A. En cada caso, la aplicación rechazó la petición con el código de error adecuado.

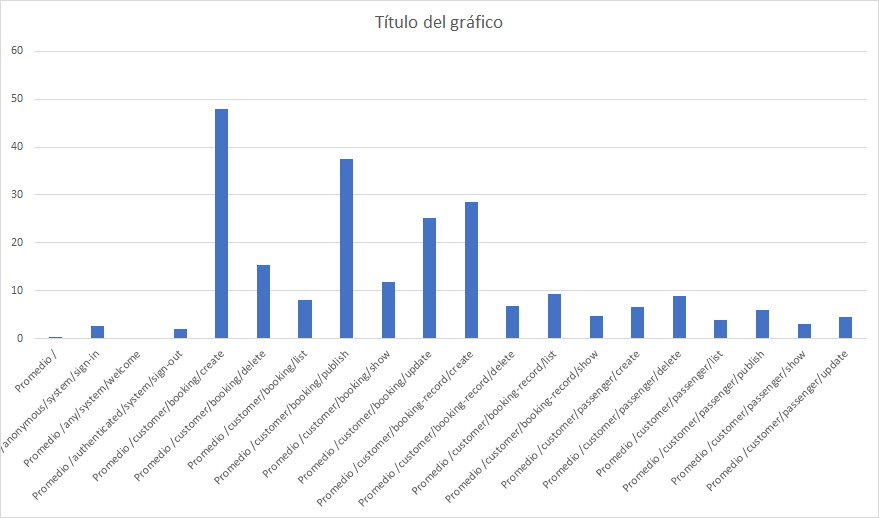
El informe de cobertura de Eclipse muestra un 100 % en controlador, servicio y repositorio para esta operación. Con ello hemos ejecutado tanto la ruta de éxito (show válido) como las comprobaciones de existencia y autorización, sin dejar ninguna lógica sin testar.

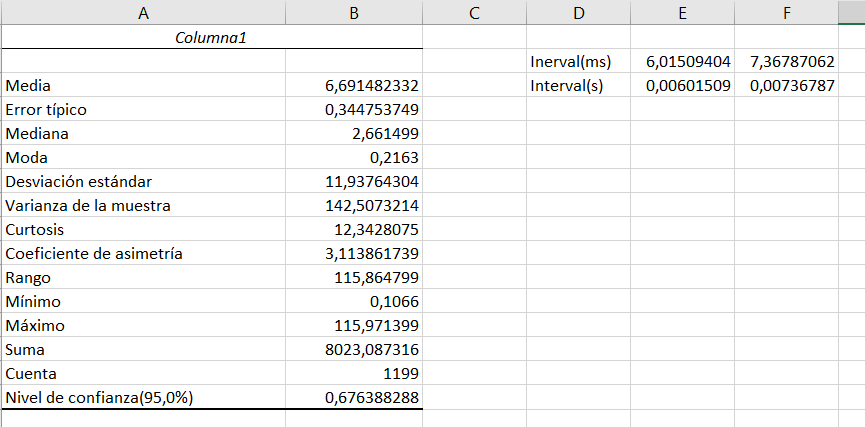
# **Testing de rendimiento**

Nos centramos ahora en el análisis del tiempo de respuesta del sistema al procesar solicitudes durante las pruebas funcionales. Se llevarán a cabo comparaciones entre nuestro portátil sin y con los índices para ver si estos han mejorado el rendimiento.

# **Sin índices**

A continuación examinamos los tiempos de respuesta del sistema al atender las peticiones de las pruebas funcionales. Para ello, registramos las métricas de latencia en dos escenarios distintos: primero sin aplicar ningún índice en la base de datos y, después, con los índices habilitados en el portátil de pruebas. Con estos datos, comparamos directamente los resultados para determinar si la introducción de índices ha logrado agilizar la ejecución de las consultas.





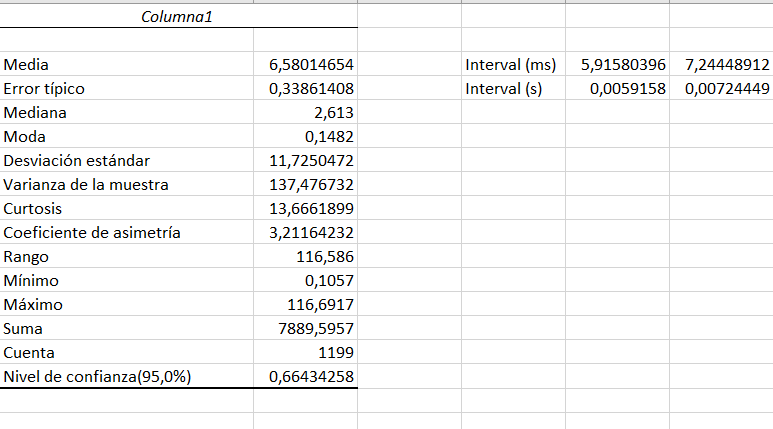
# **Con índices**

Estos son los tiempos obtenidos para las distintas funcionalidades además de un gráfico para resaltar cuales son las funcionalidades que tardan más en ejecutarse.



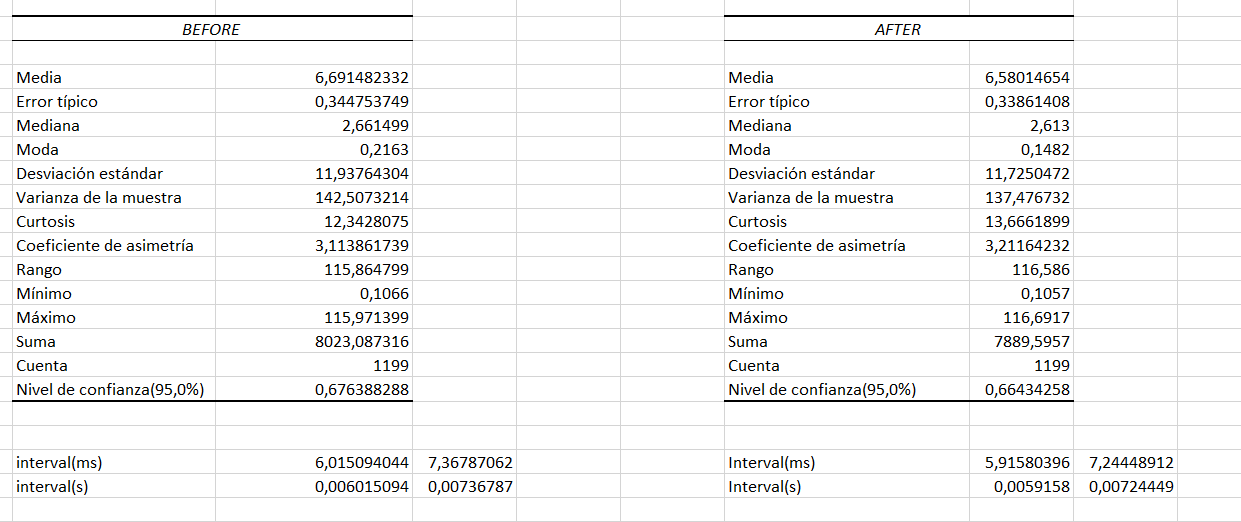
Como podemos observar el promedio general ha sido prácticamente igual (incluso un poco mayor) que sin utilizar los índices lo cual ya puede darnos un indicio de que no han mejorado las prestaciones.

Y hemos obtenido de los datos los siguientes resultados después del análisis estadístico:

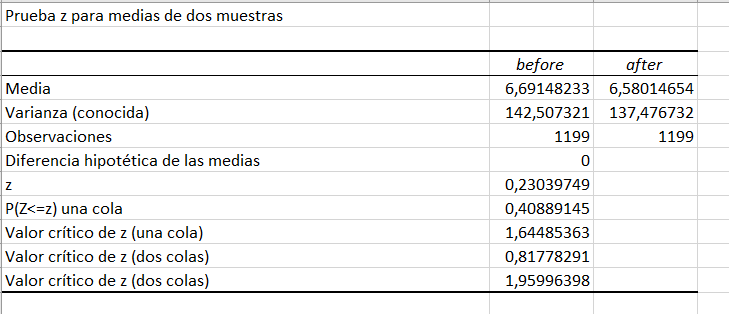


**Comparación entre utilizar y no utilizar índices**

Nos disponemos a comparar los siguientes resultados:



Tras compararlos hemos obtenido lo siguiente resultados:



Como se observa en *Valor crítico de z (dos colas)***,** el valor está en el intervalo (Alpha, 1], lo que indica que las diferencias en los tiempos medios entre las dos ejecuciones no son significativas para el análisis del rendimiento así que podemos concluir el análisis.

# **Conclusiones**

A modo de cierre, podemos destacar que las pruebas funcionales cubrieron de manera exhaustiva cada operación del sistema, incluyendo escenarios positivos, negativos y de hacking, y lograron una cobertura de código siempre muy cercana al 100%,lo que confirma la solidez de la implementación frente a errores y accesos no autorizados

En cuanto al rendimiento, la comparación de los tiempos de respuesta con y sin índices, registrada en dos equipos distintos y respaldada por intervalos de confianza al 95 %, no evidenció mejoras significativas tras la creación de índices. Este hallazgo sugiere que, bajo las condiciones ensayadas, la optimización mediante índices no aporta un beneficio apreciable al rendimiento de las consultas

En conjunto, estos resultados avalan la fiabilidad funcional y la consistencia de rendimiento del proyecto, proporcionando un punto de partida sólido para futuras iteraciones y posibles mejoras en la capa de datos.

# **Bibliografía**

Intencionalmente en blanco.