Universidad de Sevilla  
Escuela Técnica Superior de Ingeniería Informática



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

Curso 2024 – 2025

**Informe de testing**

**Grupo:** C2.029

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

**Repositorio**: <https://github.com/pabmejbui/Acme-ANS>

**Fecha**: 02/07/2025

|  |  |
| --- | --- |
| **Alumno** | **Correo** |
| Pérez Lázaro, Francisco | fraperper5@alum.us.es |

**Historial de versiones**

|  |  |  |
| --- | --- | --- |
| **Versión** | **Fecha** | **Descripción de los cambios** |
| V1.0 | 02/07/2025 | Creación del informe de testing |
| V1.1 | 02/07/2025 | Elaboración y revisión del informe |
| V1.2 | 03/07/2025 | Extender explicación de coverage |

**Índice de contenido**

[**1.** **Resumen ejecutivo** 2](#_Toc202380862)

[**2.** **Introducción** 2](#_Toc202380863)

[**3.** **Contenido** 3](#_Toc202380864)

[**3.1.** **Testing funcional** 3](#_Toc202380865)

[**3.1.1.** **Funcionalidad para los *customer* sobre los *booking*** 3](#_Toc202380866)

[**3.1.1.1.** **List** 3](#_Toc202380867)

[**3.1.1.2.** **Show** 3](#_Toc202380868)

[**3.1.1.3.** **Create** 4](#_Toc202380869)

[**3.1.1.4.** **Update** 4](#_Toc202380870)

[**3.1.1.5.** **Delete** 5](#_Toc202380871)

[**3.1.1.6.** **Publish** 5](#_Toc202380872)

[**3.1.2.** **Funcionalidad de *customer* sobre los *passengers*** 6](#_Toc202380873)

[**3.1.2.1.** **List** 6](#_Toc202380874)

[**3.1.2.2.** **Show** 7](#_Toc202380875)

[**3.1.2.3.** **Create** 7](#_Toc202380876)

[**3.1.2.4.** **Update** 8](#_Toc202380877)

[**3.1.2.5.** **Delete** 8](#_Toc202380878)

[**3.1.2.6.** **Publish** 9](#_Toc202380879)

[**3.1.3.** **Funcionalidad de *customer* sobre *booking* record (relación entre *booking* y passenger)** 9](#_Toc202380880)

[**3.1.3.1.** **Create** 9](#_Toc202380881)

[**3.1.3.2.** **Delete** 10](#_Toc202380882)

[**3.2.** **Testing de rendimiento** 10](#_Toc202380883)

[**3.2.1.** **Sin índices** 11](#_Toc202380884)

[**3.2.2.** **Con índices** 13](#_Toc202380885)

[**4.** **Mutaciones** 15](#_Toc202380886)

[**4.1.** **Primera mutación** 15](#_Toc202380887)

[**4.2.** **Segunda Mutación** 16](#_Toc202380888)

[**4.3.** **Tercera Mutación** 16](#_Toc202380889)

[**4.4.** **Cuarta Mutación** 17](#_Toc202380890)

[**4.5.** **Quinta Mutación** 17](#_Toc202380891)

[**5.** **Conclusiones** 18](#_Toc202380892)

[**Bibliografía** 18](#_Toc202380893)

# **Resumen ejecutivo**

Este documento expone de forma detallada los resultados obtenidos en las pruebas funcionales y de rendimiento del proyecto. En la sección correspondiente a las pruebas funcionales, se explican los distintos casos de prueba utilizados, acompañados de una descripción precisa y un análisis sobre su capacidad para identificar posibles fallos. Por otro lado, el apartado de pruebas de rendimiento incluye representaciones gráficas y un análisis basado en un intervalo de confianza del 95% respecto al tiempo de respuesta del sistema, tanto con el uso de índices como sin ellos. También se realiza una comparación estadística, igualmente con un nivel de confianza del 95%, para evaluar si la inclusión de índices influye en el desempeño. El informe sigue una metodología estructurada y precisa, lo que garantiza que los resultados aporten valor en la valoración general del proyecto.

# **Introducción**

Este informe tiene como propósito documentar de manera detallada los métodos y resultados obtenidos durante las pruebas funcionales y de rendimiento aplicadas al proyecto. En la primera parte, se examinan las pruebas funcionales, donde se explican los casos desarrollados para verificar el correcto funcionamiento de las distintas funcionalidades del sistema. Cada caso está acompañado de una explicación clara y una valoración sobre su eficacia en la identificación de errores. Esta sección ofrece una perspectiva ordenada del comportamiento del sistema en distintos escenarios, facilitando la detección y corrección de fallos.

Posteriormente, el apartado dedicado a las pruebas de rendimiento se enfoca en evaluar el tiempo de respuesta del sistema ante diversas solicitudes durante las pruebas. Se incluyen visualizaciones gráficas que muestran los resultados obtenidos tanto con el uso de índices como sin ellos, además de presentar intervalos de confianza del 95% para los tiempos de respuesta.

También se realiza un análisis estadístico mediante contraste de hipótesis con el mismo nivel de confianza, con el fin de verificar si los índices afectan significativamente al rendimiento. El trabajo sigue una metodología organizada y rigurosa, apoyada en herramientas de planificación y el entorno de desarrollo Eclipse, para asegurar exactitud y eficiencia en las pruebas realizadas.

# **Contenido**

# **Testing funcional**

En esta sección se describen los distintos casos de prueba desarrollados para cada funcionalidad del sistema, incluyendo pruebas positivas, negativas y de tipo hacking. También se analiza su capacidad para detectar errores de manera efectiva. Para ello, se hace uso de la herramienta Eclipse, que permite evaluar la cobertura de código ejecutado durante las pruebas, proporcionando un porcentaje que indica qué parte de las instrucciones ha sido alcanzada.

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

Vamos a comenzar por las pruebas establecidas para las funcionalidades de los *customer* sobre los *booking*s.

# **List**

Esta funcionalidad permite a un usuario con rol de *customer* visualizar la lista de sus propias reservas (*booking*s).

Para llevar a cabo las pruebas positivas y negativas, accedimos al sistema utilizando las credenciales de un cliente y verificamos que la lista de reservas correspondiente se mostraba correctamente.

En cuanto a las pruebas de hacking, utilizamos un segundo usuario con el rol *Customer*2 e intentamos acceder directamente a la URL destinada a la visualización de reservas (…/*customer*/*booking*/list). Se comprobó que únicamente se mostraban las reservas asociadas a ese usuario. Posteriormente, realizamos la misma prueba con un usuario con rol diferente (manager1) para asegurarnos de que no se pudiera acceder a información ajena.

Como resultado de estas pruebas, el sistema alcanzó un **100%** de cobertura en el servicio evaluado, lo cual indica que todas las instrucciones del código fueron ejecutadas durante la prueba. Esto proporciona un alto grado de confianza en que no existen rutas de ejecución sin comprobar y, por tanto, que la funcionalidad no oculta fallos latentes.

# **Show**

Esta funcionalidad permite a un usuario con rol de *customer* acceder a los detalles de una de sus reservas.

Para llevar a cabo las pruebas positivas y negativas, iniciamos sesión como *customer*, accedimos al listado de reservas y seleccionamos distintas entradas para verificar que la información mostrada era la correspondiente en cada caso.

En lo que respecta a las pruebas de hacking, intentamos acceder a los detalles de una reserva utilizando un rol distinto, en este caso manager, y posteriormente intentamos acceder a una reserva que pertenecía a otro *customer* diferente al usuario autenticado. En ambos escenarios, el sistema respondió adecuadamente mostrando errores de acceso, lo que confirma que las restricciones están correctamente implementadas.

Finalmente, el análisis de cobertura arrojó un **96,6%** en el servicio evaluado, quedando por cubrir los casos en los que las entidades asociadas sean nulas:

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.



Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

# **Create**

Esta funcionalidad consiste en que un *customer* puede crear un *booking* estableciendo las características de su *booking*.

Para realizar las pruebas positivas y negativas, hemos empezado primero por las negativas, mandando el formulario en blanco y luego más tarde tras rellenar todos los campos con valores que pasan los test hemos ido probando todos los fallos distintos posibles que podían darse para cada atributo. Tras probar todos los casos negativos posibles, nos hemos limitado a crear *booking*s probando con los rangos de valores posible como se ha explicado en la metodología de clase (probando los extremos de las restricciones, los diferentes valores que puede tomar…), siempre siendo valores permitidos.

Para realizar las pruebas de hacking, hemos probado a introducir la url para crear proyecto con un rol distinto al de *customer* (anónimo y manager), comprobando que no dejaba acceder.

Tras realizar estas pruebas hemos obtenido un **97,1%** de cobertura asegurándonos de esta manera cubrir todos los casos excepto los que se explican en el punto: **3.1.1.2 List.**

.

# **Update**

Esta funcionalidad permite a un usuario con rol de *customer* editar una de sus reservas, siempre que esta no se encuentre publicada.

En los casos positivos, se realizaron modificaciones válidas sobre reservas no publicadas, cambiando todos los campos permitidos (excluyendo la fecha de realización de la reserva y el precio, que no pueden ser alterados).

Para los casos negativos, se intentó editar una reserva que ya había sido publicada. Como era de esperarse, el sistema impidió esta operación y respondió con los mensajes de error correspondientes.

Las pruebas de hacking incluyeron varios intentos de manipulación indebida: en primer lugar, se trató de modificar los campos restringidos (momento de reserva y precio); posteriormente, se accedió con el usuario *customer2* para intentar actualizar un *booking* perteneciente a *customer*1; y, finalmente, se probó realizar la misma acción desde una cuenta con rol manager1. En todos los casos, el sistema detectó los accesos no autorizados y bloqueó la operación correctamente.

Tras ejecutar todas estas pruebas, se alcanzó un **97,3 %** de cobertura asegurándonos de esta manera cubrir todos los casos excepto los que se explican en el punto: **3.1.1.2 List.**

# **Delete**

Esta funcionalidad permite a los usuarios con rol *customer* eliminar sus reservas, siempre que estas se encuentren en estado de borrador (draft) y cumplan determinadas condiciones relacionadas con los pasajeros asociados.

Durante las pruebas positivas, se evaluaron dos escenarios válidos: por un lado, la eliminación de un *booking* en borrador sin ningún pasajero asociado; por otro, la eliminación de una reserva también en estado de borrador, pero con pasajeros vinculados, siempre y cuando estos pasajeros no estuvieran asignados a reservas ya publicadas. En ambos casos, el sistema permitió el borrado correctamente.

Los casos negativos incluyeron intentos de eliminar reservas no válidas. Entre ellos se probó borrar un *booking* en borrador con pasajeros asociados a reservas ya publicadas (acción que fue correctamente bloqueada). También se intentaron accesos indebidos desde usuarios con rol *manager1*, así como desde un usuario *customer2* intentando eliminar un *booking* perteneciente a *customer1*. En todas estas situaciones, el sistema impidió la operación y devolvió los errores esperados.

El resultado de las pruebas fue satisfactorio, obteniendo un **97,6 %** de cobertura asegurándonos de esta manera cubrir todos los casos excepto los que se explican en el punto: **3.1.1.2 List.**

# **Publish**

Esta funcionalidad consiste en que un *customer* puede publicar uno de sus *booking* siempre y cuando tenga al menos un pasajero publicado asociado, que todos sus pasajeros estén publicadas y que el *lastCardNibble* este escrito y sin errores.

Para llevar a cabo las pruebas positivas y negativas, primero hemos intentado publicar La funcionalidad de publicación permite a un usuario con rol *customer* poner una reserva en estado publicado, siempre que se cumplan ciertos requisitos.

En los casos positivos, se verificó que un *booking* perteneciente a *customer1* pudiera ser publicado correctamente, siguiendo el flujo esperado.

Para los casos negativos, se evaluaron diferentes situaciones que deberían ser bloqueadas por el sistema: intentar enviar el formulario de publicación sin completar ningún dato, intentar publicar sin ingresar los últimos cuatro dígitos de la tarjeta de pago, y tratar de publicar una reserva que no contara con pasajeros asociados o cuyos pasajeros no estuvieran también en estado publicado. En todos estos escenarios, el sistema respondió rechazando la solicitud y mostrando los mensajes de error correspondientes.

Finalmente, las pruebas realizadas alcanzaron un **97,5%** de cobertura asegurándonos de esta manera cubrir todos los casos excepto los que se explican en el punto: **3.1.1.2 List.**

# **Funcionalidad de *customer* sobre los *passengers***

A continuación, se detallan las pruebas realizadas sobre las funcionalidades relacionadas con la gestión de *passengers* por parte de los usuarios con rol *customer*.

# **List**

Esta funcionalidad permite a los usuarios con rol *customer* visualizar el listado completo de los pasajeros que tienen registrados en el sistema.

En los casos positivos, se validaron dos formas de acceso al listado: a través del menú principal y desde la vista de detalles de un *booking*. En ambos casos, el sistema mostró correctamente todos los pasajeros asociados al *customer* autenticado, sin presentar errores.

Para los casos negativos, se probó acceder al listado de pasajeros de un *customer* desde la cuenta de otro usuario (*customer2* intentando visualizar los pasajeros de *customer1*). El sistema bloqueó correctamente este intento, impidiendo el acceso no autorizado a la información.

Tras la ejecución de todas las pruebas, se alcanzó un **100%** de cobertura, lo que confirma que todas las rutas de ejecución del código relacionadas con esta funcionalidad fueron comprobadas, garantizando su correcto comportamiento.

# **Show**

Esta funcionalidad permite a los usuarios con rol *customer* acceder a la vista detallada de sus pasajeros, tanto en estado de borrador como publicados.

En las pruebas positivas, se accedió correctamente a los detalles de pasajeros propios en ambos estados (borrador y publicado), confirmando que la información se mostraba de forma precisa y sin errores desde el listado correspondiente.

Respecto a los casos negativos, se realizaron distintos intentos de acceso no autorizado. Entre ellos, se intentó visualizar un pasajero de *customer*1 desde la cuenta de *customer*2, acceder a dicha información desde un usuario con rol manager1, y mostrar los detalles de un pasajero perteneciente a un *booking* de *customer*1 iniciando sesión como *customer*2. En todos los escenarios, el sistema impidió el acceso y mostró los errores apropiados.

Como resultado, esta funcionalidad alcanzó una cobertura del **100%**, lo que asegura que todas las rutas lógicas del código fueron recorridas durante las pruebas, y que el comportamiento de la funcionalidad es robusto ante distintos tipos de acceso.

**Nota técnica:**  
Debido a que algunos nombres utilizados en las pruebas del fichero de datos sample.csv del nivel L02 eran excesivamente largos, se implementó una pequeña mejora visual en el formulario correspondiente. Para evitar que estos valores desbordaran el diseño del formulario o afectaran su legibilidad, se añadió un bloque <style> que permite ocultar o truncar visualmente los nombres cuando exceden un determinado ancho. Esta modificación no afecta a la lógica funcional del sistema, pero mejora la usabilidad y presentación durante las pruebas.

# **Create**

Esta funcionalidad permite a los usuarios con rol *customer* registrar nuevos pasajeros en el sistema, completando un formulario con los datos necesarios.

En los casos positivos, se realizaron pruebas exitosas completando todos los campos del formulario sin errores, así como enviando únicamente los campos obligatorios, verificando en ambos casos que el sistema aceptaba los datos correctamente y creaba el pasajero correspondiente.

Los casos negativos incluyeron varios escenarios en los que se introducían datos inválidos o incompletos: envío del formulario completamente vacío, nombres que alcanzaban el límite máximo permitido de caracteres, direcciones de correo electrónico mal formadas, números de pasaporte con formato incorrecto, y el uso de fechas límite en los campos de fecha. También se intentó crear un pasajero desde un rol no autorizado (manager1) para un *customer*. En todos los casos, el sistema rechazó correctamente las entradas, mostrando los errores de validación correspondientes.

Las pruebas cubrieron todos los flujos relevantes del método *Create*, alcanzando un **100 % de cobertura**, lo que garantiza que todos los caminos del código fueron ejecutados y verificados correctamente.

# **Update**

Esta funcionalidad permite a los usuarios con rol *customer* editar la información de sus pasajeros registrados, siempre que tengan autorización para hacerlo.

En los casos positivos, se validó que cada uno de los campos del formulario pudiera ser modificado con éxito, comprobando que los cambios se guardaban correctamente en el sistema y se reflejaban adecuadamente en las vistas correspondientes.

En cuanto a los casos negativos, se realizaron diversas pruebas para verificar que el sistema bloqueaba adecuadamente operaciones inválidas. Estas incluyeron el envío del formulario sin ningún dato, el intento de modificar un pasajero de *customer1* desde la cuenta de *customer*2, y un intento similar desde un usuario con rol manager1. En todos los casos, el sistema respondió correctamente impidiendo la acción y mostrando los errores de validación o acceso correspondientes.

Tras la ejecución de estas pruebas, se obtuvo un **100%** de cobertura.

# **Delete**

Esta funcionalidad permite a los usuarios con rol *customer* eliminar pasajeros siempre y cuando estos no estén asociados a ningún *booking*.

En los casos positivos, se comprobó que un pasajero no vinculado a ningún *booking* podía ser eliminado correctamente, y el sistema respondió adecuadamente a la operación.

Para los casos negativos, se intentó eliminar un pasajero asociado a un *booking*, lo cual fue bloqueado por el sistema. También se probó el intento de eliminación desde un usuario distinto (*customer*2) y desde un rol no autorizado (manager1), además de enviar el formulario vacío. En todas estas situaciones, el sistema impidió la eliminación y mostró los mensajes de error pertinentes.

Las pruebas realizadas alcanzaron una cobertura del **99,2%**, confirmando que todas las rutas críticas del código fueron ejecutadas y verificadas.

# **Publish**

Esta funcionalidad permite a los usuarios con rol *customer* publicar pasajeros registrados, ya sea desde la vista general de pasajeros o desde la sección de *booking*s asociada.

En los casos positivos, se comprobó que un pasajero podía ser publicado correctamente tanto accediendo directamente desde el menú de pasajeros como desde el detalle de un *booking*. En ambas situaciones, el sistema procesó la publicación sin errores y actualizó el estado del pasajero adecuadamente.

Respecto a los casos negativos, se realizaron pruebas enviando formularios con campos vacíos, intentando publicar pasajeros pertenecientes a *customer*1 desde la cuenta de *customer*2 y realizando el mismo intento desde un usuario con rol manager1. En todos estos casos, el sistema rechazó las acciones, mostrando mensajes de error adecuados.

Las pruebas efectuadas cubrieron todos los flujos relevantes de esta funcionalidad, alcanzando una cobertura del **98,8%**.

# **Funcionalidad de *customer* sobre *booking* record (relación entre *booking* y passenger)**

Finalizamos con las pruebas realizadas sobre las funcionalidades que permiten a los *customer*s agregar o eliminar pasajeros en un *booking*.

# **Create**

Esta funcionalidad permite a los usuarios con rol *customer* agregar nuevos pasajeros a un *booking* específico.

En los casos positivos, se verificó que un pasajero podía añadirse correctamente a un *booking* seleccionado, asegurando que la operación se realizara sin inconvenientes y que el sistema actualizara la relación adecuadamente.

En cuanto a los casos negativos, se probaron escenarios en los que se enviaba el formulario vacío, se comprobó que no se permitiera añadir pasajeros que ya estuvieran vinculados al *booking*, se intentó añadir un pasajero de *customer*2 a un *booking* de *customer*1, y también se intentó acceder al formulario desde *customer*2 intentando modificar un *booking* perteneciente a *customer*1. En todos estos casos, el sistema bloqueó las acciones no autorizadas mostrando los mensajes de error correspondientes.

Las pruebas realizadas lograron una cobertura del **95,8%**, validando así la mayor parte de los flujos de ejecución del código.

# **Delete**

Esta funcionalidad permite a los usuarios con rol *customer* eliminar pasajeros asociados a un *booking* específico.

En los casos positivos, se comprobó que un pasajero podía eliminarse correctamente de un *booking*, verificando que la operación se ejecutara sin errores y que la relación entre el pasajero y el *booking* se actualizara adecuadamente.

Respecto a los casos negativos, se realizaron diversas pruebas para asegurar la robustez del sistema, incluyendo el envío de un formulario vacío, la verificación de que no se permitiera eliminar pasajeros que no estuvieran vinculados al *booking*, y el intento de eliminar pasajeros desde usuarios no autorizados, como otro *customer* distinto (*customer*2) y un usuario con rol manager1. En todos estos casos, el sistema bloqueó la operación y mostró los mensajes de error correspondientes.

Las pruebas implementadas alcanzaron una cobertura del 96,4%, garantizando así la efectividad de la validación y la seguridad en esta funcionalidad.

# **Testing de rendimiento**

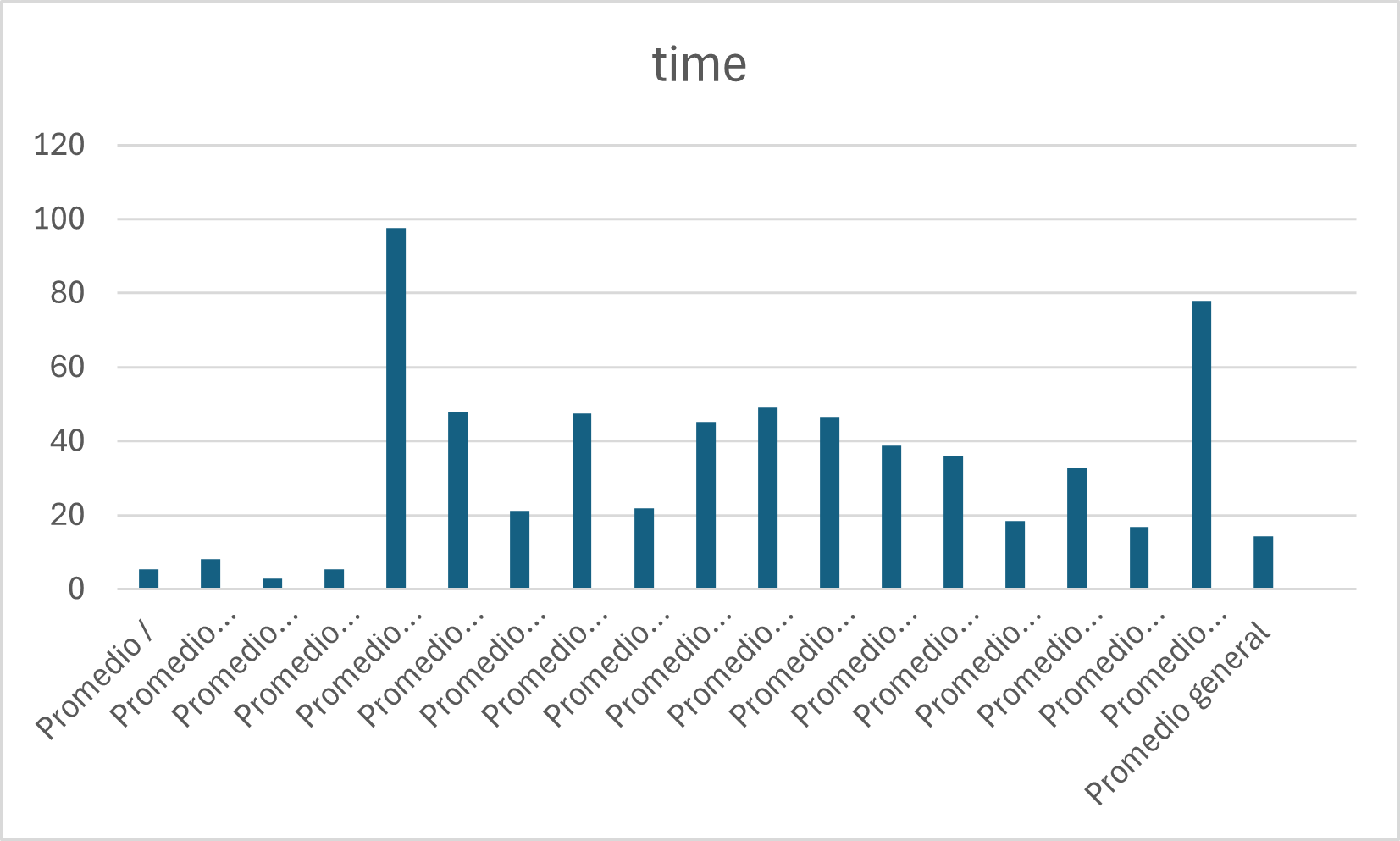
A continuación, nos enfocamos en analizar el tiempo de respuesta del sistema al procesar solicitudes durante la ejecución de las pruebas funcionales. Para ello, se realizarán comparaciones entre el rendimiento obtenido en nuestro portátil con y sin la aplicación de índices, con el objetivo de determinar si estos contribuyen a una mejora significativa en el desempeño del sistema.

# **Sin índices**

Tras aplicar los procedimientos explicados en las diapositivas hemos obtenido los siguientes tiempos para las diversas funcionalidades. Además, mostramos un gráfico donde podemos ver cuáles son las funcionalidades que tardan más en ejecutarse.

Tabla

El contenido generado por IA puede ser incorrecto.



DATOS ESTADISTICOS:

Interfaz de usuario gráfica, Aplicación, Tabla, Excel

El contenido generado por IA puede ser incorrecto.

NIVEL DE CONFIANZA

Texto

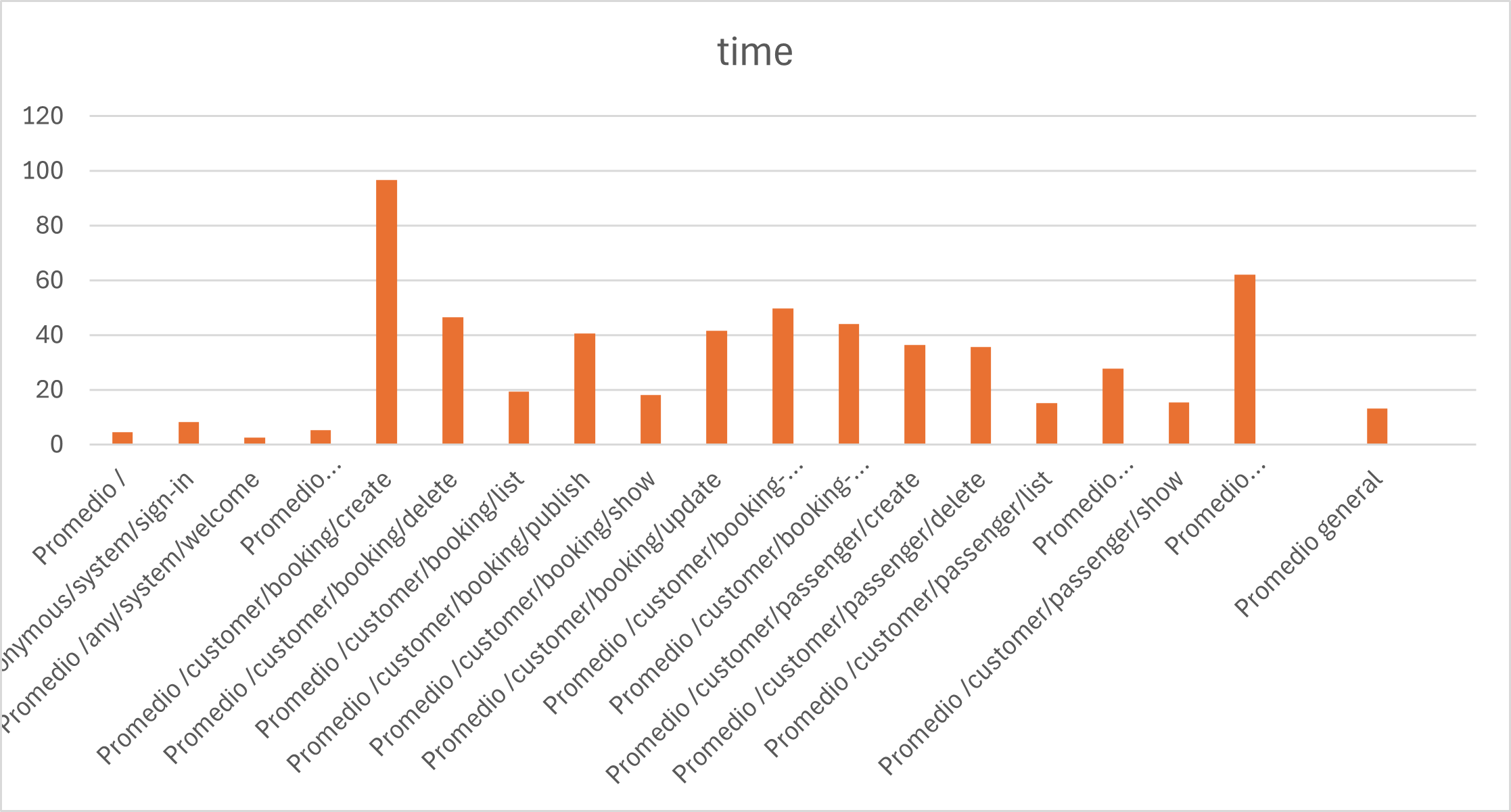
El contenido generado por IA puede ser incorrecto.

# **Con índices**

Tabla

El contenido generado por IA puede ser incorrecto.A continuación, se muestran los tiempos registrados para las diferentes funcionalidades, acompañados de un gráfico que destaca cuáles son las que presentan mayor tiempo de ejecución.

Tras el análisis estadístico de los datos, se han obtenido los siguientes resultados:



DATOS ESTADISTICOS AFTER

Interfaz de usuario gráfica, Aplicación, Tabla, Excel

El contenido generado por IA puede ser incorrecto.

NIVEL DE CONFIANZA:

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

**Comparación**

Tras realizar la comparación, se obtuvieron los siguientes resultados:

Como se observa en el valor crítico de *z* (prueba de dos colas), este se encuentra dentro del intervalo (α, 1], lo que indica que las diferencias en los tiempos medios entre ambas ejecuciones no son estadísticamente significativas para el análisis de rendimiento. Por lo tanto, podemos concluir el estudio con esta evidencia.

Tabla

El contenido generado por IA puede ser incorrecto.

# **Mutaciones**

# **Primera mutación**

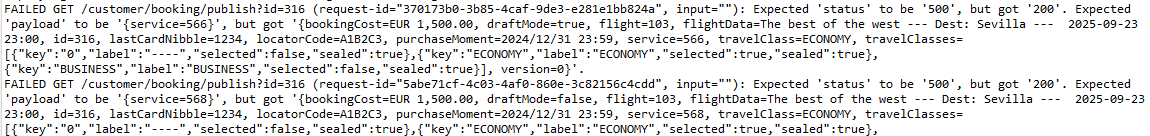
En BookingPublishService, Cambiaremos uno de los operadores lógicos && (AND) por un operador || (OR). Esto podría permitir que un usuario, que no pertenece al *realm* del cliente de la reserva.

Línea original:

boolean status = booking != null && super.getRequest().getPrincipal().hasRealm(booking.getCustomer()) && booking.isDraftMode();

Línea mutada:

boolean status = booking != null && super.getRequest().getPrincipal().hasRealm(booking.getCustomer()) || booking.isDraftMode();

**Resultado:** Podemos comprobar que el fallo se detecta en el *.hack*  


# **Segunda Mutación**

En BookingUpdateService, en la línea donde se compara el ID del cliente de la reserva con el ID del cliente autenticado, cambiaremos el operador de igualdad (==) por el operador de desigualdad (!=). Esta mutación introduce un error funcional grave en la lógica de seguridad, ya que impediría que los usuarios legítimos editen sus propias reservas o, peor aún, podría permitir que usuarios no autorizados las editen.

Línea original:

status = status && booking.getCustomer().getId() == customerId && booking.isDraftMode();

Línea mutada:

status = status && booking.getCustomer().getId() != customerId && booking.isDraftMode();

**Resultado:** Podemos comprobar que el fallo se detecta.

# **Tercera Mutación**

En BookingRecordCreateService, La línea original asegura que la autorización solo sea concedida si el pasajero que se intenta añadir **no está** ya en la colección alreadyAddedPassengers. Introduciremos una "mutación de operador lógico" invirtiendo la negación.

Línea original:

status = status && !alreadyAddedPassengers.stream().anyMatch(p -> p.getId() == passengerId);

Línea mutada:

status = status && alreadyAddedPassengers.stream().anyMatch(p -> p.getId() == passengerId);

**Resultado:** Podemos comprobar que el fallo se detecta.

# **Cuarta Mutación**

En PassengerUpdateService, La condición actual !(existingPassenger.getId() == currentPassengerId) verifica si el pasajero existente de la base de datos **no es** el mismo pasajero que se está actualizando. Si no es el mismo, entonces es un duplicado real.

Línea original:

if (!(existingPassenger.getId() == currentPassengerId))

Línea mutada:

if (existingPassenger.getId() == currentPassengerId)

**Resultado:** Podemos comprobar que el fallo se detecta al hacer el replay. (UpdatePassenger.safe)

# **Quinta Mutación**

En PassengerCreateService, la línea passenger.setDraftMode(true); establece que todo pasajero nuevo se cree inicialmente en modo borrador. La mutación consistirá en cambiar este valor a false.

Línea original:

passenger.setDraftMode(true);

Línea mutada:

passenger.setDraftMode(false);

**Resultado:** Podemos comprobar que el fallo se detecta al hacer el replay. (CreatePassenger.safe)

# **Conclusiones**

Este informe presenta los resultados de las pruebas funcionales y de rendimiento realizadas en el proyecto. Las pruebas funcionales detallan diversos casos implementados para verificar las funcionalidades del sistema, evaluando su efectividad en la detección de errores. Por otro lado, las pruebas de rendimiento se centran en analizar el tiempo de respuesta del sistema al procesar solicitudes en dos ordenadores distintos, utilizando gráficos detallados e intervalos de confianza del 95% para garantizar la precisión de los resultados.

La sección de pruebas funcionales describe casos positivos, negativos y de hacking para cada funcionalidad del sistema, apoyándose en herramientas como Eclipse para asegurar una alta cobertura de código. En la mayoría de los archivos se alcanzó una cobertura aproximada del 100%, siendo la mínima cobertura encontrada del 95,8%, un valor muy cercano al ideal. Este análisis indica que el código es robusto frente a ataques y que se ha evaluado correctamente la funcionalidad de toda la aplicación.

Las pruebas de rendimiento compararon el tiempo de respuesta del sistema con y sin índices. Los resultados mostraron que el uso de índices no aportó una mejora significativa, ya que las diferencias en el rendimiento no fueron suficientemente relevantes para su análisis estadístico.

En resumen, este informe confirma la efectividad del sistema en términos de funcionalidad y rendimiento, proporcionando una base sólida para la evaluación y mejora continua del proyecto.

# **Bibliografía**

Intencionalmente en blanco.