



INFORME DE TESTING INDIVIDUAL

Grupo 1-C1.020 | <https://github.com/PDJ6975/Acme-ANS-D04-25.5.0>



Nombre	Correo Corporativo
Antonio Rodríguez Calderón	antrodcal@alum.us.es
Adrián Ramírez Gil	adrramgil@alum.us.es
Jianwu Hu	jiahu3@alum.us.es
Pablo Castrillón Mora	pabcasmor1@alum.us.es
Pablo Olivencia Moreno	pabolimor@alum.us.es

24 DE MAYO DE 2025
PABLO OLIVENCIAS | STUDENT #2

Tabla de Contenido

1. Testing Funcional	2
1.1 Introducción.....	2
1.2 Metodología	2
1.3 Casos de prueba por característica	3
1.3.1 Booking.....	3
1.3.2 Passenger	8
1.4 Calidad de la cobertura	12
1.5 Conclusiones finales	13
2. Testing de Rendimiento	13
2.1 Entorno y protocolo.....	13
2.2 Recogida de datos	13
2.3 Estadísticas Descriptivas	15
2.4 Gráficos de eficiencia por característica	17
2.5 Hipótesis y conclusión	19
4. Testing de Mutaciones	19
4.1 Primera mutación: Inversión de la lógica de autorización en el BookingUpdateService ...	19
4.2 Segunda mutación: Asociación incorrecta del BookingRecord en el PassengerCreateService.....	20
4.3 Tercera mutación: Romper lógica para el purchaseMoment al crear un booking	21
4.4 Cuarta mutación: Eliminar dateOfBirth del binding al crear un pasajero.....	22
4.5 Quinta mutación: Eliminar del unbind para el listado de pasajeros el fullName	22
5. Tabla de Revisión	22

1. Testing Funcional

1.1 Introducción

En esta sección del documento se pretende describir un listado con los casos de prueba realizados, agrupados por características, para las entidades **Booking** y **Passenger**.

Para cada caso de prueba, se va a proporcionar una descripción y cuál es su eficacia para detectar errores.

1.2 Metodología

Para probar las entidades obligatorias del estudiante dos, se han aplicado pruebas “.safe” y “.hack”. Para las primeras se ha probado que las entidades se listen, muestren y permitan un CRUD válido y funcional, permitiendo todo tipo de valores válidos y rechazando aquellos que no cumplen con la validación.

Para las segundas, se han realizado pruebas de tipo GET/POST hacking para comprobar el correcto funcionamiento del sistema frente a acciones ilegales.

Por último, se muestra una tabla con la cobertura total alcanzada. En lo que respecta a los paquetes globales:

Paquete	Cobertura alcanzada
customer.booking	95,8%
customer.passenger	99,2%

Por otro lado, en lo que respecta a cada servicio específico de FlightAssignment:

Servicio	Cobertura alcanzada
BookingListService	100%
BookingDeleteService	93,1%
BookingUpdateService	95%
BookingPublishService	95,3%
BookingShowService	94,8%
BookingCreateService	98,2%

Para terminar, para los servicios específicos de ActivityLog:

Servicio	Cobertura alcanzada
PassengerListService	100%
PassengerDeleteService	100%
PassengerUpdateService	99,2%
PassengerPublishService	96,4%
PassengerShowService	100%

Para los controladores de ambas entidades se alcanza un 100% de la cobertura, lo que significa que se prueban todas las líneas de código contenidas en ellos (es decir, todos los servicios que ofrecen).

1.3 Casos de prueba por característica

1.3.1 Booking

Caso de prueba	Tipo	Descripción	Eficacia
list.safe	Renderizado	Se prueba que el listado se muestre correctamente	No se detectó ninguna incidencia
list.hack	Hack	Se prueba que: -Miembros con otro rol no puedan ver el listado.	No se detectó ninguna incidencia ya que se realizaba una validación correcta en el “authorised”.

Caso de prueba	Tipo	Descripción	Eficacia
show.safe	Renderizado	Se prueba que una reserva se renderice correctamente sin ningún error.	No ha detectado ninguna incidencia
show.hack	Hack	<p>Permite probar:</p> <ul style="list-style-type: none"> -Miembros con otro rol no puedan ver los detalles de una reserva. -Miembros con mismo rol no puedan ver reservas que no le corresponden. 	No se detectó ninguna incidencia

Caso de prueba	Tipo	Descripción	Eficacia
create.safe	Casos positivos y negativos del formulario	Se prueba que todos los campos del formulario estén correctamente validados, rechazando los valores no admitidos por el modelo.	No detectó ninguna incidencia
create.hack	Hack (url y valores)	<p>Permite probar:</p> <ul style="list-style-type: none"> -Miembros con otro rol no puedan acceder al formulario de creación 	No detectó ninguna incidencia

		<ul style="list-style-type: none"> - No se puedan seleccionar vuelos no existentes o no disponibles -Atributos read-only no puedan ser modificados. - Atributos de selección no puedan ser modificados con valores ilegales . 	
--	--	--	--

Caso de prueba	Tipo	Descripción	Eficacia
update.safe	Casos positivos y negativos del formulario	Se prueba que todos los campos del formulario estén correctamente validados, rechazando los valores no admitidos por el modelo.	No detectó ninguna incidencia
update-url.hack	Hack (url)	<p>Permite probar:</p> <ul style="list-style-type: none"> -Miembros con otro rol no puedan acceder a los detalles de la reserva por medio del “update”. - Miembros del mismo rol no puedan acceder a los detalles de una reserva que no les pertenece - Reservas ya publicadas no puedan ser actualizadas. 	No detectó ninguna incidencia
update.hack	Hack (values)	Permite probar que:	No detectó ninguna incidencia porque todo estaba bien

		<ul style="list-style-type: none"> -Atributos read-only no puedan ser modificados. - Atributos de selección no puedan ser modificados con valores ilegales. 	validado e implementado.
--	--	---	--------------------------

Caso de prueba	Tipo	Descripción	Eficacia
publish.safe	Casos positivos y negativos del formulario	Se prueba el correcto funcionamiento o el mensaje de error en caso de no ser publicable	No detectó ninguna incidencia ya que todos los campos estaban correctamente validados.
publish	Hack	<p>Permite probar:</p> <ul style="list-style-type: none"> -Miembros con otro rol no puedan acceder a los detalles de la reserva por medio del “publish”. - Miembros del mismo rol no puedan acceder a los detalles de una reserva que no les pertenece por medio del “publish”. - Reserva ya publicadas no puedan volver a publicarse. 	No detectó ninguna incidencia porque todo estaba bien validado e implementado.

Caso de prueba	Tipo	Descripción	Eficacia
delete.safe	Casos positivos y negativos del formulario	<p>-Se prueba que una reserva se borre correctamente, incluso al meter valores diferentes de los mostrados, bien positivos o negativos.</p> <p>- Reservas con pasajeros asociados se borren correctamente.</p>	No detectó ninguna incidencia ya que todos los campos estaban correctamente validados.
delete.hack	Hack	<p>Permite probar:</p> <p>-Miembros con otro rol no puedan acceder a los detalles de la reserva por medio del “delete”.</p> <p>- Miembros del mismo rol no puedan acceder a los detalles de una reserva que no les pertenece por medio del “delete”.</p> <p>- Reservas ya publicadas no puedan ser eliminadas.</p>	Permitió detectar que el formulario se rompía al introducir la url manualmente en el navegador.

1.3.2 Passenger

Caso de prueba	Tipo	Descripción	Eficacia
list.safe	Renderizado	Se prueba que el listado de pasajeros se muestre correctamente	No se detectó ninguna incidencia
list.hack	Hack	<p>Se prueba que:</p> <ul style="list-style-type: none"> -Miembros con otro rol no puedan ver un listado de pasajeros. -Miembros del mismo rol no puedan ver un listado de pasajeros de una reserva que no les pertenece. 	No se detectó ninguna incidencia ya que se realizaba una validación correcta en el “authorised”.

Caso de prueba	Tipo	Descripción	Eficacia
show.safe	Renderizado	Se prueba que un pasajero se renderice correctamente sin ningún error.	No ha detectado ninguna incidencia ya que el código del servicio estaba correctamente implementado.
show.hack	Hack	<p>Permite probar:</p> <ul style="list-style-type: none"> -Miembros con otro rol no puedan ver los detalles de un pasajero. -Miembros con mismo rol no puedan ver incidencia de pasajeros que no le corresponden. 	No se detectó ninguna incidencia ya que se realizaba una validación correcta en el “authorised”.

Caso de prueba	Tipo	Descripción	Eficacia
create.safe	Casos positivos y negativos del formulario	Se prueba que todos los campos del formulario estén correctamente validados, rechazando los valores no admitidos por el modelo.	No detectó ninguna incidencia ya que todos los campos estaban correctamente validados.
create.hack	Hack	<p>Permite probar:</p> <ul style="list-style-type: none"> -Miembros con otro rol no puedan acceder al formulario de creación de una reserva. - Miembros del mismo rol no puedan acceder al formulario de creación de un pasajero asociado a una reserva que no les corresponde. 	No detectó ninguna incidencia ya que el servicio estaba correctamente implementado.

		<ul style="list-style-type: none"> - Miembros del rol correcto no puedan acceder al formulario de creación de un pasajero asociado a una reserva ya publicada 	
--	--	--	--

Caso de prueba	Tipo	Descripción	Eficacia
update.safe	Casos positivos y negativos del formulario	<p>Se prueba que todos los campos del formulario estén correctamente validados, rechazando los valores no admitidos por el modelo (principalmente la longitud al ser los campos String).</p>	No detectó ninguna incidencia ya que todos los campos estaban correctamente validados.
update.hack	Hack (url)	<p>Permite probar:</p> <ul style="list-style-type: none"> -Miembros con otro rol no puedan ver los detalles de un pasajero mediante el “update”. -Miembros con mismo rol no puedan ver pasajeros de reservas que no le corresponden mediante el “update”. - Miembros del rol correcto no puedan tratar de actualizar pasajeros ya publicados. 	No detectó ninguna incidencia porque todo estaba bien validado e implementado.

Caso de prueba	Tipo	Descripción	Eficacia
publish.safe	Casos positivos y negativos del formulario	Se comprueba el correcto funcionamiento de la publicación de pasajeros. No es necesario validar campos ya que no se bindean	No detectó ninguna incidencia
Publish. hack	Hack	<p>Permite probar:</p> <ul style="list-style-type: none"> -Miembros con otro rol no puedan ver los detalles de un pasajero mediante el “publish”. -Miembros con mismo rol no puedan ver pasajeros de reservas que no le corresponden mediante el “publish.” - Miembros del rol correcto no puedan tratar de Publicar pasajeros ya publicados. 	No detectó ninguna incidencia porque todo estaba bien validado e implementado.

Caso de prueba	Tipo	Descripción	Eficacia
delete.safe	Casos positivos y negativos del formulario	Se prueba que una incidencia se borre correctamente bien al incluir valores correctos o incorrectos.	No se detectó ninguna incidencia.
Delete.hack	Hack	<p>Permite probar:</p> <ul style="list-style-type: none"> -Miembros con otro rol no puedan acceder a los detalles de la incidencia por medio del “delete”. - Miembros del mismo rol no puedan acceder a los detalles de un pasajero que no les pertenece por medio del “delete”. - Pasajeros ya publicados no puedan ser borrados. 	Permitió detectar el mismo error en el unbind que en las reservas.

1.4 Calidad de la cobertura

En esta sección se analiza la calidad de la cobertura de los servicios desarrollados para las entidades Booking y Passenger.

Booking

La cobertura alcanzada para el paquete customer.booking es del **95,8%**, lo que refleja una implementación sólida de los casos de prueba. En particular, el servicio BookingListService alcanza el **100%**, garantizando la verificación completa de la funcionalidad de listado.

Para los servicios de **crear, actualizar, eliminar, mostrar y publicar**, las coberturas se sitúan entre el **93,1% y el 98,2%**, destacando especialmente BookingCreateService con un 98,2%. Estas cifras indican que se han cubierto prácticamente todos los caminos posibles en el flujo de ejecución. Sin embargo, al igual que en otros módulos, se han incorporado validaciones adicionales –por ejemplo, en los métodos authorise y validate– que, por su naturaleza defensiva, no siempre pueden ser cubiertas mediante pruebas funcionales desde la interfaz.

Estos fragmentos incluyen validaciones redundantes o de seguridad que protegen el sistema frente a usos indebidos o manipulaciones a nivel de request, lo que dificulta lograr una cobertura total sin emplear técnicas de inyección directa o pruebas unitarias específicas.

Passenger

El paquete customer.passenger alcanza una cobertura del **99,2%**, lo que demuestra un altísimo nivel de exhaustividad en los tests. Servicios como PassengerListService, PassengerDeleteService y PassengerShowService logran el **100% de cobertura**, asegurando un control total del flujo de trabajo en estos puntos clave.

En PassengerCreateService y PassengerUpdateService, con coberturas del 99,2% y 99,2% respectivamente, se presentan pequeñas porciones de código no cubiertas debido a las mismas razones mencionadas anteriormente: validaciones defensivas o rutas imposibles de alcanzar desde la vista, como nulos no esperados o parámetros manipulados de forma maliciosa.

1.5 Conclusiones finales

En definitiva, se han desarrollado un conjunto de pruebas que cubre la gran mayoría de líneas de código de las entidades del estudiante tres. Aunque haya métodos de validación que se pueden optimizar, se ha cumplido el objetivo de obtener un sistema robusto y seguro ante la gran mayoría de escenarios que se dan en el mundo real.

2. Testing de Rendimiento

2.1 Entorno y protocolo

Para el desarrollo de esta sección se va a emplear un equipo con las siguientes características:

Modelo	RAM	CPU	GPU	Disco	SO
Ordenador de escritorio	32 GB	AMD Ryzen 9 7950x3d 5.7 GHz	Nvidia RTX 5080	1 TB	Windows 11 Pro

Para este estudio se va a emplear únicamente los casos de prueba que he ejecutado individualmente como estudiante dos, empleando la versión 25.5.0 del proyecto y del framework. En el primer lanzamiento, no se emplearán índices en mis entidades, para ver si el rendimiento mejora una vez se implementen.

2.2 Recogida de datos

Para la obtención de datos, tanto antes como después de implementar índices, se han seguido los pasos explicados en la teoría, analizando la información del “.trace”

generado por la aplicación y filtrándola para obtener un fichero limpio del que poder obtener información útil como gráficos.

2.3 Estadísticas Descriptivas

Una vez filtradas las peticiones realizadas en los casos de prueba (490 filas en total), se ha procedido a obtener una tabla de estadísticas descriptivas para los tiempos de respuesta. En ambas trazas, previa y posterior a la incorporación de índices, se observa una media de alrededor de 4,5 ms y una desviación estándar cercana a los 9 ms. Los valores son:

Estadística Descriptiva	
Media	4,4962084
Error típico	0,40005923
Mediana	1,6627
Moda	1,5586
Desviación estándar	8,8375973
Varianza de la muestra	78,1031261
Curtosis	70,1062006
Coeficiente de asimetría	7,18089147
Rango	105,2467
Mínimo	0,4402
Máximo	105,6869
Suma	2194,1497
Cuenta	488
Nivel de confianza (95,0%)	0,78605522

Ilustración 1: Datos estadísticos obtenidos a partir de la primera traza no mejorada

Estadística descriptiva	
Media	4,48761902
Error típico	0,41562997
Mediana	1,6098
Moda	3,5846
Desviación estándar	9,19096869
Varianza de la muestra	84,4739055
Curtosis	102,89113
Coeficiente de asimetría	8,39918808
Rango	137,972
Mínimo	0,4458
Máximo	138,4178
Suma	2194,4457
Cuenta	489
Nivel de confianza(95,0%)	0,81664518

Ilustración 2: Datos estadísticos obtenidos a partir de la traza mejorada con índices

Se observa que los tiempos son muy bajos en ambos casos, con distribuciones ligeramente asimétricas y valores atípicos que influyen en la varianza. A pesar de ello, los intervalos de confianza se solapan completamente, lo que sugiere que no hay diferencias significativas entre ambos conjuntos de datos.

2.4 Gráficos de eficiencia por característica

Se han generado gráficos de promedio de tiempos por feature antes y después de introducir índices. En ambos casos, se observa que **/customer/booking/create** es la petición más costosa, lo cual es coherente con su lógica de negocio, ya que requiere cargar múltiples entidades asociadas.

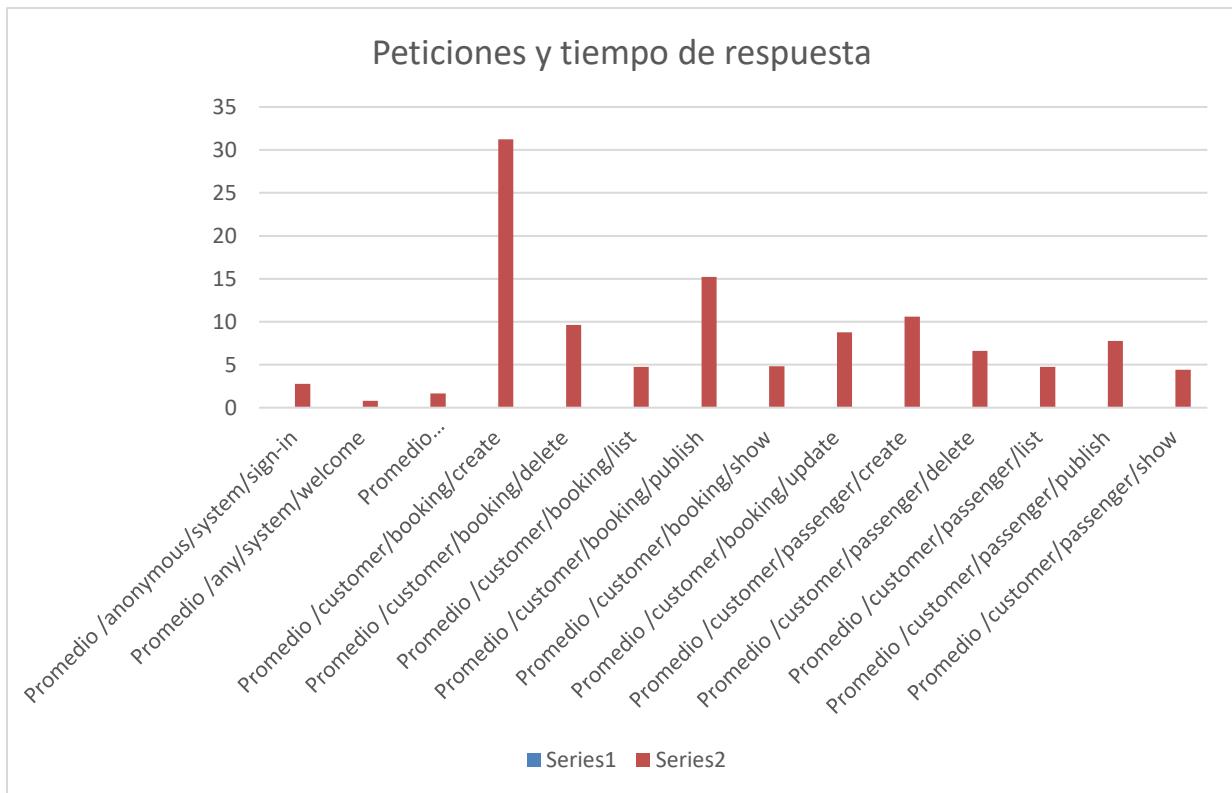


Ilustración 3: Gráfico de eficiencia de la traza sin mejorar



Ilustración 4: Gráfico de eficiencia de la traza mejorada

A pesar de que algunas features presentan pequeñas variaciones, en general no se aprecian mejoras relevantes tras la creación de índices. Esto es atribuible a que los tiempos de respuesta iniciales ya eran muy bajos, por lo que la ganancia es difícil de cuantificar.

Así, si bien no era necesario analizar el MIR porque el límite superior de los intervalos de confianza estaba muy por debajo del requisito impuesto, nos ha ayudado a perfilar un poco más las diferencias entre ambas trazas y a entender un poco mejor dónde el proyecto tiene más carga.

2.5 Hipótesis y conclusión

Para confirmar si las diferencias observadas son estadísticamente significativas, se ha realizado una prueba Z para dos muestras independientes con varianzas conocidas:

Prueba z para medias de dos muestras

	102,8225	79,7985
Media	4,29430637	4,33329344
Varianza (conocida)	78,1031261	84,4739055
Observaciones	487	488
Diferencia hipotética de las medias	0	
z	-0,06751292	
P(Z<=z) una cola	0,47308669	
Valor crítico de z (una cola)	1,64485363	
Valor crítico de z (dos colas)	0,94617338	
Valor crítico de z (dos colas)	1,95996398	

Ilustración 5: Prueba z para muestras de dos medias

El valor p es claramente mayor que α , por lo que no podemos rechazar la hipótesis nula. En consecuencia, no hay evidencia estadística para afirmar que los índices han mejorado el rendimiento de forma significativa.

4. Testing de Mutaciones

Con el objetivo de comprobar la robustez de los tests y garantizar que cubren adecuadamente los casos esperados, se han realizado un total de **cinco mutaciones** intencionadas en el código. A continuación, se describen dichas mutaciones, su impacto y el resultado observado en la ejecución de los tests:

4.1 Primera mutación: Inversión de la lógica de autorización en el BookingUpdateService

Esta mutación invierte la condición original, permitiendo actualizaciones únicamente si el draftMode está desactivado (lo contrario al comportamiento correcto).

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

Resultado: la mutación fue detectada correctamente por los tests update.safe y update.hack.

4.2 Segunda mutación: Asociación incorrecta del BookingRecord en el PassengerCreateService

En lugar de asociar el BookingRecord al Booking correspondiente, se ha establecido explícitamente como null.

Resultado: la mutación fue captada correctamente por los tests create.safe de pasajero, al detectar que el registro no puede estar desvinculado.

4.3 Tercera mutación: Romper lógica para el purchaseMoment al crear un booking

Se altera el control de disponibilidad de vuelos para que la lógica permita vuelos no disponibles (en estado draft o fuera de rango).

```
if (booking.getFlight() != null) {  
    Flight flight = booking.getFlight();  
  
    super.state(!flight.isDraftMode(), "flight",  
    "customer.booking.form.error.flight-draft");  
  
    Calendar cal = Calendar.getInstance();  
    cal.add(Calendar.MONTH, 6);  
    Date currentDate = cal.getTime();  
  
    Collection<Flight> availableFlights =  
    this.repository.findAvailableFlights(currentDate);  
  
    super.state(availableFlights.contains(flight), "flight",  
    "customer.booking.form.error.flight-not-available");  
}
```

Resultado: el error fue correctamente captado por los tests create.safe de Booking.

4.4 Cuarta mutación: Eliminar dateOfBirth del binding al crear un pasajero

@Override

```
public void bind(final Passenger passenger) {  
    super.bindObject(passenger, "fullName", "email",  
    "passportNumber", "specialNeeds");  
}
```

Se omite el campo obligatorio dateOfBirth, impidiendo que llegue del formulario al objeto.

Resultado: el test create.safe de Passenger falló como se esperaba, demostrando cobertura adecuada.

4.5 Quinta mutación: Eliminar del unbind para el listado de pasajeros el fullName

```
Dataset dataset = super.unbindObject(passenger, "email",  
    "passportNumber", "dateOfBirth", "specialNeeds");
```

Se elimina el nombre del pasajero del dataset que se envía al frontend, lo que rompe la interfaz de usuario.

Resultado: múltiples tests de listado de pasajeros (list.safe, show.safe) detectan correctamente la anomalía.

S

5. Tabla de Revisión

Versión	Fecha	Descripción de los cambios
---------	-------	----------------------------

1.0	24/05/2025	Creación inicial del documento
-----	------------	--------------------------------