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

23 DE MAYO DE 2025

ANTONIO RODRÍGUEZ | STUDENT #3

Tabla de Contenido

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 **FlightCrewMember** y **ActivityLog**.

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 tres, 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.

Por otro lado, para las segundas, generalmente se han dividido en pruebas de **“url”** y en pruebas de **valores**, principalmente por dos motivos. El primero es que solía dar problemas hacer las dos pruebas juntas, y el segundo es que permitía ser mucho más específico en cada una de las pruebas.

Del mismo modo, se ha buscado obtener la mayor variabilidad posible en los casos de prueba. Para ello, se han realizado dos pasos importantes. Por un lado, se han repetido los test “.safe” en inglés y en español, para validar que el sistema está correctamente internacionalizado (están en el proyecto para que quede constancia). Por el otro, en muchos casos se han realizado los casos de prueba con usuarios distintos para asegurar que todos los elementos se renderizan y funcionan correctamente tanto para usuarios nuevos como ya existentes.

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

Paquete	Cobertura alcanzada
crewMember.assignment	98%
crewMember.activityLog	96,8%

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

Servicio	Cobertura alcanzada
CrewMemberAssignmentListCompletedService	100%
CrewMemberAssignmentListPlannedService	100%
CrewMemberAssignmentShowService	100%
CrewMemberAssignmentCreateService	98,6%
CrewMemberAssignmentUpdateService	94,1%

CrewMemberAssignmentPublishService	97,8%
CrewMemberAssignmentDeleteService	99,4%

Para terminar, para los servicios específicos de ActivityLog:

Servicio	Cobertura alcanzada
CrewMemberActivityLogListService	99,4%
CrewMemberActivityLogShowService	100%
CrewMemberActivityLogCreateService	95,7%
CrewMemberActivityLogUpdateService	95,1%
CrewMemberActivityLogPublishService	95,7%
CrewMemberActivityLogDeleteService	97,4%

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 Para FlightAssignment

Comencemos especificando los casos de pruebas para las características **"flight-crew-member/flight-assignment/list-completed"** y **"flight-crew-member/flight-assignment/list-planned"**.

Caso de prueba	Tipo	Descripción	Eficacia
list.safe (para en y es)	Renderizado	Se prueba que el listado se muestre correctamente para asignaciones completadas y planeadas.	Permitió detectar que para miembros nuevos no se mostraba el botón de creación de asignaciones.
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".

Pasemos a **“flight-crew-member/flight-assignment/show?id=xx”:**

Caso de prueba	Tipo	Descripción	Eficacia
show.safe (para en y es)	Renderizado	Se prueba que una asignación se renderice correctamente sin ningún error.	No ha detectado ninguna incidencia ya que el código del servicio estaba correctamente implementado.
show.hack	Hack	Permite probar: -Miembros con otro rol no puedan ver los detalles de una asignación. -Miembros con mismo rol no puedan ver asignaciones que no le corresponden.	No se detectó ninguna incidencia ya que se realizaba una validación correcta en el “authorised”.

En lo que respecta a **“flight-crew-member/flight-assignment/create” (GET y POST):**

Caso de prueba	Tipo	Descripción	Eficacia
create.safe (para en y es)	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 (url y valores)	Permite probar: -Miembros con otro rol no puedan acceder al formulario de creación - No se puedan asignar etapas no válidas a la asignación (borradores y pasadas).	No detectó ninguna incidencia ya que el servicio estaba correctamente implementado (bind, selectChoices,...).

		- No se pueda asignar otro miembro que no sea el “logueado” a la asignación. -Atributos read-only no puedan ser modificados. - Atributos de selección no puedan ser modificados con valores ilegales .	

En lo que respecta a **“flight-crew-member/flight-assignment/update” (GET y POST):**

Caso de prueba	Tipo	Descripción	Eficacia
update.safe (para en y es)	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.
update-url.hack	Hack (url)	Permite probar: -Miembros con otro rol no puedan acceder a los detalles de la asignación por medio del “update”. - Miembros del mismo rol no puedan acceder a los detalles de una asignación que no les pertenece por medio del “update”. - Asignaciones ya publicadas no puedan ser actualizadas.	No detectó ninguna incidencia porque todo estaba bien validado e implementado.
update-values.hack	Hack (values)	Permite probar que:	No detectó ninguna incidencia porque todo estaba bien

		-Atributos read-only no puedan ser modificados. - Atributos de selección no puedan ser modificados con valores ilegales.	validado e implementado.

En lo que respecta a **“flight-crew-member/flight-assignment/publish” (GET y POST):**

Caso de prueba	Tipo	Descripción	Eficacia
publish.safe (para en y es)	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.
publish-url.hack	Hack (url)	Permite probar: -Miembros con otro rol no puedan acceder a los detalles de la asignación por medio del “publish”. - Miembros del mismo rol no puedan acceder a los detalles de una asignación que no les pertenece por medio del “publish”. - Asignaciones ya publicadas no puedan volver a publicarse.	No detectó ninguna incidencia porque todo estaba bien validado e implementado.
publish - values.hack	Hack (values)	Permite probar que: -Atributos read-only no puedan ser modificados. - Atributos de selección no puedan ser modificados con valores ilegales.	No detectó ninguna incidencia porque todo estaba bien validado e implementado.

Por último, para **“flight-crew-member/flight-assignment/delete” (GET y POST):**

Caso de prueba	Tipo	Descripción	Eficacia
delete.safe (para en y es)	Casos positivos y negativos del formulario	-Se prueba que una asignación se borre correctamente, incluso al meter valores diferentes de los mostrados, bien positivos o negativos. - Asignaciones con logs asociados se borren correctamente.	No detectó ninguna incidencia ya que todos los campos estaban correctamente validados.
delete-url.hack	Hack (url)	Permite probar: -Miembros con otro rol no puedan acceder a los detalles de la asignación por medio del “delete”. - Miembros del mismo rol no puedan acceder a los detalles de una asignación que no les pertenece por medio del “delete”. - Asignaciones ya publicadas no puedan ser eliminadas.	No detectó ninguna incidencia porque todo estaba bien validado e implementado.
delete-values.hack	Hack (values)	Permite probar que: -Atributos read-only no puedan ser modificados. - Atributos de selección no puedan ser modificados con valores ilegales.	Al modificar un atributo de selección el formulario se volvía read-only porque se debía eliminar una variable global añadida en el unbind.

1.3.2 Para ActivityLog

Comencemos especificando los casos de pruebas para las características **“flight-crew-member/activity-log/list?masterId=xx”**

Caso de prueba	Tipo	Descripción	Eficacia
list.safe (para en y es)	Renderizado	Se prueba que el listado de incidentes se muestre correctamente tanto para asignaciones completadas como para asignaciones publicadas planeadas cuyo “scheduledDeparture” ya haya comenzado.	Permitió detectar una discrepancia entre el botón de mostrar incidencias de las asignaciones y los servicios de ActivityLog, pues el primero tomaba el “scheduledDeparture” como inclusive y los servicios no.
list.hack	Hack	Se prueba que: -Miembros con otro rol no puedan ver un listado de incidencias. -Miembros del mismo rol no puedan ver un listado de incidencias de una asignación que no les pertenece. - Miembros del rol correcto no puedan ver el listado de incidencias de asignaciones no publicadas o planeadas (y publicadas) pero que el todavía no han comenzado	No se detectó ninguna incidencia ya que se realizaba una validación correcta en el “authorised”.

Pasemos a **“flight-crew-member/activity-log/show?id=xx”:**

Caso de prueba	Tipo	Descripción	Eficacia
show.safe (para en y es)	Renderizado	Se prueba que una incidencia se renderice correctamente sin ningún error.	No ha detectado ninguna incidencia ya que el código del servicio estaba correctamente implementado .
show.hack	Hack	Permite probar: -Miembros con otro rol no puedan ver los detalles de una incidencia. -Miembros con mismo rol no puedan ver incidencia de asignaciones que no le corresponden.	No se detectó ninguna incidencia ya que se realizaba una validación correcta en el “authorised”.

En lo que respecta a **“flight-crew-member/activity-log/create?masterId=xx” (GET y POST):**

Caso de prueba	Tipo	Descripción	Eficacia
create.safe (para en y es)	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. En esta entidad es sencillo porque solo hay tres campos String modificables.	No detectó ninguna incidencia ya que todos los campos estaban correctamente validados.
create.hack	Hack (url y valores)	Permite probar: -Miembros con otro rol no puedan acceder al formulario de creación de un incidente. - Miembros del mismo rol no puedan acceder al formulario de creación de un incidente asociado a una asignación que no les corresponde.	No detectó ninguna incidencia ya que el servicio estaba correctamente implementado.

		- Miembros del rol correcto no puedan acceder al formulario de creación de un incidente asociado a una asignación en modo borrador (asociado o no a una etapa en modo borrador) o a una asignación publicada cuyo “scheduledDeparture” no ha comenzado. - La fecha (atributo read-only) no pueda ser modificada.	

En lo que respecta a **“flight-crew-member/activity-log/update” (GET y POST):**

Caso de prueba	Tipo	Descripción	Eficacia
update.safe (para en y es)	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 (principalmente la longitud al ser los campos String).	No detectó ninguna incidencia ya que todos los campos estaban correctamente validados.
update-url.hack	Hack (url)	Permite probar: -Miembros con otro rol no puedan ver los detalles de una incidencia mediante el “update”. -Miembros con mismo rol no puedan ver incidencia de asignaciones que no le corresponden mediante el “update”. - Miembros del rol correcto no puedan tratar de actualizar incidencias ya publicadas.	No detectó ninguna incidencia porque todo estaba bien validado e implementado.
update-values.hack	Hack (values)	Permite probar que:	No se detectó ninguna incidencia ya que la fecha no

		-Atributos read-only (una fecha) no puedan ser modificados.	estaba en el “bind” del servicio.

En lo que respecta a **“flight-crew-member/ activity-log /publish” (GET y POST):**

Caso de prueba	Tipo	Descripción	Eficacia
publish.safe (para en y es)	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 (principalmente la longitud al ser los campos String).	No detectó ninguna incidencia ya que todos los campos estaban correctamente validados.
publish-url.hack	Hack (url)	Permite probar: -Miembros con otro rol no puedan ver los detalles de una incidencia mediante el “publish”. -Miembros con mismo rol no puedan ver incidencia de asignaciones que no le corresponden mediante el “publish.” - Miembros del rol correcto no puedan tratar de publicar incidencias ya publicadas.	No detectó ninguna incidencia porque todo estaba bien validado e implementado.
publish - values.hack	Hack (values)	Permite probar que: -Atributos read-only (una fecha) no puedan ser modificados.	No se detectó ninguna incidencia ya que la fecha no estaba en el “bind” del servicio.

Por último, para **“flight-crew-member/ activity-log /delete” (GET y POST):**

Caso de prueba	Tipo	Descripción	Eficacia
delete.safe (para en y es)	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-url.hack	Hack (url)	Permite probar: -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 una incidencia que no les pertenece por medio del “delete”. - Incidencias ya publicadas no puedan ser borradas.	No detectó ninguna incidencia porque todo estaba bien validado e implementado.
delete-values.hack	Hack (values)	Permite probar que: -Atributos read-only (una fecha) no puedan ser modificados (se debe borrar la incidencia aunque se modifique este campo).	Permitió detectar el mismo error en el unbind que en las asignaciones.

1.4 Calidad de la cobertura

En esta sección del documento se pretende describir la calidad de la cobertura para los servicios de las entidades FlightAssignment y ActivityLog. Tanto para el listado como para los detalles, se alcanza un 100% de cobertura (menos para el listado de los incidentes, con un 99%), lo que significa que se cubren todas y cada una de las líneas de código.

Para los servicios de CRUD, la cobertura gira alrededor del 94-98% debido a los métodos de validación. Si bien en los casos de prueba se cubren prácticamente todos los escenarios posibles, he añadido validaciones extra de seguridad con el objetivo de tener un sistema sólido y fiable, para asegurar que no hay ninguna brecha de seguridad. Por ejemplo, se han añadido validaciones duplicadas en el "authorise" y en el "validate" para asegurar la seguridad en todas las capas. También se han contemplado casos muy extremos que ni si quiera se pueden probar directamente a través del frontend, como que lleguen miembros o "logs" nulos a las validaciones, pues se deberían detectar en el authorise.

Por estos motivos alcanzar el 100% de cobertura en todos los servicios es prácticamente imposible, o por lo menos inviable.

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
Rog Strix G513RM	16.0 GB	AMD Ryzen 7 6800H 3.20 GHz	Nvidia RTX 3060 6 GB	954 GB	Windows 11 Pro

Para este estudio se va a emplear únicamente los casos de prueba que he ejecutado individualmente como estudiante tres, 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 eliminando las irrelevantes, podemos afirmar que, para el estudio, se utiliza un total de 1398 filas de datos. En ambos casos se presenta una media y desviación similares, de los que se pueden derivar, además, los siguientes datos:

Estadística Descriptiva	
Media	6,071110801
Error típico	0,204131532
Mediana	4,5267
Moda	5,2934
Desviación estándar	7,632444942
Varianza de la muestra	58,25421579
Curtosis	22,75120182
Coeficiente de asimetría	3,880883166
Rango	80,0732
Mínimo	0,5502
Máximo	80,6234
Suma	8487,4129
Cuenta	1398
Nivel de confianza(95,0%)	0,400437385

interval(ms)	5,67067342	6,47154819
interval(s)	0,00567067	0,00647155

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

Estadística Descriptiva	
Media	6,103564664
Error típico	0,202931794
Mediana	4,6203
Moda	2,0655
Desviación estándar	7,587586924
Varianza de la muestra	57,57147533
Curtosis	19,71915256
Coeficiente de asimetría	3,669247004
Rango	78,6844
Mínimo	0,5335
Máximo	79,2179
Suma	8532,7834
Cuenta	1398
Nivel de confianza(95,0%)	0,398083903

interval (ms)	5,70548076	6,50164857
interval (s)	0,00570548	0,00650165

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

Como podemos observar, las trazas “antes” y “después” presentan medias (≈ 6 ms) y dispersiones (≈ 7,5 ms) prácticamente iguales, por lo que en un principio la incorporación de índices no introduce cambios significativos en el rendimiento.

Esto se puede observar también en los intervalos de confianza, que se solapan totalmente. Sin embargo, para ambas trazas se cumple con creces el requisito de rendimiento, ya que los límites superiores (≈ 6,5 ms) son ambos menores a 1 segundo (nuestro requisito base).

Por tanto, una de las conclusiones que podemos sacar es que este equipo ejecuta los casos de prueba de una forma muy rápida. Así, aunque implementemos pequeñas mejoras como los índices, verdaderamente no se va a notar en el tiempo empleado, puesto que de por sí el tiempo base es muy pequeño.

2.4 Gráficos de eficiencia por característica

Uno de los aspectos que más nos interesa del rendimiento, es comprobar cuál es la petición más ineficiente (MIR), y ver si los índices suponen una mejora real en el rendimiento de la misma.

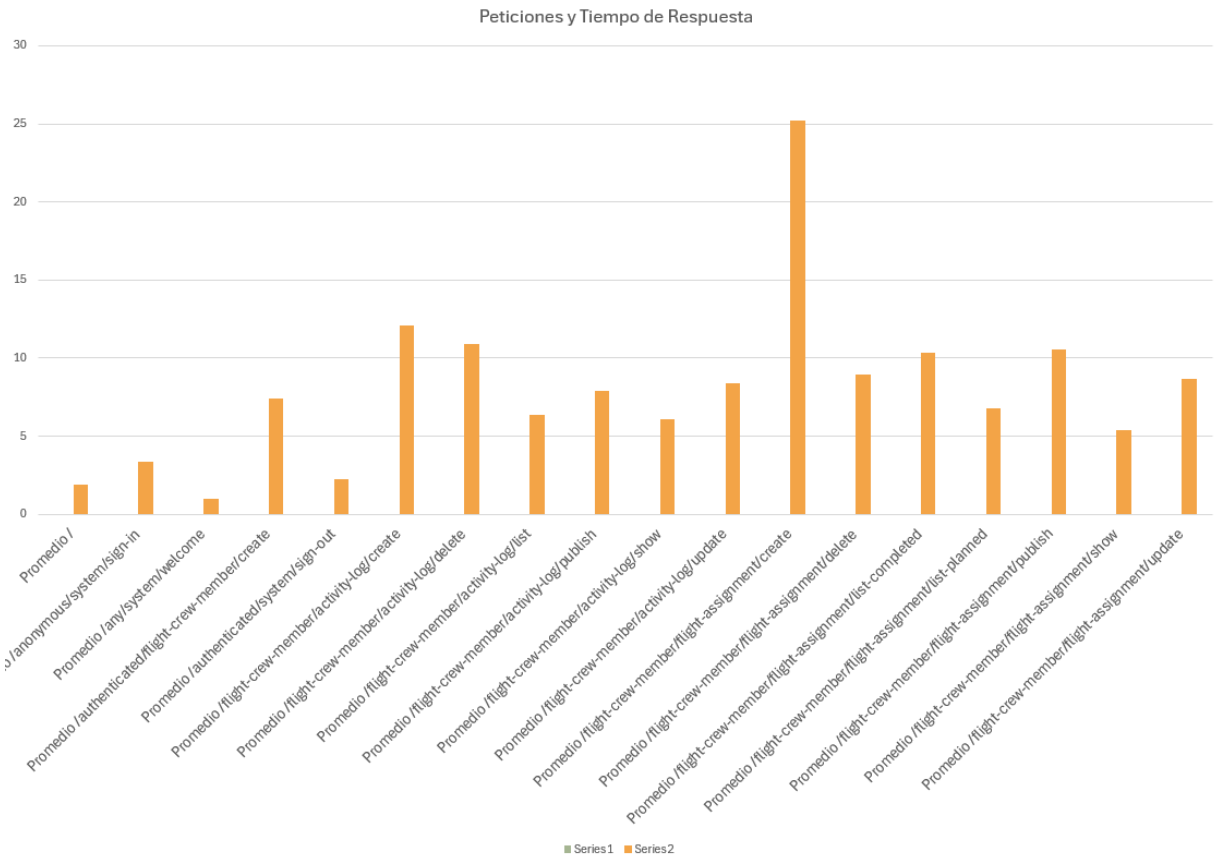

Ilustración 3: Gráfico de eficiencia de la traza sin mejorar

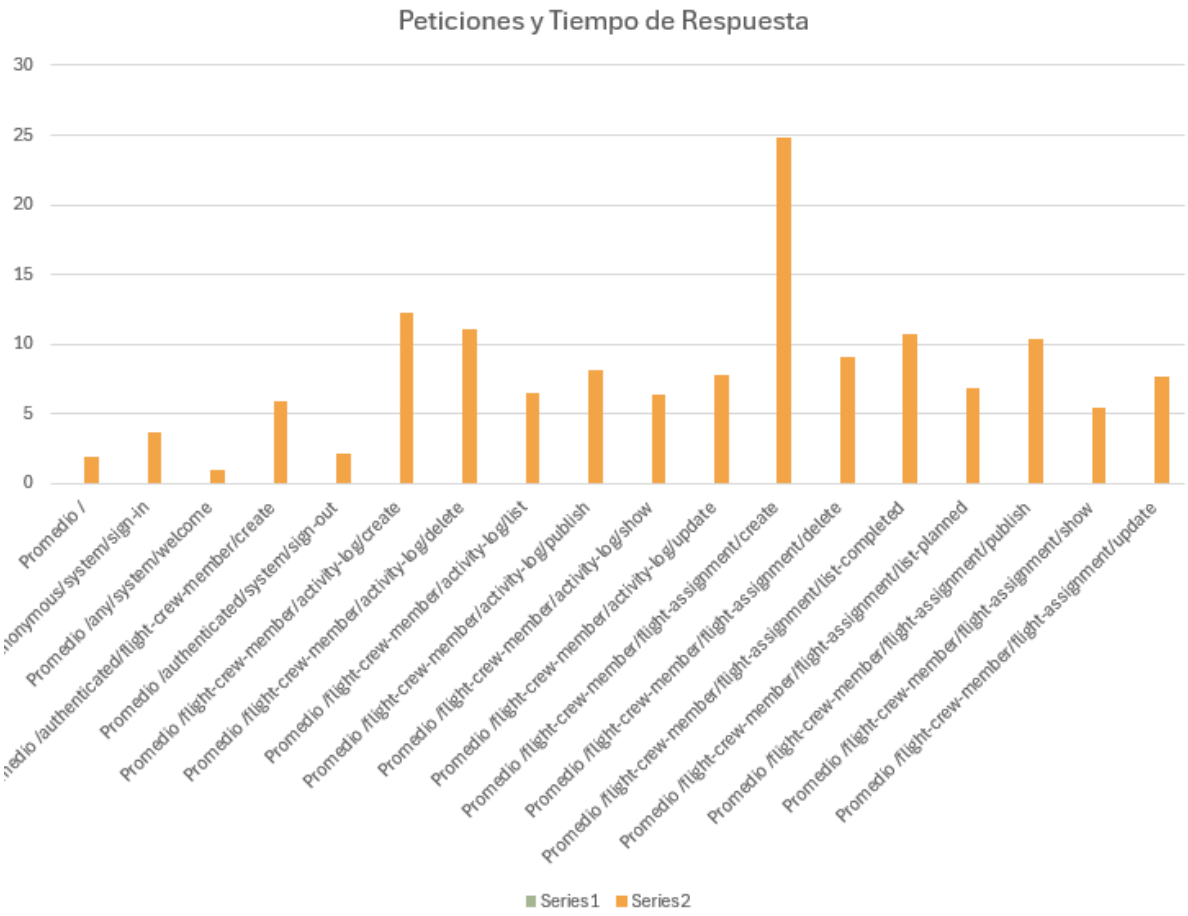

Ilustración 4: Gráfico de eficiencia de la traza mejorada

Claramente hay una petición mucho más ineficiente que el resto, **"flight-crew-member/flight-assignment/create"**. Esto se debe a que, para crear una asignación, se debe recuperar información de las etapas a vincular y del miembro asociado, lo que hace que tenga una carga de trabajo grande.

Sin embargo, si nos fijamos detenidamente, vemos que el rendimiento de esta petición mejora mínimamente tras implementar los índices. Esto puede deberse a que verdaderamente hayan supuesto una mejora real, o al ruido generado en el ordenador al ejecutar los casos de prueba, puesto que el tiempo de respuesta es tan bajo que, cualquier pequeño proceso, elemento que tenga en memoria, etc., puede afectar mínimamente al rendimiento.

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 finalizar con las pruebas de rendimiento, vamos a concluir este informe con una decisión firme sobre los resultados de las dos trazas generadas. Tras comparar el tiempo de las peticiones realizadas en ambas, hemos obtenido los siguientes datos:

	before	*after*
Media	6,071110801	6,10356466
Varianza (conocida)	58,25421579	57,5714753
Observaciones	1398	1398
Diferencia hipotética de las medias	0	
z	-0,112750254	
P(Z<=z) una cola	0,455114279	
Valor crítico de z (una cola)	1,644853627	
P(Z<=z) dos colas	0,910228558	
Valor crítico de z (dos colas)	1,959963985	

Ilustración 5: Prueba z para muestras de dos medias

El tiempo que realmente nos interesa de estos datos es el **valor "p" de dos colas (≈ 0,910228558)**. Sabiendo que **"α = 1 – nivel de confianza"**, nos queda que este valor "**p > α > 1 – 0.95 -> p > 0.05"**. Como este valor se sitúa a la derecha del intervalo [0, α], podemos afirmar aquello que hemos estado comentado en este informe, y es que verdaderamente el implementar índices no ha supuesto un cambio significativo.

En definitiva, hubiera sido mucho más interesante poder realizar estas secciones con un equipo que tarde más tiempo en ejecutar los casos de prueba, puesto que, al ser el tiempo tan pequeño, al final la varianza en el mismo se va a ver influida generalmente por el ruido en el equipo, y no por las mejoras que se hayan podido implementar en la aplicación. Sin embargo, creía interesante realizar el estudio completo, ya que permite comprender mucho más a fondo como funciona la lógica dentro de la aplicación.

4. Testing de Mutaciones

En esta sección del documento se pretende describir las mutaciones realizadas para cumplir con el **requisito 22** del estudiante tres, e informar sobre los resultados, describiendo los casos de prueba que los invalidan.

4.1 Primera mutación: Intercambio de valores

Se intercambian los valores del dataset en el **unbind** de "**CrewMemberAssignmentShowService"**. Es decir:

```
dataset.put("crewRoles", choicesCrewRole);
dataset.put("assignmentStatuses", choicesAssignmentStatus);
dataset.put("masterId", assignment.getId());

dataset.put("draftMode", assignment.isDraftMode());
```

Ilustración 6: Variables declaradas correctamente

```
dataset.put("crewRoles", choicesAssignmentStatus);
dataset.put("assignmentStatuses", choicesCrewRole);
dataset.put("masterId", assignment.getId());

dataset.put("draftMode", assignment.isDraftMode());
```

Ilustración 7: Variables intercambiadas en los datasets

Este mutante es invalidado por todos los casos de prueba ".safe" de FlightAssignment, ya que en todos ellos se realiza al menos una petición para mostrar los detalles, y, al recuperar información diferente a la grabada por el cambio en el código, salta error al intentar hacer la petición "show".

Además, este mutante también es combatido en todos los casos ".safe" de ActivityLog, ya que, para poder realizar cualquier acción sobre dicha entidad, es necesario pasar siempre por los detalles de un FlightAssignment, salvo en casos excepcionales donde se tenga la url directa al recurso y se pueda copiar directamente en la navegación.

4.2 Segunda mutación: Eliminar línea de código en validación

Se elimina la primera validación obligatoria del servicio de creación de una asignación (**CrewMemberAssignmentCreateService**). Es decir:

```
if (leg != null && member != null) {

    // 1. Evitar duplicados
    //boolean duplicateAssignment = this.repository.existsPublishedAssignmentForLegAndCrewMember(leg.getId(), member.getId());
    //super.state(!duplicateAssignment, "leg", "crewMember.assignment.error.duplicate-assignment");

    // 2. Solo un piloto y copiloto por etapa
    if (role == CrewRole.PILOT || role == CrewRole.COPILOT || role == CrewRole.LEADATTENDANT) {
        boolean roleAlreadyAssigned = this.repository.existsPublishedAssignmentForLegWithRole(leg.getId(), role);
        super.state(!roleAlreadyAssigned, "crewRole", "crewMember.assignment.error.duplicate-role");
    }

    // 3. El miembro debe estar disponible
    boolean isAvailable = member.getAvailabilityStatus() == AvailabilityStatus.AVAILABLE;
    super.state(isAvailable, "*", "crewMember.assignment.error.not-available");
```

Ilustración 8; Método "validate" con la primera restricción eliminada

Este mutante es invalidado principalmente en el caso de prueba **"flight-crew-member/flight-assignment/create"**, ya que, entre otras comprobaciones, se prueba la restricción comentada, de forma que se espera un **"leg$error=The member already has an assignment for that leg!" (sacado del error)**. Sin

embargo, al comentarse, se crea la asignación correctamente, por lo que se genera la discrepancia y el "replayer" se queja.

Este mutante también se combate en otros muchos casos de prueba porque generalmente se utiliza el miembro sin asignaciones para probar los servicios, por lo que casi siempre es obligatorio pasar a través del servicio de creación.

4.3 Tercera mutación: Cambiar lógica de authorise

Se invierte la condición del authorise en el servicio de listado de ActivityLog (**CrewMemberActivityLogListService**) sobre cuándo entender que una etapa ha comenzado. Es decir:

```
// El vuelo debe haber comenzado
//boolean legStarted = !assignment.getLeg().getScheduledDeparture().after(MomentHelper.getCurrentMoment());
boolean legStarted = assignment.getLeg().getScheduledDeparture().after(MomentHelper.getCurrentMoment());
```

Ilustración 9: Comparativa de lógica original vs invertida

Este mutante se invalida desde un primer momento en **"flight-crew-member\activity-log\create-en.safe"**, ya que para acceder a la creación de un incidente se debe pasar por el listado de los incidentes asociados a una asignación.

Como se ha invertido la lógica, ahora el servicio de listado debería comprobar que la etapa de la asignación no haya comenzado, por lo que al hacer el listado grabado en la traza sobre una asignación asociada a una etapa que ya ha ocurrido, ocurre el error **"Expected 'status' to be '200', but got '500'"**, siendo capturado en el authorise.

4.4 Cuarta mutación: Eliminar dataset en el unbind

Se eliminar una variable añadida en el servicio de borrado de un incidente (**CrewMemberActivityLogDeleteService**). Esta variable permite que el formulario sepa si estamos ante una instancia que es un borrador válido, por ejemplo, para cuando tratamos de acceder directamente al recurso a través de una url con "delete" o cuando hay un error de renderizado y se debe entrar en el "unbind". Si no se pone esta variable, el formulario no sabría si es un borrador válido y no mostraría los botones que debería tener asociados, como el de actualización y publicación. Aquí se puede apreciar el mutante:

```
dataset = super.unbindObject(log, "registrationMoment", "typeOfIncident", "description", "severityLevel");
//dataset.put("validDraft", log.isDraftMode() &&
//!log.getFlightAssignment().isDraftMode() &&
//!log.getFlightAssignment().getLeg().isDraftMode());
```

Ilustración 10: Línea de codigo comentada que produce el mutante

Este mutante se invalida a través de **"flight-crew-member\activity-log\delete-values.hack"**, ya que al meter un valor incorrecto en uno de los selectores, se debe renderizar de nuevo el formulario a partir del servicio de borrado, pero como no hay

“validDraft”, entonces esta variable no es registrada por el “replayer”. Además, como hemos mencionado, no se mostraría ningún botón asociado como borrador.

Así: ***“Expected 'payload' to be '{advertisement=437, ..., validDraft=true, ...}', but got '{advertisement=437, ...}'”***.

4.5 Quinta mutación: Alterar operador de consulta

Se cambia el operador de la consulta **“existsOverlappingAssignment”** del repositorio **“CrewMemberAssignmentRepository”** para que el scheduledDeparture sea solo **“<”** en vez de **“<=”**. Así:

```
@Query("""
        SELECT COUNT(a) > 0
        FROM FlightAssignment a
        WHERE a.flightCrewMember.id = :memberId
          AND a.draftMode = false
          AND a.leg.draftMode = false
          AND (
              a.leg.scheduledDeparture < :end AND a.leg.scheduledArrival >= :start
          )
    """)
boolean existsOverlappingAssignment(int memberId, Date start, Date end);
```

Ilustración 11: Consulta actualizada que refleja el mutante

Este mutante es invalidado directamente por **“flight-crew-member\flight-assignment\publish-en.safe”**, ya que se realiza primero la creación de una asignación en la que el “scheduledDeparture” es exactamente igual a la fecha de salida de otra asignación del miembro, pero como ahora el operador es solo “<”, la validación en el servicio va a pasar como falsa, no se va a detectar, y se va a crear la asignación con éxito. Así, el “replayer” detecta la discrepancia y salta el error.

5. Tabla de Revisión

Versión	Fecha	Descripción de los cambios
1.0	23/05/2025	Creación inicial del documento
1.1	23/05/2025	Añadir capítulo de mutaciones