26 Mayo de 2025

TESTING REPORT STUDENT 1

REPOSITORIO: https://github.com/AlbertoValenzuelaMunoz1/DP2-C1.002

GRUPO: C1.002

Antonio Roldán Pérez (antorolper@alum.us.es)

Contenido

1. Resumen ejecutivo

Este informe presenta los resultados de las pruebas funcionales y de rendimiento realizadas sobre el sistema desarrollado. En la primera parte, se detalla el procedimiento realizado para probar la aplicación, de forma general para todas las entidades y a continuación se han detallados los detalles concretos para probar algunas entidades. Las pruebas han permitido identificar y corregir diversos fallos críticos, especialmente en áreas como autenticación y validación de datos de entrada.

En la segunda parte, se analizó el rendimiento del sistema ejecutando las pruebas funcionales en dos equipos con diferente capacidad de procesamiento. Se recopilaron los tiempos de respuesta y se generaron intervalos de confianza del 95% para cada conjunto de datos. Asimismo, se realizó una prueba de hipótesis estadística con un 95% de confianza para realizar una comparativa entre el rendimiento obtenido antes y después de la creación de los índices.

Tabla de versiones

Versión	Fecha	Descripción
1.0.0	23/5/2025	Creación documento e informe de pruebas realizadas Informe de rendimiento y comparativa tras la creación de índices Comparativa de rendimiento con otro miembro del grupo y finalización del documento

2. Introducción

Este documento recoge de manera detallada los resultados obtenidos durante el proceso de pruebas del sistema desarrollado, tanto a nivel funcional como de rendimiento. El objetivo principal de este informe es evaluar la calidad del software mediante la verificación del cumplimiento de sus funcionalidades esperadas y el análisis de su en diferentes entornos de ejecución.

Las pruebas funcionales se centraron en verificar que las características clave del sistema operan correctamente en distintos escenarios, incluyendo tanto casos de uso comunes como situaciones de error. Para ello, se diseñaron casos de prueba específicos para cada funcionalidad del sistema, los cuales permitieron detectar errores, validar la lógica de negocio y comprobar la robustez de las validaciones implementadas.

Por otro lado, las pruebas de rendimiento se llevaron a cabo para medir el tiempo de respuesta del sistema durante la ejecución de las pruebas funcionales, utilizando dos ordenadores con distintas capacidades de hardware y en el mismo ordenador antes y después de la creación de índices. Esta evaluación permitió generar intervalos de confianza del 95% y realizar un contraste de hipótesis que permitió determinar en cuál de los equipos el sistema se desempeña mejor.

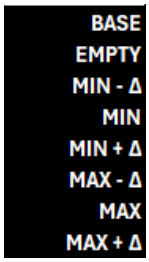
Este documento se estructura en dos capítulos principales. En el Capítulo 1, se presenta el conjunto de pruebas funcionales, organizadas por característica En el Capítulo 2, se expone el análisis de las pruebas de rendimient**o**, incluyendo gráficos representativos, los intervalos de confianza calculados y los resultados del contraste estadístico.

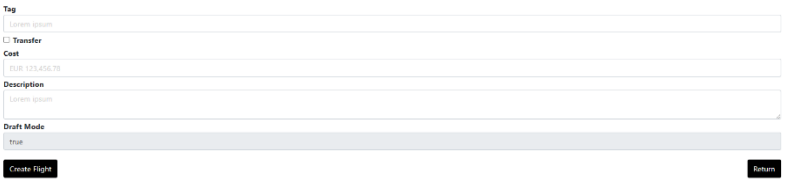
3. Pruebas realizadas

3.1. Testing flight

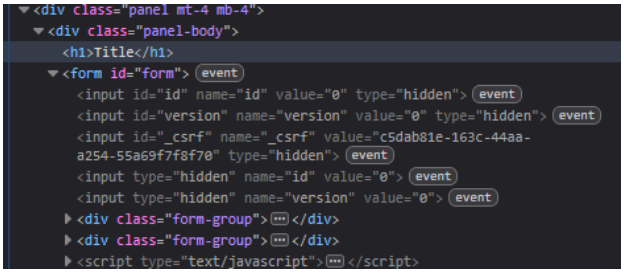
Procedimientos comunes

- Para los update/create/publish .safe, cada atributo se ha probado con los valores necesarios para probar todos los posibles casos. La propiedad tag es un string con limite de caracteres de 50, coste es un valor money con limite 1.000.000 y descripción un string con limite de caracteres de 255; primero de todo habría que enviar un formulario vacio. Hay que probar los siguientes casos:





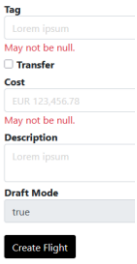
- Para la mayoría de pruebas .hack se hace uso de la herramienta de desarrollador de Firefox para poder modificar el id oculto de los formularios, siempre probaremos a poner un id de un vuelo que no pertenece a nuestro manager, de un vuelo que no exista y de un vuelo que ya está publicado. Hay que abrir la herramienta de desarrollador antes de que se inicie el récord, si no se hace una petición GET extra y al hacer el replay de las pruebas da fallo.

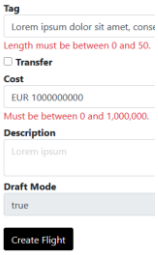


Create

- Archivo .safe

Como se comenta en los procedimientos comunes, hay que probar a enviar un formulario vacío, seguido atributo por atributo probar con cada valor que se genera en el excel “sample-data” dependiendo del tipo de atributo, por último, hay que enviar un formulario correcto para crear un flight. Hacemos shut down para acabar con el record.





- Archivo .hack

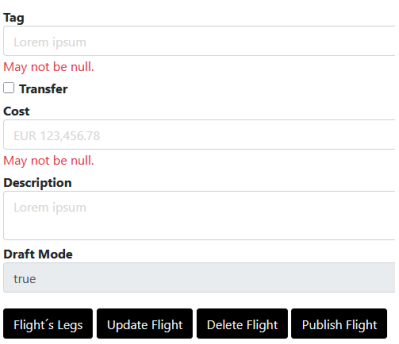
Como se comenta en los procedimientos comunes, tenemos que intentar modificar un flight de otro manager, uno ya publicado y uno que no exista, debería de saltar el autorise(). El créate siempre se envía que el id oculto sea 0, por tanto si detectamos un valor diferente no está permitido el POST.

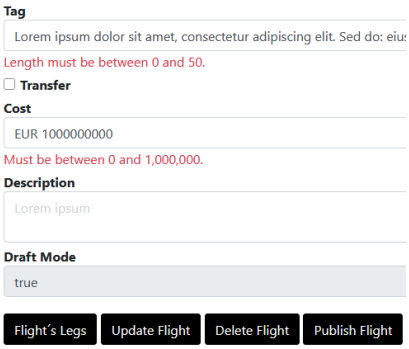
```
@Override
public void authorise() {
    boolean status = true;
    if (super.getRequest().getMethod().equals("POST")) {
        int id = super.getRequest().getData("id", int.class, 0);
        status = id == 0;
    }
    super.getResponse().setAuthorised(status);
}
```

Update

- Archivo .safe

Como se comenta en los procedimientos comunes, hay que probar a enviar un formulario vacío, seguido atributo por atributo probar con cada valor que se genera en el excel “sample-data” dependiendo del tipo de atributo, por último, hay que enviar un formulario correcto para actualizar un flight. Hacemos shut down para acabar con el record.





- Archivo .hack

Como se comenta en los procedimientos comunes, tenemos que intentar modificar un flight de otro manager, uno ya publicado y uno que no exista, debería de saltar el autorise().

```
@Override
public void authorise() {
    boolean status;
    int masterId;
    Flight flight;
    Manager manager;

    masterId = super.getRequest().getData("id", int.class);
    flight = this.repository.findFlightById(masterId).orElse(null);
    manager = flight == null ? null : flight.getManager();
    status = flight != null && flight.isDraftMode() && super.getRequest().getPrincipal().hasRealm(manager);

    super.getResponse().setAuthorised(status);
}
```

Request:
POST http://localhost:8080/Acme-ANS-D04/manager/flight/update HTTP/1.1

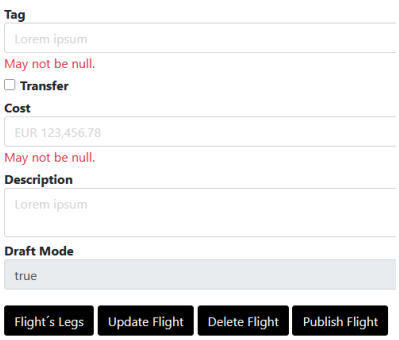
Status:
500 Internal Server Error

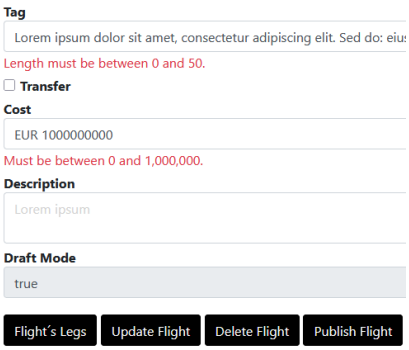
Exceptions:
acme.client.helpers.Assert.state(Assert.java:45)
↳ java.lang.AssertionError: Access is not authorised

Publish

- Archivo .safe

Como se comenta en los procedimientos comunes, hay que probar a enviar un formulario vacío, seguido atributo por atributo probar con cada valor que se genera en el excel “sample-data” dependiendo del tipo de atributo, por último, hay que enviar un formulario correcto para publicar un flight. En el publish tenemos que probar también que el flight tenga al menos una leg y que todas las legs estén publicadas. Hacemos shut down para acabar con el record.





- Archivo .hack

Como se comenta en los procedimientos comunes, tenemos que intentar modificar un flight de otro manager, uno ya publicado y uno que no exista, debería de saltar el autorise().

```
@Override
public void authorise() {
    boolean status;
    int flightId;
    Flight flight;
    Manager manager;

    flightId = super.getRequest().getData("id", int.class);
    flight = this.repository.findFlightById(flightId).orElse(null);
    manager = flight == null ? null : flight.getManager();
    status = flight != null && flight.isDraftMode() && super.getRequest().getPrincipal().hasRealm(manager);

    super.getResponse().setAuthorised(status);
}
```

Request:
POST http://localhost:8080/Acme-ANS-D04/manager/flight/publish HTTP/1.1

Status:
500 Internal Server Error

Exceptions:
acme.client.helpers.Assert.state(Assert.java:45)
↳ java.lang.AssertionError: Access is not authorised

Delete

- Archivo .safe

Para el archivo .safe lo único que tenemos que hacer es entrar a un vuelo y borrarlo. También tenemos que probar a borrar un flight que tiene una leg publicada, algo que no está permitido porque una leg ya publicada ya no se puede borrar. Hacemos shut down para acabar con el record.

- Archivo .hack

Como se comenta en los procedimientos comunes, tenemos que intentar borrar un flight de otro manager, uno ya publicado y uno que no exista, debería de saltar el autorise().

```
@Override
public void authorise() {
    boolean status;
    int masterId;
    Flight flight;
    Manager manager;

    masterId = super.getRequest().getData("id", int.class);
    flight = this.repository.findFlightById(masterId).orElse(null);
    manager = flight == null ? null : flight.getManager();
    status = flight != null && flight.isDraftMode() && super.getRequest().getPrincipal().hasRealm(manager);

    super.getResponse().setAuthorised(status);
}
```

Request:
POST http://localhost:8080/Acme-ANS-D04/manager/flight/delete HTTP/1.1

Status:
500 Internal Server Error

Exceptions:
acme.client.helpers.Assert.state(Assert.java:45)
↳ java.lang.AssertionError: Access is not authorised

List

- Archivo .safe

Para el archivo .safe lo único que tenemos que hacer es entrar el listado de los vuelos. Hacemos shut down para acabar con el record.

- Archivo .hack

Como el link http://localhost:8080/Acme-ANS-D04/manager/flight/list lista los vuelos del manager con el que se tenga iniciada sesión, habría que probar listarlo con otro real y esto no debería de ser posible. No es necesario tener nada en el autorise(), de esto se encarga la extensión de la clase a AbstractGuiService<Manager, Flight>.

```
@Override
public void authorise() {
    super.getResponse().setAuthorised(true);
}
```

Request:
GET http://localhost:8080/Acme-ANS-D04/manager/flight/list HTTP/1.1

Status:
500 Internal Server Error

Exceptions:
acme.client.helpers.Assert.state(Assert.java:45)
↳ java.lang.AssertionError: Access is not authorised

Show

- Archivo .safe

Para el archivo .safe lo único que tenemos que hacer es entrar en el listado de vuelos y entrar a uno. Hacemos shut down para acabar con el record.

- Archivos .hack

Tenemos que intentar ver el flight de otro manager, con otra entidad debería o de un flight que no existe, debería saltar el autorise().

```
@Override
public void authorise() {
    boolean status;
    int flightId;
    Manager manager;
    Flight flight;

    flightId = super.getRequest().getData("id", int.class);
    flight = this.repository.findFlightById(flightId).orElse(null);
    manager = flight == null ? null : flight.getManager();

    status = flight != null && super.getRequest().getPrincipal().hasRealm(manager);

    super.getResponse().setAuthorised(status);
}
```

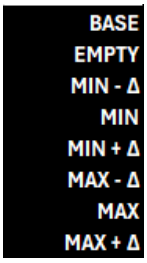
Request:
GET http://localhost:8080/Acme-ANS-D04/manager/flight/show?id=88 HTTP/1.1

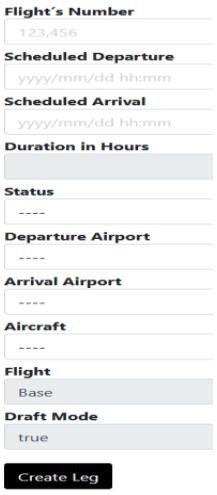
Status:
500 Internal Server Error

Exceptions:
acme.client.helpers.Assert.state(Assert.java:45)
↳ java.lang.AssertionError: Access is not authorised

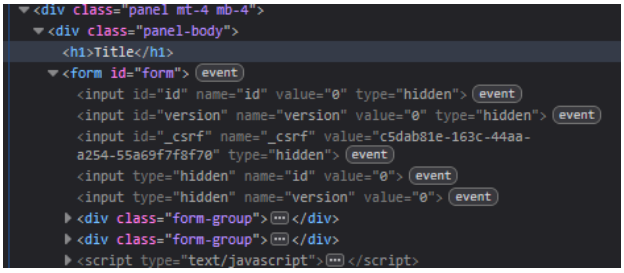
3.2. Testing leg
Procedimientos comunes

- Para los update/create/publish .safe, cada atributo se ha probado con los valores necesarios para probar todos los posibles casos. La propiedad FlightNumberDigit es un String con el formato "^[0-9]{4}$", ScheduledDeparture y ScheduledArrival son dos fotos que deben de estar en el futuro, Status es un Enum que indica el estado de la leg, DepartureAirport y ArrivalAirport son 2 aeropuertos y Aircraft es el avión asignado a la leg; primero de todo habría que enviar un formulario vacío. Hay que probar los siguientes casos:





- Para la mayoría de pruebas .hack se hace uso de la herramienta de desarrollador de Firefox para poder modificar el id oculto de los formularios, siempre probaremos a poner un id de una leg que no pertenece a nuestro manager, de una leg que no exista y de una leg que ya está publicado. Hay que abrir la herramienta de desarrollador antes de que se inicie el récord, si no se hace una petición GET extra y al hacer el replay de las pruebas da fallo. No hay que probar a meter valores en los selectChoices, puesto que este ofrece todos los valores posibles y se prueba la aptitud de este en el validate().



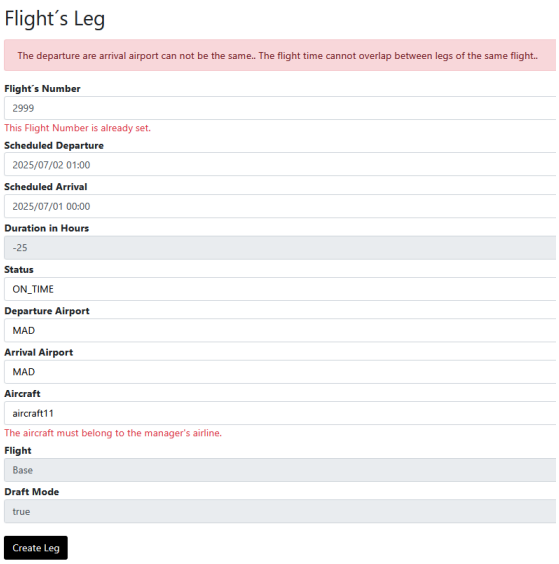
Create

- Archivo .safe

Como se comenta en los procedimientos comunes, hay que probar a enviar un formulario vacío, seguido atributo por atributo probar con cada valor que se genera en el excel “sample-data” dependiendo del tipo de atributo, por último, hay que enviar un formulario correcto para crear una leg.
También hay que probar:

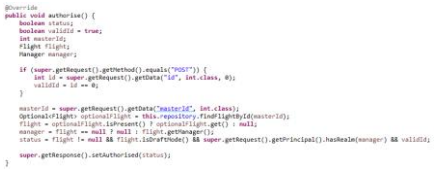
- Que las legs del mismo flight no se solapen.
- Que la hora de llegada sea después de la de salida.
- Que el aircraft no se utilice en más de una leg a la vez.
- Que el aeropuerto de salida y llegada no sea el mismo.
- Que el aircraft pertenezca a la airline del manager y que se encuentre activo.
- Que el flight number no esté ya asignado

Hacemos shut down para acabar con el record.



- Archivo .hack

Tenemos que intentar modificar el leg de otro manager, uno que ya este publicado o de uno que no existe, cambiando el id oculto. Aunque como en la entidad flight, en el créate el id se manda como 0, por tanto si se identifica un valor distinto debería de saltar el autorice()



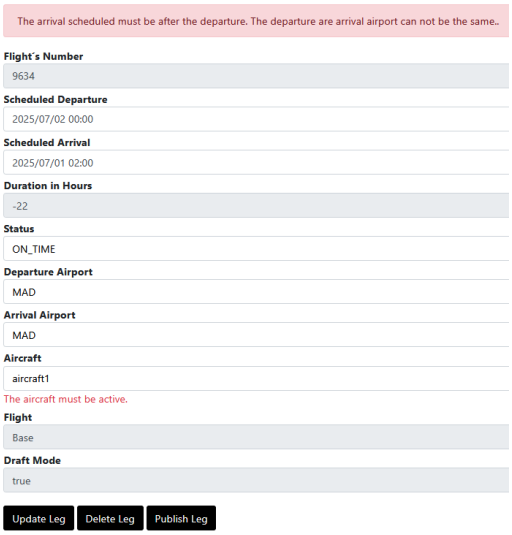
Update

- Archivo .safe

Como se comenta en los procedimientos comunes, hay que probar a enviar un formulario vacío, seguido atributo por atributo probar con cada valor que se genera en el excel “sample-data” dependiendo del tipo de atributo, por último, hay que enviar un formulario correcto para actualizar una leg.
También hay que probar:

- Que las legs del mismo flight no se solapen.
- Que la hora de llegada sea después de la de salida.
- Que el aircraft no se utilice en más de una leg a la vez.
- Que el aeropuerto de salida y llegada no sea el mismo.
- Que el aircraft pertenezca a la airline del manager y que se encuentre activo.
- Que el flight number no esté ya asignado

Hacemos shut down para acabar con el record.



- Archivo .hack

Tenemos que intentar actualizar el leg de otro manager, uno que ya este publicado o de uno que no existe, cambiando el id oculto. Esto no está permitido por lo tanto debería de saltar el autorice().

```
@Override
public void authorise() {
    boolean status;
    int masterId;
    Leg leg;
    Manager manager;

    masterId = super.getRequest().getData("id", int.class);

    Optional<Leg> optional = this.repository.findLegById(masterId);

    leg = optional.isPresent() ? optional.get() : null;

    manager = leg == null ? null : leg.getFlight().getManager();

    status = leg != null && leg.isDraftMode() && super.getRequest().getPrincipal().hasRealm(manager);

    super.getResponse().setAuthorised(status);
}
```

Request:
POST http://localhost:8080/Acme-ANS-D04/manager/leg/update HTTP/1.1

Status:
500 Internal Server Error

Exceptions:
acme.client.helpers.Assert.state(Assert.java:45)
↳ java.lang.AssertionError: Access is not authorised

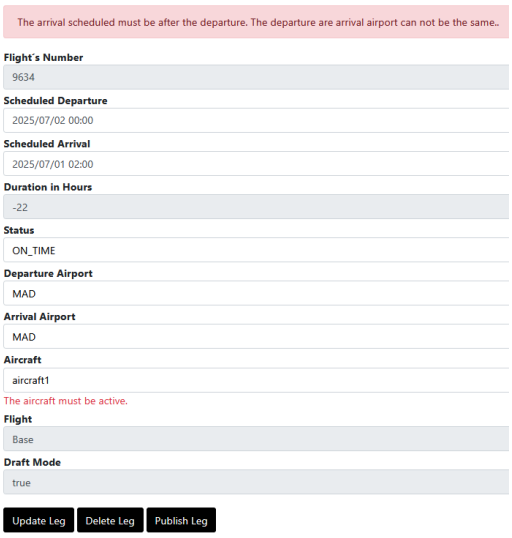
Publish

- Archivo .safe

Como se comenta en los procedimientos comunes, hay que probar a enviar un formulario vacío, seguido atributo por atributo probar con cada valor que se genera en el excel “sample-data” dependiendo del tipo de atributo, por último, hay que enviar un formulario correcto para publicar una leg.
También hay que probar:

- Que las legs del mismo flight no se solapen.
- Que la hora de llegada sea después de la de salida.
- Que el aircraft no se utilice en más de una leg a la vez.
- Que el aeropuerto de salida y llegada no sea el mismo.
- Que el aircraft pertenezca a la airline del manager y que se encuentre activo.
- Que el flight number no esté ya asignado

Hacemos shut down para acabar con el record.



- Archivo .hack

Tenemos que intentar publicar el leg de otro manager, uno que ya este publicado o de uno que no existe, cambiando el id oculto. Esto no está permitido por lo tanto debería de saltar el autorice().

```
@Override
public void authorise() {
    boolean status;
    int masterId;
    Leg leg;
    Manager manager;

    masterId = super.getRequest().getData("id", int.class);

    Optional<Leg> optional = this.repository.findLegById(masterId);

    leg = optional.isPresent() ? optional.get() : null;

    manager = leg == null ? null : leg.getFlight().getManager();

    status = leg != null && leg.isDraftMode() && super.getRequest().getPrincipal().hasRealm(manager);

    super.getResponse().setAuthorised(status);
}
```

Delete

- Archivo .safe

Para el archivo .safe lo único que tenemos que hacer es entrar a un leg y borrarlo. Hacemos shut down para acabar con el record.

- Archivo .hack

Como se comenta en los procedimientos comunes, tenemos que intentar borrar una leg de otro manager, una ya publicado y una que no exista, debería de saltar el autorise().

```
@Override
public void authorise() {
    boolean status;
    int masterId;
    Leg leg;
    Manager manager;

    masterId = super.getRequest().getData("id", int.class);
    Optional<Leg> optionalLeg = this.repository.findLegById(masterId);
    leg = optionalLeg.isPresent() ? optionalLeg.get() : null;
    manager = leg == null ? null : leg.getFlight().getManager();
    status = leg != null && leg.isDraftMode() && super.getRequest().getPrincipal().hasRealm(manager);

    super.getResponse().setAuthorised(status);
}
```

Request:
POST http://localhost:8080/Acme-ANS-D04/manager/leg/delete HTTP/1.1

Status:
500 Internal Server Error

Exceptions:
acme.client.helpers.Assert.state(Assert.java:45)
↳ java.lang.AssertionError: Access is not authorised

List

- Archivo .safe

Para el archivo .safe lo único que tenemos que hacer es entrar el listado de los legs de un vuelo. Hacemos shut down para acabar con el record.

- Archivo .hack

Tenemos que probar a listar las legs de un vuelo de otro manager o de uno que no exista. Esto no está permitido por tanto debería de salt ar el autorice().

```
@Override
public void authorise() {
    boolean status;
    int flightId;
    Manager manager;

    flightId = super.getRequest().getData("masterId", int.class);
    manager = this.repository.findFlightById(flightId).get().getManager();

    status = super.getRequest().getPrincipal().hasRealm(manager);

    super.getResponse().setAuthorised(status);
}
```

Request:
GET http://localhost:8080/Acme-ANS-D04/manager/leg/list?masterId=80 HTTP/1.1

Status:
500 Internal Server Error

Exceptions:
acme.client.helpers.Assert.state(Assert.java:45)
↳ java.lang.AssertionError: Access is not authorised

Show

- Archivo .safe

Para el archivo .safe lo único que tenemos que hacer es entrar el listado de los legs de un vuelo y entrar en un vuelo. Hacemos shut down para acabar con el record.

- Archivo .hack

Tenemos que probar a entrar en una de las legs de un vuelo de otro manager o de uno que no exista. Esto no está permitido por tanto debería de salt ar el autorice().

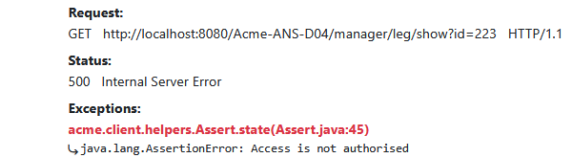
```
@Override
public void authorise() {
    boolean status;
    int legId;
    Leg leg;
    Manager manager;

    legId = super.getRequest().getData("id", int.class);
    Optional<Leg> optionalLeg = this.repository.findLegById(legId);
    leg = optionalLeg.isPresent() ? optionalLeg.get() : null;

    manager = leg != null ? leg.getFlight().getManager() : null;

    status = leg != null && super.getRequest().getPrincipal().hasRealm(manager);

    super.getResponse().setAuthorised(status);
}
```



4. Rendimiento obtenido

El objetivo de este apartado es realizar un análisis del rendimiento obtenido en las pruebas descritas anteriormente.

4.1. Gráficos rendimiento

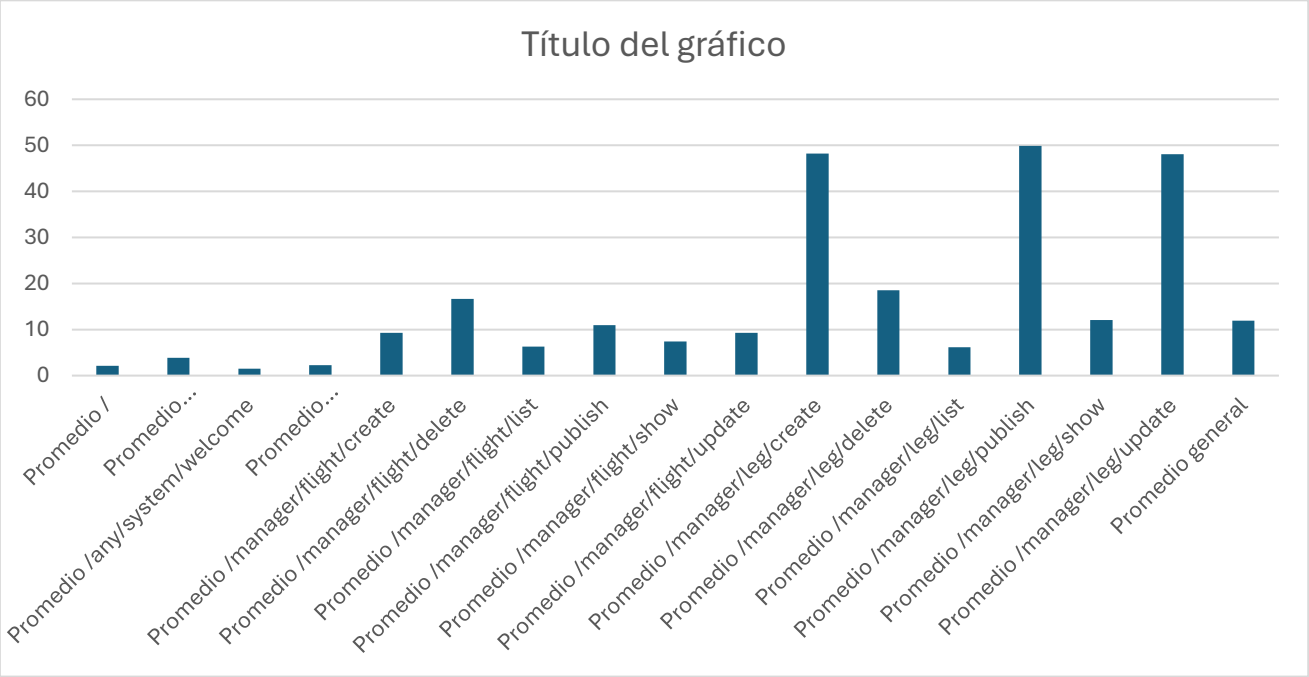


Figura 1: Promedio de tiempo de cada tipo de petición

Como se puede observar las peticiones más ineficientes son los create, update y publish, esto se debe a la gran complejidad que contienen en el validate, ya que tienen que comprobar una gran cantidad de casuísticas en cuanto a el solapamiento de las legs de un vuelo y el uso de un aircraft en más de una leg a la vez.

Por otra parte, el promedio general obtenido ha sido de 11,95 milisegundos, lo cual es una cifra que es bastante buena.

4.2. Datos estadísticos

Media	11,95606094	
Error típico	0,699052451	
Mediana	5,3771	
Moda	5,0293	
Desviación estándar	18,24245647	
Varianza de la muestra	332,787218	
Curtosis	3,802108671	
Coeficiente de asimetría	2,209409492	
Rango	102,5181	
Mínimo	0,7187	
Máximo	103,2368	
Suma	8142,0775	
Cuenta	681	
Nivel de confianza(95,0%)	1,372560638	
Intervalo de confianza	10,5835003	13,32862158

Figura 2: datos estadísticos obtenidos

Estos son los resultados estadísticos obtenidos en el análisis. El intervalo de confianza se sitúa entre 10,5 y 13,32 ms, que se puede considerar como válido, ya que en el proyecto no se impone ningún requisito relacionado con el rendimiento.

5. Comparativa rendimiento tras creación índices

No ha sido necesario implementar ningún índice, ya que no era necesario para ninguna consulta. podría haberse implementado en una consulta donde se filtran los manager por su Identifier, pero este tiene la anotación @Colum(unique = true), por tanto se hace automáticamente, sería inútil indexarlo de nuevo.

6. Comparativa entre distintos ordenadores

A continuación, se realizará una comparativa entre el rendimiento obtenido en mi ordenador y el ordenador de otro integrante del grupo (después de la creación de índices).

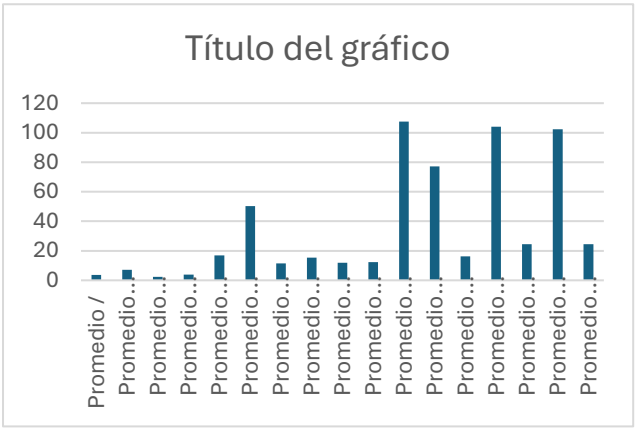


Figura 5: Promedio de tiempos obtenidos en otro ordenador

Media	24,43869163	
Error típico	1,684061605	
Mediana	8,3304	
Moda	2,4753	
Desviación estándar	43,9472324	
Varianza de la muestra	1931,359235	
Curtosis	9,742422898	
Coeficiente de asimetría	2,908535952	
Rango	305,5262	
Mínimo	0,9265	
Máximo	306,4527	
Suma	16642,749	
Cuenta	681	
Nivel de confianza(95,0%)	3,306585461	
Intervalo de confianza	21,13210617	27,74527709

Figura 6: Datos estadísticos obtenidos en otro ordenador

Como el p-value obtenido es de 9,28 E -12, que es un valor muy cercano a 0, por tanto se pueden comparar las medias obtenidas. Como en mi ordenador la media obtenida es de 11,89 ms y en el otro es de 24,33 ms, se llega a la conclusión de que el rendimiento en mi ordenador es bastante mejor (alrededor de un 48%).

7. Conclusión

La elaboración del conjunto completo de pruebas ha resultado de gran utilidad para identificar errores residuales en la aplicación, así como para detectar posibles vectores de ataque que no se habían considerado inicialmente. Este proceso ha permitido reforzar tanto la robustez funcional como la seguridad del sistema.

En lo que respecta a la comparativa de rendimiento, las diferencias observadas tras la incorporación de índices fueron mínimas, lo cual era previsible dado el reducido volumen de datos utilizado durante las pruebas. No obstante, al comparar los resultados con los de otro integrante del grupo, se constató que el rendimiento en mi equipo fue algo superior, lo que deja ver que las capacidades de cada equipo afecta.

8. Bibliografía

Intencionalmente en blanco.