

Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

## Testing Report D04 – gerojegar




Grado en Ingeniería Informática – Ingeniería del Software

Diseño y Pruebas II

Curso 2024 – 2025

| Fecha      | Versión |
|------------|---------|
| 19-05-2025 | V1.0    |
| 24-05-2025 | V1.1    |
| 25-04-2025 | V1.2    |

Repositorio de Github: <https://github.com/cesmarvan/Acme-ANS-D04.git>

| Grupo de prácticas: C1.025  |  |                      |                                  |
|---|--|----------------------|----------------------------------|
| Fotografía  | Autor  | Rol                  | Descripción del rol              |
|  | Ojeda Garrido, Germán – 29552532Q<br>UVUS: gerojegar<br>gerojegar@alum.us.es | Developer,<br>tester | Desarrollar y probar<br>features |

## Índice

|    |   |    |
|----|---|----|
| 1. | Resumen ejecutivo.....  | 3  |
| 2. | Tabla de revisión .....   | 4  |
| 3. | Introducción .....  | 5  |
| 4. | Pruebas sobre Flight.....   | 6  |
| 5. | Pruebas sobre Leg.....  | 7  |
| 6. | Cobertura del testing.....  | 8  |
| 7. | Análisis de rendimiento .....   | 9  |
| 1. | Análisis de las graficas con tiempo promedio .....                      | 9  |
| 2. | Análisis de los resultados del análisis de estadística descriptiva..... | 10 |
| 3. | Estudio del Z-Test.....   | 12 |
| 8. | Conclusiones.....   | 13 |
| 9. | Bibliografía .....  | 14 |

# 1. Resumen ejecutivo

Con este documento, se pretende analizar los resultados de haber realizado testing para las features de Flight y Leg, pertenecientes al rol Manager. Se estudiarán los resultados obtenidos para asegurar la calidad requerida.

## 2. Tabla de revisión

| Número de revisión | Fecha      | Descripción de la revisión   |
|--------------------|------------|--|
| 1.0                | 19-05-2025 | Creación del documento, portada y tabla de revisión  |
| 1.1                | 24-05-2025 | Añadida introducción, explicación de tests sobre Flight, explicación de tests sobre Leg                      |
| 1.2                | 25-04-2025 | Terminada explicación de tests sobre Flight y Leg, añadido estudio de rendimiento, conclusión y bibliografía |

### 3. Introducción

En este documento, se pretende analizar los resultados de haber realizado tests requeridos para el entregable D04, con el fin de determinar como de eficiente y cubierto está el código elaborado.

En este testing de la parte individual del Student 1, se han estudiado las entidades Flight y Leg, para asegurar un correcto funcionamiento de dichas entidades y las funcionalidades que las involucran.

Para lograr este objetivo, tal y como se explica en las distintas diapositivas de clase, se ha usado el testing-tester#record para grabar todas las peticiones que se hacen durante el testing que se realiza mientras está lanzada dicha herramienta.

El criterio seguido para nombrar los ficheros de logs resultantes de utilizar esta herramienta de testing fue el siguiente:

- Testing positivo y negativo: feature.safe.
- Testing para casos de hacking: feature .hack.

Además, con el fin de conseguir una cobertura del 100% en el código involucrado, se han añadido algunos archivos de test más aislados, que siguen la siguiente nomenclatura:

- Para testing mas aislado: feature-descripcion.hack/safe.

## 4. Pruebas sobre Flight

Pruebas realizadas sobre Flight:

- LIST: se ha probado como funciona la feature *ManagerFlightListService.java*, en la que se ha probado que un *Manager* puede acceder a la lista de sus *Flights*, y un usuario que no es *Manager* le salta una vista de *Unauthorized* al intentar acceder a la url asociada.
- SHOW: se ha probado como funciona la feature *ManagerFlightShowService.java*, en la que se ha probado que un *Manager* puede acceder a los detalles de un vuelo que el ha creado, pero otros usuarios, ya sean otros *Managers* u otro tipo de usuario, no puedan acceder a dichos detalles.
- CREATE/UPDATE/PUBLISH: se ha probado como funcionan las funcionalidades de *ManagerFlightCreateService.java*, *ManagerFlightUpdateService.java*, *ManagerFlightPublishService.java*, en los cuales se han probado peticiones legales tanto positivas, realizando la función correctamente, como negativas, en las que han saltado los errores de validación pertinentes.  
Estás restricciones para los distintos atributos se han declarado tanto en la función *validate* del *service* de cada *feature* así como, aquellas que sean comunes a los tres servicios y deban también comprobar que los datos al cargarlos desde un csv, en el validador de *Flight*.  
Se ha comprobado que los atributos no pueden ser nulos, o, aquellos que lo requieran, cumplan el formato requerido así como la extensión mínima y máxima de caracteres.  
Para las peticiones ilegales y/o hacking, se ha probado que para cada una de ellas se devuelve un *Panic* con mensaje *Unauthorized*, invalidando dichas acciones ilegales, entre las que se encuentran realizar estas acciones sin estar autenticado o con un rol inválido, alterar las *Ids*, ya sea del objeto en si o de atributos derivados de otros objetos, desde la consola de comandos o desde la url.
- DELETE: se realizó testing legal positivo para lograr que se eliminen los *Flights* que no están publicados y no tienen *Legs* publicadas, y se verificó que no se pueden borrar aquellos *Flights* que no pertenezcan al usuario que ha iniciado sesión, así como aquellos que tienen una o mas *Legs* publicadas.

## 5. Pruebas sobre Leg

Pruebas realizadas sobre Leg:

- LIST: se ha probado como funciona la feature *ManagerLegListService.java*, en la que se ha probado que un *Manager* puede acceder a la lista de los *Legs* de uno de sus *Flights*, y un usuario que no es *Manager* le salta una vista de *Unauthorized* al intentar acceder a la url asociada.
- SHOW: se ha probado como funciona la feature *ManagerLegShowService.java*, en la que se ha probado que un *Manager* puede acceder a los detalles de un *Leg* que el ha creado, pero otros usuarios, ya sean otros *Managers* u otro tipo de usuario, no puedan acceder a dichos detalles.
- CREATE/UPDATE/PUBLISH: se ha probado como funcionan las funcionalidades de *ManagerLegCreateService.java*, *ManagerLegUpdateService.java*, *ManagerLegPublishService.java*, en los cuales se han probado peticiones legales tanto positivas, realizando la función correctamente, como negativas, en las que han saltado los errores de validación pertinentes.

Al igual que con la entidad *Flight*, estas restricciones y validaciones se han declarado tanto en el método *validate* de cada *feature*, y, si son comunes y también tienen que comprobar la población de datos, se han declarado en el *LegValidator*. Para cada atributo, se ha comprobado que no pueden ser nulos, además de que cada uno de ellos deba cumplir el formato según el tipo de dato que tenga el atributo. Además, para *flightNumber*, se comprueba que al introducir uno que ya existe, muestra un mensaje de error, ya que es un atributo único y no debe haber ninguno que se repita en la base de datos.

Para las fechas, se han comprobado que la fecha de salida no está en el pasado, según la fecha que se nos proporciona en el archivo *application.properties*. Además, se comprueba que la fecha de llegada es posterior a la fecha de salida.

Por otra parte, los aeropuertos de llegada y de salida se restringen a que sean diferentes, y, a la hora de publicar varias *Leg*, se comprueba que, siempre que el atributo *Self-Transfer* del vuelo asociado a ellas sea *false*, deben coincidir el de llegada de un *Leg* con el de salida del siguiente *Leg*.

Para las peticiones ilegales y/o hacking, se ha probado que para cada una de ellas se devuelve un *Panic* con mensaje *Unauthorized*, como, por ejemplo, alterar el *Id* del formulario desde la consola del navegador o la url.

Además, para *departureAirport*, *arrivalAirport* y *aircraft*, que son atributos que relacionan la *Leg* en cuestión con objetos de las entidades *Airport* y *Aircraft*, se ha protegido el sistema impidiendo que se altere la *Id* de los mismos en el formulario.
- DELETE: se realizó testing legal positivo para lograr que se eliminen los *Legs* que no están publicados y se verificó que solo el *Manager* que haya creado el *Leg* puede borrarlo, así como que un *Leg* que ya está publicado no se puede borrar.

## 6. Cobertura del testing

Tras grabar todos los archivos .safe y .hack necesarios, se ha realizado un testing-tester#replay con la opción de Coverage, para ver que todos los test pasan satisfactoriamente y comprobar que la cobertura de código es del 100% en la mayoría de los casos, ya que, para la feature Delete de Leg, no se consiguió que se cubriera el método Unbind.

|   |                                  |         |     |   |     |
|---|----------------------------------|---------|-----|---|-----|
| ▼ | acme.features.manager.flight     | 100,0 % | 931 | 0 | 931 |
| > | ManagerFlightController.java     | 100,0 % | 35  | 0 | 35  |
| > | ManagerFlightCreateService.java  | 100,0 % | 125 | 0 | 125 |
| > | ManagerFlightDeleteService.java  | 100,0 % | 202 | 0 | 202 |
| > | ManagerFlightListService.java    | 100,0 % | 67  | 0 | 67  |
| > | ManagerFlightPublishService.java | 100,0 % | 217 | 0 | 217 |
| > | ManagerFlightShowService.java    | 100,0 % | 126 | 0 | 126 |
| > | ManagerFlightUpdateService.java  | 100,0 % | 159 | 0 | 159 |

Ilustración 1: Cobertura obtenida para Flight

|   |                               |         |       |     |       |
|---|-------------------------------|---------|-------|-----|-------|
| ▼ | acme.features.manager.leg     | 93,9 %  | 1.856 | 121 | 1.977 |
| > | ManagerLegDeleteService.java  | 45,7 %  | 102   | 121 | 223   |
| > | ManagerLegController.java     | 100,0 % | 35    | 0   | 35    |
| > | ManagerLegCreateService.java  | 100,0 % | 466   | 0   | 466   |
| > | ManagerLegListService.java    | 100,0 % | 85    | 0   | 85    |
| > | ManagerLegPublishService.java | 100,0 % | 529   | 0   | 529   |
| > | ManagerLegShowService.java    | 100,0 % | 211   | 0   | 211   |
| > | ManagerLegUpdateService.java  | 100,0 % | 428   | 0   | 428   |

Ilustración 2: Cobertura obtenida para Leg

```
@Override
public void unbind(final Leg leg) {
    Dataset dataset;

    int masterId;
    masterId = super.getRequest().getData("masterId", int.class);
    Flight flight = this.repository.findFlightById(masterId);

    SelectChoices statusChoices;
    statusChoices = SelectChoices.from(LegStatus.class, leg.getStatus());

    SelectChoices aircraftChoices;
    List<Aircraft> aircrafts = this.repository.findActivesAircrafts(AircraftStatus.ACTIVE_SERVICE);
    aircraftChoices = SelectChoices.from(aircrafts, "registrationNumber", leg.getAircraft());

    SelectChoices airportDepartureChoices;
    SelectChoices airportArrivalChoices;
    List<Airport> airports = this.repository.findAllAirports();
    airportDepartureChoices = SelectChoices.from(airports, "iataCode", leg.getDepartureAirport());
    airportArrivalChoices = SelectChoices.from(airports, "iataCode", leg.getArrivalAirport());

    dataset = super.unbindObject(leg, "flightNumber", "scheduledDeparture", "scheduledArrival", "status");

    dataset.put("duration", leg.getTravelHours());
    dataset.put("status", statusChoices);
    dataset.put("aircrafts", aircraftChoices);
    dataset.put("aircraft", aircraftChoices.getSelected().getKey());
    dataset.put("departureAirports", airportDepartureChoices);
    dataset.put("departureAirport", airportDepartureChoices.getSelected().getKey());
    dataset.put("arrivalAirports", airportArrivalChoices);
    dataset.put("flight", flight);

    super.getResponse().addData(dataset);
}
```

Ilustración 3: Método Unbind de Leg sin cobertura



## 7. Análisis de rendimiento

Para llevar a cabo el análisis de rendimiento, se analizará el `tester.trace` generado al hacer el `tester-testing#replay` con los archivos `.safe` y `.hack` de Flight y Leg en sus respectivas carpetas, guardadas estas en la carpeta `manager`, desde dos PCs diferentes.

Para analizar los resultados, se tratarán los datos tal y como se indican en la clase de teoría.

### 1. Análisis de las graficas con tiempo promedio

Primero se mostrarán las gráficas del tiempo promedio para cada *feature* analizada en los distintos PCs que se han usado para el estudio.

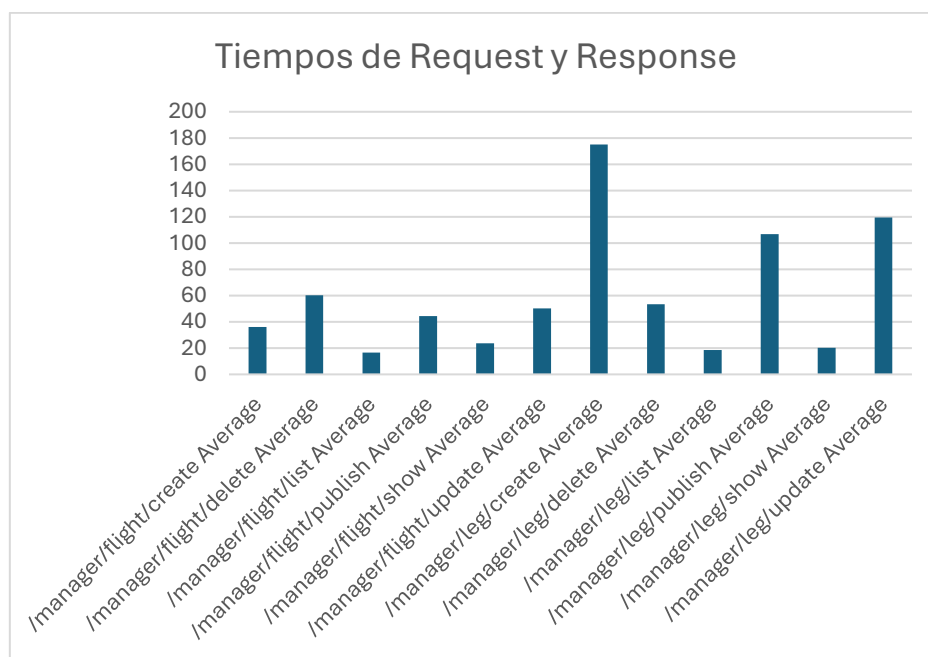


Ilustración 4: Gráfica de tiempo promedio para PC 1



Ilustración 5: Gráfica de tiempo promedio para PC 2

En estas gráficas podemos ver que la *feature* que más tiempo consume en ambos casos es el *create* de *Leg*, siendo el pico más alto en las dos gráficas, ya que, generalmente, sobre todo en las bases de datos relacionales, la insercción de una nueva fila, así como el establecer las nuevas relaciones con otras tablas, consume mas tiempo ya que conlleva mas operaciones que la actualización de un objeto ya existente.

Podemos observar que existe una gran diferencia entre los tiempos de ejecución obtenidos en el PC 1 con los obtenidos en el PC2, siendo los primeros considerablemente más elevados, con una diferencia de, usando la *feature create* de *Leg* como referencia, aproximadamente 130 milisegundos.

## 2. Análisis de los resultados del análisis de estadística descriptiva

Ahora se mostrarán los resultados obtenidos de realizar el estudio mediante la herramienta que proporciona Excel, *Analysis ToolPak*, para obtener las estadísticas descriptivas para cada PC y estudiar el intervalo de confianza del 95%.

Resultados obtenidos en la estadística descriptiva del PC 1:

| <i>Mi PC</i>            |          |               |          |          |
|-------------------------|----------|---------------|----------|----------|
| Mean                    | 56,36977 | Interval (ms) | 46,67763 | 66,0619  |
| Standard Error          | 4,919913 | Interval (s)  | 0,046678 | 0,066062 |
| Median                  | 24,2242  |               |          |          |
| Mode                    | #N/A     |               |          |          |
| Standard Deviation      | 76,06001 |               |          |          |
| Sample Variance         | 5785,126 |               |          |          |
| Kurtosis                | 12,68059 |               |          |          |
| Skewness                | 3,011419 |               |          |          |
| Range                   | 593,7701 |               |          |          |
| Minimum                 | 4,9986   |               |          |          |
| Maximum                 | 598,7687 |               |          |          |
| Sum                     | 13472,37 |               |          |          |
| Count                   | 239      |               |          |          |
| Confidence Level(95,0%) | 9,692138 |               |          |          |

Resultados obtenidos en la estadística descriptiva del PC 2:

| <i>Otro PC</i>          |             |               |          |            |
|-------------------------|-------------|---------------|----------|------------|
| Mean                    | 12,82189079 | Interval (ms) | 10,52708 | 15,1166986 |
| Standard Error          | 1,164887958 | Interval (s)  | 0,010527 | 0,0151167  |
| Median                  | 7,4192      |               |          |            |
| Mode                    | #N/A        |               |          |            |
| Standard Deviation      | 18,0087308  |               |          |            |
| Sample Variance         | 324,314385  |               |          |            |
| Kurtosis                | 20,08088641 |               |          |            |
| Skewness                | 4,097569393 |               |          |            |
| Range                   | 144,4995    |               |          |            |
| Minimum                 | 2,5509      |               |          |            |
| Maximum                 | 147,0504    |               |          |            |
| Sum                     | 3064,4319   |               |          |            |
| Count                   | 239         |               |          |            |
| Confidence Level(95,0%) | 2,294807761 |               |          |            |

Como se puede apreciar, se puede afirmar que tanto en rendimiento como en estabilidad es bastante mejor el PC 2 (Otro PC), viendo que *Median* es más baja, lo que indica un tiempo de ejecución medio menor. Además, se aprecia que la desviación estándar del PC 1 (Mi PC) es considerablemente mayor, lo que indica una gran variabilidad en los tiempos de ejecución y, port tanto, unos resultados menos consistentes.

Por otra parte, fijándonos en el nivel de confianza y, por tanto, en los intervalos de confianza de cada uno, indica que, cada uno de los PCs, si se repitieran las pruebas bastantes veces, el 95% de las veces la media estará dentro de ese rango.

Para el PC 1 (Mi PC), el intervalo es más grande que el del PC 2 (Otro PC), lo que indica una menor precisión a la hora de medir los tiempos. Además, al no superponerse (los valores mínimos y máximos del PC 2 son menores que los valores mínimos y máximos del PC 1) se puede deducir que la media de los tiempos del PC 2 estará con total seguridad por debajo de la media de los tiempos del PC 1.

### 3. Estudio del Z-Test

Para terminar la comparación, vamos a estudiar los resultados obtenidos de aplicar el Z-Test tal y como se indica en las transparencias de teoría.

Resultados obtenidos en el Z-Test para PC 1 (Mi PC) y PC 2 (Otro PC):

z-Test: Two Sample for  
Means

|                                 | <i>Mi PC</i> | <i>Otro PC</i> |
|---------------------------------|--------------|----------------|
| Mean                            | 56,36976611  | 12,82189079    |
| Known Variance                  | 5785,125665  | 324,314385     |
| Observations                    | 239          | 239            |
| Hypothesized Mean<br>Difference | 0            |                |
| z                               | 8,6132136    |                |
| P(Z<=z) one-tail                | 0            |                |
| z Critical one-tail             | 1,644853627  |                |
| P(Z<=z) two-tail                | 0            |                |
| z Critical two-tail             | 1,959963985  |                |

Como se puede apreciar, el valor del campo de P(Z<=z) two-tail es 0, que se encuentra entre el intervalo  $[0.00, \alpha]$ , donde Alpha es 0.05, definido previamente. Esto indica que, sin lugar a duda, se pueden comparar los promedios de los tiempos, y, por ende, se puede afirmar que el PC 2 (Otro PC) es superior en cuanto a rendimiento al PC 1 (Mi PC).

## 8. Conclusiones

Tras este proceso de *testing* y análisis sobre las entidades Leg y Flight, se ha podido ver como se desenvuelve el sistema a la hora de recibir y enviar peticiones, tanto en cobertura de código y, por consecuente, robustez del sistema antes datos inválidos o acciones ilegales, como en rendimiento, comparando los tiempos obtenidos del log generado al ejecutar el `tester-testing#replay`. Se ha visto que, para el mismo código y los mismos archivos `.safe` y `.hack`, se han obtenido resultados considerablemente distantes, lo que nos puede indicar que la diferencia y la eficacia no es cuestión del software desarrollado, si no del hardware de cada ordenador.

Concluyendo, podemos afirmar que se obtuvo una correcta implementación de las *features* y sus respectivas restricciones, así como unos valores de rendimiento aceptable, indicando que realizar *testing* es una parte fundamental a la hora de desarrollar un sistema, ya que nos permite ver errores que no se habían visto en el desarrollo, corregirlos y ver como se comporta el sistema para evaluar si su rendimiento es el esperado, o por el contrario, hay que realizar una refactorización. Esto añade fiabilidad y confianza al producto, aumentándolo así su valor tanto para el cliente como para el propio equipo de desarrollo.

## 9. Bibliografía

Intencionalmente en blanco.