**Testing Report**



Group: C1.025

Repository: <https://github.com/cesmarvan/Acme-ANS-D04.git>

César Martínez Van der Looven (Student#3)

Contenido

[Introducción 3](#_Toc199194439)

[Tests realizados 3](#_Toc199194440)

[FlightAssignment 3](#_Toc199194441)

[ActivityLog 5](#_Toc199194442)

[Test de Rendimiento 7](#_Toc199194443)

[Antes de los índices 7](#_Toc199194444)

[Tras los índices 9](#_Toc199194445)

[Conclusión 12](#_Toc199194446)

[Bibliografía 12](#_Toc199194447)

# Introducción

El propósito de este documento es proveer un análisis de los diferentes tests realizados con respecto al código implementado por el Student #3, así como un análisis del rendimiento del sistema con respecto a los requisitos asociados a dicho estudiante. Para acarrear dicha tarea, se mostrarán diferentes datos (estadísticas, gráficas, etc.) mostrando el rendimiento antes y después de implementar los índices en distintas entidades.

# Tests realizados

## FlightAssignment

* create.safe: dicho test se encarga de comprobar que la funcionalidad *Create* de un FlightAssignment sea la esperada, probando con distintos datos erróneos (como campos vacíos y/o nulos), longitud no válida (para atributos de texto) y las restricciones propias de los requisitos descritos por el profesorado.
* delete.safe : dicho test se encarga de comprobar que la funcionalidad *Delete* funcione correctamente. En este caso, la única restricción acerca de dicha funcionalidad es que el usuario cuente con los permisos necesarios (ser un FlightCrewMember, en este caso) y que el elemento que se quiere eliminar no se encuentre en *DraftMode* (es decir, que no esté publicado aún).
* list-completed.safe/ list-planned.safe: dicho test se encarga de comprobar que el listado de los diferentes *FlightAssignment* se realice correctamente (únicamente podemos observar aquellos que nos pertenezcan, y cada uno muestra los vuelos asignados ocurridos en el pasado y en el futuro, respectivamente).
* publish.safe: dicho test se encarga de comprobar que la funcionalidad *Publish* funcione correctamente. Para esto se ha seguido un procedimiento muy similar que el de la funcionalidad *Create*, probando con diversos datos erróneos y finalmente con unos datos válidos.
* show.safe: dicho test se encarga de comprobar que la funcionalidad *Show* funcione correctamente. En este caso las restricciones consistían en que dicho *FlighAssignment* nos perteneciera a nosotros y no a otro usuario.
* update.safe: dicho test se encarga de comprobar que la funcionalidad *Update* funcione correctamente. Para esto se ha seguid un procedimiento similar al de las funcionalidades *Create* y *Publish*, probando con diversos valores erróneos y finalmente con unos valores correctos para probar el caso positivo.
* create.hack/ delete.hack/ publish.hack/ show.hack/ update.hack: todos estos tests se encargan de comprobar que toda persona que no tenga el rol *FlightCrewMember* no puede acceder a dicha funcionalidad (la del listado se probó en el mismo test que la de show), así como comprobar que el *FlightCrewMember* que estuviera intentando acceder al id del *FlightAssignment* correspondiente fuese dueño de dicha entidad. En todos los casos se obtuvo el caso negativo esperado (error de autenticación).
* navigation-attributes.hack: este test se encarga de comprobar no se permite al usuario crear, publicar o actualizar ningún *FlightAssignment* haciendo uso de las herramientas de desarrollador (F12) y cambiando los valores manualmente. El resultado obtenido nuevamente fue el caso negativo esperado.

La cobertura para los servicios de *FlightAssignment* es del 97.6%:

Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

Vemos que no todas las clases tienen una cobertura del 100%. Vamos a verlas por orden:

* Interfaz de usuario gráfica

  El contenido generado por IA puede ser incorrecto.Publish Service

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

Vemos que las únicas 3 líneas que no llegan a correrse son, por orden, la línea *status=false;*, que nunca correrá a menos que surja alguna excepción que no se haya premeditado.

Las otras 2 líneas que no fueron ejecutadas fueron en el validate. La primera línea casi nunca será ejecutada, pues no puede asignarse un vuelo que ya haya ocurrido, y la segunda se trata del cas en el que ya haya un copiloto asignado al vuelo. Vemos que este caso sí podría ocurrir, pero simplemente no se ha probado. El resto de líneas (en amarillo) son líneas que no se han ejecutad al completo debido a que ya cumplía las condiciones mínimas para poder entrar al código del condicional correspondiente.

* Update Service, Create Service, Delete Service, Show Service: se tratan de los mismos casos que en el publish, aunque ese ligero aumento en la cobertura se trata de la primera validación (la cual no ejecuta), la cual no aparece en el update, show, create ni delete.

## ActivityLog

* create.safe: dicho test se encarga de comprobar que la funcionalidad *Create* de un *ActivityLog* sea la esperada, probando con distintos datos erróneos (como campos vacíos y/o nulos), longitud no válida (para atributos de texto), valores fuera del rango límite (en valores numéricos y de fecha), y las restricciones propias de los requisitos descritos por el profesorado.
* delete.safe : dicho test se encarga de comprobar que la funcionalidad *Delete* funcione correctamente. En este caso, la única restricción acerca de dicha funcionalidad es que el usuario cuente con los permisos necesarios (ser un FlightCrewMember, en este caso) y que el elemento que se quiere eliminar no se encuentre en *DraftMode* (es decir, que no esté publicado aún).
* publish.safe: dicho test se encarga de comprobar que la funcionalidad *Publish* funcione correctamente. Para esto se ha seguido un procedimiento muy similar que el de la funcionalidad *Create*, probando con diversos datos erróneos y finalmente con unos datos válidos.
* show.safe: dicho test se encarga de comprobar que la funcionalidad *Show* funcione correctamente. En este caso las restricciones consistían en que dicho *FlighAssignment* nos perteneciera a nosotros y no a otro usuario.
* update.safe: dicho test se encarga de comprobar que la funcionalidad *Update* funcione correctamente. Para esto se ha seguid un procedimiento similar al de las funcionalidades *Create* y *Publish*, probando con diversos valores erróneos y finalmente con unos valores correctos para probar el caso positivo.
* create.hack/ delete.hack/ publish.hack/ show.hack/ update.hack: todos estos tests se encargan de comprobar que toda persona que no tenga el rol *FlightCrewMember* no puede acceder a dicha funcionalidad (la del listado se probó en el mismo test que la de show), así como comprobar que el *FlightCrewMember* que estuviera intentando acceder al id del *ActivityLog* correspondiente fuese dueño de dicha entidad. En todos los casos se obtuvo el caso negativo esperado (error de autenticación).
* navigation-attributes.hack: este test se encarga de comprobar no se permite al usuario crear, publicar o actualizar ningún *ActivityLog* haciendo uso de las herramientas de desarrollador (F12) y cambiando los valores manualmente. El resultado obtenido nuevamente fue el caso negativo esperado.

La cobertura para los servicios de *FlightAssignment* es del 97.6%:

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Vemos que no todas las clases tienen una cobertura del 100%. En este caso, se trata siempre del mismo caso:

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

La única línea que vemos en color rojo es nuevamente la de la validación, la cual debe saltar en caso de cualquier otro caso no premeditado.

# Test de Rendimiento

Se muestra a continuación los resultados del Test de Rendimiento realizados anteriormente. En este caso, las diferencias en los tiempos puede observarse antes y después de añadir índices a distintas entidades (a *ActivityLog* y a *FlightAssignment*).

## Antes de los índices

Fichero “tester-performance-graphic-after.xlsx”. Contiene el gráfico de rendimiento ANTES de los índices.

Como podemos observar, las operaciones que más tiempo consumieron fueron el *Publish* de *Flight Assignment* y el *Create* de *ActivityLog*. Este resultado no nos toma por sorpresa, ya que estas operaciones tienen una carga superior en cuanto a complejidad computacional.

Las estadísticas obtenidas anterior a la implementación de los índices son:

Aplicación, Tabla, Excel

El contenido generado por IA puede ser incorrecto.Documento “tester-performance-times-before.xlsx”

Observamos un nivel de confianza (95%) fluctúa entre los 6.068ms hasta los 7.664ms.

## Tras los índices

Fichero “tester-performance-graphic-after.xslx”

Como podemos observar, las operaciones que más tiempo consumieron fueron el *Publish* de *Flight Assignment* y el *Create* de *ActivityLog* (resultado prácticamente idéntico a antes de los índices). Este resultado no nos toma por sorpresa, ya que estas operaciones tienen una carga superior en cuanto a complejidad computacional.

Documento “tester-performance-times-after.xlsx”

Tabla

El contenido generado por IA puede ser incorrecto.

Observamos un nivel de confianza (95%) fluctúa entre los 6.332ms hasta los 7.980ms.

A partir de los dos intervalos obtenidos, se llevará a cabo un z-test, el cual encontramos en el archivo “z-test.xlsx”

Interfaz de usuario gráfica, Aplicación, Tabla

El contenido generado por IA puede ser incorrecto.

Vemos que el valor crítico de z (dos colas) es 0.619, el cual es muy superior al valor de 1-α (siendo α=0.95), vemos que realmente no hemos mejorado en absoluto el rendimiento de las consultas y, por ende, de los servicios y la aplicación en general.

# Conclusión

En conclusión, podemos resumir en que el 97.6% de las líneas de código correspondientes al Student #3 han sido ejecutadas y testeadas, haciendo posible una detección y corrección de bugs en la aplicación. Especialmente han sido útiles los tests de POST y GET hacking (la cual, en mi caso, fue la primera toma de contacto con el mundo del hacking), así como la minuciosa validación de diferentes datos durante los tests, lo cual ha demostrado que hacer una aplicación consistente es mucho más complejo de lo que uno podría pensar inicialmente. Además, se ha demostrado estadísticamente el rendimiento de la aplicación, no superando los 7.98ms de media en un equipo potente, aunque no de última generación.

# Bibliografía

Toda la bibliografía necesaria para realizar este reporte ha sido proporcionada por el profesorado en la Plataforma Virtual de la Universidad de Sevilla.