



TESTING REPORT

[STUDENT 1] Grupo C2.052

<https://github.com/BVP2455/Acme-ANS-C2>

José Luis Cegri Marcos

joscegmar@alum.us.es

Tabla de contenidos

1. Pruebas funcionales
2. Pruebas de rendimiento

Resumen ejecutivo

En este Testing Report, presentamos las pruebas realizadas para el Student 1, tanto funcionales como de rendimiento, para las funcionalidades de vuelos (Flight) y tramos (Leg) correspondientes a las tareas del gestor de vuelos (Manager) del proyecto Acme-ANS-D04.

Tabla de revisión

Revisión	Fecha	Descripción corta
V1.0	26/05/2025	Primera versión del proyecto con los datos completados
V2.0	01/07/2025	Segunda versión del proyecto corrigiendo los errores de la anterior convocatoria y modificando los cambios. Se ha vuelto a realizar las pruebas de rendimiento debido a un cambio en los índices.

Índice

1. Pruebas funcionales.....	4
1.1 Introducción.....	4
1.2 Contenido.....	6
1.2.1 Cobertura de las funcionalidades de los vuelos (Flight).....	6
1.2.1.1 FlightListService.....	6
1.2.1.2 FlightShowService	7
1.2.1.3 FlightCreateService.....	8
1.2.1.4 FlightUpdateService	9
1.2.1.5 FlightDeleteService	10
1.2.1.6 FlightPublishService.....	11
1.2.2 Cobertura de las funcionalidades de los tramos (Leg).....	12
1.2.2.1 LegListService.....	12
1.2.2.2 LegShowService	13
1.2.2.3 LegCreateService	14
1.2.2.4 LegDeleteService	16
1.2.2.5 LegUpdateService.....	17
1.2.2.6 LegPublishService	19
2. Pruebas de rendimiento	21
3. Conclusión	24

1. Pruebas funcionales

1.1 Introducción







En las pruebas funcionales, hemos utilizado la herramienta record que nos proporciona el framework Acme-Framework-25.6.0 para probar cada una de las funcionalidades de Leg y Flight. La herramienta carga el proyecto en la dirección <http://localhost:8082/Acme-ANS-C2?debug=true&locale=en> y registra cada una de las peticiones que realizamos al servidor.

Aprovechando esto, hemos realizado un testing exhaustivo a cada una de las funcionalidades que se implementan en Leg y Flight: listado de vuelos o tramos (FlightListService.java y LegListService.java), mostrado de vuelos o tramos (FlightShowService.java y LegShowService.java), creación de vuelos o tramos (FlightCreateService.java y LegCreateService.java), borrado de vuelos o tramos (FlightDeleteService.java y LegDeleteService.java), actualización de vuelos o tramos (FlightUpdateService.java y LegUpdateService.java) y la publicación de estos (FlightPublishService.java y LegPublishService.java), que impedirían modificaciones y se mostrarían para el resto de usuarios que utilicen el servicio.





Las pruebas formales a las funcionalidades que han probado los casos positivos (que devolvían el resultado esperado) y negativos (que devolvían un error) se almacenan en ficheros de extensión “.safe” (ver figura 1). Las pruebas formales a las funcionalidades que han probado casos de GET y POST hacking (se ven más tipos a lo largo de la asignatura, pero son cubiertos por el framework) se almacenan en ficheros de extensión “.hack” (ver figura 2). Una vez realizadas todas las pruebas a las funcionalidades, utilizamos la herramienta replay proporcionada por el framework, que ejecutaba todos los ficheros de pruebas y, en caso de no devolver el resultado esperado devolvía un error. Una vez terminada la ejecución, nos devolvía la cobertura de las funcionalidades que se probaban en los ficheros.

Las pruebas han dado resultado a una cobertura del 100,00% tanto para las funcionalidades de Flight como para las funcionalidades de Leg (véase figura 3 y 4)

















A continuación, en la sección de contenidos mostraremos la cobertura de cada fichero, centrándonos principalmente en las líneas subrayadas en amarillo, que indican que ha habido condiciones dentro de estructuras de bucles o de condición que no se han cumplido en todas sus posibles ramas durante la ejecución de las pruebas.

 create.safe	26/05/2025 13:52	Archivo SAFE	120 KB
 delete.safe	26/05/2025 13:52	Archivo SAFE	39 KB
 list.safe	26/05/2025 13:52	Archivo SAFE	25 KB
 publish.safe	26/05/2025 13:52	Archivo SAFE	291 KB
 show.safe	26/05/2025 13:52	Archivo SAFE	60 KB
 update.safe	26/05/2025 13:52	Archivo SAFE	313 KB

















1. Archivos .safe

 delete.hack	26/05/2025 13:52	Archivo HACK	32 KB
 publish.hack	26/05/2025 13:52	Archivo HACK	28 KB
 show.hack	26/05/2025 13:52	Archivo HACK	21 KB
 update.hack	26/05/2025 13:52	Archivo HACK	23 KB

2. Archivos .hack

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
▼  acme.features.manager.flight	 100,0 %	761	0	761
>  FlightController.java	 100,0 %	35	0	35
>  FlightCreateService.java	 100,0 %	99	0	99
>  FlightDeleteService.java	 100,0 %	124	0	124
>  FlightListService.java	 100,0 %	71	0	71
>  FlightPublishService.java	 100,0 %	183	0	183
>  FlightShowService.java	 100,0 %	99	0	99
>  FlightUpdateService.java	 100,0 %	150	0	150

3. Cobertura de la funcionalidad de vuelos

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
▼  acme.features.manager.leg	 100,0 %	1.949	0	1.949
>  LegController.java	 100,0 %	35	0	35
>  LegCreateService.java	 100,0 %	435	0	435
>  LegDeleteService.java	 100,0 %	182	0	182
>  LegListService.java	 100,0 %	143	0	143
>  LegPublishService.java	 100,0 %	492	0	492
>  LegShowService.java	 100,0 %	224	0	224
>  LegUpdateService.java	 100,0 %	438	0	438

4. Cobertura de la funcionalidad de tramos

1.2 Contenido

Dividiremos el contenido en dos partes: primero explicaremos la cobertura de las funcionalidades de los vuelos (Flight), y después terminaremos con las explicaciones de la cobertura de las funcionalidades de los tramos (Leg).

1.2.1 Cobertura de las funcionalidades de los vuelos (Flight)

1.2.1.1 FlightListService

Para probar la funcionalidad inicio sesión con el manager1 y voy pasando por los índices. Cierro sesión, inicio sesión como administrador y apago el sistema mediante shut system down. La cobertura final es del 100%.

```
package acme.features.manager.flight;

import java.util.Collection;

import org.springframework.beans.factory.annotation.Autowired;

import acme.client.components.models.Dataset;
import acme.client.services.AbstractGuiService;
import acme.client.services.GuiService;
import acme.entities.flight.Flight;
import acme.realms.manager.Manager;

@GuiService
public class FlightListService extends AbstractGuiService<Manager, Flight> {

    // Internal state -----

    @Autowired
    private FlightRepository repository;

    //AbstractGuiService interface -----

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

    @Override
    public void load() {
        Collection<Flight> flights;
        Manager manager;
        int airlineId;

        manager = (Manager) super.getRequest().getPrincipal().getActiveRealm();
        airlineId = manager.getAirline().getId();
        flights = this.repository.findFlightsByAirlineId(airlineId);

        super.getBuffer().addData(flights);
    }

    @Override
    public void unbind(final Flight flight) {
        Dataset dataset;

        dataset = super.unbindObject(flight, "tag", "description", "selfTransfer", "draftMode");
        dataset.put("layovers", flight.getNumberLayovers());
        super.addPayload(dataset, flight, "cost");

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

1.2.1.2 FlightShowService

- show.safe: Inicio sesión como manager1, y visualizo un vuelo normal, otro con el atributo selfTransfer en true, y un último con el atributo draftMode en true (no publicado). Después inicio sesión como administrador y apago el sistema (shut system down).
- show.hack: Desde manager3 y sin estar registrado, pruebo a hacer un GET hacking de un flight que tiene el atributo draftMode a true, desde ambos usuarios salta el authorise.

En la línea 34 parte de bloque condicional no es recorrido (se recorren 3 de las 4 ramas). Esto es debido a que no hemos utilizado el show en un vuelo publicado que no nos pertenece. La cobertura final es del 100%.

```
package acme.features.manager.flight;

import org.springframework.beans.factory.annotation.Autowired;

import acme.client.components.models.Dataset;
import acme.client.services.AbstractGuiService;
import acme.client.services.GuiService;
import acme.entities.flight.Flight;
import acme.realms.manager.Manager;

@GuiService
public class FlightShowService extends AbstractGuiService<Manager, Flight> {

    // Internal state -----

    @Autowired
    private FlightRepository repository;

    // AbstractGuiService interface -----

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

        flightId = super.getRequest().getData("id", int.class);
        flight = this.repository.getFlightById(flightId);
        manager = (Manager) super.getRequest().getPrincipal().getActiveRealm();

        if (manager.getAirline().getId() == flight.getAirline().getId() || !flight.getDraftMode())
            authorise = true;
        super.getResponse().setAuthorised(authorise);
    }

    @Override
    public void load() {
        int flightId;
        Flight flight;

        flightId = super.getRequest().getData("id", int.class);
        flight = this.repository.getFlightById(flightId);

        super.getResponse().addGlobal("flightDraftMode", flight.getDraftMode());

        super.getBuffer().addData(flight);
    }

    @Override
    public void unbind(final Flight flight) {
        Dataset dataset;

        dataset = super.unbindObject(flight, "tag", "selfTransfer", "cost", "description");
        dataset.put("confirmation", false);

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

1.2.1.3 FlightCreateService

Pruebo con cada uno de los atributos del flight: límite inferior - 1, límite inferior, límite inferior + 1, límite superior - 1, límite superior, límite superior + 1. En el caso de los atributos de texto, también he probado con caracteres no latinos e injections. La cobertura final es del 100%.

```
@GuiService
public class FlightCreateService extends AbstractGuiService<Manager, Flight> {
    // Internal state -----
    @Autowired
    private FlightRepository repository;
    // AbstractGuiService interface -----

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

    @Override
    public void load() {
        Flight flight;
        Manager manager;
        Airline airline;

        manager = (Manager) super.getRequest().getPrincipal().getActiveRealm();
        airline = manager.getAirline();

        flight = new Flight();
        flight.setDraftMode(true);
        flight.setAirline(airline);

        super.getBuffer().addData(flight);
    }

    @Override
    public void bind(final Flight flight) {
        super.bindObject(flight, "tag", "selfTransfer", "cost", "description");
    }

    @Override
    public void validate(final Flight flight) {
        boolean confirmation;
        confirmation = super.getRequest().getData("confirmation", boolean.class);
        super.state(confirmation, "confirmation", "acme.validation.confirmation.message");
    }

    @Override
    public void perform(final Flight flight) {
        this.repository.save(flight);
    }

    @Override
    public void unbind(final Flight flight) {
        Dataset dataset;

        dataset = super.unbindObject(flight, "tag", "selfTransfer", "cost", "description", "draftMode");
        dataset.put("confirmation", false);

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


1.2.1.4 FlightUpdateService

- update.safe: Pruebo con cada uno de los atributos del flight: límite inferior - 1, límite inferior, límite inferior + 1, límite superior - 1, límite superior, límite superior + 1. En el caso de los atributos de texto, también he probado con caracteres no latinos e injections. Además, pruebo la restricción de que el atributo selftransfer no puede ser false si los aeropuertos en tramos publicados no son consecutivos.
- update.hack: Desde manager3 y sin estar registrado he intentado un GET hacking de update tanto de un vuelo publicado como un vuelo que no lo está

En la línea 41, parte del bloque condicional no es recorrido (se ejecutan 3 de las 4 ramas posibles). Esto se debe a que no se ha probado el caso de un vuelo en modo borrador que no pertenece al manager. La cobertura final es del 100%.

```
package acme.features.manager.flight;

import org.springframework.beans.factory.annotation.Autowired;

import acme.client.components.models.Dataset;
import acme.client.services.AbstractGuiService;
import acme.client.services.GuiService;
import acme.entities.flight.Flight;
import acme.realms.manager.Manager;

@GuiService
public class FlightUpdateService extends AbstractGuiService<Manager, Flight> {

    // Internal state -----

    @Autowired
    private FlightRepository repository;

    // AbstractGuiService interface -----

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

        flightId = super.getRequest().getData("id", int.class);
        flight = this.repository.getFlightById(flightId);
        manager = (Manager) super.getRequest().getPrincipal().getActiveRealm();

        if (manager.getAirline().getId() == flight.getAirline().getId() && flight.getDraftMode())
            authorise = true;
        super.getResponse().setAuthorised(authorise);
    }

    @Override
    public void load() {
        int flightId;
        Flight flight;

        flightId = super.getRequest().getData("id", int.class);
        flight = (Flight) this.repository.findById(flightId).get();

        super.getBuffer().addData(flight);

        boolean draftMode = flight.getDraftMode();
        super.getResponse().addGlobal("flightDraftMode", draftMode);
    }

    @Override
    public void bind(final Flight flight) {
        super.bindObject(flight, "tag", "selfTransfer", "cost", "description");
    }

    @Override
    public void validate(final Flight flight) {
        boolean isDraftMode;

        isDraftMode = flight.getDraftMode();

        super.state(isDraftMode, "=", "acme.validation.flight.draftMode.updated.message");
    }

    @Override
    public void perform(final Flight flight) {
        this.repository.save(flight);
    }

    @Override
    public void unbind(final Flight flight) {
        Dataset dataset;

        dataset = super.unbindObject(flight, "tag", "selfTransfer", "cost", "description", "draftMode");
        dataset.put("confirmation", false);

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

1.2.1.5 FlightDeleteService

- delete.safe: He borrado un vuelo no publicado
- delete.hack: Desde manager3 y sin estar publicado he intentado un GET hacking de delete tanto para un vuelo publicado como el que no. En ambos casos salta el authorise.

En la línea 40, se han recorrido 3 de las 4 ramas posibles. La cobertura es del 100%

```
@GuiService
public class FlightDeleteService extends AbstractGuiService<Manager, Flight> {

    // Internal state -----
    @Autowired
    private FlightRepository repository;

    @Autowired
    private LegRepository legRepository;

    // AbstractGuiService interface -----

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

        flightId = super.getRequest().getData("id", int.class);
        flight = this.repository.getFlightById(flightId);
        manager = (Manager) super.getRequest().getPrincipal().getActiveRealm();

        if (manager.getAirline().getId() == flight.getAirline().getId() && flight.getDraftMode())
            authorise = true;
        super.getResponse().setAuthorised(authorise);
    }

    @Override
    public void load() {
        int flightId;
        Flight flight;

        flightId = super.getRequest().getData("id", int.class);
        flight = (Flight) this.repository.findById(flightId).get();

        super.getBuffer().addData(flight);

        boolean draftMode = flight.getDraftMode();
        super.getResponse().addGlobal("flightDraftMode", draftMode);
    }

    @Override
    public void bind(final Flight flight) {
        super.bindObject(flight, "tag", "selftransfer", "cost", "description");
    }

    @Override
    public void validate(final Flight flight) {
        boolean isDraftMode = flight.getDraftMode();
        boolean status = isDraftMode == true;

        super.state(status, "", "acme.validation.flight.draftMode.deleted.message");
    }

    @Override
    public void perform(final Flight flight) {
        Collection<Leg> legs = this.legRepository.findLegsByFlightId(flight.getId());
        this.legRepository.deleteAll(legs);
        this.repository.delete(flight);
    }
}
```

1.2.1.6 FlightPublishService

- **publish.safe:** Desde manager1 cojo el vuelo con draftMode en true, intento publicar con todos los campos vacíos, luego pruebo uno por uno los límites inferiores y superiores, incluyendo los caracteres no latinos y las injections en los campos de texto. Una vez probados todos, verifico las restricciones con los campos base.
- **publish.hack:** Desde manager3 y sin estar registrado intento un GET hacking de publish tanto para un flight publicado como el que no. En ambos salta el authorise.

En la línea 41, parte del bloque condicional no es recorrido (se ejecutan 3 de las 4 ramas posibles). Esto se debe a que no se ha probado el caso de un vuelo en modo borrador que no pertenece al manager. La cobertura final es del 100%.

```
package acce.features.manager.flight;

import java.util.Collection;

import org.springframework.beans.factory.annotation.Autowired;

import acce.client.components.models.Dataset;
import acce.client.services.AbstractUIService;
import acce.client.services.GuiServices;
import acce.entities.flight.Flight;
import acce.entities.leg.Leg;
import acce.features.manager.leg.LegRepository;
import acce.features.manager.Manager;

@UIService
public class FlightPublishService extends AbstractUIService<Manager, Flight> {

    // Internal state -----

    @Autowired
    private FlightRepository repository;

    @Autowired
    private LegRepository legRepository;

    // AbstractUIService interface -----

    @Override
    public void authorize() {
        int flightId;
        Flight flight;
        Manager manager;
        boolean authorize = false;

        flightId = super.getRequest().getData("id", int.class);
        flight = this.repository.getFlightById(flightId);
        manager = (Manager) super.getRequest().getPrincipal().getActiveRole();

        if (manager.getAirline().getId() == flight.getAirline().getId() && flight.getDraftMode())
            authorize = true;
        super.getResponse().setAuthorized(authorize);
    }

    @Override
    public void load() {
        int flightId;
        Flight flight;

        flightId = super.getRequest().getData("id", int.class);
        flight = (Flight) this.repository.findById(flightId).get();

        super.getBuffer().addData(flight);

        boolean draftMode = flight.getDraftMode();
        super.getResponse().addGlobal("flightDraftMode", draftMode);
    }

    @Override
    public void bind(final Flight flight) {
        super.bindObject(flight, "tag", "selftransfer", "cost", "description");
    }

    @Override
    public void validate(final Flight flight) {
        Collection<Leg> legs = this.repository.findLegsByFlightId(flight.getId());
        boolean haslegs = !legs.isEmpty();

        // R1: No dejar publicar si no hay tramos
        super.state(haslegs, "", "acce.validation.flight.no-legs.message");

        // R2: No dejar publicar si algún tramo está en borrador
        boolean alllegsPublished = legs.stream().allMatch(leg -> !leg.getDraftMode());
        super.state(alllegsPublished, "", "acce.validation.flight.legs-not-published.message");
    }

    @Override
    public void perform(final Flight flight) {
        flight.setDraftMode(false);
        this.repository.save(flight);
    }

    @Override
    public void unbind(final Flight flight) {
        Dataset dataset;

        dataset = super.unbindObject(flight, "tag", "selftransfer", "cost", "description", "draftMode");
        dataset.put("confirmation", false);

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

1.2.2 Cobertura de las funcionalidades de los tramos (Leg)

1.2.2.1 LegListService

- list.safe: Inicio sesión con manager1 y miro la lista de legs de un vuelo publicado y de un vuelo con draftMode en true y luego miro la lista de legs de un vuelo publicado que no me pertenece
- list.hack: Inicio sesión con manager3 e intento mirar la lista de legs de un vuelo que no está publicado, me salta el authorise La cobertura final es del 100%.

```
@GuiService
public class LegListService extends AbstractGuiServiceManager, Leg {

    @Autowired
    private LegRepository repository;

    @Autowired
    private FlightRepository flightRepository;

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

        flightId = super.getRequest().getData("flightId", int.class);
        flight = this.flightRepository.getFlightById(flightId);
        manager = (Manager) super.getRequest().getPrincipal().getActiveRealm();

        if (manager.getAirline().getId() == flight.getAirline().getId() || flight.getDraftMode())
            authorise = true;
        super.getResponse().setAuthorised(authorise);
    }

    @Override
    public void load() {
        int flightId = super.getRequest().getData("flightId", int.class);
        Flight flight = (Flight) this.flightRepository.findById(flightId).get();
        Boolean flightDraftMode = flight.getDraftMode();

        Collection<Leg> legs = this.repository.findLegsByFlightId(flightId);
        super.getBuffer().addData(legs);
        super.getResponse().addGlobal("flightDraftMode", flightDraftMode);
        super.getResponse().addGlobal("flightId", flightId);
    }

    @Override
    public void unbind(final Leg leg) {
        Dataset dataset = super.unbindObject(leg, "flightNumber", "scheduledDeparture", "scheduledArrival", "status", "draftMode");
        dataset.put("airportDeparture", leg.getDepartureAirport().getCity());
        dataset.put("airportArrival", leg.getArrivalAirport().getCity());
        dataset.put("aircraft", leg.getAircraft().getModel());
        dataset.put("flightId", super.getRequest().getData("flightId", int.class));
        super.getResponse().addData(dataset);
    }
}
```

1.2.2.2 LegShowService

- show.safe: Inicio sesión con manager1 y miro un leg publicado y uno que no lo está. Después, miro un leg publicado por otro manager
- show.hack: Inicio sesión con manager3 e intento mirar un leg que no me pertenece y no está publicado, me salta el authorise.

La cobertura es del 100%.

```
@GuiService
public class LegShowService extends AbstractGuiService<Manager, Leg> {

    @Autowired
    private LegRepository repository;

    @Autowired
    private FlightRepository flightRepository;

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

        legId = super.getRequest().getData("id", int.class);
        leg = (Leg) this.repository.findById(legId).get();
        manager = (Manager) super.getRequest().getPrincipal().getActiveRealm();

        if (manager.getAirline().getId() == leg.getFlight().getAirline().getId() || !leg.getFlight().getDraftMode())
            authorise = true;
        super.getResponse().setAuthorised(authorise);
    }

    @Override
    public void load() {
        Leg leg;
        int id;

        id = super.getRequest().getData("id", int.class);
        leg = (Leg) this.repository.findById(id).get();

        Flight flight = (Flight) this.flightRepository.findById(leg.getFlight().getId()).get();
        boolean flightDraftMode = flight.getDraftMode();
        boolean legDraftMode = leg.getDraftMode();
        super.getResponse().addGlobal("flightDraftMode", flightDraftMode);
        super.getResponse().addGlobal("legDraftMode", legDraftMode);

        super.getBuffer().addData(leg);
    }

    @Override
    public void unbind(final Leg leg) {
        SelectChoices statusChoices;
        SelectChoices aircraftChoices;
        SelectChoices departureChoices;
        SelectChoices arrivalChoices;
        Dataset dataset;
        Manager manager;
        int airlineId;

        manager = (Manager) super.getRequest().getPrincipal().getActiveRealm();
        airlineId = manager.getAirline().getId();

        Collection<Aircraft> aircrafts = this.repository.findAircraftsByAirlineId(airlineId);
        // Si el tram lo está visitando un manager al que no le pertenece, se añaden las aeronaves de esa aerolinea
        if (airlineId != leg.getFlight().getAirline().getId())
            aircrafts.addAll(this.repository.findAircraftsByAirlineId(leg.getFlight().getAirline().getId()));
        Collection<Airport> airports = this.repository.findAllAirports();

        statusChoices = SelectChoices.from(leg.getStatus.class, leg.getStatus());
        aircraftChoices = SelectChoices.from(aircrafts, "registrationNumber", leg.getAircraft());
        departureChoices = SelectChoices.from(airports, "iataCode", leg.getDepartureAirport());
        arrivalChoices = SelectChoices.from(airports, "iataCode", leg.getArrivalAirport());

        dataset = super.unbindObject(leg, "flightNumber", "scheduledDeparture", "scheduledArrival", "status");
        dataset.put("statuses", statusChoices);
        dataset.put("aircraft", aircraftChoices.getSelected().getKey());
        dataset.put("aircrafts", aircraftChoices);
        dataset.put("airportDeparture", departureChoices.getSelected().getKey());
        dataset.put("airportDepartures", departureChoices);
        dataset.put("airportArrival", arrivalChoices.getSelected().getKey());
        dataset.put("airportArrivals", arrivalChoices);

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

1.2.2.3 LegCreateService

- create.safe: Inicio sesión con manager1 y con un vuelo con draftMode en true y pruebo con los campos vacíos, cada uno de los atributos del leg: límite inferior - 1, límite inferior, límite inferior + 1, límite superior - 1, límite superior, límite superior + 1. En cada campo departureAirport y arrivalAirport he probado con todos los aeropuertos, y en el campo aircraft he probado con todas las aeronaves disponibles para ese manager. Además, he probado todas las restricciones de la creación de legs.
- create2.safe: Olvidé probar la restricción de no permitir que el aeropuerto de llegada y de salida sea el mismo.
- create.hack: Inicio sesión con manager1 y mediante POST hacking pruebo a añadirle valor -1, 999 a cada uno de los campos desplegables. Además, para el campo desplegable aircraft, pruebo también con el id de algún aircraft que no pertenezca al manager.
- create.hack: Inicio sesión con manager1 y mediante GET hacking pruebo a intentar crear un leg en un vuelo que ya está publicado, salta el authorise

En la línea 42, parte del bloque condicional no se recorre (se ejecutan 3 de las 4 ramas), ya que no se ha probado el caso en el que el vuelo no pertenece al manager, por lo que la condición no se evalúa como false.

```

package acme.features.manager;

import java.util.Collection;

import org.springframework.beans.factory.annotation.Autowired;

import acme.client.components.models.Dataset;
import acme.client.components.views.SelectChoices;
import acme.client.helpers.NotFoundHelper;
import acme.client.services.AbstractUserService;
import acme.client.services.UserService;
import acme.entities.aircraft;
import acme.entities.airport;
import acme.entities.flight;
import acme.entities.log;
import acme.entities.log.logStatus;
import acme.features.manager.flight;
import acme.reales.manager;

@UserService
public class LegCreateService extends AbstractUserService<Manager, Leg> {

    @Autowired
    private LegRepository repository;

    @Autowired
    private FlightRepository flightRepository;

    @Override
    public void authorize() {
        int flightId;
        flight flight;
        Manager manager;
        boolean authorize = false;

        flightId = super.getRequest().getData("flightId", int.class);
        flight = (flight) this.flightRepository.findById(flightId).get();
        manager = (Manager) super.getRequest().getPrincipal().getActiveRealm();

        if (manager.getAirline().getId() == flight.getAirline().getId() && flight.getCraftId() != null)
            authorize = true;

        if (authorize) {
            String method;
            int arrivalAirportId, departureAirportId, aircraftId, airlineId;
            Aircraft aircraft;
            Airport arrivalAirport;
            Airport departureAirport;

            method = super.getRequest().getMethod();
            if (method.equals("GET"))
                authorize = true;
            else {
                airlineId = manager.getAirline().getId();
                aircraftId = super.getRequest().getData("aircraft", int.class);
                arrivalAirportId = super.getRequest().getData("airportArrival", int.class);
                departureAirportId = super.getRequest().getData("airportDeparture", int.class);
                aircraft = this.repository.findAircraftById(airlineId, aircraftId);
                arrivalAirport = this.repository.findAirportById(arrivalAirportId);
                departureAirport = this.repository.findAirportById(departureAirportId);
                authorize = (aircraftId != 0 || aircraft != null) && (arrivalAirportId != 0 || arrivalAirport != null) && (departureAirportId != 0 || departureAirport != null);
            }
        }

        super.getResponse().setAuthorized(authorize);
    }

    @Override
    public void load() {
        Leg leg = new Leg();
        int flightId = super.getRequest().getData("flightId", int.class);
        flight flight = (flight) this.flightRepository.findById(flightId).get();

        leg.setFlight(flight);
        leg.setCraftId(true);
        super.getBuffer().addData(leg);
    }

    @Override
    public void bind(final Leg leg) {
        int aircraftId = super.getRequest().getData("aircraft", int.class);
        int departureAirportId = super.getRequest().getData("airportDeparture", int.class);
        int arrivalAirportId = super.getRequest().getData("airportArrival", int.class);

        Aircraft aircraft = this.repository.findAircraftById(aircraftId);
        Airport departure = this.repository.findAirportById(departureAirportId);
        Airport arrival = this.repository.findAirportById(arrivalAirportId);

        leg.setAircraft(aircraft);
        leg.setDepartureAirport(departure);
        leg.setArrivalAirport(arrival);

        super.bindObject(leg, "flightNumber", "scheduledDeparture", "scheduledArrival", "status");
    }

    @Override
    public void validate(final Leg leg) {
        // R1: momento de salida y de llegada deben ser posterior a la fecha actual
        if (leg.getScheduledDeparture() != null) {
            boolean futureDepartureDate = NotFoundHelper.isFuture(leg.getScheduledDeparture());
            super.state(futureDepartureDate, "scheduledDeparture", "ace.validation.leg.scheduled-departure-not-future.message");
        }
        if (leg.getScheduledArrival() != null) {
            boolean futureArrivalDate = NotFoundHelper.isFuture(leg.getScheduledArrival());
            super.state(futureArrivalDate, "scheduledArrival", "ace.validation.leg.scheduled-arrival-not-future.message");
        }
        // R2: no puede existir otro leg con el mismo flight number
        String flightNumber = leg.getFlightNumber();
        boolean coincide = this.repository.existsByFlightNumber(flightNumber);
        super.state(coincide, "flightNumber", "ace.validation.leg.duplicate-code.message");
        // R3: no puede ser el mismo aeropuerto de llegada que el de salida
        if (leg.getArrivalAirport() != null && leg.getDepartureAirport() != null) {
            boolean isDifferent = !leg.getArrivalAirport().equals(leg.getDepartureAirport());
            super.state(isDifferent, "airportArrival", "ace.validation.leg.same-airports.message");
        }
        // R4: los aeropuertos de llegada y salida no pueden ser nulos
        if (leg.getDepartureAirport() == null)
            super.state(false, "airportDeparture", "ace.validation.leg.departure-airport-not-null.message");
        if (leg.getArrivalAirport() == null)
            super.state(false, "airportArrival", "ace.validation.leg.arrival-airport-not-null.message");
    }

    @Override
    public void perform(final Leg leg) {
        this.repository.save(leg);
    }

    @Override
    public void unbind(final Leg leg) {
        Manager manager = (Manager) super.getRequest().getPrincipal().getActiveRealm();
        int airlineId = manager.getAirline().getId();

        Collection<Aircraft> aircrafts = this.repository.findAircraftsByAirlineId(airlineId);
        Collection<Airport> airports = this.repository.findAllAirports();

        SelectChoices statusChoices = SelectChoices.from(logStatus.class, leg.getStatus());
        SelectChoices aircraftChoices = SelectChoices.from(aircrafts, "registrationNumber", leg.getAircraft());
        SelectChoices departureChoices = SelectChoices.from(airports, "iataCode", leg.getDepartureAirport());
        SelectChoices arrivalChoices = SelectChoices.from(airports, "iataCode", leg.getArrivalAirport());

        Dataset dataset = super.unbindObject(leg, "flightNumber", "scheduledDeparture", "scheduledArrival", "status");
        dataset.put("status", statusChoices);
        dataset.put("aircraft", aircraftChoices.getSelected().getKey());
        dataset.put("airportDeparture", departureChoices.getSelected().getKey());
        dataset.put("airportArrival", arrivalChoices.getSelected().getKey());
        dataset.put("airportArrivals", arrivalChoices.getSelected().getKey());
        dataset.put("flightId", leg.getFlight().getId());

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

```

La cobertura final es del 100%, ya que aunque no se evalúan todas las ramas, todas las líneas de código se ejecutan.

1.2.2.4 LegDeleteService

- delete.safe: desde manager1, he borrado un leg no publicado que me pertenecía
- delete.hack: inicio sesión con manager3 y he intentado un GET hacking de delete tanto para un leg publicado como para el que no. En ambos salta el authorise.

En la línea 36, parte del bloque condicional no es recorrido (se recorren 3 de las 4 ramas posibles). Esto se debe a que no se ha probado el caso en el que el leg pertenece a una aerolínea distinta a la del manager y, al mismo tiempo, está en modo borrador. La cobertura es del 100%.

```
@GuiService
public class LegDeleteService extends AbstractGuiService<Manager, Leg> {

    @Autowired
    private LegRepository repository;

    @Autowired
    private FlightRepository flightRepository;

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

        legId = super.getRequest().getData("id", int.class);
        leg = (Leg) this.repository.findById(legId).get();
        manager = (Manager) super.getRequest().getPrincipal().getActiveRealm();

        if (manager.getAirline().getId() == leg.getFlight().getAirline().getId() && leg.getDraftMode())
            authorise = true;
        super.getResponse().setAuthorised(authorise);
    }

    @Override
    public void load() {
        int id = super.getRequest().getData("id", int.class);
        Leg leg = (Leg) this.repository.findById(id).get();
        super.getBuffer().addData(leg);
        Flight flight = (Flight) this.flightRepository.findById(leg.getFlight().getId()).get();
        boolean draftMode = flight.getDraftMode();
        super.getResponse().addGlobal("flightDraftMode", draftMode);
        super.getResponse().addGlobal("legDraftMode", leg.getDraftMode());
    }

    @Override
    public void bind(final Leg leg) {
        int aircraftId = super.getRequest().getData("aircraft", int.class);
        int departureAirportId = super.getRequest().getData("airportDeparture", int.class);
        int arrivalAirportId = super.getRequest().getData("airportArrival", int.class);

        Aircraft aircraft = this.repository.findAircraftByAircraftId(aircraftId);
        Airport departure = this.repository.findAirportByAirportId(departureAirportId);
        Airport arrival = this.repository.findAirportByAirportId(arrivalAirportId);

        leg.setAircraft(aircraft);
        leg.setDepartureAirport(departure);
        leg.setArrivalAirport(arrival);

        super.bindObject(leg, "flightNumber", "scheduledDeparture", "scheduledArrival", "status");
    }

    @Override
    public void validate(final Leg leg) {
        boolean isDraftMode = leg.getDraftMode();

        boolean status = isDraftMode == true;

        super.state(status, "", "acme.validation.leg.draftMode.deleted.message");
    }

    @Override
    public void perform(final Leg leg) {
        this.repository.delete(leg);
    }
}
```


1.2.2.5 LegUpdateService

- update.safe: inicio sesión como manager1 y con un leg con draftMode en true pruebo con: todos los campos vacíos, cada uno de los atributos del leg: límite inferior - 1, límite inferior, límite inferior + 1, límite superior - 1, límite superior, límite superior + 1. En cada campo departureAirport y arrivalAirport he probado con todos los aeropuertos, y en el campo aircraft he probado con todas las aeronaves disponibles para ese manager. Además, he probado todas las restricciones de la actualización de legs.
- update2.safe: Olvidé probar la restricción de cuando un flightNumber ya está en uso
- update.hack: Inicio sesión con manager1 y mediante POST hacking pruebo a añadirle valor -1, 999 a cada uno de los campos desplegables. Además, para el campo desplegable aircraft, pruebo también con el id de algún aircraft que no pertenezca al manager. Por último, desde manager3 hago un GET hacking de update de un leg que no pertenece a este, tanto si está publicado como si no.

En la línea 42, se recorren 3 de las 4 posibles ramas. Pese a esto, la cobertura es del 100%, ya que todo el código es ejecutado.

```

package acme.features.manager.leg;

import java.util.Collection;

import org.springframework.beans.factory.annotation.Autowired;

import acme.client.components.models.Dataset;
import acme.client.components.views.SelectChoices;
import acme.client.helpers.NomentHelper;
import acme.client.services.AbstractGuiService;
import acme.client.services.GuiService;
import acme.entities.aircraft.Aircraft;
import acme.entities.airport.Airport;
import acme.entities.flight.Flight;
import acme.entities.leg.Leg;
import acme.entities.leg.LegStatus;
import acme.features.manager.flight.FlightRepository;
import acme.reales.manager.Manager;

@GuiService
public class LegUpdateService extends AbstractGuiServiceManager, Legs {

    @Autowired
    private LegRepository repository;

    @Autowired
    private FlightRepository flightRepository;

    @Override
    public void authorize() {
        int legId;
        Leg leg;
        Manager manager;
        boolean authorize = false;

        legId = super.getRequest().getData("id", int.class);
        leg = (Leg) this.repository.findById(legId).get();
        manager = (Manager) super.getRequest().getPrincipal().getActiveRealm();

        if (manager.getAirline().getId() == leg.getFlight().getAirline().getId() && leg.getDraftNode() != null)
            authorize = true;

        if (authorize) {
            int arrivalAirportId, departureAirportId, aircraftId, airlineId;
            Aircraft aircraft;
            Airport arrivalAirport;
            Airport departureAirport;

            airlineId = manager.getAirline().getId();
            aircraftId = super.getRequest().getData("aircraft", int.class);
            arrivalAirportId = super.getRequest().getData("airportArrival", int.class);
            departureAirportId = super.getRequest().getData("airportDeparture", int.class);
            aircraft = this.repository.findAircraftByAirlineId(airlineId, aircraftId);
            arrivalAirport = this.repository.findAirportByAirportId(arrivalAirportId);
            departureAirport = this.repository.findAirportByAirportId(departureAirportId);
            authorize = (aircraftId == 0 || aircraft != null) && (arrivalAirportId == 0 || arrivalAirport != null) && (departureAirportId == 0 || departureAirport != null);
        }

        super.getResponse().setAuthorised(authorize);
    }

    @Override
    public void load() {
        int id = super.getRequest().getData("id", int.class);
        Leg leg = (Leg) this.repository.findById(id).get();
        super.getOffer().addData(leg);
        Flight flight = (Flight) this.flightRepository.findById(leg.getFlight().getId()).get();
        boolean draftNode = flight.getDraftNode();
        super.getResponse().addGlobal("flightDraftNode", draftNode);
        super.getResponse().addGlobal("legDraftNode", leg.getDraftNode());
    }

    @Override
    public void bind(final Leg leg) {
        int aircraftId = super.getRequest().getData("aircraft", int.class);
        int departureAirportId = super.getRequest().getData("airportDeparture", int.class);
        int arrivalAirportId = super.getRequest().getData("airportArrival", int.class);

        Aircraft aircraft = this.repository.findAircraftByAircraftId(aircraftId);
        Airport departure = this.repository.findAirportByAirportId(departureAirportId);
        Airport arrival = this.repository.findAirportByAirportId(arrivalAirportId);

        leg.setAircraft(aircraft);
        leg.setDepartureAirport(departure);
        leg.setArrivalAirport(arrival);

        super.bindObject(leg, "flightNumber", "scheduledDeparture", "scheduledArrival", "status");
    }

    @Override
    public void validate(final Leg leg) {
        // R1: momento de salida y de llegada deben ser posterior a la fecha actual
        if (leg.getScheduledDeparture() != null) {
            boolean futureDepartureDate = NomentHelper.isFuture(leg.getScheduledDeparture());
            super.state(futureDepartureDate, "scheduledDeparture", "acme.validation.leg.scheduled-departure-not-future.message");
        }

        if (leg.getScheduledArrival() != null) {
            boolean futureArrivalDate = NomentHelper.isFuture(leg.getScheduledArrival());
            super.state(futureArrivalDate, "scheduledArrival", "acme.validation.leg.scheduled-arrival-not-future.message");
        }

        //R3: no puede existir otra leg con el mismo flight number
        String flightNumber = leg.getFlightNumber();
        boolean isUnique = !this.repository.existsByFlightNumberAndIdNot(flightNumber, leg.getId());
        super.state(isUnique, "flightNumber", "acme.validation.leg.duplicated-code.message");

        //R3: no puede ser el mismo aeropuerto de llegada que el de salida
        if (leg.getArrivalAirport() != null && leg.getDepartureAirport() != null) {
            boolean isDifferent = !leg.getArrivalAirport().equals(leg.getDepartureAirport());
            super.state(isDifferent, "airportArrival", "acme.validation.leg.same-airports.message");
        }

        //R7: los aeropuertos de llegada y salida no pueden ser nulos
        if (leg.getDepartureAirport() == null)
            super.state(false, "airportDeparture", "acme.validation.leg.departure-airport-not-null.message");
        if (leg.getArrivalAirport() == null)
            super.state(false, "airportArrival", "acme.validation.leg.arrival-airport-not-null.message");
    }

    @Override
    public void perform(final Leg leg) {
        this.repository.save(leg);
    }

    @Override
    public void unbind(final Leg leg) {
        Manager manager = (Manager) super.getRequest().getPrincipal().getActiveRealm();
        int airlineId = manager.getAirline().getId();

        Collection<Aircraft> aircrafts = this.repository.findAircraftByAirlineId(airlineId);
        Collection<Airport> airports = this.repository.findAllAirports();

        SelectChoices statusChoices = SelectChoices.from(LegStatus.class, leg.getStatus());
        SelectChoices aircraftChoices = SelectChoices.from(aircrafts, "registrationNumber", leg.getAircraft());
        SelectChoices departureChoices = SelectChoices.from(airports, "iataCode", leg.getDepartureAirport());
        SelectChoices arrivalChoices = SelectChoices.from(airports, "iataCode", leg.getArrivalAirport());

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

        dataset.put("statuses", statusChoices);
        dataset.put("aircraft", aircraftChoices.getSelected().getKey());
        dataset.put("aircrafts", aircraftChoices);
        dataset.put("airportDeparture", departureChoices.getSelected().getKey());
        dataset.put("airportDepartures", departureChoices);
        dataset.put("airportArrival", arrivalChoices.getSelected().getKey());
        dataset.put("airportArrivals", arrivalChoices);
        dataset.put("flightId", leg.getFlight().getId());

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

```

1.2.2.6 LegPublishService

- publish.safe: Inicio sesión con manager1 y con un vuelo con draftMode en true y pruebo con los campos vacíos, cada uno de los atributos del leg: límite inferior - 1, límite inferior, límite inferior + 1, límite superior - 1, límite superior, límite superior + 1. En el caso de los campos de texto, he probado también con caracteres no latinos e injections. En cada campo departureAirport y arrivalAirport he probado con todos los aeropuertos, y en el campo aircraft he probado con todas las aeronaves disponibles para ese manager. Además, he probado todas las restricciones de la publicación de legs.
- publish2.safe: Olvidé verificar la restricción de que no se pueden publicar los tramos cuando se solapan.
- publish.hack: Inicio sesión con manager1 y mediante POST hacking pruebo a añadirle valor -1, 999 a cada uno de los campos desplegables. Además, para el campo desplegable aircraft, pruebo también con el id de algún aircraft que no pertenezca al manager. Por último, desde manager3 hago un GET hacking de update de un leg que no pertenece a este, tanto si está publicado como si no.

En la línea 42, solo se recorre 1 de las 4 ramas posibles del condicional. La cobertura final es del 100%

```

package acme.features.manager.leg;

import java.util.Collection;

import org.springframework.beans.factory.annotation.Autowired;

import acme.client.components.models.Dataset;
import acme.client.components.vIEWS.SelectChoices;
import acme.client.helpers.NomemHelper;
import acme.client.services.AbstractUserService;
import acme.client.services.GuiService;
import acme.entities.aircraft.Aircraft;
import acme.entities.airport.Airport;
import acme.entities.flight.Flight;
import acme.entities.leg.Leg;
import acme.entities.leg.LegStatus;
import acme.features.manager.flight.FlightRepository;
import acme.reales.manager.Manager;

@GuiService
public class LegPublishService extends AbstractGuiServiceManager, Legs {

    @Autowired
    private LegRepository repository;

    @Autowired
    private FlightRepository flightRepository;

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

        legId = super.getRequest().getData("id", int.class);
        leg = (Leg) this.repository.findById(legId).get();
        manager = (Manager) super.getRequest().getPrincipal().getActiveRole();

        if (manager.getAirline().getId() == leg.getFlight().getAirline().getId() && leg.getDraftNode() == null)
            authorise = true;

        if (authorise) {
            int arrivalAirportId, departureAirportId, aircraftId, airlineId;
            Aircraft aircraft;
            Airport arrivalAirport;
            Airport departureAirport;

            airlineId = manager.getAirline().getId();
            aircraftId = super.getRequest().getData("aircraft", int.class);
            arrivalAirportId = super.getRequest().getData("airportArrival", int.class);
            departureAirportId = super.getRequest().getData("airportDeparture", int.class);
            aircraft = this.repository.findById(aircraftId).get();
            arrivalAirport = this.repository.findById(arrivalAirportId).get();
            departureAirport = this.repository.findById(departureAirportId).get();
            authorise = (aircraftId != 0 || aircraft != null) && (arrivalAirport != null || departureAirport != null);
        }

        super.getResponse().setAuthorised(authorise);
    }

    @Override
    public void load() {
        int id = super.getRequest().getData("id", int.class);
        Leg leg = (Leg) this.repository.findById(id).get();
        super.getTherfer().addData(leg);
        Flight flight = (Flight) this.flightRepository.findById(leg.getFlight().getId()).get();
        boolean draftNode = leg.getDraftNode();
        super.getResponse().addGlobal("flight-draftNode", draftNode);
        super.getResponse().addGlobal("leg-draftNode", leg.getDraftNode());
    }

    @Override
    public void bind(final Leg leg) {
        int aircraftId = super.getRequest().getData("aircraft", int.class);
        int departureAirportId = super.getRequest().getData("airportDeparture", int.class);
        int arrivalAirportId = super.getRequest().getData("airportArrival", int.class);

        Aircraft aircraft = this.repository.findById(aircraftId).get();
        Airport departure = this.repository.findById(departureAirportId).get();
        Airport arrival = this.repository.findById(arrivalAirportId).get();

        leg.setAircraft(aircraft);
        leg.setDepartureAirport(departure);
        leg.setArrivalAirport(arrival);

        super.bindObject(leg, "flightNumber", "scheduledDeparture", "scheduledArrival", "status");
    }

    @Override
    public void validate(final Leg leg) {
        Collection<Leg> legs = this.repository.findPublishedLegsByFlightId(leg.getFlight().getId());
        legs.add(leg);

        // 66: número de salida y de llegada deben ser posterior a la fecha actual
        if (leg.getScheduledDeparture() != null) {
            boolean futureDepartureDate = NomemHelper.isFuture(leg.getScheduledDeparture());
            super.state(futureDepartureDate, "scheduledDeparture", "acme.validation.leg.scheduled-departure-not-future.message");
        }

        if (leg.getScheduledArrival() != null) {
            boolean futureArrivalDate = NomemHelper.isFuture(leg.getScheduledArrival());
            super.state(futureArrivalDate, "scheduledArrival", "acme.validation.leg.scheduled-arrival-not-future.message");
        }

        // 64: no puede existir otro leg con el mismo flight number
        String flightNumber = leg.getFlightNumber();
        boolean unique = this.repository.existsByFlightNumberAndNot(flightNumber, leg.getId());
        super.state(unique, "flightNumber", "acme.validation.leg.duplicated-code.message");

        // 63: no puede ser el mismo aeropuerto de llegada que el de salida
        if (leg.getArrivalAirport() != null && leg.getDepartureAirport() != null) {
            boolean isDifferent = leg.getArrivalAirport().equals(leg.getDepartureAirport());
            super.state(isDifferent, "airportArrival", "acme.validation.leg.same-airports.message");
        }

        // 67: los aeropuertos de llegada y salida no pueden ser nulos
        if (leg.getDepartureAirport() == null)
            super.state(false, "airportDeparture", "acme.validation.leg.departure-airport-not-null.message");
        if (leg.getArrivalAirport() == null)
            super.state(false, "airportArrival", "acme.validation.leg.arrival-airport-not-null.message");

        // 68: no se debe publicar si los tramos se solapan
        if (legs.size() > 1) {
            boolean noOverlap = true;
            Leg previous = null;
            for (Leg current : legs) {
                if (previous != null)
                    if (NomemHelper.isAfterOrEqual(current.getScheduledDeparture(), previous.getScheduledArrival())) {
                        noOverlap = false;
                        break;
                    }
                previous = current;
            }
            super.state(noOverlap, "", "acme.validation.flight-legs-overlap.message");
        }
    }

    @Override
    public void perform(final Leg leg) {
        leg.setDraftNode(false);
        this.repository.save(leg);
    }

    @Override
    public void unbind(final Leg leg) {
        Manager manager = (Manager) super.getRequest().getPrincipal().getActiveRole();
        int airlineId = manager.getAirline().getId();

        Collection<Aircraft> aircrafts = this.repository.findAircraftByAirlineId(airlineId);
        Collection<Airport> airports = this.repository.findAirports();

        SelectChoices aircraftChoices = SelectChoices.from(aircrafts, "registrationNumber", leg.getAircraft());
        SelectChoices departureChoices = SelectChoices.from(airports, "iatacode", leg.getDepartureAirport());
        SelectChoices arrivalChoices = SelectChoices.from(airports, "iatacode", leg.getArrivalAirport());

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

        dataset.put("status", statusChoices);
        dataset.put("aircraft", aircraftChoices.getSelected().getKey());
        dataset.put("airportDeparture", departureChoices.getSelected().getKey());
        dataset.put("airportArrival", arrivalChoices.getSelected().getKey());
        dataset.put("flight", leg.getFlight().getId());

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

```

2. Pruebas de rendimiento

Para realizar las pruebas de rendimiento, nos hemos valido de la traza final que devuelve la herramienta replay proporcionada por el framework. Al devolver el fichero ".trace" y convertirlo a CSV extraemos los datos de tiempo de cada una de las peticiones. Vamos a analizar los rendimientos antes y después de optimizar los índices.

Especificaciones de la computadora en la que se han realizado las primeras pruebas:

- Procesador
 - Ryzen 4800H
 - 8 núcleos (16 hilos)
 - 4.2 GHz
- Memoria
 - 32 GiB de RAM
 - DDR4 @ 3200 MHz
 - Dual Channel (2x16GiB)
- Almacenamiento
 - 512 GiB de capacidad
 - NVMe @ Lectura 1600MB/s, Escritura 800MB/s
 - SSD

Se ha realizado un contraste de hipótesis mediante Z-Test que demuestra que los cambios han provocado mejoras en el rendimiento.

z-Test: Two Sample for Means		
	<i>before</i>	<i>after</i>
Mean	34,04023978	27,47757468
Known Variance	833,8759974	528,7625089
Observations	1169	1169
Hypothesized Mean Difference	0	
z	6,07851047	
P(Z<=z) one-tail	6,0652E-10	
z Critical one-tail	1,644853627	
P(Z<=z) two-tail	1,21E-09	
z Critical two-tail	1,959963985	

El dato relevante en el que nos tenemos que fijar es en valor P de dos colas, en el que estar muy cerca de cero nos ayuda a determinar que la media de tiempo antes y después de los cambios son diferentes y se pueden comparar.

En las siguientes tablas, se observa el rendimiento antes y después de aplicar los cambios:

Before			After		
Mean	34,04023978		Mean	27,47757468	
Standard Error	0,844585227		Standard Error	0,672547669	
Median	31,2351		Median	25,9952	
Mode	1,8927		Mode	1,5811	
Standard Deviation	28,87691115		Standard Deviation	22,99483657	
Sample Variance	833,8759974		Sample Variance	528,7625089	
Kurtosis	-1,412769446		Kurtosis	-1,440186073	
Skewness	0,271824757		Skewness	0,233340754	
Range	114,4927		Range	83,587	
Minimum	1,1443		Minimum	1,2489	
Maximum	115,637		Maximum	84,8359	
Sum	39793,0403		Sum	32121,2848	
Count	1169		Count	1169	
Confidence Level(95,0%)	1,657073775		Confidence Level(95,0%)	1,319536582	
Interval (ms)	32,383166	35,69731355	Interval (ms)	26,1580381	28,79711126
Interval (s)	0,032383166	0,035697314	Interval (s)	0,026158038	0,028797111

Tras la adición de los índices, se reduce el Intervalo en 6.9 milisegundos, una diferencia bastante notable, provocando una reducción del 26.7%

He instalado el espacio de trabajo en otra computadora de familia ligeramente peor en especificaciones:

- Procesador
 - Intel Core 5 8250U, 8va generación
 - 4 núcleos (8 hilos)
 - 3.4 GHz
- Memoria
 - 16 GiB de RAM
 - DDR4 @ 2400 MHz
 - Dual Channel (2x8GiB)
- Almacenamiento
 - 240 GiB de capacidad
 - SATA-III @ Lectura 500MB/s, Escritura 450MB/s
 - SSD

z-Test: Two Sample for Means		
	<i>before</i>	<i>after</i>
Mean	46,07674243	41,08152831
Known Variance	1766,190094	1262,776999
Observations	1169	1169
Hypothesized Mean Differ	0	
z	3,103230868	
P(Z<=z) one-tail	0,000957101	
z Critical one-tail	1,644853627	
P(Z<=z) two-tail	0,001914203	
z Critical two-tail	1,959963985	

El valor P de dos colas vuelve a estar muy cerca de cero, lo que nos indica que la media de tiempos antes y después de implementar los índices son diferentes y se pueden comparar.

En las siguientes tablas, se observa el rendimiento antes y después de aplicar los cambios:

<i>Before</i>				<i>After</i>			
Mean	46,07674243			Mean	41,08152831		
Standard Error	1,229168626			Standard Error	1,039336252		
Median	46,2731			Median	40,0679		
Mode	#N/A			Mode	7,1385		
Standard Deviation	42,02606446			Standard Deviation	35,53557372		
Sample Variance	1766,190094			Sample Variance	1262,776999		
Kurtosis	0,003308548			Kurtosis	-1,291279059		
Skewness	0,681401125			Skewness	0,258287456		
Range	209,7417			Range	153,8907		
Minimum	1,157			Minimum	1,1096		
Maximum	210,8987			Maximum	155,0003		
Sum	53863,7119			Sum	48024,3066		
Count	1169			Count	1169		
Confidence Level(95,0%)	2,411625292			Confidence Level(95,0%)	2,039174723		
Interval (ms)	43,66511714	48,48836772		Interval (ms)	39,04235359	43,12070304	
Interval (s)	0.043665117	0.048488368		Interval (s)	0.039042354	0.043120703	

Tras la adición de los índices, se reduce el Intervalo en 5.37 milisegundos, una diferencia bastante notable, provocando una reducción del 11.07%.

3. Conclusión

Este informe evidencia que el proceso de testing formal implementado en el proyecto ha sido riguroso y orientado a la detección y corrección de posibles errores. Las pruebas realizadas cubren una amplia variedad de escenarios, en especial aquellos relacionados con los requisitos 8 y 9 del Student 1, lo cual permite minimizar significativamente la probabilidad de fallos en el sistema.

En cuanto al análisis de rendimiento, se ha llevado a cabo Z-Test con un nivel de confianza del 95%. Los resultados muestran que, gracias a las optimizaciones aplicadas para acelerar las consultas a la base de datos, se ha logrado una mejora apreciable en los tiempos de respuesta.

En términos generales, se puede afirmar que el sistema presenta una funcionalidad sólida y estable gracias al alto grado de cobertura de pruebas con un 100% en todas, y con mejoras en el rendimiento gracias a los índices que, pese a que la reducción de tiempos es menor en equipos menos potentes, es bastante notable.