

TESTING REPORT

Grupo C1.052

<https://github.com/BVP2455/Acme-ANS-D04>

José Luis Cegrí Marcos

joscegmar@alum.us.es

Carlos Galea Magro

Miguel Yan García Azuara

Francisco Javier Martos Romero

Francisco Gago Vázquez

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.

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.....	23

1. Pruebas funcionales

1.1 Introducción







En las pruebas funcionales, hemos utilizado la herramienta *record* que nos proporciona el framework Acme-Framework-25.5.0 para probar cada una de las funcionalidades de *Leg* y *Flight*. La herramienta carga el proyecto en la dirección <http://localhost:8082/Acme-ANS-D04?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% para las funcionalidades de *Flight*; y una cobertura del 99,9% 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 %	904	0	904
>  FlightController.java	 100,0 %	35	0	35
>  FlightCreateService.java	 100,0 %	114	0	114
>  FlightDeleteService.java	 100,0 %	124	0	124
>  FlightListService.java	 100,0 %	71	0	71
>  FlightPublishService.java	 100,0 %	247	0	247
>  FlightShowService.java	 100,0 %	99	0	99
>  FlightUpdateService.java	 100,0 %	214	0	214

3. Cobertura de la funcionalidad de vuelos

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
▼  acme.features.manager.leg	 99,9 %	2.057	3	2.060
>  LegController.java	 100,0 %	35	0	35
>  LegCreateService.java	 100,0 %	446	0	446
>  LegDeleteService.java	 100,0 %	182	0	182
>  LegListService.java	 100,0 %	143	0	143
>  LegShowService.java	 100,0 %	224	0	224
>  LegUpdateService.java	 100,0 %	457	0	457
>  LegPublishService.java	 99,5 %	570	3	573

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%.

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

    // Internal state -----

    @Autowired
    private FlightRepository repository;

    // AbstractGuiService interface -----

    @Override
    public void authorize() {
        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);
    }
}
```

1.2.1.2 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%.

```
@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. En la línea 78, el bucle for se ejecuta, pero no en todos los escenarios posibles, lo que deja parte de su lógica sin recorrer. En la línea 80, la condición interna del bucle no se evalúa en ambas ramas (true y false), ya que en los tests no se fuerza un caso en el que el aeropuerto de llegada de la etapa anterior no coincida con el de salida de la etapa actual. La cobertura final es del 100%.

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

    // Internal state -----

    @Autowired
    private FlightRepository repository;

    @Autowired
    private LegRepository legRepository;

    // AbstractGuiService interface -----

    @Override
    public void authorize() {
        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;
        boolean confirmation;

        isDraftMode = flight.getDraftMode();
        confirmation = super.getRequest().getData("confirmation", boolean.class);

        //RS: si hay legs publicados y los aeropuertos no son consecutivos, no se puede modificar el atributo selftransfer
        Collection<Leg> publishedLegs = this.legRepository.findPublishedLegsByFlightId(flight.getId());
        if (flight.getSelftransfer() && publishedLegs.size() > 1) {
            boolean airportsAreConsecutive = true;
            Leg previous = null;
            for (Leg current : publishedLegs) {
                if (previous != null)
                    if (!previous.getArrivalAirport().equals(current.getDepartureAirport())) {
                        airportsAreConsecutive = false;
                        break;
                    }
                previous = current;
            }
            super.state(airportsAreConsecutive, "", "acme.validation.flight.selftransfer-legs-not-consecutive.message");
        }
        super.state(isDraftMode, "", "acme.validation.flight.draftMode.updated.message");
        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);
    }
}
```

1.2.1.5 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. En la línea 86, el bucle for se ejecuta, pero no en todos los escenarios posibles, lo que deja parte de su lógica sin recorrer. En la línea 88, la condición interna del bucle no se evalúa en ambas ramas (*true* y *false*), ya que en los tests no se fuerza un caso en el que el aeropuerto de llegada de la etapa anterior no coincida con el de salida de la etapa actual. La cobertura final es del 100%.

```
@UIService
public class FlightPublishService extends AbstractUIServiceManager, Flights {

    // Internal state -----
    @Autowired
    private FlightRepository repository;
    @Autowired
    private LegRepository legRepository;
    // AbstractUIService 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, "leg", "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, "", "acme.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, "", "acme.validation.flight.legs-not-published.message");

        // R4: cañilla de confirmación
        boolean confirmation = super.getRequest().getData("confirmation", boolean.class);
        super.state(confirmation, "confirmation", "acme.validation.confirmation.message");

        //R5: si hay legs publicados y los aeropuertos no son consecutivos, no se puede modificar el atributo selfTransfer
        Collection<Leg> publishedLegs = this.legRepository.findPublishedLegsByFlightId(flight.getId());
        if (!flight.getSelfTransfer() && publishedLegs.size() > 1) {
            boolean airportsAreConsecutive = true;
            Leg previous = null;
            for (Leg current : publishedLegs) {
                if (previous != null)
                    if (!previous.getArrivalAirport().equals(current.getDepartureAirport())) {
                        airportsAreConsecutive = false;
                        break;
                    }
                previous = current;
            }
            super.state(airportsAreConsecutive, "", "acme.validation.flight.selfTransfer-legs-not-consecutive.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, "leg", "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 AbstractGuiService<Manager, 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.findById(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 AbstractGuiServiceManager, 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 tramo lo está visitando un manager al que no le pertenece, se añaden las aeronaves de esa aerolínea
        if (airlineId != leg.getFlight().getAirline().getId())
            aircrafts.addAll(this.repository.findAircraftsByAirlineId(leg.getFlight().getAirline().getId()));
        Collection<Airport> airports = this.repository.findAllAirports();

        statusChoices = SelectChoices.from(LegStatus.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*.
- *create.hack*: Inicio sesión con *manager1* y mediante POST hacking pruebo a añadirle valor -1, 999 a cada uno de los campos desplegados. Además, para el campo desplegable *aircraft*, pruebo también con el id de algún *aircraft* que no pertenezca al manager.
- *create2.safe*: Olvidé probar con caracteres no latinos e injections en el campo *FlightNumber*.

En la línea 42, parte del bloque condicional no se recorre (se ejecutan 1 de las 2 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.

En la línea 45, ocurre lo mismo: solo se ejecuta una de las ramas posibles del condicional if (authorise), ya que en los tests no se evalúa el caso en el que authorise es false.

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

```

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

    @Autowired
    private LegRepository repository;

    @Autowired
    private FlightRepository flightRepository;

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

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

            method = super.getRequest().getMethod();

            if (method.equals("GET"))
                authorise = 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.findAircraftByAirlineId(airlineId, aircraftId);
                arrivalAirport = this.repository.findAirportByAirportId(arrivalAirportId);
                departureAirport = this.repository.findAirportByAirportId(departureAirportId);
                authorise = (aircraftId == 0 || aircraft != null) && (arrivalAirportId == 0 || arrivalAirport != null) && (departureAirportId == 0 || departureAirport != null);
            }
        }

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

    @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.setDraftMode(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.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 = MomentHelper.isFuture(leg.getScheduledDeparture());
            super.state(futureDepartureDate, "scheduledDeparture", "acme.validation.leg.scheduled-departure-not-future.message");
        }
        if (leg.getScheduledArrival() != null) {
            boolean futureArrivalDate = MomentHelper.isFuture(leg.getScheduledArrival());
            super.state(futureArrivalDate, "scheduledArrival", "acme.validation.leg.scheduled-arrival-not-future.message");
        }
        //R3: no puede existir otro leg con el mismo flight number
        String flightNumber = leg.getFlightNumber();
        boolean isUnique = !this.repository.existsByFlightNumber(flightNumber);
        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");

        //R5: requisito de confirmacion
        boolean confirmation = super.getRequest().getData("confirmation", boolean.class);
        super.state(confirmation, "confirmation", "acme.validation.confirmation.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.findAllAircraftsByAirlineId(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);
    }
}

```


1.2.2.4 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.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) {
        boolean isDraftMode = leg.getDraftMode();

        boolean status = isDraftMode == true;

        super.state(status, "a", "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*.
- *update.hack*: Inicio sesión con *manager1* y mediante POST hacking pruebo a añadirle valor -1, 999 a cada uno de los campos despleables. Además, para el campo despleable *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.

```

@Override
public void authorize() {
    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;

    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.findAircraftByAirlineId(airlineId, aircraftId);
        arrivalAirport = this.repository.findAirportByAirportId(arrivalAirportId);
        departureAirport = this.repository.findAirportByAirportId(departureAirportId);
        authorise = (aircraftId == 0 || aircraft != null) && (arrivalAirportId == 0 || arrivalAirport != null) && (departureAirportId == 0 || departureAirport != null);
    }

    super.getResponse().setAuthorized(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) {
    // R1: momento de salida y de llegada deben ser posterior a la fecha actual
    if (leg.getScheduledDeparture() != null) {
        boolean futureDepartureDate = MomentHelper.isFuture(leg.getScheduledDeparture());
        super.state(futureDepartureDate, "scheduledDeparture", "acme.validation.leg.scheduled-departure-not-future.message");
    }
    if (leg.getScheduledArrival() != null) {
        boolean futureArrivalDate = MomentHelper.isFuture(leg.getScheduledArrival());
        super.state(futureArrivalDate, "scheduledArrival", "acme.validation.leg.scheduled-arrival-not-future.message");
    }

    //R3: no puede existir otro 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");

    //R5: requisito de confirmacion
    boolean confirmation = super.getRequest().getData("confirmation", boolean.class);
    super.state(confirmation, "confirmation", "acme.validation.confirmation.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(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*.
- *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 43, solo se recorre 1 de las 4 ramas posibles del condicional. En la línea 97, se ejecuta solo 1 de las 2 condiciones. En la línea 112 se ejecuta solo 1 de las 2 ramas posibles. Por ello, la cobertura es del 99,5%, ligeramente inferior al resto de clases.

```

@GuiService
public class LegPublishService extends AbstractGuiServiceManager, 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;

        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.findAircraftByAirlineId(airlineId, aircraftId);
            arrivalAirport = this.repository.findAirportByAirportId(arrivalAirportId);
            departureAirport = this.repository.findAirportByAirportId(departureAirportId);
            authorise = (aircraftId == 0 || aircraft != null) && (arrivalAirportId == 0 || arrivalAirport != null) && (departureAirportId == 0 || 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.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.findAircraftByAirlineId(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) {
        Collection<Leg> legs = this.repository.findPublishedByFlightId(leg.getFlight().getId());
        legs = legs.stream().filter(l -> l.getId() != leg.getId()).collect(Collectors.toList());
        legs.add(leg);

        // R1: momento de salida y de llegada deben ser posterior a la fecha actual
        if (leg.getScheduledDeparture() != null) {
            boolean futureDepartureDate = MomentHelper.isFuture(leg.getScheduledDeparture());
            super.state(futureDepartureDate, "scheduledDeparture", "acme.validation.leg.scheduled-departure-not-future.message");
        }
        if (leg.getScheduledArrival() != null) {
            boolean futureArrivalDate = MomentHelper.isFuture(leg.getScheduledArrival());
            super.state(futureArrivalDate, "scheduledArrival", "acme.validation.leg.scheduled-arrival-not-future.message");
        }

        //R4: no puede existir otro 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");

        //R6: requisito de confirmacion
        boolean confirmation = super.getRequest().getData("confirmation", boolean.class);
        super.state(confirmation, "confirmation", "acme.validation.confirmation.message");

        //R9: No dejar publicar si los tramos se solapan
        if (legs.size() > 1) {
            boolean noOverlap = true;
            Leg previous = null;
            for (Leg current : legs) {
                if (previous != null)
                    if (!MomentHelper.isAfterOrEqual(current.getScheduledDeparture(), previous.getScheduledArrival())) {
                        noOverlap = false;
                        break;
                    }
                previous = current;
            }
            super.state(noOverlap, "", "acme.validation.flight.legs-overlap.message");
        }

        //R10: aeropuertos deben ser consecutivos si el vuelo no está en tránsito
        if ((leg.getFlight().getIsTransfer() && legs.size() > 1) {
            boolean airportsAreConsecutive = true;
            Leg previous = null;
            for (Leg current : legs) {
                if (previous != null)
                    if (!previous.getArrivalAirport().equals(current.getDepartureAirport())) {
                        airportsAreConsecutive = false;
                        break;
                    }
                previous = current;
            }
            super.state(airportsAreConsecutive, "", "acme.validation.flight.legs-not-consecutive.message");
        }
    }

    @Override
    public void perform(final Leg leg) {
        leg.setDraftMode(false);
        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("aircrafts", aircraftChoices.getSelected().getKey());
        dataset.put("aircrafts", aircraftChoices);
        dataset.put("airportDeparture", departureChoices.getSelected().getKey());
        dataset.put("airportDeparture", departureChoices);
        dataset.put("airportArrival", arrivalChoices.getSelected().getKey());
        dataset.put("airportArrival", arrivalChoices);
        dataset.put("flightId", 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. Se ha realizado un contraste de hipótesis mediante Z-Test que demuestra que los cambios no han tenido ningún cambio en el rendimiento.

	<i>Before</i>	<i>After</i>
Media	28,98011249	28,6525234
Varianza (conocida)	482,636973	485,235251
Observaciones	1113	1113
Diferencia hipotética de las medias	0	
z	0,351291829	
P(Z<=z) una cola	0,362684712	
Valor crítico de z (una cola)	1,644853627	
Valor crítico de z (dos colas)	0,725369424	
Valor crítico de z (dos colas)	1,959963985	

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

<i>Before</i>			<i>After</i>		
Media	28,9801125		Media	28,6525234	
Error típico	0,65851051		Error típico	0,66028068	
Mediana	28,5866		Mediana	27,2992	
Moda	1,6492		Moda	46,1072	
Desviación estándar	21,9690003		Desviación estándar	22,028056	
Varianza de la muestra	482,636973		Varianza de la muestra	485,235251	
Curtosis	-1,41610559		Curtosis	-1,42972427	
Coefficiente de asimetría	0,06781816		Coefficiente de asimetría	0,10749149	
Rango	100,1707		Rango	89,1454	
Mínimo	1,0653		Mínimo	1,1476	
Máximo	101,236		Máximo	90,293	
Suma	32254,8652		Suma	31890,2585	
Cuenta	1113		Cuenta	1113	
Nivel de confianza(95,0%)	1,29206322		Nivel de confianza(95,0%)	1,29553646	
Interval (ms)	27,6880493	30,2721757	Interval (ms)	27,3569869	29,9480598
Interval (s)	0,02768805	0,03027218	Interval (s)	0,02735699	0,02994806

Tras la adición de los índices, se reduce el Intervalo en 0.2941159 milisegundos.

He probado con otra computadora de mi familia, y los resultados son más o menos parecidos.

After			After PC_2		
Media	28,6525234		Media	30,0851495	
Error típico	0,66028068		Error típico	0,69329471	
Mediana	27,2992		Mediana	28,66416	
Moda	46,1072		Moda	46,1072	
Desviación estándar	22,028056		Desviación estándar	23,1294588	
Varianza de la muestra	485,235251		Varianza de la muestra	509,497014	
Curtosis	-1,42972427		Curtosis	-1,42972427	
Coficiente de asimetría	0,10749149		Coficiente de asimetría	0,11286607	
Rango	89,1454		Rango	93,60267	
Mínimo	1,1476		Mínimo	1,20498	
Máximo	90,293		Máximo	94,80765	
Suma	31890,2585		Suma	33484,7714	
Cuenta	1113		Cuenta	1168,65	
Nivel de confianza(95,0%)	1,29553646		Nivel de confianza(95,0%)	1,36031328	
Interval (ms)	27,3569869	29,9480598	Interval (ms)	30,1610781	33,017736
Interval (s)	0,02735699	0,02994806	Interval (s)	0,03166913	0,03466862

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, pese a las optimizaciones aplicadas para acelerar las consultas a la base de datos, no se ha logrado una mejora apreciable en los tiempos de respuesta. Esto ha permitido detectar ciertos métodos que, a pesar de haber sido revisados e incluso refactorizados, no han producido el impacto esperado en el rendimiento.

En términos generales, se puede afirmar que el sistema presenta una funcionalidad sólida y estable gracias al alto grado de cobertura de pruebas, llegando al 100% en la mayoría de estas, aunque aún existen aspectos relacionados con el rendimiento que podrían afectar a usuarios con dispositivos o entornos menos potentes.