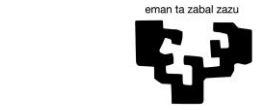


Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Facultad de Informática

Grado de Ingeniería Informática

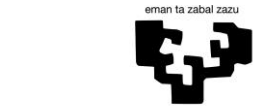
▪ Trabajo Fin de Grado ▪

Ingeniería de Software

Desarrollo de un servicio web que provee datos IATA los sistemas aeroportuarios de IKUSI

Ricardo Herradura Collazos

Febrero 2020



Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Facultad de Informática

Grado de Ingeniería Informática

▪ Trabajo Fin de Grado ▪

Ingeniería de Software

Desarrollo de un servicio web que provee datos IATA los sistemas aeroportuarios de IKUSI

Ricardo Herradura Collazos

Febrero 2020

Dirigido por: Juan Ignacio Iturrioz Sánchez

Agradecimientos

Me gustaría dedicar este Trabajo de Fin de Grado a mi familia: a mis padres, Javier y Blanqui y mi pareja, Sarai. Han sido, indudablemente, las personas que más me han ayudado en todo este tiempo, estando tanto en las buenas como en las malas. Por los consejos y ánimos recibidos, por ayudarme a crecer y enseñarme que los objetivos de la vida se consiguen gracias al esfuerzo, constancia y dedicación. En definitiva, por nunca dejar de confiar y creer en mí. Gracias familia.

Agradecer también a todos mis compañeros y amigos que he conocido durante este tiempo. Compañeros con los cuales he compartido mucho tiempo, he aprendido y me han ayudado, espero yo también haberles servido de ayuda. Gracias amigos.

También quiero dar las gracias a mi tutor, Jon Iturrioz, por ser mi director de Trabajo de Fin de Grado, por dedicarme el tiempo necesario y orientarme en el desarrollo de dicho proyecto. Gracias Jon.

Para finalizar, querría dar la gracias a la empresa IKUSI por confiar en mí para la realización de este proyecto y permitirme aprender de profesionales del sector. En especial, al equipo Condor, por brindarme la oportunidad de formar parte de ellos, por enseñarme y ayudarme con el proyecto, aunque nunca se cansaron de la broma de "*te has equivocado de camiseta, es Condor no Puma*". Gracias equipo.

Resumen

Este documento corresponde a la memoria del proyecto de fin de grado “Desarrollo de un servicio web que forma parte del ecosistema de IKUSI AMS (Airport Management System) y que se encargará de proveer información actualizada de los datos IATA (aeropuertos, aerolíneas y aeronaves) a las implantaciones realizadas en los aeropuertos de los sistemas que forman parte de IKUSI AMS: Condor, Beluga y Dolphin”, desarrollado por Ricardo Herradura Collazos para la titulación de Grado en Ingeniería Informática de la Universidad UPV/EHU de Donostia-San Sebastián en la especialidad de Ingeniería del Software.

El proyecto ha sido realizado para la empresa IKUSI perteneciente al grupo *Velatia*, experta en soluciones digitales y servicios de alto valor añadido basados en tecnologías avanzadas, en la línea de negocio *Soluciones de negocio especializadas*, dentro del sector *Aeroportuario*.

Concretamente, se ha desarrollado una API REST con la herramienta *Spring Boot*, que simplifica el desarrollo de aplicaciones basadas en el *framework Spring Core*; y por otra parte se ha desarrollado una interfaz web (SPA) con la herramienta ReactJS, una biblioteca de JavaScript para la creación de interfaces. Se ha realizado en lenguaje *Java* dentro de la plataforma de software *Eclipse* para el desarrollo de la API Rest, y en lenguaje *JavaScript* con la plataforma *IntelliJ IDEA* para el desarrollo del SPA.

Índice de contenidos

Índice de ilustraciones

Índice de tablas

Capítulo 1

Introducción

Este documento corresponde a la memoria del proyecto *IK-IATA*, el cual se realiza en la empresa *IKUSI*, dentro de la línea de negocio *Soluciones de negocio especializadas*, dentro del sector *Aeroportuario*. El objetivo principal del proyecto es el desarrollo de una API (*Application Programming Interface*), en español, *Interfaz de Programación de Aplicaciones*, la cual será una API común para el resto de aplicaciones de IKUSI AMS (Airport Management System) y la cual manejará datos maestros almacenados en el cloud de IKUSI. Las aplicaciones de IKUSI AMS accederán a los datos y podrán descargase los datos mediante una conexión *Kafka* siempre y cuando estén subscritos a este servicio. Para modificar y gestionar los datos, se creará una interfaz web (*SPA, Single-Page Application*) de mantenimiento y configuración a la cual solo tendrá acceso el administrador.

Todo el proyecto deberá cumplir con unos criterios de calidad muy elevados requeridos por la empresa, y, desarrollarse en función de una metodología de trabajo bastante estricta.

1.1 Antecedentes

En el verano del 2018 empecé el periodo de prácticas voluntarias en la empresa IKUSI, con el objetivo de conocer el entorno laboral/profesional, integrándome en un equipo de trabajo, adaptándome a la metodología del grupo y principalmente, para aprender mucho más.

Las prácticas consistieron en el desarrollo de una aplicación Web en Java en el proyecto Condor. Las primeras semanas fueron únicamente dedicadas al estudio del sector aeroportuario para a posteriori, tener mejor conocimiento y manejo de los datos ofrecidos en la aplicación Web.

A raíz de esta experiencia la empresa me ofreció realizar el Trabajo de Fin de Grado con ellos (de ahora en adelante le llamaremos TFG), y un año más tarde, tras terminar todas las asignaturas de la universidad, inicie una práctica de 2 meses para la realización del TFG, y tras terminar esta práctica sin haber acabado el proyecto, me ofrecieron un contrato de 6 meses gracias a la organización *Fundación Novia Salcedo*, que se encarga de buscar oportunidades a jóvenes que necesitan ayudan para iniciarse en el mundo laboral ofreciendo contratos de prácticas.

1.2 Descripción de objetivo

El proyecto de la API se desarrollará en la plataforma de software *Eclipse* mediante la herramienta *Maven* para la gestión y construcción de proyectos Java, usando el framework (entorno de trabajo) *Spring Framework* para el desarrollo de aplicaciones y se implementará en el lenguaje Java. La otra parte del proyecto, la interfaz web, se desarrollará en la plataforma de software *IntelliJ IDEA* mediante la herramienta *ReactJS*, una biblioteca de *JavaScript*.

Estos serían los objetivos principales:

- Desarrollar el proyecto IK-IATA:
  - Analizar los modelos de datos de nuestros sistemas CONDOR, BELUGA y DOLPHIN en cuanto a tablas maestras.
  - Desplegar un servidor en el cloud de IKUSI con un modelo de datos que acorde a la información proporcionada por el proveedor de servicio.
  - Crear un mecanismo de actualización de nuestro servidor hosteado.
  - Crear un microservicio REST que sea común para CONDOR, BELUGA y DOLPHIN, pero que cada uno de los sistemas actualice en su base de datos los datos que necesite cada sistema.
  - Desarrollar una interfaz web, al estilo de IKUSI, para el mantenimiento y configuración del microservicio.
- Realizar un documento de memoria que recoja toda la información sobre el TFG.
- Realizar una presentación para la defensa de este proyecto.

A parte de los objetivos principales del proyecto, también existen otros objetivos de segundo grado, los cuales se basan en lograr ciertas capacidades:

- Ser capaces de desarrollar un diseño del dominio ante un problema real.
- Elegir las tecnologías que se van a usar.
- Investigar y aprender sobre una *API-REST* e interfaz web *ReactJS*.
- Aprender a trabajar con un equipo de trabajo profesional.
- Adaptarme a trabajar con el método SCRUM MASTER en la vida real.
- Fomentar el aprendizaje autodidacta.

1.3 Metodología de desarrollo

Para llevar a cabo el correcto desarrollo del proyecto, se ha seguido la metodología de trabajo de la empresa IKUSI que se basa en la metodología *Scrum* [1]. Esta metodología sirve para el desarrollo de proyectos gracias a un entorno de trabajo ágil y muy completo; es adaptable, iterativo, rápido, flexible y eficaz, cual objetivo primordial es satisfacer las necesidades principales del cliente a través de un entorno de comunicación trasparente.

El desarrollo de realiza de manera iterativa e incremental. Es decir, empezar el proyecto con funcionalidades básicas e ir dotándolo de nuevas funciones hasta lograr terminar todas las funcionalidades de cada caso de uso. Casos de uso que se detallaran posteriormente.

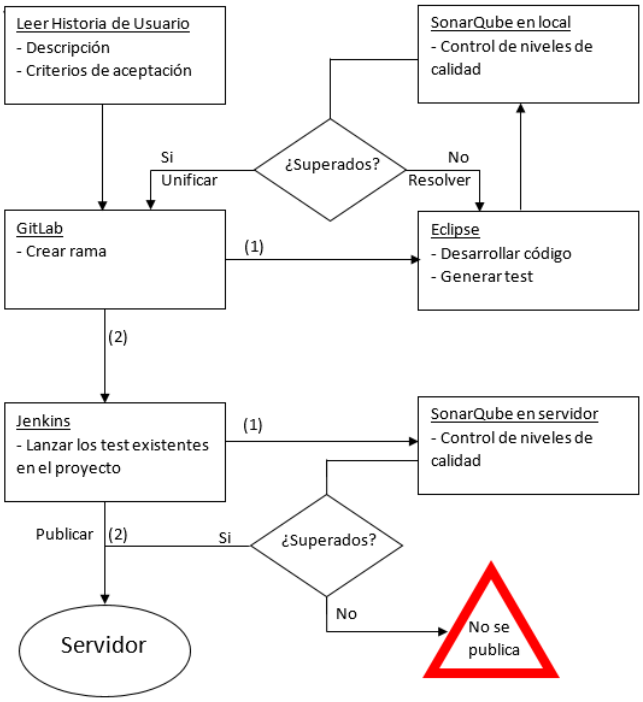


Ilustración 1 – Metodología de desarrollo

Se empieza leyendo y entendiendo las *Historias de Usuario (HU)* en las cuales aparece una descripción de la tarea y los criterios de aceptación. Tras entender la historia que se ha escogido para desarrollar, suponiendo que previamente ya estén montados y funcionando los entornos de trabajo, desarrollo y test, iremos al servicio web de control de versiones *GitLab* en el cual crearemos una nueva rama para desarrollar la HU. Una vez creada la rama, iremos a la plataforma de software *Eclipse* en la cual desarrollaremos el/los métodos necesarios para completar la HU y posteriormente generaremos unos test gracias a los cuales nos aseguraremos del correcto funcionamiento del código recién implementado.

Tras terminar la implementación, ejecutaremos una plataforma *(SonarQube)* para evaluar el código fuente de manera local, para antes de subir una nueva versión, tengamos la mayor certeza de que nuestro código pasara los niveles de calidad establecidos por la empresa.

Tras ejecutar *SonarQube* en nuestro ordenador, si no pasamos el control de calidad, tendremos que realizar los cambios que sean necesarios y volver a ejecutarlo hasta pasar el control; sin embargo, si pasamos los niveles de calidad establecidos, podremos subir el código a *GitLab* donde automáticamente tras subirlo y unificar todos los cambios en una sola rama de desarrollo principal, *Jenkins,* ejecutará de nuevo todos los test existentes hasta el momento en el proyecto, avisando si existiese algún fallo de test tras la subida al GitLab o sino ejecutando un *SonarQube* implantado en el servidor general de IKUSI, el cual volverá a evaluar el código fuente.

Si todo transcurre correctamente, *Jenkins* automáticamente despliega el proyecto con los cambios nuevos en el servidor.

En conclusión, gracias a esta metodología, que ya anteriormente había usado en alguna asignatura de la universidad, he sido capaz de ir completando el trabajo poco a poco, entendiendo en cada momento que se me exigía y de este modo poder terminarlo correctamente y sin sobrecargas de trabajo innecesarias.

1.4 Resumen

A continuación, se describen brevemente los apartados que contiene este documento:

- En el apartado **Planificación** se describe cómo va a ser la trayectoria del proyecto con sus límites de entrega y demás.
- En el apartado **Análisis de requisitos** se especifican los requisitos, casos de uso, requisitos no funcionales… y se detallan los modelos de datos.
- En el apartado **Diseño** se detalla cual va ser la arquitectura y diseño del proyecto.
- En el apartado **Implementación** se describe cómo se va a desarrollar el proyecto y con qué herramientas de trabajo.
- En el apartado **Verificación y pruebas**, se van a detallar todas las pruebas pasadas para asegurar un correcto desarrollo sin fallos de código y calidad elevada del producto.
- En el apartado **Seguimiento y control** se especifica el existo o fracaso del proyecto completo o de alguna de las fases.
- En el apartado **Conclusiones y futuras mejoras** se revisa el proyecto y se plantean futuras mejorar para el proyecto.

A lo largo de la memoria se mencionan el uso de varias herramientas de trabajo, que en el punto *5.3 Herramientas utilizadas* se detallan que son y cuando se han usado.

Capítulo 2

Planificación

Para realizar un proyecto de este tipo, es imprescindible realizar una planificación inicial para poder completarlo. Para ello, desglosaremos el proyecto en varios paquetes de trabajo, cada uno con sus tareas, y definiremos una estimación inicial sobre estas para ver el tiempo que dedicaremos en cada fase.

2.1 Estructura del trabajo

Es muy importante repartir el proyecto en varios paquetes de trabajo para poder alcanzar los objetivos definidos. Para realizar esta repartición de trabajo, es muy recomendable la técnica de EDT (Estructura de Descomposición de Trabajo) dividendo el proyecto en tareas y sub-tareas.

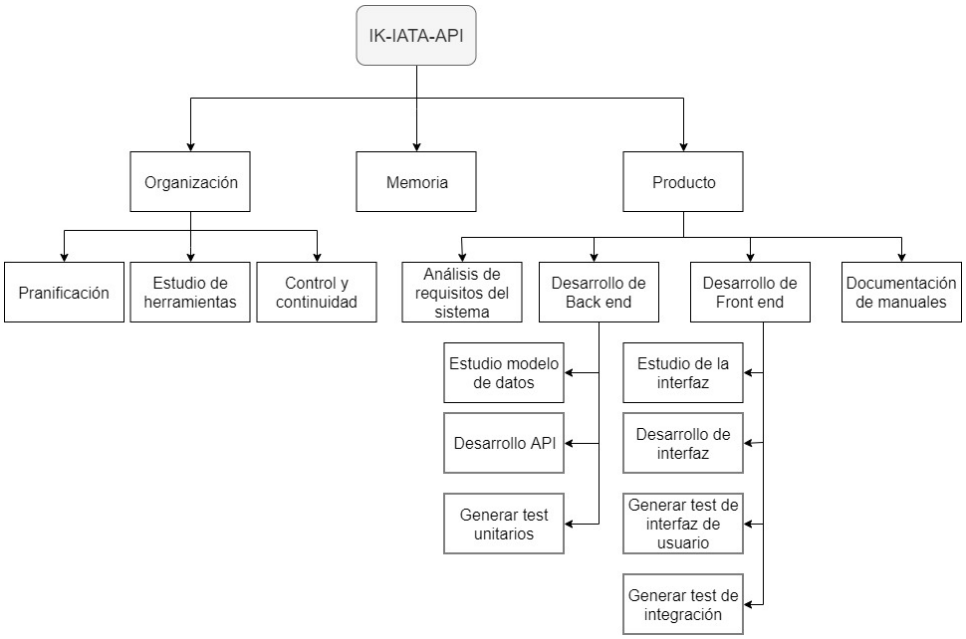


Ilustración 2 – Diagrama EDT

El paquete **Planificación** integrará las tareas principales para realizar una primera planificación, y en caso de ser necesario, se realizará una replanificación para adecuar la planificación con el día a día.

El paquete **Estudio herramientas** se basará en las tareas necesarias para el estudio y aprendizaje de las herramientas usadas en este proyecto, desde las herramientas para la organización, hasta las herramientas para el desarrollo del proyecto.

El paquete **Control y continuidad** integrará las tareas necesarias para garantizar el desarrollo correcto del proyecto. Concretamente, la dedicación, continuidad y la realización de las tareas en las fechas especificadas.

El paquete **Memoria** constituirá en el desarrollo correcto de la documentación de dicho proyecto.

El paquete **Análisis de Requisitos del sistema** integrará las tareas necesarias para un análisis principal de los requisitos necesarios para el desarrollo del producto correctamente.

El paquete **Estudio modelo de datos** integrará las tareas para el estudio de los datos maestros que usará la API.

El paquete **Desarrollo API** estará constituido de las tareas tanto como la preparación del entorno para el desarrollo, como, el desarrollo de la API.

El paquete **Generar test unitarios** consiste en el desarrollo de los test unitarios para la verificación del correcto desarrollo y función de cada método implementado.

El paquete **Estudio de la interfaz** integrará las tareas necesarias del estudio de la interfaz y estilo común de IKUSI.

El paquete **Desarrollo de interfaz** consiste en la preparación del entorno para el desarrollo y las tareas del desarrollo de la web.

El paquete **Generar test de interfaz de usuario** agrupa las tareas de preparación de entorno y desarrollo de test de interfaz de usuario para el correcto funcionamiento de las interfaces.

El paquete **Generar test de integración** agrupa las tareas de preparación de entorno y desarrollo de test de integración para el correcto funcionamiento de cada llamada realizada a la aplicación.

El paquete **Documentación de manuales** integrará las tareas para el desarrollo correcto de los manuales del proyecto.

2.2 Desglose y descripción de tareas

Planificación (P):

P.1. Identificación de los requisitos y las decisiones principales.
P.2. Realizar una primera planificación.
P.3. Actualizar la planificación en caso de ser necesario.

Estudio herramientas (EH):

EH.1. Estudio de las herramientas de organización: *Jira* (gestión), *GitLab* (versionado del código) y *Jenkins* (integración continua).
EH.2. Estudio de las herramientas de desarrollo:
- **EH.2.1.** *Rest.*
- **EH.2.2.** *React.*
- **EH.2.3.** *JUnit*.
- **EH.2.4.** *Cucumber* y *Selenium.*
- **EH.2.5.** *Karate*.

Control y continuidad (CC):

CC.1. Recoger la información respecto al desarrollo del proyecto.
CC.2. Comparación del trabajo realizado con la planificación, observar los desvíos importantes e identificación de posibles peligros.
CC.3. Reunión con el director del proyecto.
CC.4. Metodología de Scrum master (daily, spring planning, refinement…).

Memoria (M): Desarrollo de la documentación del proyecto.

Análisis de requisitos del sistema (ARS): Analizar los requisitos necesarios del sistema para el correcto desarrollo de la aplicación.

Estudio modelo de datos (EMD): Análisis los modelos de datos obtenidos en las tablas maestras.

Desarrollo API (DA):

- DA.1. Preparar entorno de desarrollo de la API.
- DA.2. Preparar entorno Gitlab para el versionado del proyecto.
- DA.3. Preparar entorno Jenkins para levantamiento de proyecto en servidor.
- DA.4. Crear esquema y tablas de Base de Datos.
- DA.5. Desarrollo de la API.

Generar test unitarios (GTU):

- GTU.1. Preparar entorno para desarrollo de test.
- GTU.2. Generar Docker para levantamiento de SonarQube local.
- GTU.3. Preparar entorno SonarQube antes del levantamiento de proyecto en servidor.
- GTU.4. Desarrollo de test unitarios.

Estudio de la interfaz (EI): Análisis del estilo web IK-WI-4 (estilo IKUSI).

Desarrollo de interfaz (DI):

- DI.1. Preparar entorno de desarrollo de la Web.
- DI.2. Preparar entorno Gitlab para el versionado del proyecto.
- DI.3. Preparar entorno Jenkins para levantamiento de proyecto en servidor.
- DI.4. Desarrollo de la Web.

Generar test de interfaz de usuario (GTIU):

- GTIU.1. Preparar entorno para desarrollo de test de interfaz de usuario.
- GTIU.2. Generar Docker para levantamiento de SonarQube en local.
- GTIU.3. Preparar entorno SonarQube antes del levantamiento de proyecto en servidor.
- GTIU.4. Desarrollo de test de interfaz de usuario.

Generar test de integración (GTI):

- GTI.1. Preparar entorno para desarrollo de test de integración.
- GTI.2. Generar Docker para levantamiento de SonarQube en local.
- GTI.3. Preparar entorno SonarQube antes del levantamiento de proyecto en servidor.
- GTI.4. Desarrollo de test automatizados y de integración.

Documentación de manuales (DM):

- DM.1. Realizar manual de instalación.
- DM.2. Realizar manual de configuración.
- DM.3. Realizar manual de usuario.

2.2.1 Dependencia entre tareas

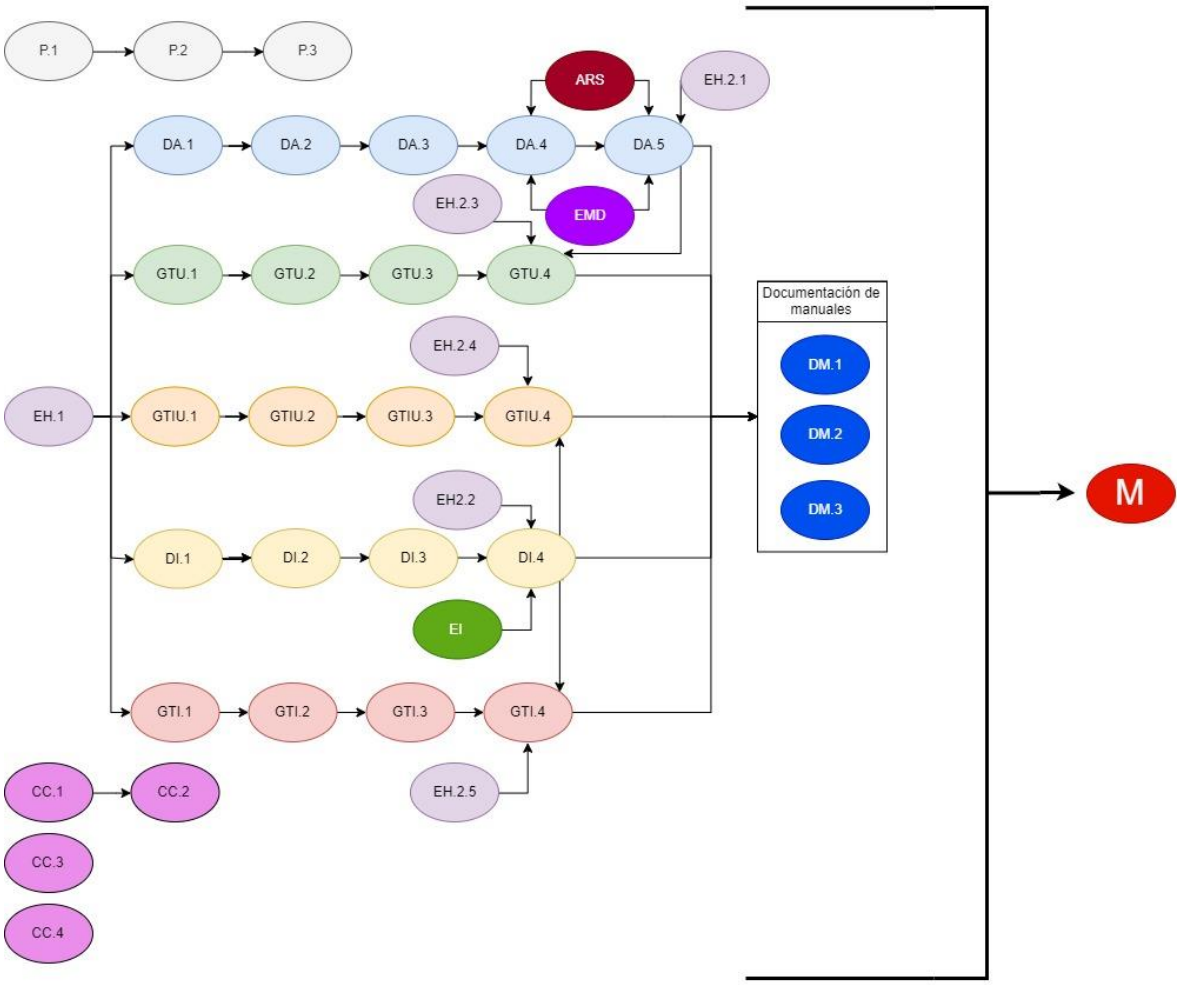


Ilustración 3 – Dependencia entre tareas

2.3 Estimación de las tareas

Tareas	Estimación en horas
P - Planificación	9
P.1 - Presentación proyecto	2
P.2 - Planificación	5
P.3 - Revisión planificación	2
EH - Estudio de herramientas	19
EH.1 - Estudio herramientas organización	1
EH.2.1 - Estudio Rest	2
EH.2.2 - Estudio React	5
EH.2.3 - Estudio Junit	1
EH.2.4 - Estudio Cucumber y Selenium	5
EH.2.5 - Estudio Karate	5
CC - Control y continuidad	38
CC.1 - Revisión trabajo	4

CC.2 - Comparación trabajo	4
CC.3 - Reunión con el tutor	10
CC.4 - Spring planning	20
ARS - Análisis de requisitos	10
EMD - Estudio de datos maestros	2
DA- Desarrollo API	40
DA.1 - Preparar entorno desarrollo API	2
DA.2 - Entorno GitLab	2
DA.3 - Preparar entorno Jenkins	2
DA.4 - Crear Base de Datos	4
DA.5 - Desarrollo de la API	30
GTU - Generar test unitarios	34
GTU.1 - Preparar entorno desarrollo test	2
GTU.2 - Generar Docker SonarQube local	1
GTU.3 - Preparar SonarQube en servidor	1
GTU.4 - Desarrollo de test unitarios	30
EI - Estudio del estilo IK-WI-4	2
DI - Desarrollo interfaz	34
DI.1 - Preparar entorno desarrollo web	2
DI.2 - Preparar entorno Gitlab	1
DI.3 - Preparar entorno Jenkins	1
DI.4 - Desarrollo de la web	30
GTIU - Generar test de interfaz de usuario	28
GTIU.1 - Entorno para test de interfaz	1
GTIU.2 - Generar Docker SonarQube local	1
GTIU.3 - Preparar SonarQube en servidor	1
GTIU.4 - Desarrollo de test de interfaz	25
GTI - Generar test de integración	24
GTI.1 - Entorno para test automatizados	2
GTI.2 - Generar Docker SonarQube local	1
GTI.3 - Preparar SonarQube en servidor	1
GTI.4 - Desarrollo de test automatizados	20
DM - Documentación de manuales	10
DM.1 - Realizar manual de instalación	5
DM.2 - Realizar manual de configuración	3
DM.3 - Realizar manual de usuario	2
M - Memoria del proyecto	50
TOTAL	300

Tabla 1 - Estimación de las tareas

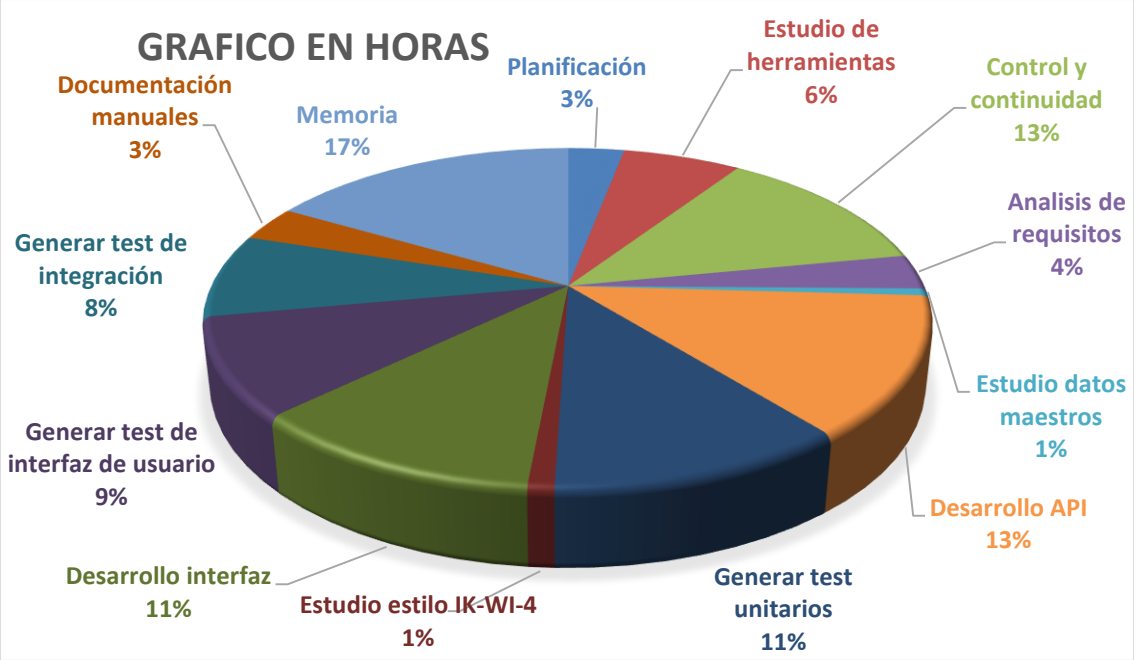


Ilustración 4 – Grafico estimación en horas

2.4 Límites de entrega

Tareas	Límites de entrega
P - Planificación	06/08/2019
ARS - Análisis de requisitos	09/08/2019
EMD - Estudio de datos maestros	09/08/2019
DA- Desarrollo API	13/08/2019
GTU - Generar test unitarios	13/08/2019
DI - Desarrollo interfaz	06/12/2019
GTIU - Generar test de interfaz de usuario	13/12/2019
GTI - Generar test de integración	10/01/2019
DM - Documentación de manuales	17/01/2019
M - Memoria del proyecto	20/01/2019

Tabla 2 - Límites de entrega

2.5 Gestión de riesgos

1. Perder los archivos.
2. Caída del servidor.
3. Retraso en las entregas.
4. Base de datos dañada o caída.
5. Planificación demasiado optimista.
6. Diseño inadecuado (hay que volver a diseñar).
7. Eliminar o añadir nuevas características.
8. Falta de permisos.
9. Código de baja calidad.
10. No cumplir los plazos de entrega.

Probabilidad / Consecuencia	15%	30%	45%	60%	75%	90%
Bajo		3	5			
Mediano				9	8	7
Alto	4, 10		2			
Muy alto	6	1				

Tabla 3 - Gestión de riesgos

Para poder prevenir los riesgos arriba definidos, y en caso de ocurrir, afrontarlos obteniendo el mínimo daño posible, se van a seguir varios criterios. La **pérdida de archivos** puede llegar a ser muy peligrosa, por ello, siempre se va a tener un back-up de los archivos, se almacenarán dichos archivos en el ordenador de trabajo, en el Google Drive y en el ordenador personal. Dicho back-up se realizará al finalizar la jornada laboral, en caso de escribir un documento extenso o desarrollar parte importante del código.

En caso de **caída del servidor** o que la **Base de Datos caiga o resulte dañada**, es un riesgo poco probable que ocasionaría un problema a la hora del proceso del desarrollo de la aplicación, el cual se podría solucionar teniendo siempre alguna parte de documentación por realizar hasta la solución de dicho riesgo.

El **retraso en las entregas**, es un riesgo que no tendría demasiadas consecuencias en este proyecto debido a que es un proyecto individual. También, el realizar una primera **planificación demasiado optimista** es otro riesgo muy probable que se solucionará una vez iniciado el desarrollo del proyecto y viendo las complicaciones que se pueden encontrar.

El riesgo que existe en realizar un **diseño inadecuado** es poco probable, pero de muy grandes consecuencias, puesto que rediseñar sería muy costoso y daría pie a cometer fallos a la hora del desarrollo. Por lo tanto, para evitar este riesgo, lo ideal sería obtener una idea bastante firme con lo que se desea, y posteriormente, según se avance en el proyecto ir viendo si es un diseño adecuado o no.

El riesgo de que tras haber iniciado el desarrollo se **eliminen o añadan nuevas características**, es bastante probable, debido que al ir viendo el desarrollo se va perfeccionando la idea del proyecto. Este riesgo tiene un abanico muy amplio de consecuencias, desde no afectar en la aplicación, hasta el cambiar alguna parte importante y afectar en el comportamiento de la aplicación y tener que actualizar la planificación y plazos de entrega.

El riesgo de la **falta de permisos** para acceder a ciertas partes es un riesgo muy probable debido a la política de la empresa, y los retrasos que pueden conllevar este riesgo son medianos, ya que puede retrasar el desarrollo del proyecto un par de horas hasta conseguir los permisos necesarios.

El riesgo de **código de baja calidad** es un riesgo poco probable de suceder, debido a que es un desarrollo iterativo e incremental, que de este modo cada iteración deberá de haber pasado satisfactoriamente las pruebas de calidad establecidas. De este modo que la nueva iteración no pase el control de calidad a la primera, no tiene una gran consecuencia.

El **no cumplir los plazos de entrega**, es un riesgo que existe en todo momento puesto que depende del día a día y de posibles fallos y retrasos establecidos en los textos superiores. Las consecuencias serían atrasos en el resto de tareas por realizar.

2.6 Recursos

Siempre y cuando sea posible, se usarán herramientas libres o herramientas con versiones gratuitas para evitar cualquier tipo de conflicto con licencias. El ordenador será suministrado por la empresa y la versión de Windows que usará el ordenador será Windows 10.

Para la parte de servidores, se usará un servidor propio de la empresa IKUSI en el cual se subirá el proyecto una vez terminado.

2.7 Interesados

El interesado principal de la finalización del proyecto de manera exitosa es el propio autor de esta memoria. Es suya la responsabilidad de que el proyecto se desarrolle correctamente.

Otros interesados serian tanto el tutor como el responsable de la empresa. El tutor, Jon Iturrioz, por ser responsabilidad suya la explicación y buena orientación del proyecto.

Para finalizar, los últimos interesados el jefe de producto, Alain Urbeltz, por ser quien desea tener terminado el proyecto cuanto antes para su uso, y el responsable del equipo (*Scrum)*, Jokin Santodomingo, por ser responsabilidad de él la explicación correcta del proyecto.

Capítulo 3

Análisis de requisitos

El objetivo es crear una API y una interfaz web, la cual maneje ciertos datos maestros del sector aeroportuario, almacenados en una Base de Datos. Principalmente el servicio REST que ofrece la API lo usara el cliente, y la parte de interfaz web principalmente es para el administrador.

Un usuario no registrado, podrá hacer peticiones a la API, siempre y cuando tenga acceso a este host el cual será privado de la empresa IKUSI y no cualquier persona podrá tener acceso.

Situándonos en el papel de cliente:

- Tendrá acceso a los datos que el necesite siempre y cuando sea un cliente previamente dado de alta en el sistema.
- De los datos almacenados, el cliente podrá descargar todos los datos o solo los que a él le interese.
- El cliente recibirá una notificación desde la API cuando los datos almacenados en la Base de Datos se actualicen, se borren los datos existentes o se inserten nuevos datos.

Cambiando al papel del administrador:

- Podrá realizar una carga inicial desde un archivo *CSV*, que tendrá un formato fijo.
- Crear, leer, actualizar o borrar uno o varios datos desde la interfaz web o desde un archivo *CSV.*
- Podrá gestionar desde la interfaz web cada usuario que este en el sistema pudiendo dar de alta, dar de baja o modificarlos.

3.1 Requisitos funcionales: Casos de uso

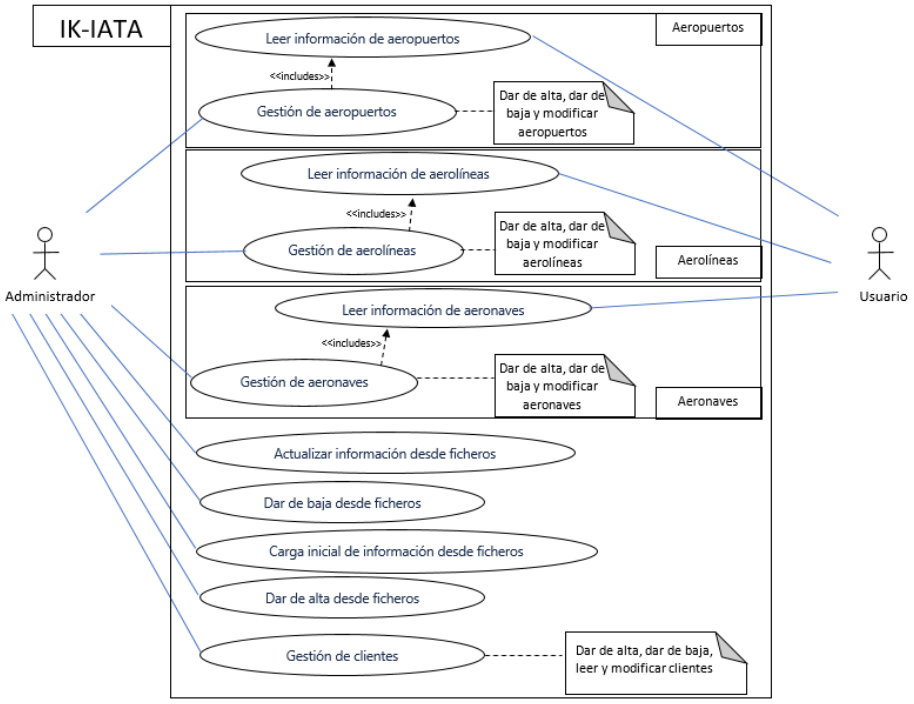


Ilustración 5 – Casos de uso

Descripción

Leer información de aeropuertos: Como usuario logeado o administrador poder leer la información de uno, varios o todos los aeropuertos existentes.

Gestión de aeropuertos: Únicamente como administrador, dar de alta, dar de baja o modificar datos de uno o varios aeropuertos desde la interfaz.

Leer información de aerolíneas: Como usuario logeado o administrador poder leer la información de una, varias o todas las aerolíneas existentes.

Gestión de aerolíneas: Únicamente como administrador, dar de alta, dar de baja o modificar datos de uno o varias aerolíneas desde la interfaz.

Leer información de aeronaves: Como usuario logeado o administrador poder leer la información de una, varias o todas las aeronaves existentes.

Gestión de aeronaves: Únicamente como administrador, dar de alta, dar de baja o modificar datos de una o varias aeronaves desde la interfaz.

Actualizar información desde ficheros: Como administrador poder actualizar información de uno o varios de los modelos existentes (aeropuerto, aerolínea o aeronave) desde fichero en formato CSV.

Dar de baja desde fichero: Como administrador poder dar de baja uno o varios de los modelos existentes desde fichero CSV.

Carga inicial de información desde ficheros: Como administrador poder realizar una carga inicial de cada modelo de datos existentes desde fichero CSV.

Dar de alta desde fichero: Como administrador poder crear uno o varios de los modelos existentes desde fichero CSV.

Gestión de clientes: Como administrador poder realizar operaciones *CLAB* (Crear, Leer, Actualizar o Modificar) clientes.

3.1.1 Flujo de eventos

3.1.1.1 Usuario

Leer información de aeropuertos

- Flujo normal:
    1. El usuario desde su sistema de AMS solicita la información de uno o varios aeropuertos mediante el código Iata o Icao.
    2. El sistema busca el o los aeropuertos solicitados.
    3. El sistema devuelve la información.
- Flujo alternativo:
    - El código Iata o Icao no existe, por lo tanto, no devuelve nada.
    - De la lista de códigos Iata o Icao enviados, si uno o varios no existen, devuelve solo los aeropuertos existentes.
    - De la lista de códigos Iata o Icao enviados, si no existe ninguno, no devuelve nada.

Leer información de aerolíneas

- Flujo normal:
    1. El usuario desde su sistema de AMS solicita la información de una o varias aerolíneas mediante el código Iata o Icao.
    2. El sistema busca la o las aerolíneas solicitadas.
    3. El sistema devuelve la información.
- Flujo alternativo:
    - El código Iata o Icao no existe, por lo tanto, no devuelve nada.
    - De la lista de códigos Iata o Icao enviados, si uno o varios no existen, devuelve solo las aerolíneas existentes.
    - De la lista de códigos Iata o Icao enviados, si no existe ninguno, no devuelve nada.

Leer información de aeronaves

- Flujo normal:
    1. El usuario desde su sistema de AMS solicita la información de una o varias aeronaves mediante el código Iata o Icao.
    2. El sistema busca la o las aeronaves solicitadas.
    3. El sistema devuelve la información.
- Flujo alternativo:
    - El código Iata o Icao no existe, por lo tanto, no devuelve nada.
    - De la lista de códigos Iata o Icao enviados, si uno o varios no existen, devuelve solo las aeronaves existentes.
    - De la lista de códigos Iata o Icao enviados, si no existe ninguno, no devuelve nada.

3.1.1.2 Administrador

Gestión de aeropuertos: Dar de alta

- Flujo normal:
    1. El administrador inserta la información (código Iata, código Icao, nombre, código del país, dst, latitud, longitud, zona horaria y si está activo o inactivo) del aeropuerto nuevo.
    2. El sistema recoge la información introducida y lo pasa por varias validaciones.
    3. El sistema guarda los datos nuevos introducidos y devuelve un OK con el aeropuerto introducido.

- Flujo alternativo:
  - Si los datos introducidos son incorrectos o no pasa alguna de las validaciones, devuelve un error de validación.
  - Si aun siendo correctos los datos, el código Iata o Icao está duplicado, devuelve un error de duplicidad.

Gestión de aeropuertos: Dar de baja

- Flujo normal:
  1. El administrador inserta el Id, el código Iata y/o el código Icao del aeropuerto que desea borrar.
  2. El sistema recoge el dato y comprueba que existe el aeropuerto.
  3. El sistema da de baja el aeropuerto y no devuelve nada.
- Flujo alternativo:
  - Si los datos introducidos no existen devuelve un error de aeropuerto no encontrado.
  - Si los datos introducidos no concuerdan entre sí (por ejemplo, introduciendo el id y el código Iata, para el id introducido el Iata que se obtiene es distinto al introducido), devuelve un error de aeropuerto no encontrado.

Gestión de aeropuertos: Modificar

- Flujo normal:
  1. El administrador selecciona un aeropuerto que desea modificar.
  2. El administrador modifica los datos que el vea necesarios.
  3. El sistema recoge los datos modificados y pasa varias validaciones de los datos introducidos.
  4. El sistema modifica los datos nuevos introducidos y devuelve un OK.
- Flujo alternativo:
  - Si los datos introducidos son incorrectos o no pasa alguna de las validaciones, devuelve un error de validación.
  - Si aun siendo correctos los datos, el código Iata o Icao esta duplicado, devuelve un error de duplicidad.

Gestión de aerolíneas: Dar de alta

- Flujo normal:
  1. El administrador inserta la información (código Iata, código Icao, nombre, código del país, activa o inactiva y tipo) de la aerolínea nueva.
  2. El sistema recoge la información introducida y lo pasa por varias validaciones.
  3. El sistema guarda los datos nuevos introducidos y devuelve un OK con la aerolínea introducida.
- Flujo alternativo:
  - Si los datos introducidos son incorrectos o no pasa alguna de las validaciones, devuelve un error de validación.
  - Si aun siendo correctos los datos, el código Iata o Icao este duplicado siempre y cuando la duplicidad ocurra con otra aerolínea que en su estado este activa, devuelve un error de duplicidad.

Gestión de aerolíneas: Dar de baja

- Flujo normal:
  1. El administrador inserta el id, el código Iata y/o el código Icao de la aerolínea que desea borrar.
  2. El sistema recoge el dato y comprueba que existe la aerolínea y está activa.
  3. El sistema da de baja la aerolínea y no devuelve nada.

- Flujo alternativo:
  - Si los datos introducidos no existen devuelve un error de aerolínea no encontrada.
  - Si los datos introducidos no concuerdan entre sí (por ejemplo, introduciendo el id y el código Iata, para el id introducido el Iata que se obtiene es distinto al introducido), devuelve un error de aerolínea no encontrada.

Gestión de aerolíneas: Modificar

- Flujo normal:
  1. El administrador selecciona una aerolínea que desea modificar.
  2. El administrador modifica los datos que el vea necesarios.
  3. El sistema recoge los datos modificados y pasa varias validaciones de los datos introducidos.
  4. El sistema modifica los datos nuevos introducidos y devuelve un OK con la aerolínea modificada.
- Flujo alternativo:
  - Si los datos introducidos son incorrectos o no pasa alguna de las validaciones, devuelve un error de validación.
  - Si aun siendo correctos los datos, el código Iata o Icao están duplicados siempre y cuando la duplicidad ocurra con otra aerolínea que en su estado este activa, devuelve un error de duplicidad.

Gestión de aeronaves: Dar de alta

- Flujo normal:
  1. El administrador inserta la información (código Iata, código Icao, nombre, longitud, largura, ancho y si está activo o inactivo) de la aeronave nueva.
  2. El sistema recoge la información introducida y lo pasa por varias validaciones.
  3. El sistema guarda los datos nuevos introducidos y devuelve un OK con la aeronave introducida.
- Flujo alternativo:
  - Si los datos introducidos son incorrectos o no pasa alguna de las validaciones, devuelve un error de validación.
  - Si aun siendo correctos los datos, el código Iata este duplicado, devuelve un error de duplicidad. (En este caso, puede existir un grupo de aeronaves que tengan el mismo código Icao, pero diferentes códigos Iata).
  - Si aun siendo los datos correctos, el código Iata y el código Icao están duplicados, devuelve un error de duplicidad.

Gestión de aeronaves: Dar de baja

- Flujo normal:
  1. El administrador inserta el id, el código Iata y/o el código Icao de la aeronave que desea borrar.
  2. El sistema recoge el dato y comprueba que existe la aeronave.
  3. El sistema da de baja la aeronave y no devuelve nada.
- Flujo alternativo:
  - Si los datos introducidos no existen devuelve un error de aeronave no encontrada.
  - Si los datos introducidos no concuerdan entre sí (por ejemplo, introduciendo el id y el código Iata, para el id introducido el Iata que se obtiene es distinto al introducido), devuelve un error de aeronave no encontrada.

Gestión de aeronaves: Modificar

- Flujo normal:
    1. El administrador selecciona una aeronave que desea modificar.
    2. El administrador modifica los datos que el vea necesarios.
    3. El sistema recoge los datos modificados y pasa varias validaciones de los datos introducidos.
    4. El sistema modifica los datos nuevos introducidos y devuelve un OK con la aeronave modificada.
- Flujo alternativo:
    - Si los datos introducidos son incorrectos o no pasa alguna de las validaciones, devuelve un error de validación.
    - Si aun siendo correctos los datos, el código Iata o Icao esta duplicado, devuelve un error de duplicidad.

Actualizar información desde ficheros

- Flujo normal:
    1. El administrador carga un fichero CSV sobre un modelo concreto (aeropuerto, aerolínea o aeronave) indicando que atributos deben ser actualizados.
    2. El sistema recibe el fichero y lo procesa uno a uno.
    3. Cada línea de información recibida pasa, dependiendo que modelo de datos sea, para unas validaciones específicas.
    4. El sistema devuelve las líneas que se han procesado correctamente.
- Flujo alternativo:
    - Si se ha introducido un atributo incorrecto, el sistema detecta que es un atributo erróneo y pasa al siguiente devolviendo al final una lista de líneas que se han actualizado correctamente y otras que no.
    - Si se intenta actualizar un código Iata y/o Icao, y previamente existentes en el sistema, pasa al siguiente devolviendo al final una lista de líneas que se han actualizado correctamente y otras que no.

Dar de baja desde ficheros

- Flujo normal:
    1. El administrador carga un fichero CSV con los códigos Iata y/o códigos Icao del modelo de objeto que desea borrar.
    2. El sistema recibe el fichero y lo procesa.
    3. Cada línea que procesa, pasa una verificación específica para cada modelo.
    4. El sistema da de baja los objetos del modelo seleccionado.
- Flujo alternativo:
    - Si los códigos introducidos no existen, devuelve un error de objeto no encontrado.
    - Si los códigos introducidos no concuerdan entre sí, devuelve un error de objeto no encontrado.

Carga inicial de información desde ficheros

- Flujo normal:
    1. El administrador desea realizar una primera y una carga de información cuando aún no existe ningún dato y carga un fichero CSV con toda la información de cada modelo.
    2. El sistema procesa las líneas una a una.
    3. El sistema pasa unas validaciones varias específicas para cada modelo.
    4. El sistema devuelve una lista con las líneas que se han cargado correctamente.

- Flujo alternativo:
  - Si alguna de las líneas procesadas no pasa alguna validación, se pasa a la siguiente marcando en la lista que se devuelve al final no línea no correcta.
  - Si se intenta cargar un dato que previamente existe (código Iata y/o Icao duplicado) genera un error, marcando como error esa línea y devolviéndola al final en la lista de líneas erróneas.

Dar de alta desde ficheros

- Flujo normal:
  1. El administrador carga un fichero CSV de estructura predefinida con toda la información que se desea dar de alta de un modelo en concreto: aeropuerto, aerolínea o aeronave.
  2. El sistema recibe el fichero y lo procesa uno a uno.
  3. Cada línea de información recibida pasa, dependiendo que modelo de datos sea, para unas validaciones específicas.
  4. El sistema devuelve las líneas que se han procesado correctamente.
- Flujo alternativo:
  - Si se intenta dar de alta un objeto previamente ya existente, pasa al siguiente devolviendo al final una lista de líneas que se han dado de alta correctamente y otras que no.
  - Si se intenta dar de alta con un código Iata o Icao y previamente existentes en el sistema, pasa al siguiente devolviendo al final una lista de líneas que se han actualizado correctamente y otras que no.
- Flujo excepcional (Caso de aeronave):
  - Cuando se intenta dar de alta con código Iata existente, pasa al siguiente devolviendo al final una lista de líneas que se han actualizado correctamente y otras que no.
  - Cuando se intenta dar de alta con código Iata y código Icao ya existentes, pasa al siguiente devolviendo al final una lista de líneas que se han actualizado correctamente y otras que no.

Gestión de clientes: Dar de alta

- Flujo normal:
  1. El administrado introduce los datos de un cliente en el sistema.
  2. El sistema comprueba y verifica que los datos son correctos.
  3. El sistema crea el usuario y devuelve un OK y los datos del cliente recién creado.
- Flujo alternativo:
  - Si el cliente ya existe previamente y no está dado de baja, devuelve un erro de duplicidad.
  - Si el cliente ya existe previamente y está dado de baja, se da de alta el cliente.
  - Si los datos introducidos son incorrectos o no pasan alguna verificación, devuelve un error.

Gestión de clientes: Dar de baja

- Flujo normal:
  1. El administrador introduce el cliente que desea dar de baja.
  2. El sistema procesa y válida la petición.
  3. El sistema da de baja el usuario seleccionado.
- Flujo alternativo:
  - Si el cliente introducido no existe, devuelve un error de cliente no encontrado.

Gestión de clientes: Leer

- Flujo normal:
    1. El administrador selecciona un cliente.
    2. El sistema recibe esa selección y devuelve el usuario solicitado.
- Flujo alternativo:
    - Si se solicita información de un cliente que no existe, devuelve un error de cliente no encontrado.

Gestión de clientes: Modificar

- Flujo normal:
    1. El administrador selecciona un cliente.
    2. El sistema carga los datos del cliente seleccionado.
    3. El administrador cambia los datos que el necesite cambiar.
    4. El sistema recoge la información modificada y pasa unas verificaciones.
    5. El sistema actualiza los datos del cliente y devuelve un OK y el cliente actualizado.
- Flujo alternativo:
    - Si los datos actualizados no pasan alguna validación, el sistema devuelve un error de validación.

3.2 Requisitos de datos: Modelo de dominio

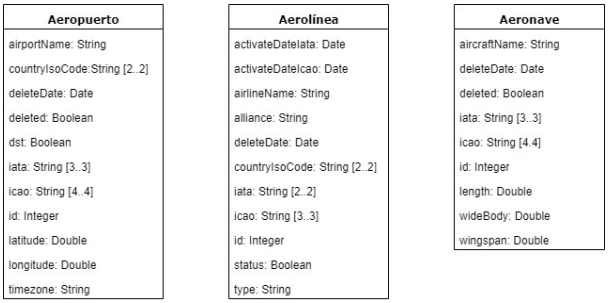


Tabla 4 - Modelo de dominio

En las 3 tablas de arriba podemos observar cuales van a ser las entidades y los atributos de cada entidad. En la tabla de *Aeropuerto* encontramos los siguientes datos:

- **airportName:** Es el nombre del aeropuerto.
- **countryIsoCode:** Es el código ISO del país.
- **deleteDate:** Fecha de cuando se da de baja el aeropuerto.
- **deleted:** Indica si el aeropuerto esta dado de baja.
- **dst:** Indica si el país en el cual se encuentra el aeropuerto sufre cambio horario durante el año.
- **iata:** Código *único* del aeropuerto formado por 3 caracteres.
- **icao:** Código *único* del aeropuerto formado por 4 caracteres.
- **id:** Identificador *único* autogenerado.
- **latitude:** Latitud a la que se encuentra el aeropuerto.
- **longitude:** Longitud a la que se encuentra el aeropuerto.
- **timezone:** Zona horaria del país en la que se encuentra el aeropuerto.

En la siguiente tabla de *Aerolínea* encontramos los siguientes datos:

- activateDateIata: Fecha de activación del código Iata.
- activateDateIcao: Fecha de activación del código Icao.
- airlineName: Es el nombre de la aerolínea.
- alliance: Alianza a la que pertenece la aerolínea.
- deleteDate: Fecha de cuando se da de baja la aerolínea.
- countryIsoCode: Es el código ISO del país.
- iata: Código *único* de la aerolínea formado por 2 caracteres.
- icao: Código *único* de la aerolínea formado por 3 caracteres.
- id: Identificador *único* autogenerado.
- status: Estado actual de la aerolínea, es decir, si esta activa o inactiva.
- type: Tipos de vuelos que realiza la aerolínea, regulares o chárter.

Para finalizar, en la tabla de *Aeronave* encontramos los siguientes datos:

- aircraftName: Es el nombre de la aeronave.
- deleteDate: Fecha de cuando se da de baja la aerolínea.
- deleted: Indica si la aeronave está dada de baja.
- iata: Código *único* del aeropuerto formado por 3 caracteres.
- icao: Código *único* del aeropuerto formado por 4 caracteres.
- id: Identificador *único* autogenerado.
- length: Largura de la aeronave.
- wideBody: Ancho del cuerpo de la aeronave.
- wingspan: Envergadura de la aeronave.

3.3 Requisitos no funcionales

En este apartado, se van a explicar requisitos, que, aunque no modifiquen la funcionalidad del producto, si imponen ciertas restricciones. No son una parte fundamental del producto, pero son necesarios para hacer funcionar el producto y describir la experiencia del usuario.

3.3.1 Requisitos de apariencia

Uno de los requisitos principales a la hora de desarrollar la interfaz, es desarrollar dicha interfaz siguiendo el estilo IK-WI-4, un estilo propio de la empresa IKUSI que actualmente se usa como estilo común de todas las aplicaciones y sistemas desarrollados por la empresa, por lo tanto, en este proyecto es una exigencia de la parte de la interfaz.

3.3.2 Requisitos de usabilidad

Otro de los requisitos principales es la usabilidad del producto, es decir, por una parte, el desarrollo de manuales de usuario y manuales para la instalación y configuración, y, por otra parte, conseguir que la parte de la API y la interfaz sean completamente independientes, es decir, que sea de diseño *front end* y *back end*. En diseño de software, el termino *front end* se usa para la parte de software que interactúa con los usuarios, y el termino *back end*, se usa para la parte de software que es el motor del sistema. Gracias a esta separación obtenemos una abstracción entre la capa de interfaz y la capa de acceso de datos la cual nos ayuda en la usabilidad del producto.

3.3.3 Requisitos de mantenibilidad

Otro requisito fundamental es conseguir la mayor mantenibilidad del producto para en un futuro poder seguir mejorándolo. Esta mantenibilidad la conseguiremos gracias a la capa de abstracción entre la interfaz y la API, y también gracias a no mezclar la lógica de negocio de la API.

3.3.4 Requisitos legales

Al ser un producto que la empresa IKUSI va a comercializar, tiene que desarrollarse de manera totalmente legal a la hora del uso de herramientas externas, estas herramientas deben ser gratuitas, tener una parte de la aplicación gratuita con la cual poder trabajar, o se debe de obtener la licencia para su uso y explotación.

3.3.5 Requisitos de calidad

A continuación, definimos los objetivos de calidad que debería de superar el producto:

1. Procesamiento del servidor:
   - Descripción: Fluidez con la cual es servidor procesa los datos.
   - Modo de medición: Calculando el tiempo de procesado del servidor.
   - Máximo aceptable: 10 segundos para subir archivos.
   - Objetivo: Menos de 5 segundos.
2. Fiabilidad:
   - Descripción: Funcionar correctamente y como se espera de cada uno de los casos de uso de los usuarios.
   - Modo de medición: Mediante los test realizados y pruebas con usuarios.
   - Mínimo aceptable: Que el 90% de peticiones funcionen.
   - Objetivo: Que el 100% de peticiones funcionen.
3. Claridad del código
   - Descripción: Implementar el código de manera limpia y organizada, de manera, que si en un futuro otro programador lee el código sea capaz de entenderlo sin problema.
   - Modo de medición: Utilizando nombres lógicos en variables y métodos y de manera ordenada siguiendo una metodología de implementación.
   - Mínimo aceptable: Usar anotaciones o comentarios pequeños para aclaraciones
   - Objetivo: Implementar de manera lógica y ordenada.
4. Diseño de interfaz
   - Descripción: Lograr una interfaz funcional, organizada e intuitiva para el usuario.
   - Modo de medición: Probando con usuarios los clics realizados y tiempo necesitado para la ejecución de algunas funciones.
   - Mínimo aceptable: alguna parte sea menos intuitiva.
   - Objetivo: Una interfaz perfectamente usable por cualquier usuario.
5. Calidad de la memoria
   - Descripción: Realizar una memoria del proyecto de manera ordenada y de fácil comprensión, sin ser muy extensa.
   - Modo de medición: Realizando la documentación de cada apartado en su momento exacto.
   - Mínimo aceptable: Seguir la estructura de memoria definida al principio.
   - Objetivo: A parte de seguir la estructura, que sea clara y precisa.

Capítulo 4

Diseño

En este apartado se va a hablar del diseño del proyecto, es decir, se crea una perspectiva de cómo va a ser el producto, y se especifica cómo se va a interactuar con él.

4.1 Arquitectura

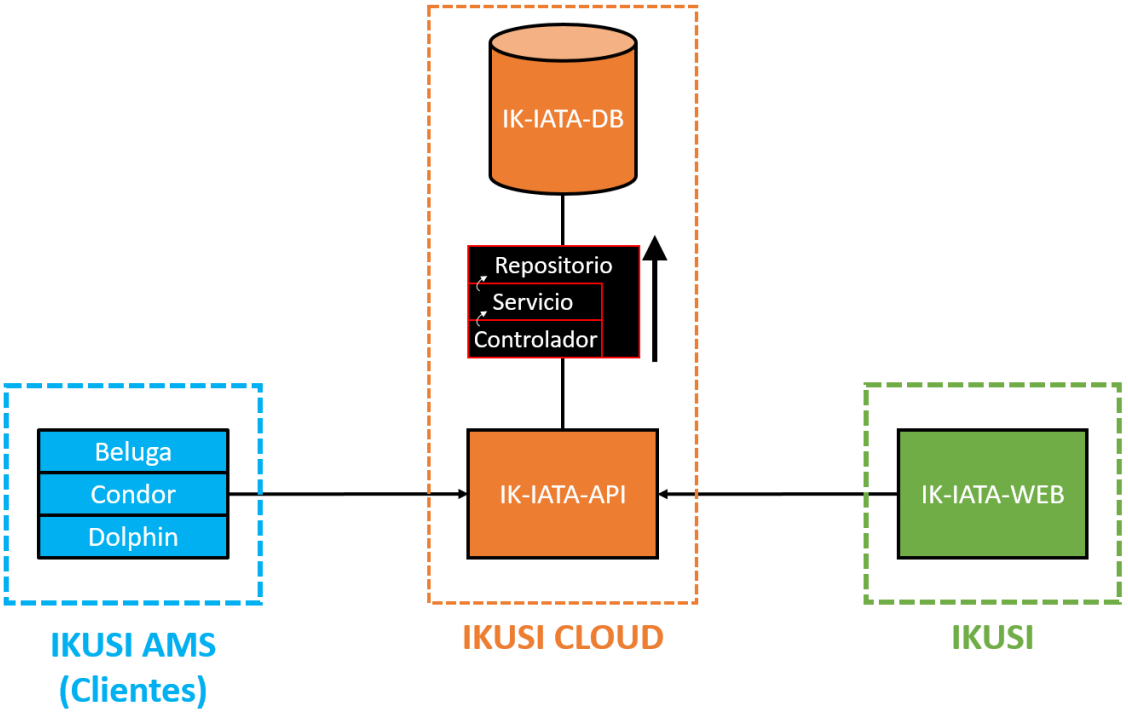


Ilustración 6 – Arquitectura

En la imagen de arriba podemos observar cuál sería la arquitectura del proyecto.

Por una parte, a la izquierda en el cuadrado azul podemos observar cuales serían los clientes principales, es decir, las aplicaciones de IKUSI AMS, las cuales estarían instaladas en sus aeropuertos correspondientes con usuarios muy distintos. Debido a esta variedad de usuarios, el sistema debe estar internacionalizado; principalmente se deberá de desarrollar en Ingles, pero dejando la posible opción, de manera sencilla y sin tener que realizar apenas cambios, la posibilidad de adaptación del sistema al idioma que usuario final desee.

Por otro lado, en el medio en el cuadrado naranja, podemos ver la parte que sería el motor de la aplicación, como antes se ha comentado, la parte del back end que es la única que tendría acceso a la base de datos. También podemos ver cuál sería la arquitectura que va a tener el back end: *Arquitectura de microservicio*. Esta arquitectura contiene ciertas características que nos benefician a la hora de obtener la mayor abstracción, el sistema se divide en distintos servicios y es muy importante limitar la responsabilidad de cada servicio; las aplicaciones de microservicios poseen su propia lógica: reciben una solicitud, aplican la lógica y produce una respuesta; tienen una gestión de datos descentralizada de modo que cada servicio gestiona su propia base de datos; y también necesitan un diseño tolerante a fallos, es decir, al ser un

microservicio puede recibir llamadas de diferentes servicios que pueden realizar de forma incorrecta las peticiones y el microservicio necesita dar una respuesta rápida ante el fallo.

Para finalizar, al lado derecho en el cuadrado verde, podemos ver la parte de la interfaz web, es decir, front end. Esta parte seria especialmente creada para la correcta utilización del motor de la aplicación con un diseño especial de la empresa IKUSI.

4.2 Modelo de Base de Datos

Como previamente en el punto *3.2. Requisitos de datos: Modelo de dominio* se ha mostrado, tenemos 3 tablas diferentes.

Para el desarrollo y conexión con la Base de Datos se ha utilizado un complemento de *Maven* llamado *Flyway*, que en la sección de herramientas definiros mejor.

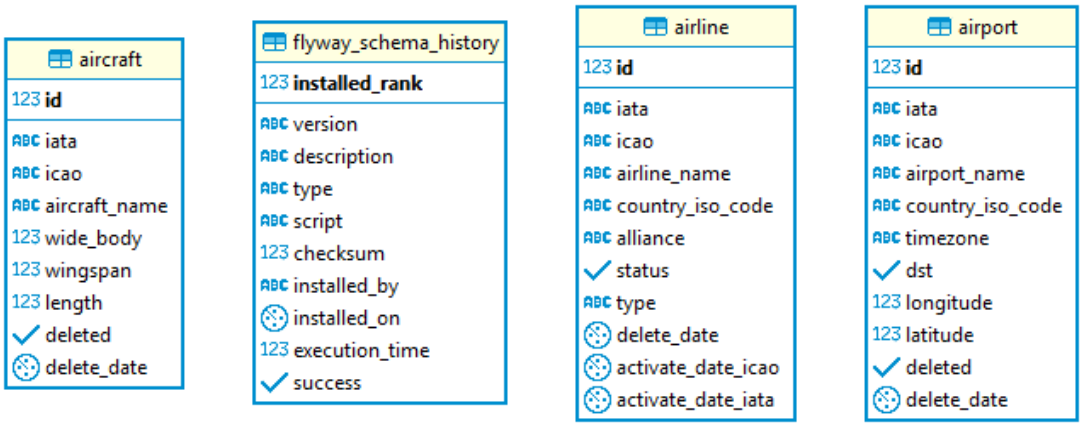


Tabla 5 - Modelo de Base de Datos

Como podemos observar en la imagen de arriba, Flyway crea una tabla gracias a la cual se puede mantener un control de versiones de esquema, viendo cuando se ha realizado cada modificación y de este modo, en caso de existir errores identificarlos rápido.

Por otra parte, en cada tabla se han establecido una serie de triggers (procedimientos que se ejecutan tras un evento en la Base de Datos) para mayor seguridad respecto a las fechas de borrado y activación. Para ello, en el propio fichero de Flyway se han generado estos triggers para evitar que se solapen las fechas entre otras cosas, por ejemplo, en caso de aerolíneas (tabla *airline*) puede ocurrir que una aerolínea pueda reutilizar el código Iata de otra aerolínea previamente dada de baja, por ello, es importante que la fecha de la aerolínea dada de baja sea anterior a la fecha de activación de la nueva aerolínea para evitar conflictos.

4.3 Diseño del Back end

En los siguientes puntos realizaremos un diagrama de secuencia con su explicación correspondiente de cada una de las funcionalidades del sistema, pero para evitar que los diagramas sean demasiado repetitivos debido a la similitud de las funcionalidades de leer, dar de alta, dar de baja o de modificar de cada modelo de datos existente, se realizará uno o dos diagrama de flujo por funcionalidad y modelo de datos, para así, por una parte, evitar sobrecargas, y por otra, mostrar el flujo cada uno de los modelos de datos.

Cabe aclarar, que, para las operaciones con la Base de Datos, gracias a *Spring Boot* se simplifica bastante este trabajo, ya que, esta herramienta "entiende" que las clases creadas en el código, son clases relacionadas con alguna tabla de la Base de Datos. Por ello, gracias a esta comprensión de las clases, vienen algunos métodos predefinidos y que, por otra parte, gracias a

palabras clave, así como: find, exists… se pueden generar nuevos métodos que simplifiquen el acceso a la BD. Métodos como save(), delete()… vienen predefinidos, pero para su correcto uso y que no generen ningún tipo de problema en la BD, aunque necesitan un tratamiento previo y superar unas validaciones que se comprueban en capas superiores sobrescribiendo la lógica predefinida de los métodos.

Debido a la arquitectura de una API, hay ciertos métodos que no necesitan ningún tipo de tratamiento ni lógica especial, estos métodos suelen ser los métodos de *find, exists…* y gracias a lo explicado anteriormente de las facilidades que ofrece *Spring Boot*, al no necesitar ningún tratamiento, se definen en el repositorio y directamente se realiza la llamada sobre dichos *endpoints.*

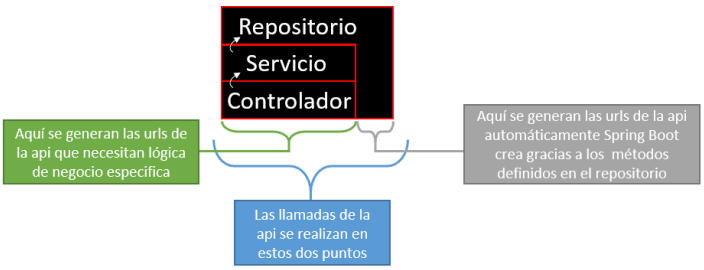


Ilustración 7 – Arquitecura API

Como se puede ver en la imagen, en zona gris es donde Spring Boot publica los endpoints que se han definido y no necesitan lógica. Por otro lado, en la zona verde es donde se han definido o sobreescrito los metodos de los endpoints definidos en el reposiotorio. Son metodos que si necesitan una logica de negocio personalizada o cierta transformacion de los datos. Aunque exista una diferencia entre ciertos metodos, deberian definirse de tal modo que un usuario de la API no sea capaz de identificar que metodos han sido alterados, y de este modo mostrar al usuario de la API una lógica, equilibrada e intuitiva. De este modo, cualquier usuario de la API piensa que realiza las llamadas a la zona azul de la imagen sin saber nada respecto a la lógica que puedan tener estas llamadas.

4.3.1 Gestión de aeropuertos

Leer información de aeropuertos

Para evitar repetitividad con las distintas llamadas que se pueden realizar para buscar la información de uno, varios o todos los aeropuertos, se van a mostrar en un solo diagrama de flujo todas las llamadas posibles, ya que, por otra parte, también son llamadas a *endpoints* que genera *Spring Boot* sin lógica:

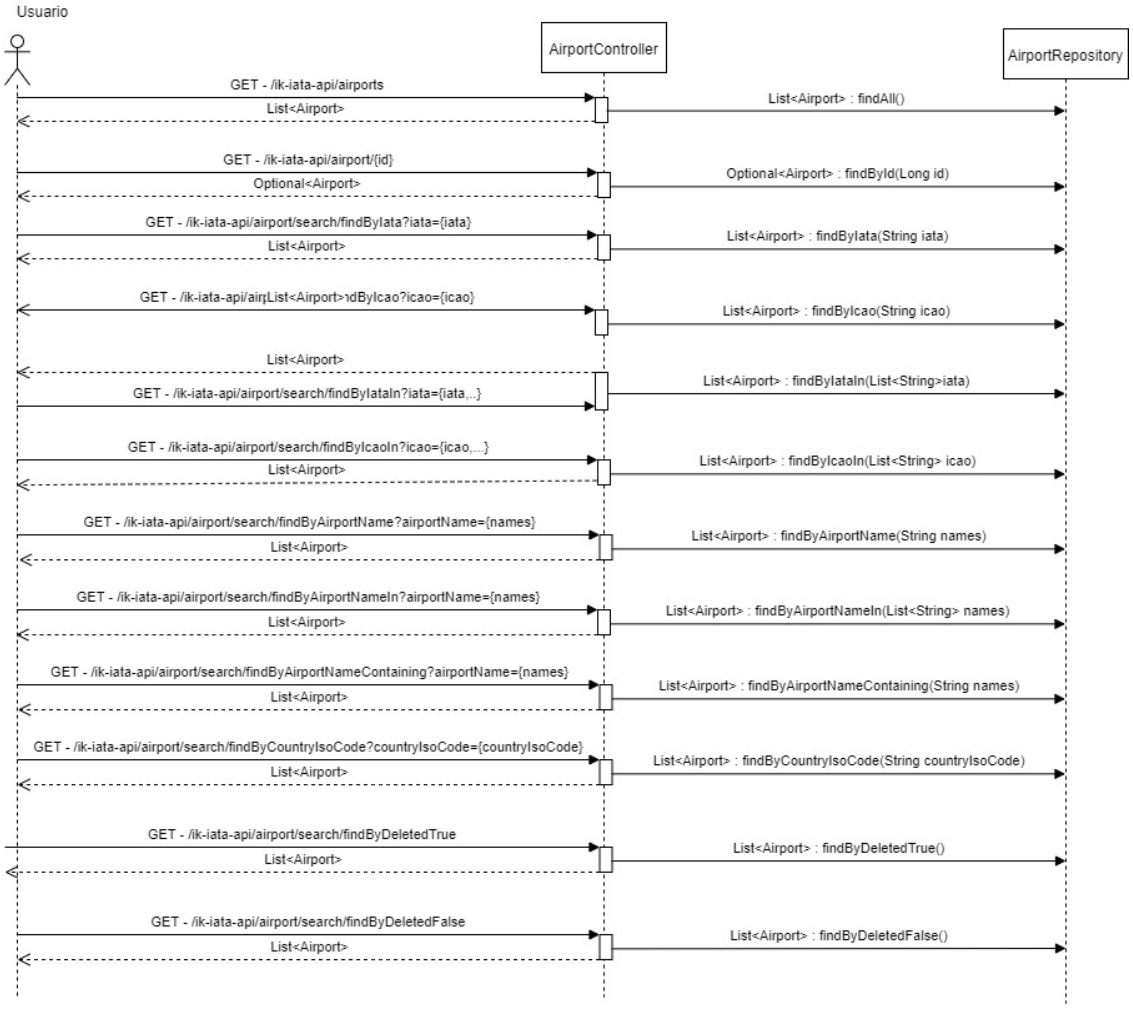


Ilustración 8 – Leer información de aeropuertos

Dar de baja un aeropuerto

Este caso de uso es bastante genérico. Para darle un poco de lógica de negocio distinta, a la hora de realizar el borrado de un elemento, normalmente no interesa realizar un borrado completo de la Base de Datos del elemento, principalmente porque en las aplicaciones de IKUSI, se crean reportes históricos de todos los datos tanto activos como inactivos, por lo tanto, lo común es realizar un borrado lógico; y, por otra parte, el realizar un borrado completo de un elemento puede llegar a tener un fuerte impacto en las aplicaciones que usen la API para la actualización de sus datos de la BD, y esto generaría un problema bastante grave.

Para evitar este tipo de problemas, este método principalmente se hará de modo lógico, salvo para puntos excepcionales, que tras un pequeño estudio con el *Product Owner*, se llegó a la conclusión que sí tendría sentido también poder realizar un borrado completo de un elemento dándose el caso que, a la hora de introducir un dato nuevo, se crea con varios datos erróneos y en vez de cambiar uno por uno, se realiza un borrado completo y se empieza de nuevo.

Flujo normal:

Como podemos observar en el flujo de abajo, se han englobado en un mismo *endpoint* el borrado lógico y completo. El método *DELETE* no devuelve ningún tipo de código HTTP.

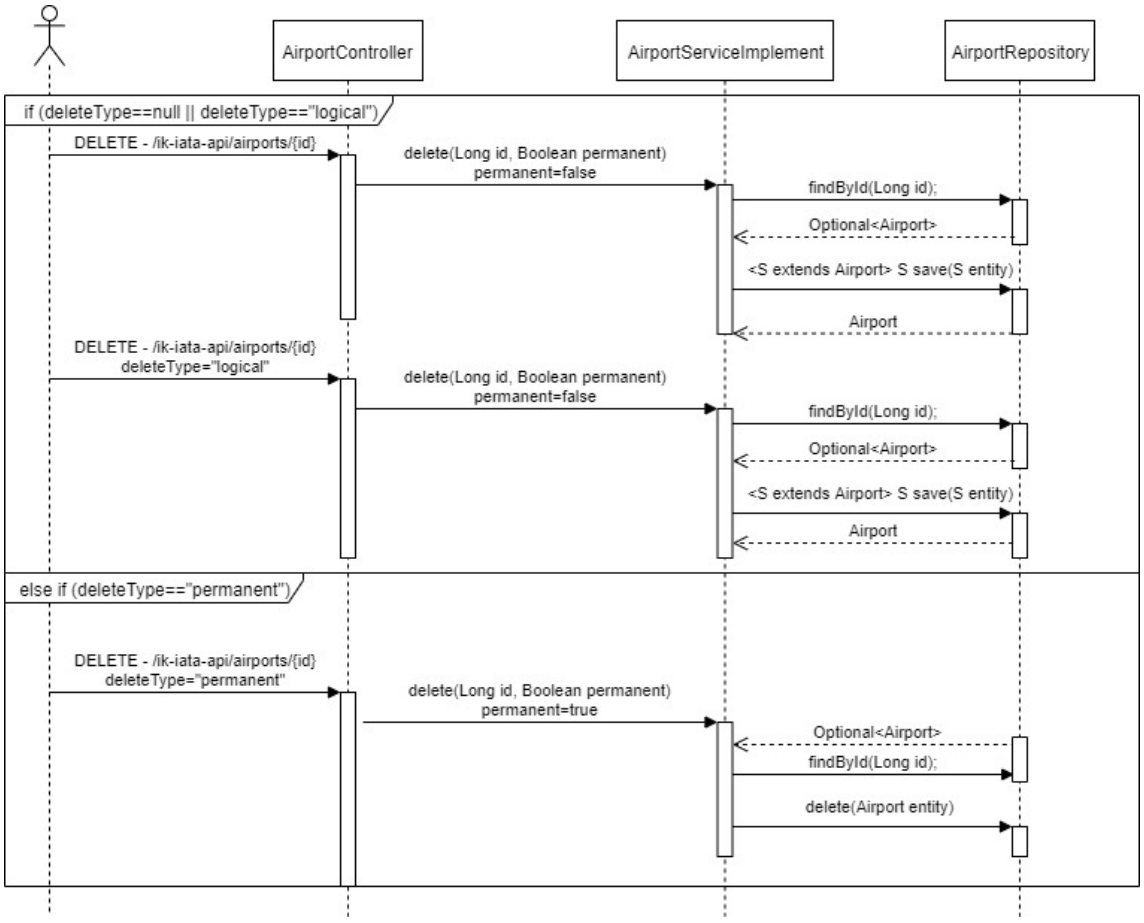


Ilustración 9 – Dar de baja un aeropuerto – Flujo normal

Flujo alternativo:

En el primer flujo de abajo, está el caso en el que se realiza la llamada de manera incorrecta intentado realizar un tipo de borrado no existente, por lo que la API devuelve un 400. En el segundo flujo, nos encontramos con el caso de que se intenta realizar algún tipo de borrado sobre un *id* que no existe, por lo tanto, devuelve un *400 Not Found.* En el tercer flujo y para

terminar, se da el caso en el que se introduce un *id* nulo o que no equivale a un número, por lo cual devuelve un 400 también.

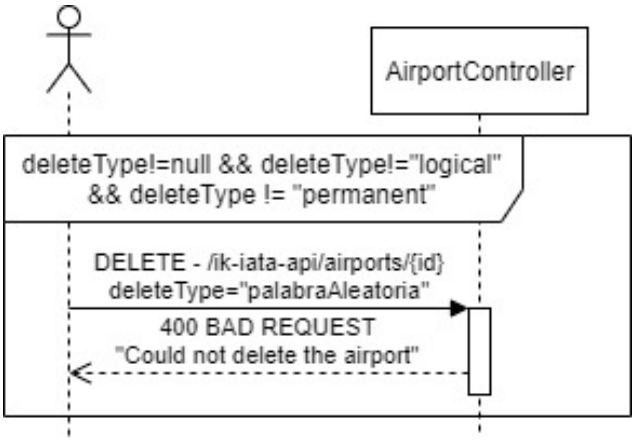


Ilustración 10 – Dar de baja un aeropuerto – Flujo alternativo 1

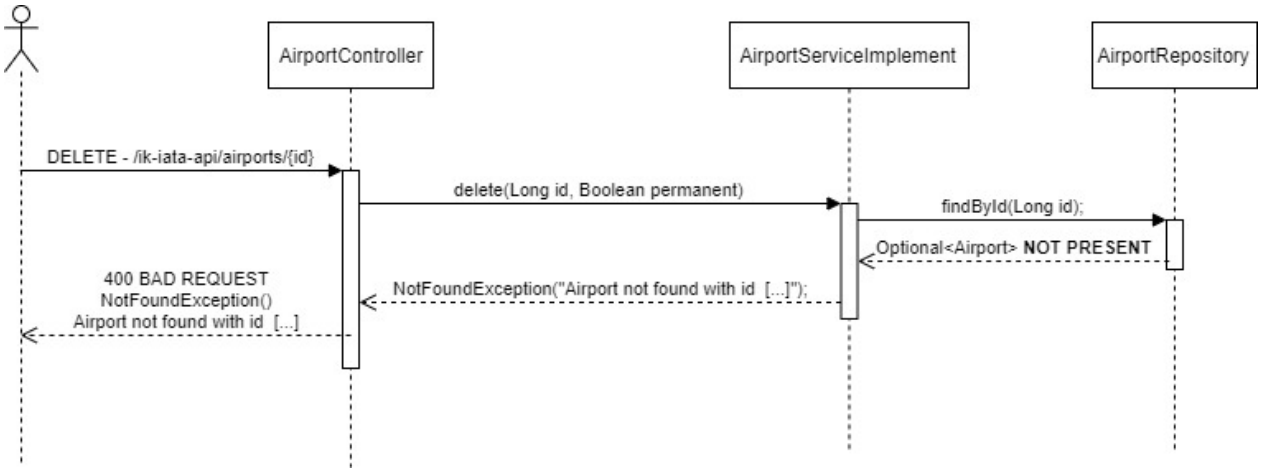


Ilustración 11 – Dar de baja un aeropuerto – Flujo alternativo 2

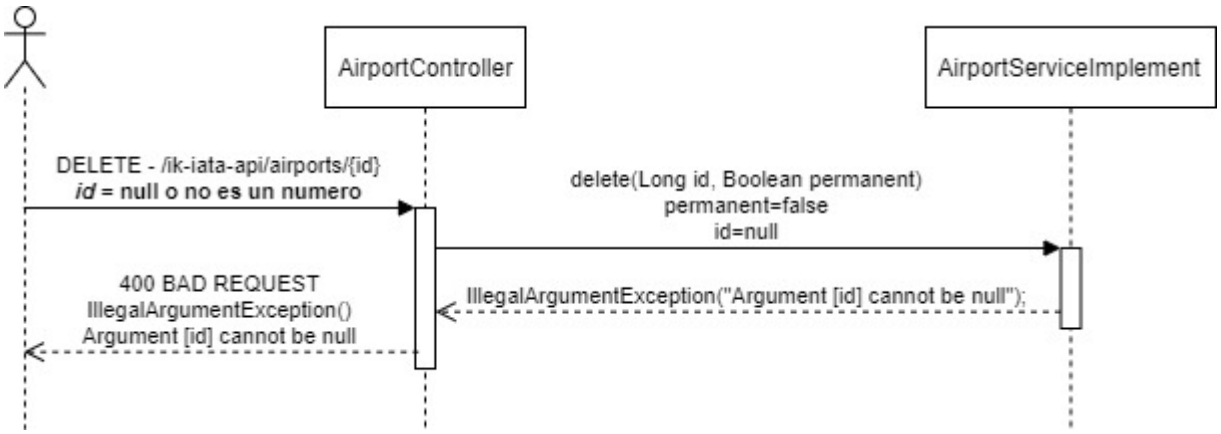


Ilustración 12 – Dar de baja un aeropuerto – Flujo alternativo 3

Dar de baja varios aeropuertos:

Este caso de uso es bastante similar al de arriba, pero en este caso, no se incluye ningún *id* como parte de la url, en este caso, se envía mediante el *body* una lista de *id* para la realización del borrado. Al ser tan similares, solo se incluye el diagrama de flujo normal:

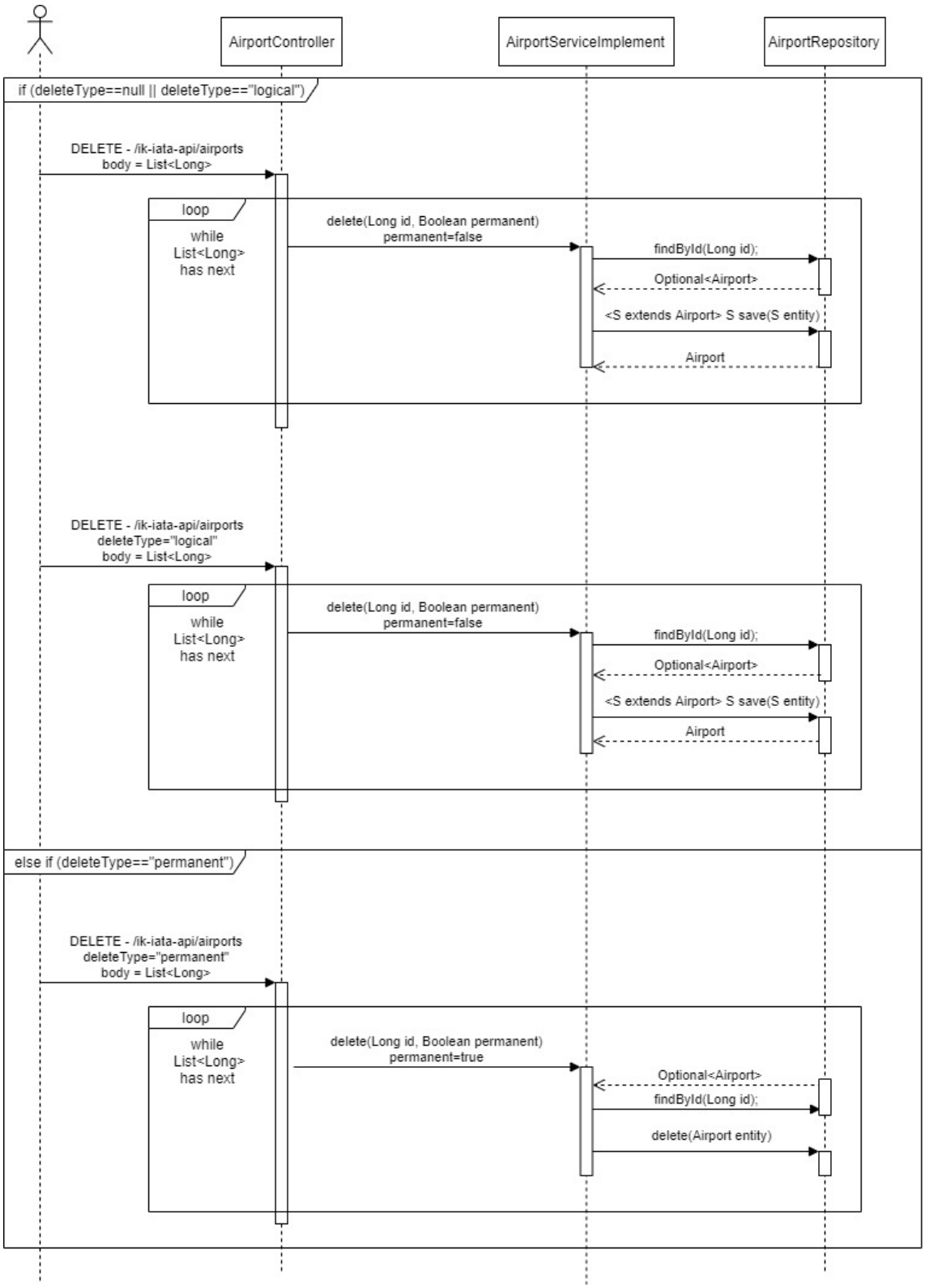


Ilustración 13 – Dar de baja varios aeropuertos

4.3.2 Gestión de aerolíneas

Dar de alta aerolíneas

En el caso de dar de alta una aerolínea, tiene alguna validación algo distinta, ya que, el código Iata o el código Icao puede ser reutilizado siempre y cuando la aerolínea haya dejado de existir, y esta aerolínea haya sido borrada anteriormente a la fecha que se quiera dar de alta esta nueva aerolínea. Por un lado, se deben comprobar los campos Iata e Icao que cumpla con el formato establecido para cada uno, que no exista duplicidad con alguna aerolínea activa y que puede haber más de una aerolínea con estos datos nulos. Por otro lado, en caso de sobrescribir códigos, la fecha de activación de Iata y fecha de activación de Icao de la aerolínea nueva, que no solapen la fecha de borrado de la aerolínea vieja.

Flujo normal:

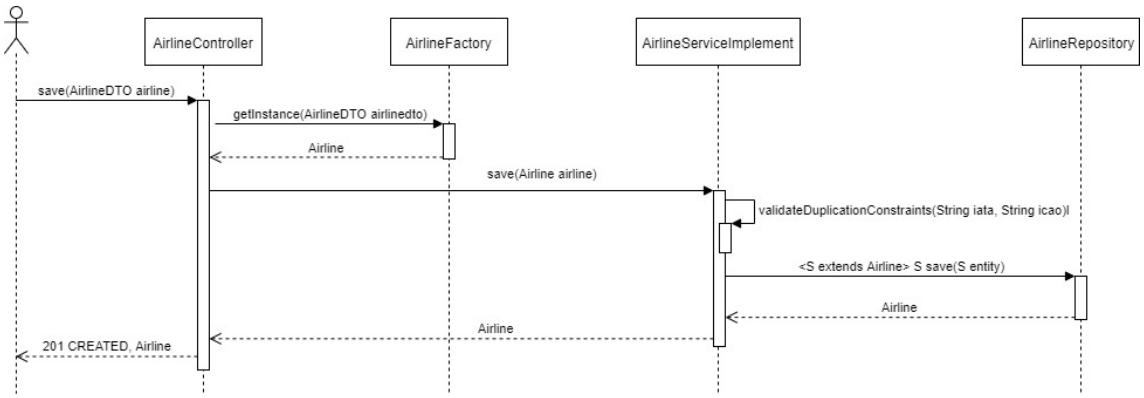


Ilustración 14 – Dar de alta aerolíneas – Flujo normal

Flujo alternativo:

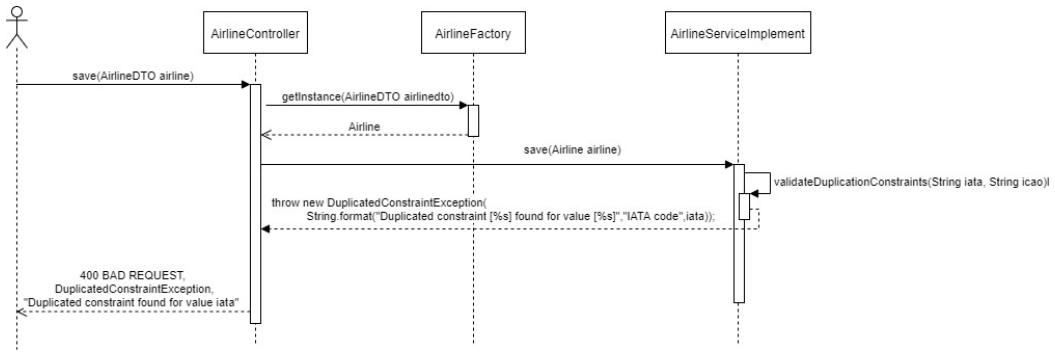


Ilustración 15 – Dar de alta aerolíneas – Flujo alternativo 1

En el flujo de la parte de arriba, indica el caso de intentar dar de alta una aerolínea con código Iata o código Icao repetido, y en el siguiente caso, se intenta guardar algún dato que no cumpla con los requisitos previamente definidos.

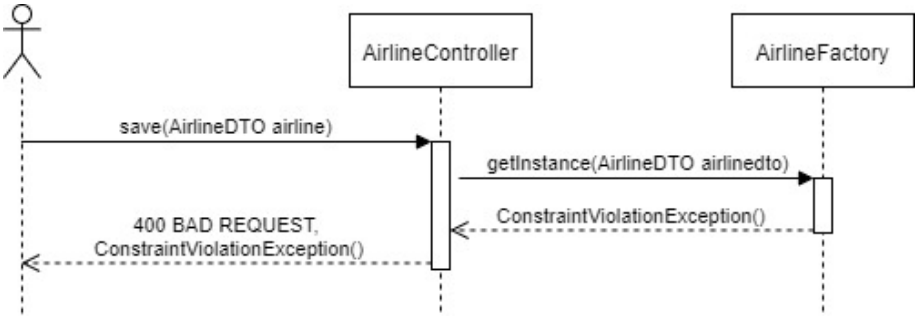


Ilustración 16 – Dar de alta aerolíneas – Flujo alternativo 2

4.3.3 Gestión de aeronaves

En este caso de uso, para actualizar las aeronaves, la diferencia principal respecto a los aeropuertos y aerolíneas, es que una aeronave puede tener el mismo código Icao que otras aeronaves, pero nunca el mismo código Iata, es decir, al existir tanto tipos de modelos y variantes muy similares, rehúsan el código Icao como grupo de aeronaves, pero siempre teniendo códigos Iata distintos y nunca se pueden duplicar. En este caso se puede realizar la modificación de dos tipos: modificaciones completas o modificaciones de ciertos atributos de uno o varios objetos.

Modificación completa de una aeronave:

Flujo normal:

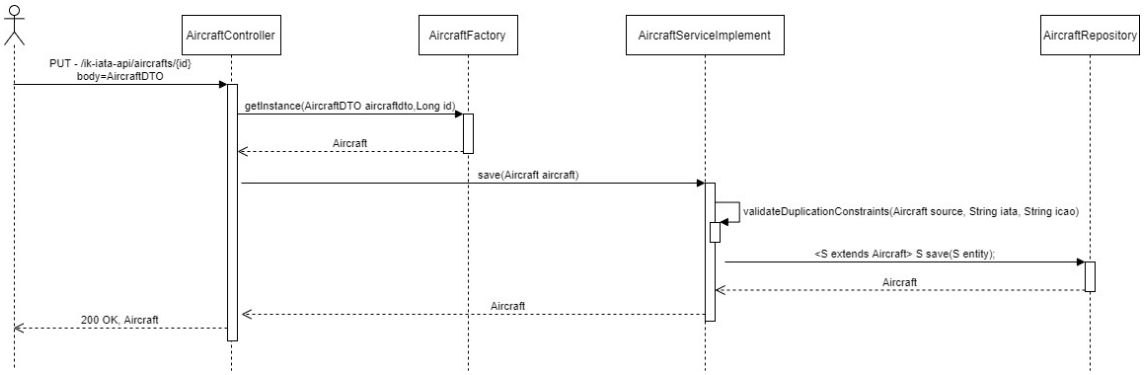


Ilustración 17 – Modificación completa de una aeronave – Flujo normal

Flujo alternativo:

En los próximos flujos de diagrama alternativos, nos encontramos con 3 casos distintos:

1) La aeronave que estamos intentando actualizar, el código Iata y/o el código Icao son distintos al de la aeronave seleccionada, pero uno o los dos códigos ya existen.

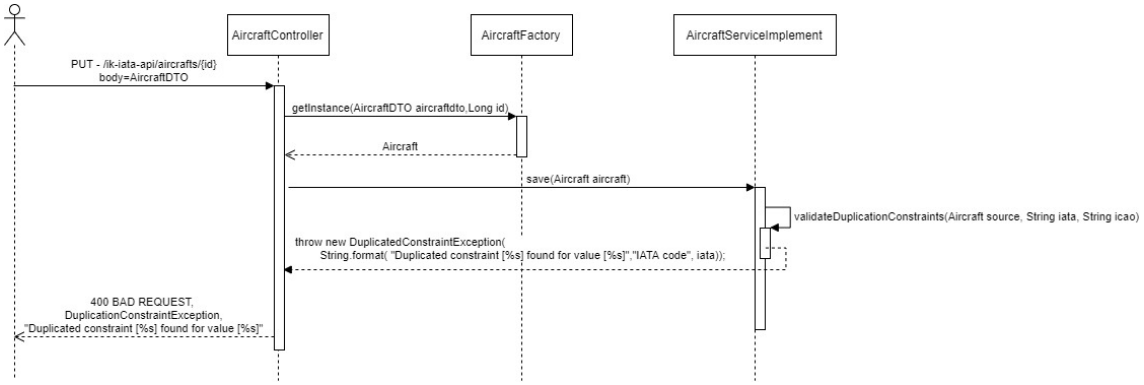


Ilustración 18 – Modificación completa de una aeronave – Flujo alternativo 1

2) Que el *id* con el que se hace la llamada al *endpoint* no exista en la BD.

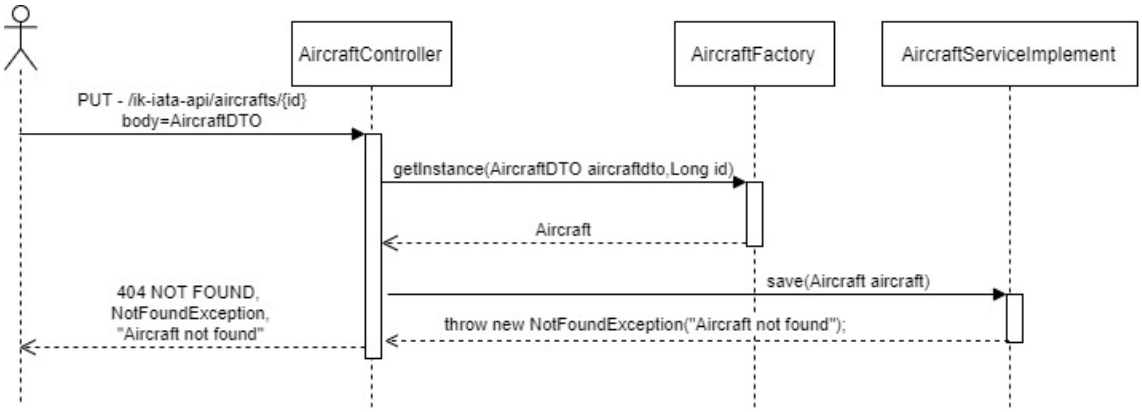


Ilustración 19 – Modificación completa de una aeronave – Flujo alternativo 2

3) Que se envié algún parámetro en el *body* que no cumpla con los requisitos establecidos.

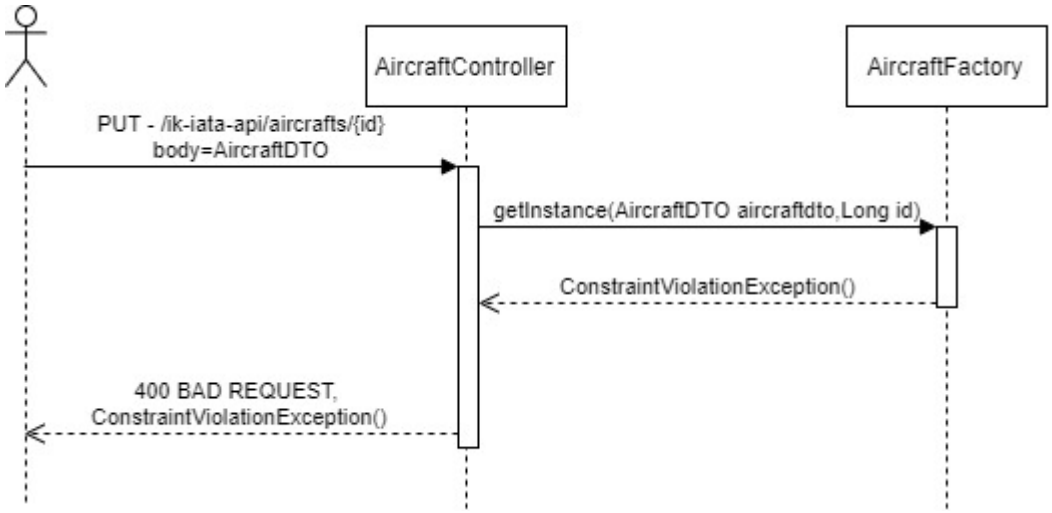


Ilustración 20 – Modificación completa de una aeronave – Flujo alternativo 3

Modificación parcial de una aeronave:

Flujo normal:

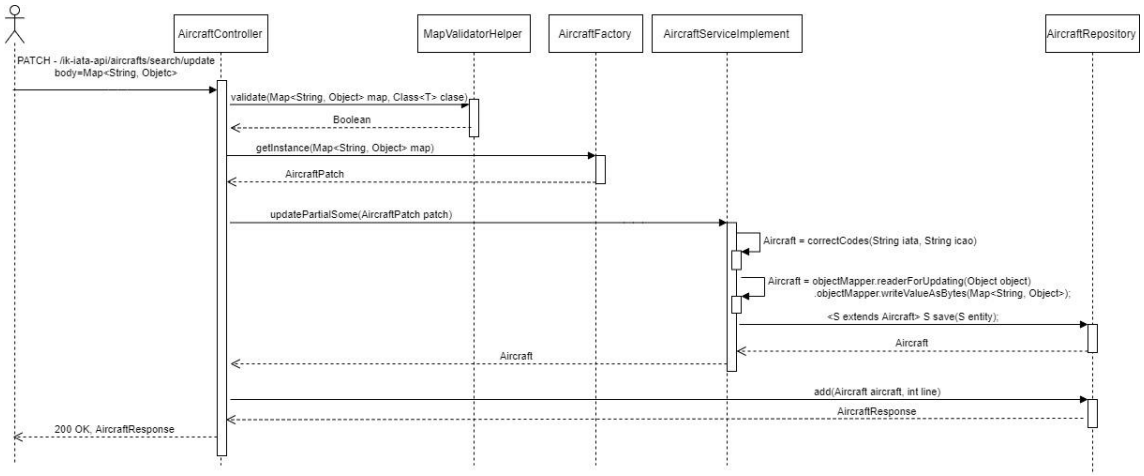


Ilustración 21 – Modificación parcial de una aeronave – Flujo normal

Flujo alternativo:

Aunque en los flujos de abajo existen 2 que devuelve un OK como llamada al *endpoint*¸ y a primera vista un OK parece que sería un flujo normal, realmente son errores que el sistema controla y maneja, es decir, no existe una excepción como para devolver un error, simplemente existe algún dato mal y se maneja como una petición correcta (200 OK) aunque en el *body* aparezca el resultado obtenido.

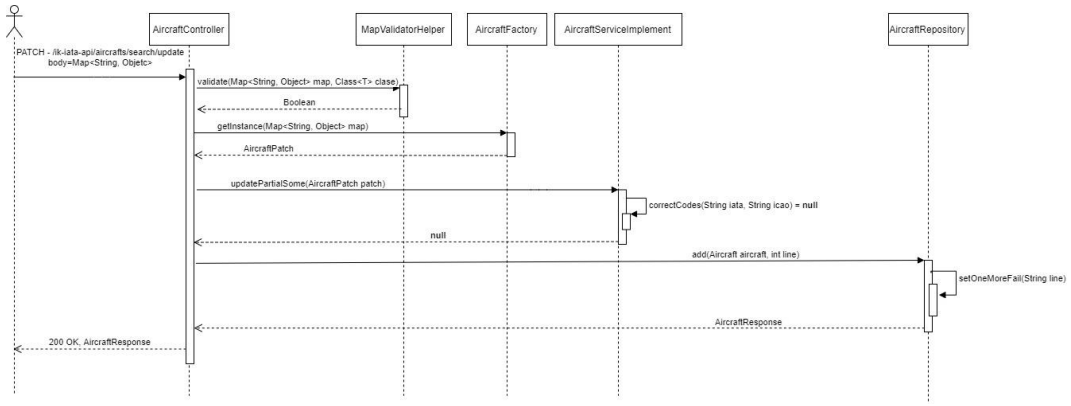


Ilustración 22 – Modificación parcial de una aeronave – Flujo alternativo 1

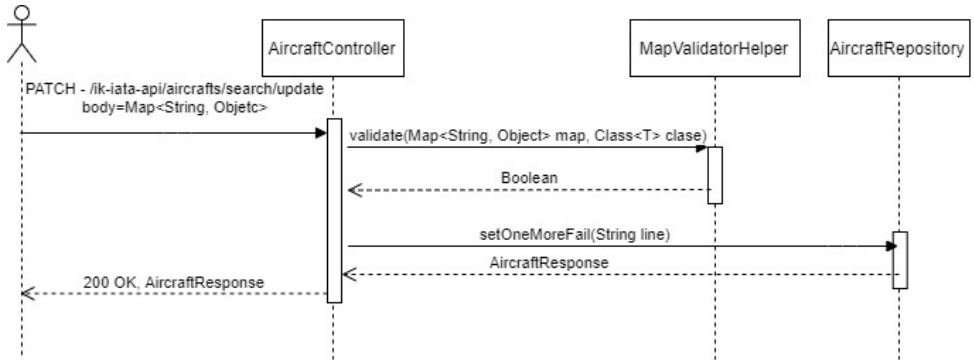


Ilustración 23 – Modificación parcial de una aeronave – Flujo alternativo 2

Como se puede observar en los casos de arriba, devuelve un 200 por lo explicado anteriormente, en el primero falla ya que existe algún tipo de incongruencia entre el código Iata y/o el código Icao, pero están correctamente escritos, y en segundo, el mapa (*Map<String, Object>*) que se envía en el *body* tiene alguna clave mal escrita o inexistente para este objeto.

En el caso de los de abajo, si existe un error ya que la petición está mal realizada, en el primero, algún dato enviado en el mapa no cumple con los requisitos prestablecidos para el sistema, y el segundo caso, se envía un *body* nulo lo cual es un error obvio.

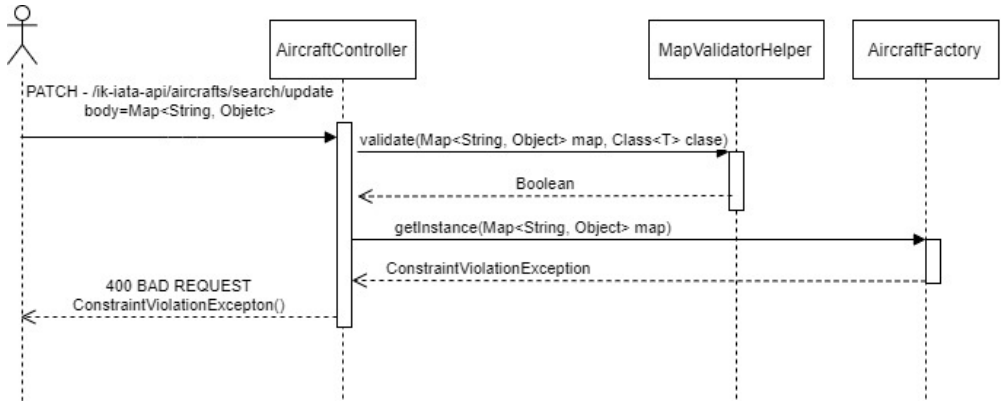


Ilustración 24 – Modificación parcial de una aeronave – Flujo alternativo 3

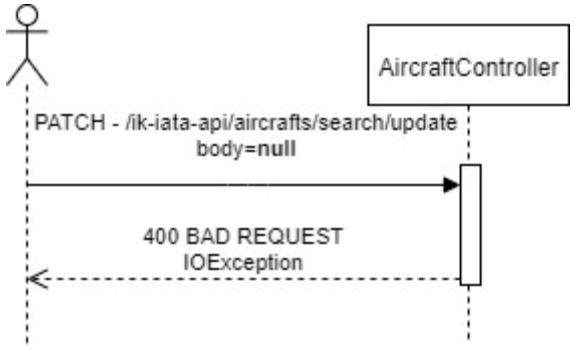


Ilustración 25 – Modificación parcial de una aeronave – Flujo alternativo 4

Modificación parcial de varias aeronaves:

Este caso, es como el de arriba, lo único que le diferencia es que en este caso se envía una lista de mapas, por ello, se va a mostrar solo el diagrama de flujo normal para ver cuál sería la pequeña diferencia. En los flujos de error que devuelven 200, sería el mismo flujo, pero con una iteración en función de tamaño de la lista, y en los casos que devuelven 400, sería el mismo flujo.

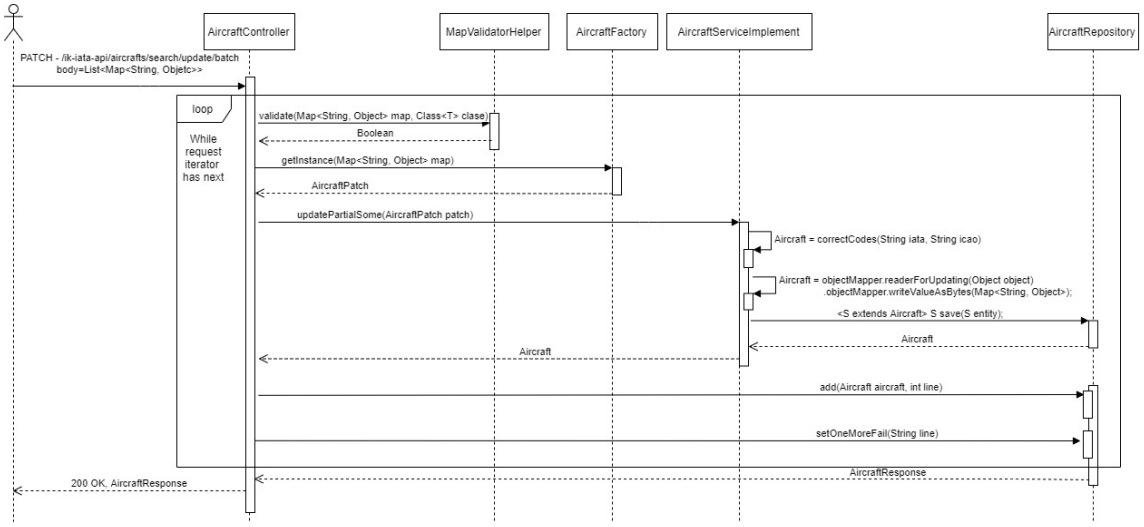


Ilustración 26 – Modificación parcial de varias aeronaves

4.3.4 Gestión de clientes

IK-SS

Esta parte de la gestión de clientes, es un pequeño paréntesis dentro de la API, ya que para esta gestión, se ha reutilizado un componente de la empresa IKUSI que está diseñada para esta gestión de los proyectos creados en la empresa. En el anexo B se puede encontrar más información.

4.3.5 Gestión de ficheros

Para el procesado de ficheros, las operaciones *CRUD* son muy similares entre las clases existentes salvo por el tema de validaciones y como esta parte de validaciones se ve algo más clara en los diagramas de flujo anteriores, ahora se van a mostrar los métodos de gestión de ficheros de una sola clase. También ha de constar que solo se va a mostrar el diagrama de flujos alternativos del primero ya que los siguientes métodos se tratan de la misma manera y seria sobrecargar con diagramas idénticos.

Carga inicial desde ficheros (Aeropuertos):

Flujo normal:

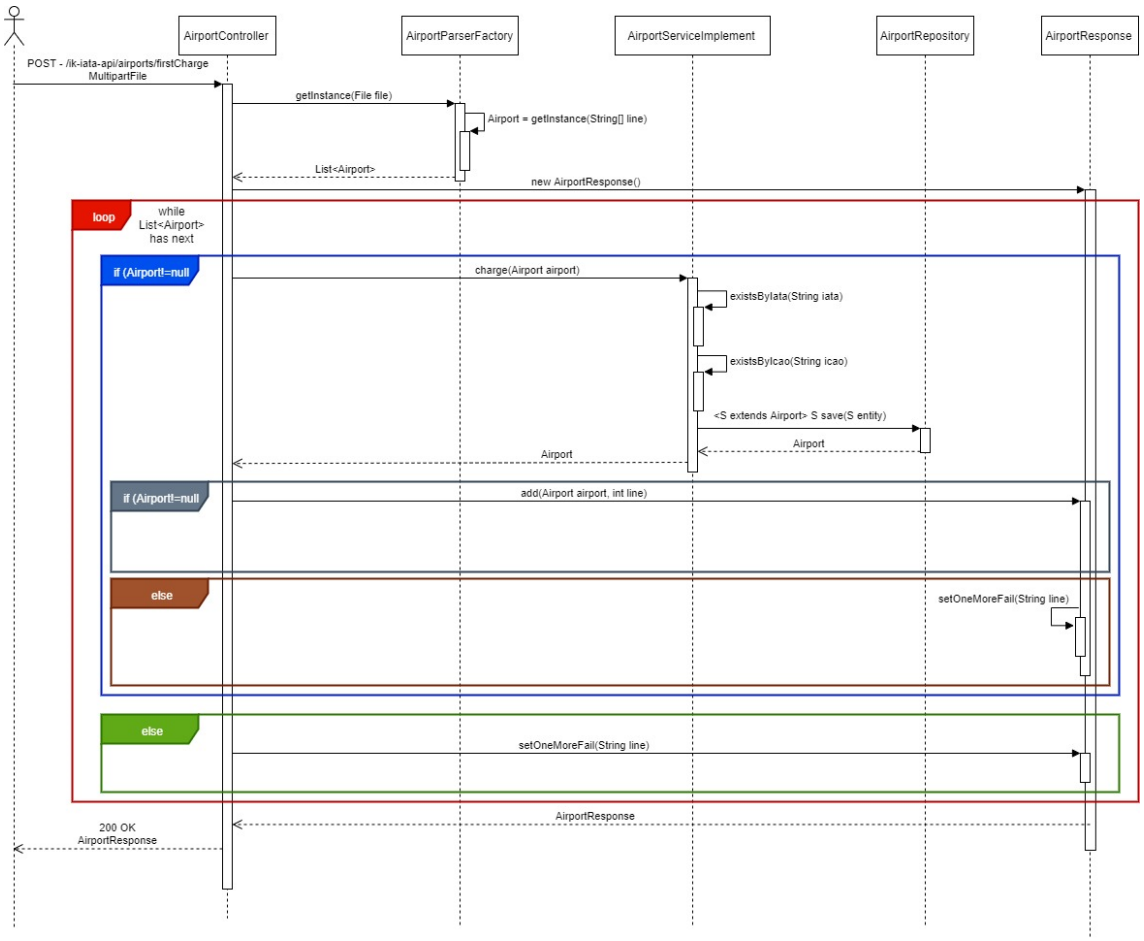


Ilustración 27 – Carga inicial desde ficheros – Flujo normal

En este caso de arriba, un usuario sube un archivo tipo *csv* a la API, la cual transforma el archivo recibido en una lista de objetos (Aeropuertos en este caso) y luego manda a las capas inferiores para gestionar cada objeto. Como se puede ver, ese método admite errores gestionándolos, y más tarde, tras procesar el archivo completo, enviar una respuesta con los aeropuertos correctamente creados y cuales fallaron. De este modo evitamos que un usuario repita la subida del archivo por cada objeto fallido.

Flujo alternativo:

En los siguientes flujos, vemos como si el archivo subido es corrupto, no es del tipo permitido o directamente, si no se sube un archivo, como se gestionan los fallos. (Al igual que estos diagramas, en el resto de métodos de gestión de ficheros, se manejan los fallos de archivo del mismo modo entre la primera y segunda capa).

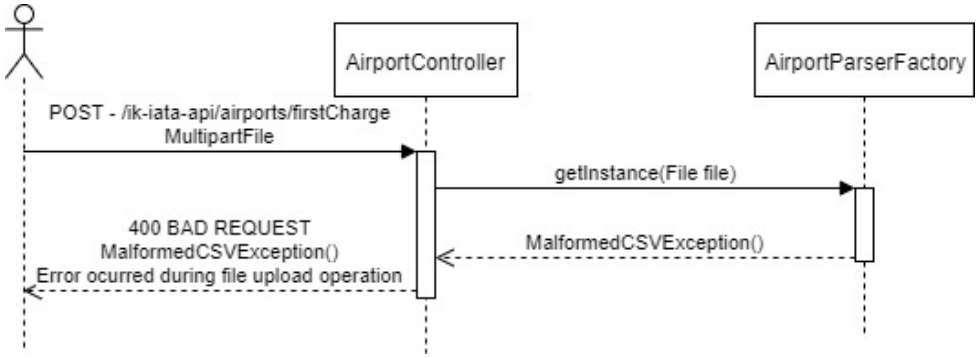


Ilustración 28 – Carga inicial desde ficheros – Flujo alternativo 1

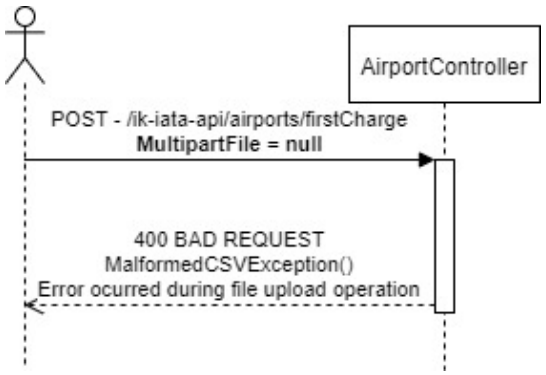


Ilustración 29 – Carga inicial desde ficheros – Flujo alternativo 2

Modificar desde fichero (Aeropuertos):

En este caso de modificar desde ficheros, se ha establecido una serie de requisitos previos para poder crear estos archivos de manera correcta, es decir, esta acción es una modificación parcial de objetos, por ello, hay que indicar en el fichero que objeto se quiere modificar y que parámetros se van a modificar. Tras una validación y transformación, se intenta realizar la modificación, y como en el primer caso de *Carga inicial desde fichero*, se devuelve que objetos han sido modificados correctamente y cuales han tenido algún tipo de fallo.

Flujo normal:

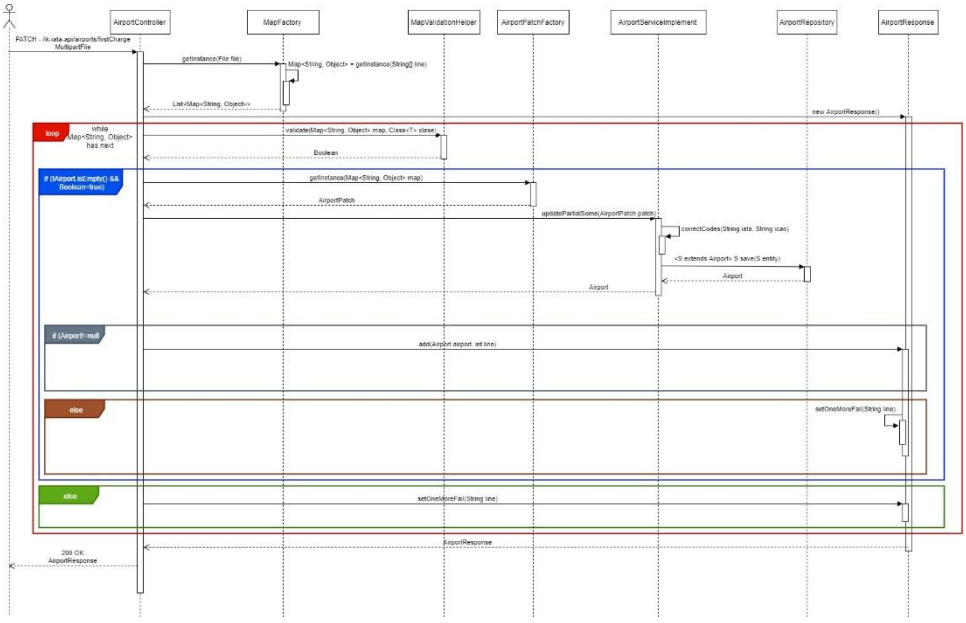


Ilustración 30 – Modificar desde fichero – Flujo normal

Dar de alta desde fichero (Aeropuertos):

Para dar de alta objetos desde fichero, se ha establecido un formato concreto. En este caso, se gestiona de manera muy similar a la *Carga inicial desde fichero* en una primera versión de la API, aunque a futuro las lógicas cambian y se verá como fue necesario esta separación entra la primera carga y cargas posteriores, aunque a día de hoy parezca que no tiene mucho sentido mantener 2 *endpoints* idénticos.

Flujo normal:

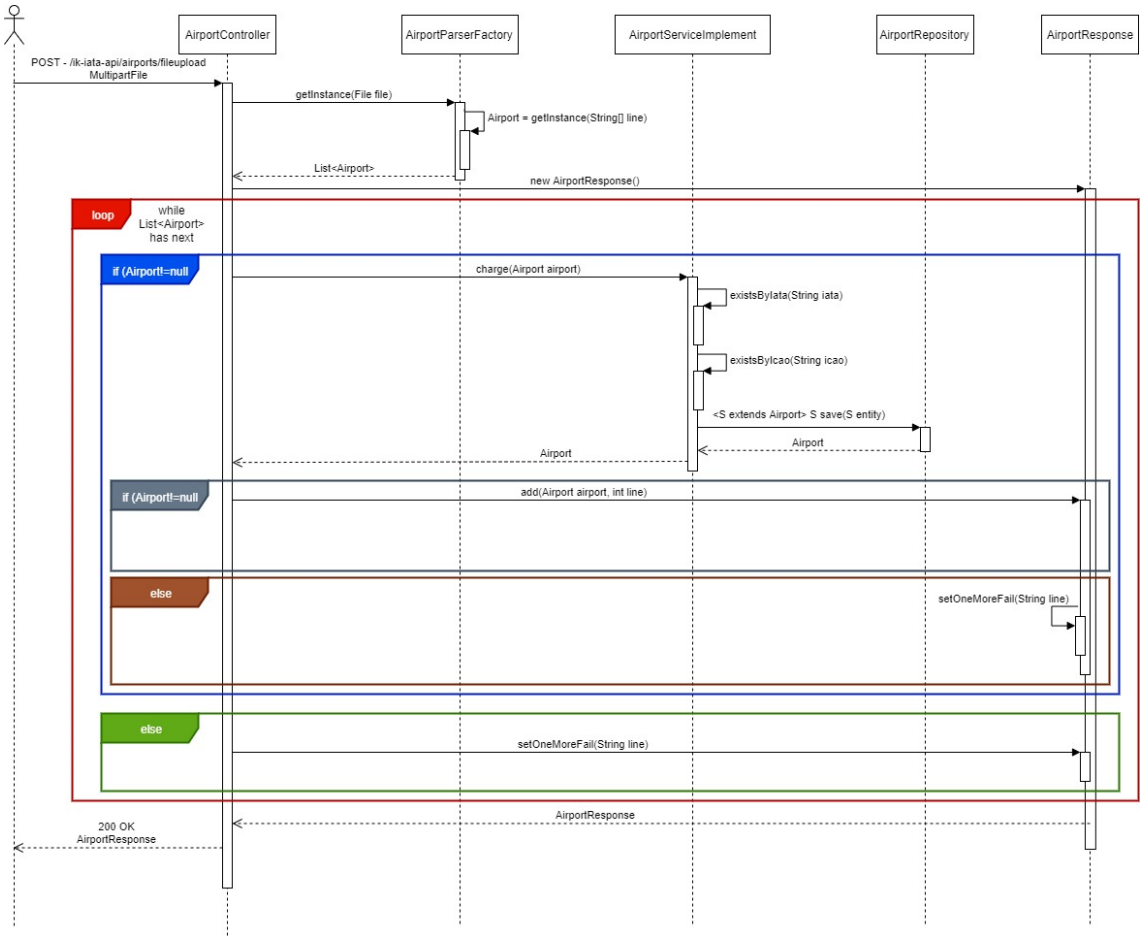


Ilustración 31 – Dar de alta desde fichero – Flujo normal

Dar de baja desde fichero (Aeropuertos):

Para dar de baja desde fichero, es necesario de nuevo cumplir con los requisitos previos establecidos para la correcta creación del archivo. En este caso, se solicitan los códigos Iata e Icao para el borrado del elemento. En ocasiones que solo exista un objeto con el código enviado, bastará con enviar cualquiera de los códigos, pero cuando puedan existir más de uno (grupos de aeronaves, aeropuertos con códigos con valores nulos…), será necesario enviar los 2 códigos y en este caso también se devuelve que peticiones se han desarrollado correctamente y cuáles no.

Flujo normal:

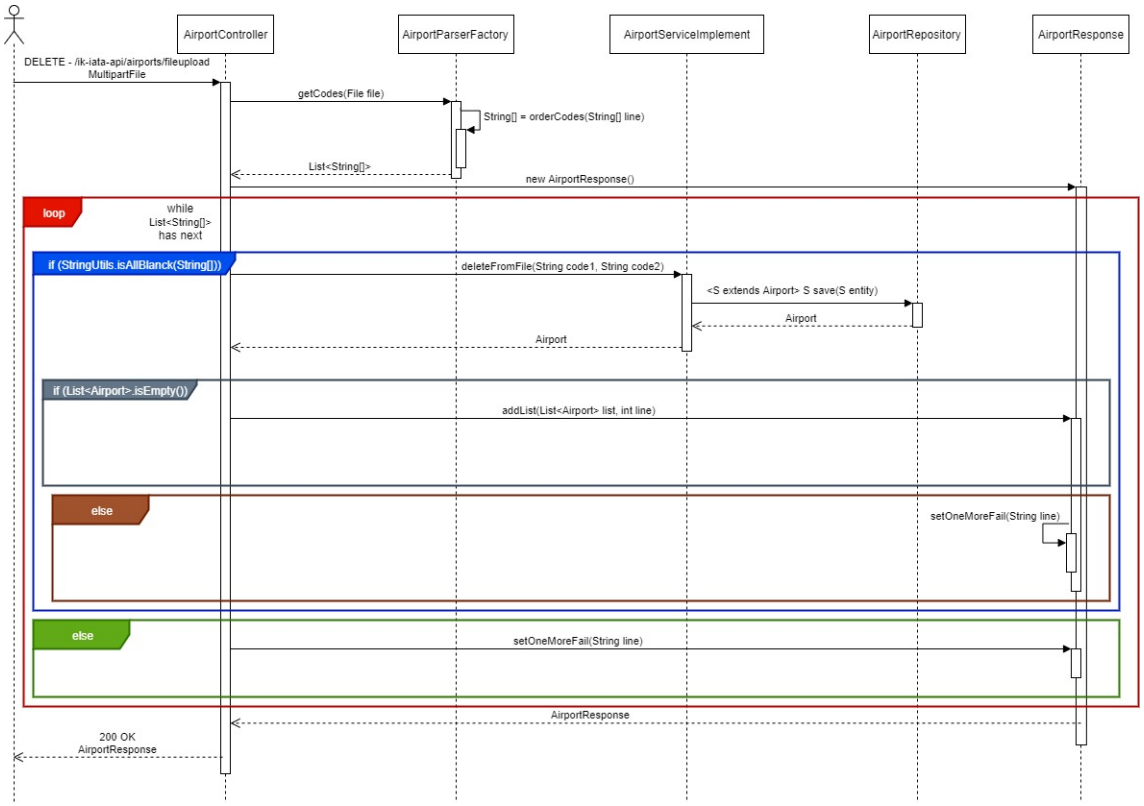


Ilustración 32 – Dar de baja desde fichero – Flujo normal

4.4 Diseño del Front end

Como antes en esta memoria se ha comentado, el diseño de la parte del Front end debe seguir el estilo propio de Ikusi, *IK-WI-4*, esta obligación por una parte nos limita a la hora de poder desarrollar las interfaces, pero también, nos ayuda ya que no hace falta que pensemos en estilos y formas para cada pantalla ya que esta predefinido el modo de cómo hacerlo.

Para empezar con el diseño, tenemos que tener en cuenta, que un usuario que pueda entrar a la aplicación tiene que ser un administrador que vaya a realizar un mantenimiento o configuración, ya que un usuario normal va ser cualquier cliente que se suscriba a la aplicación mediante *Kafka*. Por ello, para empezar, el administrador debe loguearse en la página de login.

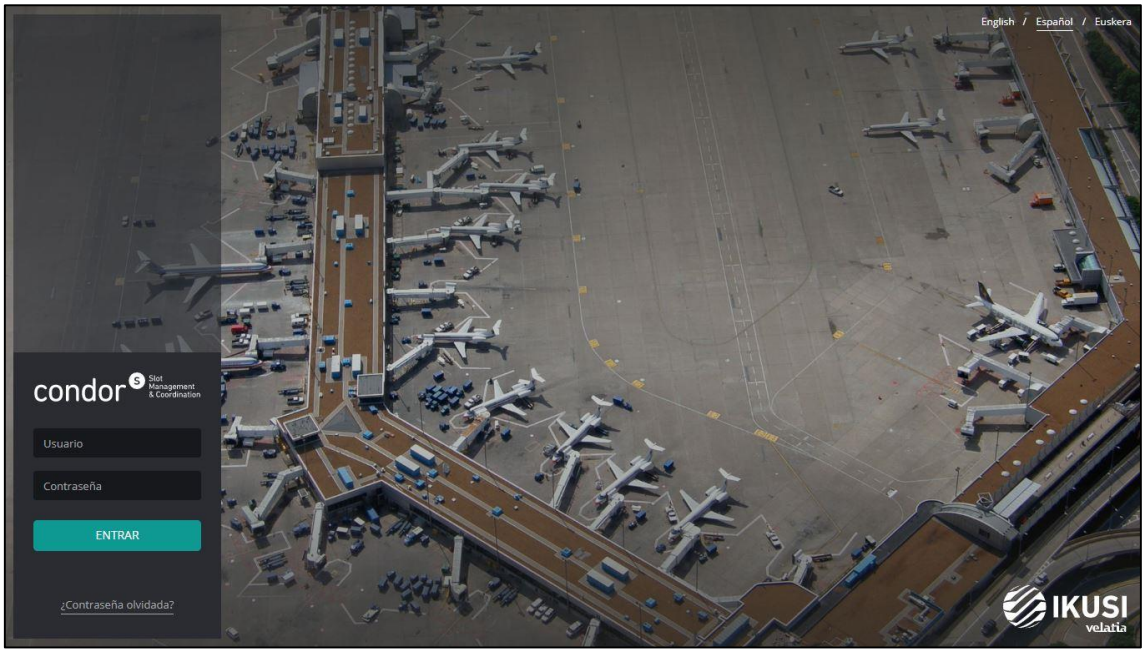


Ilustración 33 – Diseño de interfaces – Pantalla iniciar sesión

En este punto, se puede ver en la imagen de arriba que se puede elegir el idioma en el que queremos usar la aplicación. Si el administrador introduce mal el usuario o contraseña la aplicación le a avisar que ha introducido mal los datos y cómo podemos ver a continuación. También en la página de inicio vemos la opción de recuperar la contraseña (*¿Contraseña olvidada?*), la cual nos abrirá una ventana en la cual deberemos de introducir el email con el que creamos el perfil de administrador:

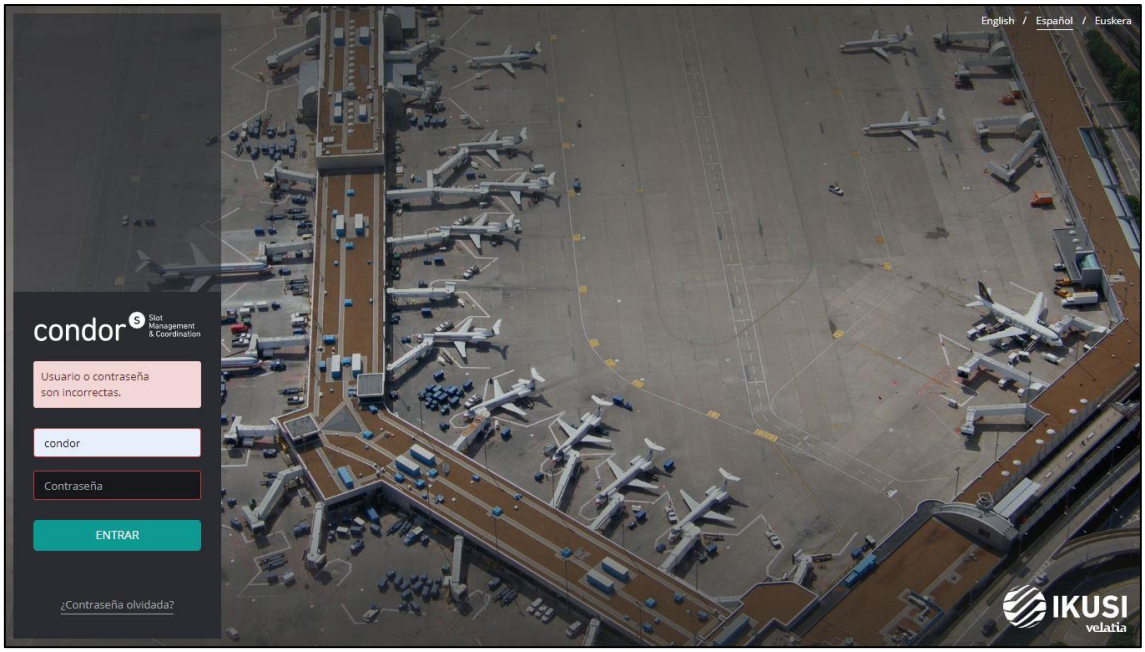


Ilustración 34 – Diseño de interfaces – Usuario incorrecto

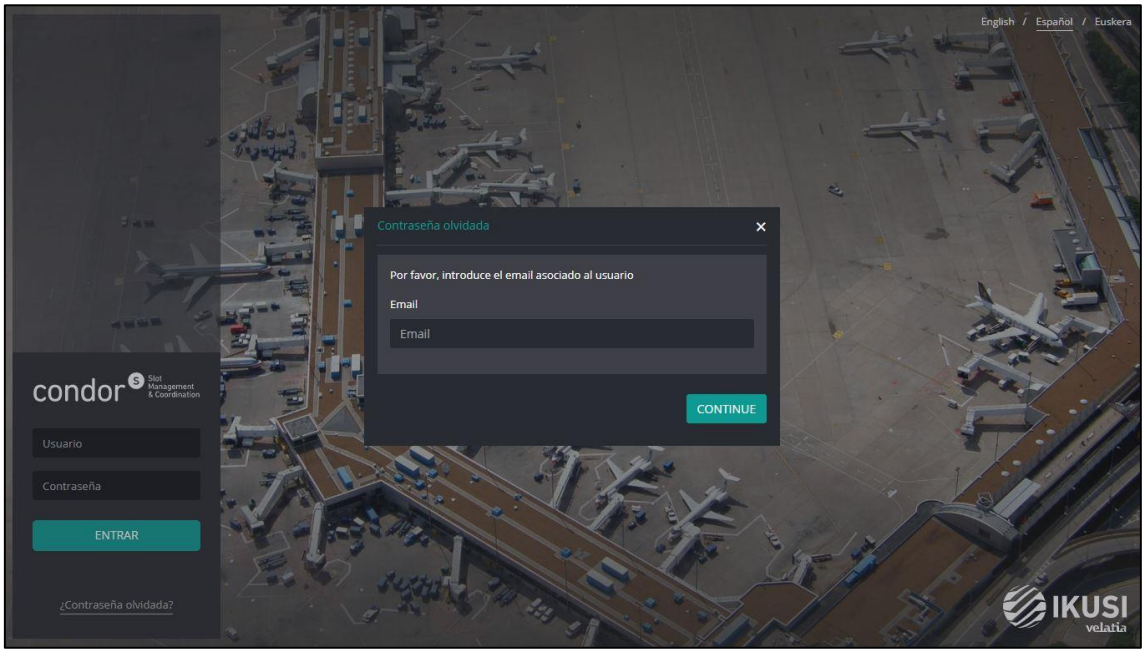


Ilustración 35 – Diseño de interfaces – Contraseña olvidada

Tras introducir los datos, entraremos en la página de bienvenida la cual nos mostrara unas instrucciones para el correcto funcionamiento de la aplicación y de este modo evitar dudas de uso:

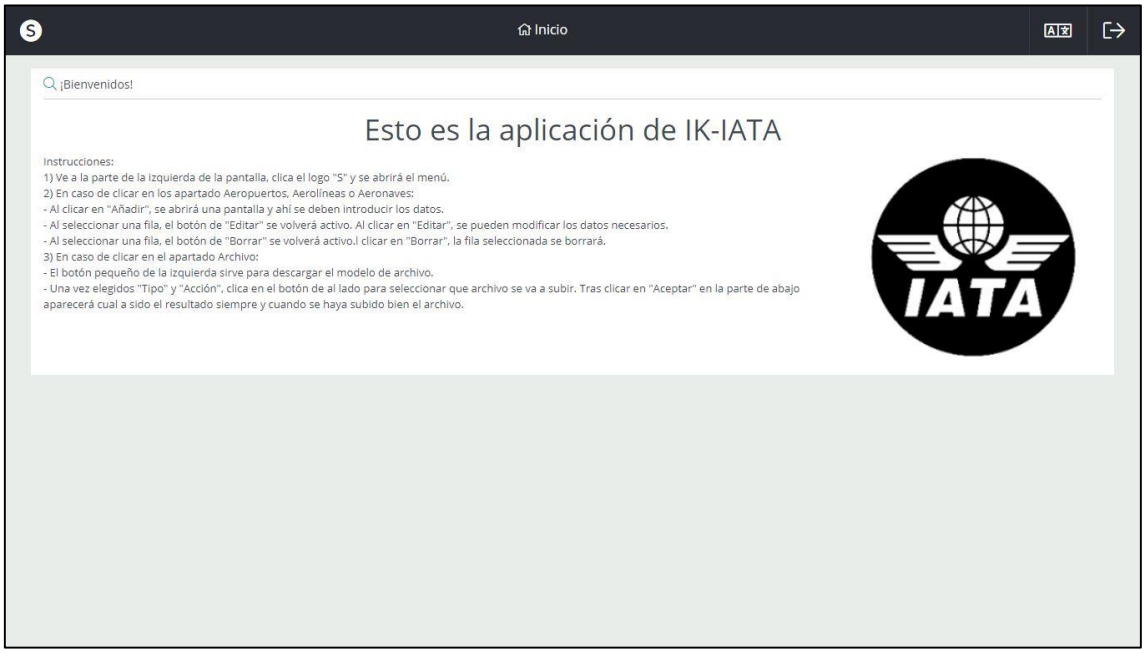


Ilustración 36 – Diseño de interfaces – Pantalla inicio

En la página de bienvenida nos encontramos con ciertas opciones en la parte de arriba: en la esquina izquierda se ve un logo de una “S” el cual nos abrirá el menú para poder navegar por el resto de funcionalidades que ofrece. En la parte derecha nos encontramos con 2 botones, el de la derecha nos da la opción de salir de la sesión, y el de la izquierda nos da la opción de cambiar el idioma de la aplicación:

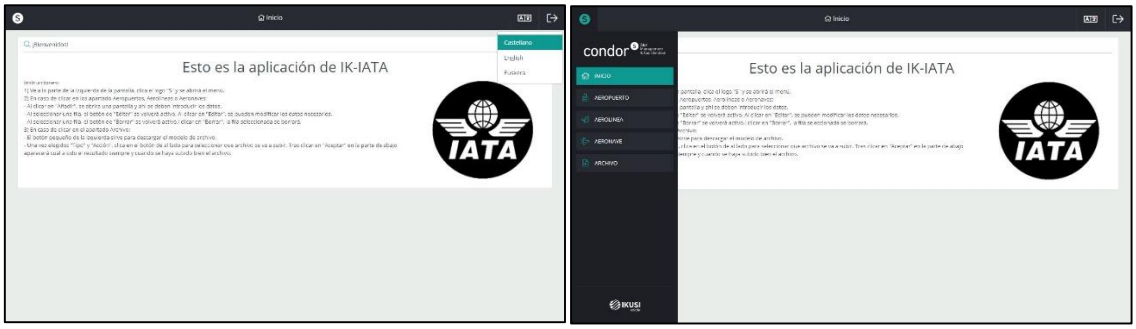


Ilustración 37 – Diseño de interfaces – Pantalla inicio y menú (Español)

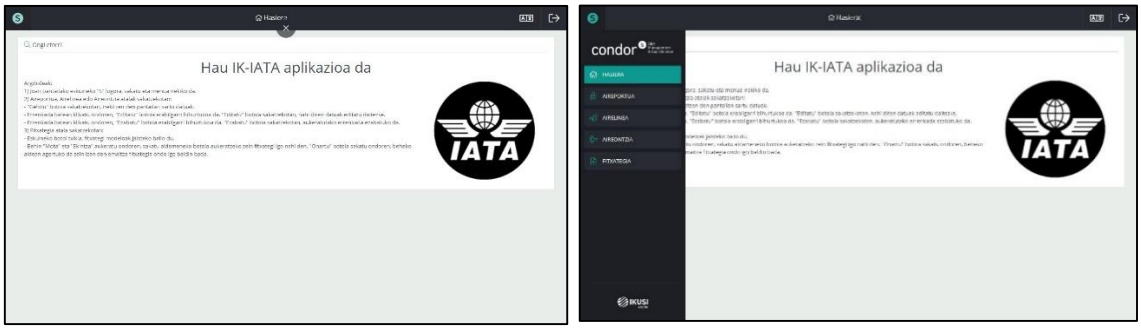


Ilustración 38 – Diseño de interfaces – Pantalla inicio y menú (Euskera)

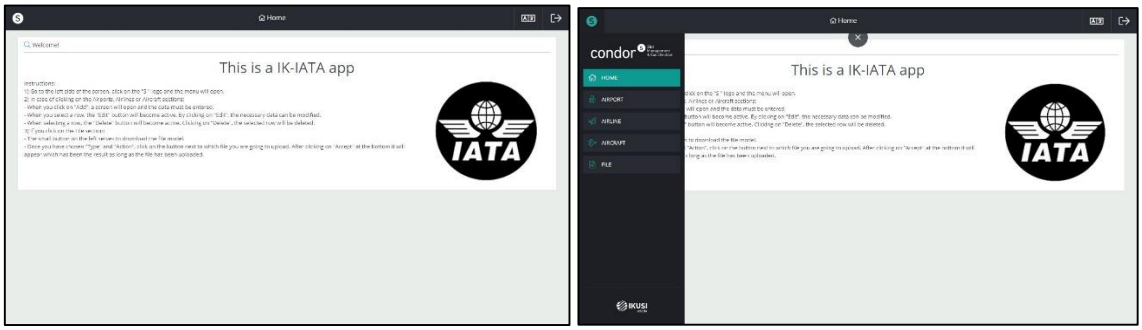


Ilustración 39 – Diseño de interfaces – Pantalla inicio y menú (English)

Como el proyecto está desarrollado en castellano, se continuará con imágenes en castellano.

Para continuar, seleccionamos en el menú la primera opción que aparece, Aeropuertos. En la sección de aeropuertos vemos una tabla con los aeropuertos existentes que están activos, cabe destacar que son datos de prueba, unas 10-20 líneas, pero realmente la aplicación va a manejar miles y miles de datos ya que va a operar con los aeropuertos, aerolíneas y aeronaves que existen en todo el mundo, y también va a recoger los históricos que existen en la Base de Datos de las aplicaciones aeroportuarias de Ikusi, es decir millones de datos:

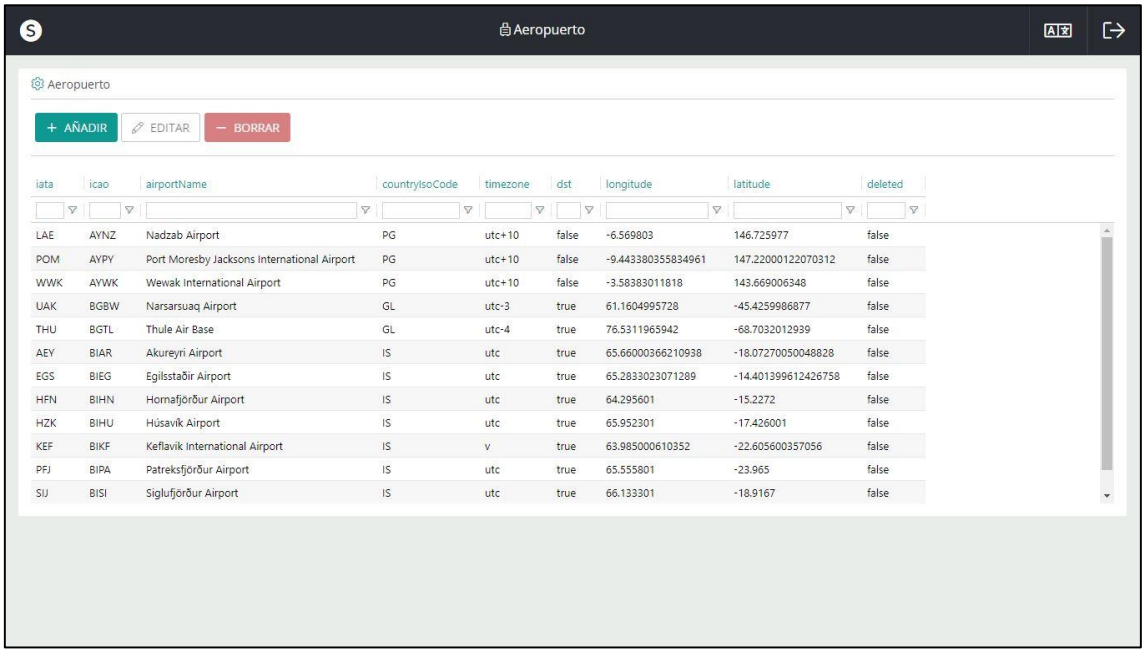


Ilustración 40 – Diseño de interfaces – Aeropuertos

La tabla que vemos, como hemos dicho es pequeña, pero previniendo que se van a manejar muchos datos, se ha desarrollado un buscador por columna para facilitar al usuario el trabajo de búsqueda.

Por un lado, tenemos la opción de añadir un nuevo aeropuerto, que tras clicar nos mostrara una ventana para rellenar los datos. Como vemos los botones de editar y borrar están deshabilitados mientras no se seleccione ninguna fila y tras seleccionar se activarán. Al clicar en editar, nos mostrará la misma venta que a la hora de añadir salvo que en este caso viene con los datos ya incluidos del aeropuerto seleccionado y nos permite modificarlos. Si clicásemos en borrar, nos aparecería una ventana de seguridad indicándonos a ver si estamos seguros de querer borrar, y de este modo evitar borrados accidentales.

Como a continuación veremos, la ventana que aparece tras clicar en añadir o editar, tiene ciertos límites, por ejemplo, el código Iata tiene que ser obligatorio de 3 caracteres o vacío (lo que equivale a “null”) y el Icao de 4 o vacío; el nombre del aeropuerto es un campo obligatorio; el código ISO del país tiene que tener 2 caracteres o vacío; y la zona horaria tiene 3 opciones: vacío, utc únicamente o utc y el resto de valores numéricos (utc ± 2 dígitos). Sin cumplir estos límites, la aplicación no dejara realizar ningún guardado:

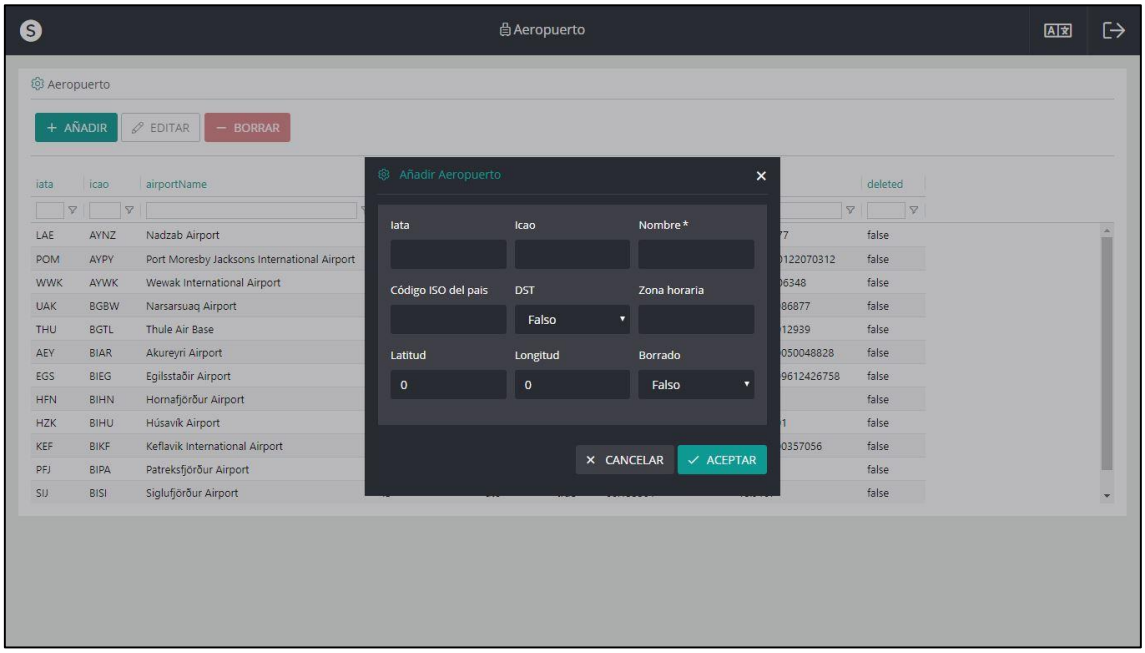


Ilustración 41 – Diseño de interfaces – Añadir aeropuerto

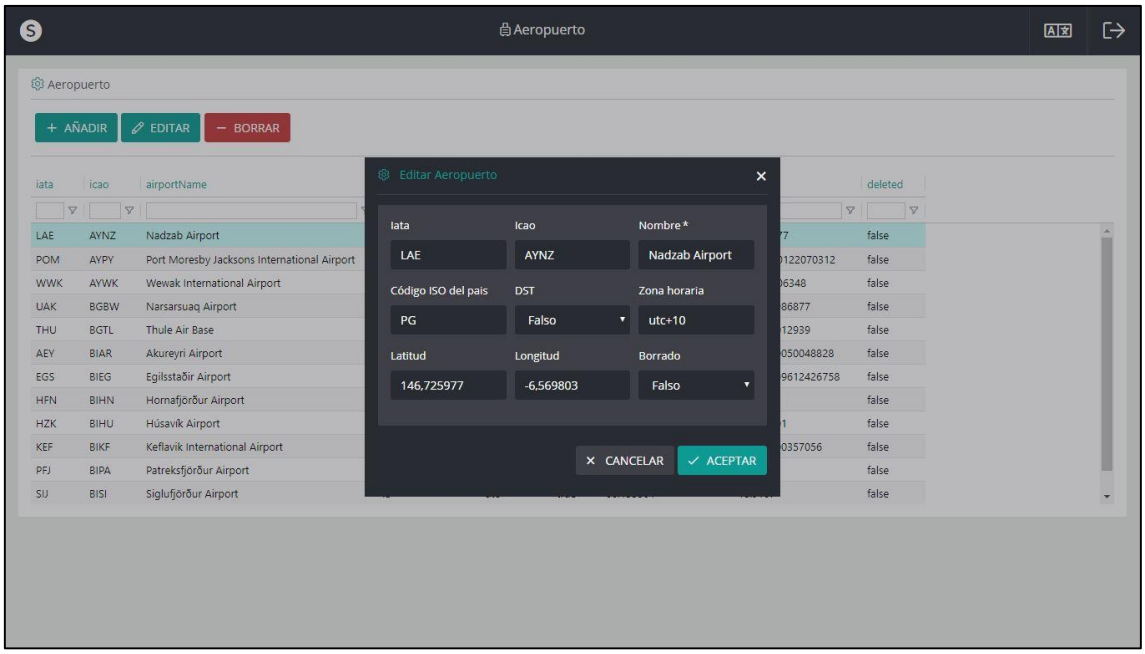


Ilustración 42 – Diseño de interfaces – Editar aeropuerto

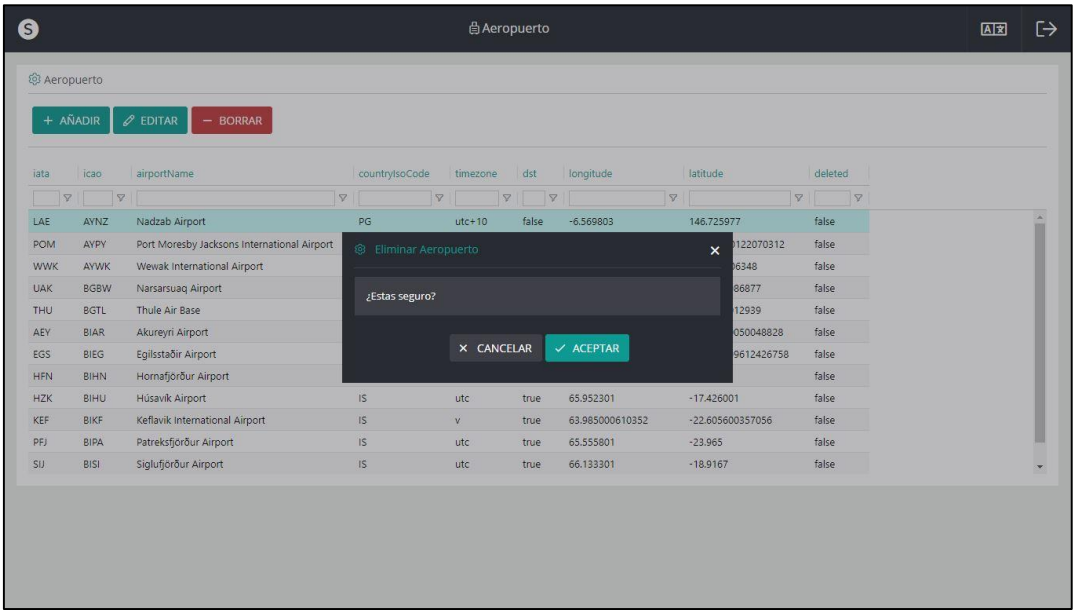


Ilustración 43 – Diseño de interfaces – Borrar aeropuerto

Para las aerolíneas y para las aeronaves, se ha creado una pantalla muy similar: la tabla que se muestra en cada pantalla está construida por los parámetros de cada caso y con el buscador para columna también.

En el caso de las aerolíneas, la ventana de añadir o editar tiene ciertas limitaciones distintas a la del aeropuerto: el código Iata tiene que ser vacío o de 2 caracteres y el Icao vacío o de 3; el nombre en este caso también es obligatorio y el código ISO del país también vacío o de 2 caracteres.

En el caso de las aeronaves, es igual que en el caso de los aeropuertos respecto al código Iata, código Icao y nombre, aunque en este caso, las aeronaves no cuentan con un código ISO del país, por lo tanto, no entra este límite.

En ambos casos, la lógica del borrado es idéntica al del aeropuerto:

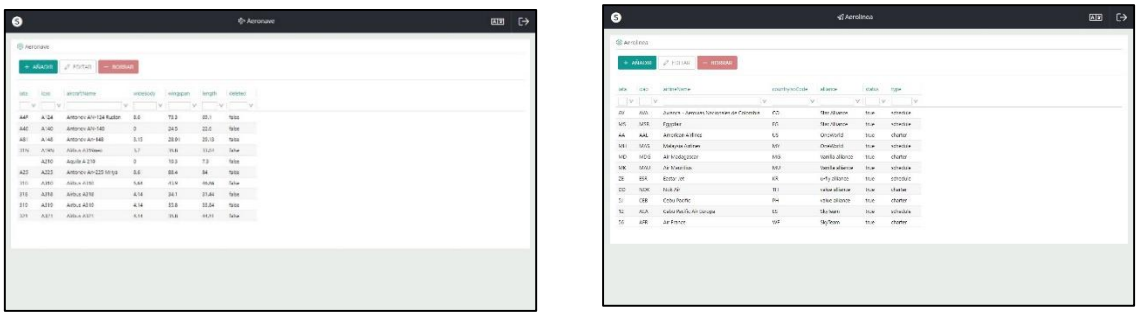


Ilustración 44 – Diseño de interfaces – Aeronaves/Aerolíneas

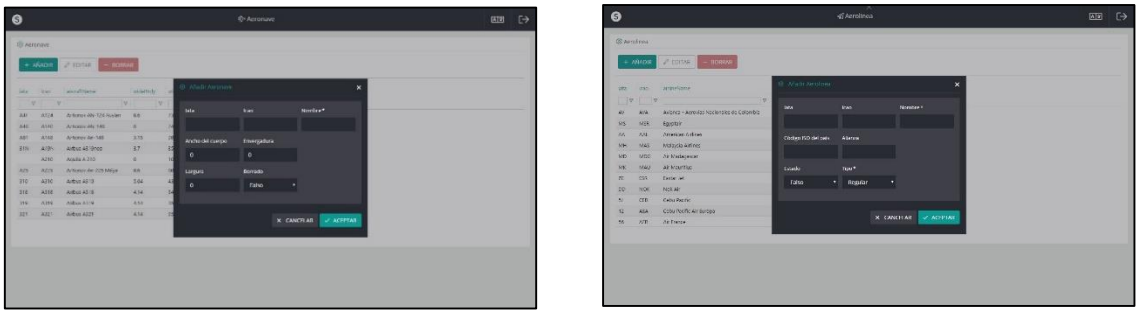


Ilustración 45 – Diseño de interfaces – Añadir aeronave/aerolínea

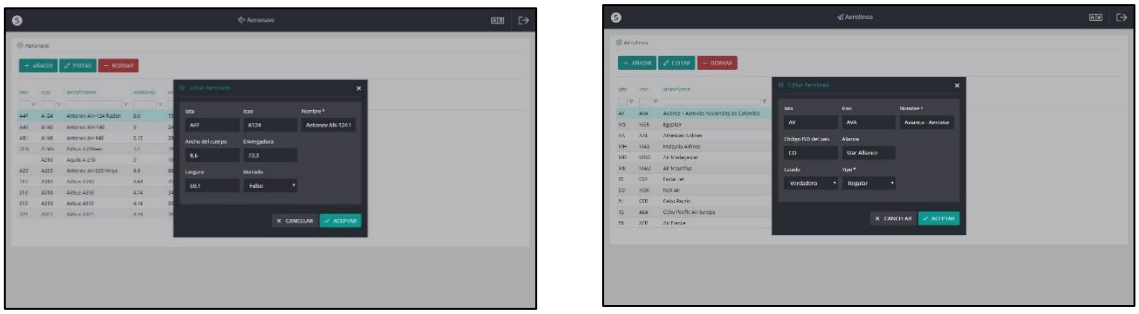


Ilustración 46 – Diseño de interfaces – Editar aeronave/aerolínea

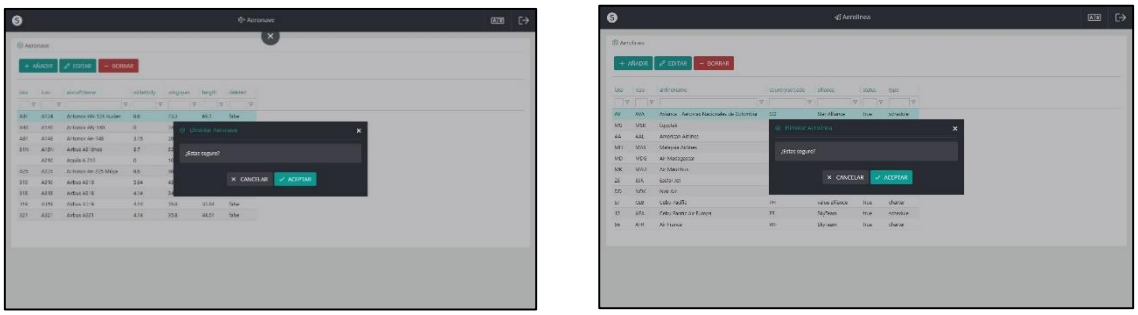


Ilustración 47 – Diseño de interfaces – Borrar aeronave/aerolínea

En la sección **Archivo** del menú, nos encontramos con la pantalla en la cual podemos subir los archivos para crear, editar o borrar los aeropuertos, las aerolíneas o las aeronaves. En la siguiente imagen podemos ver pantalla diseñada para esta funcionalidad, nos encontramos un botón a la izquierda del todo en el cual podemos descargarnos el modelo de datos para subir los archivos de manera correcta y de este modo evitar cualquier tipo de confusión:

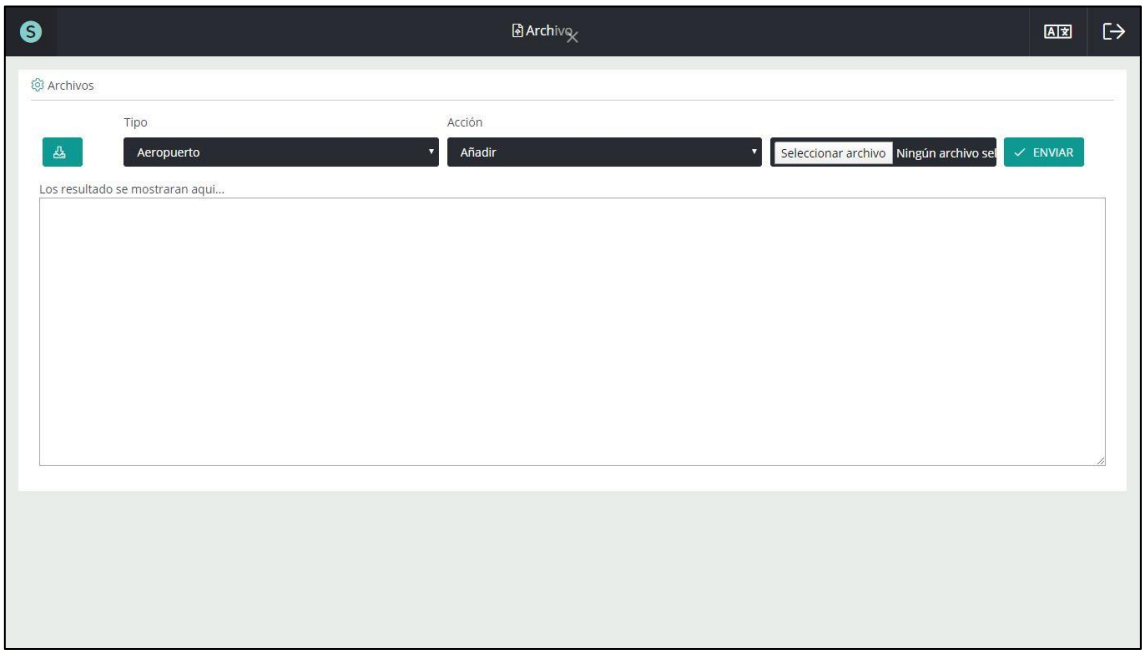


Ilustración 48 – Diseño de interfaces – Archivos

Siguiendo con la imagen de arriba, al lado derecho del boton de descargar modelos, nos encontramos con 2 desplegables en el cual hay que indicar que subida de archivo vamos a realizar, es decir, si es del tipo aeropuerto, aerolínea o aeronave, y si la accion va a ser añadir, modificar o borrar. Continuando, al lado derecho nos encontramos con el boton para buscar los archivos que queramos subir, en este caso, de momento solo se admite archivos tipo *csv*:

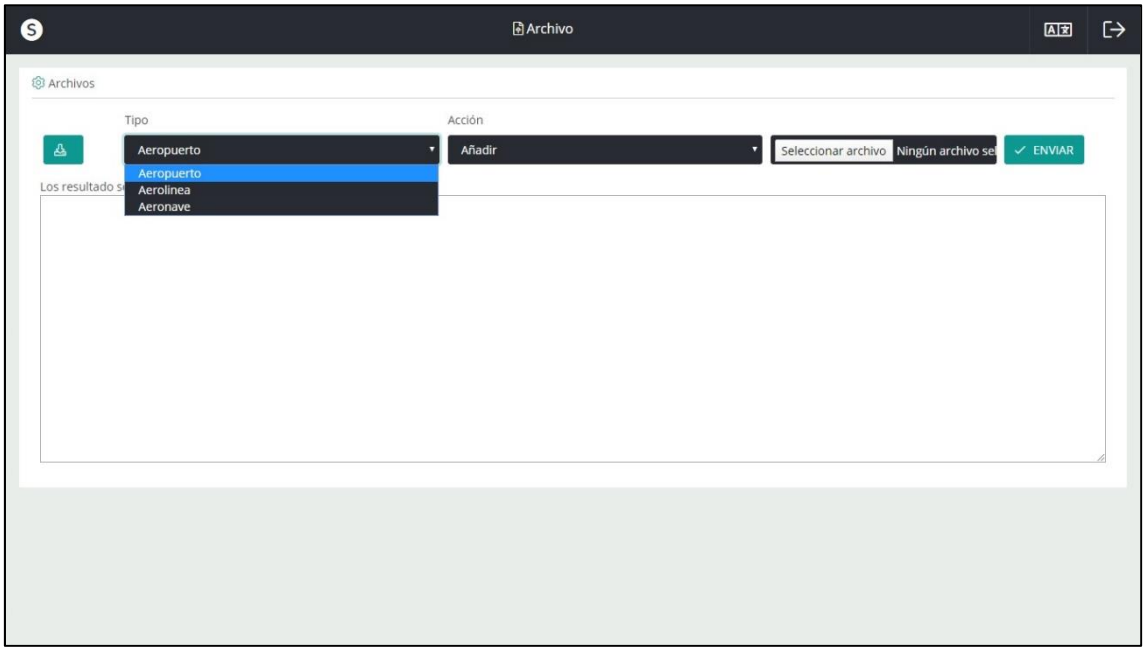


Ilustración 49 – Diseño de interfaces – Archivos selección tipo

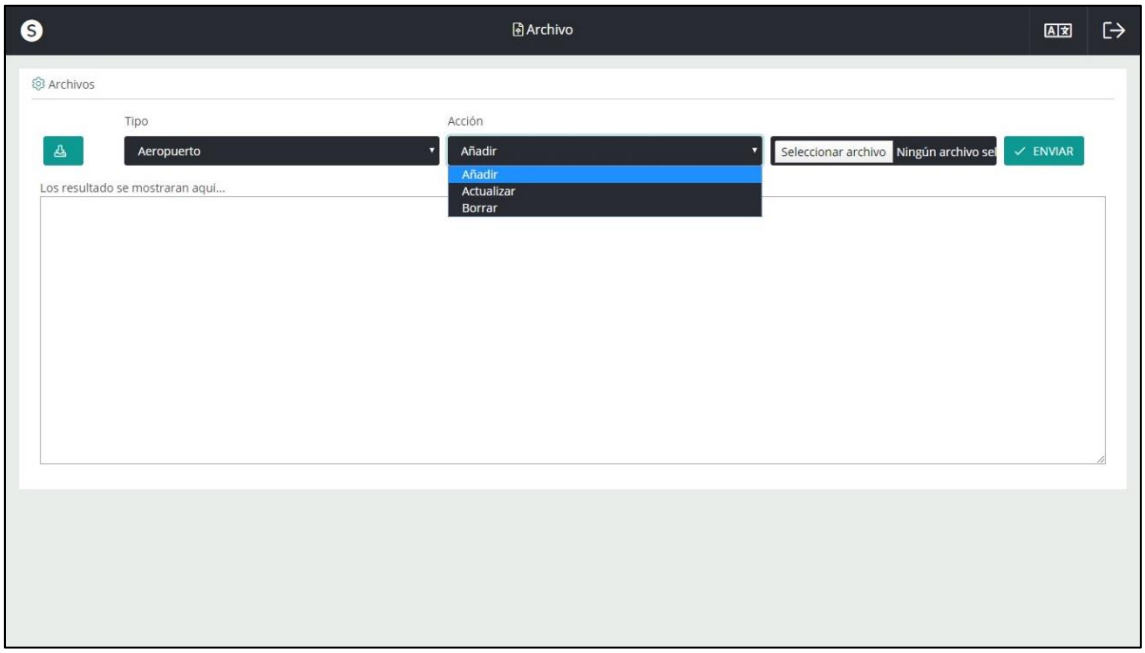


Ilustración 50 – Diseño de interfaces – Archivos selección acción

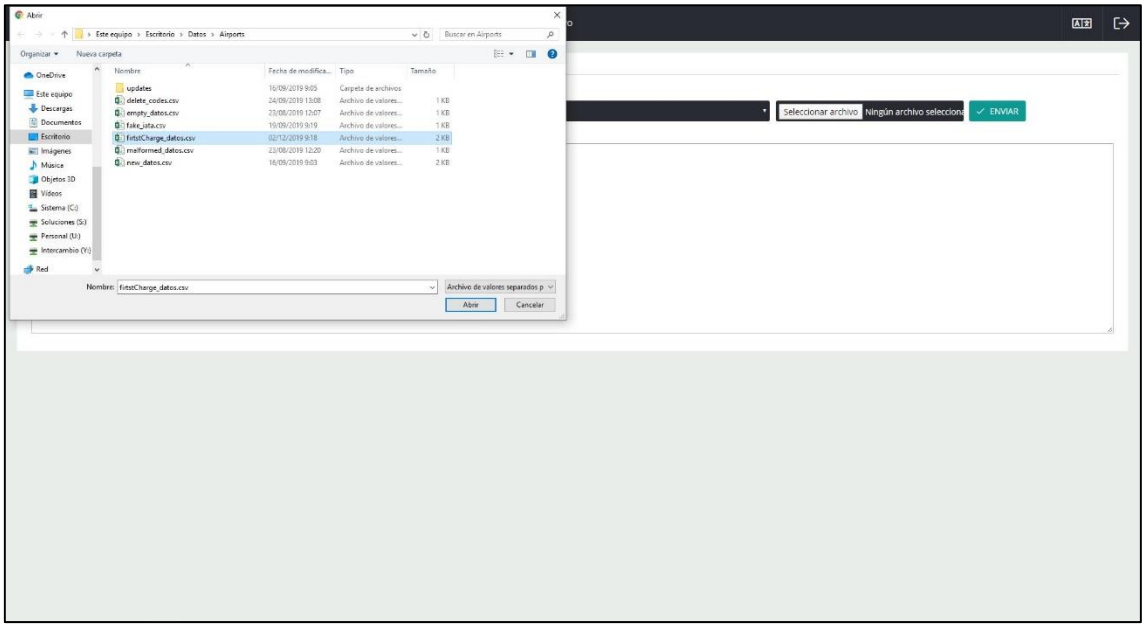


Ilustración 51 – Diseño de interfaces – Archivos selección archivo

Tras realizar la selección de tipo, accion y archivo, y tras clicar en el boton enviar, en el recuadro de abajo que tiene como titulo *Los resultados se mostraran aquí...*, se van a mostrar como bien dice el titulo, los resultado obtenidos de la subida de los archivos:

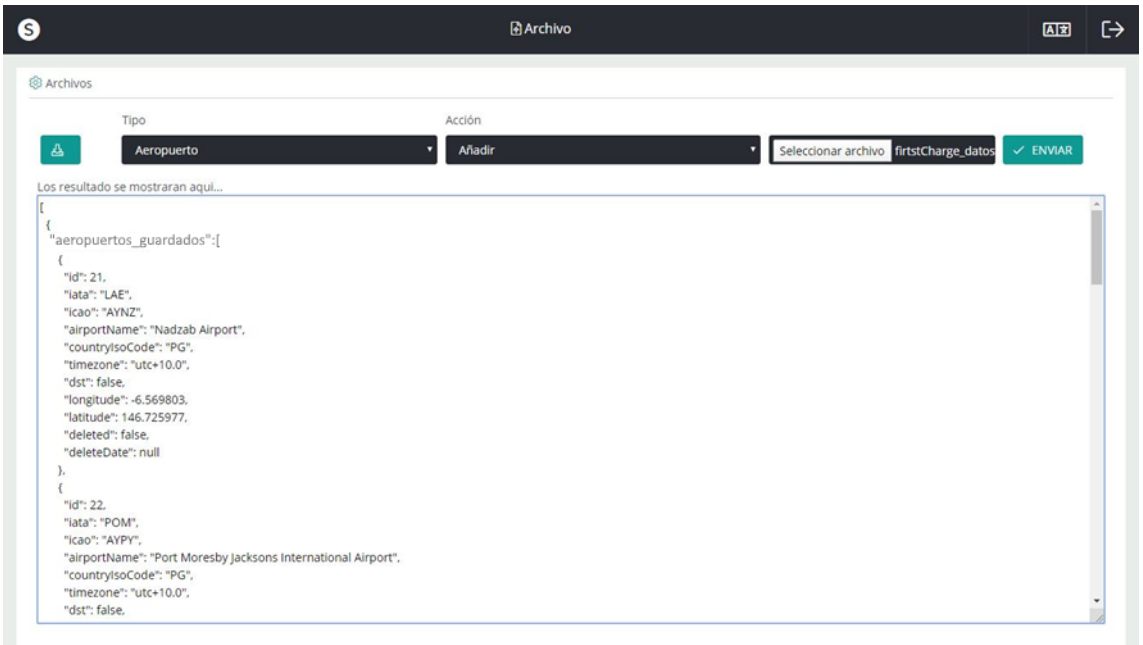


Ilustración 52 – Diseño de interfaces – Archivos resultado 1

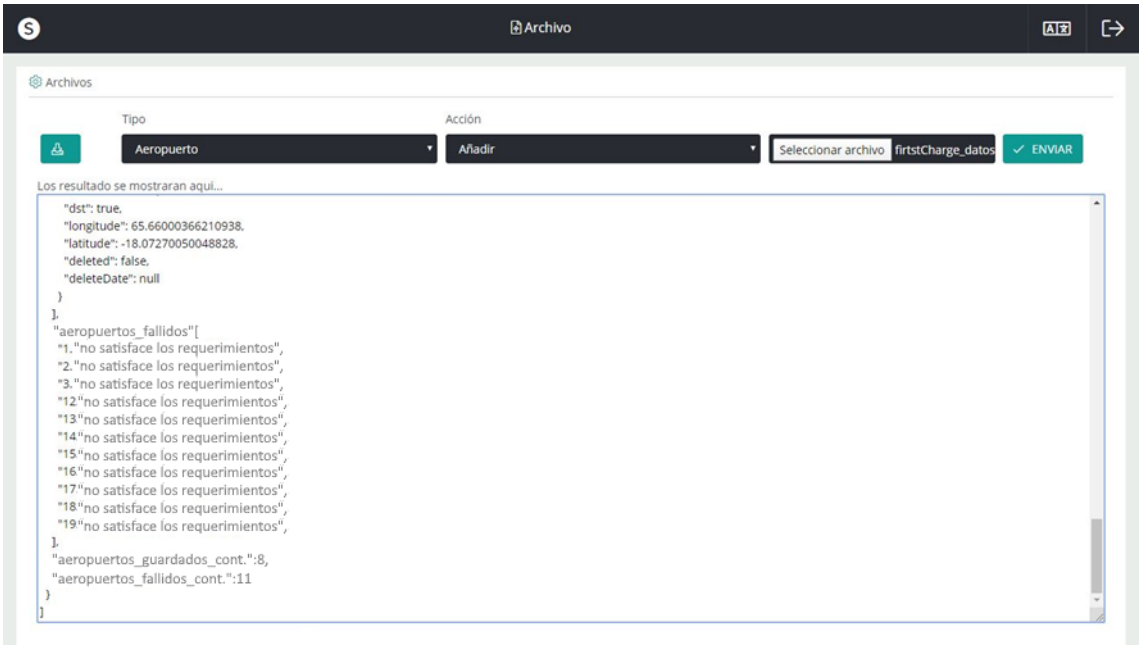


Ilustración 53 – Diseño de interfaces – Archivos resultado 2

Como se puede apreciar en las imágenes anteriores, tras realizar la subida de un archivo (firstCharge_datos.csv) para añadir aeropuertos nuevos, se ha probado con ciertos aeropuertos correctamente creados y otros con varios fallidos, de este modo se puede ver en la primera imagen que se crean aeropuertos y en la segunda, se ven que lineas han fallado en el guardado, y para finalizar un recuento de cuantos han funcionado y cuantos no.

Capítulo 5

Implementación

En este apartado se va hablar sobre el proceso seguido para realizar la implementación del proyecto y las herramientas utilizadas tanto para el desarrollo del Back end, Front end, calidad del producto y gestión de proyecto.

5.1 Aclaraciones previas

Por un lado, cabe destacar, que es la primera vez que realizaba una API. Gracias a las prácticas voluntarias realizadas en año pasado en la misma empresa, ya estaba un poco más relacionado con el uso de Spring Boot y el desarrollo de proyectos *Maven*. El uso del *POM*, aunque al principio resulta complicado por tema de tema de conflictos con algunas versiones de dependencias, al final resulta muy útil y sencillo de manejar.

En este proyecto, se ha utilizado un repositorio propio de la empresa IKUSI en la cual existen ciertos *plugins*, y de este modo evitar que, si en un *plugin* de algún repositorio común realizan cambios, estos cambios afecten a los productos previamente desarrollados y fallen o den algún tipo de problema inesperado.

Por otro lado, es la primera vez que usaba ReactJS, aunque con JavaScript ya se había trabajo antes y ReactJS sea una biblioteca de JavaScript, es bastante difícil de entender, pero, poco a poco se va aprendiendo sobre esta herramienta, y es cuando se descubre la capacidad inmensa de dicha herramienta.

Principalmente, ReactJS trabaja con componentes, los cuales pueden ser reutilizables. Para poder sacarle el mayor partido a la herramienta, se ha realizado un pequeño estudio para poder ver que componentes pueden ser reutilizables, y cuáles no.

Cabe añadir, que se ha ayudado a generar una base común para poder desarrollar el resto de interfaces web de la empresa a raíz de una base común. Para ello, se ha iniciado el proyecto analizando cada parte que puede ser común en todos los futuros proyectos y se ha creado esta base, que posteriormente, se ha continuado desde esa base común el desarrollo del Front end, viendo los puntos fuertes y las debilidades que ofrece la base común recién creada.

5.2 Configuración de entornos

5.2.1 Entorno para el Back end

Para un correcto desarrollo del proyecto y siguiendo la metodología de trabajo de la empresa, lo principal son configurar los entornos con los cuales se va a trabajar. Lo primero es configurar *GitLab*, para ello hay que generar un nuevo proyecto con su nombre y permisos a las personas correspondientes. Tras crear el proyecto hace falta crear la rama principal (*master*) en la cual se van a guardar las versiones finales, las cuales se entregarían al cliente, o en nuestro caso, siguiendo la metodología de Scrum, al *Product Owner.*

La rama *master*, al ser la rama de la cual se generan los entregables, hay que crear otra rama en la cual se pueda desarrollar el producto sin problema, para ello se crea la rama *develop*. En esta rama se pueden realizar las implementaciones o cambios de código, pero para un mejor control de versiones, por cada Historia de Usuario se crea una rama secundaria de la rama

develop, en la cual implementan los métodos o cambios necesarios para llevar a cabo la tarea asignada, y posteriormente unificar (*merge)* la rama con cambios y la rama *develop*.

Como siguiente paso, seria configurar el entorno de *Jenkins*, en el cual también habría que crear un proyecto nuevo con sus parámetros adecuados para el correcto despliegue del proyecto, indicándole donde se están implementados los test para que *Jenkins* pueda ejecutarlos y posteriormente pasar el *SonarQube*. Tras pasar los niveles de calidad, este servidor automáticamente cambia o guarda por primera vez el jar creado en el proyecto y lo despliega en el servidor de IKUSI.

Para finalizar, configuramos un *Docker* para poder levantar en él, un *SonarQube* que lo tengamos en nuestro pc para así evitarnos subidas de código innecesarias a *GitLab* y posteriormente que el *SonarQube* del servidor de IKUSI, nos detecte algún tipo de fallo y tengamos que reabrir la rama de implementación cerrada para realizar cambios. Gracias a este método, ahorramos muchísimo tiempo en la verificación del código fuente ya que no tenemos que estar a la espera de todo el proceso para ver los posibles errores. Aunque pueda parecer repetitivo tener que pasar 2 veces el *SonarQube*, es necesario ya que, de este modo, para el desarrollador es más rápido, ya que solo se realiza una subida a *GitLab.*

5.2.2 Entorno para el Front end

La configuración de este entorno, es bastante parecida a la configuración de entorno de la API. Los primeros pasos de crear el proyecto en GitLab y el despliegue en Jenkins son idénticos, lo único que les diferencia, es que, en esta parte del proyecto, debido a todavía no existir un buen motor de reglas para la comprobación del código respecto a bugs, código repetido… no se pasa el análisis de calidad del código *Sonarqube.*

5.3 Herramientas utilizadas

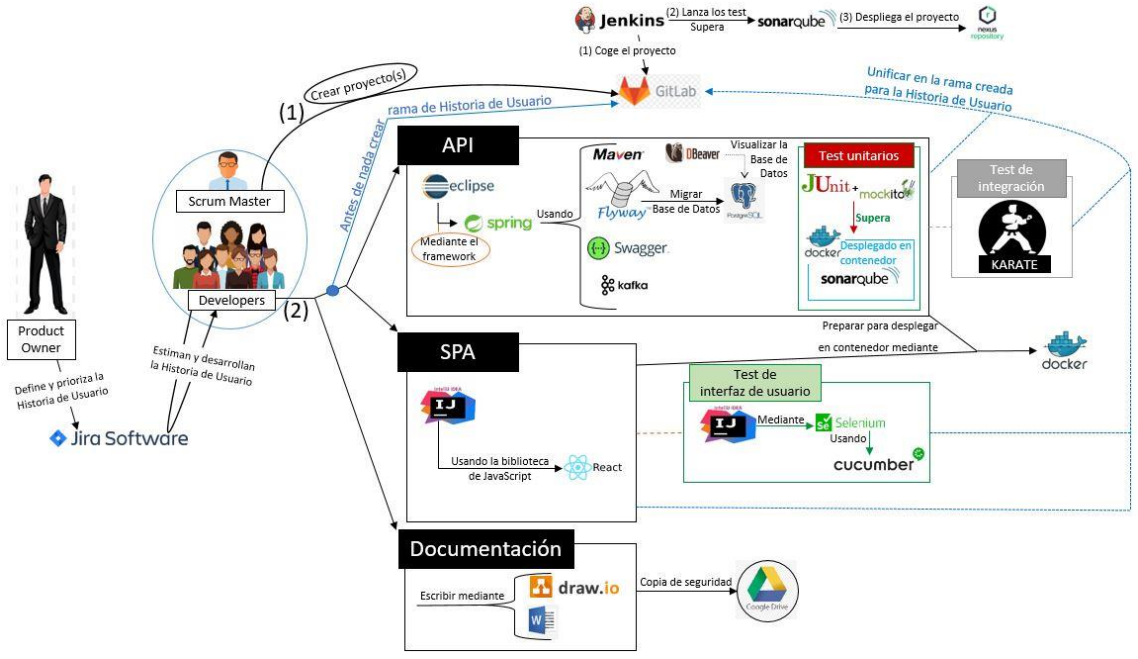


Ilustración 54 – Herramientas utilizadas

En el gráfico de arriba, podemos ver las herramientas que se han utilizado para llevar a cabo el proyecto, por ello a continuación, se van a explicar brevemente las herramientas utilizadas en cada fase del proyecto:

5.3.1 Gestión y despliegue del proyecto

5.3.1.1 Jira

Es una herramienta en línea para la administración de tareas de un proyecto, el seguimiento de errores e incidencias y para la gestión operativa de proyectos, permitiendo una mejora en la gestión y organización de flujos de trabajo.

Jira [2] se ha usado principalmente para el correcto desarrollo del proyecto consiguiendo funcionar de manera iterativa e incremental a la hora del desarrollo del código.

5.3.1.2 Jenkins

Es un servidor de automatización de código abierto escrito en Java. *Jenkins* [3] ayuda en la automatización de parte del proceso de desarrollo de software mediante integración continua y facilita ciertos aspectos en la integración continua.

Este servidor se ha usado principalmente para ejecutar la aplicación en el servidor y los test con cada historia unificada en la rama principal del desarrollo en *GitLab* y de este modo lanzar *SonarQube* para el control de calidad.

5.3.1.3 GitLab

Es un servicio web de control de versiones y desarrollo software colaborativo basado en *Git* [4].

Gracias a *GitLab* [5] conseguimos obtener varias ramas de versionado del proyecto las cuales posteriormente nos ayudaran a mejorar nuestra forma de trabajar en equipo y en caso de existir algún fallo a posteriori, poder localizarlo con facilidad según las versiones del código.

5.3.1.4 Nexus

Nexus [6] es un lugar de almacenamiento en el cual pueden ser recuperados e instalados los paquetes de software en un ordenador. Mediante varias herramientas distintas se pueden descargar e instalar estos paquetes.

En este proyecto, se ha usado el repositorio de la empresa Ikusi (Nexus Ikusi) para subir los proyectos. Esta subida se realiza creando una *imagen* del proyecto, es decir, el código binario.

5.3.1.5 Docker

Es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones múltiples en sistemas operativos.

Docker [7] se ha usado para lanzar la aplicación *SonarQube* en el ordenador de modo local y ver los fallos antes de subirlo a GitLab; y también, tras finalizar el proyecto completo, poder desplegar el conjunto de proyectos de manera local.

5.3.2 Desarrollo del código

5.3.2.1 Eclipse

Es una plataforma de software compuesta por un conjunto de herramientas de programación de código abierto multiplataforma. Soporta varios lenguajes de programación, aunque en este proyecto se ha usado Java. *Eclipse* [8] también tiene varias versiones de plataforma y en este desarrollo se ha usado *Eclipse Photon*.

Por otra parte, *Eclipse* tiene una gran comunidad de apoyo, esto y que en la universidad casi siempre se ha enseñado en esta plataforma, han sido una causa principal del uso de esta plataforma.

5.3.2.2 Spring Boot

Es un entorno de trabajo para el desarrollo de aplicaciones y sirve para Java, aunque Spring [9] tenga su propia plataforma, se ha optado por realizarlo en Eclipse. *Spring framework* contiene diferentes módulos que proveen un rango de servicios muy útiles para el proyecto:

- Contenedor de inversión de control: permite la configuración de componentes y la administración del ciclo de vida de los objetos Java.
- Acceso a datos: herramienta de mapeo objeto relacional.
- Gestión de transacciones: gestiona y coordina las transacciones de objetos Java.
- Modelo vista controlador: framework basado e HTTP para el desarrollo Web.
- Testing: soporta varios tipos de pruebas.

Con esta herramienta principalmente se ha conseguido eliminar posibles *bugs* y obtener un porcentaje de código cubierto con test unitarios. En conclusión, mejorar la calidad de código.

5.3.2.3 DBeaver

Es un cliente SQL y una herramienta de administración de Base de Datos (BD). Utiliza una API JDBC junto con un controlador JDBC para interactuar con los BD relacionales, y otros controladores patentados para BD no relaciones.

DBeaver [10] se ha utilizado para interactuar y configurar la Base de Datos creada para el almacenaje de los datos maestros.

5.3.2.4 PostgreSQL

Es un sistema de gestión de Base de Datos relacional orientado a objetos y de código abierto.

En este proyecto, PostgreSQL [11] se usará como Base de Datos principal en la cual se almacenarán todos los datos.

5.3.2.5 IntelliJ IDEA

Es un entorno de desarrollo integrado para el desarrollo de programas informáticos, está disponible en dos ediciones: edición para la comunidad (gratuita) y edición comercial. IntelliJ IDEA [12], aunque sea parecida a Eclipse, no está basada en dicha herramienta.

El uso de este entorno diferente se debe a un problema de compatibilidad entre la biblioteca que se usa para el desarrollo de interfaz y la plataforma Eclipse.

5.3.2.6 ReactJS

Es una biblioteca *JavaScript* de código abierto diseñada para crear *interfaces de usuario* con el objetivo de facilitar el desarrollo de aplicaciones de una sola página que usan datos que cambian todo el tiempo. Su objetivo es ser sencillo, declarativo y fácil de combinar. Esta biblioteca tiene varias características las cuales ayudan mucho al desarrollador a la hora de desarrollar interfaces:

- Virtual DOM: ReactJS [13] mantiene su propio DOM virtual, en lugar de confiar solamente en el DOM del navegador, permitiendo comparar el DOM anterior con uno actualizado.
- Las propiedades: Por otra parte, en ReactJS pueden definirse propiedades (más conocidas como “props”) como atributos de configuración que se definen en niveles superiores y luego pueden ser transferidos y modificados por la cadena de componentes que se pueden llegar a crear, cuando normalmente una vez definidos son inmutables.
- El estado: el estado de un componente permite definir un estado inicial de un componente en un momento concreto.
- Ciclos de vida: el ciclo de vida de los componentes son una serie de estados por los cuales pasa cada componente, esto nos ayuda a poder establecer un estado más concreto en cada momento de los componentes.

5.3.2.7 Apache Kafka

Kafka [14] es un proyecto de intermediación de mensajes desarrollado por LinkedIn y posteriormente donado a Apache escrito en Java y Escala. Tiene como objetivo proporcionar una plataforma unificada de alto rendimiento y baja latencia para la manipulación en tiempo real de fuentes de datos.

5.3.2.8 Swagger

Es un marco de software que ayuda a los desarrolladores a diseñar, construir, documentar y consumir servicios web REST. Swagger [15] incluye un soporte para la documentación automatizada, generada a raíz del código y casos de prueba.

En este proyecto se ha utilizado Swagger para la generación automatizada de la documentación.

5.3.2.9 Flyway

Flyway [16] es una herramienta que sirve para migrar Bases de Datos, es decir, sirve para la gestión de cambios incrementales, reversibles y de control de versiones de esquema de BD.

En este proyecto, se ha usado para migrar la BD de PostgreSQL mediante el lenguaje SQL. Esta migración se consigue mediante la migración de la versión del esquema de BD creado en la API, la cual recoge los cambios definidos en cada versión implementada en el proyecto y permite de este modo llegar a la versión de esquema deseada.

5.3.2.10 Maven

Es una herramienta de software para la gestión y construcción de proyectos Java. *Maven* [17] utiliza un *Project Object Model* (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

Una característica clave de *Maven* es que está listo para ser usado en red. El motor incluido en su núcleo puede dinámicamente descargar *plugins* de un repositorio, el mismo repositorio que provee acceso a muchas versiones de diferentes proyectos *Open Source* en Java, de *Apache* y otras organizaciones y desarrolladores.

Maven se ha utilizado para descargar las dependencias necesarias y poder desarrollar el proyecto de manera correcta.

5.3.3 Control de calidad

5.3.3.1 JUnit

Es un conjunto de bibliotecas utilizadas en programación para hacer pruebas unitarias de aplicaciones Java. *JUnit* [18] es un *framework* que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera, es decir, se observa si dando un valor de entrada al método, el valor de retorno es el esperado y de este modo comprobar que el método funciona correctamente.

5.3.3.2 Mockito

Mockito [19] es un marco de prueba de código abierto que permite la creación de objetos que verifiquen el correcto funcionamiento sin establecer expectativas de antemano.

En este proyecto se ha utilizado para desarrollar test unitarios completamente aislados sin establecer expectativas.

5.3.3.3 SonarQube

SonarQube [20] es una plataforma para evaluar código fuente. Es un software libre y usa diversas herramientas de análisis estático de código fuente para obtener métricas que pueden ayudar a mejorar la calidad del código de un programa.

5.3.3.4 Selenium

Selenium [21] es un entorno de pruebas de software para aplicaciones basadas en la web. Provee una herramienta para grabar y reproducir, y para crear pruebas sin usar un lenguaje de *scripting*. Las pruebas pueden ejecutarse usando la mayoría de los navegadores web en los diferentes sistemas operativos.

En este caso, usaremos en el sistema operativo de *Windows* con el navegador web *Google Chrome.*

5.3.3.5 Cucumber

Es una herramienta de software utilizada para el desarrollo basado en el comportamiento. Su lenguaje analizador llamado *Gherkin* permite que los comportamientos esperados del software se especifiquen en un lenguaje lógico que los clientes puedan entender. Cucumber [22] se ha utilizado en el proyecto para la generación de pruebas de interfaz de usuario.

5.3.3.6 Karate

Karate [23] es un marco de automatización de pruebas de API Web que pueden ejecutar llamadas a puntos finales de HTTP (*endpoint*) y afirmar que las respuestas XML o JSON son las esperadas. Karate se implementa en Java, pero los scripts están escritos en Gherkin, ya que en un principio fue una extensión de *Cucumber*.

5.3.4 Documentación

5.3.4.1 Draw.io

Es una tecnología de código abierto para la construcción de diagramas de las aplicaciones. Draw.io [24] se ha usado para la creación de los dominios y diagramas del proyecto.

5.3.4.2 Google Drive

Es un servicio de alojamiento de archivos gratuito, accesible a través del sitio web o desde aplicaciones.

Google Drive [25] se ha usado para obtener un alojamiento gratuito en el que realizar la copia de seguridad del proyecto. Para ello, se sube casi diariamente cada cambio generado tanto en el código como en la documentación.

5.3.4.3 Microsoft Word 2016

Es un programa informático orientado al procesado y edición de textos. Microsoft Word [26] se ha usado para la documentación y memoria del proyecto.

5.4 Desarrollo de Back end

A continuación, se van a mostrar algunas trazas del código implementado en el Back end:

El código siguiente hace referencia al controlador de aeropuerto, quien recibe la llamada que se realiza al *endpoint* y lo gestiona, en este caso observamos el tratamiento que recibe una subida de archivo.

```
1     @ApiOperation("First charge of airports file")
2    @PostMapping(path = "/airports/firstCharge")
3          public  ResponseEntity<Resource<Object>>  processFile(@RequestParam("file")
MultipartFile file) {
4           AirportResponse airportResponse = new AirportResponse();
5           List<Airport> airportList = new ArrayList<>();
6           File convFile = new File(file.getOriginalFilename());
7           try (FileOutputStream fos = new FileOutputStream(convFile);){
8                 fos.write(file.getBytes());
9                 airportList = responseFactory.getInstance(convFile);
10                int line=1;
11                for(Airport airport:airportList) {
12                      if(airport!=null) {
13                            airportResponse.add(service.charge(airport),line);

14                      } else {
15                            airportResponse.setOneMoreFail(line+LINE_STRING);
16                      }
17                      line++;
18                }
19                return ok(new Resource<>(airportResponse));
20          } catch (MalformedCSVException | IOException exception) {
21                LOGGER.error(FILE_EXCEPTION_MESSAGE, exception);
22                return                                    status(BAD_REQUEST).body(new
Resource<>(exception.getMessage()));
23          }
24    }
```

Ilustración 55 – Implementación – Desarrollo del Back end (Controller)

- En la línea 1 se define cual va a ser la operación a realizar para que *Swagger* la identifique.
- En la línea 2 definimos la ruta del *endpoint*.
- En la línea 6 el archivo enviado se pasa a bytes.
- En la línea 7 se envía a una subclase para conseguir los aeropuertos existentes en el archivo.
- De la línea 11 a la 18 se procesa cada aeropuerto obtenido intentado guardarlo en la BD.
- En la línea 19 si todo ha ido bien devuelve un OK con la lista de aeropuertos guardados y cuales han fallado.
- En la línea 22 devuelve un BAD REQUEST en caso de existir algún fallo en el archivo.

A continuación, se puede ver el proceso que se realiza para la actualización parcial de una aerolínea:

```
1    public Airline updatePartialSome(AirlinePatch patch) throws IOException{
2            try{
3                 ObjectMapper objectMapper = new ObjectMapper();
4                 Airline current = correctCodes(patch.getIata(),patch.getIcao());
5                 if (current==null) return null;
6                 current = objectMapper.readerForUpdating(current}
                              .readValue(objectMapper.writeValueAsBytes(patch.getMap()));
7                 repository.save(current);
8                 return current;
9            } catch (IOException exception){
10                return null;
11           }
12   }
```

Ilustración 56 – Implementación – Desarrollo del Back end (Service)

- En la línea 4 verificamos que los códigos enviados en el objeto creado a raíz de *Map<String, Object>*, son existen en la BD.
- En la línea 5 verificamos que no ha habido ningún problema, hay que tener en cuenta que puede ser que envíen 2 códigos nulos, y es un caso posible en el cual pueden llegar a existir más de una aerolínea con los 2 códigos nulos y en ese caso el método anterior nos devolvería un nulo.
- En la línea 6 realizamos la actualización de los campos enviados.
- En la línea 7 se guarda en la BD la aerolínea actualizada.

Como se puede ver, son clases sencillas de entender ya que esto era un requisito no funcional muy importante. Por eso, de un método complejo, se han creado varios métodos sencillos que simplifica la comprensión y reducen la complejidad.

5.4.1 Documentación del Back end

Para desarrollar la documentación del Back end, aparte de manuales de instalación y configuración, también se ha implementado una documentación online para cualquiera que tenga acceso a la API, es decir, gracias a la herramienta *Swagger-ui* se ha generado una documentación en la cual se muestra que tipo de datos gestiona la aplicación y cuáles son los *endpoints* existentes.

Esto principalmente se realiza para que, en un futuro, cuando una persona ajena al desarrollo del Back end realice una actualización del Front end, no le haga falta tener que ver el código fuente, y tener que estudiar y entender cada método generado, cómo y con qué parámetros se realizan las llamadas a los *endpoints* y cuáles serían los resultados obtenidos.

Para poder acceder a dicha documentación, se debe escribir ***swagger-ui.html*** al final de la dirección IP en la cual se encuentra la API ejecutándose.

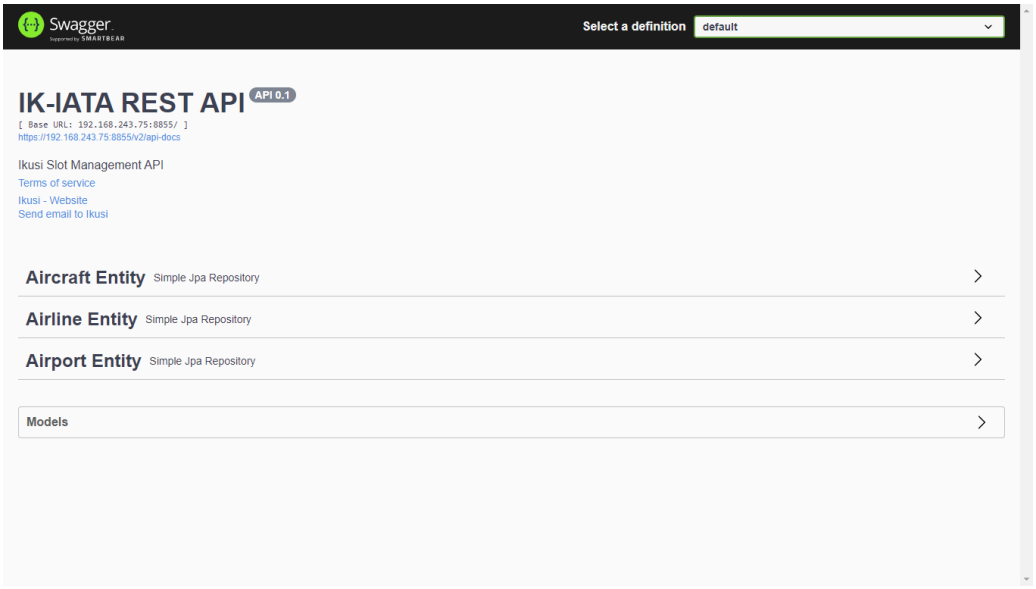


Ilustración 57 – Documentación de Back end – Pantalla principal swagger-ui

Como podemos ver en la imagen de arriba, esa es la página principal de *swagger-ui*. Para empezar, observamos cual es el nombre de la API y cuál es su versión, se puede ver cuál es la dirección IP en la que se encuentra la API, en nuestro caso en el servidor de IKUSI. También podemos ver los términos de uso, y el contacto con la empresa encargada. En la parte de abajo, se encuentran agrupados los endpoints, y después las entidades y atributos creados en la API:

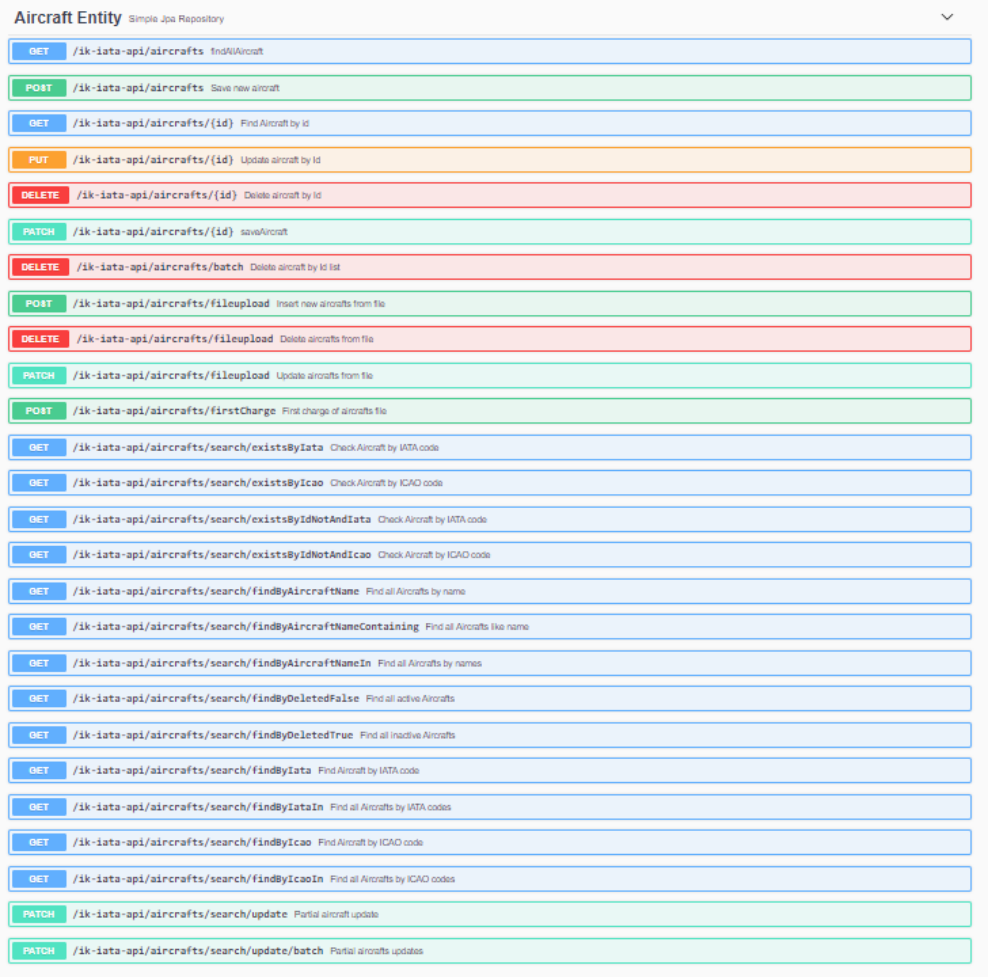


Ilustración 58 – Documentación de Back end – Pantalla endpoints swagger-ui

Como podemos observar de nuevo en la imagen de arriba, en cada grupo aparecen cuales son los endpoints existentes, echando un poco atrás, antes se ha comentado que, para la realización correcta de una API, un usuario de esta no debería de ser capaz de identificar que *endpoints* tiene una lógica diferente a la que se genera automáticamente *Spring-Boot.* Además, se le añade una pequeña descripción al lado para mejorar la comprensión:

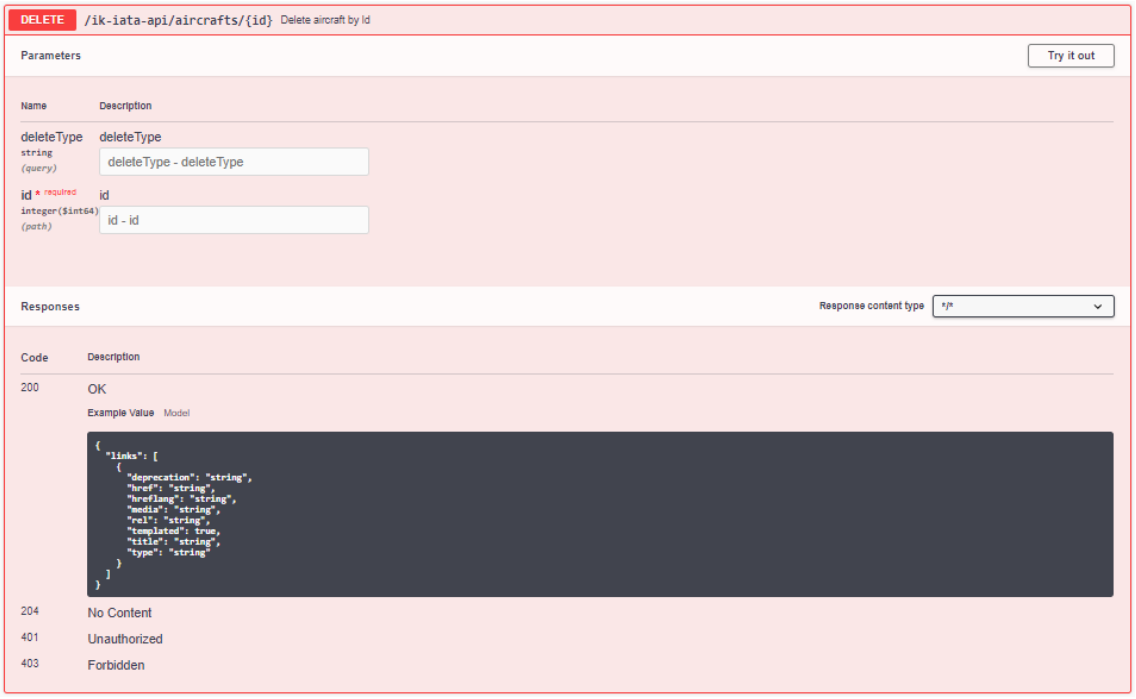


Ilustración 59 – Documentación de Back end – Pantalla pruebas swagger-ui

Continuando con *swagger-ui*, podemos ver en la imagen que cada método que aparece definido, tiene una gran ayuda para la comprensión total de cada método. Podemos ver qué tipo de parámetros necesita, si son obligatorios o no y cuáles serían las respuestas posibles; y para finalizar, se ve en la esquina derecha de el botón *Try it out*, gracias al cual podemos probar al momento el método y poder entenderlo por completo.

Por otra parte, se han desarrollado también los manuales de instalación, configuración y usuario para su correcto uso e instalación, que se añaden en el anexo C de esta memoria.

5.5 Desarrollo de Front end

Como se ha comentado en el punto *5.1 Aclaraciones previas*, el desarrollo de este Front end se empieza desde una base común previamente desarrollada gracias a la herramienta ReactJS. Para poder realizar este desarrollo de manera correcta se ha realizado un estudio previo respecto al funcionamiento de esta herramienta y como programar en esta herramienta, y aunque en un principio es difícil de comprender, tras varios días programando en ella las cosas que vuelven más sencillas y se empieza a comprender el potencial tan grande de esta herramienta.

Debido a cambios en la programación de cómo va a ser la aplicación, el tema de gestión de clientes se realiza a través de un proyecto de la empresa IKUSI llamado *IK-SS*, el cual se encarga de la gestión de clientes realizando varias verificaciones de certificado de seguridad, conectividad, permisos, *tokens*, … para ello se ha realizado una conexión con el proyecto *IK-SS* que creando previamente los usuarios, nos permite realizar una conexión segura con este proyecto y ahorrando un trabajo de verificación que ya está implementado. Al fin y al cabo desarrollar algo que ya está desarrollado y que es completamente funcional, es duplicar un trabajo ya existente y realizar un esfuerzo en vano. En la imagen de abajo se puede observar cual va a ser la arquitectura interna de la parte de Front end.

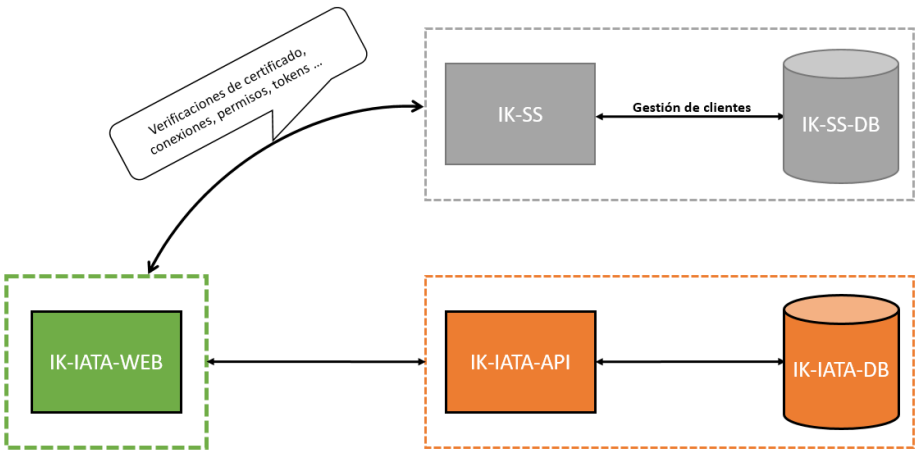


Ilustración 60 – Arquitectura SPA

Continuando con el desarrollo de la interfaz, se ha desarrollado de manera independiente respecto al Back end, es decir, sin usar ninguna lógica de negocio en la parte interfaz salvo la de gestión interna de las pantallas. El desarrollo de este Front end, se realiza en función de los *endpoints* y de las respuestas obtenidas.

Para finalizar, para una correcta estructura de proyecto, se ha realizado un estudio previo de los componentes que la interfaz puede tener en común para así desarrollar componentes genéricos y evitar duplicidades y exceso de código. Por ejemplo, las tablas se podían prever fácilmente que iban a ser muy similares, por lo tanto, se utilizó un componente de tabla genérico. Otro ejemplo claro serían los *inputs* de las ventanas de añadir y editar, y la ventana en si, por ello se creó una ventana común que dependiendo que botón realizase la llamada, realizar una función u otra, y los *inputs* se generaron varios componentes genéricos ya que de uno a otro si podían cambiar ciertas cosas (un *input* de texto y un *input* de selección) pero siempre siendo reusable.

En este caso, las clases si son más complejas y por eso se van a mostrar un componente de un *input* reutilizable. Si lo viésemos como un esquema, este *input* seria la parte más baja del esquema, lo que conlleva a ser del código más legible y sencillo que se puede encontrar:

```
1    class InputFile extends Component {
2      constructor(props) {
3         super(props);
4         this.state = {};
5       }
6      render() {
7        return (
8          <div className="form-row col-4">
9            <div>
10              <div className="form-row" encType="multipart/form-data"  >
11               <div className="form-row bg-dark rounded text-white qa-file"
style={{width:'70%',
12                                        marginTop:'29px', marginLeft:'10px'}}
>
13                  <input type="file" accept=".csv" style={{marginTop:'5px',
marginLeft:'5px'}}
14                                                        className="rounded"
onChange={this.props.onChange}/>
15               </div>
16                              <div  align="right"  style={{marginTop:'29px',
marginLeft:'10px'}}>
17                       <Submit className="text-uppercase ml-1 qa-btn-accept"
18                         icon="fal fa-check mr-2"
19                         loading={this.props.loading}
20                         text={this.props.submit}
21                         variant="primary"
22                         disabled={this.props.disabled}
23                         onClick={this.props.onSubmit}/>
24               </div>
25             </div>
26            </div>
27         </div>
28       );}}
29   InputFile.propTypes = {
30          nameOfClass: PropTypes.string.isRequired,
31          title: PropTypes.string.isRequired,
32          submit: PropTypes.string.isRequired,
33          onSubmit: PropTypes.func.isRequired,
34          onChange: PropTypes.func.isRequired
35   };
36   export default InputFile;
```

– *Implementación – Desarrollo del Front end (InputFile)*

- De la línea 2 a la línea 5 se inicia el componente que en este caso no tiene ningún atributo propio, pero mediante *props* se envían atributos definidos de la línea 29 a la línea 35.
- De la línea 6 a la línea 28 se muestra el código de JavaScript que se va a ejecutar.

Como se puede ver, este componente no tiene mucha complejidad y se debe a que es de los últimos eslabones de la cadena y no hay gran cosa. El resto de clases más complejas y rondan las 300-400 línea que sería inviable explicar.

5.6 Desarrollo de complementos

5.6.1 Apache Kafka

Para la elección de intermediario de comunicación, se ha realizado un estudio previo de qué tipo de intermediarios existen, sus estructuras… que se adjuntara en el anexo D.

Principalmente, se ha escogido Kafka como intermediario ya que ofrece todo lo que se busca como intermediario, y su estructura es la más apropiada para este proyecto. La estructura que nos ofrece es de *publicador/suscriptor*, es decir existe un componente que publica eventos/mensajes en un servidor de cola, el cual luego un componente que está suscrito al servicio, tiene acceso al servidor y le permite descargar el evento/mensaje de la cola. También Kafka está diseñada de modo centralizado con 1 servidor, es decir, solo existe un servidor al cual el publicador tiene que enviar los eventos/mensajes y los suscriptores tienen que acceder a este. Por ello, es lógico que cada suscriptor tenga acceso a todos los datos publicados en el servidor sin restricción alguna. En nuestro caso, mantener un servidor es lo más sencillo ya que no hace falta distinguir ningún tipo de dato, cada suscriptor tiene que tener acceso y luego cada suscriptor tiene que decidir por el mismo si le interesa descargar ese evento/mensaje.

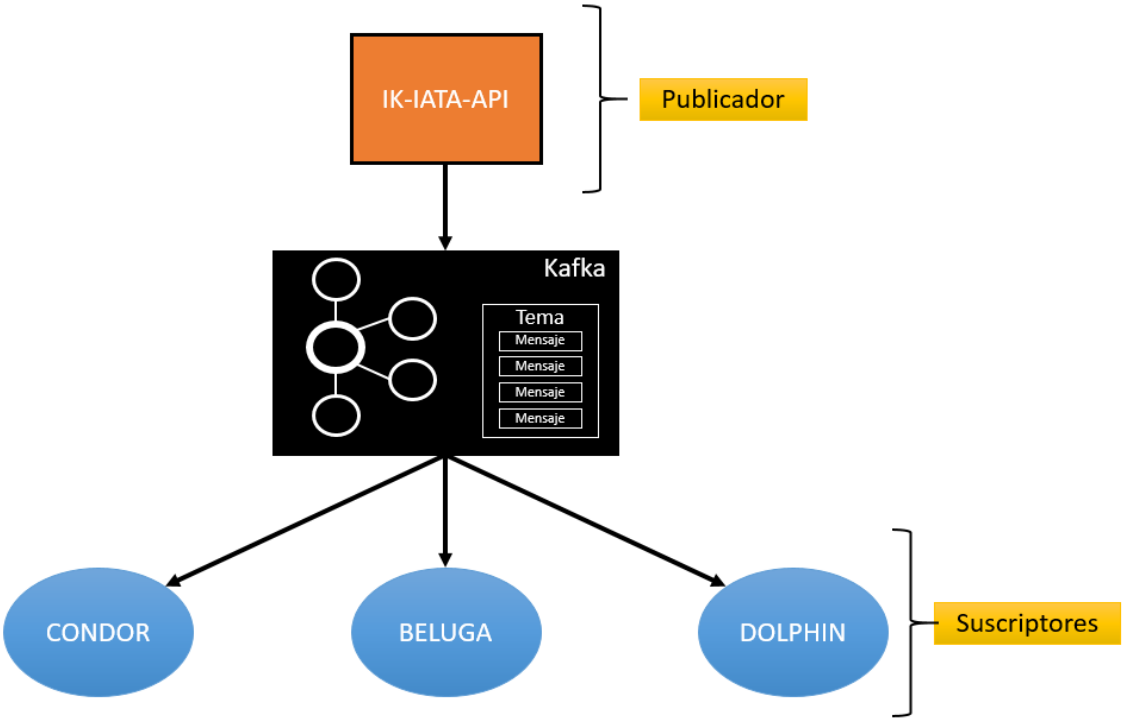


Ilustración 61 – Arquitectura Kafka

El tipo de conexión que necesitamos en nuestro proyecto es una conexión asíncrona puesto que cuando realizamos un cambio, no tienen porque estar los suscriptores conectados para recibir los cambios y un mensaje añadido a la cola no se elimina hasta que todos los suscriptores lo hayan visto o descargado, y si un suscriptor recibe el mensaje y viene otro nuevo mensaje, cojera el segundo mensaje de la cola puesto que el primero ya lo tendría.

Esta conexión se genera gracias a *Kafka-connect* que es un complemento que está escuchando todo el rato los cambios que lanza el publicador y de este modo los publica en el servidor, y este avisa a los suscriptores para que sean conscientes de que existen nuevos eventos/mensajes.

5.6.2 Docker

Como este proyecto será un producto en el cual se realicen en varios sitios implantaciones, se decidió crear un *docker-compose* para terminar de completar el proyecto. Docker, como antes se ha explicado, es un proyecto que automatiza el despliegue de imágenes de aplicaciones dentro de contenedores de software. En la parte de la API y el SPA se han desarrollado de manera que se generen imágenes y se suban al Nexus de Ikusi, para así poder desplegarlos en un Docker.

Para ello, se ha creado un proyecto que unifica todas las imágenes previamente creadas y subidas al *nexus* de IKUSI, y los despliega en un contenedor de Docker, esta unificación se denomina *docker-compose*. Estos despliegues, generalmente se usan en un servidor local, por ello, tanto en el Back end como en el Front end se ha creado un fichero de propiedades, que, a la hora de crear imágenes del proyecto, las direcciones IP y puertos sean las correspondientes.

Crear un despliegue del *docker-compose*, es un proceso algo sencillo ya que hoy en día existe bastante documentación al respecto, y en la propia empresa también se crean manuales de funcionamiento que ayudan bastante a la hora de realizar cosas de este tipo. Básicamente hay que crear un archivo *yml* indicando que imagen ha de descargarse, y si depende o no de otra imagen. En caso de depender, también hay que especificar la imagen necesaria, y esta segunda descargarla también. Existen ciertas variables de entorno recogidas en el fichero *.env* que define los parámetros de las rutas del *nexus* al cual hay que acceder para descargar las imágenes. En el caso de la Base de Datos, se crea una imagen del tipo de Base de Datos y posteriormente, mediante comandos se crea una configuración específica para el proyecto, así como, el usuario, contraseña y nombre de BD, y se rellena con los datos previamente creados a base de sqls en la imagen del Back end. *IK-SS* y *Kafka*, se generan de la misma manera.

En la siguiente imagen se muestra cual es la estructura de un archivo *yml* y como se puede ver es bastante sencillo crearlo siempre y cuando los proyectos estén correctamente creados y se tenga un conocimiento respecto a las imágenes necesarias:

```
version: "3"
services:

  ik-iata-spa:
    container_name: ik-iata-spa
    image: ${IMAGES_ROOT}/ik-iata-spa:${IK_IATA_SPA_VERSION}
    ports:
      - "80:80"
    depends_on:
      - ik-iata-api
      - ik-ss

  ik-ss:
    container_name: ik-ss
    image: ${IMAGES_ROOT}/ik-ss:latest
    ports:
      - 8980:8080
    environment:
      - TZ=Europe/Madrid
    network_mode: bridge
    links:
      - redis

  redis:
    image: redis:alpine
    container_name: redis
    ports:
      - 6379:6379
    network_mode: bridge

  ik-iata-api:
    container_name: ik-iata-api
```

```
    image: ${IMAGES_ROOT}/ik-iata-api:${IK_IATA_API_VERSION}
    ports:
      - "8855:8855"
    depends_on:
      - ik-iata-db

  ik-iata-db:
    container_name: ik-iata-db
    build:
      context: ./ik-iata-db
    environment:
      - DB_NAME=ikiatadb
      - TEST_DB_NAME=ikiatadb_test
      - DB_USER=ikiata
      - DB_PASSWORD=ikiata
      - POSTGRES_USER=postgres
    ports:
      - "54320:5432"

  zookeeper:
    image: wurstmeister/zookeeper

  kafka:
    image: wurstmeister/kafka
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: localhost
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
```

Ilustración 62 – Docker – Proyectos y dependencias

A continuación, se van a explicar el significado de los atributos arriba mostrados:

- container_name: cómo se va a llamar el contenedor a crear.
- image: ruta del nexus (definida en el *.env*)/nombre de la imagen : versión de la imagen.
- ports: puerto en el que se va a ejecutar.
- depends_on: nombre de la dependencia de otra imagen en esta imagen.
- enviroment: variables de entorno fijas.
- network_mode: modo de conexión con la imagen.
- links: atributo similar al de *depends_on*.
- builds: en este caso al no ser una imagen, como se tiene que generar este contenedor.

A continuación, se ven cuáles son los comandos que ejecuta este *docker-compose* respecto a la Base de Datos para la creación:

```
1  psql -v ON_ERROR_STOP=1 --username $POSTGRES_USER -c "CREATE ROLE
    ${DB_USER} with CREATEROLE INHERIT LOGIN SUPERUSER PASSWORD
    '${DB_PASSWORD}';"
2  psql -v ON_ERROR_STOP=1 --username $POSTGRES_USER -c "CREATE
    DATABASE ${DB_NAME}"
3  psql -v ON_ERROR_STOP=1 --username $POSTGRES_USER -c "GRANT ALL
   PRIVILEGES ON DATABASE ${DB_NAME} TO ${DB_USER};"
4  psql -v ON_ERROR_STOP=1 --username $POSTGRES_USER -c "CREATE
    DATABASE \"${TEST_DB_NAME}\";"
5  psql -v ON_ERROR_STOP=1 --username $POSTGRES_USER -c "GRANT ALL
    PRIVILEGES ON DATABASE \"${TEST_DB_NAME}\" TO ${DB_USER};"
```

Ilustración 63 – Docker – Definición Base de Datos

- En la línea 1 se crea un nuevo usuario (rol en PostgreSQL).
- En la línea 2 se crea la Base de Datos.
- En la línea 3 se le asignan todos los privilegios de la BD creada al usuario creado en la línea 1.
- En las líneas 4 y 5 se crea una Base de Datos de test y se le asigna al usuario creado.

Para finalizar, se ven cuáles son las variables de entorno que hay que definir en el archivo *.env:*

```
IMAGES_ROOT=nexus.soluciones.ikusi.com:8282

IK_IATA_API_VERSION=0.0.1-SNAPSHOT
IK_IATA_SPA_VERSION=0.0.1-SNAPSHOT
```

Ilustración 64 – Docker – Versiones proyectos

En este caso, la dirección que se encuentra las imágenes es el *nexus* propio de la empresa IKUSI (IMAGES_ROOT), y la versión de los proyectos que se van a usar.

Capítulo 6

Verificación y pruebas

A lo largo del desarrollo del producto, tanto en la parte del back end como en la parte del front end, se han generado una serie de test para verificar que todos y cada uno de los métodos e interfaces funcionen como se espera y que no ocurra ningún tipo de error inesperado.

Para ello se han realizado unos test que a continuación se explicaran uno por uno el procedimiento de generación de dichos test y las herramientas o complementos usados.

6.1 Pruebas unitarias

El desarrollo de estas pruebas se basa en la utilización de *Mockito* y *Junit*, con los que se han generado flujos de éxito y flujos de error, para así asegurarnos que en todo momento va a ocurrir algo esperado. El flujo de éxito (*Happy flow)* consiste en introducir datos para los cuales se ha creado ese método obteniendo el resultado esperado, es decir, que funciona bien y tal y como se esperaba. El flujo de error (*Error flow)* se introducen datos de manera incorrecta para corromper el método, y que este, al encontrarse con datos erróneos sepa actuar y devuelva una respuesta que no cause ningún tipo de bloqueo o error inesperado en el sistema, es decir, que ante un posible error sepa actuar y no lanzar excepciones inesperadas.

Cada método se ha probado independientemente del método que le llame o al método que este pueda llamar, es decir, de manera aislada. A la hora de que el método a probar realice una llamada a otro método es cuando actúa *Mockito* inyectando un resultado para así probar todas las variables posibles.

Para desarrollar correctamente los test unitarios, primero se ha generado un diseño de pruebas en el que cada método se estudia que valores pueden llegar como parámetros y ver cuál sería el resultado que se esperaría obtener.

Para evitar sobrecargar la memoria, se van a mostrar algunos test de las clases que forman parte de los diagramas de secuencia presentados en el punto *4.4 Diseño del Back end,* en resto de test se pueden encontrar en el anexo E.

Comenzamos con alguno de los test del punto *4.4.1. Gestión de aeropuertos* aquí, veremos los test relacionados por una parte con el controlador y otros con el repositorio. Veremos los casos de dar de baja en la clase del controlador y los de leer del repositorio:

Test dar de baja de AirportController

```
@DeleteMapping(path = "/airports/{id}")
public ResponseEntity<Resource<String>> delete(@PathVariable("id") Long id,
@RequestParam(required=false) String deleteType)
```

Entrada	Adecuado	No Adecuado
Long	Cualquier valor para id es correcto **(1)**	id<=0 **(2)**
		id>0 NO existente **(3)**
String	deleteType==”logical” **(4)**	deleteType!= ”logical” y deleteType!= ”permanent” y deleteType!="" **(7)**
	deleteType==”permanent” **(5)**	
	deleteType=="" **(6)**	

Suponiendo que:

- id = 1 si existe
- id = 76 no existe
- **palabraAleatoria** = cualquier palabra diferente a *logical* o *permanent*

Entrada	Equivalencia	Resultado
1, ”logical”	1, 4	OK (200)
1, ”permanent”	1, 5	OK (200)
1, “ “	1, 6	OK (200)
1, palabraAleatoria	1, 7	BAD REQUEST (400) y mensaje *no se puede borrar*
0, ”logical”	2, 4	IllegalArgumentException BAD REQUEST (400)
0, ”permanent”	2, 5	IllegalArgumentException BAD REQUEST (400)
0, “ “	2, 6	IllegalArgumentException BAD REQUEST (400)
0, palabraAleatoria	2, 7	BAD REQUEST (400) y mensaje *no se puede borrar*
76, ”logical”	3, 4	NotFoundException BAD REQUEST (400)
76, ”permanent”	3, 5	NotFoundException BAD REQUEST (400)
76, “ “	3, 6	NotFoundException BAD REQUEST (400)
76, palabraAleatoria	3, 7	BAD REQUEST (400) y mensaje *no se puede borrar*

Ilustración 65 – Prueba unitaria – Dar de baja

Test de AirportRepository (varios test de leer)

Para poder realizar estas pruebas, se necesita levantar parte de la API, y una de la Base de Datos especial de test, pre-insertando algunas líneas para poder testear los métodos de manera correcta. Como estos métodos son generados mediante *Spring-Boot*, no son se muestra el diseño de test creado, y, en su caso, se muestran pruebas realizadas para comprobar que devuelve lo esperado.

```
public void testFindByIata()
```

Entrada	Respuesta esperada	Resultado
`repository.findByIata("");`	Devuelve una lista vacía	OK
`repository.findByIata("MAG");`	Devuelve una lista con 1 elemento	OK

Ilustración 66 – Prueba unitaria – Encontrar mediante Iata

```
public void testFindByIcao()
```

Entrada	Respuesta esperada	Resultado
`repository.findByIcao("");`	Devuelve una lista vacía	OK
`repository.findByIcao("AYMD");`	Devuelve una lista con 1 elemento con el *Id=9L*	OK

Ilustración 67 – Prueba unitaria – Encontrar mediante Icao

```
public void testFindByAirportName()
```

Entrada	Respuesta esperada	Resultado
`repository.findByAirportName("AAA");`	Devuelve una lista vacía	OK
`repository.findByAirportName("Thula Airport");`	Devuelve una lista con 1 elemento con el *Id=8L*	OK

Ilustración 68 – Prueba unitaria – Encontrar mediante nombre

```
public void testFindById()
```

Entrada	Respuesta esperada	Resultado
`repository.findById(100L);`	Devuelve un objeto **Optional<>** vacío	OK
`repository.findById(3L);`	Devuelve un objeto **Optional<>** no vacío	OK

Ilustración 69 – Prueba unitaria – Encontrar mediante ID

```
public void testFindDeletedTrue ()
```

Entrada	Respuesta esperada	Resultado
`repository.findDeletedTrue ()`	Devuelve una lista con 5	OK

Ilustración 70 – Prueba unitaria – Encontrar cuando este borrado

Cambiamos a alguno de los test del punto *4.4.2 Gestión de aerolíneas* y en el *4.4.3 Gestión de aeronaves*, aquí, veremos los test relacionados con el servicio. Veremos los casos dar de alta aerolíneas y modificar aeronaves:

Test guardar o actualizar aerolíneas

`public Airline save(Airline airline) throws NotFoundException, DuplicatedConstraintException`

Entrada	Adecuado	No Adecuado
Airline (creando un nuevo registro)	Airline != null **(1)**	Airline ==null **(2)**
	Airline con iata e icao inexistentes **(3)**	Airline con iata o icao existentes **(4)**
Airline (actualizando un objeto existente)	Airline!=null **(5)**	Airline==null **(6)**
	Airline actualización de iata o icao inexistentes **(7)**	Airline!=null inexistente **(8)**
	Airline actualización del resto de atributos **(9)**	Airline actualización de iata o icao existentes **(10)**

Suponiendo que:

- **airline_C** = {iata= "MG", icao="MDG", airlineName="Air Madagascar" …}
- **airline_C_Repe** = {iata= "AA", icao="AAL", airlineName="American Airlines" …}
- **airline_C_RepeIata** = {iata= "AA", icao="TLD"…}
- **airline_C_RepeIcao** = {iata= "TL", icao="AAL"}
- **airline_U** = {iata= "UX", icao="AEA", airlineName="Cebu Pacific Air Europa" …}
- **airline_U_NF** = {iata= "NF", icao="NFD", airlineName="Not Found Airlines" …}
- **airline_U_Repe** = {iata= "AA", icao="AAL" …}
- **airline_U_RepeIata** = {iata= "AA", icao="AEA" …}
- **airline_U_RepeIcao** = {iata= "UX", icao="AAL" …}

Entrada	Equivalencia	Resultado
airline_C	1, 3	Devuelve el objeto guardado
airline_C_Repe	1, 4	DuplicatedConstraintException
airline_C_RepeIata	1, 4	DuplicatedConstraintException
airline_C_RepeIcao	1, 4	DuplicatedConstraintException
airline_U	5, 7, 9	Devuelve el objeto guardado
airline_U_NF	5, 10	NotFoundException
airline_U_Repe	5, 8	DuplicatedConstraintException
airline_U_RepeIata	5, 8	DuplicatedConstraintException
airline_U_RepeIcao	5, 8	DuplicatedConstraintException
null	2, 6	NotFoundException

Ilustración 71 – Prueba unitaria – Guardar o actualizar

Test modificación parcial de aeronaves

En este caso, por *AircraftPatch* nos referimos a una clase especial creada para cuando se recibe un objeto tipo *Map<String, Object>* en el *endpoints* y previamente ha sido trato para obtener esta clase.

`public Aircraft updatePartial(AircraftPatch patch) throws IOException`

Entrada	Adecuado	No Adecuado
AircraftPatch	AircraftPatch != null **(1)**	AircraftPatch == null **(2)**
	iata sea igual a algún valor **(3)**	iata == null **(5)**
	icao sea igual a algún valor **(4)**	
	iata == "NotExists" **(6)**	icao == null **(8)**
	icao == "NotExists" **(7)**	
	iata exista en BD y la lista obtenida solo sea de un elemento **(9)**	iata no exista en BD **(10)**
	iata exista en BD y la lista obtenida de varios elementos y icao exista en la BD **(11)**	iata exista en BD y la lista obtenida sea de varios elementos y icao no exista **(12)**
	icao exista en BD **(13)**	icao no exista en BD **(14)**

Suponiendo que:

- **patch_1** = {iata="321", icao="A321" …} y obtenga una lista con un elemento
- **patch_2** = {iata="321", icao="NotExists" …} y obtenga una lista con un elemento
- **patch_3** = {iata="321", icao="A321" …} y obtenga una lista con varios elementos
- **patch_4** = {iata="321", icao="NotExists" …} y obtenga una lista con varios elementos
- **patch_5** = {iata="NotExists", icao="A321" …}
- **patch_6** = {iata="NotExists", icao="NotExists" …}
- **patch_7** = {iata="321 } y obtenga una lista con varios elementos
- **patch_8** = {iata="321…} y obtenga una lista con varios elementos e icao no existe
- **patch_9** = {iata=null …}
- **patch_10** = {icao=null …}
- **patch_11** = {iata="A4F", icao="A321"…}
- **patch_12** = {iata="A4F", icao="A124"…}

Entrada	Equivalencia	Resultado
patch_1	1, 3, 4, 8, 13	Devuelve el *Aircraft* actualizado
patch_2	1, 3, 7, 9	Devuelve el *Aircraft* actualizado
patch_3	1, 3, 4, 11	Devuelve el *Aircraft* actualizado
patch_4	1, 3, 7, 12	Null
patch_5	1, 4, 6, 13	Devuelve el *Aircraft* actualizado

patch_6	1, 6, 7	Null
patch_7	1, 3, 12, 13	Null
patch_8	1, 3, 12	Null
patch_9	1, 5	Null
patch_10	1, 8	Null
patch_11	1, 3, 4, 10, 13	Devuelve el *Aircraft* actualizado
patch_12	1, 3, 4, 10, 14	Null
null	2	Null

Ilustración 72 – Prueba unitaria – Actualización parcial

Para finalizar con los test respecto al punto *4.4.5 Gestión de ficheros*, mostraremos un test relacionado con el controlador de aeropuertos:

```
@PostMapping(path = "/airports/fileupload")
public ResponseEntity<Resource<Object>> newAirportsFromFile
(@RequestParam("file") MultipartFile files)
```

Entrada	Adecuado	No Adecuado
MultipartFile	file!=null **(1)**	file==null **(2)**
	file bien formado **(3)**	file malformado **(4)**

Suponiendo que:

- **file** = es un archivo ".csv" debidamente formado según la estructura predefinida para la subida de archivos del tipo aeropuerto.
- **fileNull** = es un archivo incorrectamente creado o mal subido al servicio.
- **fileMalformado =** es un archivo ".csv" creado sin seguir la estructura predefinida.

Entrada	Equivalencia	Resultado
file	1, 3	OK (200)
fileNull	2	MalformedException BAD REQUEST (400)
fileMalformado	4	MalformedException BAD REQUEST (400)

Ilustración 73 – Prueba unitaria – Dar de alta desde archivo

6.2 Pruebas de interfaz de usuario

Para el desarrollo de estas pruebas de interfaz de usuario, se ha basado en la utilización del entorno de pruebas para aplicaciones web *Selenium*, junto con la herramienta *Cucumber* que admite el desarrollo basado en comportamientos, escrito en el lenguaje *Gherkin*. Este lenguaje está diseñado para ser legible y no técnico, es decir, se define una serie de pasos lógicos que la aplicación web debería de realizar.

Más información sobre el proceso de desarrollo de estos test se puede encontrar en el anexo F.

Para el desarrollo de estos test, se han diseñado unos casos de prueba sobre que esperamos recibir en cada llamada de los *endpoints* y cómo reacciona la interfaz. Como veremos a continuación para empezar aparecerá en negrita cual es nombre de la acción que se desea

hacer, y después, aparecerá una tabla describiendo el nombre del escenario, los pasos que realiza y resultado obtenido.

Para evitar sobrecargar la memoria con excesos de test, solo se van a mostrar uno pocos y en el anexo G:

COMO coordinador QUIERO crear un nuevo los aeropuertos

Escenario de prueba	Descripción	Resultado
Crear airport con datos correctos	When se rellenan los campos de "airport" con los datos "airport_default_value" And pulso el botón "accept" en la modal "add" de "airport" Then se muestra un mensaje de correcto And el airport generado a partir del set de datos "airport_default_value" se ha creado correctamente	PASA
Crear airport con datos duplicados	When se rellenan los campos de "airport" con los datos "airport_duplication_value" And pulso el botón "accept" en la modal "add" de "airport" Then se muestra un mensaje de duplicado	PASA
Comprobación de campo obligatorio "<field>" al pulsar el botón "accept"	When se rellenan los campos de "airport" con los datos "airport_default_value" And completo el campo "<field>" con el valor "" en la modal "add" de la pantalla "Airport" And pulso el botón "accept" en la modal "add" de "airport" Then se muestra un mensaje de campo obligatorio And el campo obligatorio "<required_field_class>" se muestra en rojo	PASA
Comprobación de resultados erróneos	When se rellenan los campos de "airport" con los datos "airport_error_size" Then el tamaño máximo de caracteres del campo "<input_field>" de la modal "add" de "airport" es "<number>" y sub-cadena de "airport_error_size"	PASA
Comprobación de respuestas para datos erróneos	When se rellenan los campos de "airport" con los datos "airport_iata_icao_error_value" And pulso el botón "accept" en la modal "add" de "airport" Then se muestra un mensaje de error	PASA

Ilustración 74 – Prueba de interfaz – Crear nuevo aeropuerto

COMO coordinador QUIERO dar de baja un aeropuerto

Escenario de prueba	Descripción	Resultado
Mostrar modal de borrado de Airport	When selecciono la fila con el dataset "airport_delete" en la tabla "airport" de la pantalla "airport" And pulso el botón "delete" de la ventana de "airport" And aparece la ventana modal "delete" de "airport" And pulso el botón "accept" en la modal "delete" de "airport" Then se muestra un mensaje de correcto Then no encuentra la fila con el dataset "airport_delete" en la tabla "airport" de la pantalla "airport"	PASA

Ilustración 75 – Prueba de interfaz – Dar de baja aeropuerto

6.3 Pruebas de integración

Para asegurar una mayor calidad del producto generado, se han desarrollado pruebas respecto a los *endpoints* para asegurar el estado y su función, es decir, comprobar si siguen estando activos y si a la hora de realizar la llamada recibimos una respuesta esperada. Para ello, se han generado en el lenguaje *Gherkin (Cucumber)* con el marco de automatización de pruebas de API Web, *Karate*. Estas pruebas se desarrollan siguiendo la estructura de las pruebas de automatización de interfaz, a continuación, mostramos algún ejemplo, y posteriormente, mostraremos el reporte generado por *Cucumber*:

Escenario de prueba	Descripción	Resultado
upload multipart/form-data file	Given path <basePath> + '/fileupload' And def value = 'testLoad' And multipart file file = { value: '#(value)', filename: 'testLoad.csv'} And header Content-Type = 'multipart/form-data' And multipart field message = <message> When method post Then status 200	Devuelve 200

Ilustración 76 – Prueba de integración – Carga de archivo

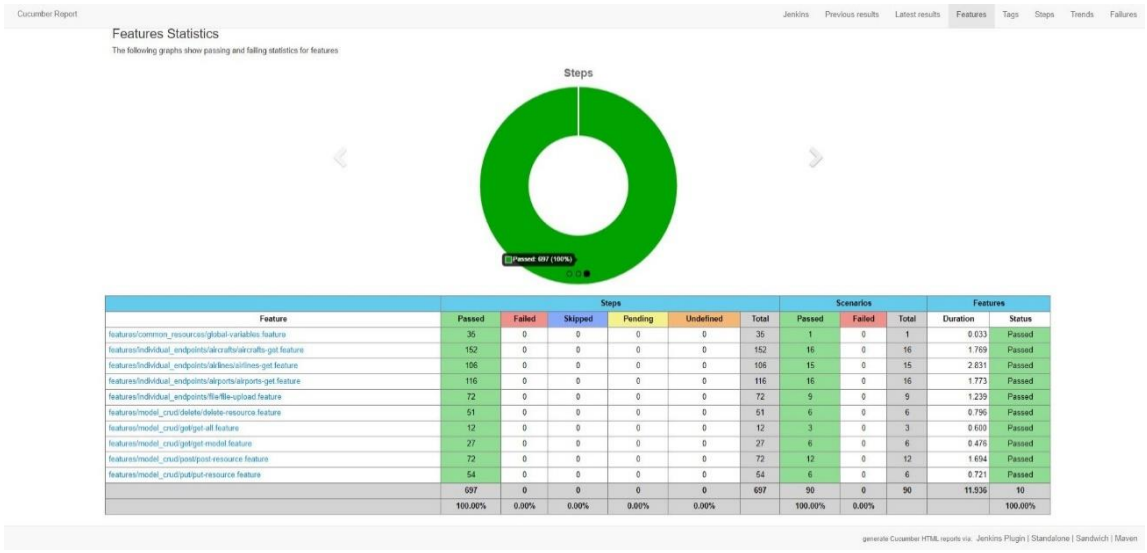
Escenario de prueba	Descripción	Resultado
Find Airline by Iata Code [Found]	Given path basePath + '/search/findByIata' And param iata = global.airlineIata When method get Then status 200 And match response == '#object'	Devuelve 200 El objeto devuelto cumple con el esquema definido

Ilustración 77 – Prueba de integración – Encontrar aerolínea mediante Iata

Escenario de prueba	Descripción	Resultado
Update <basePath>	Given path <basePath> And request read ('classpath:features/model_crud/put/valid/' + <expectedJsonName> + '-put-bg.json') And method post Then status 201 * def resource_id = get response.id Given request read('classpath:features/model_crud/put/valid/' + <expectedJsonName> + '-put.json') And path <basePath> + '/' + resource_id When method put * def expected = read('classpath:features/model_crud/get/models/' + <expectedJsonName> + '.json') Then status 200 And match response == expected	Devuelve 200 El objeto devuelto cumple con el esquema definido

Ilustración 78 – Prueba de integración – Actualización completa

Como se puede ver, en este tipo de pruebas, aunque sean en el mismo lenguaje que el de las pruebas de interfaz de usuario, no es igual de intuitivo de leer el código, por ello solo se mostraran estos ejemplos y el resto de test se pueden encontrar en el anexo H.



	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
features/common_resources/global-variables.feature	35	0	0	0	0	35	1	0	1	0.033	Passed
features/individual_endpoints/aircrafts/aircrafts-get.feature	152	0	0	0	0	152	16	0	16	1.769	Passed
features/individual_endpoints/airlines/airlines-get.feature	106	0	0	0	0	106	15	0	15	2.831	Passed
features/individual_endpoints/airports/airports-get.feature	116	0	0	0	0	116	16	0	16	1.773	Passed
features/individual_endpoints/file/file-upload.feature	72	0	0	0	0	72	9	0	9	1.239	Passed
features/model_crud/delete/delete-resource.feature	51	0	0	0	0	51	6	0	6	0.796	Passed
features/model_crud/get/get-all.feature	12	0	0	0	0	12	3	0	3	0.600	Passed
features/model_crud/get/get-model.feature	27	0	0	0	0	27	6	0	6	0.476	Passed
features/model_crud/post/post-resource.feature	72	0	0	0	0	72	12	0	12	1.694	Passed
features/model_crud/put/put-resource.feature	54	0	0	0	0	54	6	0	6	0.721	Passed
	697	0	0	0	0	697	90	0	90	11.536	10
	100.00%	0.00%	0.00%	0.00%	0.00%		100.00%	0.00%			100.00%

Ilustración 79 – Resultados de pruebas de integración

Más información sobre el proceso de desarrollo de estos test se puede encontrar en el anexo I.

6.4 Calidad del código

Como antes en esta memoria se ha comentado, gracias a la herramienta *Sonarqube,* que mediante ciertas reglas analiza el código en busca de fallos para poder solucionarlos durante el desarrollo del producto, y, asegurar que el día de mañana el producto tenga pocas posibilidades de fallo ante imprevistos. Esto, sumado a los test anteriores expuestos, se puede asegurar una calidad del código bastante elevada.

Una vez dado por terminado el desarrollo de la API y tras realizar la última subida al *cloud* de IKUSI, se ejecuta automáticamente el *Sonarqube* que IKUSI tiene implantado en su *cloud* y en la siguiente imagen aparece cual sería el resultado obtenido:

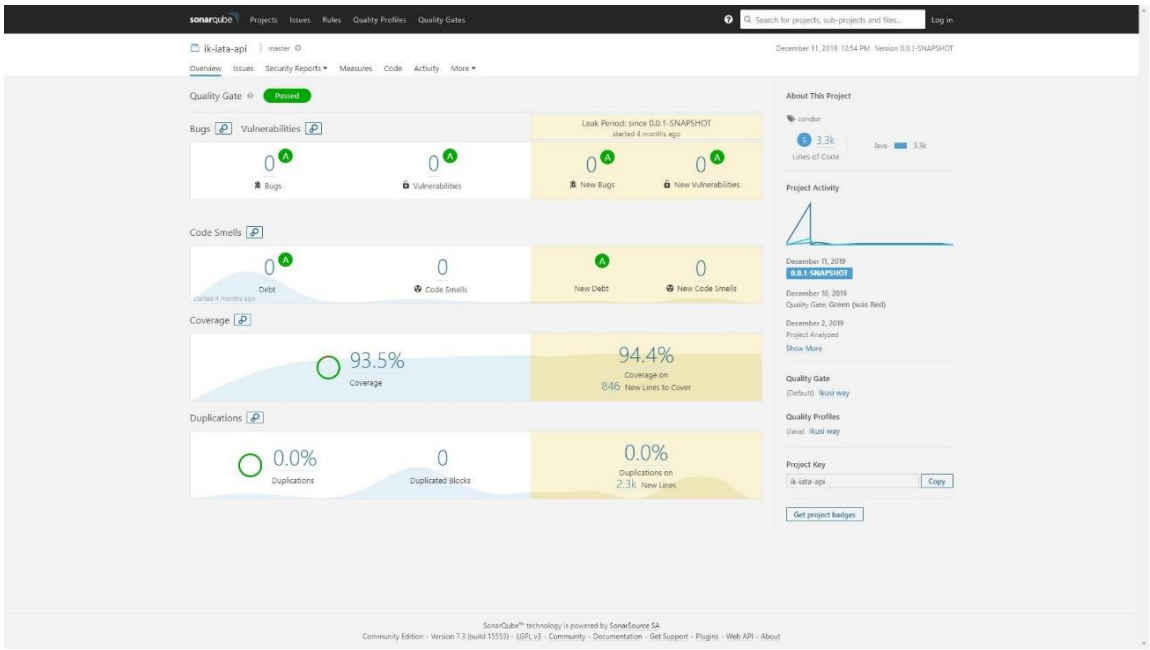


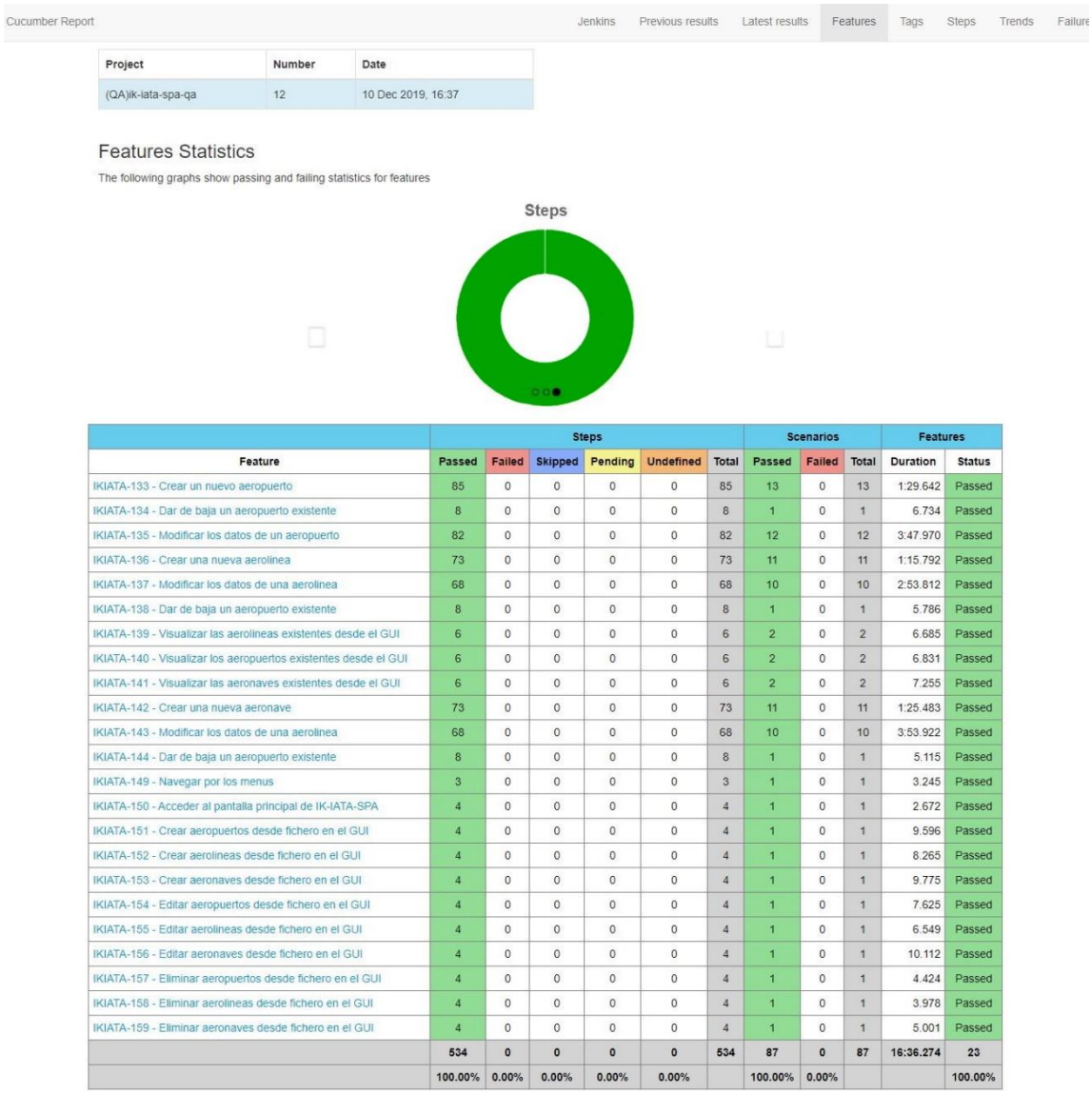
Ilustración 80 – Resultados SonarQube

La imagen de arriba es la página principal de *Sonarqube* en la cual podemos ver los resultados obtenidos del análisis de nuestro código. Se puede observar que, a nivel de calidad, la API desarrollada pasa todas las pruebas: el análisis no encuentra ningún *bug*, ninguna vulnerabilidad, ningún *code smells* (errores de código que indican posibles problemas más profundos, que no afecten técnicamente al programa, pero indican debilidades en el diseño, futuros fallos, ralentización,) y ningún bloque de código duplicado. También se puede ver cuánto porcentaje de código está cubierto mediante test unitarios.

En la parte más amarillenta se muestran los resultados obtenidos en el anterior análisis. En la izquierda, en el gráfico, también se puede ver cuál ha sido la evolución de los fallos en la aplicación desde que se realizó el primer análisis. En un principio ante el desconocimiento de esta herramienta, se generaron una gran cantidad de fallos que tras unos pocos análisis más, se empezó a programar de tal modo que se podía anticipar que tipos de fallos podía generar el código que se estaba implementando en ese momento.

El uso de esta herramienta tiene sus pros y contras: por una parte, sin tener unas bases de programación usar esta herramienta no es recomendable porque nos diría los fallos y no se aprendería, pero tras tener una base de programación, usar esta herramienta ayuda a mejorar en la calidad del código, que a día de hoy es muy importante.

Para la parte de interfaz, aunque esta herramienta también tenga ciertas reglas respecto a la interfaz (en nuestro caso sobre ReactJS), no funcionan de manera correcta ya que principalmente genera “falsos positivos” debido a que ReactJS juega con el estado de los componentes y sus ciclos de vida, y *Sonarqube* no llega a comprenderlo tan bien como comprende una implementación de Java. Aun así, *Cucumber* es capaz de generar unos informes con porcentajes de aciertos y fallos de los test de interfaz de usuario, y en caso de fallar, mostrar la última pantalla a la que llega el test antes de fallar. Estos son los resultados obtenidos de los test de interfaz de usuario realizados:



Cucumber Report | Jenkins | Previous results | Latest results | Features | Tags | Steps | Trends | Failure

Project	Number	Date
(QA)ik-iata-spa-qa	12	10 Dec 2019, 16:37

Features Statistics

The following graphs show passing and failing statistics for features

Steps

	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
IKIATA-133 - Crear un nuevo aeropuerto	85	0	0	0	0	85	13	0	13	1:29.642	Passed
IKIATA-134 - Dar de baja un aeropuerto existente	8	0	0	0	0	8	1	0	1	6.734	Passed
IKIATA-135 - Modificar los datos de un aeropuerto	82	0	0	0	0	82	12	0	12	3:47.970	Passed
IKIATA-136 - Crear una nueva aerolinea	73	0	0	0	0	73	11	0	11	1:15.792	Passed
IKIATA-137 - Modificar los datos de una aerolinea	68	0	0	0	0	68	10	0	10	2:53.812	Passed
IKIATA-138 - Dar de baja un aeropuerto existente	8	0	0	0	0	8	1	0	1	5.786	Passed
IKIATA-139 - Visualizar las aerolineas existentes desde el GUI	6	0	0	0	0	6	2	0	2	6.685	Passed
IKIATA-140 - Visualizar los aeropuertos existentes desde el GUI	6	0	0	0	0	6	2	0	2	6.831	Passed
IKIATA-141 - Visualizar las aeronaves existentes desde el GUI	6	0	0	0	0	6	2	0	2	7.255	Passed
IKIATA-142 - Crear una nueva aeronave	73	0	0	0	0	73	11	0	11	1:25.483	Passed
IKIATA-143 - Modificar los datos de una aerolinea	68	0	0	0	0	68	10	0	10	3:53.922	Passed
IKIATA-144 - Dar de baja un aeropuerto existente	8	0	0	0	0	8	1	0	1	5.115	Passed
IKIATA-149 - Navegar por los menus	3	0	0	0	0	3	1	0	1	3.245	Passed
IKIATA-150 - Acceder al pantalla principal de IK-IATA-SPA	4	0	0	0	0	4	1	0	1	2.672	Passed
IKIATA-151 - Crear aeropuertos desde fichero en el GUI	4	0	0	0	0	4	1	0	1	9.596	Passed
IKIATA-152 - Crear aerolineas desde fichero en el GUI	4	0	0	0	0	4	1	0	1	8.265	Passed
IKIATA-153 - Crear aeronaves desde fichero en el GUI	4	0	0	0	0	4	1	0	1	9.775	Passed
IKIATA-154 - Editar aeropuertos desde fichero en el GUI	4	0	0	0	0	4	1	0	1	7.625	Passed
IKIATA-155 - Editar aerolineas desde fichero en el GUI	4	0	0	0	0	4	1	0	1	6.549	Passed
IKIATA-156 - Editar aeronaves desde fichero en el GUI	4	0	0	0	0	4	1	0	1	10.112	Passed
IKIATA-157 - Eliminar aeropuertos desde fichero en el GUI	4	0	0	0	0	4	1	0	1	4.424	Passed
IKIATA-158 - Eliminar aerolineas desde fichero en el GUI	4	0	0	0	0	4	1	0	1	3.978	Passed
IKIATA-159 - Eliminar aeronaves desde fichero en el GUI	4	0	0	0	0	4	1	0	1	5.001	Passed
	534	0	0	0	0	534	87	0	87	16:36.274	23
	100.00%	0.00%	0.00%	0.00%	0.00%		100.00%	0.00%			100.00%

Ilustración 81 – Resultado de pruebas de interfaz de usuario

Estos test resultan necesarios ya que cuando se desarrolla una página, en principio se comprueba manualmente su correcto funcionamiento, pero lo que no se suele comprobar es si a la hora de crearla se ha modificado alguna cosa que genere error en otra página anteriormente diseñada, por ello, son necesarios estos test y así evitar realizar manualmente todas y cada una de las comprobaciones.

6.5 Calidad del producto

Una de las principales causas de la calidad elevada del código y la necesidad de la creación de este tipo de test (unitarios, interfaz de usuario e integración), se debe a que es un proyecto de *integración continua*. Con este término, nos referimos a que la implementación creada en el desarrollo, directamente se integra en el producto que se encuentra en producción, por ello se necesitan tantos test que corroboren y aseguren una calidad en la subida al producto.

Por otro lado, al principio del documento se definieron unos puntos clave para controlar la calidad del producto que se iba a desarrollar:

1. Procesamiento del servidor:
   - El tiempo de proceso del servidor para peticiones normales es menor a 1 segundo.
   - Para el procesado de aeropuertos considerados de nivel 2 y 3 (los niveles se asignan en función de la cantidad de tránsitos diarios, el aeropuerto de Hondarribi es de nivel 2 y el aeropuerto de Madrid es de nivel 3), se estiman unos 3.000 aeropuertos en el mundo, y para esta petición tarda aproximadamente 1-2 segundos.
   - Para el procesado de todos los aeropuertos de todo el mundo (se incluye cualquier pista pequeña de aterrizaje), se calcula que rondan los 70.000, y para este procesa si se tarda bastante, unos 5 minutos, pero como esta petición se realizara una vez como mucho en la vida de la aplicación, se considera aceptable.
2. Fiabilidad:
   - Respecto a la fiabilidad, siempre pueden influir factores ajenos al proyecto, así como, la conexión a internet, rendimiento del pc… son factores que depende en que situación pueden inferir y causar un fallo, pero principalmente se puede asegurar que cada uno de los componentes desarrollados funcionan a la perfección y hasta el momento, en el 100% de los casos. Esto se puede asegurar gracias a todos los test y análisis de código realizados.
3. Claridad del código:
   - Aunque este apartado sea cuestión de gustos, hay ciertos puntos clave que se pueden destacar con los cuales se intuye que puede llegar a ser un código claro y legible: Se han usado nombres lógicos para la definición de los métodos en función de lo que realizaban; las variables tienen un nombre sencillo de interpretar; y en caso de ser alguna parte un poco enrevesada, se han hecho comentarios que ayuden a la comprensión. Por otra parte, varios informáticos expertos han observado el código y les ha parecido de comprensión sencilla.
4. Diseño de interfaz:
   - Para parte de la interfaz, se ha realizado una prueba que valora el tiempo medio entre un usuario *con conocimientos* de informática y un usuario *sin conocimiento* de informática y se compara con el tiempo realizado por el experto, en este caso el desarrollador. El anexo H recoge las pruebas realizadas
5. Calidad de la memoria:
   - Este aspecto es muy variable, principalmente depende de la persona que lo lea: si le resulta interesante, si es por obligación o no… A nivel técnico se ha seguido la estructura diseñada, aunque se han añadido algunos puntos. A nivel personal me parece una memoria de proyecto bastante sencilla de comprender, aunque a lo mejor un poco más extensa de lo deseado.

Tras estudiar los puntos clave definidos al principio del proyecto, se puede decir que cumple con los objetivos de calidad propuestos.

Capítulo 7

Seguimiento y control

Como en todo proyecto de estas dimensiones suele suceder, ocurren ciertos desvíos de la planificación inicial, y en ocasiones desviaciones respecto a las tares definidas. Para ello, en este apartado vamos a analizar ciertos puntos clave que nos dirán el resultado de nuestro proyecto como ha sido y si hemos realizado una correcta planificación.

7.1 Control de alcance

Al inicio de esta memoria, en el capítulo 2, definimos una cierta cantidad de tareas a realizar. El objetivo inicial de este proyecto es crear un microservicio REST (API) común para las aplicaciones de IKUSI AMS, con su propia interfaz web para el mantenimiento y configuración, superando unos niveles elevados de calidad sin sobrepasar lo que es el proyecto final de la Universidad que consume 12 créditos, es decir, unas 300 horas de trabajo.

Viendo esta planificación inicial definida, se puede decir que el microservicio está creado, funciona y supera los niveles de calidad exigidos por la empresa y, por otra parte, la interfaz web cumple con las funcionalidades de mantenimiento y configuración de la API cumpliendo los requisitos de estilo.

Aunque en un principio no se estimó ni se tuvo en cuenta en el proyecto, existen 2 tareas que no se definieron:

Por un lado, ya que va a ser un proyecto que se incluirá en las actuales y futuras implantaciones aeroportuarias que la empresa IKUSI maneja, por lo tanto, para poder distribuir este producto también se ha generado un Docker.

Por otro lado, para poder ser microservicio común que envié notificaciones al resto de aplicaciones conectadas a la API, se ha desarrollado mediante *Kafka* este conector.

7.2 Control de planificación

En la siguiente tabla aparece cual es el desglose completo del tiempo invertido en horas de cada tarea planificada, pudiendo ver la diferencia entre lo estimado en un inicio del proyecto y el tiempo real. También podremos ver la desviación obtenida y el porcentaje equivalente; se han marcado como porcentaje aceptable un máximo del ±25%. Se marcan de color amarillo los valores aceptables, en rojo las desviaciones negativas (estimación demasiado positiva) y en verde las desviaciones positivas (trabajo realizado en menos horas de lo esperado).

Tareas	Estimación en horas	Tiempo real en horas	Desviación en horas	Porcentaje
P - Planificación	9	11	-2	22,2%
P.1 - Presentación proyecto	2	2	-2	0,0%
P.2 - Planificación	5	6	-1	20,0%
P.3 - Revisión planificación	2	0	2	-100,0%
EH - Estudio de herramientas	19	18	1	-5,3%
EH.1 - Estudio herramientas organización	1	1	0	0,0%
EH.2.1 - Estudio Rest	2	1	1	-50,0%
EH.2.2 - Estudio React	5	10	-5	100,0%
EH.2.3 - Estudio Junit	1	0	1	-100,0%
EH.2.4 - Estudio Cucumber y Selenium	5	3	2	-40,0%
EH.2.5 - Estudio Karate	5	3	2	-40,0%
CC - Control y continuidad	38	24,5	13,5	-35,5%
CC.1 - Revisión trabajo	4	2	2	-50,0%
CC.2 - Comparación trabajo	4	2,5	1,5	-37,5%
CC.3 - Reunión con el tutor	10	6	4	-40,0%
CC.4 - Spring planning	20	14	6	-30,0%
ARS - Análisis de requisitos	10	7	3	-30,0%
EMD - Estudio de datos maestros	2	1	1	-50,0%
DA- Desarrollo API	40	44	-4	10,0%
DA.1 - Preparar entorno desarrollo API	2	1,5	0,5	-25,0%
DA.2 - Entorno GitLab	2	0	2	-100,0%
DA.3 - Preparar entorno Jenkins	2	0	2	-100,0%
DA.4 - Crear Base de Datos	4	1,5	2,5	-62,5%
DA.5 - Desarrollo de la API	30	41	-11	36,7%
GTU - Generar test unitarios	34	50	-16	47,1%
GTU.1 - Preparar entorno desarrollo test	2	4	-2	100,0%
GTU.2 - Generar Docker SonarQube local	1	0,5	0,5	-50,0%
GTU.3 - Preparar SonarQube en servidor	1	0,5	0,5	-50,0%
GTU.4 - Desarrollo de test unitarios	30	45	-15	50,0%
EI - Estudio del estilo IK-WI-4	2	1	1	-50,0%
DI - Desarrollo interfaz	34	42	-8	23,5%

DI.1 - Preparar entorno desarrollo web	2	1	1	-50,0%
DI.2 - Preparar entorno Gitlab	1	0	1	-100,0%
DI.3 - Preparar entorno Jenkins	1	0	1	-100,0%
DI.4 - Desarrollo de la web	30	41	-11	36,7%
GTIU - Generar test de interfaz de usuario	28	20,5	7,5	-26,8%
GTIU.1 - Entorno para test de interfaz	1	1,5	-0,5	50,0%
GTIU.2 - Generar Docker SonarQube local	1	0	1	-100,0%
GTIU.3 - Preparar SonarQube en servidor	1	0	1	-100,0%
GTIU.4 - Desarrollo de test de interfaz	25	19	6	-24,0%
GTI - Generar test de integración	24	16	8	-33,3%
GTI.1 - Entorno para test automatizados	2	1	1	-50,0%
GTI.2 - Generar Docker SonarQube local	1	0	1	-100,0%
GTI.3 - Preparar SonarQube en servidor	1	0	1	-100,0%
GTI.4 - Desarrollo de test integración	20	15	5	-25,0%
DM - Documentación de manuales	10	17	-7	70,0%
DM.1 - Realizar manual de instalación	5	8	-3	60,0%
DM.2 - Realizar manual de configuración	3	4	-1	33,3%
DM.3 - Realizar manual de usuario	2	5	-3	150,0%
M - Memoria del proyecto	50	64	-14	28,0%
K - Desarrollo de Kafka	0	10	-10	-----
D - Creación de Docker	0	8	-8	-----
TOTAL	300	334	-34	11,3%

Tabla 6 - Tiempo real y deviaciones

7.3 Estimación y desviación de las tareas

Planificación

En el paquete de trabajo de la planificación, todo ha ido según lo planeado, salvo que no ha hecho falta una revisión de planificación ya que gracias a la metodología utilizada en la empresa las revisiones se realizaban en los *spring-planning* y los *reviews*.

En general ha estado bien estimada.

Estudio de herramientas

El estudio de las herramientas ha sido más sencillo de lo esperado salvo por la herramienta ReactJS, en este caso han hecho falta un par de tutoriales.

En general ha estado bien estimada.

Control y continuidad

Gracias a la metodología de trabajo (SCRUM) de la empresa IKUSI, la realización de este paquete ha sido más sencilla y rápida de lo estimado.

En general se ha estimado más tiempo del debido.

Análisis de requisitos

El análisis se ha realizado más o menos en el tiempo esperado.

Estudio de datos maestros

El estudio de datos maestros se ha realizado en el tiempo estimado.

Desarrollo API

Aunque haya habido ciertos problemas en este paquete, se ha respetado la estimación inicial bastante bien.

Generar test unitarios

Al igual que en el paquete de arriba, también han existido ciertos problemas que en el siguiente punto detallaremos, y tras un desconocimiento inicial de la importancia que tenían los test se ha estimado bastante mal este aspecto.

Estudio del estilo IK-WI-4

El estudio del estilo propio de la empresa para el interfaz de la aplicación ha sido bastante sencillo ya que existía documentación, y tras leerla quedaba bastante claro. En general ha ido mejor de lo estimado, aunque a nivel de horas no haya diferencia, se ha realizado en un 50% menos de lo estimado, por lo cual, está mal estimado.

Desarrollo interfaz

En el asunto de la interfaz también han existido problemas, los mismos que en el desarrollo de la API y test unitarios. En este caso hay que tener en cuenta que ReactJS es mucho más inmenso que simple JavaScript, por lo tanto, la estimación se queda corta.

Generar test de interfaz de usuario

Al igual que el resto de paquetes de desarrollo, ha habido problemas, pero, aun así, se ha conseguido estar dentro del tiempo estimado.

Generar test de integración

Nuevamente, ha habido problemas, pero, aun así, se ha conseguido realizar dentro del tiempo estimado.

Documentación de manuales

La estimación de este paquete se ha realizado mal, no se han tenido en cuenta ciertos aspectos que afectaban al desarrollo.

Memoria

Ha existido una pequeña variación entre lo estimado y el tiempo real invertido, se escapa por poco del porcentaje aceptable, por lo tanto, aunque se considere que este mal estimado, no esta tan mal.

Desarrollo de Kafka

Es un paquete de trabajo que no estaba incluido.

Creación de Docker

Es un paquete de trabajo que no estaba incluido.

7.4 Razonas más significativas de las desviaciones

Las razones más significativas de las desviaciones arriba expuestas, alguna se debe a riesgos que ya estaban asumidos que podían pasar y con lo cual, aunque se ha creado una pequeña variación en lo planificado, se ha gestionado bien. Este es el caso de la falta de permisos para poder crear proyectos en GitLab y Jenkins, que la solución era que el Scrum Master o el QA tester (*Quality*

Assurance, Asegurador de Calidad), que eran las únicas que tenían derecho a crear proyectos y posteriormente dar los permisos para poder editarlos.

Por otro lado, también se intuía el riesgo de posibles cambios en las tareas a realizar, y aunque no ha sido un gran problema, el gestionarlo ha supuesto un total de 18 horas extras que no entraban en la planificación inicial.

Otras razones que no se valoraron en un principio fueron que el estudio de una de las herramientas fuese más complicado de lo esperado, puesto que en un principio a la hora de recibir la presentación del proyecto no se mostraba como tal. Y, otra razón para el posible desarrollo algo lento del proyecto, es cuestión de que en ciertos aspectos cuando se necesitaba un rendimiento un poco elevado, el ordenador portátil con el que se desarrolló el proyecto se quedaba corto de rendimiento y se bloqueaba demasiadas veces.

7.5 Diagrama de Gantt

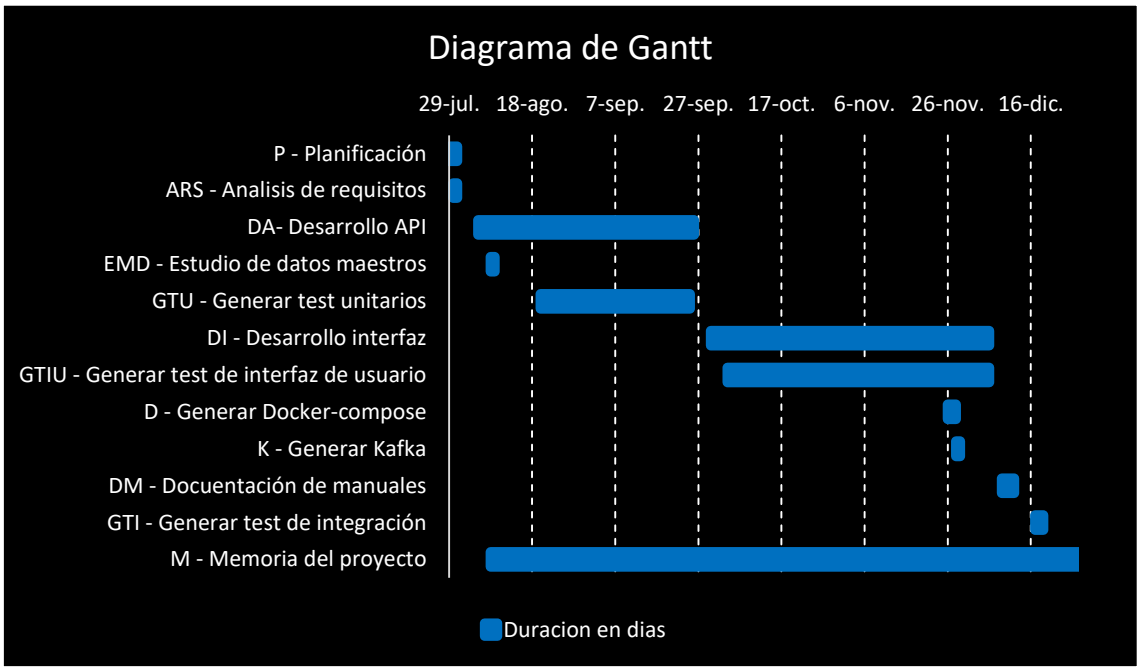


Ilustración 82 – Diagrama de Gantt

En el diagrama de Gantt podemos ver cuál ha sido el desarrollo del proyecto por paquetes de trabajo. En este gráfico, introducir los paquetes de trabajo **Estudio de herramientas**, **Estudio de IK-WI-4** y **Control y continuidad**, no tendría mucha lógica ya que son tareas que se desarrollan en momentos puntuales del proyecto que realmente sirven como complementos. Aparte tampoco generarían ningún entregable.

7.6 Control límites de entrega

A sabiendas que iba a ser un proyecto que se iba a desarrollar a la vez que se trabajaba en la empresa, se establecieron grandes márgenes para el desarrollo de las tareas y así poder cumplir con los plazos establecidos, a fin de cuentas, aunque el proyecto se ha desarrollado para una empresa y dentro de ella, no se ha trabajado 100% solo en el desarrollo del proyecto. Por ello, vamos a analizar los límites de entrega establecidos en el inicio de la planificación, y los vamos a comparar con las fechas de entrega realizadas:

Tareas	Límites de entrega	Fechas de entrega	¿Entregada?
P - Planificación	06/08/2019	30/07/2019	Si
ARS - Análisis de requisitos	09/08/2019	30/07/2019	Si
EMD - Estudio de datos maestros	09/08/2019	08/08/2019	Si
DA- Desarrollo API	13/08/2019	25/09/2019	No
GTU - Generar test unitarios	13/08/2019	24/09/2019	No
DI - Desarrollo interfaz	06/12/2019	05/12/2019	Si
GTIU - Generar test de interfaz de usuario	13/12/2019	05/12/2019	Si
GTI - Generar test de integración	10/01/2019	18/11/2019	Si
DM - Documentación de manuales	17/01/2019	11/12/2019	Si
M - Memoria del proyecto	20/01/2019	15/01/2020	Si

Tabla 7 - Control límites de entrega

Como se aprecia en la tabla, salvo el desarrollo de la API y la generación de test unitarios, el resto si cumplen con el plazo establecido. Este fallo es debido a que se producen ciertos problemas ajenos al proyecto que retrasan el desarrollo planificado. Este aspecto de retraso se podía prever, por ello, se establecieron márgenes amplios, aunque para este caso se falló con la fecha planificada.

En general, se terminó el proyecto antes de lo estimado.

Capítulo 8

Conclusiones y futuras mejoras

En este apartado vamos a mostrar las conclusiones a las que hemos llegado tras completar el desarrollo de este proyecto y las competencias obtenidas.

8.1 Conclusiones

8.1.1 Conclusiones a nivel técnico

A nivel técnico se puede decir que se ha desarrollado un proyecto muy completo, cubre el desarrollo de Base de Datos, Back end (API), Front end (Interfaz web), integración con otros proyectos (IK-SS), conector con la API para las aplicaciones suscritas (Kafka) y la posibilidad de levantar el conjunto del proyecto en cualquier sistema (Docker).

Es un proyecto en el cual predomina la calidad a la funcionalidad, es mejor que tenga una funcionalidad sencilla pero de calidad muy elevada, a muchas funcionalidad con calidad regular y que pueda fallar en cualquier momento. Hay que tener en cuenta que aplicaciones en tiempo real la usaran para la gestion de aeroportuaria.

8.1.2 Conclusiones a nivel personal

Este tipo de proyecto es la primera vez que desarrollaba con herramientas que ni conocía, por lo cual, la conclusión más importante es que se ha aprendido mucho sobre bastantes herramientas y sobre la calidad que se exige sobre un proyecto. En mi opinión, un proyecto de esta estructura, ayuda a poner en práctica las habilidades desarrolladas durante los años de universidad, pero, sobre todo, el saber desenvolverse antes problemas reales y generar un producto completamente funcional.

8.2 Posible mejoras

Como futura mejora, se añadirán tablas a la Base de Datos que relacionen entre si las tablas, para la siguiente versión del producto se crearan más tablas, entre ellas las tablas que gestionen el código ISO del país, tabla que controle la zona horaria según su código ISO y otra que controle el valor *dst*. El añadir estas tablas creara una nueva complejidad en la lógica de negocio, y para estas también abra que añadir nuevos *endpoints* y una parte en la interfaz que los gestione.

En la parte del conector de Kafka, se podría crear una división entre los mensajes enviados en temas, para que así los suscriptores se suscriban al tema que deseen.

En la parte del Front end, siguiendo el hilo de la mejora de Kafka, se podría añadir un acceso para los suscriptores de la API, para que puedan gestionar ciertos aspectos tales como los temas a los que se suscriben.

8.3 Competencias adquiridas

Gracias a la realización de este proyecto, el haberlo realizarlo para una empresa y dentro de un grupo de trabajo, se han obtenido ciertas competencias:

- Capacidad para afrontar problemas reales.
- Mejora de trabajo en equipo.
- Mejora de la gestión del tiempo y planificación.
- Capacidad para adaptarse a nuevas situaciones y tomar decisiones.
- Capacidad de aprendizaje autodidacta.
- Capacidad de presentar tanto de forma oral y escrita, el material y la argumentación técnica.

A) Actas de reunión con el profesor

Acta Reunión de Control y Seguimiento

Numero de acta: 1

Duración: 50 minutos

Fecha y hora: 10/09/2019 – 12:30

Lugar: Despacho del tutor

Asistentes:

- Ricardo Herradura Collazos
- Jon Iturrioz Sánchez

Temas tratados:

- Orientación Trabajo de Fin de Grado.
- Establecer comunicación alumno-tutor.

Decisiones tomas:

- Buscar cómo mejorar el proyecto.

Compromisos para la siguiente reunión:

- Realizar casos de uso.
- Definir modelo de datos.
- Realizar modelo de Base de Datos.

Próxima reunión:

17/09/2019 – 15:30

Acta Reunión de Control y Seguimiento

Numero de acta: 2 **Duración:** 50 minutos

Fecha y hora: 17/09/2019 – 15:30 **Lugar:** Despacho del tutor

Asistentes:

- Ricardo Herradura Collazos
- Jon Iturrioz Sánchez

Temas tratados:

- Analizar proyecto y ver cómo mejorarlo.
- Revisión de documentación hasta el momento.

Decisiones tomas:

- Enfocar proyecto en tema de calidad.
- Reestructurar la documentación de la memoria.
- Detallar mejor varios puntos.

Compromisos para la siguiente reunión:

- Realizar decisiones tomadas.
- Rellenar punto análisis de requisitos.
- Mejorar vocabulario de esquema Estructura de Descomposición de Trabajo.
- Continuar con documentación.

Próxima reunión:

16/10/2019 – 15:00

Acta Reunión de Control y Seguimiento

Numero de acta: 3

Duración: 1 hora y 20 minutos

Fecha y hora: 16/10/2019 – 15:00

Lugar: Despacho del tutor

Asistentes:

- Ricardo Herradura Collazos
- Jon Iturrioz Sánchez

Temas tratados:

- Revisión de los cambios acordados.
- Orientación respecto al cambio de enfoque de proyecto.

Decisiones tomas:

- Mejorar varios aspectos: índice, EDT y casos de uso.

Compromisos para la siguiente reunión:

- Realizar cambios acorados.
- Continuar con la documentación.

Próxima reunión:

24/10/2019 – 15:00

Acta Reunión de Control y Seguimiento

Numero de acta: 4 **Duración:** 1 hora y 30 minutos

Fecha y hora: 24/10/2019 – 15:00 **Lugar:** Despacho del tutor

Asistentes:

- Ricardo Herradura Collazos
- Jon Iturrioz Sánchez

Temas tratados:

- Valoración sobre presentar en la primera convocatoria (noviembre).
- Revisar documentación para valorar.

Decisiones tomas:

- Mejor esperar a la siguiente convocatoria (febrero).
- Realizar de manera más detallada y directa la documentación.

Compromisos para la siguiente reunión:

- Realizar diseños de casos de test.
- Continuar con la documentación.

Próxima reunión:

08/01/2020 – 12:30

Acta Reunión de Control y Seguimiento

Numero de acta: 5

Duración: 1 hora

Fecha y hora: 08/01/2020 – 12:30

Lugar: Despacho del tutor

Asistentes:

- Ricardo Herradura Collazos
- Jon Iturrioz Sánchez

Temas tratados:

- Repaso de la documentación casi finalizada.

Decisiones tomas:

- Restructurar índice de documentación.
- Realizar pequeños cambios de expresión.

Compromisos para la siguiente reunión:

- Realizar cambios acorados.
- Continuar con la documentación para dejarla ya casi terminada por completo.

Próxima reunión:

13/01/2020 – 12:30

Acta Reunión de Control y Seguimiento

Numero de acta: 6 **Duración:** 30 minutos

Fecha y hora: 13/01/2020 – 12:30 **Lugar:** Despacho del tutor

Asistentes:

- Ricardo Herradura Collazos
- Jon Iturrioz Sánchez

Temas tratados:

- Revisar últimos detalles de la documentación.

Decisiones tomas:

- Adaptar en lo posible documentación a la normativa 2019-2020.
- Dado visto bueno a la documentación para presentar en febrero.

Compromisos a realizar:

- Realizar adaptación de la documentación a la normativa 2019-2020.
- Realizar pequeños cambios de expresión.
- Explicar un poco el código introducido.

Próxima reunión:

No va a haber más.

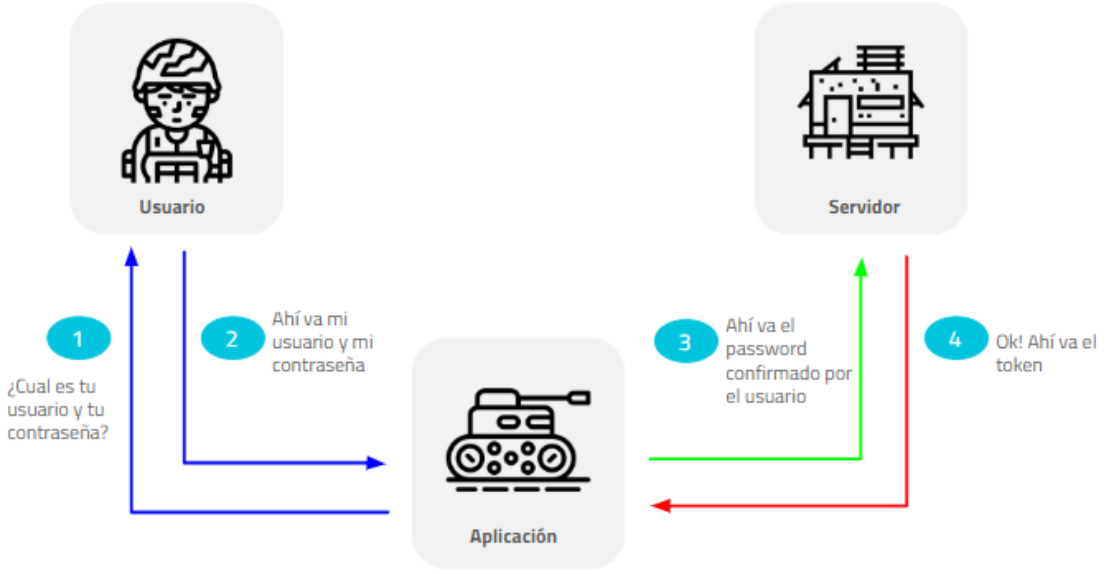
B) IK-SS

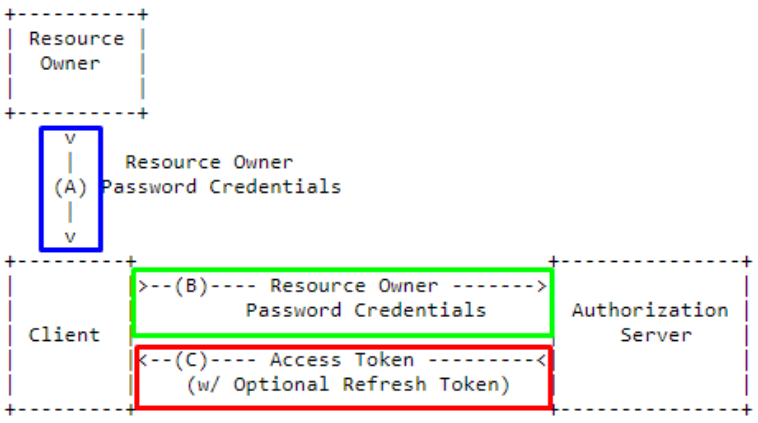
IK-SS es un componente de IKUSI para la verificación y autorización de usuarios basado en el protocolo *oauth2*. Se trata de un protocolo que permite el acceso autorizado a una API.

El flujo de este protocolo seria el siguiente:

1. El cliente solicita una autorización para acceder a la aplicación.
2. La aplicación envía dicha solicitud a IK-SS.
3. IK-SS comprueba la existencia de dicho usuario y permisos, y genera un *token* en caso de estar autorizado.
4. En caso positivo la aplicación le devuelve el *token* al cliente, gracias al cual tendrá acceso por las diferentes partes de la aplicación.

Gracias a este componente, en vez de que la aplicación tenga realizar una solicitud de existencia del cliente, simplemente gracias al *token* que genera IK-SS, la aplicación comprueba que ese *token* ha sido generado por IK-SS y le permite el cliente navegar por las diferentes zonas de la aplicación.





C) Manuales

A continuación, se van a mostrar los manuales desarrollados para la correcta instalación, configuración y uso del servicio web.

Para empezar, mostraremos los manuales referidos a IK-IATA-API, posteriormente, los manuales referidos a IK-IATA-SPA, para ir finalizando los manuales de instalación del servicio web mediante Docker en Linux y Windows; y, para finalizar, el manual de usuario de IK-IATA:

IK-IATA-API

Manual de Instalación

Ikusi – Ángel Iglesias
Paseo Miramón, 170
2009 San Sebastián
SPAIN
Tel: +34 943 44 88 00
Fax: +34 943 44 88 20
ikusi@ikusi.com
www.ikusi.com

CONTROL DE VERSIONES

Versión	Fecha	Responsable	Comentarios
1.0	11/12/2019	Ricardo Herradura	Versión inicial

INDICE

1. Introducción

El presente documento describe los pasos a seguir para realizar la instalación de la aplicación **IK-IATA-API**, la cual se encarga de la gestión y manejo de las tablas maestras del sistema Iata/Icao.

2. Requisitos

Al ser una aplicación auto contenida, es decir, solo hace falta lo siguiente para su instalación:

- ✓ Sistema operativo Linux (RHEL / CentOS 6 o superior)
- ✓ Java JDK versión 1.8.0 (32/64 bits) (ver manual instalación Java JDK 8)
- ✓ Base de datos PostgreSQL versión 9.X (ver manual instalación PostgreSQL 9)
- ✓ Configuración base de datos (ver manual configuración PostgresSQL – IK-IATA-API)

3. Instalación

- Realizar el proceso de instalación como usuario **root**
- Descargar la última versión del paquete de la aplicación del repositorio: ***ik-iata-api-dist.tar.gz***
- Copiar el paquete en el directorio de instalación: **/opt/ikusi**

```
# cp ik-iata-api-dist.tar.gz /opt/ikusi
```

- Descomprimir el fichero **ik-iata-api-dist.tar.gz** y borrar el mismo:

```
# cd /opt/ikusi
# tar xvf ik-iata-api-dist.tar.gz
# rm -f ik-iata-api-dist.tar.gz
```

- Renombrar el directorio a **/opt/ikusi/ik-iata-api**:

```
# mv /opt/ikusi/ik-iata-api-dist /opt/ikusi/ik-iata-api
```

- Editar el fichero ***application.properties*** ubicado en el directorio **/opt/ikusi/ik-iata-api/** siguiendo el manual de configuración: IK-IATA - Manual de Configuración - IK-IATA-API
- Una vez seguidos los pasos anteriores, para lanzar la aplicación hay que ejecutar la siguiente instrucción desde la ruta donde se encuentra el .jar y las properties:

```
# /opt/java_8/bin/java -jar ik-iata-api.jar &.
```

4. Servicio ik-iata-api

En este punto se detalla la forma de arranque/parada de ik-iata-api para CentOS 7

4.1 Systemd unit file sobre CentOS 7 y RHEL 7

Para habilitar la aplicación como un servicio, crear el fichero **/usr/lib/systemd/system/ik-iata-api.service**, como usuario root, configurando la ruta donde está instalada la aplicación (`WorkingDirectory`), así como la ruta de la instalación de java (primera parte del `ExecStart`) y el binario de la aplicación (segunda parte del `ExecStart`)

```
[Unit]
Description=Ik Iata API - Spring Boot application

[Service]
ExecStart=/opt/java_8/bin/java -jar /opt/ikusi/ik-iata-api/ik-iata-
api.jar
WorkingDirectory=/opt/ikusi/ik-iata-api/
SuccessExitStatus=143
User=root
Group=root

[Install]
WantedBy=multi-user.target
```

Para cargar el fichero unit, es necesario recargar systemd como usuario root mediante el siguiente comando:

```
# systemctl daemon-reload
```

4.1.1 Habilitar arranque en el inicio del Sistema

Para habilitar el servicio y que arranque con el inicio del sistema, ejecutar:

```
# systemctl enable ik-iata-api
```

4.2 Script de arranque con init.d para CentOS y RHEL 6 versiones inferiores

Para habilitar la aplicación como un servicio, se crea el fichero /etc/init.d/ik-iata-api (se adjunta en la instalación), configurando la ruta donde está instalada la aplicación (`/opt/ikusi/ik-iata-api`), el nombre del jar (`IK_IATA_API_BIN`), así como la ruta de la instalación de java 8 (`JAVA_HOME`).

```
#! /bin/bash
# chkconfig: 2345 98 2
# description: ik-iata-api start or stop
#
# ik-iata-api application start or stop
#
#
# processname: IK-IATA-API
# pidfile: /var/run/ik-iata-api.pid

# Source function library
. /etc/init.d/functions

RETVAL=0
USER_OWNER="root"

prog="ik-iata-api"
TIMEOUT=20
ERROR=256

#Variables de entorno para ik-iata-api
export JAVA_HOME=/opt/java_8
export PATH=$JAVA_HOME/bin:$PATH:$HOME/bin

export IK_IATA_API_BIN="ik-iata-api"
export IK_IATA_API_PATH="/opt/ikusi/ik-iata-api"
export IK_IATA_API_LOG_TMP="ik-iata-api.log.tmp"

checkstatus() {

RETVAL=256

RETVAL1=`ps -fea| grep java | grep $ IK_IATA_API_BIN| wc -l`
[ $RETVAL1 -eq 1 ] && echo "$ IK_IATA_API_BIN is running" || echo
"$IK_IATA_API_BIN stopped"

[ $RETVAL1 -eq 1 ]
RETVAL=$?

return $RETVAL
}


start() {
echo -n $"Starting $prog: "

retorno=`checkstatus | grep running | wc -l`
```

```
if [ "$retorno" != "0" ]; then
   echo "ERROR: $IK_IATA_API_BIN is launched"
   return $ERROR
fi

echo "Starting ik-iata-api"
# Vaciar fichero de log
echo 0 > $IK_IATA_API_PATH/$IK_IATA_API_LOG_TMP

cd $IK_IATA_API_PATH
java -jar $IK_IATA_API_PATH/$IK_IATA_API_BIN.jar 2>&1 >
$IK_IATA_API_PATH/$IK_IATA_API_LOG_TMP &

x=0
retorno=1
until [ $x -eq $TIMEOUT -o  $retorno -eq 0 ]; do
   cat $IK_IATA_API_PATH/$IK_IATA_API_LOG_TMP | grep "Started
IkIataApiApplication" >> /dev/null
   retorno=$?
   x=$((x+1))
   sleep 1
   echo -n .
done
if [ $x -eq 30 ]
then
   echo "ERROR: $IK-IATA_API_BIN not started"
else
   echo "$IK_IATA_API_BIN started"
   RETVAL1=0
fi


echo
[ $RETVAL1 -eq 0 ]&&touch /var/lock/subsys/ik-iata-api
RETVAL=$?

return $RETVAL
}

stop() {

RETVAL1=1

PID1=`ps -fea| grep java | grep $IK_IATA_API_BIN | awk '{ print $2 }'`
if [ "$PID1" != "" ];then
   kill -9 $PID1 >> /dev/null
   RETVAL1=$?
   echo -n $"Stopped $IK_IATA_API_BIN"
fi
```

```
[ $RETVAL1 -eq 0 ]&&rm -f /var/lock/subsys/ik-iata-api
RETVAL=$?

return $RETVAL
}

restart() {
stop
sleep 2
start
}

case "$1" in
   start)
      start
      ;;

   stop)
      stop
      ;;

   status)
      checkstatus
      ;;

   restart)
      restart
      ;;

   *)
      echo $"Usage: $0 {start|stop|restart|status}"
      exit 1
      ;;
esac

exit $?
```

4.2.1 Habilitar arranque en el inicio del Sistema

Ejecutar el siguiente comando para que la aplicación arranque en el inicio del sistema operativo:

```
# chkconfig ik-iata-api on
```

5. Arranque y para del servicio ik-iata-api

A continuación, se especifica los pasos para arrancar y parar la aplicación.

5.1 Arrancar

Ik-iata-api puede arrancarse ejecutando el siguiente comando como root:

- CentOS 7

```
# systemctl start ik-iata-api
```

- CentOS 6

```
# service ik-iata-api start
```

5.2 Visualizar el estado

Para visualizar el estado de la aplicación, ejecutar

- CentOS 7

```
# systemctl status ik-iata-api
```

- CentOS 6

```
# service ik-iata-api status
```

5.3 Parar

Para parar ik-iata-api, se ejecuta el siguiente comando como root:

- CentOS 7

```
# systemctl stop ik-iata-api
```

- CentOS 6

```
# service ik-iata -api stop
```

6. Firewall

6.1.1 Reglas del Firewall para postgresql - CentOS 7

Para el acceso desde el exterior a ik-iata-api, el puerto 8855 deberá estar abierto

```
# firewall-cmd --permanent --zone=public --add-port=8855/tcp
Success
```

Recargar la configuración del firewall:

```
# firewall-cmd --reload
success
```

6.1.2 Reglas del Firewall para postgresql - CentOS 6

Para el acceso desde el exterior a ik-iata-api, el puerto 8855 deberá estar abierto

```
# iptables -I INPUT 1 -m tcp -p tcp --dport 8855 -j ACCEPT
# service iptables save
```

Recargar la configuración del firewall:

```
# service iptables restart
```

PostgreSQL – IK-IATA-API

Manual de configuración de PostgreSQL – IK-IATA-API

Ikusi SLU - Velatia
Paseo Miramón, 170
20014 San Sebastián
SPAIN
Tel: +34 943 44 88 00
Fax: +34 943 44 88 20
ikusi@ikusi.com
www.ikusi.com

CONTROL DE VERSIONES

Versión	Fecha	Responsable	Comentarios
1.0	11/12/2019	Ricardo Herradura	Versión Inicial

INDICE

1. INTRODUCCIÓN

Este documento explica los pasos a seguir para configurar la base de datos de IK-IATA-API en postgreSQL.

2. REQUISITOS

- ✓ Base de datos PostgreSQL versión 9.X (ver manual instalación PostgreSQL 9)

3. CONFIGURACIÓN

3.1 Creación de la base de datos “ikiatadb”

Loguearse con el usuario `postgres`:

```
# su - postgres
```

Crear la base de datos “ikiatadb”:

```
$ psql
postgres=# create database ikiatadb;
postgres=# \q
```

3.2 Crear usuario para la base de datos “ikiatadb”

Loguearse con el usuario `postgres`:

```
# su - postgres
```

Crear usuario de la base de datos ‘**ikiata** con la contraseña:

```
$ psql ikiatadb
postgres=# create user ikiata with password 'ikiata';
postgres=# \q
```

Cambiar el dueño de la base de datos ‘**ikiatadb**’ para que sea el usuario ‘**ikiata**’

```
$ psql
postgres=# ALTER DATABASE ikiatadb OWNER TO ikiata;
postgres=# \q
```

3.3 Crear schema “ikiata”

Loguearse como usuario `postgres`:

```
# su - postgres
```

Crear el schema **ikiata** cuyo dueño es el usuario de la base de datos **ikiatadb**:

```
$ psql
postgres=# CREATE SCHEMA ikiata AUTHORIZATION ikiata;
```

IK-IATA-API

Manual de configuración

Ikusi – Ángel Iglesias
Paseo Miramón, 170
20014 San Sebastián
SPAIN
Tel: +34 943 44 88 00
Fax: +34 943 44 88 20
ikusi@ikusi.com
www.ikusi.com

CONTROL DE VERSIONES

Versión	Fecha	Responsable	Comentarios
1.0	11/12/2019	Ricardo Herradura	Versión Inicial

INDICE

1. Introducción

El presente documento describe la configuración de la aplicación **ik-iata-api**. La configuración se realiza mediante la edición del fichero "***application.properties"*** que se encuentran en el directorio /**opt**/**ikusi**/ **ik-iata -api**/

Información necesaria para la configuración:

- **IKIATADB_IP**: dirección IP de la máquina virtual en la que está instalada la base de datos de IK-IATA-API.

2. Archivos de configuración de la aplicación

2.1 Configuración general de ik-iata-api

Los parámetros generales de la aplicación se configuran editando las siguientes propiedades en el fichero: */opt/ikusi/ik-iata-api/application.**properties***:

Propiedad	Descripción	Valor
`spring.datasource.url`	URL para la conexión a la base de datos	`jdbc:postgresql://IKIATADB_IP:5432/ikiatadb`
`spring.datasource.username`	Usuario para la conexión a la base de datos	`ikiata`
`spring.datasource.username`	Contraseña para la conexión a la base de datos	`ikiata`
`logging.config`	Ruta del fichero para la generación de logs	`./logback.xml`
`server.port`	Puerto en el que se va a ejecutar la aplicación	`8855`

IKIATADB_IP: dirección IP de la máquina virtual en la que está instalada la base de datos de IK-IATA-API.

IK-IATA-SPA

Manual de Instalación

Ikusi – Ángel Iglesias
Paseo Miramón, 170
2009 San Sebastián
SPAIN
Tel: +34 943 44 88 00
Fax: +34 943 44 88 20
ikusi@ikusi.com
www.ikusi.com

CONTROL DE VERSIONES

Versión	Fecha	Responsable	Comentarios
1.0	11/12/2019	Ricardo Herradura	Versión inicial

INDICE

1. Introducción

El presente documento describe los pasos a seguir para realizar la instalación de la aplicación **IK-IATA-SPA**, interfaz web que permite el acceso a IK-IATA-API.

2. Requisitos

Al ser una aplicación auto contenida, es decir, solo hace falta lo siguiente para su instalación:

- ✓ Sistema operativo Linux (RHEL / CentOS 6 o superior)
- ✓ Java JDK versión 1.8.0 (32/64 bits) (ver manual instalación Java JDK 8)
- ✓ Base de datos PostgreSQL versión 9.X (ver manual instalación PostgreSQL 9)
- ✓ Configuración base de datos (ver manual configuración PostgresSQL – IK-IATA-API)
- ✓ Instalación IK-IATA-API (ver manual instalación – IK-IATA-API)
- ✓ Configuración IK-IATA-API (ver manual configuración – IK-IATA-API)

3. Instalacion

- Realizar el proceso de instalación como usuario **root**
- Descargar la última versión del paquete de la aplicación del repositorio: ***ik-iata-spa-dist.tar.gz***
- Copiar el paquete en el directorio de instalación: **/var/www**

```
# cp ik-iata-spa-dist.tar.gz /var/www
```

- Descomprimir el fichero **ik-iata-spa-dist.tar.gz** y borrar el mismo:

```
# cd /var/www
# tar xvf ik-iata-spa-dist.tar.gz
# rm -f ik-iata-spa-dist.tar.gz
```

- Renombrar el directorio a **/var/www/ik-iata-spa**:

```
# mv /var/www/ik-iata-spa-dist /var/www/ik-iata-spa
```

- Editar el fichero .env ubicado en el directorio **/var/www/ik-iata-spa/** siguiendo el manual de configuración: IK-IATA - Manual de Configuración - IK-IATA-SPA

- Instalar las dependencias y crear el ejecutable:

```
# npm install
# npm run build
```

- Una vez seguidos los pasos anteriores, se necesita instalar **Nginx**

```
# apt-get install nginx
```

- Configurar el archivo /etc/nginx/sites-available/default con el editor ***vim***:

```
server {
  listen 80 default_server;
  root /var/www/ik-iata-spa/build;
  index index.html index.htm;
  location / {
  }
}
```

4. Arranque y para del servicio ik-iata-api

A continuación, se especifica los pasos para arrancar y parar la aplicación.

4.1 Arrancar

Ik-iata-spa puede arrancarse iniciando el servivio de nginx ejecutando el siguiente comando como root:

```
# service nginx start
```

En caso de que fuese necesario con el siguiente comando se reiniciaría el servicio de **Nginx**

```
# service nginx restart
```

4.2 Parar

Para parar ik-iata-api, se ejecuta el siguiente comando como root:

```
# service nginx stop
```

5. Firewall

5.1 Reglas del Firewall

Para el acceso desde el exterior a ik-iata-spa, los puertos deben estar abiertos, para ello debemos ejecutar los siguientes comandos:

```
# firewall-cmd --permanent --zone=public --add-service=http
# firewall-cmd --permanent --zone=public --add-service=https
```

En caso de querer abrir algun puerto distinto, ejecutar el siguiente comando

```
# firewall-cmd --permanent --zone=public --add-port=8850/https
```

Recargar la configuración del firewall:

```
# firewall-cmd --reload
```

IK-IATA-SPA

Manual de configuración

Ikusi – Ángel Iglesias
Paseo Miramón, 170
20014 San Sebastián
SPAIN
Tel: +34 943 44 88 00
Fax: +34 943 44 88 20
ikusi@ikusi.com
www.ikusi.com

CONTROL DE VERSIONES

Versión	Fecha	Responsable	Comentarios
1.0	11/12/2019	Ricardo Herradura	Versión Inicial

INDICE

1. Introducción

El presente documento describe la configuración de la aplicación **ik-iata-spa**. La configuración se realiza mediante la edición del fichero "***.env"*** que se encuentran en el directorio /**var**/**www**/ **ik-iata -spa**/

Información necesaria para la configuración:

- **`IKIATA_IP`**: dirección IP de la máquina virtual en la que está instalada la aplicación de IK-IATA-API.

2. Archivos de configuración de la aplicación

2.1 Configuración general de ik-iata-spa

Los parámetros generales de la aplicación se configuran editando las siguientes propiedades en el fichero: */var/www/ik-iata-spa/.env*

Propiedad	Descripción	Valor
`api.ip`	URL para la conexión a la aplicacion	`192.168.243.75`
`server.port`	Puerto en el que se va a ejecutar la aplicación	`8855`

IKIATA_IP: dirección IP de la máquina virtual en la que está instalada la aplicación de IK-IATA-API.

DOCKER-IK-IATA

Manual de Instalación en CentOS 7

Ikusi – Ángel Iglesias
Paseo Miramón, 170
2009 San Sebastián
SPAIN
Tel: +34 943 44 88 00
Fax: +34 943 44 88 20
ikusi@ikusi.com
www.ikusi.com

CONTROL DE VERSIONES

Versión	Fecha	Responsable	Comentarios
1.0	11/12/2019	Ricardo Herradura	Versión inicial

INDICE

1. Introducción

El presente documento describe los pasos a seguir para realizar la instalación de la aplicación **DOCKER-IK-IATA**, proyecto que engloba y reúne los diferentes proyectos de IK-IATA-DB, IK-IATA-API e IK-IATA-SPA.

2. Requisitos

Al ser una aplicación auto contenida, es decir, solo hace falta lo siguiente para su instalación:

- ✓ Sistema operativo Linux (RHEL / CentOS 7 o superior)
- ✓ Java JDK versión 1.8.0 (32/64 bits) (ver manual instalación Java JDK 8)
- ✓ Base de datos PostgreSQL versión 9.X (ver manual instalación PostgreSQL 9)
- ✓ Una cuenta en Docker Hub

3. Instalación y configuración

- Ir a la dirección de abajo, seleccionar la versión deseada y descargar **Docker**

```
https://download.docker.com/linux/centos/7/x86_64/stable/Packages/
```

- Instalar ***Docker Engine – Community*** mediante el comando siguiente, en donde `/path/to/package` es la ruta en la cual se ha descargado el archivo seleccionado:

```
# apt-get install /path/to/package.rpm
```

- Iniciar ***Docker Engine – Community***:

```
# systemctl start docker
```

- Descargar la última versión del proyecto de la aplicación del repositorio: ***docker-ik-iata***
- Crear el directorio y copiar en el directorio el proyecto: /**docker/ik-iata**

4. Arranque y para del servicio docker-ik-iata

A continuación, se especifica los pasos para arrancar y parar la aplicación.

4.1 Arrancar

- Ejecutar el archivo:

```
start.product-owner.bat
```

- Si todo va bien, la ventana de la consola se mantendrá abierta.

- Cuando en la consola aparezca una línea como la siguiente, ya han arrancado todos los servicios. **No cerrar la ventana**:

```
[com.ikusi.ikiata.api.IkIataApiApplication:57] Started IkIataApiApplication
in 34.224 seconds (JVM running for 39.163)
```

- Abrir el navegador e ir a la página

```
https://localhost
```

4.2 Notas

- Si durante la ejecución se cierra la ventana de la consola, es probable que haya habido un error en el lanzamiento. En ese caso, reiniciar Docker Desktop e intentar de nuevo. Si siguiera ocurriendo, ejecutar este fichero desde una consola y reportarnos el error que ocurra.

4.3 Parar todos los servicios

- Ejecutar el archivo:

```
stop.bat
```

DOCKER-IK-IATA

Manual de Instalación en Windows 10

Ikusi – Ángel Iglesias
Paseo Miramón, 170
2009 San Sebastián
SPAIN
Tel: +34 943 44 88 00
Fax: +34 943 44 88 20
ikusi@ikusi.com
www.ikusi.com

CONTROL DE VERSIONES

Versión	Fecha	Responsable	Comentarios
1.0	11/12/2019	Ricardo Herradura	Versión inicial

INDICE

1. Introducción

El presente documento describe los pasos a seguir para realizar la instalación de la aplicación **DOCKER-IK-IATA**, proyecto que engloba y reúne los diferentes proyectos de IK-IATA-DB, IK-IATA-API e IK-IATA-SPA.

2. Requisitos

Al ser una aplicación auto contenida, es decir, solo hace falta lo siguiente para su instalación:

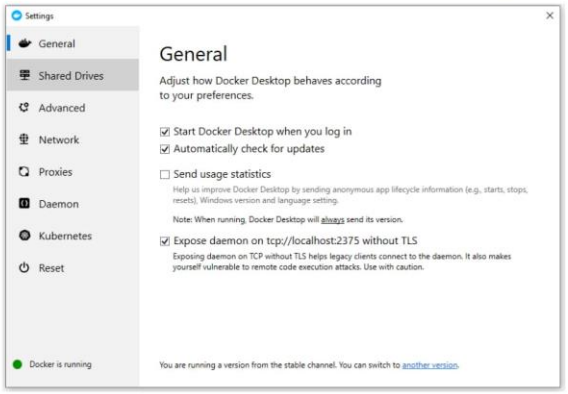
- ✓ Sistema operativo Windows (Windows 10 o superior)
- ✓ Java JDK versión 1.8.0 (32/64 bits) (ver manual instalación Java JDK 8)
- ✓ Base de datos PostgreSQL versión 9.X (ver manual instalación PostgreSQL 9)
- ✓ Una cuenta en Docker Hub

3. Instalación y configuración

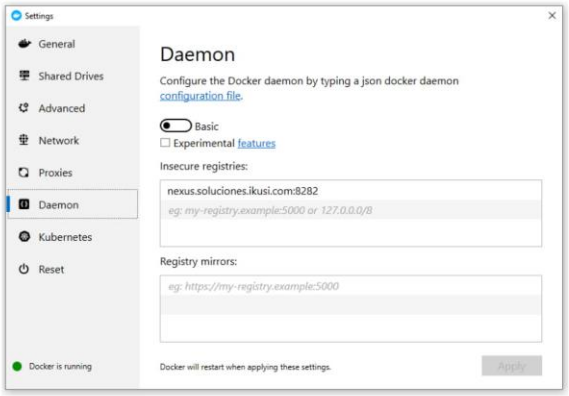
- Descargar **Docker Desktop**

  https://hub.docker.com/editions/community/docker-ce-desktop-windows

- Iniciar la aplicación y entrar en *Settings*
- En la pestaña *Shared Drives* marcar el recuadro *Expose daemon on tcp://localhost:2375 without TLS*



- En la pestaña *Daemon* escribir la ruta del *nexus*



- Descargar la última versión del proyecto de la aplicación del repositorio: ***docker-ik-iata***
- Crear el directorio y copiar en el directorio el proyecto: /**docker/ik-iata**

4. Arranque y para del servicio docker-ik-iata

A continuación, se especifica los pasos para arrancar y parar la aplicación.

4.1 Arrancar

- Ejecutar el archivo:

```
start.product-owner.bat
```

-
- Si todo va bien, la ventana de la consola se mantendrá abierta.
- Cuando en la consola aparezca una línea como la siguiente, ya han arrancado todos los servicios. **No cerrar la ventana**:

```
[com.ikusi.ikiata.api.IkIataApiApplication:57] Started IkIataApiApplication
in 34.224 seconds (JVM running for 39.163)
```

- Abrir el navegador e ir a la página

```
https://localhost
```

4.2 Notas

- Si durante la ejecución se cierra la ventana de la consola, es probable que haya habido un error en el lanzamiento. En ese caso, reiniciar Docker Desktop e intentar de nuevo. Si siguiera ocurriendo, ejecutar este fichero desde una consola y reportarnos el error que ocurra.

4.3 Parar todos los servicios

- Ejecutar el archivo:

```
stop.bat
```

IK-IATA

Manual de Usuario

Ikusi – Ángel Iglesias

Paseo Miramón, 170
2009 San Sebastián
SPAIN
Tel: +34 943 44 88 00
Fax: +34 943 44 88 20
ikusi@ikusi.com
www.ikusi.com

CONTROL DE VERSIONES

Versión	Fecha	Responsable	Comentarios
1.0	11/12/2019	Ricardo Herradura	Versión inicial

INDICE

TABLA ILUSTRACIONES

1. Introducción

IK-IATA es una aplicación web desarrollada por IKUSI que maneja y gestiona las tablas maestras con informacion IATA y las provee a las implantaciones de IKUSI AMS (Airport Management System).

El presente documento está dirigido a todos los usuarios de la aplicación. Su finalidad es explicar de forma detallada el interfaz gráfico de la herramienta, las funcionalidades principales y la forma de uso de la misma.

2. Acceso a IK-IATA

Para acceder a la aplicación (http://IP_ADDRESS/) es necesario superar el proceso de autenticación del usuario mediante la introducción del usuario / contraseña asignados para al usuario previamente.

En la ventana de autenticación hay dos campos de texto: **"Usuario"** y **"Contraseña"** y un botón: **"Entrar"**

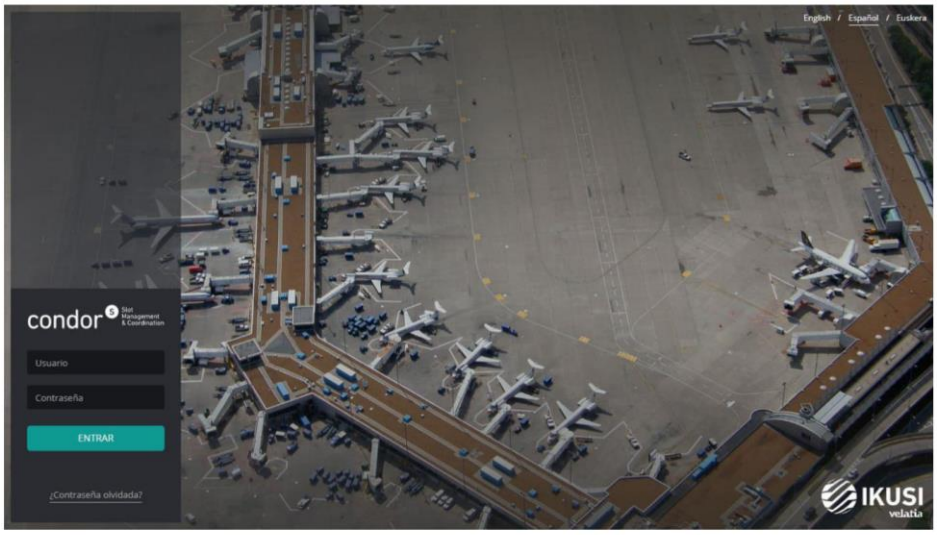


Ilustración 1 – Autenticación - Inicio

Una vez rellenados los campos con los datos del usuario y pulsado el botón de "**Entrar**", en caso de que el usuario sea correcto accederá a la pantalla de inicio de la aplicación. En cambio, si alguno de los dos campos no es correcto, aparecerá en pantalla el mensaje: **"Usuario o contraseña son incorrectas".**

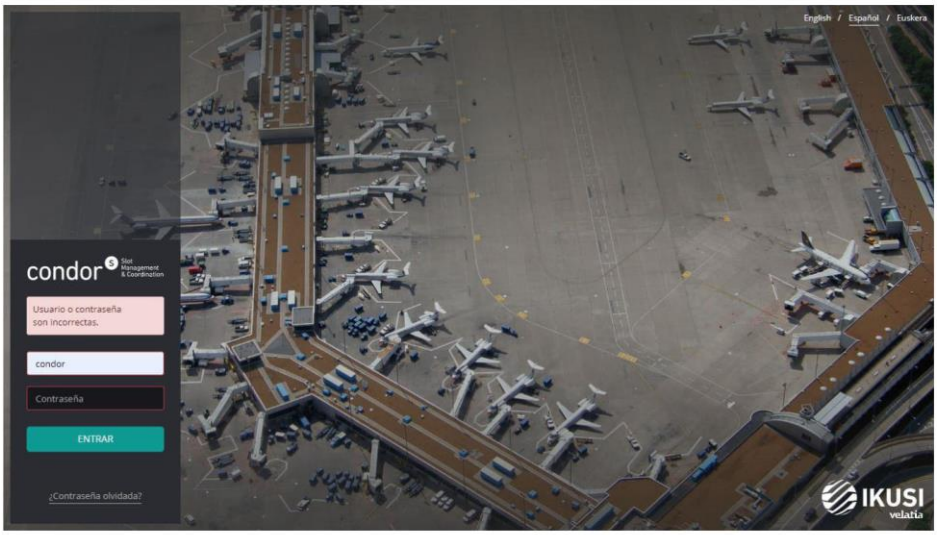


Ilustración 2 – Autenticación - Error

En la parte superior del formulario de autenticación, se podrá cambiar el idioma de la aplicación. Las opciones disponibles son EN (inglés), ES (castellano) y EU (euskera).

En la parte inferior del formulario de autenticación se podrá solicitar la contraseña si se ha olvidado introduciendo el email del registro.

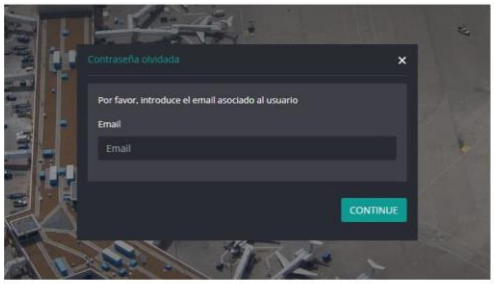


Ilustración 3 – Contraseña olvidada

Una vez superado este proceso el usuario permanecerá autenticado durante toda la sesión.

3. Vista general

Si el proceso de autenticación se ha realizado de manera correcta, el usuario accederá a la pantalla de inicio donde encontramos una pequeña introducción con las instrucciones de manera resumida.

Ilustración 4 – Vista general – Pantalla de inicio

En la parte superior izquierda, encontramos dos iconos: con el de la izquierda podemos seleccionar el idioma y con el otro salir de la sesión. En la parte superior derecha podemos ver el icono **S** que oculta el menú principal.

Ilustración 5 – Vista general – Menú e idiomas

3.1 Estructura de las pantallas

Las interfaces de *aeropuertos, aerolíneas y aeronaves* se dividen en tres aéreas simetricas:

- Panel superior
- Operaciones
- Tabla de datos

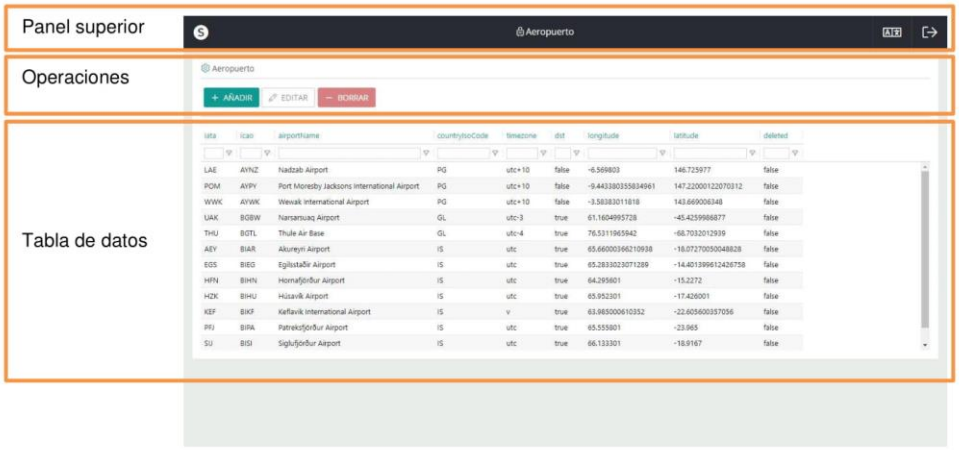


Ilustración 6 – Vista general - Ejemplo de pantalla

En cambio, la interfaz de *archivos*, aunque siga la misma estructura, es un poco distinta:

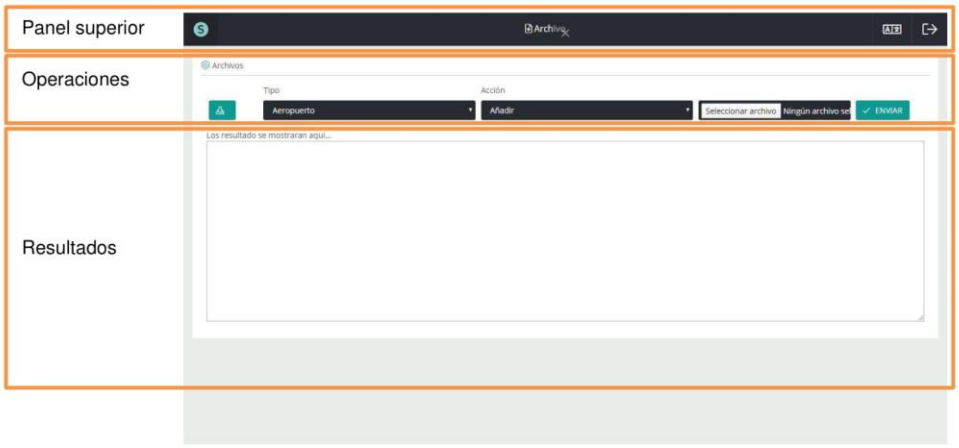


Ilustración 7 – Vista general – Ejemplo de pantalla archivo

En los siguientes apartados se detalla el funcionamiento de los elementos comunes de las interfaces.

3.2 Panel superior

Ilustración 8 – Vista general - Panel superior

A continuación, se detallan las funcionalidades del panel superior:

1. **Icono del menú:** Al pulsar en el icono se abre el menú principal.
2. **Título de la pantalla actual**
3. **Icono de idiomas:** Al pulsar se despliega un menú con las opciones para cambiar de idioma
4. **Icono de salir:** Al pulsar el icono cerramos sesión.

3.3 Operaciones

El área de operaciones permite realizar ciertas operaciones para la gestión de los datos.

En las interfaces de *aeropuerto, aerolínea y aeronave*, nos encontramos con 3 botones que abren un modal con un formulario para rellenar, editar o borrar el objeto. En este punto, aunque sean parecidas las modales de cada interfaz, contienen datos distintos:

- **AÑADIR**: Nos permite crear un nuevo objeto.

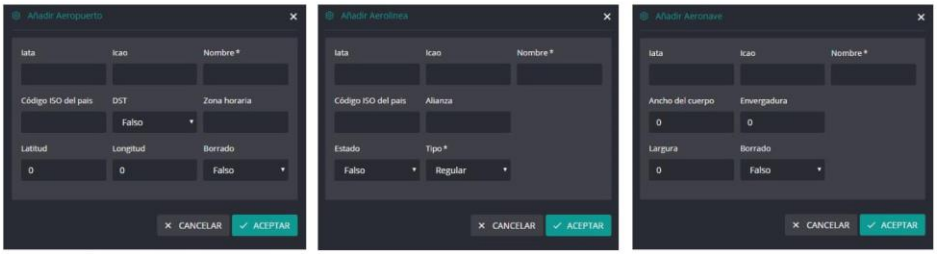


Ilustración 9 – Vista general – Operaciones Añadir

- **EDITAR**: Nos permite editar un objeto existente. Al principio el botón aparece deshabilitado hasta que se selecciona una fila.

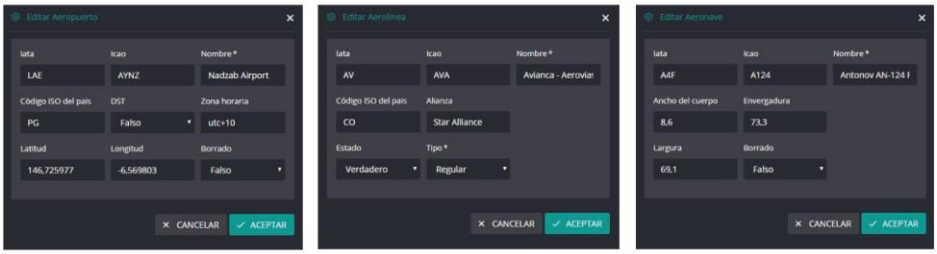


Ilustración 10 – Vista general – Operaciones Editar

- **BORRAR**: Nos permite borrar un objeto existente. Al principio el botón aparece deshabilitado hasta que se selecciona una fila.

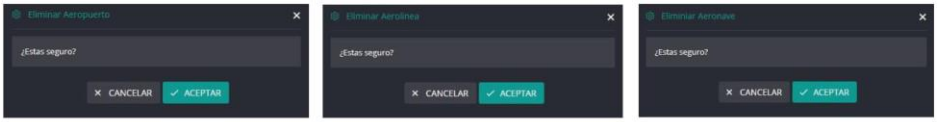


Ilustración 11 – Vista general – Operaciones Borrar

En el caso de las operaciones de la interfaz de *archivos*, podemos descargar el *modelo de subida de archivos* y, podemos elegir el objeto con el que deseamos realizar la operación, que operación queremos realizar y seleccionar el archivo:

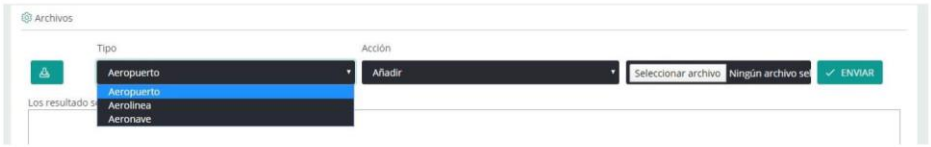
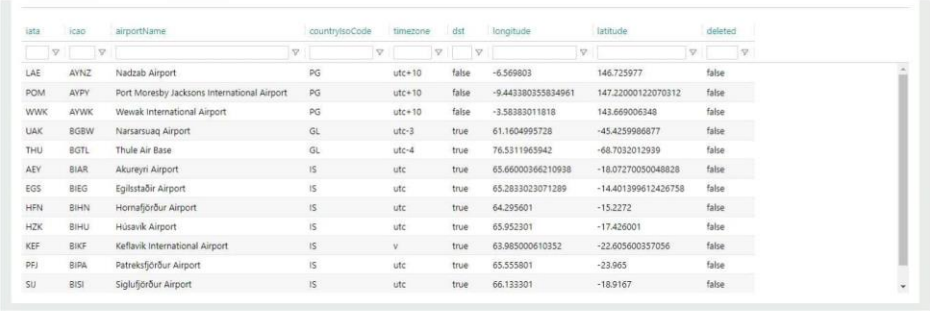


Ilustración 12 – Vista general – Selección tipo subida archivo

Ilustración 13 – Vista general – Selección acción subida archivo

3.4 Tabla de datos

En el área de *Tabla de datos* podemos visualizar los registros que activados existen. En este apartado se detallas las funcionalidades comunes de dichas tablas.



iata	icao	airportName	countryIsoCode	timezone	dst	longitude	latitude	deleted
LAE	AYNZ	Nadzab Airport	PG	utc+10	false	-6.569803	146.725977	false
POM	AYPY	Port Moresby Jacksons International Airport	PG	utc+10	false	-9.443380355834961	147.22000122070312	false
WWK	AYWK	Wewak International Airport	PG	utc+10	false	-3.58383011818	143.669006348	false
UAK	BGBW	Narsarsuaq Airport	GL	utc-3	true	61.1604995728	-45.4259986877	false
THU	BGTL	Thule Air Base	GL	utc-4	true	76.5311965942	-68.7032012939	false
AEY	BIAR	Akureyri Airport	IS	utc	true	65.66000366210938	-18.07270050048828	false
EGS	BIEG	Egilsstaðir Airport	IS	utc	true	65.2833023071289	-14.401399612426758	false
HFN	BIHN	Hornafjörður Airport	IS	utc	true	64.295601	-15.2272	false
HZK	BIHU	Húsavík Airport	IS	utc	true	65.952301	-17.426001	false
KEF	BIKF	Keflavik International Airport	IS	v	true	63.985000610352	-22.605600357056	false
PFJ	BIPA	Patreksfjörður Airport	IS	utc	true	65.555801	-23.965	false
SU	BISI	Siglufjörður Airport	IS	utc	true	66.133301	-18.9167	false

Ilustración 14 – Vista general - Ejemplo tabla de datos

3.4.1 Búsqueda

Se podrán realizar búsquedas en los registros cargados en las tablas de datos mediante la caja de texto situada en la parte superior de cada columna. Al escribir en la caja de texto, la tabla se recarga con los registros que contengan el valor introducido. Para volver a visualizar los registros iniciales se debe borrar el texto introducido en la búsqueda.

iata	icao	airportName	countryIsoCode	timezone	dst	longitude	latitude	deleted

Ilustración 15 – Vista general – Búsqueda en tabla

También, se puede realizar la búsqueda buscando registros que **no contengan** el valor indicado. Este cambio de búsqueda se realiza clicando en el icono de al lado de las cajetillas y seleccionando ***Not Contains***

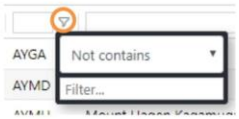


Ilustración 16 – Vista general – Búsqueda tabla no contenga

3.5 Resultados

En esta área de la interfaz *archivos*, recibimos una respuesta a la subida del archivo que hayamos realizado indicando:

1) Objetos se han guardado correctamente
2) Líneas del archivo enviado han fallado
3) La cantidad de objetos correctamente procesados y objetos fallados.

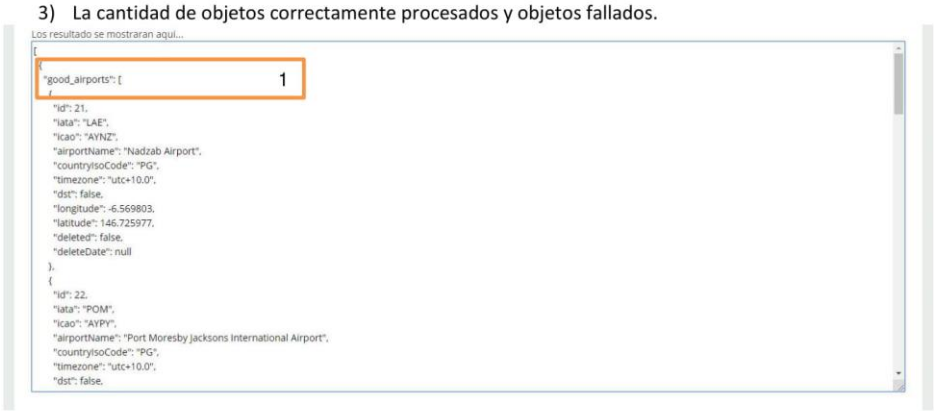


Ilustración 17 – Vista general – Resultado subida archivo

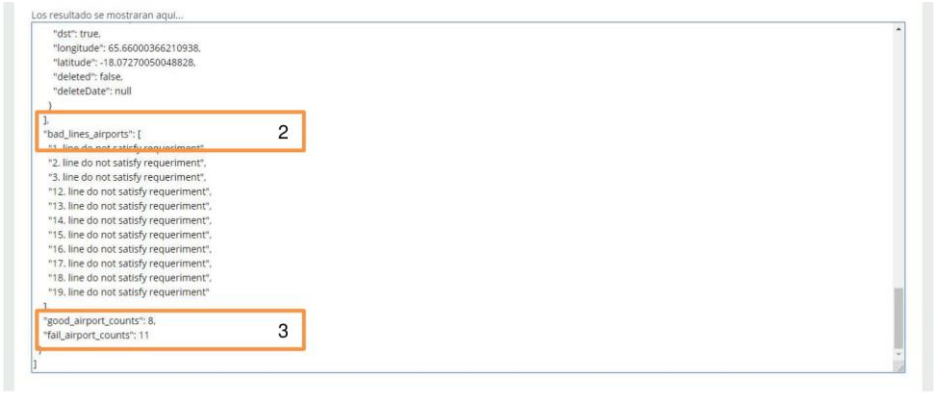


Ilustración 18 – Vista general – Resultado subida archivo 2

4. Mantenimiento

Como se ha visto en el punto 4.3 Operaciones, podemos realizar las operaciones de Añadir, Editar o Borrar en las diferentes interfaces. En los siguientes puntos se explicará con detalle cada una de las acciones en cada interfaz.

4.1 Aeropuerto

En la pestaña de *aeropuerto*, en la tabla nos encontramos con los siguientes datos:

- **iata:** Código IATA del aeropuerto.
- **icao:** Código ICAO del aeropuerto.
- **airportName:** Nombre del aeropuerto.
- **countryIsoCode:** Código ISO del país.
- **timezone:** Zona horaria del aeropuerto.
- **dst:** Si existe cambio horario durante el año.
- **longitude:** Longitud a la que se encuentra el aeropuerto.
- **latitude:** Latitud a la que se encuentra el aeropuerto.

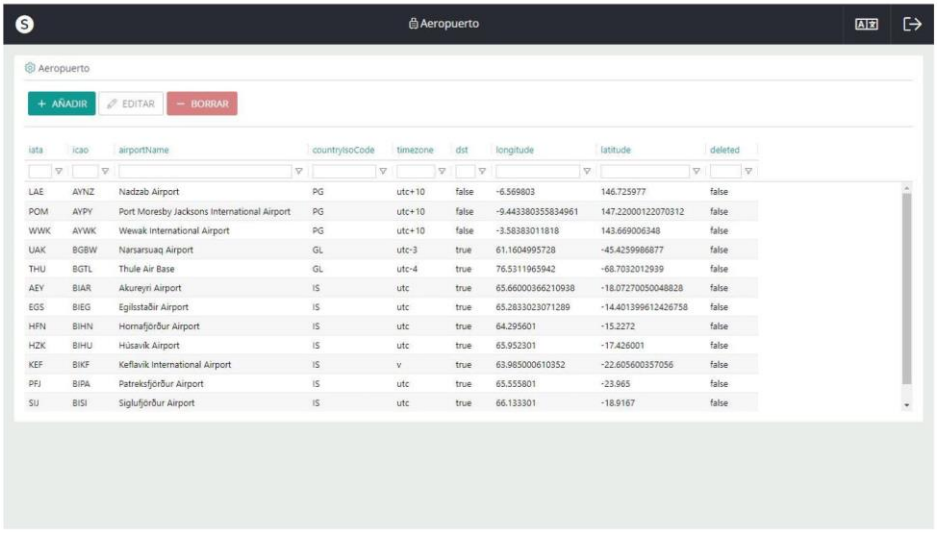


Ilustración 19 – Aeropuertos

4.1.1 Añadir Aeropuerto

Clicando sobre el botón + AÑADIR se abre una ventana modal con un formulario para rellenar los datos

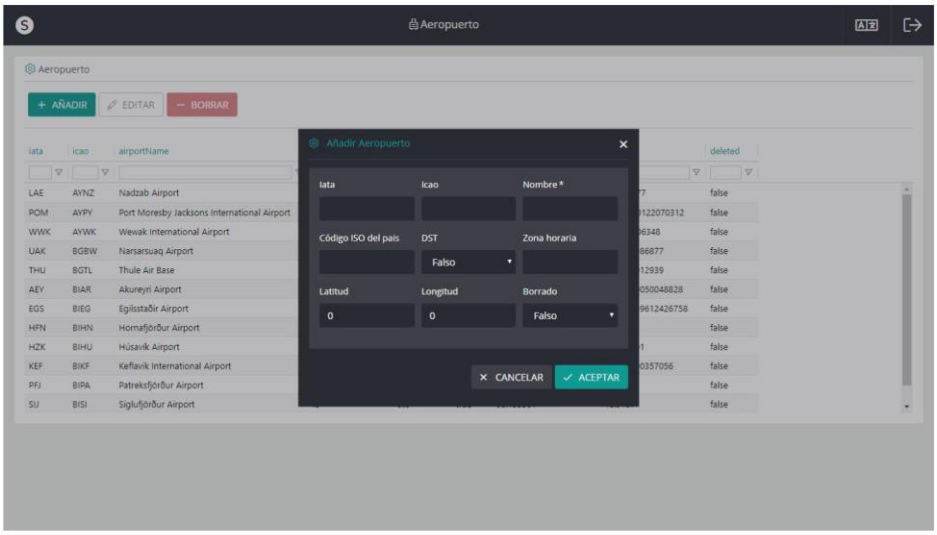


Ilustración 20 – Nuevo aeropuerto

Los campos para rellenar tienen ciertas limitaciones que vamos a explicar a continuación:

- **Iata:** Tiene que ser un código de **3** caracteres o vacío, tiene que ser único no puede estar repetido ni, aunque sea de otro aeropuerto que esta desactivado.
- **Icao:** Tiene que ser un código de **4** caracteres o vacío, tiene que ser único no puede estar repetido ni, aunque sea de otro aeropuerto que esta desactivado.
- **Nombre**(*airportName*)**:** Tiene un límite de **20** caracteres.
- **Código ISO del país**(*countryIsoCode*)**:** Tiene que ser un código que real que exista y solo de **2** letras.
- **DST:** Indicar si es verdadero o falso.
- **Zona horaria**(*timezone*): Tiene que cumplir que ser *utc* o *utc ± y 2 dígitos . 2 dígitos (utc±02.00)*
- **Longitud**(*longitude*)**:** Tiene que ser un número de 17 dígitos máximo.
- **Latitud**(*latitude*)**:** Tiene que ser un número de 17 dígitos máximo.
- **Borrado:** Si el aeropuerto está activado o desactivado.

Los campos con (*), son campos de entrada obligatoria.

Al pulsar el botón "**Cancelar**" se cierra el formulario sin guardar datos.

Al pulsar el botón "**Aceptar**" se realiza el siguiente proceso:

- Se validarán los datos introducidos para poder continuar con el proceso:
  - Comprobación de los campos obligatorios.
  - Comprobación de las limitaciones arriba explicadas

- Si se superan las validaciones, se realizarán las siguientes acciones:
  - Se guardarán los datos en la base de datos.
  - Se cerrará el formulario.
  - Se recargará la tabla de datos.
  - Se muestra un mensaje de existo:

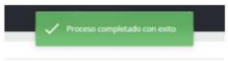


Ilustración 21 – Mensaje OK aeropuerto

- Si no supera las validaciones, se muestran mensajes indicando que existe algún error:
  - En el caso de no completar los campos requeridos:

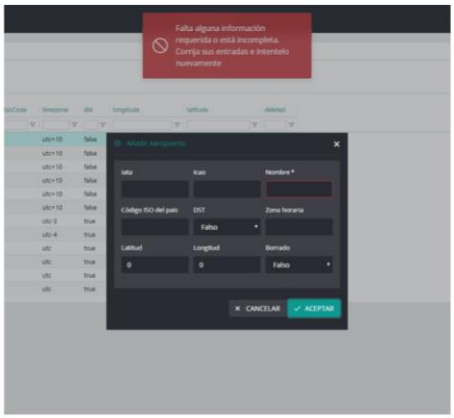


Ilustración 22 – Mensaje campo requerido aeropuerto

- En caso de no superar las limitaciones indicadas:

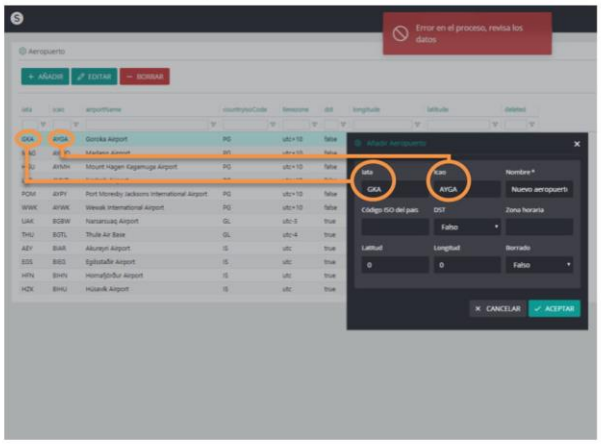


Ilustración 23 – Mensaje error proceso aeropuerto

4.1.2 Editar Aeropuerto

Para editar un aeropuerto se debe seleccionar primero un aeropuerto, y posteriormente clicar sobre el icono EDITAR, sino el boton esta inactivo. El formulario y los campos para la edición son los mismos que en el proceso de añadir aeropuerto y se rigen por las mismas validaciones.

4.1.3 Borrar Aeropuerto

Para borrar un aeropuerto se debe seleccionar primero un aeropuerto, y posteriormente clicar sobre el icono – BORRAR, sino el boton esta inactivo. En este caso, tras pulsar nos aparecera una ventana emergentes para evitar borrados accidentales.

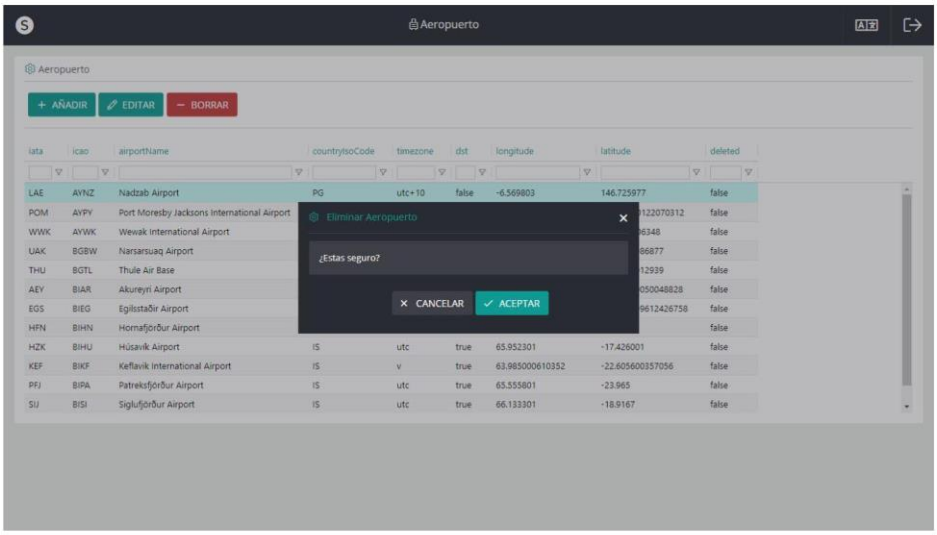


Ilustración 24 – Mensaje borrar aeropuerto

En el mensaje de confirmación:

- Pulsando en "**Aceptar**" se completará el borrado del aeropuerto
- Pulsando en "**Cancelar**" se cerrará el mensaje y no se borrará el aeropuerto.

4.2 Aerolínea

En la pestaña de *aerolínea*, en la tabla nos encontramos con los siguientes datos:

- **iata:** Código IATA de la aerolínea.
- **icao:** Código ICAO de la aerolínea.
- **airlineName:** Nombre de la aerolínea.
- **countryIsoCode:** Código ISO del país.
- **alliance:** Alianza a la que pertenece la aerolínea.
- **type:** Tipo de vuelos que realiza la aerolínea.

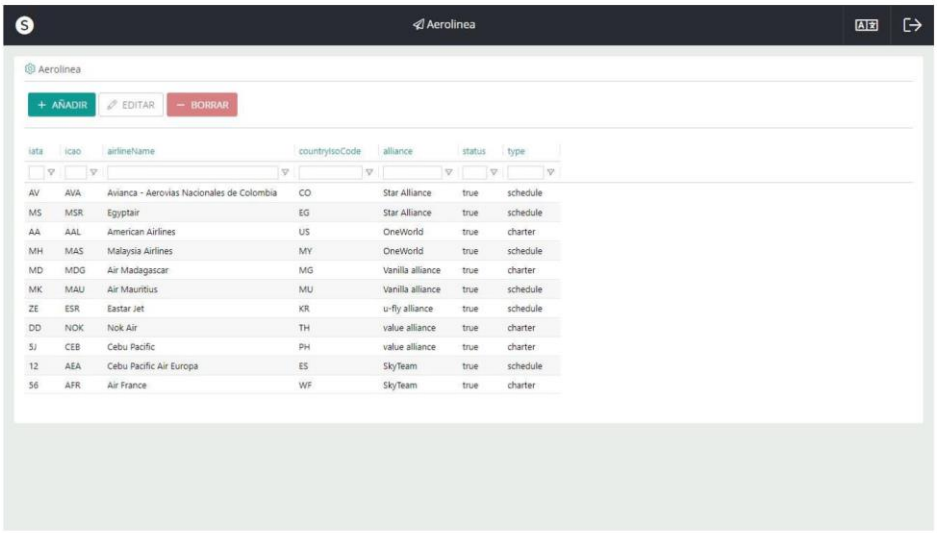


Ilustración 25 – Aerolíneas

4.2.1 Añadir Aerolínea

Clicando sobre el botón + AÑADIR se abre una ventana modal con un formulario para rellenar los datos

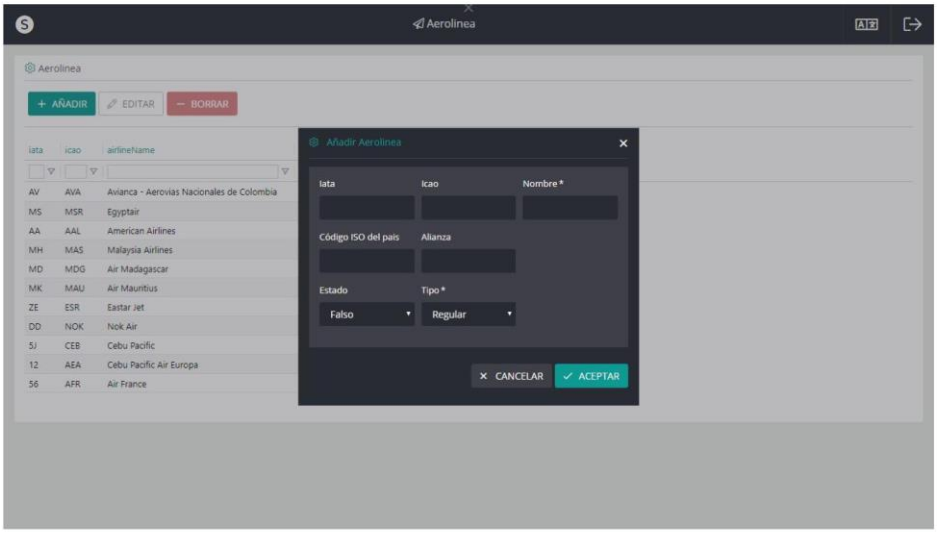


Ilustración 26 – Nueva aerolínea

Los campos para rellenar tienen ciertas limitaciones que vamos a explicar a continuación:

- **Iata:** Tiene que ser un código de **2** caracteres o vacío, tiene que ser único no puede estar repetido salvo que sea de otra aerolínea que esta desactivado.
- **Icao:** Tiene que ser un código de **3** caracteres o vacío, tiene que ser único no puede estar repetido salvo que sea de otra aerolínea que esta desactivado.
- **Nombre**(*airlineName*)**:** Tiene un límite de **20** caracteres.
- **Código ISO del país**(*countryIsoCode*)**:** Tiene que ser un código que real que exista y solo de **2** letras.
- **Alianza**(*alliance*)**:** Tiene un límite de **20** caracteres.
- **Estado:** Si la aerolínea está dada de baja o no.
- **Tipo**(*type*)**:** Indicar si el vuelo el *Regular* o *Charter*.

Los campos con (*), son campos de entrada obligatoria.

Al pulsar el botón "**Cancelar**" se cierra el formulario sin guardar datos.

Al pulsar el botón "**Aceptar**" se realiza el siguiente proceso:

- Se validarán los datos introducidos para poder continuar con el proceso:
  - Comprobación de los campos obligatorios.
  - Comprobación de las limitaciones arriba explicadas
- Si se superan las validaciones, se realizarán las siguientes acciones:
  - Se guardarán los datos en la base de datos.
  - Se cerrará el formulario.
  - Se recargará la tabla de datos.
  - Se muestra un mensaje de existo:

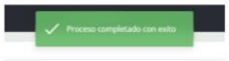


Ilustración 27 – Mensaje OK aerolínea

- Si no supera las validaciones, se muestran mensajes indicando que existe algún error:
    - En el caso de no completar los campos requeridos:

Ilustración 28 – Mensaje campo requerido aerolínea

    - En caso de no superar las limitaciones indicadas:

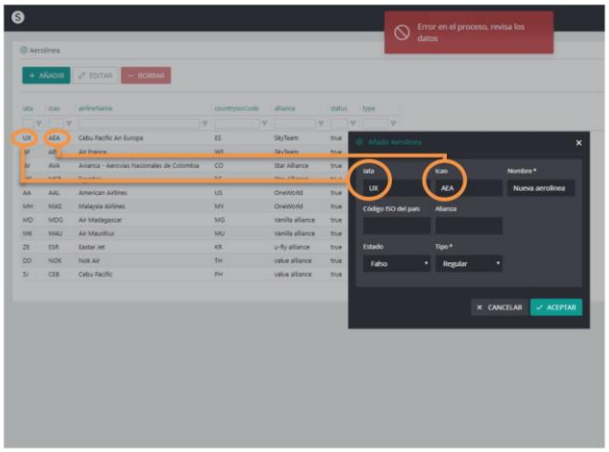


Ilustración 29 – Mensaje error proceso aerolínea

4.2.2 Editar Aerolínea

Para editar una aerolínea se debe seleccionar primero una aerolínea, y posteriormente clicar sobre el icono EDITAR, sino el boton estara inactivo. El formulario y los campos para la edición son los mismos que en el proceso de añadir aerolínea y se rigen por las mismas validaciones.

4.2.3 Borrar Aerolínea

Para borrar una aerolínea se debe seleccionar primero una aerolínea, y posteriormente clicar sobre el icono BORRAR, sino el boton estara inactivo. En este caso, tras pulsar nos aparecera una ventana emergentes para evitar borrados accidentales.

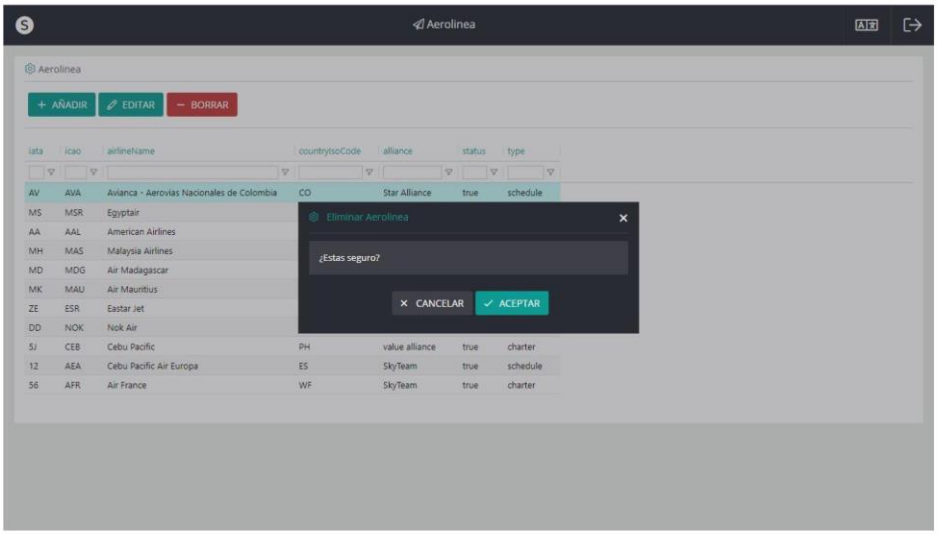


Ilustración 30 – Mensaje borrar aerolínea

En el mensaje de confirmación:

- Pulsando en "**Aceptar**" se completará el borrado de la aerolínea.
- Pulsando en "**Cancelar**" se cerrará el mensaje y no se borrará la aerolínea.

4.3 Aeronave

En la pestaña de *aeronave*, en la tabla nos encontramos con los siguientes datos:

- **iata:** Código IATA de la aeronave.
- **icao:** Código ICAO de la aeronave.
- **aircraftName:** Nombre de la aeronave.
- **wideBody:** Ancho del cuerpo de la aeronave.
- **wingspan:** Envergadura de la aeronave.
- **lenght:** Largura de la aeronave.

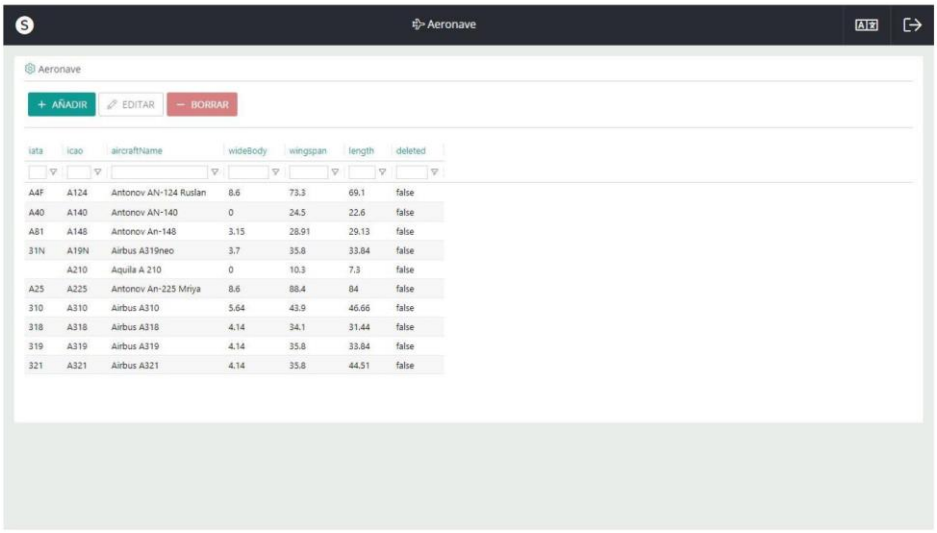


Ilustración 31 – Aeronaves

4.3.1 Añadir Aeronave

Clicando sobre el botón + AÑADIR se abre una ventana modal con un formulario para rellenar los datos

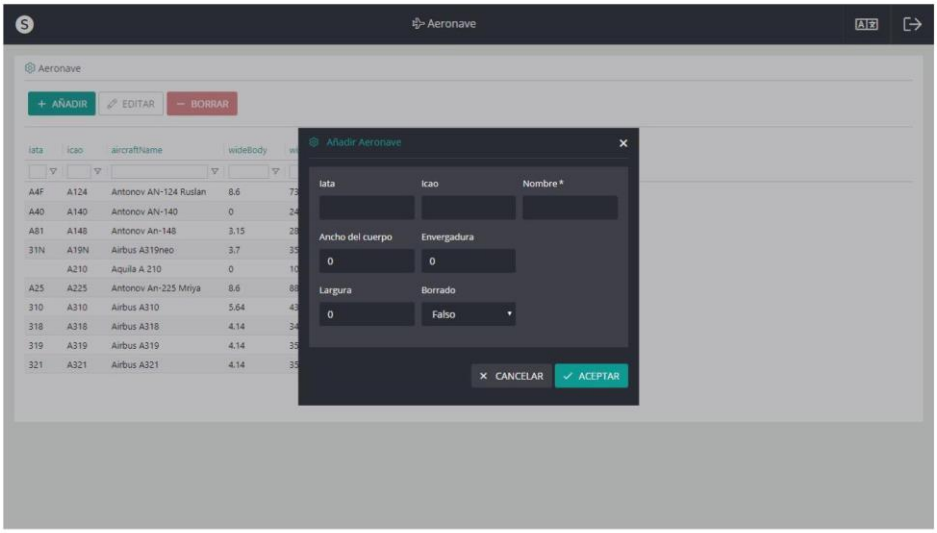


Ilustración 32 – Nueva aeronave

Los campos para rellenar tienen ciertas limitaciones que vamos a explicar a continuación:

- **Iata:** Tiene que ser un código de **3** caracteres o vacío, puede estar repetido.
- **Icao:** Tiene que ser un código de **4** caracteres o vacío, tiene que ser único no puede estar repetido ni, aunque sea de otro aeropuerto que esta desactivado.
- **Nombre**(*aircraftName*)**:** Tiene un límite de **20** caracteres.
- **Ancho del cuerpo**(*wideBody*)**:** Tiene que ser un número de 17 dígitos máximo.
- **Envergadura**(*wingspan*)**:** Tiene que ser un número de 17 dígitos máximo.
- **Largura**(*length*)**:** Tiene que ser un número de 17 dígitos máximo.

Los campos con (*), son campos de entrada obligatoria.

Al pulsar el botón "**Cancelar**" se cierra el formulario sin guardar datos.

Al pulsar el botón "**Aceptar**" se realiza el siguiente proceso:

- Se validarán los datos introducidos para poder continuar con el proceso:
  - Comprobación de los campos obligatorios.
  - Comprobación de las limitaciones arriba explicadas

- Si se superan las validaciones, se realizarán las siguientes acciones:
  - Se guardarán los datos en la base de datos.
  - Se cerrará el formulario.
  - Se recargará la tabla de datos.
  - Se muestra un mensaje de existo:

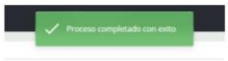


Ilustración 33 – Mensaje OK aeronave

- Si no supera las validaciones, se muestran mensajes indicando que existe algún error:
  - En el caso de no completar los campos requeridos:

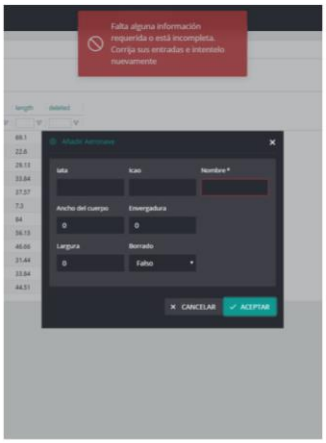


Ilustración 34 – Mensaje campo requerido aeronave

- En caso de no superar las limitaciones indicadas:

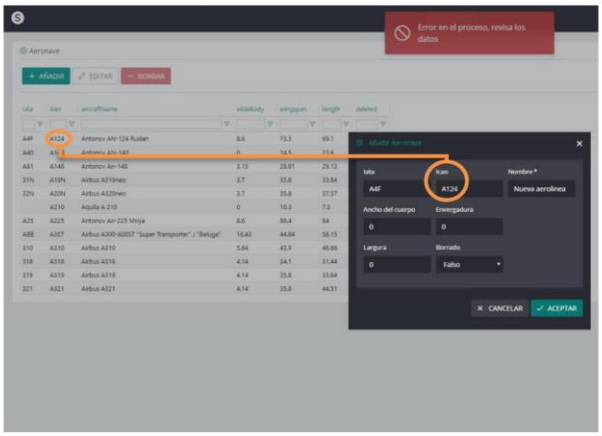


Ilustración 35 – Mensaje error proceso aeronave

4.3.2 Editar Aeronave

Para editar una aeronave se debe seleccionar primero una aeronave, y posteriormente clicar sobre el icono EDITAR, sino el boton esta inactivo. El formulario y los campos para la edición son los mismos que en el proceso de añadir aeronave y se rigen por las mismas validaciones.

4.3.3 Borrar Aeronave

Para borrar una aeronave se debe seleccionar primero una aeronave, y posteriormente clicar sobre el icono BORRAR, sino el boton esta inactivo. En este caso, tras pulsar nos aparecera una ventana emergentes para evitar borrados accidentales.

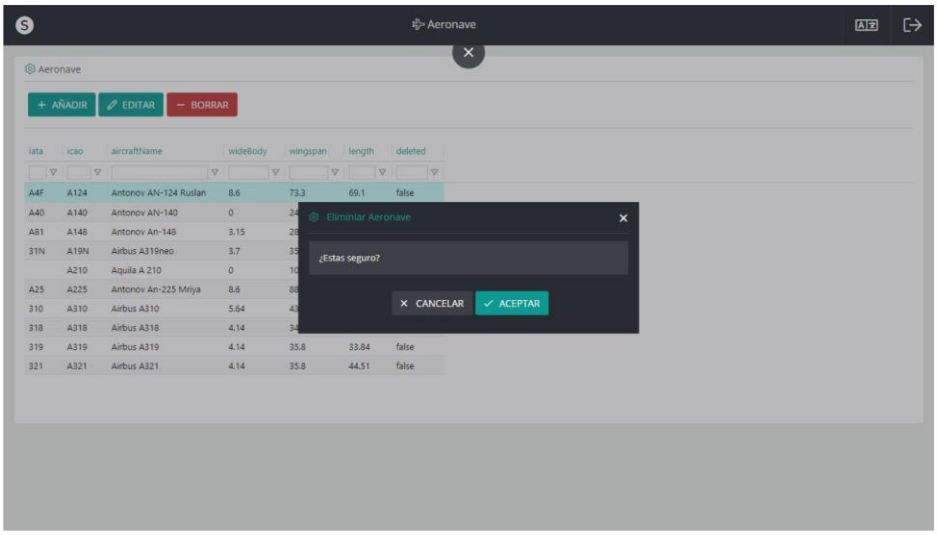


Ilustración 36 – Mensaje borrar aeronave

En el mensaje de confirmación:

- Pulsando en "**Aceptar**" se completará el borrado de la aeronave.
- Pulsando en "**Cancelar**" se cerrará el mensaje y no se borrará la aeronave.

4.4 Archivo

En la pestaña de *aeropuerto*, nos encontramos con:

- **Modelo:** Descarga el modelo que hay que seguir para realizar una correcta subida de archivo.
- **Tipo:** Desplegable con los tipos de objetos que existen (*Aeropuerto, Aerolínea* y *Aeronave*).
- **Acción:** Desplegable con las acciones que están disponibles (*Añadir, Actualizar* y *Borrar*).
- **Seleccionar Archivo:** Permite seleccionar el archivo para subir.
- **Enviar:** Envía el archivo realizando el tipo de acción seleccionada.

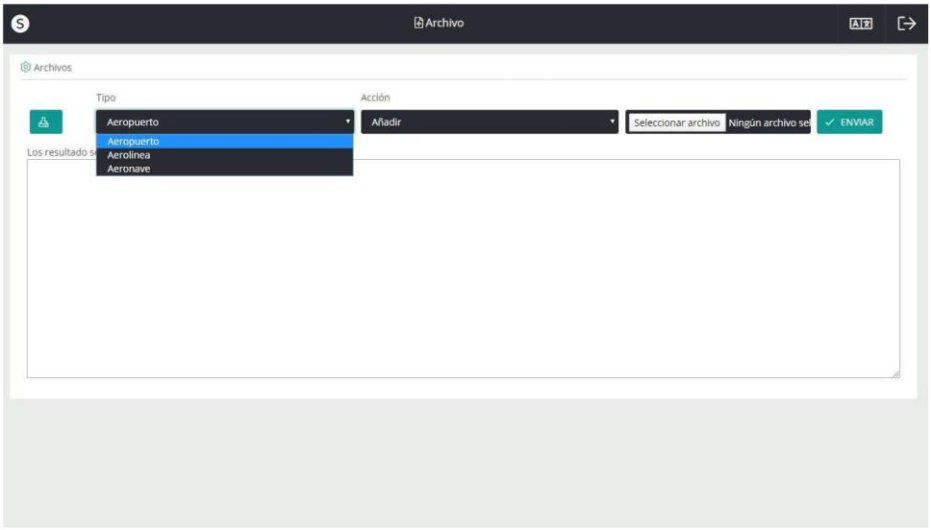


Ilustración 37 – Subida archivo – selección de tipo

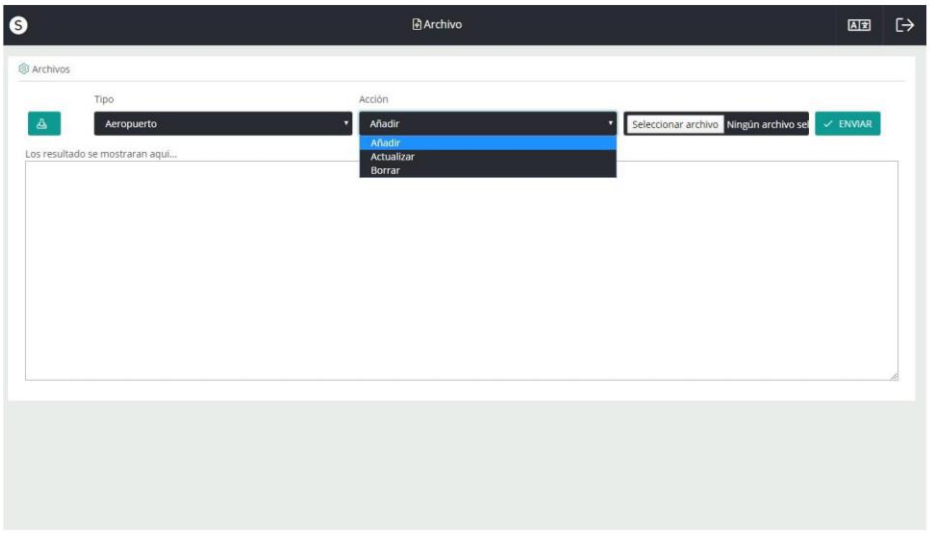


Ilustración 38 – Subida de archivo – selección de acción

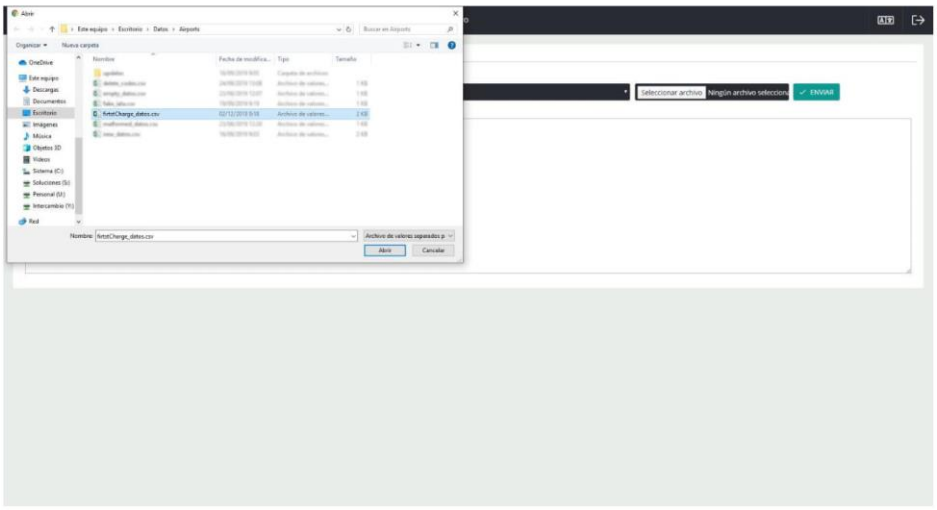


Ilustración 39 – Subida de archivo – selección de archivo

Tras realizar la subida del archivo, nos aparece un mensaje indicando si se ha realizado de manera correcta la subida del archivo o si ha existido algún fallo.

En caso de una subida exitosa:

- En la parte de abajo nos devolverá una respuesta:
  - **Objetos guardados con éxito:** Muestra los objetos guardados con éxito indicando los objetos exitosos con sus valores nuevos.
  - **Objetos fallidos:** Indica que líneas del fichero han dado algún problema.
  - **Cantidad de objetos guardado:** Indica el total de objetos exitosos.
  - **Cantidad de objetos fallidos:** Indica el total de objetos fallidos.
- Mostrará un mensaje de éxito

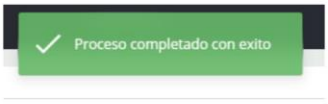


Ilustración 40 – Mensaje OK archivo

En caso de intentar realizar un envió sin archivo, aparecerá un mensaje de error:

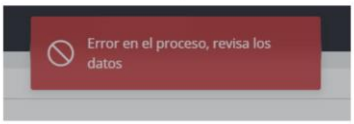


Ilustración 41 – Mensaje error proceso archivo

D) Análisis de sistema de suscripciones

Para poder decidirnos que sistema de colas de mensajes debemos de elegir, debemos de valorar los siguientes puntos:

Tipo de conexión

Antes de nada, debemos entender qué tipo de conexión necesitamos: síncrona o asíncrona.

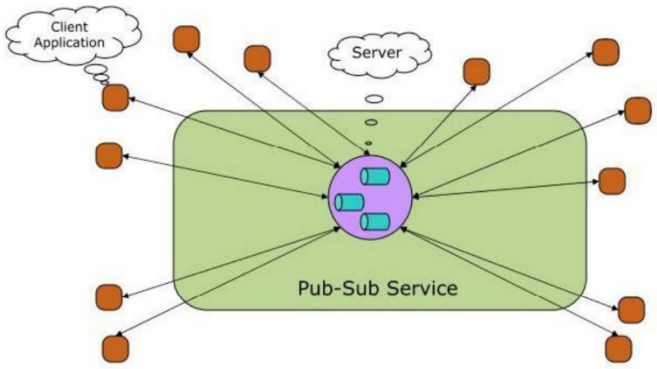
- Síncrona: Nuestro sistema es dependiente y necesita que los dos sistemas (publicador y receptor) coincidan conectados para realizar la comunicación.
- Asíncrona: Nuestro sistema es independiente y no necesita que los dos sistemas coincidan para realizar la comunicación. El mensaje se envía y ya lo recibirá el receptor cuando se conecte.

Tipo de sistema de mensaje

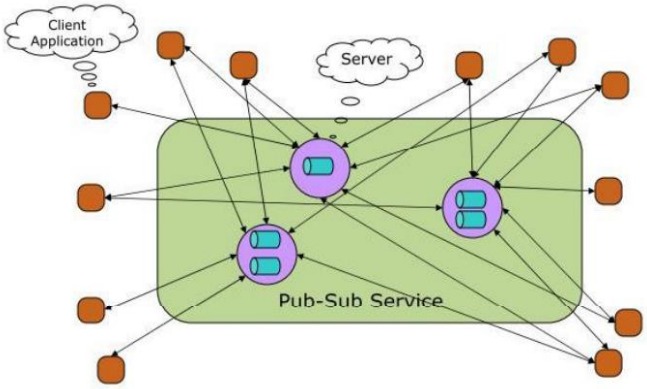
Por otro lado, existen 2 tipos de paradigma de comunicación:

- Punto a punto: El emisor envía el mensaje al receptor directamente. Esta comunicación se genera solo con 2 nodos:

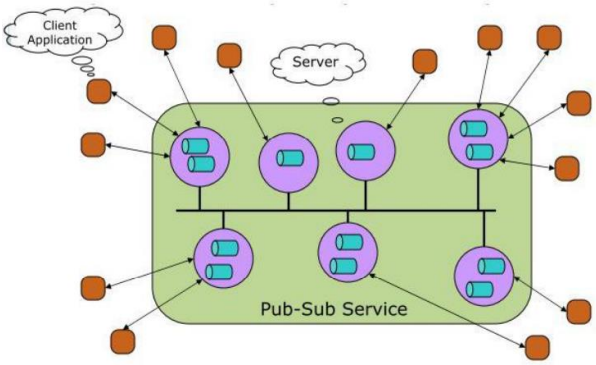
- Publicador/suscriptor: Pueden existir varios nodos que pueden ejercer tanto el papel de publicador o suscriptor. El publicador envía el mensaje mediante un evento o topic (dependiendo de la tecnología) y el suscriptor consume el nexo generado para recibir el mensaje. En este tipo de paradigma, existen 4 tipo de modelos:
  1) Centralizada con intermediarios: Un servidor central recibe el mensaje del publicador y los suscriptores consumen del servidor:



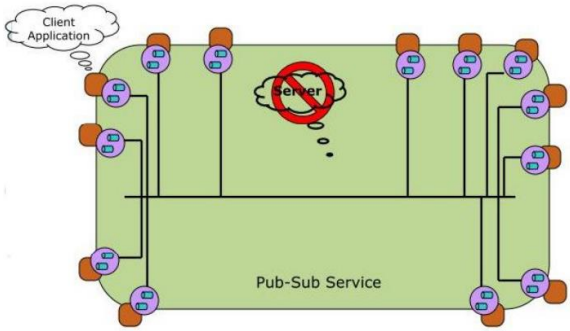
  2) Centralizada con multi-intermediarios: Son varios servidores interconectados entre ellos, en publicador envía el mensaje y los suscriptores consumes el servidor deseado:



3) Descentralizada con multi-intermediarios: Son también varios servidos, pero en este caso, el publicador envía a un servidor en concreto cada mensaje:

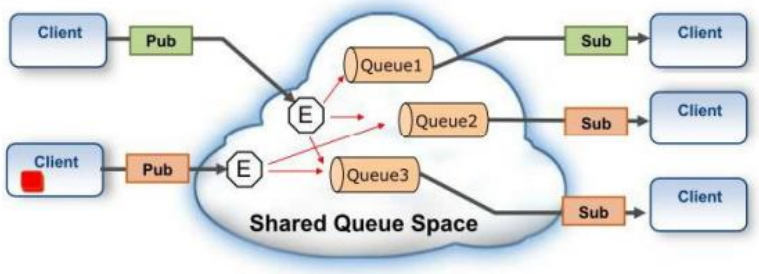


4) Descentralizada sin intermediarios: En este caso, no existes servidores y los mensajes enviados por el publicador generan la cola en el sistema de cada suscriptor:



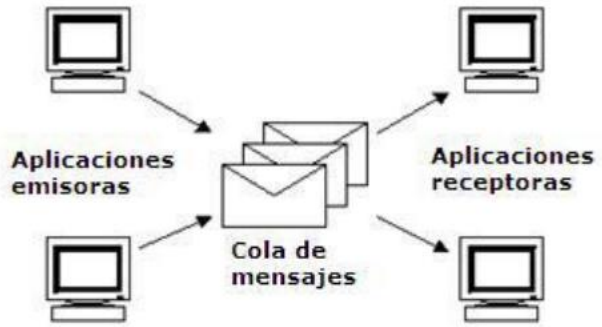
Tecnologías

AMQP: Es un protocolo estándar abierto para la comunicación orientado a mensajería. Esta tecnología está disponible a nivel de API y a nivel de datos enviados mediante cable. Tiene una estructura fija con unas propiedades opcionales relacionadas con la prioridad de los datos, el tiempo de vida de los mismos, el modo de entrega, o la perdurabilidad. Si la aplicación suscriptora no puede procesar los mensajes entrantes, el bróker los almacena en una cola.

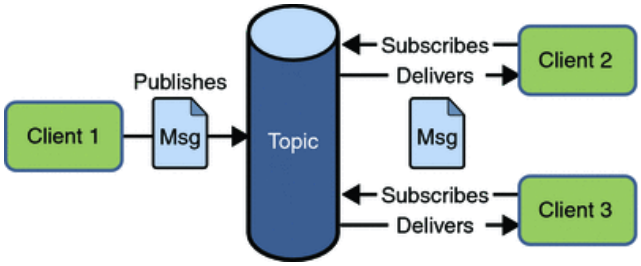


En la imagen se aprecia como cada cliente puede dirigir sus publicaciones al servidor (*Queue)* deseado y luego los suscriptores suscribirse al servidor deseado.

MSMQ: Es una tecnología propietaria de Microsoft, que permite a los sistemas estar inactivos generando cola de mensajes, mientras otros sistemas activos pueden leer la cola generada. Sus características importantes son: Garantiza la entrega, comunicación asíncrona, entrega ordenada, seguridad de certificados…

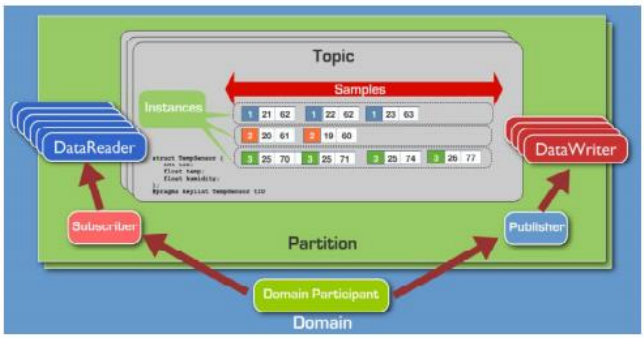


JMS: Es un servicio de mensajería de Java diseñada como una API. Permite crear, enviar, recibir y leer mensajes, definiendo un conjunto de interfaces y posibilitando comunicarse los programas Java mediante mensajes. Generalmente está compuesta por proveedores, clientes, mensajes y objetos administrados (son objetos JMS pre-configurados para el uso de los clientes). Pueden ser de tipo *punto a punto* o *publicador/suscriptor*.

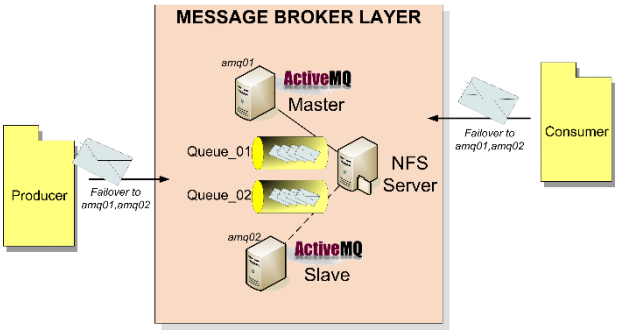


CORBA: Es un servicio de notificaciones, extensión del *Servicio de Eventos de CORBA*, por lo que hereda todas sus características. Esta tecnología se basa en el *publicador/suscriptor* mediante los métodos *push/pull*. Una desventaja del sistema, es que no existe sistema de filtrado por lo que todos los clientes suscritos recibirán todas las notificaciones, aunque si existen filtros que se adhieren a cada *proxy* para poder "redirigir" las notificaciones.

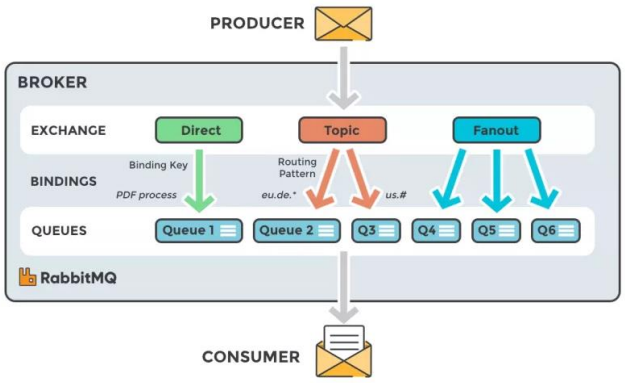
DDS: Es un sistema para el intercambio de datos en sistemas distribuidos en tiempo real, basándose en el paradigma publicador/*suscriptor* y mediante el método *descentralizado sin intermediaros.*



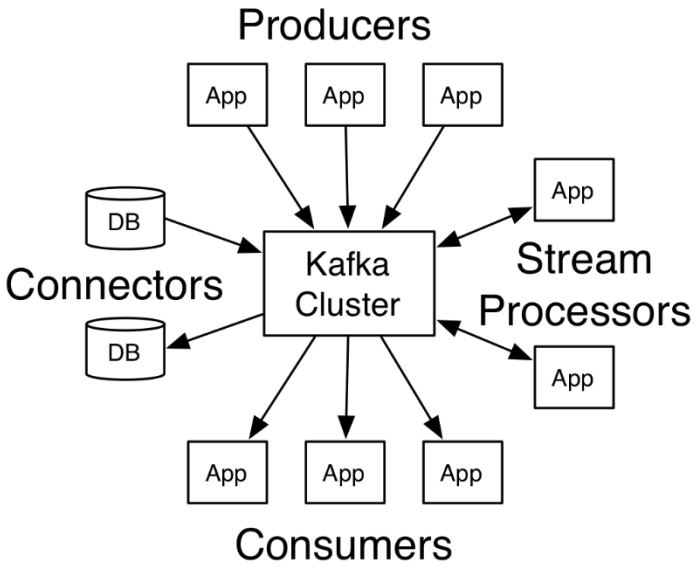
ActiveMQ: En su interior implementa JMS. Ofrece característica como *clustering*, múltiples almacenes de mensajes y la capacidad de emplear cualquier Base de Datos, VM, Cache… Se puede implementar en múltiples lenguajes de programación. Normalmente se usa en proyectos de infraestructuras *SOA*, para la integración entre aplicaciones y servicios.



RabbitMQ: En su interior implementa AMQP. Ofrece características como un servidor de intercambio RabbitMQ en sí mismo, admite varios tipos de lenguaje de programación y tiene un plugin *Shovel* para la réplica de mensajes entre corredores de mensajes. Es un agente de mensajes confiable.



Kafka: Es un proyecto de intermediación de mensajes que tiene como objetivo proporcionar una plataforma unificada de alta rendimiento y baja latencia en tiempo real. Es una cola de mensajes que se basa en el paradigma *publicador/suscriptor* que es escalable a transacciones distribuidas. Se usa comúnmente para la transmisión de datos en tiempo real.



E) Test unitarios

Para empezar, vamos a ver la carpeta Controller que engloba todos los controladores. Como en este punto son bastante parecidos los métodos de las clases, se mostrará solo la clase de *AirportController*. En esta capa de la API, al ser la capa superior con la cual un usuario puede interactuar, se evita realizar cualquier tipo de lógica:

Guardar un aeropuerto

```
@PostMapping(path = "/airports")
public ResponseEntity<Resource<AirportDTO>> save(@Valid @RequestBody
AirportDTO airport)
```

Entrada	Adecuado	No Adecuado
AirportDTO	airport!=null **(1)**	airport==null **(2)**
	airport.iata o airport.icao NO existente **(3)**	airport.iata o airport.icao existente **(4)**

Suponiendo que:

- ***airport*** = { "iata" : "EAS", "icao" : "LESO", "airportName" : "Hondarribi Airport", "countryIsoCode" : "ES", "timezone" : 2, "dst" : true, "longitude" : -6.081689835, "latitude" : 145.3919983, "deleted": false }
- ***airportDuplicado*** = { "iata" : "GKA", "icao" : "AYGA" …}

Entrada	Equivalencia	Resultado
airport	1 , 3	CREATED (200)
null	2	IllegalArgumentException BAD REQUEST (400)
airportDuplicado	4	DuplicatedConstraintException BAD REQUEST (400)

Actualización completa un aeropuerto

```
@PutMapping(path = "/airports/{id}")
public ResponseEntity<Resource<AirportDTO>> update(@ApiParam(value =
"Identifier of the airport to update", required = true) @PathVariable("id")
Long id, @Valid @RequestBody AirportDTO airport)
```

Entrada	Adecuado	No Adecuado
Long	id>0 existente **(1)**	id<=0 **o** NO existente **(2)**
AirportDTO	airport!=null **(3)**	airport==null **(4)**
	airport.iata o airport.icao existentes **(5)**	airport.iata o airport.icao NO existente **(6)**

Suponiendo que:

- ***airport*** = { "iata" : "EAS", "icao" : "LESO", "airportName" : "Hondarribi Airport", "countryIsoCode" : "ES", "timezone" : 2, "dst" : true, "longitude" : -6.081689835, "latitude" : 145.3919983, "deleted": false }
- ***airportDuplicado*** = { "iata" : "GKA", "icao" : "AYGA" …}

Entrada	Equivalencia	Resultado
8, airport	1,3,5	OK (200)
0, airport	2	IllegalArgumentException BAD REQUEST (400)
1, Null	4	IllegalArgumentException BAD REQUEST (400)
1,airportDuplicado	6	DuplicatedConstraintException BAD REQUEST (400)

Realizar primera carga de aeropuertos desde archivo

```
@PostMapping(path = "/airports/firstCharge")
public ResponseEntity<Resource<Object>> processFile(@RequestParam("file")
MultipartFile file)
```

Entrada	Adecuado	No Adecuado
MultipartFile	file!=null **(1)**	file==null **(2)**
	file bien formado **(3)**	file malformado **(4)**

Suponiendo que:

- **file** = es un archivo ".csv" debidamente formado según la estructura predefinida para la subida de archivos del tipo aeropuerto.
- **fileNull** = es un archivo incorrectamente creado o mal subido al servicio.
- **fileMalformado =** es un archivo ".csv" creado sin seguir la estructura predefinida.

Entrada	Equivalencia	Resultado
file	1, 3	OK (200)
fileNull	2	MalformedException BAD REQUEST (400)
fileMalformado	4	MalformedException BAD REQUEST (400)

Actualización parcial de un aeropuerto

```
@PatchMapping(path="/airports/search/update")
public ResponseEntity<Resource<Object>> updateOne(@RequestBody Map<String,
Object> request)
```

Entrada	Adecuado	No Adecuado
Map<String, Object>	request no vacío **(1)**	request vacía **(2)**
	request bien formado **(3)**	request malformado **(4)**

Suponiendo que:

- **request** = está bien formado con los atributos necesarios y correctamente escritos (*{iata: EAS, latitude: 98.5433243}*).
- **requestMalformado =** tiene atributos que no existen en el modelo de datos (*{iata: EAS, capacity:40} capacity en este caso no existe*) .
- **requestVacia =** no envía ningún atributo, es una solicitud vacía.
- **requestErronea =** tiene caracteres que no se reconocen y no se pueden procesar.

Entrada	Equivalencia	Resultado
request	1, 3	OK (200) y objeto actualizado
requestMalformado	4	OK (200) pero fallo en solicitud por no satisfacer los requerimientos
requestVacia	2	BAD REQUEST (400) y mensaje *no puede estar vacía la solicitud*
requestErronea	4	IOException BAD REQUEST (400)

Actualización parcial de varios aeropuertos

```
@PatchMapping(path="/airports/search/update/batch")
public ResponseEntity<Resource<Object>> updatesome(@RequestBody
List<Map<String, Object>> request)
```

Entrada	Adecuado	No Adecuado
List<Map<String, Object>>	request no vacía **(1)**	request vacía **(2)**
	Objetos del request bien formados **(3)**	objetos de request malformados **(4)**

Suponiendo que:

- **request** = es una lista bien formada con los *mapas* correctamente escritos **requestMalformado =** es una lista que tiene *mapas* con atributos que no existen en el modelo de datos (*{iata: EAS, capacity:40} capacity en este caso no existe*).
- **requestVacia =** es una lista vacía.
- **requestErronea =** la lista tiene *mapas* que contienen caracteres que no se reconocen y no se pueden procesar.

Entrada	Equivalencia	Resultado
request	1, 3	OK (200) y lista de objetos actualizados
requestMalformado	4	OK (200) pero mostrando las solicitudes que no satisfacen los requerimientos
requestVacia	2	BAD REQUEST (400) y mensaje *no puede estar vacía la lista*
requestErronea	4	IOException BAD REQUEST (400)

Actualización parcial de uno o varios aeropuertos

```
@PatchMapping(path = "/airports/fileupload")
public ResponseEntity<Resource<Object>>
updateAirportsFromFile(@RequestParam("file") MultipartFile files)
```

Entrada	Adecuado	No Adecuado
MultipartFile	file!=null **(1)**	file==null **(2)**
	file bien formado **(3)**	file malformado **(4)**

Suponiendo que:

- **file** = es un archivo ".csv" debidamente formado según la estructura predefinida para la subida de archivos del tipo aeropuerto.
- **fileNull** = es un archivo incorrectamente creado o mal subido al servicio.
- **fileMalformado =** es un archivo ".csv" creado sin seguir la estructura predefinida.

Entrada	Equivalencia	Resultado
file	1, 3	OK (200)
fileNull	2	MalformedException BAD REQUEST (400)
fileMalformado	4	MalformedException BAD REQUEST (400)

Borrado de varios aeropuertos

```
@DeleteMapping(path = "/airports/batch")
public ResponseEntity<Resource<String>> deleteBatch(@RequestParam
(required=false) String deleteType, @RequestBody List<Long> idList)
```

Entrada	Adecuado	No Adecuado
List<Long>	idList no vacía **(1)**	idList vacía **(2)**
	Objetos del idList bien formados **(4)**	objetos de la idList no existentes **(5)**
String	deleteType=="logical" **(6)**	deleteType!= "logical" y deleteType!=permanent y deleteType!="" **(9)**
	deleteType=="permanent" **(7)**	
	deleteType=="" **(8)**	

Suponiendo que:

- **list** = es una lista con *id* existentes mayores que 0 y bien formados
- **listVacia** = es una lista vacía.
- **listNotExists** = es una lista con alguna de las *id* no existentes.
- **palAleatoria** = cualquier palabra diferente a *logical* o *permanent.*

Entrada	Equivalencia	Resultado
list, "logical"	1, 4, 6	OK (200)
list, "permanent"	1, 4, 7	OK (200)
list, " "	1, 4, 8	OK (200)
list, palAleatoria	1, 4, 9	BAD REQUEST (400) y mensaje *no se puede borrar*
listVacia, "logical"	2, 6	IllegalArgumentException BAD REQUEST (400)

listVacia, ”permanent”	2, 7	IllegalArgumentException BAD REQUEST (400)
listVacia, “ “	2, 8	IllegalArgumentException BAD REQUEST (400)
listVacia, palAleatoria	2, 9	BAD REQUEST (400) y mensaje *no se puede borrar*
listNotExists, ”logical”	5, 6	NotFoundException BAD REQUEST (400)
listNotExists, ”permanent”	5, 7	NotFoundException BAD REQUEST (400)
listNotExists, “ “	5, 8	NotFoundException BAD REQUEST (400)
listNotExists, palAleatoria	5, 9	BAD REQUEST (400) y mensaje *no se puede borrar*

Borrado de uno o varios aeropuertos desde archivo

```
@DeleteMapping(path = "/airports/fileupload")
public ResponseEntity<Resource<Object>> deleteAirportsFromFile(
@RequestParam("file") MultipartFile files)
```

Entrada	Adecuado	No Adecuado
MultipartFile	file!=null **(1)**	file==null **(2)**
	file bien formado **(3)**	file malformado **(4)**

Suponiendo que:

- **file** = es un archivo “.csv” debidamente formado según la estructura predefinida para la subida de archivos del tipo aeropuerto.
- **fileNull** = es un archivo incorrectamente creado o mal subido al servicio.
- **fileMalformado =** es un archivo “.csv” creado sin seguir la estructura predefinida.

Entrada	Equivalencia	Resultado
file	1, 3	OK (200)
fileNull	2	MalformedException BAD REQUEST (400)
fileMalformado	4	MalformedException BAD REQUEST (400)

La próxima carpeta *Service* engloba los métodos referidos que desarrollan la lógica de negocio, en este caso aunque sean métodos muy similares, desarrollan una lógica completamente diferente debido a las limitaciones y validaciones de cada clase:

Aeropuertos

Guardar o actualizar aeropuertos

```
public Airport save(Airport airport) throws NotFoundException,
DuplicatedConstraintException
```

Entrada	Adecuado	No Adecuado
Airport (creando un nuevo registro)	Airport != null **(1)**	Airport ==null **(2)**
	Airport con iata e icao inexistentes **(3)**	Airport con iata o icao existentes **(4)**
Airport (actualizando un objeto existente)	Airport!=null **(5)**	Airport==null **(6)**
	Airport actualización de iata o icao inexistentes **(7)**	Airport!=null inexistente **(8)**
	Airport actualización del resto de atributos **(9)**	Airport actualización de iata o icao existentes **(10)**

Suponiendo que:

- **airport_C** = {iata= "EAS", icao="LESO", airportName="Hondarribi Airport" …}
- **airport_C_Repe** = {iata= "GKA", icao="AYGA", airportName="Goroka Airport" …}
- **airport_C_RepeIata** = {iata= "GKA", icao="TLDA"…}
- **airport_C_RepeIcao** = {iata= "TLA", icao="AYGA"}
- **airport_U** = {iata= "POM", icao="AYPY", airportName="Moresby Airport" …}
- **airport_U_NF** = {iata= "NFD", icao="NFOD", airportName="Not Found Airports" …}
- **airport_U_Repe** = {iata= "GKA", icao="AYGA" …}
- **airport_U_RepeIata** = {iata= "GKA", icao="AYPY" …}
- **airport_U_RepeIcao** = {iata= "POM", icao="AYGA" …}

Entrada	Equivalencia	Resultado
airport_C	1, 3	Devuelve el objeto guardado
airport_C_Repe	1, 4	DuplicatedConstraintException
airport_C_RepeIata	1, 4	DuplicatedConstraintException
airport_C_RepeIcao	1, 4	DuplicatedConstraintException
airport_U	5, 7, 9	Devuelve el objeto guardado
airport_U_NF	5, 10	NotFoundException
airport_U_Repe	5, 8	DuplicatedConstraintException
airport_U_RepeIata	5, 8	DuplicatedConstraintException
airport_U_RepeIcao	5, 8	DuplicatedConstraintException
null	2, 6	NotFoundException

Guardar tras realizar la carga de aeropuertos desde archivo

public Airport charge(Airport airport)

Entrada	Adecuado	No Adecuado
Airport	Iata no existe **(1)**	Iata existe **(3)**
	Iata==null **(2)**	
	Icao no existe **(4)**	Icao existe **(5)**
	Icao==null **(6)**	Airport==null **(7)**

Suponiendo que:

- **airport_0** = {iata= "EAS", icao="LESO" …}
- **airport_1** = {iata= "EAS", icao="AYGA" …}
- **airport_2** = {iata= "EAS", icao= null …}
- **airport_3** = {iata= "GKA", icao="EAS" …}
- **airport_4** = {iata= "GKA", icao="AYGA" …}
- **airport_5** = {iata= "GKA", icao= null …}
- **airport_6** = {iata= null, icao="EAS" …}
- **airport_7** = {iata= null, icao=" AAL " …}
- **airport_8** = {iata= null, icao= null …}

Entrada	Equivalencia	Resultado
airport_0	1, 4	Devuelve el objeto guardado
airport_1	1, 5	null
airport_2	1, 7	Devuelve el objeto guardado
airport_3	3, 4	null
airport_4	3, 5	null
airport_5	3, 7	null
airport_6	2, 4	Devuelve el objeto guardado
airport_7	2, 5	null
airport_8	2, 7	Devuelve el objeto guardado
null	9	null

Actualización parcial de un aeropuerto

public Airport updatePartial(AirportPatch patch) throws IOException

Entrada	Adecuado	No Adecuado
AirportPatch	iata sea igual a algún valor **(1)**	AirportPatch == null **(2)**
	icao sea igual a algún valor **(3)**	iata == null **(4)**
	iata == "NotExists" **(5)**	icao == null **(7)**
	icao == "NotExists" **(6)**	
	iata exista en BD y la lista obtenida solo sea de un elemento **(8)**	iata no exista en BD **(9)**
	iata exista en BD y la lista obtenida de varios elementos y icao exista en la BD **(10)**	iata exista en BD y la lista obtenida sea de varios elementos y icao no exista **(11)**
	icao exista en BD y la lista obtenida solo sea de un elemento **(12)**	icao no exista en BD **(13)**
	icao exista en BD y la lista obtenida de varios elementos y iata exista en la BD **(14)**	icao exista en BD y la lista obtenida sea de varios elementos y iata no exista **(15)**

Suponiendo que:

- **patch_1** = {iata="GKA", icao="AYGA" …} y obtenga una lista con un elemento
- **patch_2** = {iata="GKA", icao="NotExists" …} y obtenga una lista con un elemento
- **patch_3** = {iata="GKA", icao="NotExists" …} y obtenga una lista con varios elementos
- **patch_4** = {iata="NotExists", icao="AYGA" …} y obtenga una lista con un elemento
- **patch_5** = {iata="NotExists", icao="AYGA" …} y obtenga una lista con varios elementos
- **patch_6** = {iata="NotExists", icao="NotExists" …}
- **patch_7** = {iata="GKA" } y obtenga una lista con un elemento
- **patch_8** = {iata="GKA"…} y obtenga una lista con varios elementos
- **patch_9** = {icao="AYGA" } y obtenga una lista con un elemento
- **patch_10** = {icao="AYGA"…} y obtenga una lista con varios elementos
- **patch_11** = {iata=null …}
- **patch_12** = {icao=null …}
- **patch_13** = {iata="EAS"…}
- **patch_14** = {… icao="LESO"…}
- **patch_15** = {iata="EAS", icao="LESO"…}

Entrada	Equivalencia	Resultado
patch_1	1, 3, 8, 12	Devuelve el *Airport* actualizado
patch_2	1, 6, 8	Devuelve el *Airport* actualizado
patch_3	1, 6, 11	Null
patch_4	3, 5, 12	Devuelve el *Airport* actualizado
patch_5	3, 5, 15	Null
patch_6	5, 6, 9, 13	Null
patch_7	1, 8	Devuelve el *Airport* actualizado
patch_8	1, 11	Null
patch_9	3, 12	Devuelve el *Airport* actualizado
patch_10	3, 15	Null
patch_11	4	Null
patch_12	7	Null
patch_13	9	Null
patch_14	13	Null
patch_15	9, 13	Null
null	2	Null

En el siguiente método, a la hora de realizar el borrado de un objeto, se han implementado dos tipos de borrado: borrado lógico en el cual simplemente se le da el valor de *deleted* como *true*, y por otra parte, borrado permanente en el cual el registro entero se borra por completo de la Base de Datos. Para testear este caso, en el caso de borrado lógico se testea que se realiza una actualización, y en el otro caso, se testea que el proceso de borrar no genera ningún fallo. Para saber si funciona realmente se borra, seria testear el método de borrar de la capa repositorio que después se van a mostrar, si testeásemos también que ese borrado se efectúa, dejarían de ser test unitarios:

Borrado de un aeropuerto

```
public void delete(Long id, Boolean permanent) throws NotFoundException
```

Entrada	Adecuado	No Adecuado
Long	*Id*!=null **(1)**	*Id*==null **(2)**
	Id existente **(3)**	*Id* no existente **(4)**
Boolean	*delete*=true **(5)**	*delete*==null **(7)**
	delete=false **(6)**	

Suponiendo que:

- **id=1** -> es un elemento que si existe.
- **id=20** -> es un elemento que no existe.

Entrada	Equivalencia	Resultado
1, true	1, 3, 5	No devuelve nada y se realiza el borrado
1, false	1, 3, 6	No devuelve nada y se realiza el borrado*
1, null	1, 3, 7	No devuelve nada y se realiza el borrado*
20, true	1, 4, 5	NotFoundException
20, false	1, 4, 6	NotFoundException
20, null	1, 4, 7	NotFoundException
null, true	2, 5	IllegalArgumentException
null, false	2, 6	IllegalArgumentException
null, null	2, 7	IllegalArgumentException

*En los casos de false y null se realiza un borrado lógico y en el caso de true, un borrado permanente.

Borrado un aeropuerto desde archivo

public List<Airport> deleteFromFile(String code1, String code2)

Entrada	Adecuado	No Adecuado
String	code1!=null **(1)**	code1==null **(2)**
	code1 exista **(3)**	code2 no exista **(4)**
String	code2!=null **(5)**	code2==null **(6)**
	code2 exista **(7)**	code2 no exista **(8)**

Suponiendo que:

- **UAK** -> Este código Iata no existe.
- **BGBW** -> Este código Icao no existe.
- **GKA** -> Este código Iata existe.
- **AYGA** -> Este código Icao existe.

Entrada	Equivalencia	Resultado
GKA,AYGA	1, 3, 5, 7	**Devuelve el objeto eliminado**
GKA,BGBW	1, 3, 5, 8	Devuelve un objeto vacío
UAK,BGBW	1, 4, 5, 8	Devuelve un objeto vacío
UAK,AYGA	1, 4, 5, 7	Devuelve un objeto vacío
AYGA, null	1, 3, 5, 6	**Devuelve el objeto eliminado**
BGBW, null	1, 4, 5, 6	Devuelve un objeto vacío
null, null	2, 6	Devuelve un objeto vacío
null, UAK	2, 5, 8	Devuelve un objeto vacío
null, BGBW	2, 5, 8	Devuelve un objeto vacío
null, GKA	2, 5, 7	Devuelve un objeto vacío
null, AYGA	2, 5, 7	Devuelve un objeto vacío

Aerolíneas

Guardar aerolínea tras realizar la carga desde archivo

`public Airline charge(Airline airline)`

Entrada	Adecuado	No Adecuado
Airline	Iata no existe **(1)**	Iata existe en una Airline activa **(4)**
	Iata==null **(2)**	
	Iata existe de una Airline inactiva **(3)**	
	Icao no existe **(5)**	Icao existe en una Airline activa **(8)**
	Icao==null **(6)**	
	Icao existe de una Airline inactiva **(7)**	
		Airline==null **(9)**

Suponiendo que:

- **airline_0** = {iata= "MG", icao="MDG" …}
- **airline_1** = {iata= "MG", icao="AAL" …}
- **airline_2** = {iata= "MG", icao= null …}
- **airline_3** = {iata= "AA", icao=" MDG" …}
- **airline_4** = {iata= "AA", icao=" AAL" …}
- **airline_5** = {iata= "AA", icao= null …}
- **airline_6** = {iata= null, icao=" MDG" …}
- **airline_7** = {iata= null, icao=" AAL " …}
- **airline_8** = {iata= null, icao= null …}

Entrada	Equivalencia	Resultado
airline_0	1/3, 5	Devuelve el objeto guardado
airline_1	1/3, 8	null
airline_2	1/3, 6	Devuelve el objeto guardado
airline_3	4, 5	null
airline_4	4, 8	null
airline_5	4, 6	null
airline_6	2, 5	Devuelve el objeto guardado
airline_7	2, 8	null
airline_8	2, 6	Devuelve el objeto guardado
null	9	null

Actualización parcial de una aerolínea

public Airline updatePartial(AirlinePatch patch) throws IOException

Entrada	Adecuado	No Adecuado
AirlinePatch	iata sea igual a algún valor **(1)**	AirlinePatch == null **(2)**
	icao sea igual a algún valor **(3)**	iata == null **(4)**
	iata == "NotExists" **(5)**	icao == null **(7)**
	icao == "NotExists" **(6)**	
	iata exista en BD y la lista obtenida solo sea de un elemento **(8)**	iata no exista en BD **(9)**
	iata exista en BD y la lista obtenida de varios elementos y icao exista en la BD **(10)**	iata exista en BD y la lista obtenida sea de varios elementos y icao no exista **(11)**
	icao exista en BD y la lista obtenida solo sea de un elemento **(12)**	icao no exista en BD **(13)**
	icao exista en BD y la lista obtenida de varios elementos y iata exista en la BD **(14)**	icao exista en BD y la lista obtenida sea de varios elementos y iata no exista **(15)**

Suponiendo que:

- **patch_1** = {iata="AA", icao="AAL" …} y obtenga una lista con un elemento.
- **patch_2** = {iata="AA", icao="NotExists" …} y obtenga una lista con un elemento.
- **patch_3** = {iata="AA", icao="NotExists" …} y obtenga una lista con varios elementos.
- **patch_4** = {iata="NotExists", icao="AAL" …} y obtenga una lista con un elemento.
- **patch_5** = {iata="NotExists", icao="AAL" …} y obtenga una lista con varios elementos.
- **patch_6** = {iata="NotExists", icao="NotExists" …}
- **patch_7** = {iata="AA" } y obtenga una lista con un elemento
- **patch_8** = {iata="AA"…} y obtenga una lista con varios elementos
- **patch_9** = {icao="AAL" } y obtenga una lista con un elemento
- **patch_10** = {icao="AAL"…} y obtenga una lista con varios elementos
- **patch_11** = {iata=null …}
- **patch_12** = {icao=null …}
- **patch_13** = {iata="MG"…}
- **patch_14** = {… icao="MDG"…}
- **patch_15** = {iata="MG", icao="MDG"…}

Entrada	Equivalencia	Resultado
patch_1	1, 3, 8, 12	Devuelve el *Airline* actualizado
patch_2	1, 6, 8	Devuelve el *Airline* actualizado
patch_3	1, 6, 11	Null

patch_4	3, 5, 12	Devuelve el *Airline* actualizado
patch_5	3, 5, 15	Null
patch_6	5, 6, 9, 13	Null
patch_7	1, 8	Devuelve el *Airline* actualizado
patch_8	1, 11	Null
patch_9	3, 12	Devuelve el *Airline* actualizado
patch_10	3, 15	Null
patch_11	4	Null
patch_12	7	Null
patch_13	9	Null
patch_14	13	Null
patch_15	9, 13	Null
null	2	Null

El caso de **borrado** siguiente se rige por lo mismo explicado anteriormente en *aeropuertos*.

Borrado de una aerolínea

`public void delete(Long id, Boolean permanent) throws NotFoundException`

Entrada	Adecuado	No Adecuado
Long	*Id*!=null **(1)**	*Id*==null **(2)**
	Id existente **(3)**	*Id* no existente **(4)**
Boolean	*delete*=true **(5)**	*delete*==null **(7)**
	delete=false **(6)**	

Suponiendo que:

- **id=1** -> es un elemento que si existe.
- **id=20** -> es un elemento que no existe.

Entrada	Equivalencia	Resultado
1, true	1, 3, 5	No devuelve nada y se realiza el borrado
1, false	1, 3, 6	No devuelve nada y se realiza el borrado*
1, null	1, 3, 7	No devuelve nada y se realiza el borrado*
20, true	1, 4, 5	NotFoundException
20, false	1, 4, 6	NotFoundException
20, null	1, 4, 7	NotFoundException
null, true	2, 5	IllegalArgumentException
null, false	2, 6	IllegalArgumentException
null, null	2, 7	IllegalArgumentException

*En los casos de false y null se realiza un borrado lógico y en el caso de true, un borrado permanente.

Borrar aerolínea desde archivo

```
public List<Airline> deleteFromFile(String code1, String code2)
```

Entrada	Adecuado	No Adecuado
String	code1!=null **(1)**	code1==null **(2)**
	code1 exista **(3)**	code2 no exista **(4)**
String	code2!=null **(5)**	code2==null **(6)**
	code2 exista **(7)**	code2 no exista **(8)**

Suponiendo que:

- **UX** -> Este código Iata no existe.
- **AEA** -> Este código Icao no existe.
- **MG** -> Este código Iata existe.
- **MDG** -> Este código Icao existe.

Entrada	Equivalencia	Resultado
MG,MDG	1, 3, 5, 7	**Devuelve el objeto eliminado**
MG,AEA	1, 3, 5, 8	Devuelve un objeto vacío
UX,AEA	1, 4, 5, 8	Devuelve un objeto vacío
UX,MDG	1, 4, 5, 7	Devuelve un objeto vacío
MDG, null	1, 3, 5, 6	**Devuelve el objeto eliminado**
AEA, null	1, 4, 5, 6	Devuelve un objeto vacío
null, null	2, 6	Devuelve un objeto vacío
null, UX	2, 5, 8	Devuelve un objeto vacío
null, AEA	2, 5, 8	Devuelve un objeto vacío
null, MG	2, 5, 7	Devuelve un objeto vacío
null, MDG	2, 5, 7	Devuelve un objeto vacío

Aeronaves

Guardar o actualizar una aeronave

```
public Aircraft save(Aircraft aircraft) throws NotFoundException,
DuplicatedConstraintException
```

Entrada	Adecuado	No Adecuado
Aircraft (creando un nuevo registro)	Aircraft != null **(1)**	Aircraft ==null **(2)**
	Aircraft con iata e icao inexistentes **(3)**	Aircraft con iata o icao existentes **(4)**
Aircraft (actualizando un objeto existente)	Aircraft!=null **(5)**	Aircraft==null **(6)**
	Aircraft actualización de iata o icao inexistentes **(7)**	Aircraft actualización de iata o icao existentes **(8)**
	Aircraft actualización del resto de atributos **(9)**	

Suponiendo que:

- **aircraft** = {iata= "A4F", icao="A124", aircraftName="Antonov AN-124 Ruslan" …}
- **aircraft_Repe** = {iata= "321", icao="A321", aircraftName="Airbus 321" …}
- **aircraft_Update** = {iata= "A70", icao="A170", aircraftName="Antonov AN-170" …}
- **aircraft_UpdateSameIata** = {iata= "A40", icao="A140", aircraftName="Antonov" …}
- **aircraft_SameIcaoDifIata** = {iata= "A80", icao="A140", aircraftName="Antonov" …}
- **aircraft_SameIcaoSameIata** = {iata= "A40", icao="A140", aircraftName="Antonov" …}

Entrada	Equivalencia	Resultado
aircraft	1, 3	Devuelve el objeto guardado
aircraft_Repe	1, 4	DuplicatedConstraintException
aircraft_Update	5, 7, 9	Devuelve el objeto guardado
aircraft_UpdateSameIata	5, 8, 9	DuplicatedConstraintException
aircraft_SameIcaoDifIata	5, 8, 9	Devuelve el objeto guardado*
aircraft_SameIcaoSameIata	5, 8, 9	DuplicatedConstraintException*
null	2, 6	NotFoundException

*Esto se debe a que puede existir una aeronave (aircraft) con que tenga el mismo código Icao que otros debido a que pueden pertenecer a un grupo y dentro del grupo diferenciarse gracias al código Iata.

Guardar aeronave tras realizar la carga desde archivo

public Aircraft charge(Aircraft aircraft)

Entrada	Adecuado	No Adecuado
Aircraft	Iata no existe **(1)**	Iata existe **(6)**
	Iata==null **(2)**	
	Icao no existe **(3)**	
	Icao existe **(4)**	
	Icao==null **(5)**	

Suponiendo que:

- **aircraft_0** = {iata= "321", icao="A321" …}
- **aircraft_1** = {iata= "321", icao="A140" …}
- **aircraft_2** = {iata= "321", icao= null …}
- **aircraft_3** = {iata= "A4F", icao="A321" …}
- **aircraft_4** = {iata= "A4F", icao="A140" …}
- **aircraft_5** = {iata= "A4F", icao= null …}
- **aircraft_6** = {iata= null, icao="A321" …}
- **aircraft_7** = {iata= null, icao="A140" …}
- **aircraft_8** = {iata= null, icao= null …}

Entrada	Equivalencia	Resultado
aircraft_0	3, 6	null
aircraft_1	4, 6	null
aircraft_2	5, 6	null
aircraft_3	1, 3	Devuelve el objeto guardado
aircraft_4	1, 4	Devuelve el objeto guardado
aircraft_5	1, 5	null
aircraft_6	2, 3	Devuelve el objeto guardado
aircraft_7	2, 4	Devuelve el objeto guardado
aircraft_8	2, 5	null

Borrado de una aeronave

public void delete(Long id, Boolean permanent) throws NotFoundException

Entrada	Adecuado	No Adecuado
Long	*Id*!=null **(1)**	*Id*==null **(2)**
	Id existente **(3)**	*Id* no existente **(4)**
Boolean	*delete*=true **(5)**	*delete*==null **(7)**
	delete=false **(6)**	

Suponiendo que:

- **id=1** -> es un elemento que si existe
- **id=20** -> es un elemento que no existe

Entrada	Equivalencia	Resultado
1, true	1, 3, 5	No devuelve nada y se realiza el borrado
1, false	1, 3, 6	No devuelve nada y se realiza el borrado*
1, null	1, 3, 7	No devuelve nada y se realiza el borrado*
20, true	1, 4, 5	NotFoundException
20, false	1, 4, 6	NotFoundException
20, null	1, 4, 7	NotFoundException
null, true	2, 5	IllegalArgumentException
null, false	2, 6	IllegalArgumentException
null, null	2, 7	IllegalArgumentException

*En los casos de false y null se realiza un borrado lógico y en el caso de true, un borrado permanente.

Borrado de una aeronave desde archivo

```
public List<Aircraft> deleteFromFile(String code1, String code2)
```

Entrada	Adecuado	No Adecuado
String	code1!=null **(1)**	code1==null **(2)**
	code1 exista **(3)**	code2 no exista **(4)**
String	code2!=null **(5)**	code2==null **(6)**
	code2 exista **(7)**	code2 no exista **(8)**

Suponiendo que:

- **A4F** -> Este código Iata no existe.
- **A124** -> Este código Icao no existe.
- **321** -> Este código Iata existe.
- **A321** -> Este código Icao existe.

Entrada	Equivalencia	Resultado
321,A321	1, 3, 5, 7	**Devuelve el objeto eliminado**
321,A124	1, 3, 5, 8	Devuelve un objeto vacío
A4F,A124	1, 4, 5, 8	Devuelve un objeto vacío
A4F,A321	1, 4, 5, 7	Devuelve un objeto vacío
A321,null	1, 3, 5, 6	**Devuelve el objeto eliminado**
A124,null	1, 4, 5, 6	Devuelve un objeto vacío
null, null	2, 6	Devuelve un objeto vacío
null, A4F	2, 5, 8	Devuelve un objeto vacío
null, A124	2, 5, 8	Devuelve un objeto vacío
null, 321	2, 5, 7	Devuelve un objeto vacío
null, A321	2, 5, 7	Devuelve un objeto vacío

Para continuar, vamos a seguir con la carpeta del repositorio, en este caso, sí que casi todos los métodos son iguales, solo les diferencia algún pequeño detalle sin importancia tales como el nombre. Por eso mismo, solo se van a mostrar los diseños de pruebas de aeropuertos:

Existe mediante Iata

public void testExistsByIata()

Entrada	Respuesta esperada	Resultado
repository.existsByIata("")	Falso -> no puede ser vacío	OK
repository.existsByIata(" EAS")	Falso -> no puede tener espacios por delante	OK
repository.existsByIata("EAS ")	Falso -> no puede tener espacios por detrás	OK
repository.existsByIata("EAS")	Verdadero	OK

No existe por Id pero si mediante Iata

public void testExistsByIdNotAndIata()

Entrada	Respuesta esperada	Resultado
repository.existsByIdNotAndIata(2L, "")	Falso -> No vacío	OK
repository.existsByIdNotAndIata(2L, " GKA")	Falso -> Sin espacios	OK
repository.existsByIdNotAndIata(2L, "GKA ")	Falso -> Sin espacios	OK
repository.existsByIdNotAndIata(2L, "GKA")	Falso -> No existe	OK
repository.existsByIdNotAndIata(1L, "GKA")	Verdadero	OK

Existe mediante Icao

public void testExistsByIcao()

Entrada	Respuesta esperada	Resultado
repository.existsByIcao("")	Falso -> No vacío	OK
repository.existsByIcao(" LESO")	Falso -> Sin espacios	OK
repository.existsByIcao("LESO ")	Falso -> Sin espacios	OK
repository.existsBycao ("LESO")	Verdadero	OK

No existe por Id pero si mediante Icao

public void testExistsByIdNotAndIcao()

Entrada	Respuesta esperada	Resultado
repository.existsByIdNotAndIcao(2L, "")	Falso -> No vacío	OK
repository.existsByIdNotAndIcao (2L, " AYGA")	Falso -> Sin espacios	OK
repository.existsByIdNotAndIcao (2L, "AYGA ")	Falso -> Sin espacios	OK
repository.existsByIdNotAndIcao (2L, "AYGA")	Falso -> No existe	OK
repository.existsByIdNotAndIcao (1L, "AYGA")	Verdadero	OK

Devuelve los aeropuertos utilizando una lista de Iata

public void testFindByIataIn()

Entrada	Respuesta esperada	Resultado
repository.findByIataIn(new ArrayList<>());	Devuelve una lista vacía	OK
repository.findByIataIn(Arrays.asList("MAG"));	Devuelve una lista con 1 elemento con el *Id=9L*	OK
repository.findByIataIn(Arrays.asList("MAG", "ABB", "310"));	Devuelve una lista con 3 elementos que tienen los *Id igual a 9L, 10L y 11L*	OK

Devuelve los aeropuertos utilizando una lista de Icao

public void testFindByIcaoIn()

Entrada	Respuesta esperada	Resultado
repository.findByIcaoIn(new ArrayList<>());	Devuelve una lista vacía	OK
repository.findByIcaoIn(Arrays.asList("AYMD"));	Devuelve una lista con 1 elemento con el *Id=9L*	OK
repository.findByIcaoIn(Arrays.asList("AYMD", "BIAR", "BISI"));	Devuelve una lista con 3 elementos que tienen los *Id igual a 9L, 10L y 11L*	OK

Devuelve los aeropuertos utilizando una lista de nombres

```
public void testFindByAirportNameIn()
```

Entrada	Respuesta esperada	Resultado
repository.findByAirportNameIn(new ArrayList<>());	Devuelve una lista vacía	OK
repository.findByAirportNameIn(Arrays.asList("Thula Airport"));	Devuelve una lista con 1 elemento con el *Id=8L*	OK
repository.findByAirportNameIn(Arrays.asList("Thula Airport", "Nadzab Airport"));	Devuelve una lista con 2 elementos que tienen los *Id igual a 8L y 9L*	OK

Encuentra los aeropuertos que contengan en su nombre la palabra que se pasa

```
public void testFindByAirportNameContaining()
```

Entrada	Respuesta esperada	Resultado
repository.findByAirportNameContaining("Thula");	Devuelve una lista con 1 elemento el cual su *AirportName= 'Thula Airport'*	OK

Encuentra todos

```
public void testFindAll()
```

Entrada	Respuesta esperada	Resultado
repository.findAll();	Devuelve una colección de elementos **Airport** el cual su tamaño es de 15	OK

Encuentra los aeropuertos utilizando una lista de códigos ISO

```
public void testFindByCountryIsoCode ()
```

Entrada	Respuesta esperada	Resultado
repository.findByCountryIsoCode (new ArrayList<>())	Devuelve lista vacía	OK
repository.findByCountryIsoCode (Arrays.asList("TT"))	Devuelve lista vacía	OK
repository.findByCountryIsoCode (Arrays.asList("ES"))	Devuelve lista con **1** elemento	OK
repository.findByCountryIsoCode (Arrays.asList("TT",”ES”,”PG”))	Devuelve lista con **2** elementos	OK
repository.findByCountryIsoCode (null)	Devuelve lista vacía	OK

Encuentra los aeropuertos no dados de baja

public void testFindDeletedFalse ()

Entrada	Respuesta esperada	Resultado
repository.findDeletedFalse ()	Devuelve una lista con 10	OK

Para ir finalizando, vamos a ver la carpeta de *Factory* que engloba las clases intermediarias entre las capas principales de la API. Estas clases se usan para transformar los datos recibidos, por ejemplo, de un *Data Transfer Object (DTO)* a una entidad existente. Al fin y al cabo, son trasformaciones muy similares, por ello solo se van a mostrar las de aeropuertos:

Factory

Transformación de *DTO de aeropuertos* a *Airport*

public Airport getInstance(AirportDTO airportdto)

Entrada	Adecuado	No Adecuado
AirportDTO	airportdto!=null **(1)**	airportdto==null **(2)**
	iata cumple requisitos **(3)**	iata no cumple requisitos **(5)**
	iata = "" o null **(4)**	
	icao cumple requisitos **(6)**	icao no cumple requisitos **(8)**
	icao = "" o null **(7)**	
	countryIsoCode = ""o null **(9)**	countryIsoCode una o más de dos letras **(11)**
	countryIsoCode 2 letras **(10)**	
	timezone = null**(12)**	timezone no cumple con el formato predefinido para este atributo**(16)**
	timezone = utc **(13)**	
	timezone = utc-10,0 **(14)**	
	timezone = utc+10,0 **(15)**	

Suponiendo que:

- **airportdto** = {iata: "GKA", Icao: "AYGA", airportName: "Goroka Airport", countryIsoCode: "PG", timezone: utc, dst: false, longitude: -6.09169893346, latitude: 145.3919983, deleted: false}
- **dto_NoIata_Time+** = {iata: "",… timezone: utc+10,0,…}
- **dto_NoIata_Time+_2** = {iata: null,… timezone: utc+10,0,…}
- **dto_NoIcao_Time-** = {… Icao: "",… timezone: utc-10,0,…}
- **dto_NoIcao_Time-_2**= {… Icao: null,… timezone: utc-10,0, …}
- **dto_NoIso_Time0** = {… countryIsoCode: "", timezone: utc, …}
- **dto_NoIso_Time0_2** = {… countryIsoCode: null, timezone: null, …}
- **AIRPORT_1**= {iata: "GKA", Icao: "AYGA", airportName: "Goroka Airport", countryIsoCode: "PG", timezone: utc+10,0, dst: false, longitude: -6.09169893346, latitude: 145.3919983, deleted: false}
- **AIRPORT_2** = {iata: null, Icao: "AYGA", airportName: "Goroka Airport", countryIsoCode: "PG", timezone: utc+10,0, dst: false, longitude: -6.09169893346, latitude: 145.3919983, deleted: false}
- **AIRPORT_3** = {iata: "GKA", Icao: null, airportName: "Goroka Airport", countryIsoCode: "PG", timezone: utc-10,0, dst: false, longitude: -6.09169893346, latitude: 145.3919983, deleted: false}
- **AIRPORT_4**= {iata: "GKA", Icao: "AYGA", airportName: "Goroka Airport", countryIsoCode: null, timezone: utc, dst: false, longitude: -6.09169893346, latitude: 145.3919983, deleted: false}

Entrada	Equivalencia	Resultado
airportdto	1, 3, 5, 9, 12	AIRPORT_1
dto_NoIata_Time+	1, 4, 5, 9, 14	AIRPORT_2
dto_NoIata_Time+_2	1, 4, 5, 9, 14	AIRPORT_2
dto_NoIcao_Time-	1, 3, 6, 9, 13	AIRPORT_3
dto_NoIcao_Time-_2	1, 3, 6, 9, 13	AIRPORT_3
dto_NoIso_Time0	1, 3, 5, 8, 11	AIRPORT_4
dto_NoIso_Time0	1, 3, 5, 8, 11	AIRPORT_4
null	2	ConstraintViolationException

Transformación de *DTO de aeropuertos* e *Id* a *Airport*

public Airport getInstance(AirportDTO airportdto, Long id)

Entrada	Adecuado	No Adecuado
Id	Id!=null **(1)**	Id==null **(2)**
AirportDTO	airportdto!=null **(3)**	airportdto==null **(4)**
	iata cumple requisitos **(5)**	iata no cumple requisitos **(7)**
	iata = "" o NULL **(6)**	
	icao cumple requisitos **(8)**	icao no cumple requisitos **(10)**
	icao = "" o NULL 9**()**	
	countryIsoCode = NULL **(11)**	countryIsoCode una o más de dos letras **(13)**
	countryIsoCode 2 letras **(12)**	
	timezone = null **(14)**	timezone no cumple con el formato predefinido para este atributo **(18)**
	timezone = utc **(15)**	
	timezone = utc -10 **(16)**	
	timezone = utc +10 **(17)**	

Suponiendo que:

- **airportdto** = {iata: "GKA", Icao: "AYGA", airportName: "Goroka Airport", countryIsoCode: PG, timezone: utc+10,0, dst: false, longitude: -6.09169893346, latitude: 145.3919983, deleted: false}
- **dto_NoIata_Time+** = {iata: "", … timezone: utc +10, …}
- **dto_NoIata_Time+_2** = {iata: null, … timezone: utc +10, …}
- **dto_NoIcao_Time-** = {… Icao: "", … timezone: utc -10, …}
- **dto_NoIcao_Time-_2**= {… Icao: null, … timezone: utc -10, …}
- **dto_NoIso_Time0** = {… countryIsoCode: "", timezone: utc, …}
- **dto_NoIso_Time0_2** = {… countryIsoCode: null, timezone: null, …}
- **AIRPORT_1**= { id: 1, iata: "GKA", Icao: "AYGA", airportName: "Goroka Airport", countryIsoCode: "PG", timezone: utc+10,0, dst: false, longitude: -6.09169893346, latitude: 145.3919983, deleted: false}
- **AIRPORT_2** = { id: 1, iata: null, Icao: "AYGA", airportName: "Goroka Airport", countryIsoCode: "PG", timezone: utc+10,0, dst: false, longitude: -6.09169893346, latitude: 145.3919983, deleted: false}
- **AIRPORT_3** = { id: 1, iata: "GKA", Icao: null, airportName: "Goroka Airport", countryIsoCode: "PG", timezone: utc-10,0, dst: false, longitude: -6.09169893346, latitude: 145.3919983, deleted: false}
- **AIRPORT_4**= { id: 1, iata: "GKA", Icao: "AYGA", airportName: "Goroka Airport", countryIsoCode: null, timezone: utc, dst: false, longitude: -6.09169893346, latitude: 145.3919983, deleted: false}

Entrada	Equivalencia	Resultado
airportdto, 1	1, 3, 5, 8, 12, 15	AIRPORT_1
dto_NoIata_Time+, 1	1, 3, 6, 8, 12, 17	AIRPORT_2
dto_NoIata_Time+_2, 1	1, 3, 6, 8, 12, 17	AIRPORT_2
dto_NoIcao_Time-, 1	1, 3, 5, 9, 12, 16	AIRPORT_3
dto_NoIcao_Time-_2, 1	1, 3, 5, 9, 12, 16	AIRPORT_3
dto_NoIso_Time0, 1	1, 3, 5, 8, 11, 14	AIRPORT_4
dto_NoIso_Time0, 1	1, 3, 5, 8, 11, 14	AIRPORT_4
null, 1	1, 4	ConstraintViolationException
airportdto, null	2, 3	ConstraintViolationException

Transformación de *Airport* a *DTO de aeropuertos*

public AirportDTO getInstance(Airport airport)

Entrada	Adecuado	No Adecuado
Airport	airport!=null **(1)**	airport==null **(2)**
	iata cumple requisitos **(3)**	iata no cumple requisitos **(5)**
	iata = "" o null **(4)**	
	icao cumple requisitos **(6)**	icao no cumple requisitos **(8)**
	icao = "" o null **(7)**	
	countryIsoCode = ""o null **(9)**	countryIsoCode una o más de dos letras **(11)**
	countryIsoCode 2 letras **(10)**	
	timezone = null **(12)**	timezone no cumple con el formato predefinido para este atributo **(16)**
	timezone = utc **(13)**	
	timezone = utc -10 **(14)**	
	timezone = utc +10 **(15)**	

Suponiendo que:

- **airport** = {iata: "GKA", Icao: "AYGA", airportName: "Goroka Airport", countryIsoCode: "PG", timezone: utc+10,0, dst: false, longitude: -6.09169893346, latitude: 145.3919983, deleted: false}
- **airport _NoIata_Time+** = {iata: "", … timezone: utc utc +10, …}
- **airport _NoIata_Time+_2** = {iata: null, … timezone: utc +10, …}
- **airport _NoIcao_Time-** = {… Icao: "", … timezone: utc -10, …}
- **airport _NoIcao_Time- _2**= {… Icao: null, … timezone: utc -10, …}
- **airport _NoIso_Time0** = {… countryIsoCode: "", timezone: utc, …}
- **airport _NoIso_Time0_2** = {… countryIsoCode: null, timezone: null, …}
- **DTO_1**= {iata: "GKA", Icao: "AYGA", airportName: "Goroka Airport", countryIsoCode: "PG", timezone: utc+10,0, dst: false, longitude: -6.09169893346, latitude: 145.3919983, deleted: false}
- **DTO _2** = {iata: null, Icao: "AYGA", airportName: "Goroka Airport", countryIsoCode: "PG", timezone: utc+10,0, dst: false, longitude: -6.09169893346, latitude: 145.3919983, deleted: false}
- **DTO _3** = {iata: "GKA", Icao: null, airportName: "Goroka Airport", countryIsoCode: "PG", timezone: utc-10,0, dst: false, longitude: -6.09169893346, latitude: 145.3919983, deleted: false}
- **DTO _4**= {iata: "GKA", Icao: "AYGA", airportName: "Goroka Airport", countryIsoCode: null, timezone: utc, dst: false, longitude: -6.09169893346, latitude: 145.3919983, deleted: false}

Entrada	Equivalencia	Resultado
airport	1, 3, 5, 9, 12	DTO_1
airport _NoIata_Time+	1, 4, 5, 9, 14	DTO _2
airport _NoIata_Time+_2	1, 4, 5, 9, 14	DTO _2
airport _NoIcao_Time-	1, 3, 6, 9, 13	DTO _3
airport_NoIcao_Time-_2	1, 3, 6, 9, 13	DTO _3
airport _NoIso_Time0	1, 3, 5, 8, 11	DTO _4
airport _NoIso_Time0	1, 3, 5, 8, 11	DTO _4
null	2	ConstraintViolationException

ParserFactory

Transformación de archivo a una lista de aeropuertos

`public List<Airport> getInstance(File file) throws MalformedCSVException, IOException`

Entrada	Adecuado	No Adecuado
File	file!=null **(1)**	file==null **(2)**
	iata cumple requisitos **(3)**	iata no cumple requisitos **(5)**
	iata = "" o null **(4)**	
	icao cumple requisitos **(6)**	icao no cumple requisitos **(8)**
	icao = "" o null **(7)**	
	countryIsoCode = ""o null **(9)**	countryIsoCode una o más de dos letras **(11)**
	countryIsoCode 2 letras **(10)**	
	timezone = null **(12)**	timezone distinto al formato predefinido en las características previas de archivos**(16)**
	timezone = utc **(13)**	
	timezone = utc-10.0 **(14)**	
	timezone = utc+10.0 **(15)**	

Suponiendo que:

- **file** = es un archivo .csv y cada atributo de cada objeto de la lista cumple con los requisitos establecidos
- **fileWithNulls** = dentro del archivo .csv, uno o varios objetos tienen los atributos *iata, icao, countryIsoCode y timezone* que tienen los valores null.
- **fileWithEmptys** = dentro del archivo .csv, uno o varios objetos tienen los atributos *iata, icao, countryIsoCode y timezone* que tienen los valores vacíos ("").
- **fileBAD** = dentro del archivo .csv, uno o varios objetos tienen los atributos *iata, icao, countryIsoCode y timezone* que tienen los valores que NO cumplen los requisitos establecidos.
- **fileWithDifferentTimezones** = dentro del archivo .csv, existen varios objetos que tienen diferentes valores para *timezone*: null, utc, utc-10,0 y utc+10,0.
- **fileBADTimezones** = dentro del archivo .csv, un objeto tiene el atributo *timezone* que NO cumple con los requisitos establecidos.

Entrada	Equivalencia	Resultado
file	1, 3, 6, 10	Lista de *Airport*
fileWithNulls	1, 4, 7, 9, 12	Lista de *Airport*
fileWithEmptys	1, 4, 7, 9, 12	Lista de *Airport*
fileWithDifferentTimezones	1, 12, 13, 14, 15	Lista de *Airport*
fileBAD	1, 5, 8, 11	ConstraintViolationException NumberFormatException
fileBADTimezones	1, 16	ConstraintViolationException
null	2	ConstraintViolationException

Conseguir códigos desde un archivo (Para borrado desde archivo)

```
public List<String[]> getCodes(File file) throws MalformedCSVException
```

Entrada	Adecuado	No Adecuado
File	file!=null **(1)**	file==null **(2)**
	2 palabras por línea **(3)**	Línea vacía o más de dos palabras **(5)**
	1 palabra por línea **(4)**	

Suponiendo que:

- **file2word =** es un archivo .csv con 2 palabras por líneas.
- **file1word =** es un archivo .csv con 1 palabra por líneas.
- **fileEmpty =** es un archivo .csv con líneas vacías.
- **fileExcess =** es un archivo .csv con más de 2 palabras por línea.

Entrada	Adecuado	No Adecuado
file2word	1, 3	Lista con objetos con 2 String por objeto
file1word	1, 4	Lista de objetos con 1 String por objeto
fileEmpty	1, 5	Lista de objetos vacíos
fileExcess	1, 5	Lista de objetos vacíos
null	2	MalformedCSVException

PatchFactory

Conseguir clase de actualización auxiliar desde un mapa de claves y valores

```
public AirportPatch getInstance(Map<String, Object> map)
```

Entrada	Adecuado	No Adecuado
Map<String, Object>	map contiene la palabra *iata* **(1)**	map == null **(5)**
	map contiene la palabra *icao* **(2)**	
	map NO contiene la palabra *iata* **(3)**	
	map NO contiene la palabra *icao* **(4)**	

Suponiendo que:

- **map =** [iata: GKA, Icao: AYGA …]
- **mapIata =** [Icao: AYGA …]
- **mapIcao =** [iata: GKA …]
- **mapEmpty =** […]
- **AirportPatch =** {GKA, AYGA, …}
- **AirportPatchIata =** {"", AYGA,…}
- **AirportPatchIcao =** {GKA, "", …}
- **AirportPatchEmtpy =** {"", "", ,…}

*… significa que existen otros atributos los cuales no se comprueban en este punto.

Entrada	Equivalencia	Resultado
map	1, 2	AirportPatch
mapIata	2, 3	AirportPatchIata
mapIcao	1, 4	AirportPatchIcao
mapEmpty	3, 4	AirportPatchEmpty
null	5	null

MapFactory

Conseguir lista de mapa de claves y valores desde archivo (Actualización parcial desde archivo)

```
public List<Map<String, Object>> getInstance (File file) throws MalformedCSVException, IOException
```

Entrada	Adecuado	No Adecuado
File	file!=null **(1)**	file==null **(2)**
	file cumple con los requisitos de estructura **(3)**	file **no** cumple con los requisitos de estructura **(4)**

Suponiendo que:

- **file** = un archivo .csv que cumple con los requisitos de estructura predeterminados (iata: YAZ,icao:TTTT,countryIsoCode: ES,longitude:49.079833).
- **badFile**= es un archivo .csv pero no cumple con los requisitos establecidos (YBB,ZZDS,Kugaaruk Airport,CA,-7,true,68.534401,-89.808098 → Faltan datos).
- **listmapOfFile =** es una lista de objetos tipo *Map<String, Object>* obtenidos como resultado del proceso [{iata: YAZ, icao:TTTT,countryIsoCode: ES,longitude:49.079833},{…}, {…}].

Entrada	Equivalencia	Resultado
file	1, 3	listmapOfFile
badFile	1, 4	MalformedCSVException
null	2	MalformedCSVException

Para finalizar, queda por mostrar el diseño de caso de pruebas de la carpeta *Helpers* que de momento solo consta de una clase que valida los mapas de claves y valores (*Map<String, Object>*) con la clase indicada:

MapValidatorHelper

Valida que el mapa de claves y valores hace referencia a la clase indicada

`public <T> Boolean validate(Map<String, Object> map, Class<T> clase)`

Entrada	Adecuado	No Adecuado
Map<String, Object>	map contiene atributos que existen en la Class<T> **(1)**	map NO contiene atributos que existen en la Class<T> **(2)**
		map==null **(3)**
Class<T>	clase!=null **(4)**	clase==null **(5)**
	clase existe **(6)**	clase NO existe **(7)**

Suponiendo que:

- **map** = contiene atributos que existen [iata: GKA, icao: AYGA, airportName: Goroka Airport].
- **mapBad** = contiene atributos que NO existen [iata: GKA, icao: AYGA, badAttribute: fail].

Entrada	Equivalencia	Resultado
map, Airport	1, 4, 6	True
mapBad, Airport	2, 4, 6	False
null, Airport	3, 4, 6	False
map, FakeClass	1, 4, 7	False
mapBad, FakeClass	2, 4, 7	False
null, null	3, 5	False

F) Proceso de test de interfaz de usuario

Para el desarrollo de los test de interfaz de usuario, antes de nada, para facilitar las tareas de identificación de los componentes, en el desarrollo de la interfaz, se han identificado las clases de los componentes que se van a usar con nombres accesibles y de manera estándar previamente estudiado para ahorrar el trabajo a posterior, es decir, en la identificación de la clase se añade la palabra ***qa*** y separado mediante guiones el identificador específico para cada componente. Para mayor comprensión mejor, vamos a explicarlo mediante un ejemplo real:

Mostrar modal con datos de airport

```
…
And pulso el botón "edit" de la ventana de "airport"
…
```

Como podemos observar en los pasos de arriba, *edit* y *airport* son palabras para la identificación de un componente en concreto, y gracias al previo estudio realizado, ese mismo paso podríamos reutilizarlo, ya que para la identificación de la modal *edit*, se identifica mediante la clase ***qa-edit-modal***, en el caso de añadir, seria ***qa-add-modal*** y en caso de borrar, ***qa-delete-modal***. Como podemos ver, gracias a este simple paso, a la hora de realizar la búsqueda desde los test, se realiza de manera más sencilla y lógica, lo que ahorra cierta cantidad de tiempo en el desarrollo, sobre todo, cuando se pueden reutilizar los pasos.

Una vez claro el proceso de identificación, para empezar a desarrollar los test, lo primero es ver que opciones pueden existir y como debería de actuar en cada caso. Para el caso general de la edición de un aeropuerto, nos encontramos con varios casos y cada uno de ellos se debe de testear:

- Mostrar modal con datos de Airport
- Comprobación de campo obligatorio "<field>" al pulsar el botón "Accept"
- Comprobación de respuestas para datos erróneos
- Edición correcta de Airport con cambios en <field>

Ha de constar que *<field>* seria cada input de la modal.

Lo primero que se debe hacer el crear un *Background*, ya que la pagina que vamos testear tiene una base común que sería (0) loguearse y acceder a la página de aeropuertos. En el caso **Mostrar modal con datos de Airport**, (1) deberíamos de seleccionar una fila previamente insertada, tras seleccionar la fila, (2) deberíamos de pulsar el botón de editar. Para finalizar deberíamos de asegurarnos que (3) se ha abierto la modal del aeropuerto seleccionado y con (4) los datos de la fila anteriormente seleccionada.

Tras definir cuáles son los pasos a seguir, lo único que quedaría seria esos pasos plasmarlos en el lenguaje de *Gherkin* y ejecutarlos:

```
(0) Background: Airport
        Given que estoy logueado con usuario "ikiata" y contrasena "ikiata"
        And que estoy en la pagina de "airport"

(1) Then selecciono la fila con el dataset "airport_edit" en la tabla "airport"
    de la pantalla "airport"
(2) And pulso el botón "edit" de la ventana de "airport"
(3) And aparece la ventana modal "edit" de "airport"
(4) And me muestra los datos de la fila seleccionada de "airport"
```

G) Test de interfaz de usuario

Para empezar vamos a ver test relacionados con el acceso y navegación por la interfaz:

COMO usuario QUIERO acceder a la pantalla principal de IK-IATA-SPA

Escenario de prueba	Descripción	Resultado
Se permite el login con datos correctos	When inserto el usuario "ikiata" And inserto la contraseña "ikiata" And pulso el botón de login Then se muestra la página de inicio de la aplicación	PASA

COMO coordinador QUIERO navegar por la aplicación PARA poder acceder a todos los menús

Escenario de prueba	Descripción	Resultado
Navegar por las páginas existentes	When se accede a la página "<page>" desde el menú de opciones When navego a la página "<page>" desde el menú de opciones Then se muestra la página de "<page>"	PASA

Para continuar con los test, vamos a ver los relacionados con las interfaces de aeropuertos, aerolíneas y aeronaves, aunque por su similitud en las interfaces, solo se van a mostrar las de aeropuertos:

COMO coordinador QUIERO ver los aeropuertos existentes

Escenario de prueba	Descripción	Resultado
Ver tabla de Airport vacía	When selecciona la opción airport y no existen aeropuertos en la BBDD IATA-DB Then la tabla de resultados de "airport" está vacía	PASA
Ver tabla de Airport con datos	When selecciona la opción airport y existen aeropuertos en la BBDD IATA-DB Then la tabla de resultados de "airport" no está vacía	PASA

COMO coordinador QUIERO modificar los datos de un aeropuerto

Escenario de prueba	Descripción	Resultado
Mostrar modal con datos de Airport	Then selecciono la fila con el dataset "airport_edit" en la tabla "airport" de la pantalla "airport" And pulso el botón "edit" de la ventana de "airport" And aparece la ventana modal "edit" de "airport" And me muestra los datos de la fila seleccionada de "airport"	PASA
Comprobación de campo obligatorio "<field>" al pulsar el botón "Accept"	When estoy en la modal "edit" de "airport" con los datos del dataset "airport_edit" elegidos en la tabla "airport" And completo el campo "<field>" con el valor "" en la modal "edit" de la pantalla "airport" And pulso el botón "accept" en la modal "edit" de "airport" And se muestra un mensaje de campo obligatorio And el campo obligatorio "<required_field_class>" se muestra en rojo	PASA
Comprobación de respuestas para datos erróneos	When estoy en la modal "edit" de "airport" con los datos del dataset "airport_edit" elegidos en la tabla "airport" And se rellenan los campos de "airport" con los datos "airport_iata_icao_error_value" And pulso el botón "accept" en la modal "edit" de "airport" Then se muestra un mensaje de error	PASA
Edición correcta de Airport con cambios en <field>	When estoy en la modal "edit" de "airport" con los datos del dataset "airport_edited" elegidos en la tabla "airport" And completo el campo "<field>" con el valor "airport_edited" en la modal "edit" de la pantalla "airport" And pulso el botón "accept" en la modal "edit" de "airport" Then se muestra un mensaje de correcto And el campo "<field>" de la tabla "airport" en la pantalla "airport" tiene los datos del dataset "airport_edited" Then el tamaño máximo de caracteres del campo "<input_field>" de la modal "add" de "airport" es "<number>" y sub-cadena de "airport_error_size"	PASA
Comprobación de respuestas para datos erróneos	When se rellenan los campos de "airport" con los datos "airport_iata_icao_error_value" And pulso el botón "accept" en la modal "add" de "airport" Then se muestra un mensaje de error	PASA

Para finalizar, se van a mostrar los test referidos a la subida de archivos, en este punto al ser también muy parecidos, solo se van a mostrar los de aeropuertos:

COMO coordinador QUIERO modificar aeropuertos desde archivo

Escenario de prueba	Descripción	Resultado
Cargar archivo csv, enviarlo y recibir respuesta	When selecciono "airport" en el input "type" y "update" en el input "action" And selecciono el archivo "updateDatos.csv" And pulso el botón "submit" Then se muestra un mensaje de update file	PASA

COMO coordinador QUIERO añadir aeropuertos desde archivo

Escenario de prueba	Descripción	Resultado
Cargar archivo csv, enviarlo y recibir respuesta	When selecciono "airport" en el input "type" y "create" en el input "action" And selecciono el archivo "datos.csv" And pulso el botón "submit" Then se muestra un mensaje de create file	PASA

COMO coordinador QUIERO borrar aeropuertos desde archivo

Escenario de prueba	Descripción	Resultado
Cargar archivo csv, enviarlo y recibir respuesta	When selecciono "airport" en el input "type" y "delete" en el input "action" And selecciono el archivo "deleteDatos.csv" And pulso el botón "submit" Then se muestra un mensaje de delete file	PASA

H) Test de integración

Pruebas de integración de *endpoints* propios de aeropuertos

Escenario de prueba	Descripción	Resultado
Check If Airport Exists By Iata Code [True]	Given path basePath + '/search/existsByIata' And param iata = global.airportIata When method get Then status 200 And match response == 'true'	Devuelve 200 y response=true
Check If Airport Exists By Iata Code [False]	Given path basePath + '/search/existsByIata' And param iata = global.stringNotFound When method get Then status 200 And match response == 'false'	Devuelve 200 y response=false
Check If Airport Exists By Icao Code [True]	Given path basePath + '/search/existsByIcao' And param icao = global.airportIcao When method get Then status 200 And match response == 'true'	Devuelve 200 y response=true
Check If Airport Exists By Icao Code [False]	Given path basePath + '/search/existsByIcao' And param icao = global.stringNotFound When method get Then status 200 And match response == 'false'	Devuelve 200 y response=false
Check If Airport Exists By Id Not And Iata Code [True]	Given path basePath + '/search/existsByIdNotAndIata' And param iata = global.airportIata And param id = global.numberNotFound When method get Then status 200 And match response == 'true	Devuelve 200 y response=true
Check If Airport Exists By Id Not And Iata Code [False]	Given path basePath + '/search/existsByIdNotAndIata' And param iata = global.airportIata And param id = global.airportId When method get Then status 200 And match response == 'false'	Devuelve 200 y response=false

Check If Airport Exists By Id Not And Icao Code [True]	Given path basePath + '/search/existsByIdNotAndIcao' And param icao = global.airportIcao And param id = global.numberNotFound When method get Then status 200 And match response == 'true'	Devuelve 200 y response=true
Check If Airport Exists By Id Not And Icao Code [False]	Given path basePath + '/search/existsByIdNotAndIcao' And param icao = global.airportIcao And param id = global.airportId When method get Then status 200 And match response == 'false'	Devuelve 200 y response=false
Find airport by Name [Found]	Given path basePath + '/search/findByAirportName' And param name = global.airportName When method get Then status 200 And match response == '#object'	Devuelve 200 y un *#object* tipo *airport*
Find Airport by Name [Not Found]	Given path basePath + '/search/findByAirportName' And param name = global.stringNotFound When method get Then status 200 And match response == '#object'	Devuelve 200 y un *#object* tipo *airport* vacío
Find Airport by Name Containing [Found]	Given path basePath + '/search/findByAirportNameContaining' And param name = global.airportNameContain When method get Then status 200 And match response == '#object'	Devuelve 200 y un *#object* tipo *airport*
Find Airport by Name Containing [Not Found]	Given path basePath + '/search/findByAirportNameContaining' And param name = global.stringNotFound When method get Then status 200 And match response == '#object'	Devuelve 200 y un *#object* tipo *airport* vacío
Find Airport by Iata Code [Found]	Given path basePath + '/search/findByIata' And param iata = global.airportIata When method get Then status 200 And match response == '#object'	Devuelve 200 y un *#object* tipo *airport*

Find Airport by Iata Code [Not Found]	Given path basePath + '/search/findByIata' And param iata = global.stringNotFound When method get Then status 200 And match response == '#object'	Devuelve 200 y un *#object* tipo *airport* vacío
Find Airport by Icao Code [Found]	Given path basePath + '/search/findByIcao' And param icao = global.airportIcao When method get Then status 200 And match response == '#object'	Devuelve 200 y un *#object* tipo *airport*
Find Airport by Icao Code [Not Found]	Given path basePath + '/search/findByIcao' And param icao = global.stringNotFound When method get Then status 200 And match response == '#object'	Devuelve 200 y un *#object* tipo *airport* vacío

Pruebas de integración de endpoints propios de aerolíneas

Escenario de prueba	Descripción	Resultado
Check If Airline Exists By Iata Code And Status True [True]	Given path basePath + '/search/existsByIataAndStatusTrue' And param iata = global.airlineIata When method get Then status 200 And match response == 'true'	Devuelve 200 y response=true
Check If Airline Exists By Iata Code And Status True [False]	Given path basePath + '/search/existsByIataAndStatusTrue' And param iata = global.stringNotFound When method get Then status 200 And match response == 'false'	Devuelve 200 y response=false
Check If Airline Exists By Icao Code And Status True [True]	Given path basePath + '/search/existsByIdNotAndIataAndStatusTrue' And param iata = global.airlineIata And param id = global.numberNotFound When method get Then status 200 And match response == 'true'	Devuelve 200 y response=true
Check If Airline Exists By Id Not And Iata Code And Status True [False]	Given path basePath + '/search/existsByIdNotAndIataAndStatusTrue' And param icao = global.airlineIata And param id = global.airlineId When method get Then status 200	Devuelve 200 y response=false

Check If Airline Exists By Id Not And Icao Code And Status True [True]	Given path basePath + '/search/existsByIdNotAndIcaoAndStatusTrue' And param icao = global.airlineIcao And param id = global.numberNotFound When method get Then status 200 And match response == 'true'	Devuelve 200 y response=true
Find Airline by Name [True]	Given path basePath + '/search/findByAirlineName' And param name = global.airlineName When method get Then status 200 And match response == '#object'	Devuelve 200 y un *#object* tipo *airline*
Find Airline by Name [False]	Given path basePath + '/search/findBydAirlineName' And param name = global.stringNotFound When method get Then status 404 And match response == ''	Devuelve 200 y response vacío
Find Airline by Status False	Given path basePath + '/search/findByStatusFalse' When method get Then status 200 And match response._embedded.airlines=='#[]'	Devuelve 200 y lista de *airlines*
Find Airline by Status True	Given path basePath + '/search/findByStatusTrue' When method get Then status 200 And match response._embedded.airlines == '#[]'	Devuelve 200 y lista de *airline*
Find Airline by Iata Code [Not Found]	Given path basePath + '/search/findByIata' And param iata = global.stringNotFound When method get Then status 200 And match response == '#object'	Devuelve 200 y *#object* tipo *airline* vacío
Find Airline by Icao Code [Found]	Given path basePath + '/search/findByIcao' And param icao = global.airlineIcao When method get Then status 200 And match response == '#object'	Devuelve 200 y *#object* tipo *airline*
Find Airline by Icao Code [Not Found]	Given path basePath + '/search/findByIcao' And param icao = global.stringNotFound When method get Then status 200	Devuelve 200 y *#object* tipo *airline* vacío

Pruebas de integración de endpoints propios de aeronaves

Escenario de prueba	Descripción	Resultado
Check If Aircraft Exists By Iata Code [True]	Given path basePath + '/search/existsByIata' And param iata = global.aircraftIata When method get Then status 200 And match response == 'true'	Devuelve 200 y response=true
Check If Aircraft Exists By Iata Code [False]	Given path basePath + '/search/existsByIata' And param iata = global.stringNotFound When method get Then status 200 And match response == 'false'	Devuelve 200 y response=false
Check If Aircraft Exists By Icao Code [True]	Given path basePath + '/search/existsByIcao' And param icao = global.aircraftIcao When method get Then status 200 And match response == 'true'	Devuelve 200 y response=true
Check If Aircraft Exists By Icao Code [False]	Given path basePath + '/search/existsByIcao' And param icao = global.stringNotFound When method get Then status 200 And match response == 'false'	Devuelve 200 y response=false
Check If Aircraft Exists By Id Not And Iata Code [True]	Given path basePath + '/search/existsByIdNotAndIata' And param iata = global.aircraftIata And param id = global.numberNotFound When method get Then status 200 And match response == 'true	Devuelve 200 y response=true
Check If Aircraft Exists By Id Not And Iata Code [False]	Given path basePath + '/search/existsByIdNotAndIata' And param iata = global.aircraftIata And param id = global.aircraftId When method get Then status 200 And match response == 'false'	Devuelve 200 y response=false

Check If Aircraft Exists By Id Not And Icao Code [True]	Given path basePath + '/search/existsByIdNotAndIcao' And param icao = global.aircraftIcao And param id = global.numberNotFound When method get Then status 200 And match response == 'true'	Devuelve 200 y response=true
Check If Aircraft Exists By Id Not And Icao Code [False]	Given path basePath + '/search/existsByIdNotAndIcao' And param icao = global.aircraftIcao And param id = global.aircraftId When method get Then status 200 And match response == 'false'	Devuelve 200 y response=false
Find Aircraft by Name [Found]	Given path basePath + '/search/findByAircraftName' And param name = global. aircraftName When method get Then status 200 And match response == '#object'	Devuelve 200 y un *#object* tipo *aircraft*
Find Aircraft by Name [Not Found]	Given path basePath + '/search/findByAircraftName' And param name = global.stringNotFound When method get Then status 200 And match response == '#object'	Devuelve 200 y un *#object* tipo *aircraft* vacío
Find Aircraft by Name Containing [Found]	Given path basePath + '/search/findByAircraftNameContaining' And param name = global.aircraftNameContain When method get Then status 200 And match response == '#object'	Devuelve 200 y un *#object* tipo *aircraft*
Find Aircraft by Name Containing [Not Found]	Given path basePath + '/search/findByAircraftNameContaining' And param name = global.stringNotFound When method get Then status 200 And match response == '#object'	Devuelve 200 y un *#object* tipo *aircraft* vacío
Find Aircraft by Iata Code [Found]	Given path basePath + '/search/findByIata' And param iata = global.aircraftIata When method get Then status 200 And match response == '#object'	Devuelve 200 y un *#object* tipo *aircraft*

Find Aircraft by Iata Code [Not Found]	Given path basePath + '/search/findByIata' And param iata = global.stringNotFound When method get Then status 200 And match response == '#object'	Devuelve 200 y un *#object* tipo *aircraft* vacío
Find Aircraft by Icao Code [Found]	Given path basePath + '/search/findByIcao' And param icao = global.aircraftIcao When method get Then status 200 And match response == '#object'	Devuelve 200 y un *#object* tipo *aircraft*
Find Aircraft by Icao Code [Not Found]	Given path basePath + '/search/findByIcao' And param icao = global.stringNotFound When method get Then status 200 And match response == '#object'	Devuelve 200 y un *#object* tipo *aircraft* vacío

Pruebas de integración para la gestión de archivos con las 3 entidades

Escenario de prueba	Descripción	Resultado
file upload <basePath>	Given path <basePath> + '/fileupload' And def value = 'testUpload' And multipart file file = { value: '#(value)', filename: 'testUpload.csv'} And header Content-Type = 'multipart/form-data' And multipart field message = <message> When method patch Then status 200	Devuelve 200
file upload <basePath>	Given path <basePath> + '/fileupload' And def value = 'testDelete' And multipart file file = { value: '#(value)', filename: 'testDelete.csv'} And header Content-Type = 'multipart/form-data' And multipart field message = <message> When method patch Then status 200	Devuelve 200

Pruebas de integración comunes para las 3 entidades:

Borrar

Escenario de prueba	Descripción	Resultado
Delete <basePath>	Given path <basePath> And request read('classpath:features/model_crud/delete/delete/' + <expectedJsonName> + '-delete-bg.json') And method post Then status 201 * def resource_id = get response.id Given path <basePath> + '/' + resource_id When method delete Then status 200	Devuelve 200
Can't update <basePath> with non-existent id	Given path <basePath> And request read('classpath:features/model_crud/delete/non_existent_id/' + <expectedJsonName> + '-delete-non-existent-id-bg.json') And method post Then status 201 Given path <basePath> + '/' + global.numberNotFound When method delete Then status 400	Devuelve 400

Leer

Escenario de prueba	Descripción	Resultado
Get <basePath> Model [Found]	Given path <basePath> + '/1' When method get * def expected = read('classpath:features/model_crud/get/models/' + <expectedJsonName> +'.json') Then match response == expected	Devuelve la respuesta esperada
Check <basePath> Model [Not Found]	Given path <basePath> + '/' + global.numberNotFound When method get Then status 404	Devuelve 400
Get all <basePath>	Given path <basePath> When method get Then status 200	Devuelve 200

Guardar

Escenario de prueba	Descripción	Resultado
Save new <basePath> with required fields	Given path <basePath> And request read('classpath:features/model_crud/post/required_fields/' + <expectedJsonName> + '-post-required.json') When method post * def reponse = read('classpath:features/model_crud/get/models/' + <expectedJsonName> + '.json') Then status 201 And match response == response	Devuelve 201 y el objeto recién creado
Save new <basePath> with all fields	Given path <basePath> And request read('classpath:features/model_crud/post/all_fields/' + <expectedJsonName> + '-post-all.json') When method post * def reponse = read('classpath:features/model_crud/get/models/' + <expectedJsonName> + '.json') Then status 201 And match response == response	Devuelve 201 y el objeto recién creado
Can't save duplicated <basePath>	Given path <basePath> And request read('classpath:features/model_crud/post/duplicated/' + <expectedJsonName> + '-post-duplicated.json') When method post Then status 400	Devuelve un 400
Can't save <basePath> with invalid Json	Given path <basePath> And request read('classpath:features/model_crud/post/invalid/' + <expectedJsonName> + '-post-invalid.json') When method post Then status 400	Devuelve un 400

Actualizar

Escenario de prueba	Descripción	Resultado
Can't update <basePath> with non-existent id	Given path <basePath> And request read('classpath:features/model_crud/put/non_existent_id/' + <expectedJsonName> + '-put-non-existent-id-bg.json') And path <basePath> + '/' + global.numberNotFound When method put Then status 404	Devuelve un 404

I) Proceso de test de integración

El proceso de generación de test de integración es bastante sencillo, ya que, principalmente, en la documentación generada de la API (*swagger-ui*), hemos definido cuales son las posibilidades de respuestas, y para desarrollar estos test, simplemente deberemos de acceder al *swagger-ui* e implementarlos en función de ellos.

Para empezar con el desarrollo se debe (0) definir cuál va a ser la dirección de la API, que va a ser común para todos los test, (1) que ruta queremos acceder (1.1) y en caso de existir algún parámetro definir cuál, (2) de qué modo y (3) cuál va a ser la respuesta esperada. En caso de ser un objeto o una lista esperada, deberemos de comprobar que la respuesta trae ese objeto o lista, para ello definimos un objeto y una lista común, que en vez de comprar que devuelve unos valores en concreto, comprobamos que devuelve un tipo de valor en concreto, ya que el correcto funcionamiento respecto a los valores ya está testeado en los test unitarios y aquí lo que realmente importa es que devuelve el tipo de objeto esperado.

```
(0) Background:
       * url BASE_URL
       * def basePath = global.airport
…
(1) Given path basePath + '/search/existsByIata'
(1.1) And param iata = global.airportIata
(2) When method get
(3) Then status 200
(3) And match response == 'true'
```

J) Pruebas de diseño de interfaz

Esta prueba se ha realizado para saber si el diseño de la interfaz es intuitivo, para ello se han realizado ciertas pruebas con un gestor aeroportuario como experto y una persona sin conocimiento. Para saber si los resultados son aceptables o no, se multiplica por 1,5 el tiempo que realiza el desarrollador, que es la persona que más conoce el sistema, y posteriormente se realiza una media del tiempo del experto y la persona sin conocimiento:

Descripción de la prueba	¿Quién la realiza?	Tiempo	Tiempo del desarrollador	Resultado
Loguearse en la aplicación	Experto	00:05.81	00:04.77	Tiempo máximo = **7,15** Media = 6,75
	Persona sin conocimiento	00:07.69		
Acceder a la pantalla de aeropuertos y editar un aeropuerto	Experto	00:11.72	00:10.54	Tiempo máximo = **15,81** Media = 14,59
	Persona sin conocimiento	00:17.46		
Acceder a la pantalla de aerolíneas y crear una aerolínea	Experto	00:31.93	00:24.47	Tiempo máximo = **36,7** Media = 34,71
	Persona sin conocimiento	00:37.50		
Acceder a la pantalla de aeronaves y eliminar una aeronave	Experto	00:06.81	00:05.89	Tiempo máximo = **8,83** **Media = 8,9**
	Persona sin conocimiento	00:11.02		
Acceder a la pantalla de subir archivos y subir uno (aeronave y actualizar)	Experto	00:16.00	00:13.99	Tiempo máximo = **20,98** Media = 20,26
	Persona sin conocimiento	00:24.53		

Como se puede ver, los resultados son correctos, y entran dentro del tiempo establecido. Aunque, en la tercera prueba no entra dentro del límite de tiempo por 7 milesias de segundo, la damos por válida por ser un mínimo tiempo inapreciable.

Bibliografía

[1] Metodología Scrum: https://proyectosagiles.org/que-es-scrum/

[2] Jira: https://www.atlassian.com/es/software/jira

[3] Jenkins: https://jenkins.io/

[4] Git: https://git-scm.com/

[5] GitLab: https://about.gitlab.com/

[6] Nexus: https://www.sonatype.com/product-nexus-repository

[7] Docker: https://www.docker.com/

[8] Eclipse: https://www.eclipse.org/

[9] Spring-Boot: https://spring.io/

[10] DBeaver: https://dbeaver.io/

[11] PostgreSQL: https://www.postgresql.org/

[12] IntelliJ IDEA: https://www.jetbrains.com/es-es/idea/

[13] ReactJS: https://es.reactjs.org

[14] Apache Kafka: https://kafka.apache.org/

[15] Swagger: https://swagger.io/

[16] Flyway: https://flywaydb.org/

[17] Maven: https://maven.apache.org/

[18] Junit: https://junit.org/junit5/

[19] Mockito: https://site.mockito.org/

[20] SonarQube: https://www.sonarqube.org/

[21] Selenium: https://selenium.dev/

[22] Cucumber: https://cucumber.io/

[23] Karate: https://github.com/intuit/karate

[24] Draw.io: https://www.draw.io/

[25] Google Drive: https://www.google.com/intl/es_ALL/drive/

[26] Microsoft Word: https://products.office.com/es-es/word