Práctica final

Programación de Aplicaciones Telemáticas:

Aplicación para gestión de vuelos

Ignacion López López

Javier Ibarra González de la Ballina

Paula Sánchez Bodas

Pedro Marcos Plaza

Contenido

Introducción

1. Contexto y Objetivos

En el marco de la asignatura Programación de Aplicaciones Telemáticas (PAT), nuestro equipo —compuesto por Ignacio, Javier, Paula y Pedro— ha desarrollado una aplicación web de gestión de vuelos integrada con la API de Amadeus para ofrecer datos en tiempo real.

El objetivo principal del proyecto ha sido simular un entorno profesional de desarrollo, aplicando metodologías ágiles, integración continua (CI/CD) y trabajo colaborativo mediante GitHub. La aplicación permite:

- **Búsqueda de vuelos** (origen, destino, fechas)
- **Consulta de aeropuertos** por ciudad
- **Gestión de reservas** (creación y almacenamiento)
- **Despliegue automático** en Render.com

2. Motivación y Alcance

Este proyecto no solo ha servido para afianzar los conceptos vistos en clase, sino también para:

❖ Aplicar un flujo de desarrollo real:
- Uso de ramas Git (main, develop, feature branches)
- Integración Continua (CI) con GitHub Actions
- Despliegue Automático (CD) en Render

❖ Trabajar en equipo de manera coordinada:
- División de roles (frontend, backend, testing, documentación)
- Reuniones semanales para revisar avances
- Uso de GitHub Projects para gestión de tareas

❖ Resolver desafíos técnicos:
- Conexión con API externa (Amadeus)
- Pruebas unitarias, de integración y E2E
- Optimización del rendimiento en consultas de vuelos

3. Tecnologías Utilizadas

Área	Tecnología
Frontend	HTML5, CSS3, JavaScript
Backend	Java 17, Sprong Boot, Sping WebClient (API Amadeus)
Base de Datos	H2 (desarrollo) + PostgreSQL (producción)
Testing	Junit 5, Mockito
DevOps	GitHub Actions (CI), Docker, Render.com (CD)
Gestión	Github Projects, wiki, Issues

Planificación

Enfoque Ágil basado en SCRUM

Para el desarrollo de nuestra aplicación de gestión de vuelos, adoptamos una metodología ágil basada en Scrum, adaptada a las necesidades académicas y los plazos del curso. El proyecto comenzó oficialmente el 1 de abril de 2025, fecha en la que se conformaron los equipos, y culminó con la entrega final el 20 de mayo de 2025.

Sprints y Distribución del Trabajo

Debido a la coincidencia con el período de exámenes finales, organizamos el trabajo en 4 sprints de duración variable, optimizando el tiempo disponible:

Sprint 1 (01/04 – 13/4): Definición y Estructuración

Objetivo: Establecer las bases del proyecto.
Tareas clave:

- Elección del tema y API (Amadeus para datos de vuelos).
- Diseño inicial de la arquitectura (frontend/backend).
- Reparto de roles y tareas entre los miembros del equipo.
- Creación del repositorio GitHub y configuración inicial.

Sprint 2 (14/04 – 28/4): Desarrollo Inicial

Objetivo: Avance paralelo en frontend y backend.
Distribución de trabajo:

Equipo	Tareas
Javier y Pedro	Desarrollo del frontend (HTML/CSS/JS). Implementación de búsqueda de vuelos y formularios.
Ignacio y Paula	Configuración del backend (Spring Boot). Conexión con API Amadeus y modelo de datos.

Sprint 3 (29/04 – 16/04): Pausa por Exámenes

Ajustes:

- Reducción de actividad debido a los exámenes finales.
- Tareas mínimas de mantenimiento:
    - Revisión de código existente.
    - Documentación parcial en la Wiki de GitHub.

Sprint 4 (17/04 – 20/05): Integración y Finalización

Objetivo: Unificar componentes y pulir detalles.
Tareas clave:

1. **Integración frontend-backend**:
    - Conexión de los formularios con la API.
    - Pruebas de usabilidad.
2. **Documentación**:

- Completar la Wiki técnica.
- Redactar el manual de usuario.

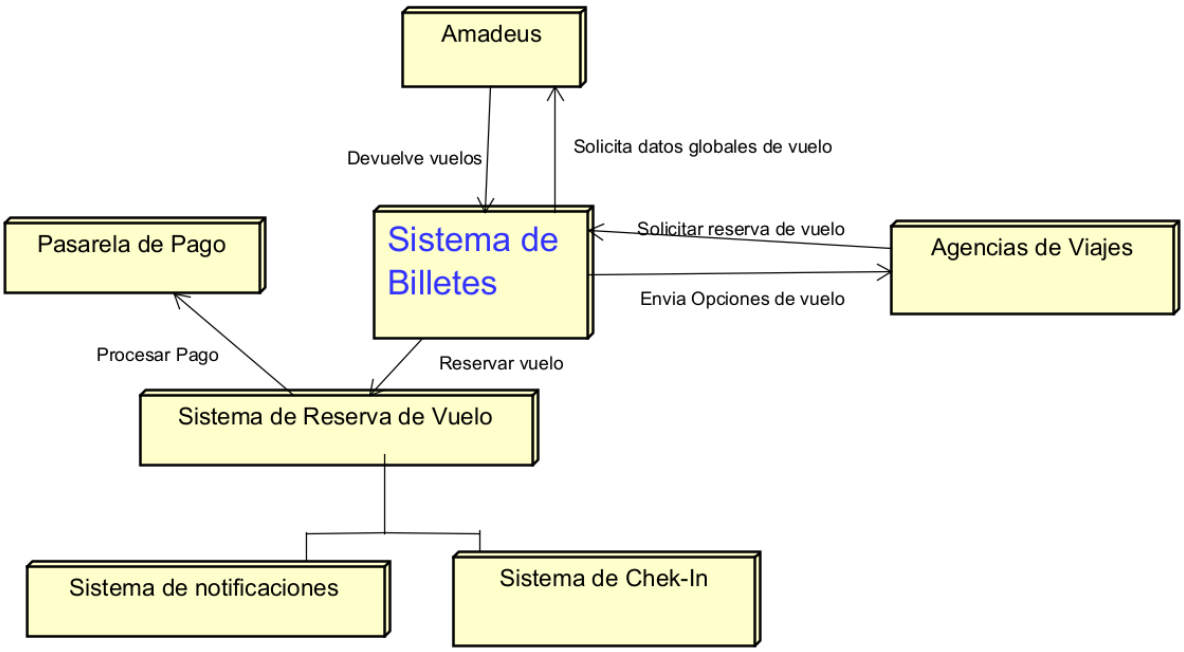
3. **Despliegue**:
   - Configuración final en Render.com.

Roles y Responsabilidades

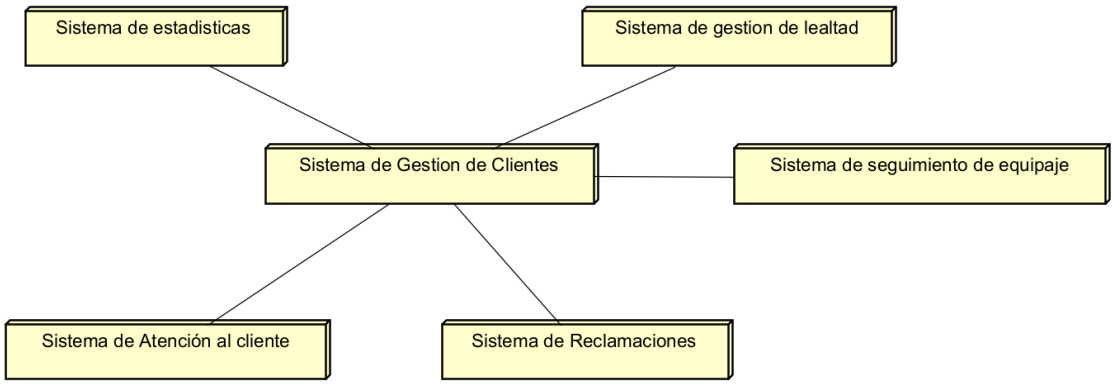
Miembro	**Rol Principal**	**Contribuciones Clave**
Pedro	Frontend/Coordinación	Diseño de interfaz, integración con API.
Javier	Frontend	Maquetación (HTML/CSS), lógica JavaScript.
Ignacio	Backend	Servicios Spring Boot, conexión con API Amadeus.
Paula	Backend/Testing	Modelo de datos, pruebas unitarias y de integración.

Modelos de contexto

Modelo de contexto de procesos empresariales: en este diagrama reflejamos las relaciones que tienen entre los distintos sistemas. Esto nos permite decidir sobre las fronteras que tiene un sistema en concreto y que funcionalidades le ofrece el entorno a un sistema. Por ejemplo, nuestro sistema de billetes se comporta de la siguiente manera:



Modelo de contexto arquitectónico: Este diagrama tiene el mismo propósito, sin embargo, no se muestra el tipo de relaciones entre los sistemas. En nuestro caso, el modelo de contexto arquitectónico del sistema de gestión de clientes:



Funcionalidades del sitio web

La plataforma construida es una aplicación basada en la web que permite a los usuarios buscar y reservar vuelos a diferentes ciudades. Esta aplicación es similar a motores de búsqueda de vuelos reales como Iberia, Skyscanner, Google Flights o Kayak, proporcionando resultados dinámicos a través de una API externa.

Desde una perspectiva funcional, el sistema ofrece los siguientes servicios:

- **Consulta de aeropuertos**: Ofrece la funcionalidad al usuario de buscar códigos de aeropuerto a partir del nombre de una ciudad. Así, ayuda al usuario en la selección del origen y destino de su vuelo.
- **Búsqueda de vuelos**: Una vez definidos los aeropuertos y las fechas, el sistema conecta con la API pública de Amadeus, que ofrece datos reales de vuelos, para mostrar al usuario las ofertas más relevantes.
- **Realización de reservas**: El usuario puede realizar una reserva, que se guardará en una base de datos local.
- **Consulta de reservas anteriores**: Cada usuario puede consultar las reservas que ha realizado anteriormente.

Todas estas funcionalidades están sustentadas por un backend modular, diseñado con Spring Boot, y basado en el paradigma REST para facilitar la interacción con cualquier cliente.

Estructura del Backend y resolución técnica

El backend ha sido implementado usando **Spring Boot**, un framework muy utilizado en la industria por velocidad de desarrollo y alta escalabilidad. Se ha estructurado siguiendo una arquitectura **en capas**, dividiendo el sistema en componentes independientes que se comunican entre sí:

Capa de Controlador (@RestController)

Esta capa en el desarrollo se encarga de define los endpoints que reciben las peticiones HTTP desde el cliente (como un navegador o aplicación frontend). Cada controlador se encarga de:

- Recoger parámetros de consulta o cuerpos JSON.
- Validar los datos de entrada.
- Delegar el procesamiento a la capa de servicios.

Ejemplo: cuando el usuario consulta vuelos, el controlador recibe los parámetros (origen, destino, fechaIda, fechaRegreso) y llama al servicio correspondiente.

Capa de Servicio (@Service)

El servicio contiene la lógica de negocio. Es la capa que:

- Trabaja con la API externa de Amadeus para buscar vuelos y aeropuertos.
- Realiza la transformación y procesamiento de la respuesta en objetos Java que se ajustan a lo que el frontend necesita.

- Se comunica con la base de datos para guardar y consultar reservas.

Gracias a esta capa, el sistema puede desacoplar la lógica compleja de los controladores y centralizar el comportamiento del negocio.

Capa de Repositorio (@Repository)

El repositorio se comunica directamente con la base de datos. Utiliza Spring Data JPA para realizar operaciones CRUD (crear, leer, actualizar, eliminar) de forma automática, sin necesidad de escribir consultas SQL manuales. Además de las operaciones CRUD básicas que proporciona el repositorio de reservas, se ha implementado una consulta adicional la cual permite obtener el listado completo de reservas asociadas a un usuario específico.

Esta funcionalidad no viene incluida en los métodos generados por Spring Data JPA, por lo que manualmente se tuvo que definir una query específica en el repositorio de reservas.

Cliente API externa: Amadeus WebClient

Para obtener los datos de vuelos reales, se ha integrado la **API de Amadeus**. Se ha creado un cliente HTTP (WebClient) que:

1. Solicita un token de acceso con client_id y client_secret.
2. Realiza peticiones autenticadas para:
    - Buscar vuelos.
    - Buscar aeropuertos.

Estas respuestas se reciben en formato JSON, que luego se transforma en objetos Java adaptados al dominio del sistema (VueloResponse, OfertaResponse, etc.).

Entidades y persistencia de datos

En el backend, se ha definido una entidad principal denominada **ReservaEntity,** que representa la reserva que realiza un usuario. Esta entidad se mapea directamente a una tabla en la base de datos relacional (como MySQL, PostgreSQL, H2, etc.).

ATRIBUTO	TIPO	DESCRIPCIÓN
id	Long	Clave primaria generada automáticamente.
precio	double	Precio total del vuelo reservado.
origen	String	Código IATA del aeropuerto de origen.
destino	String	Código IATA del aeropuerto de destino.
fechaSalida	String	Fecha de salida de vuelo.
fechaRegreso	String	Fecha de llegada de vuelo.
usuario	String	Nombre usuario que realiza la reserva.

Estas reservas se almacenan mediante Spring Data JPA y pueden ser recuperadas fácilmente mediante consultas personalizadas (por ejemplo, findByUsuario).

EndPoints rest Facilitados por el Backend

GET /vuelos

Este endpoint realiza la función de buscar vuelos entre dos aeropuertos en unas fechas dadas. Devuelve un listado con las ofertas más relevantes.

Parámetros:

- origen (String): Código IATA del aeropuerto de origen.
- destino (String): Código IATA del destino.
- fechaIda (String): Fecha de salida.
- fechaVuelta (String, opcional): Fecha de regreso.

Códigos HTTP devueltos:

- 200 OK: Datos obtenidos correctamente.
- 400 Bad Request: Parámetros incorrectos o mal formateados.
- 500 Internal Server Error: Fallo en la conexión con Amadeus o errores internos.

GET /aeropuertos

Este endpoint permite obtener los códigos de aeropuerto disponibles para una ciudad.

Parámetros:

- nombreCiudad (String): Nombre de la ciudad (ej. "Londres").

Códigos HTTP devueltos:

- 200 OK: Lista de aeropuertos obtenida correctamente.
- 400 Bad Request: Campo obligatorio no proporcionado.
- 500 Internal Server Error: este error se corresponde con un error en la llamada a Amadeus.

Este endpoint resulta muy útil en ciudades que tienen más de un aeropuerto, como es el caso de Londres. Así el usuario podrá ver cual de todos los aeropuertos es el que está buscando.

POST /reservas

Este endpoint permite registrar una nueva reserva de vuelo por parte de un usuario. Se espera un objeto JSON que represente la reserva. Este endpoint se realizó para que se creara una reserva “ficticia” y verificar que se guarda en la base de datos.

Cuerpo esperado (@RequestBody):

{
 "precio": 123.45,
 "origen": "MAD",
 "destino": "CDG",
 "fechaSalida": "2025-06-12",

"fechaRegreso": "2025-06-20",
"usuario": "Paula"
}

Códigos HTTP devueltos:

- 201 Created: Reserva almacenada correctamente.
- 400 Bad Request: Faltan campos o formato inválido.
- 500 Internal Server Error: Error al guardar en la base de datos.

GET /reservas

Este endpoint devuelve el listado de reservas realizadas por un usuario determinado.

Parámetros:

- usuario (String): Identificador del usuario.

Códigos HTTP devueltos:

- 200 OK: Lista de reservas devuelta correctamente.
- 400 Bad Request: Campo obligatorio no incluido.
- 500 Internal Server Error: Error en la base de datos.

Servicio implementado

En el servicio se han implementado los siguientes métodos:

-String getAccessToken()

-VueloResponse buscarVuelos(String origen, String destino, String fechaIda, String fechaVuelta)

-List<AeropuertoResponse> buscarAeropuertos(String nombreCiudad)

- ReservaEntity reservar(ReservaEntity reservaRequest)

-List<ReservaEntity>buscarReservas(String usuario)

-List<ItinerarioResponse> retrieveItinerarios(JsonNode jsonNode)

El componente más desafiante de todo el sistema backend, sin duda, ha sido la implementación de la funcionalidad de búsqueda de vuelos utilizando la API externa de Amadeus. Su implementación ha requerido el desarrollo de un flujo de autenticación completo junto con el consumo de servicios REST externos, procesamiento avanzado de datos en formato JSON y conversión a objetos de dominio interno

A continuación, se detalla cada etapa de este proceso:

1. Obtención del access token (OAuth2)

El primer paso antes de que se puedan realizar solicitudes a los diversos puntos finales de Amadeus es realizar la autenticación a través del protocolo OAuth 2.0 para garantizar comunicaciones seguras.

• Amadeus (/v1/security/oauth2/token) tiene un punto final de autenticación que acepta solicitudes POST y requiere credenciales del sistema (client*id y client*secret).

• En respuesta, Amadeus envía un token de acceso temporalmente válido (access_token), y este token debe ser proporcionado en el encabezado de todas las solicitudes posteriores en este formato: Authorization: Bearer {token}.

• El backend debe manejar los problemas de expiración del token y asegurarse de que siempre se utilice un token válido, lo que añade mayor complejidad.

3.Petición de vuelos (GET /v2/shopping/flight-offers)

En el estado autenticado, se puede realizar una búsqueda de vuelos utilizando una solicitud GET:

- originLocationCode: código IATA del aeropuerto de origen.
- destinationLocationCode: código IATA del destino.
- departureDate: fecha del vuelo.
- returnDate: (opcional) fecha de regreso.
- adults: número de pasajeros.
- Otros filtros como clase (económica, business) o aerolínea también pueden incluirse.

Esta petición devuelve un objeto JSON extremadamente grande y jerarquizado, que puede contener cientos de resultados con diferentes combinaciones de vuelos, escalas, precios y aerolíneas.

3. Procesamiento de la respuesta JSON

Este fue uno de los principales retos técnicos. La respuesta devuelta por Amadeus tiene una estructura altamente anidada, que exige un procesamiento en profundidad para extraer la información útil para el frontend.

4. Construcción de modelos de respuesta

Para proteger al frontend de la complejidad del formato original de Amadeus, se construyeron DTOs personalizados en Java:

• OfertaResponse: encapsula información general sobre la oferta, por ejemplo, el precio total.
• ItinerarioResponse: define el viaje (de ida o de regreso) con todos sus segmentos.
• SeguimientoResponse: contiene información particular de cada tramo: compañía, horario, el aeropuerto de salida y el aeropuerto de llegada, duración.

Esto tiene las siguientes ventajas:

• Mejora de la usabilidad desde el frontend (estructura limpia y controlada).
• Desacoplamiento del sistema de futuros cambios en la API de Amadeus.
• Rendimiento al enviar solo la información requerida.

El servicio buscarVuelos, a pesar de su lógica compleja, completamente funcional en torno a respuestas estructuradas y optimizadas, representa uno de los pilares del sistema. Sienta las bases para futuras mejoras como:

• Filtrado por precio, duración, número de escalas.
• Selección de aerolíneas preferidas.
• Combinación de opciones más baratas en la visualización.

Manual de Usuario

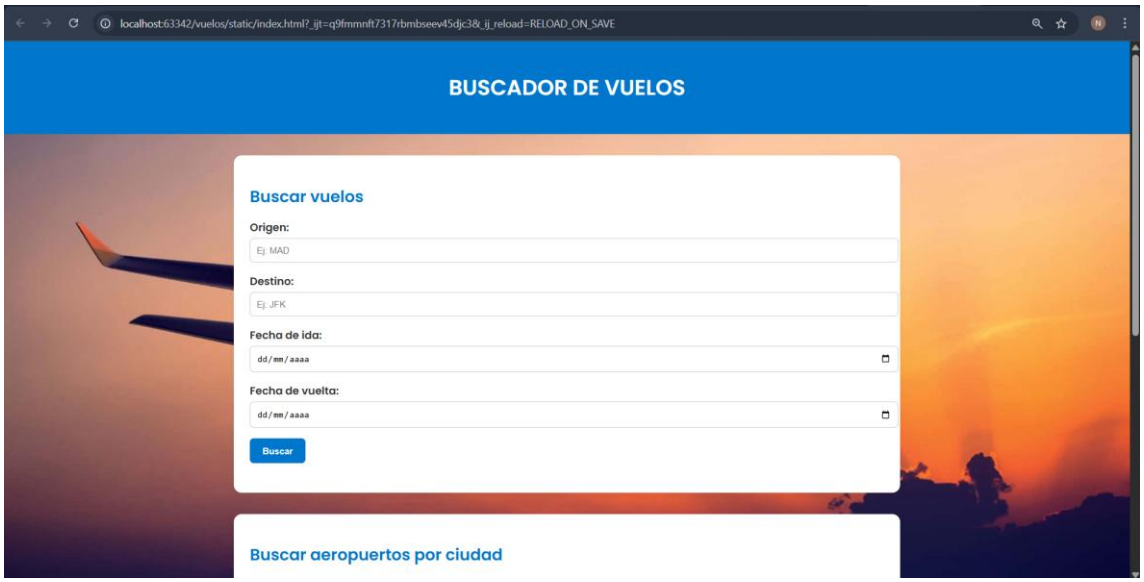


Ilustración 1 Buscar Vuelos

Comenzamos por la página principal donde tenemos la funcionalidad de buscar vuelos por nombre de aeropuerto de origen, destino, fecha de ida y fecha de vuelta. Para que el usuario conozca el nombre del aeropuerto de su ciudad, existe debajo un bloque en el que se puede consultar:

Ilustración 2 Buscar nombre aeropuerto

Una vez ya has elegido el nombre del aeropuerto, puedes buscar el origen, destino y fechas que más se correspondan a tus necesidades y presionas el botón buscar:

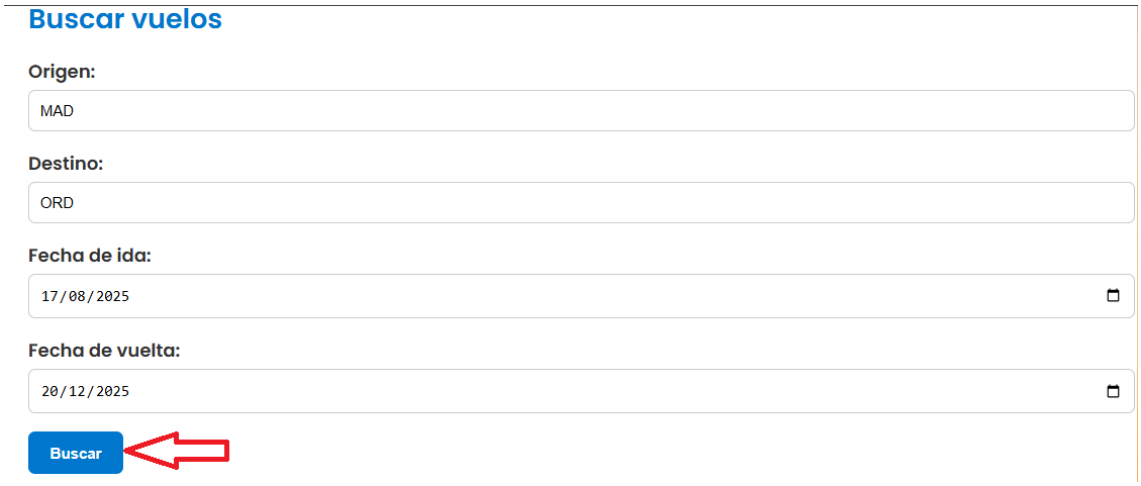


Ilustración 3 Buscar vuelo

Obtendrás un resultado similar a este:

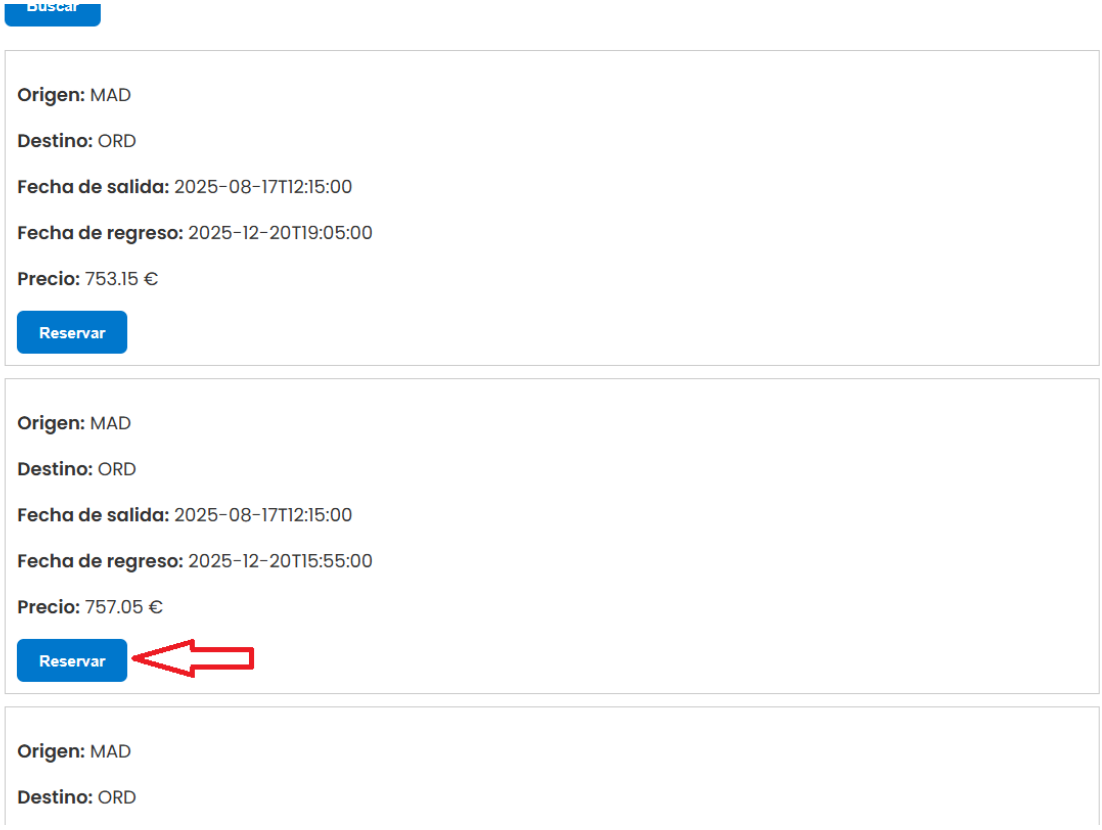


Ilustración 4 Reservar vuelo

Aparece el origen, destino, fecha y hora de ida y fecha y hora de vuelta. Además, aparece el precio del billete. Si lo desea puedes reservar tu plaza presionando el botón de reservar, se desplegará una alerta para ingresar tu nombre y confirmar tu reserva:

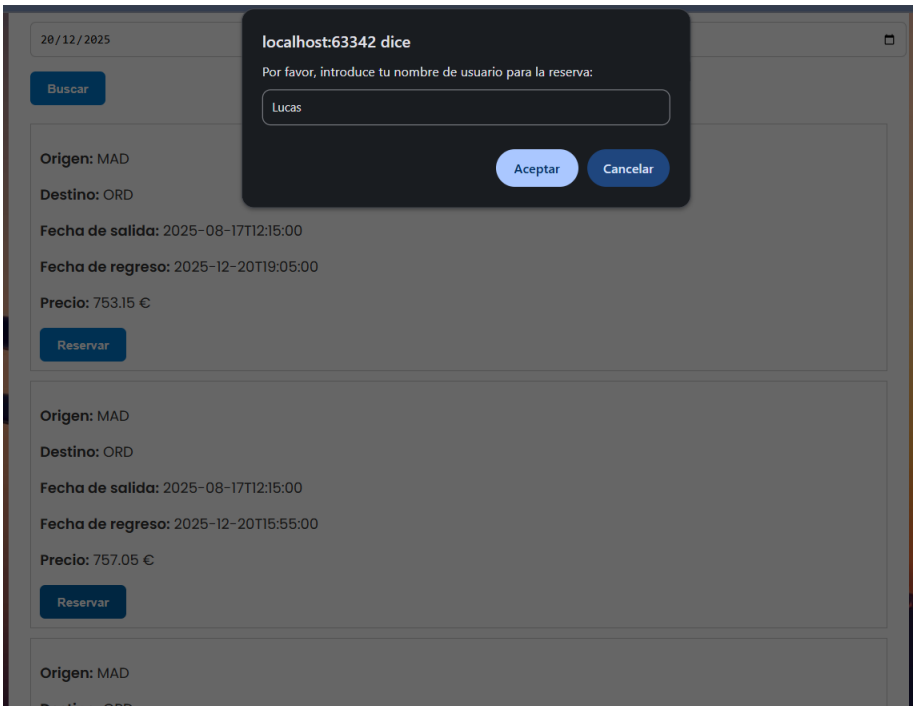


Ilustración 5 Hacer Reserva

Una vez confirmes tu reserva aparece tu ID de vuelo:

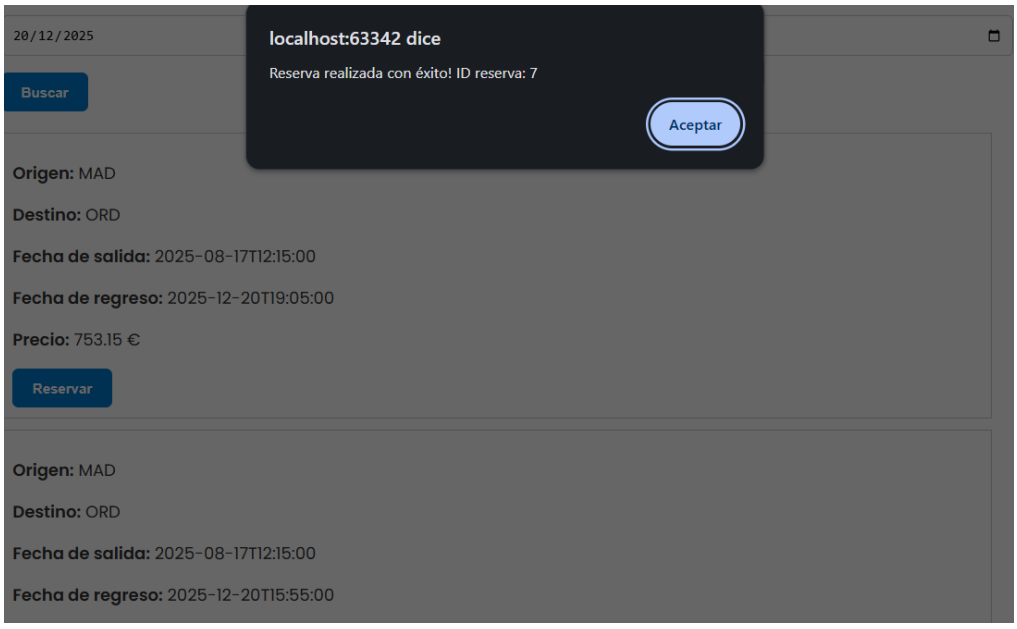


Ilustración 6 Id de vuelo

Una vez tengas tu ID significa que la reserva esta completada. Puedes consultar tus reservas justo debajo de buscar aeropuertos:

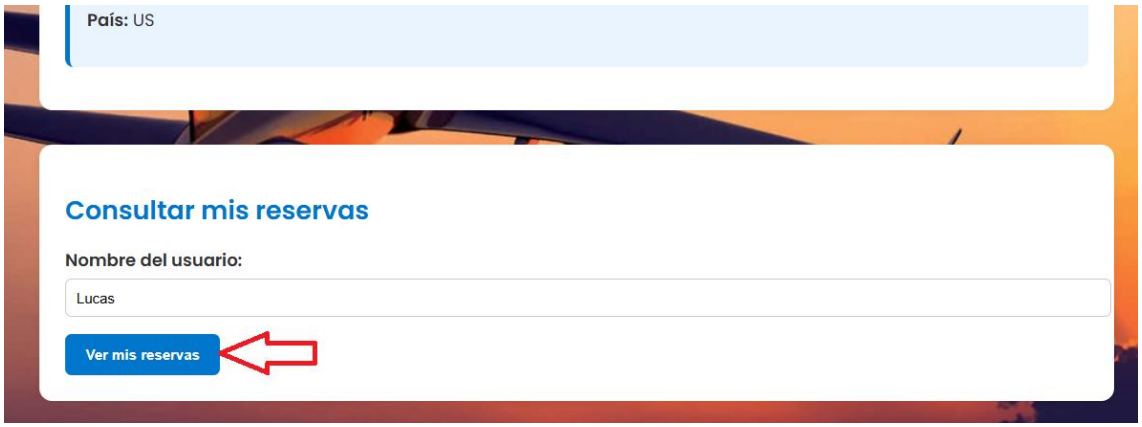


Ilustración 7 Consultar reservas

En este momento se le abrirá una página similar a esta donde puede comprobar sus reservas de vuelo:

Puede presionar el botón Volver a la búsqueda para volver a la página principal del buscador de vuelos.