|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mostrando Marca_Txurdinaga_Color_H_SUP_01_01.jpg | Curso / *Kurtsoa* | Fecha / *Data* | Nivel / *Maila* | Eval. /*Ebal.* |
| 2025-26 | 09/02/2026 | 2º | 2ª evaluación |
| Módulo / *Modulua* | *Kodea / Código* | *U.Didak /Unid didác* | *tipo* | Calificación/*Kalifikazioa* |
| DWC | 2DW3 | 1, 2, 3, 4, 5, 6, Vue.js | P |  |
| Nombre Alumno/a / Ikaslearen izena | Daniel CABALLERO | | | |

**EXAMEN**

* Este examen es de tipo procedimental.
* Tiene un valor total de 10 puntos.
* Si la propuesta del examen se resuelve de forma correcta se le aplica el valor especificado en cada apartado.
* Descargar el scaffolding. En tu equipo, haz los commits necesarios. Cuando quieras entregar sube el repositorio.

**Ejercicio 1. Javascript (2 ptos)**

Una aerolínea necesita una aplicación web para calcular las **tasas de equipaje** que deben pagar los pasajeros según el peso de su equipaje y la clase de vuelo.

Deberás crear una aplicación usando ***JavaScript vanilla*** que aplique las siguientes reglas:

1. **Peso incluido**: Cada pasajero tiene **20 kg** de equipaje incluido sin coste adicional.
2. **Cargo por exceso de peso**:
   1. Si el peso supera los 20 kg, se cobra **10€ por cada kilo de exceso**.
   2. Ejemplo: 25 kg → 5 kg de exceso → 50 EUR
3. **Cargo por clase Business**:
   1. Los pasajeros de clase **Business** pagan un cargo adicional de 50€ (independiente del peso).
   2. Los pasajeros de clase **Turista** NO pagan este cargo.
4. **Cálculo del total**:
   1. Total = Cargo por exceso + Cargo por clase

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EJEMPLOS** | | | | | |
| Clase | Peso | Exceso | Cargo Exceso | Cargo Business | **TOTAL** |
| Turista | 15 kg | 0 kg | 0€ | 0€ | **0€** |
| Turista | 25 kg | 5 kg | 50€ | 0€ | **50€** |
| Business | 18 kg | 0 kg | 0€ | 50€ | **50€** |
| Business | 35kg | 15 kg | 150€ | 50€ | **200€** |

Se te entrega un archivo **index.html** con la estructura básica y los estilos CSS. Tu tarea es programar la lógica en dos archivos externos: **Reserva.js** y **app.js**..

1. **Clase Reserva.js**

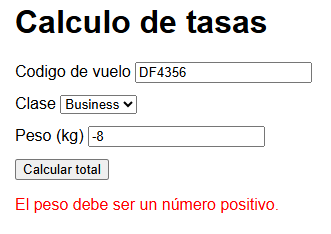
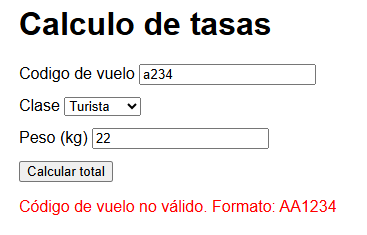
Crea una clase llamada Reserva. El constructor debe capturar y almacenar los siguientes datos:

* Código de vuelo.
* Clase.
* Peso.

1. **Validación del formulario (script.js)**

Al pulsar el botón **"Cálculo Total"**, debes validar los datos de los input de texto:

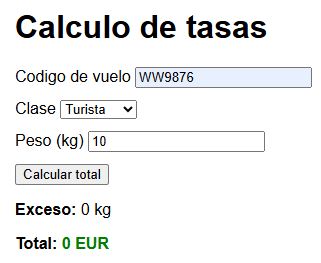
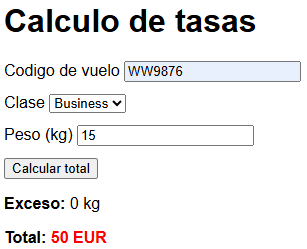
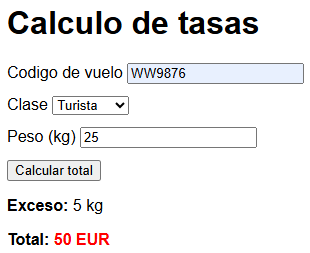
1. **Código de vuelo:** el código de vuelo tiene que tener el formato AA9999. Es decir, dos letras mayúsculas y 4 dígitos (usa una expresión regular).
2. **Peso:** el peso tiene que ser un número positivo menor de 100 kg.
3. **Errores:** si alguna validación falla, escribe el mensaje de error en el div #resultado y detén la ejecución.



1. **Creación de un objeto y generación de la reserva:**

Si los datos son válidos:

1. Instancia un nuevo objeto de la clase Reserva.
2. Crea dos elementos <p> en el div #resultado.
3. Dentro del primer <p> mostrará la información del exceso de peso.
4. Dentro del segundo <p> mostrará el precio asociado al peso:
   * Color verde si el precio es 0€
   * Color rojo si el precio es > 0€
5. **Estructura interna (se valorará positivamente el uso de métodos del DOM, en lugar de InnerHTML para crear los elementos):**

|  |  |
| --- | --- |
| **Total ejercicio - 1** |  |
| * 1. Se ha creado el objeto Reserva correctamente. | 0,10 |
| * 1. Se ha instanciado el objeto de tipo Socio correctamente si el formulario es correcto. | 0,10 |
| * 1. Se ha validado el código de vuelo correctamente (dos letras mayúsculas y 4 números). | 0,50 |
| * 1. Se ha validado el peso correctamente ( 0 < peso <100). | 0,15 |
| * 1. En caso de que el formulario no se valide correctamente, el formulario no se envía y se muestra el error. | 0,10 |
| * 1. En caso de que el formulario no se valide correctamente, el formulario no se envía y se muestra el error en el **div #resultado** | 0,40 |
| * 1. Se crea un elemento de tipo <p> en el div #resultado con el exceso de equipaje. | 0,20 |
| * 1. Se crea un elemento de tipo <p> en el div #resultado con el precio del equipaje | 0,20 |
| * 1. El color del equipaje se pone en verde o rojo en función del resultado. | 0,25 |

**Ejercicio 2. Vue.js: (8 ptos)**

**Txurdinaga Fly** es una compañía aérea con sede en Bilbao que ofrece vuelos entre las ciudades más importantes del mundo. En su página web es posible realizar multitud de operaciones; entre ellas: visualizar todos los vuelos disponibles con la información más importante, hacer prerreserva de vuelos con solo el correo electrónico y consultar todos los vuelos que ha prerreservado una persona a partir del correo electrónico o del correo y del código de reserva.

Para ello la app dispondrá de diferentes vistas (Vue-Router). Las diferentes vistas (Views) nos permitirán navegar por la SPA (Single Page Application).

Para la carga de datos utilizaremos **Axios** (ya incluido en package.json) y el fichero json (vuelos.json) que están en el directorio **public/data**.

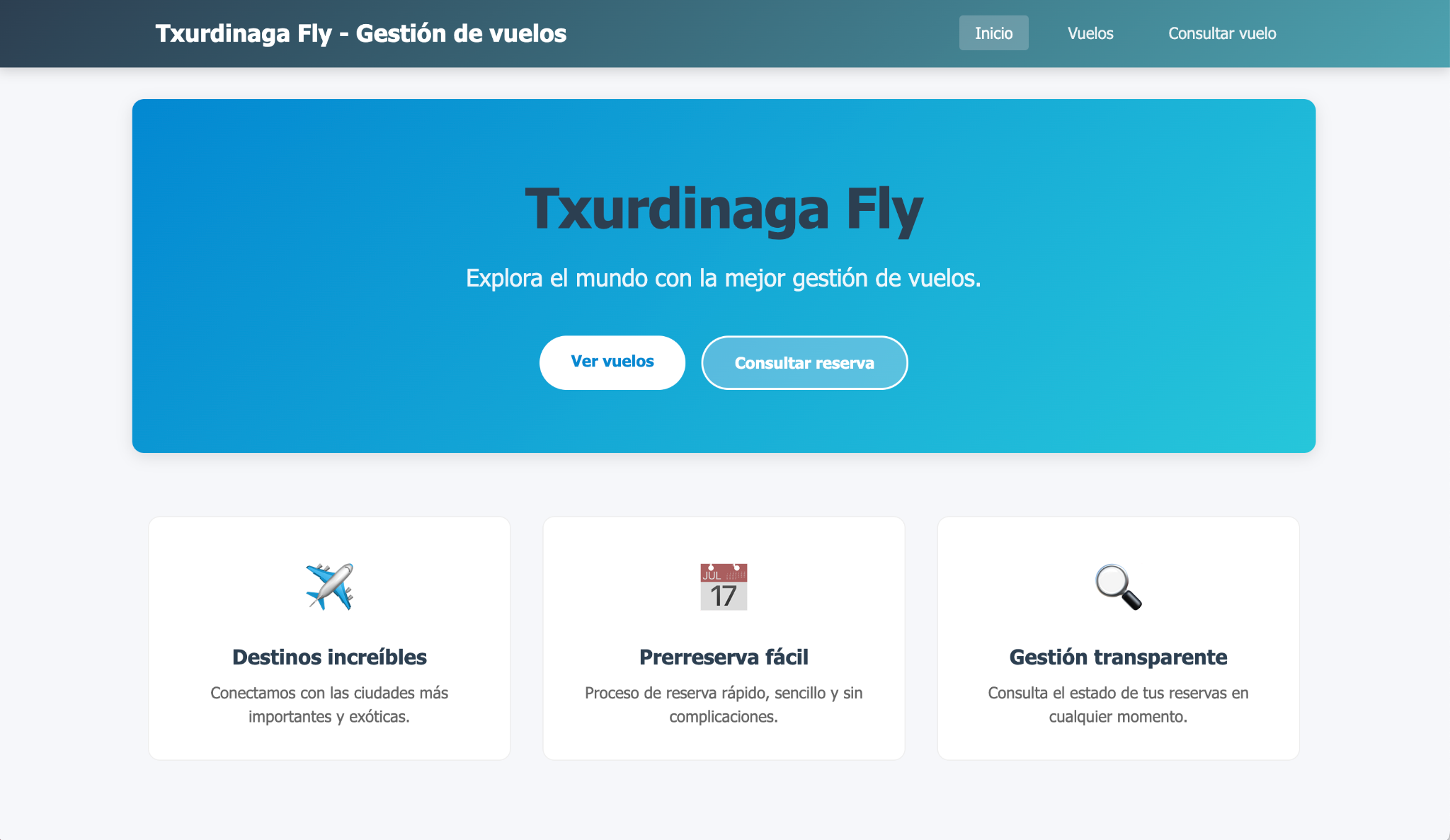
Este proyecto ya incluye la vista ***HomeView.vue*** que se carga al inicio del programa desde ***App.vue***

Tendréis que crear la vistas ***VuelosView.vue, VueloDetalleView.vue y ConsultaVueloView.vue*** para conseguir la siguiente app.

|  |
| --- |
| **IMPORTANTE**   1. VueloDetalleView.vue: es una vista con parámetro a la que se accede desde VuelosView.vue 2. La aplicación deberá implementar la gestión del estado global utilizando Pinia como store oficial del proyecto para guardar las prerreservas que se hagan desde el navegador, almacenando:    1. Email del usuario que quiere hacer la prerreserva    2. Vuelo que quiere prerreservar (desde la vista detalle)    3. Número de pasajeros que anotar en la prerreserva. 3. El elemento del menú “Consultar vuelo” solo será visible si hay algún vuelo guardado en reservas. |

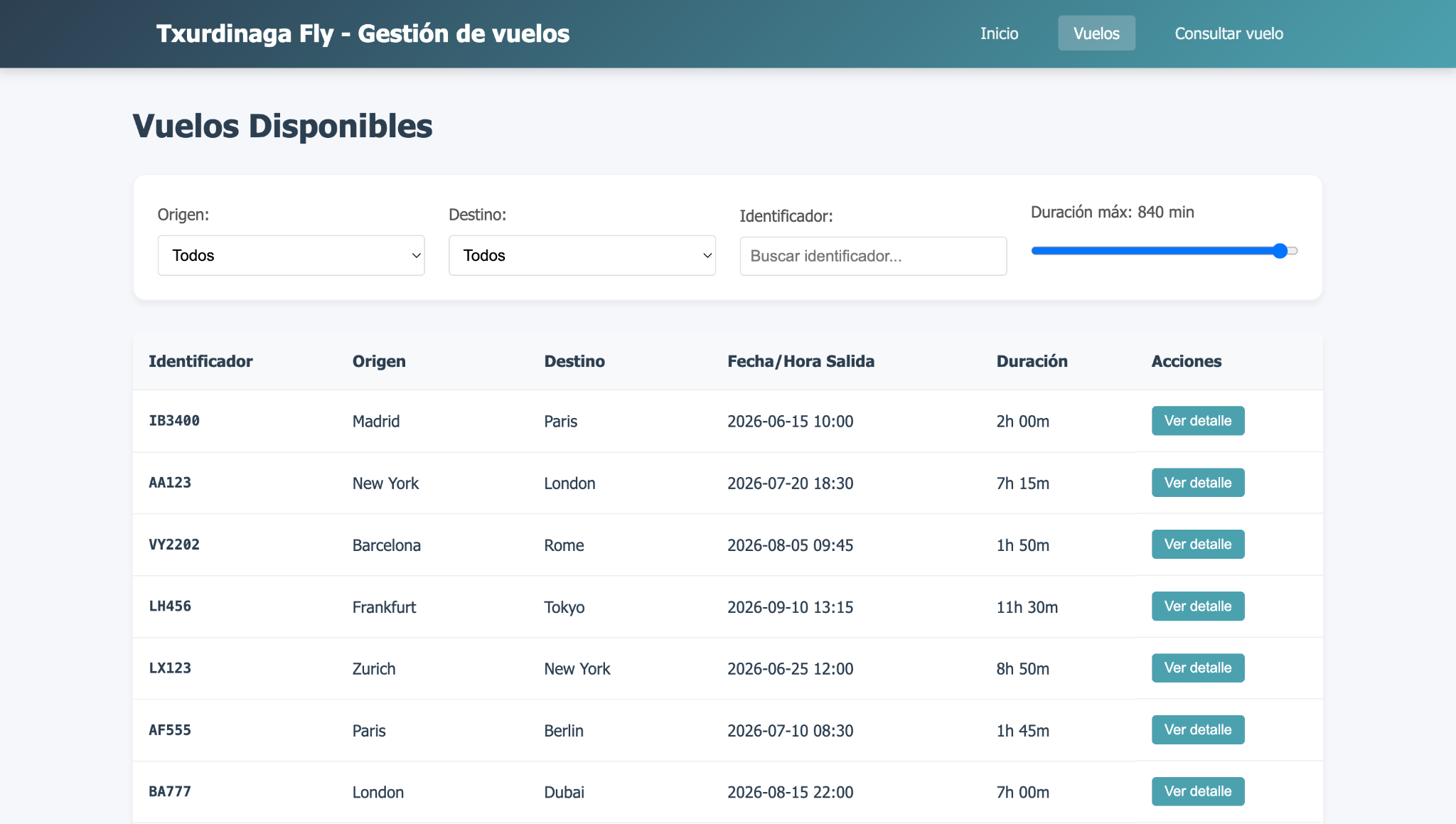
1. **HomeView**

Cuando se inicia la app se visualizarán todas las rutas a las que se podrá acceder haciendo click sobre cada elemento del menú: Inicio, Vuelos y Consultar vuelo; y también sobre los botones: Ver vuelos y Consultar reserva.



1. **Vuelos (filtros y propiedades computadas)**

Al hacer click en el botón “Vuelos” se cargará una lista con todos los vuelos que se extraerán del fichero json indicado previamente y que se encuentra en el directorio **/public/data/vuelos.json**.



En la lista se deberá mostrar la siguiente información que se puede extraer del fichero json.

* Identificador.
* Origen.
* Destino.
* Fecha / Hora de salida: puedes mostrar los dos separados por un espacio
* Duración: en el json está en minutos: se valorará que se muestre en el formato indicado en la imagen.
* En cada fila se mostrará también un botón adicional (**Ver detalle**) que nos dará toda la información del vuelo en una página aparte.

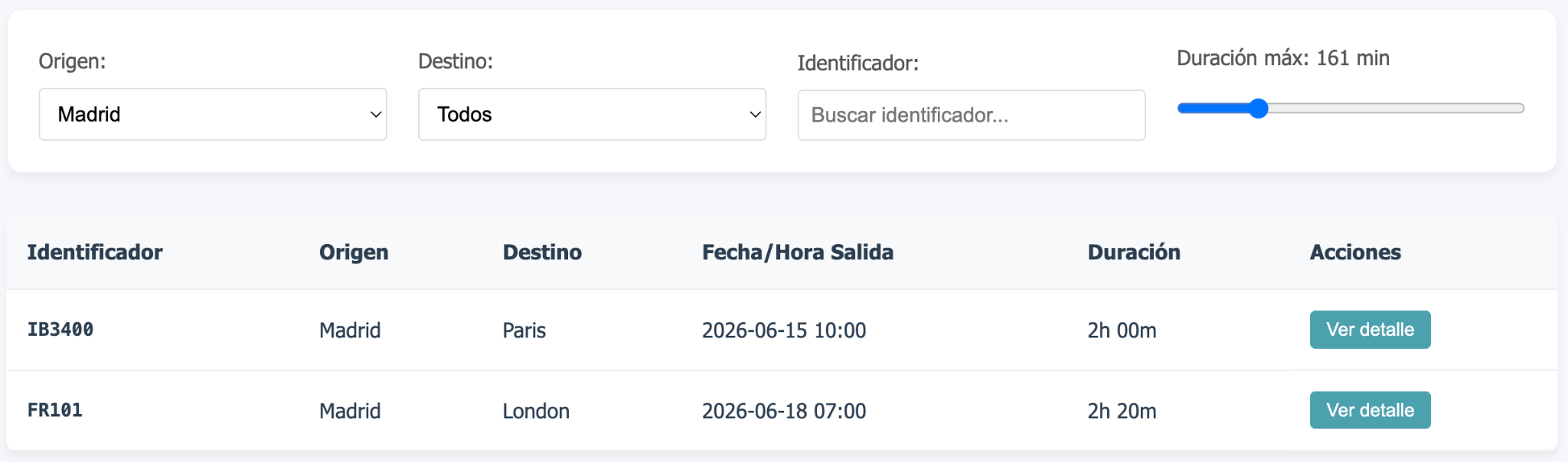
Además, hay varios **FILTROS** que permitirán realizar búsquedas sobre dicha lista, sin tener en cuenta las mayúsculas o minúsculas. Para ello se hará uso de **propiedades computadas**.

Los filtros son :

* origen y destino: es un campo desplegable con los nombres de las ciudades. Los valores no deberán estar harcodeados ni repetidos.
* identificador: es un campo de tipo texto para escribir el nombre del identificador. No se tendrán en cuenta mayúsculas y minúsculas.
* duración: es un elemento tipo range que filtra los vuelos por duración máxima, de manera que si, por ejemplo, se coloca el punto en el 100, se mostrarán todos los vuelos de menos de 100 minutos.

**IMPORTANTE**: Los datos mostrados deben cumplir **TODOS** los filtros seleccionados.

Un ejemplo de funcionamiento podría ser este:



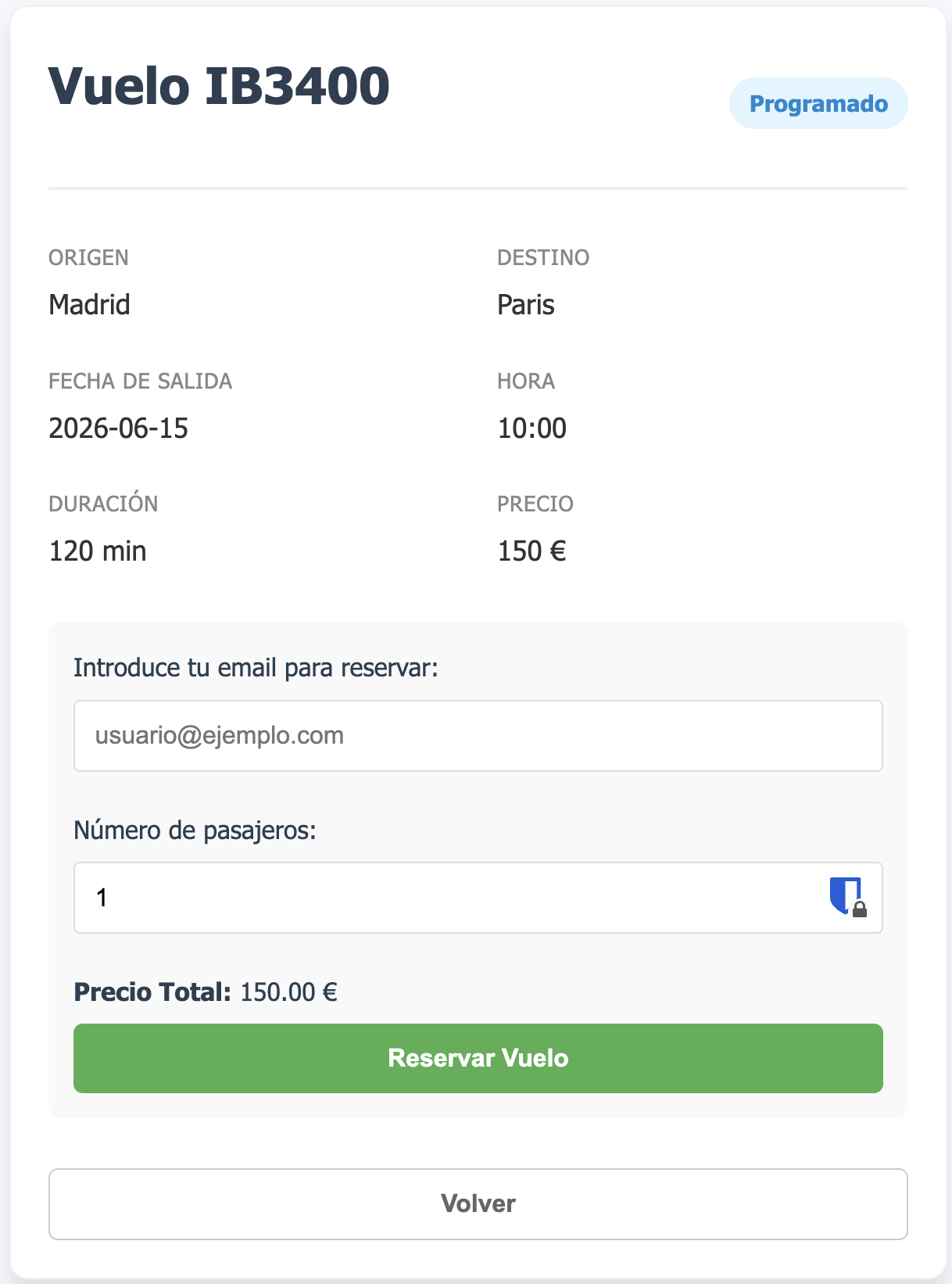
1. **Detalle del vuelo**

Al pinchar en el botón de **Ver detalle**, nos mostrará toda la información de ese vuelo. La vista en la que se muestra es **VueloDetalleView.vue,** y esta vista se muestra en una ruta con parámetro **/vuelos/idVuelo,** teniendo en cuenta que el parámetro será el código o identificador del vuelo.

Además se añadirá:

* **un botón Volver**, para regresar a la lista de vuelos,
* **una formulario de reserva**, que permitirá guardar reservas provisionales en el navegador.

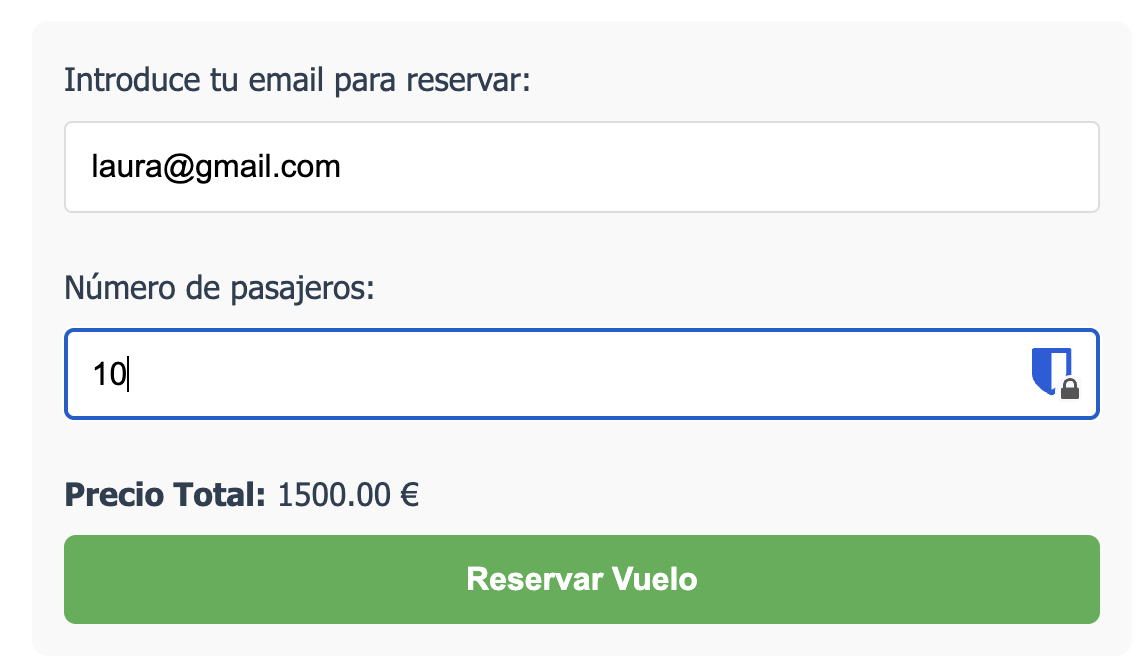
Un ejemplo de cómo podría ser esta página sería la siguiente:



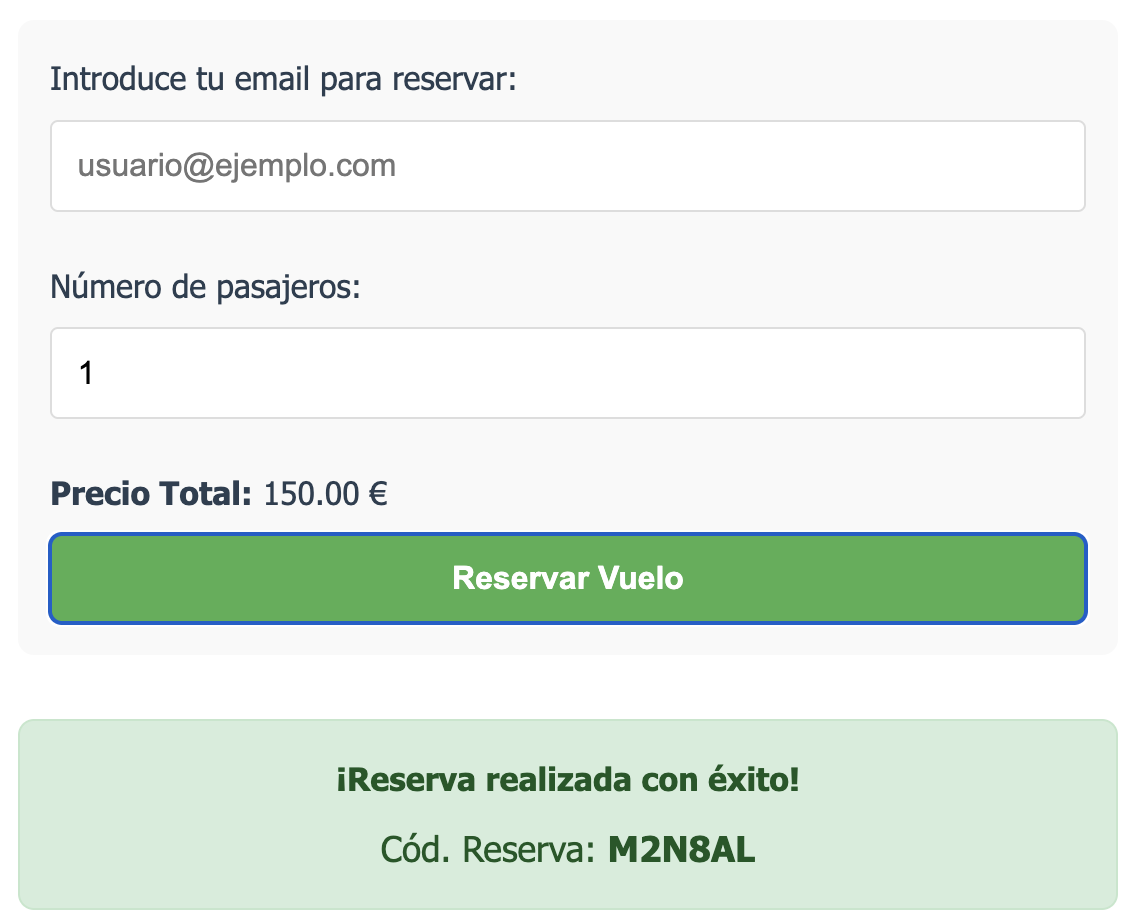
1. **Reservas**

Cuando se intenta hacer una reserva hay que tener en cuenta lo siguiente:

1. El usuario debe introducir un e-mail válido y un número de pasajeros antes de pulsar el botón (puedes utilizar cualquier mecanismo, incluso validación con HTML).
2. La etiqueta “Precio total” debe actualizar el precio automáticamente en función del valor que se meta en el campo de Número de pasajeros, tal y como se ve en la imagen:

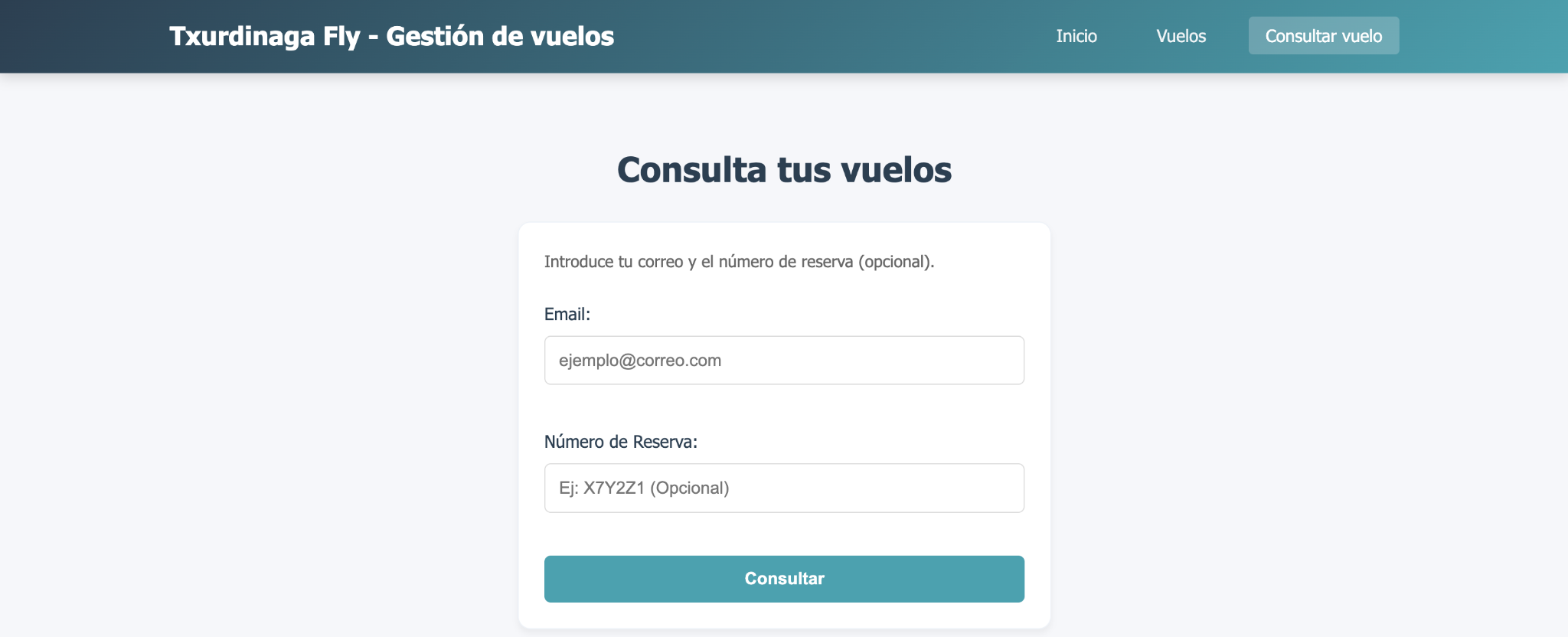


1. Cuando el usuario pulsa el botón “Reservar vuelo”, el formulario se resetea (se vacía el campo email y se inicializa a 1 el valor del número de pasajeros) y se genera un código aleatorio único (puede ser un número, letras…) que se muestra en la pantalla, tal y como se muestra a continuación:

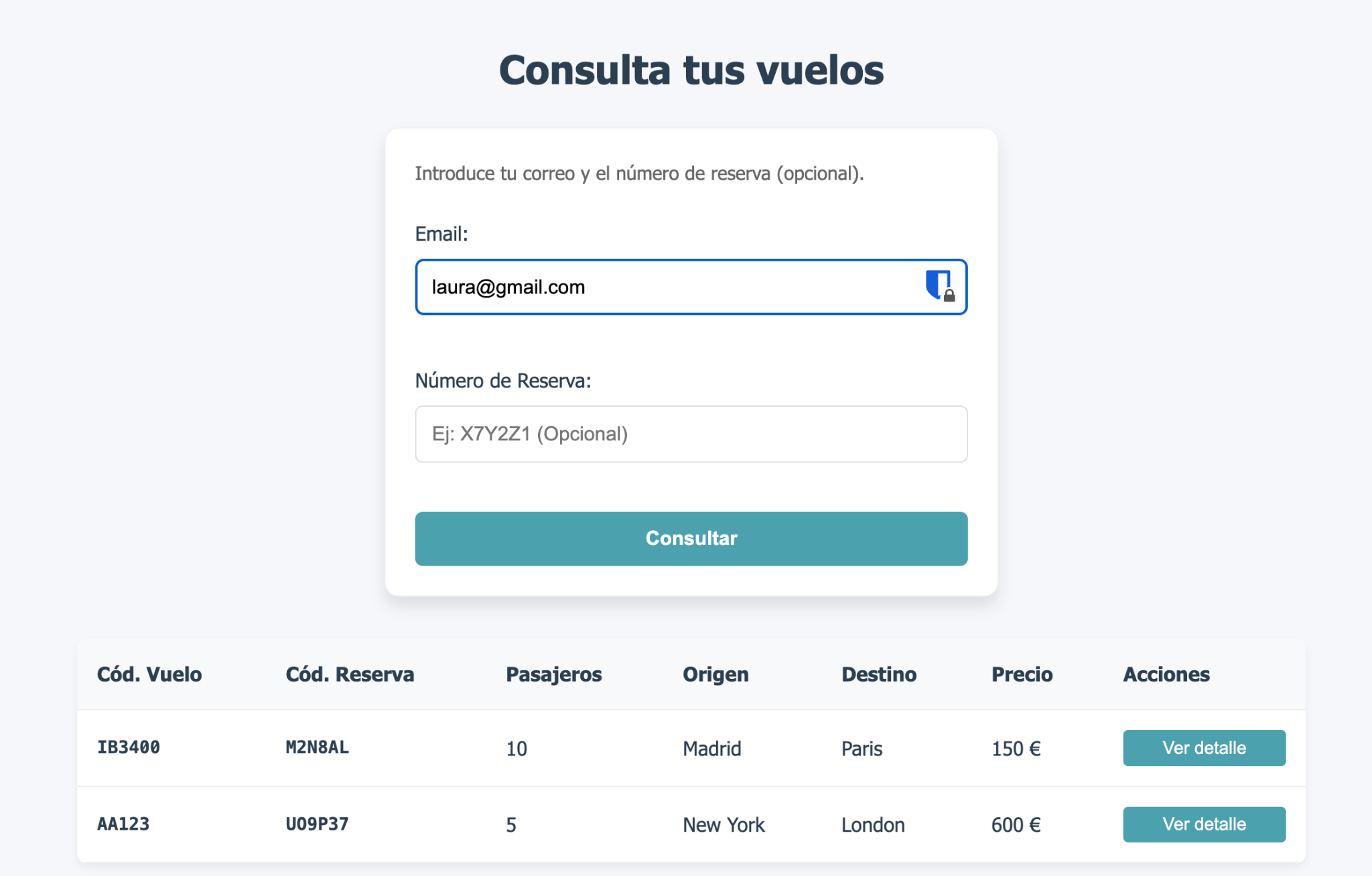


1. Al reservar, los datos de e-mail, número de pasajeros, código de vuelo y código de reserva se almacenan en una store con Pinia para poder ser utilizados en la pantalla Consultar vuelo.
2. **Importante**: se deberá comprobar que **el mismo usuario no ha hecho una reserva del mismo vuelo** con anterioridad, indicándolo en caso contrario (por ejemplo, con un mensaje “Ya tienes una reserva con ese e-mail para ese vuelo”.
3. **Consultar vuelos**

El usuario puede consultar los vuelos que se han realizado con un e-mail en concreto, o incluso si hay una reserva con un e-mail y un código de reserva concreto. La página debe contener un formulario como el siguiente:



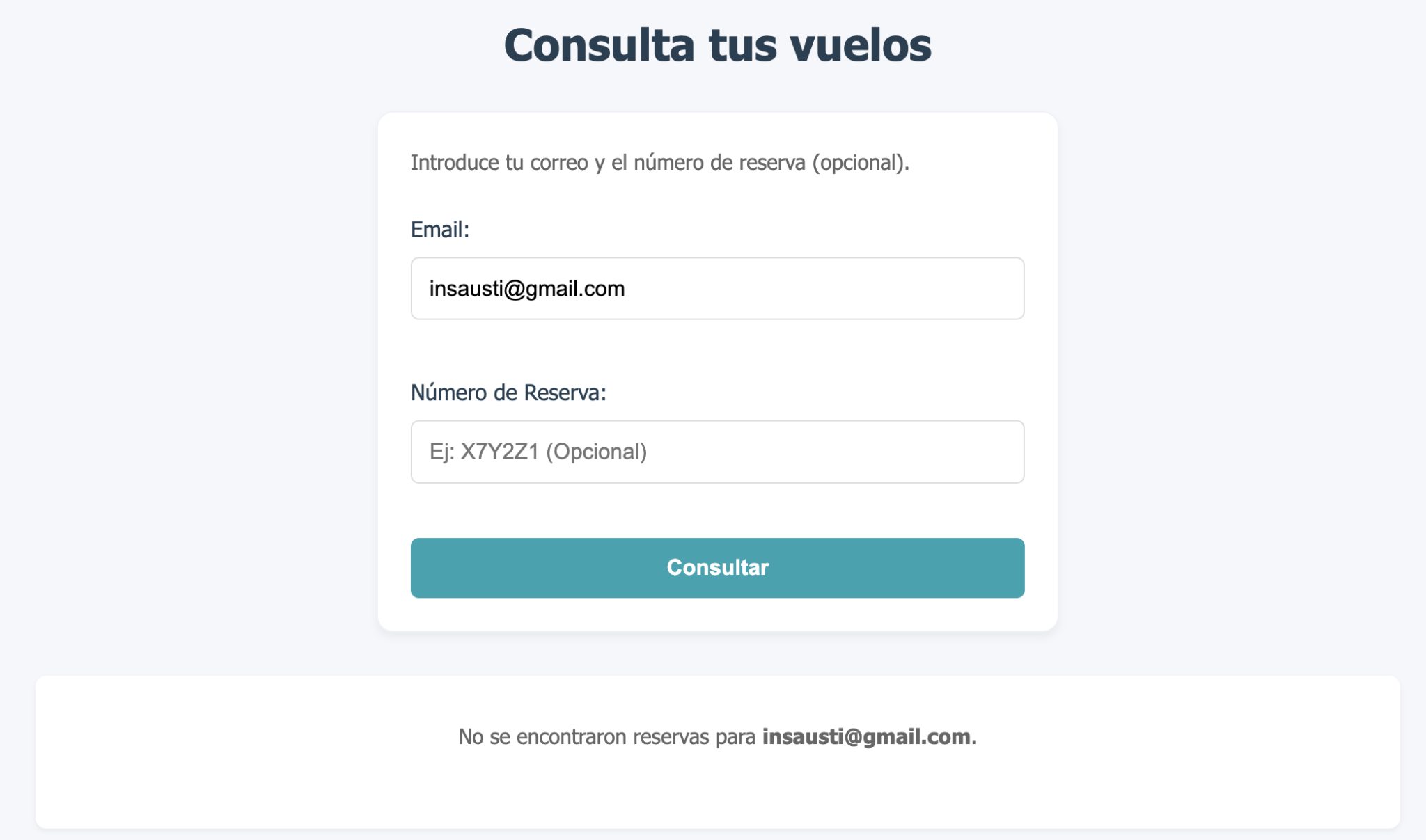
1. Si el usuario introduce un e-mail y pulsa el botón consultar, le aparecerán debajo todos los vuelos asociados a ese correo electrónico en una lista similar a la de la página Vuelos:



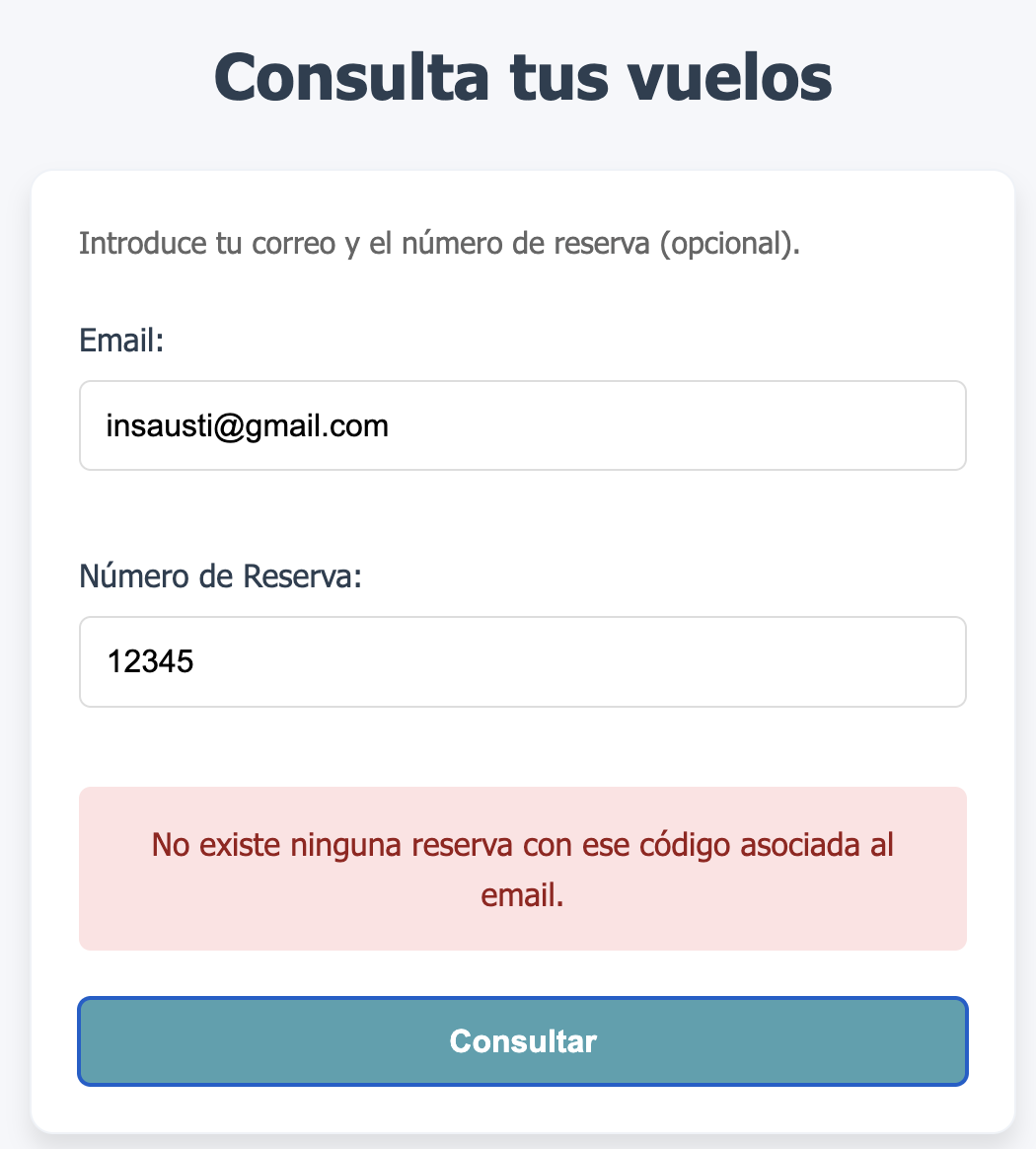
1. Si el usuario introduce un e-mail y un código de reserva (no código de vuelo) y existe esa reserva, le aparecerán una tabla con una fila con los datos del vuelo en una lista como la anterior:



1. Si el usuario introduce un e-mail que no tiene reservas deberá mostrar un mensaje indicándolo:



Del mismo modo que si se introduce un email y un número de reserva que no coinciden con ninguna reserva:



**Nota**: puedes mostrar el mensaje en cualquier parte de la página, no necesariamente como se muestra en las imágenes anteriores.

|  |  |
| --- | --- |
| **Total ejercicio - 2** | **8** |
| 1. **Estructura SPA y Router** | **3.8** |
| * 1. Se han creado correctamente las vistas VuelosView.vue, VueloDetalleViey.vue y ConsultaVueloView.vue. | 0,10 |
| * 1. Configuración correcta de rutas con Vue Router. | 0,10 |
| * 1. Enlazado correcto de los botones de la página HomeView. | 0,10 |
| * 1. Ruta paramétrica /vehiculos/:id funcionando | 0,20 |
| * 1. “Consultar vuelo” visible si hay algún vuelo reservado previamente (Pinia) | 0,40 |
| 1. **Arquitectura de estado con Pinia** |  |
| * 1. Store correctamente creado e integrado en la app | 0,25 |
| * 1. Carga de vuelos directamente en la store (y no desde el view) | 0,25 |
| * 1. Función que añade la reserva a la lista de reservas desde la store funciona | 0,50 |
| 1. **Carga y visualización de vuelos** |  |
| * 1. La carga de datos de los vuelos con Axios es correcta | 0,25 |
| * 1. Se muestran todos los datos de cada vuelo | 0,10 |
| * 1. La duración del vuelo se muestra en formato hora / minuto | 0,25 |
| * 1. Botón “Ver detalle” integrado en la fila y enlazado a la página de detalle | 0,15 |
| 1. **Filtros con propiedades computadas** |  |
| * 1. Filtro Identificador (no sensible a mayúsculas / minúsculas) funciona | 0,35 |
| * 1. Filtro de Origen y Destino cargado dinámicamente | 0,10 |
| * 1. Los elementos Origen y Destino no se repiten | 0,1 |
| * 1. Filtros Origen y Destino funcionan | 0,35 |
| * 1. Filtro tipo range limita vuelos con menor duración que el valor elegido | 0,35 |
| * 1. Cumple todos los filtros simultáneamente | 0,40 |
| * 1. Filtro implementado con propiedad computada | 0,40 |
| 1. **Vista Detalle de Vuelo** |  |
| * 1. Carga correcta según parámetro de ruta | 0,35 |
| * 1. Información completa del vuelo | 0,15 |
| * 1. Botón volver funcional | 0,15 |
| 1. **Lógica de reservas** |  |
| * 1. Comprobación de e-mail y número de pasajeros válidos | 0,10 |
| * 1. Etiqueta precio total se actualiza automáticamente con el número de pasajeros | 0,40 |
| * 1. Genera un código aleatorio para la reserva y lo muestra en la página | 0,40 |
| * 1. Reserva añadida correctamente al estado (Pinia) | 0,50 |
| * 1. Comprobación de que no existe una reserva con el mismo usuario y vuelo. | 0,25 |
| 1. **Consulta de vuelos** |  |
| * 1. Si el usuario introduce un e-mail, aparecen todos los vuelos de ese e-mail | 0,30 |
| * 1. Si el usuario introduce un e-mail y un número de reserva aparece el vuelo que coincide con el e-mail y la reserva. | 0,30 |
| * 1. Si no hay ninguna reserva asociada al e-mail se muestra un mensaje | 0,20 |
| * 1. Si no hay ninguna reserva asociada al e-mail+reserva se muestra un mensaje | 0,20 |