DESARROLLO WEB EN ENTORNO CLIENTE

PREPARACIÓN SEGUNDA EVALUACIÓN

1. Queremos desarrollar un script que simule el planetario del sistema solar, girando los planetas en sus órbitas a un radio y velocidad determinadas.

   Disponemos de una clase **`Planeta`** con los siguientes elementos:

Propiedad (pública)	Descripción
`id: entero`	Identificador del planeta
`nombre:cadena`	Nombre del planeta
`descripcion:cadena`	Descripción del planeta
`posicionX:entero`	Coordenada X de la posición actual del planeta. Inicialmente 0
`posicionY: entero`	Coordenada X de la posición actual del planeta. Inicialmente 0

Propiedad (privada)	Descripción
`posicion:entero`	Posición lineal del planeta. Se usa para actualizar la posición
`velocidad:entero`	Velocidad del planeta

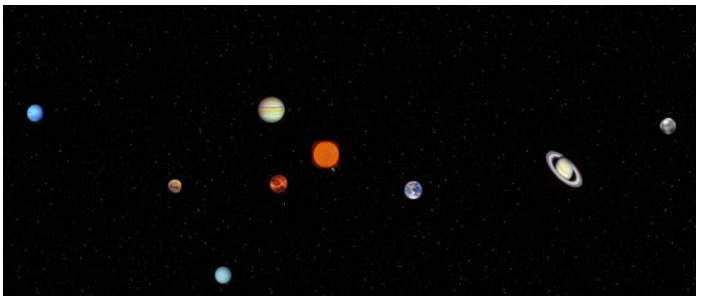
Método	Descripción
`mover(entero,entero)`	Mueve el planeta, actualizando la posición del mismo. La función recibe el ancho y el alto del tablero donde va a moverse

También disponemos del script **`planetas.php`** que devuelve un listado en JSON un listado de planetas con toda la información relativa a los mismos (el script no recibe ningún parámetro).

Al cargar el script:

- Se realizará una petición AJAX para obtener la información de los planetas.
- Se instanciarán los objetos **`Planeta`** que va a usar nuestro programa a partir de la información recibida.
- Se dibujarán los planetas en su posición inicial.

A continuación se pondrán los planetas en órbita. Para ello se simulará el movimiento refrescando cada 50 milisegundos la posición de los planetas. A la hora de calcular el movimiento de los planetas, pasaremos a la función de movimiento el tamaño interno de la ventana (variables **`window.innerWidth`** y **`window.innerHeight`**).



Por otra parte, queremos que cuando se sitúe el ratón sobre un planeta, se detenga el planetario para mostrar una capa al lado del planeta con el nombre e información sobre él. Al mover el ratón fuera del planeta, continuará el movimiento de los mismos.

2. La aplicación consiste en un buscador universal de vuelos.

Consideraciones generales:

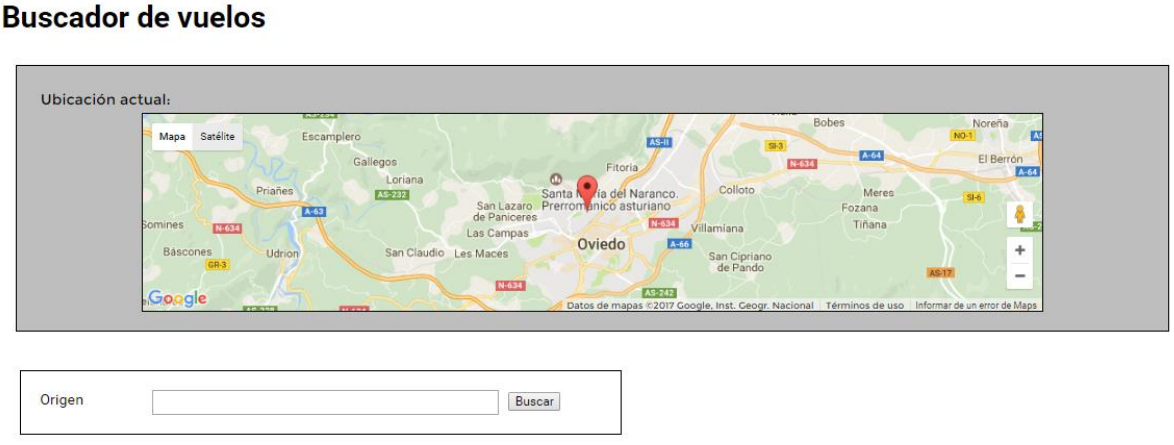
- Despliega la aplicación que se proporciona sobre un servidor Web instalado en tu equipo, respetando la estructura de directorios proporcionada. Se proporcionan los siguientes archivos:
  - Un script de creación de una base de datos dwec_08_act1_ejercicio2, donde se creará un usuario homónimo con derechos de acceso a la misma y 3 tablas (***airlines, airports*** y ***routes***).
  - Todo el código PHP necesario para implementar la aplicación.
  - Ficheros JS con librerías y funciones auxiliares.
  - Fichero HTML con código del formulario.
  - Ficheros CSS con los estilos usados en el formulario, además reglas de selección con estilos que se aplicarán a los elementos que se vayan creando.

  NOTA: Dado el tamaño del fichero SQL, para poder importar correctamente la BD necesitamos realizar los siguientes cambios
  - Archivo php.ini: upload_max_filesize = 8M.
    - En XAMPP, dicho archivo se encuentra en el directorio de instalación de XAMPP, carpeta php.
  - Archivo my.ini: max_allowed_packet = 16M
    - En XAMPP, dicho archivo se encuentra en el directorio de instalación de XAMPP, carpeta mysql\bin.

- No podrán modificarse los archivos HTML, CSS y JS proporcionados, si bien podrán añadirse nuevos archivos Javascript y añadir en el HTML las referencias a los mismos.

- Se utilizarán las funciones jQuery para manejo del documento (DOM), eventos y AJAX

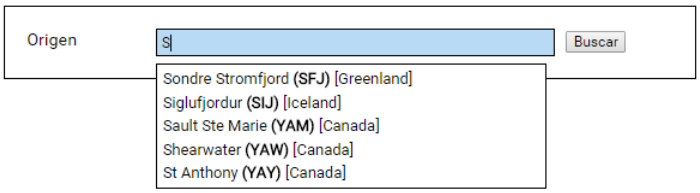
El aspecto inicial de la interfaz será el siguiente:



En la capa con id `informacion-posicion` se mostrará sobre un mapa de Google Maps[1] la ubicación actual del usuario, obtenida a partir de la API de Geolocalización.

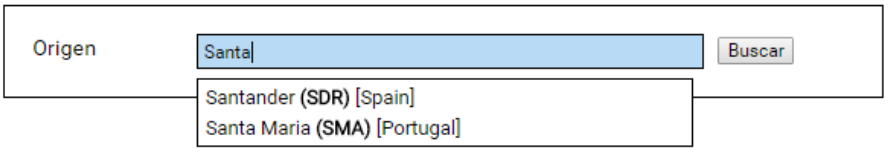
Al pulsarse una tecla sobre la caja de texto deberían mostrarse un listado los aeropuertos que comienzan con dicha letra. Para cada aeropuerto se mostrará el nombre del aeropuerto, su código IATA (entre paréntesis) y el país del mismo [entre corchetes]. Dicho listado deberá mostrase sobre la capa con id **`sugerencia-origen`**, que inicialmente está oculta.

[1] Para mostrar las coordenadas en el mapa habrá que hacer uso de la API de Google Maps.

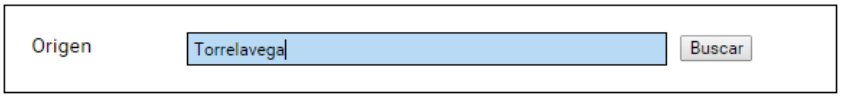


El listado se obtendrá a partir de una petición AJAX al script **`sugerencia_origen.php`**, que recibe por POST un parámetro llamado **origen** que contiene la cadena por la que comienza el aeropuerto. El script devuelve un archivo XML con el listado de aeropuertos, cuya sintaxis se detalla más adelante.

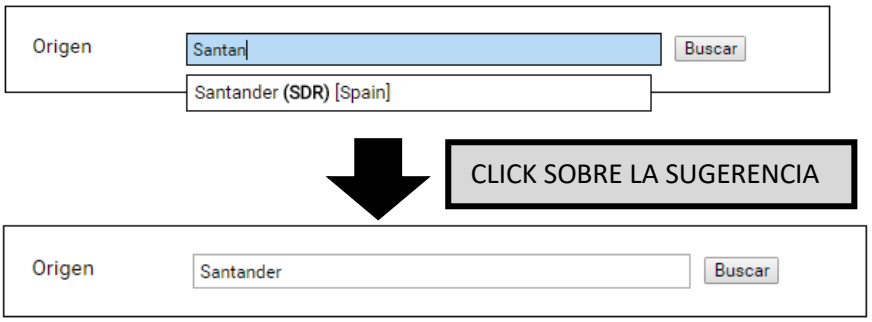
Al escribir nuevas letras o actualizar la letra introducida se actualizará el listado de sugerencias:



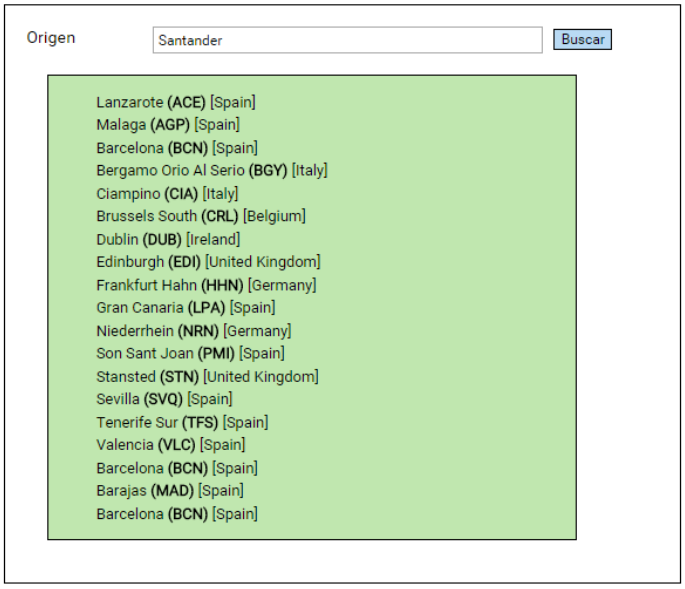
En caso de que no haya ninguna sugerencia el listado no se mostrará:



En caso de pulsar una sugerencia, se mostrará el nombre de la ciudad en la caja de texto (únicamente el nombre de ciudad):

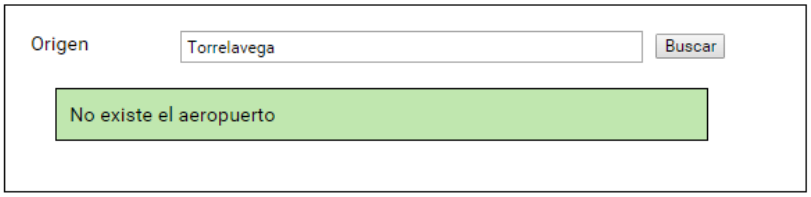


Al pulsar el botón de *Buscar* se mostrarán todos los aeropuertos para los cuales existen vuelos:

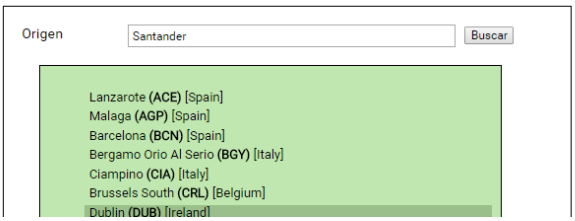


El listado se mostrará en la capa con id **`informacion-destinos`**, que inicialmente está oculto, y se obtendrá a través de una petición AJAX al script **`peticion_destinos.php`**, que recibe por GET el parámetro origen (que contiene el nombre del aeropuerto destino) y devuelve mediante XML un listado de aeropuertos con la misma sintaxis que la usada en **`sugerencia_origen.php`**.

En caso de no existir el aeropuerto cuyo nombre se ha introducido, se mostrará un mensaje indicativo:



Al pulsar sobre el nombre de un aeropuerto se mostrará en la capa con id **`informacion-vuelo`** la información referente al vuelo. Dicha información se obtendrá a través de una petición AJAX (hacerse usando las correspondientes funciones jQuery para el manejo de AJAX) al script **`info_vuelo.php`**. El script recibe como parámetros POST los nombres de os aeropuertos de origen y destino, y devuelve en formato JSON **un único objeto objeto** que representa algunos datos del vuelo: nombre de la compañía, país de la compañía y un entero que representa el número de enlaces necesarios [en caso de que el vuelo sea directo dicho número será 0].



En caso de pulsarse sobre otro aeropuerto de destino la información se actualizará.

Script PHP	Recibe	Devuelve
sugerencia_origen.php	**[POST]** Parámetro `origen`. Contiene la cadena a partir de la cual deseo buscar sugerencias.	Documento XML que contiene un listado de aeropuertos comenzando con la sugerencia [ver ejemplo]
peticion_destinos.php	**[GET]** Parámetro `origen`. Contiene el nombre del aeropuerto para el cual vamos a buscar vuelos.	Documento XML que contiene un listado de aeropuertos a los cuales se vuela desde el origen.
info_vuelo.php	**[POST]** Parámetro `origen`. Contiene el nombre del aeropuerto origen del vuelo. **[POST]** Parámetro `destino`. Contiene el nombre del aeropuerto destino del vuelo.	Documento JSON que representa información del vuelo [ver ejemplo]

Ejemplo de respuesta XML:

- En caso de existir aeropuertos que comienzan por la cadena que se proporciona (o coinciden con el nombre que se proporciona):

```
<aeropuertos>
      <aeropuerto>
            <nombre>Santander</nombre>
            <ciudad>Santander</ciudad>
            <pais>Spain</pais>
            <iata>SDR</iata>
  </aeropuerto>
  <aeropuerto>
            <nombre>Santa Maria</nombre>
            <ciudad>Santa Maria</ciudad>
            <pais>Portugal</pais>
            <iata>SMA</iata>
  </aeropuerto>
 </aeropuertos>
```

- En caso de no existir aeropuertos que comienzan por dicha cadena (o con dicho nombre):

```
 <aeropuertos>
       VACIO
</aeropuertos>
```

Ejemplo de respuesta JSON:

```
{ "aerolinea":"Ryanair",
  "pais":"Ireland",
 "paradas":"0" }
```

1. Queremos implementar una pequeña tienda con un carrito de la compra.

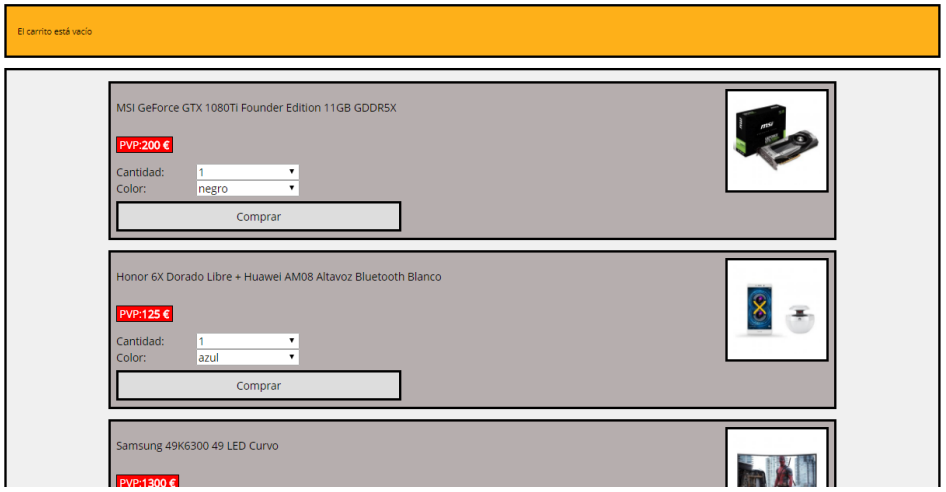
   Al cargar la página, se mostrarán todos los productos ofertados. Para ello se realizará una petición AJAX GET sin parámetros a la página **`productos.php`**, que devuelve en formato JSON un listado de productos. Cada producto del listado recibido tendrá las siguientes propiedades:

   - `id` (entero)
   - `titulo` (cadena)
   - `img` (cadena)
   - `pvp` (entero)
   - `existencias` (entero)
   - `colores` (array de cadenas)

   Por ejemplo:

```
{ "id": 0,
 "titulo": "MSI GeForce GTX 1080Ti Founder Edition 11GB GDDR5X",
 "img": "msi.jpg",
 "pvp": 200,
 "existencias": 8,
 "colores": [ "negro", "blanco"]}
```

   Para cada producto se mostrará su nombre, imagen asociada, un desplegable para seleccionar el número de unidades a comprar (cuyo máximo depende de las existencias), un desplegable con los colores disponibles, el precio y el botón *Comprar*.



Deberían seguirse las siguientes directrices para generar la página (se recomienda consultar el archivo **`estilos.css`** y usar las reglas descritas en el mismo):

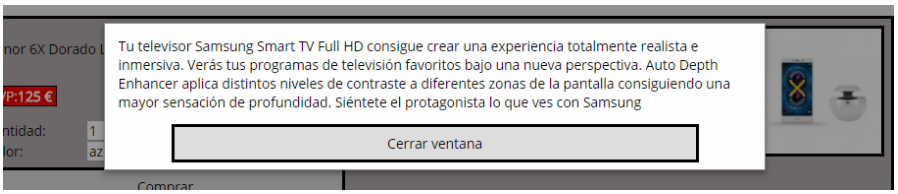
- Se creará una capa **`#carrito`** y otra capa **`#productos`**.
- La capa del carrito inicialmente solo tiene un párrafo con un mensaje.
- La capa de productos está formada por un listado de capas **`.articulo`** (una por cada producto leído) cada una de las cuales tiene dos capas **`.columna`**:
  - La primera columna contiene el nombre, precio, listados de cantidad y color y el botón para comprar.
  - La segunda columna contiene la imagen.

Al pulsar sobre el nombre de un producto o su imagen, se mostrará una ventana con información detallada sobre el producto. La información del producto también se obtiene con AJAX, haciendo una

petición GET al script **`productos.php`**. Dicho script recibe un parámetro llamado **`id`** con el id del producto y devuelve la información en formato JSON. Se recibirá un único objeto con las siguientes propiedades:

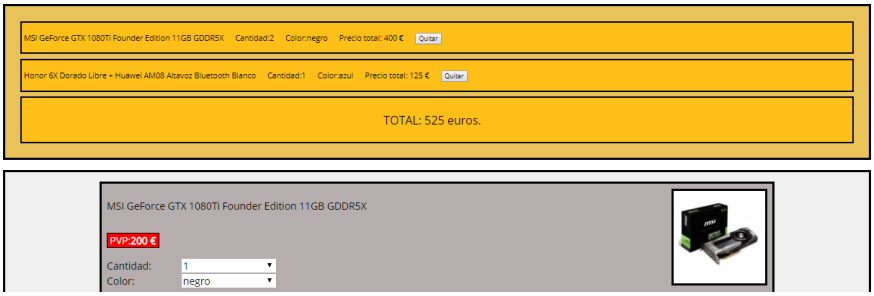
- `id` (entero)
- `titulo` (cadena)

Por ejemplo: `{ "id": 2, "info": "Tu televisor Samsung Smart TV..." }`



La información se cargará en la ventana modal **`#ventana-info`**, que se mostrará usando la propiedad CSS **`display:block`**. Al pulsar en el botón *Cerrar ventana* la ventana se ocultará usando la propiedad CSS **`display:none`**

Cada vez que se pulse en el botón *Comprar* de un producto, se añadirá el producto a un carrito de la compra. Automáticamente, el carrito debería mostrarse actualizado. Además, por cada línea de producto, debería haber un botón que permita eliminar el producto.



Para resolver este apartado, será obligatorio implementar una clase **`Carrito`** que tenga al menos un listado de productos y que proporcione métodos para al menos:

- Añadir producto al listado de productos.
- Eliminar producto del listado de productos.
- Calcular el coste actual del carrito

NOTA: Queda a criterio del alumno la utilización de una clase **`Producto`** si lo considera necesario.