



**Universitat**  
de les Illes Balears

## GRADO DE INGENIERÍA INFORMÁTICA

Tecnologías Multimedia

# Documentación Puertos Mallorca

Jaume Adrover Fernández  
[jaume.adrover3@estudiant.uib.cat](mailto:jaume.adrover3@estudiant.uib.cat)

Marc Cañellas Gomez  
[marc.canellas@estudiant.uib.cat](mailto:marc.canellas@estudiant.uib.cat)

Diego Bermejo Cabañas  
[diego.bermejo@estudiant.uib.cat](mailto:diego.bermejo@estudiant.uib.cat)

Joan Balaguer Llagostera  
[joan.balaguer2@estudiant.uib.cat](mailto:joan.balaguer2@estudiant.uib.cat)

23 de maig de 2023

# Índex

<b>1</b>	<b>Introducción</b>	<b>2</b>
<b>2</b>	<b>URL</b>	<b>2</b>
<b>3</b>	<b>Funcionalidades de la Web-App</b>	<b>2</b>
3.1	Home Page . . . . .	2
3.2	Página de un Puerto . . . . .	5
3.3	Página de Contacto . . . . .	7
<b>4</b>	<b>Herramientas y Librerías</b>	<b>8</b>
4.1	index.html . . . . .	8
4.1.1	Barra de Navegación . . . . .	8
4.1.2	Carousel . . . . .	9
4.1.3	Sección introductoria . . . . .	11
4.1.4	Mapa de puertos . . . . .	12
4.1.5	Barra de filtros . . . . .	13
4.1.6	Cards de puertos . . . . .	14
4.1.7	Footer . . . . .	15
4.2	puerto.html . . . . .	17
4.2.1	Nombre del puerto . . . . .	17
4.2.2	Descripción y Video . . . . .	17
4.2.3	Características i Tiempo . . . . .	18
4.2.4	Botones de Favoritos . . . . .	21
4.2.5	Galería de Imagenes . . . . .	22
4.2.6	Lugares de interés cercanos . . . . .	22
4.3	sobreNosotros.html . . . . .	23
<b>5</b>	<b>Ficheros JavaScript</b>	<b>25</b>
5.1	main.js . . . . .	26
5.2	port.js . . . . .	30
5.3	weather.js . . . . .	40
5.4	email.js . . . . .	41
<b>6</b>	<b>APIs utilizadas</b>	<b>41</b>
6.1	Google Maps . . . . .	41
6.2	OpenWeatherAPI . . . . .	43
6.3	EmailJS . . . . .	44
6.4	WebStorage . . . . .	45
<b>7</b>	<b>Web Semántica</b>	<b>46</b>
<b>8</b>	<b>Inclusión de media: SVG y video propios</b>	<b>48</b>
<b>9</b>	<b>Evaluación de la Web-App</b>	<b>48</b>
9.1	nibbler.insites.com . . . . .	49
9.1.1	Accesibilidad . . . . .	49
9.1.2	Experiencia . . . . .	50
9.1.3	Marketing . . . . .	51
9.1.4	Tecnología . . . . .	51
9.2	pagespeed.web.dev . . . . .	52
9.3	validator.schema.org . . . . .	52
<b>10</b>	<b>Conclusiones</b>	<b>53</b>

# 1 Introducción

En este documento se explicará el contenido de la web-app que hemos desarrollado de forma que se pueda entender como se ha desarrollado. Además, se mostrarán todas sus funcionalidades para que el usuario tenga en cuenta todo lo que puede hacer con la web-app. Por otro lado, se explicarán que herramientas se han usado para el desarrollo de esta, además de las librerías, APIs y extras utilizados para la composición gráfica de la página.

## 2 URL

Primero de todo se mostrará como se puede acceder a la web-app desarrollada. En nuestro caso, hemos comprado un servicio de hosting en la página [dondominio.com](http://dondominio.com) gracias a los códigos proporcionados por nuestro profesor de la asignatura. En nuestro caso, hemos elegido como nombre de la URL para acceder a nuestra web-app [puertosmallorca.com](http://puertosmallorca.com). Hemos elegido este nombre, ya que lo encontramos relativamente corto, conciso y muestra muy claramente lo que podrás encontrar en nuestra web-app. principalmente, lo hemos elegido siguiendo las recomendaciones que se nos especificaban en la página de contratación del hosting.

## 3 Funcionalidades de la Web-App

Una vez explicado como hemos contratado el servicio de hosting y podemos acceder a la web-app, mediante el link proporcionado en la sección anterior, pasaremos a explicar todas las funcionalidades que el usuario puede realizar con ella.

### 3.1 Home Page

Nada más entrar al link proporcionado, encontraremos una barra de navegación que se mantiene fija en la parte superior de la página. En la parte de la izquierda de esta, encontramos el logo y el nombre de la web-app y tanto el logo como el nombre, servirán para que el usuario pueda volver a la *homepage* des cualquier desde de la web-app con tan solo clicar encima de una de ellas. En la parte de la derecha de la barra de navegación, encontramos 3 botones para acceder a las diferentes funcionalidades que ofrece la página:

- **Ver Puertos:** este nos llevará más abajo de la *homepage*, donde podremos obtener información sobre los puertos de Mallorca registrados en la página.
- **Contacto:** nos llevará a una página donde el usuario podrá contactar con nosotros en caso de tener cualquier problema.



Figura 1: Barra de navegación de la web-app

Un poco más abajo de la barra de navegación, encontramos un carrusel de 3 fotos con textos encima de estas. En la primera de ellas se da la bienvenida al usuario, en la segunda, se muestra que puedes encontrar información sobre los puertos de mallorca y en la tercera que puedes crear tu plan de navegación. Con esto, damos a conocer de manera simple y concisa lo que el usuario puede hacer con nuestra web-app. El usuario se puede desplazar de forma manual por las fotos del carrusel, y también existe una flecha que apunta hacia abajo que nos llevará a la siguiente sección de la *homepage*.



Figura 2: Las 3 fotos del carrusel con sus respectivos textos

Una vez pasamos el carrusel de fotos, llegamos a la parte introductoria de la página, donde hay una pequeña descripción en forma de texto muy conocido sobre lo que se puede hacer en la página.

**SOBRE NOSOTROS**

**Bienvenido a nuestra pagina**

En esta página podrás obtener toda la información sobre los puertos de mallorca. Información de contacto, localización, valoraciones, numero de amarres...

Además, te damos la posibilidad de crear tu hoja de navegación para poder navegar por las aguas mallorquinas de la forma en que más te guste.

**Puertos increíbles**  
Tenemos registrados los principales puertos de mallorca y subiendo!

**Navega**  
Crea tu propio plan de navegación para disfrutar al máximo de tus calas y lugares de interés favoritos de mallorca

**Garantizado**  
Con nuestro plan seguro que que acabas descubriendo más de lo que pensabas!

Figura 3: Sección introductoria de la web-app

Más abajo, encontramos la primera funcionalidad de nuestra web-app. Se muestra un mapa con marcadores rojos encima de los puertos registrados en la página. El mapa es interactivo, ya que se ha realizado mediante la API de Google Maps (la especificación técnica se explica más adelante en la documentación). Permite al usuario clicar encima de uno de los marcadores y así poder ver el nombre de este y, en caso de que el usuario quiera, tendrá un link que lo redirigirá a la página con la información del puerto seleccionado.



Figura 4: Mapa interactivo de la web-app con todos los puertos disponibles en esta

Debajo del mapa, encontramos otra funcionalidad de la web-app. Mediante la barra de filtros, podemos tanto filtrar los puertos por sus características como ordenar los puertos buscados por una de sus características. De esta forma, el usuario podrá buscar los puertos con mayor capacidad, los

de mayor valoración, saber los puertos con capacidad mínima de 300 barcos y mostrar los puertos que estén en su grupo de favoritos.

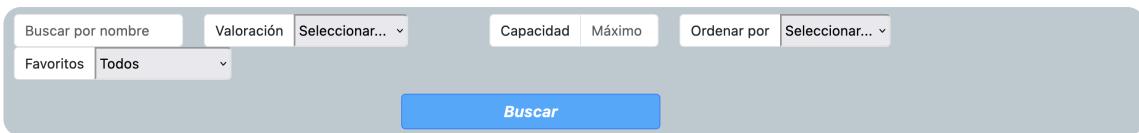


Figura 5: Barra de filtros

Obviamente, la barra de filtros se aplica a las **cards** de los puertos que se muestran más abajo. En estas, se muestra su capacidad y su valoración en estrellas. También, si clicamos encima de la foto de uno de los puertos, nos redirigirá a la página del puerto en concreto.

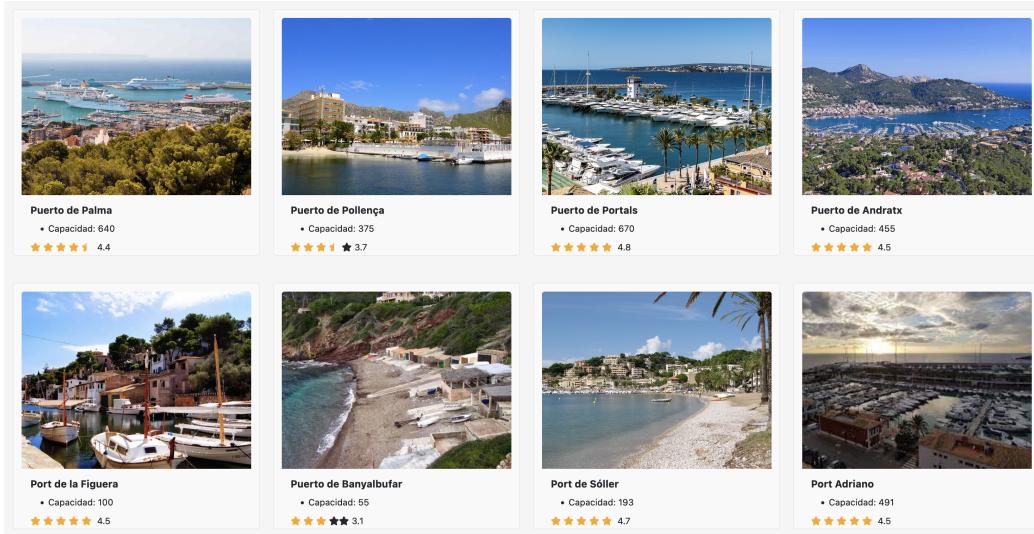


Figura 6: Ejemplos de **cards** donde se muestra una previsualización de los puertos

Para finalizar con la *homepage*, encontramos el footer que encontramos en todas las sub páginas de nuestra web-app. En este, se muestran los nombres de los creadores de la página con sus respectivos links a sus páginas de GitHub, LinkedIn e Instagram.



Figura 7: Footer de la web-app

### 3.2 Página de un Puerto

Cuando seleccionamos uno de los puertos, la web-app nos redirige a la página con información del puerto. Esta tiene diversas partes:

- **Título:** simplemente muestra el nombre del puerto seleccionado en grande en la parte superior de la página.



Figura 8: Sección donde se muestra el título del puerto (en este caso el de palma)

- **Descripción y video:** muestra el atributo de la descripción del puerto, además del video que tiene también como atributo.

A screenshot of a web page showing the description of Puerto de Palma. On the left is a text box with descriptive text about the port's history and services. On the right is a video player showing a video of the port's skyline.

Figura 9: Sección donde se muestra la descripción del puerto junto al video de youtube del puerto

- **Características:** Muestra de forma concisa las características del puerto seleccionado. Se muestra su capacidad, su horario de apertura, si se permite fumar o no, la dirección, el correo electrónico, el teléfono y la valoración del puerto.

<b>Capacidad de Barcos:</b>	375
<b>Permite Fumar:</b>	Si
<b>Horario entre semana:</b>	Mo-Fr 06:30-20:00
<b>Horario fin de semana:</b>	Sa-Su 06:30-13:30
<b>Calle:</b>	Passeig Saralegui
<b>Localidad:</b>	Pollensa
<b>Código Postal:</b>	07470
<b>Correo:</b>	info@portpollensa.com
<b>Teléfono:</b>	+34 971 86 46 35
<b>Valoración:</b>	3.7 Número de valoraciones: 131

Figura 10: Sección donde se muestran los atributos propios de ese puerto en concreto

- **Tiempo:** en este apartado se muestra el tiempo que hace en ese puerto en concreto.

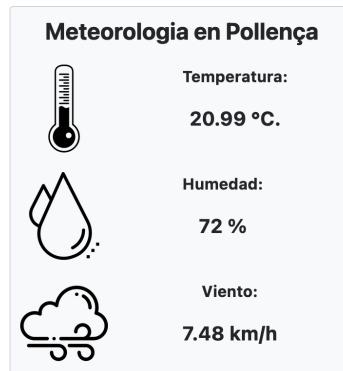


Figura 11: Sección donde se muestran el tiempo en el puerto

- **Botones para añadir favoritos:** nuestra web-app, ofrece la posibilidad al ususario de guardar sus puertos favoritos para posteriormente consultarlos en la *homepage* mediante la barra de filtros. En esta parte, hemos añadido 2 botones. Uno para añadir el puerto a tus puertos favoritos y otro para eliminarlo de favoritos.



Figura 12: Botones para añadir y eliminar puertos favoritos

- **Galería de Imágenes:** se muestran las imágenes que tenga como atributo el puerto seleccionado en el fichero json. Estas se pueden ir pasando mediante las flechas situadas a los lados de las imágenes.



Figura 13: Footer de la web-app

- **Lugares de interés cercanos:** sección en la cual se mostraran las cafeterias, restaurantes y otros puertos cercanos al puerto seleccionado por el ususario. Contendrá dos partes principales. Por un lado, un mapa en el cual se mostraran en este los restaurantes i cafeterias mas proximos al puerto y una segunda sección donde se mostraran unas **cards** con los puertos más cercanos.

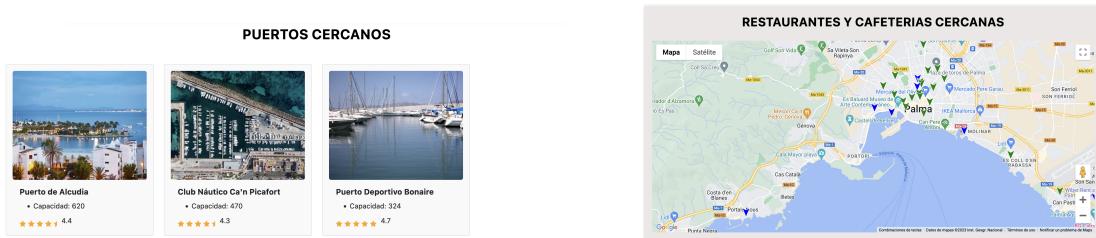


Figura 14: Como se muestran los puertos cercanos y el mapa con los restaurantes y cafeterías

### 3.3 Página de Contacto

Para finalizar con las funcionalidades de la web-app, llegamos a la página donde el usuario puede contactar con nosotros, los creadores para aportar sugerencias o resolver dudas que le puedan surgir durante la utilización de la web-app. Lo que nos encontramos en la sección llamada *Sobre Nosotros* es lo siguiente:

- **Video:** video explicativo donde nos presentamos al usuario como creadores de la página.
- **Sección de Contacto:** en esta sección, el usuario tiene la posibilidad de enviarnos un correo, pudiendo apuntar el tipo de asunto del correo, su nombre, su correo y el mensaje que nos quiere hacer llegar. En caso que el usuario quiera satisfacer sus inquietudes sobre algo relacionado con la web-app, podrá hacerlo mediante este formulario.

The contact form is titled 'Contáctanos'. It contains the following fields:

- Nom**: Input field labeled 'Escríu el teu nom aquí'
- Email**: Input field labeled 'Escríu el teu email'
- No compartiré el teu correu mai**: A checkbox labeled 'Categoría:' with the option 'Dudas' selected.
- Descripció**: Input field labeled 'Escríu el que ens vulguis fer saber'
- Envia**: A blue 'Send' button at the bottom.

Figura 15: Sección en la qual el usuario nos puede contactar

## 4 Herramientas y Librerías

Una vez se han explicado todas las funcionalidades que el usuario puede realizar en nuestra web-app, pasaremos a la explicación técnica de como se han implementado todas esas funcionalidades. Iremos revisando una por una las componentes de todas las páginas de la web-app y explicando que herramientas y librerías se han usado para su desarrollo.

### 4.1 index.html

El archivo `index.html` contiene el código HTML de la *homepage*. El contenido funcional de esta parte de la web-app ya se ha explicado a nivel de usuario, en esta sección se describirá como se han desarrollado esas funcionalidades a nivel técnico.

#### 4.1.1 Barra de Navegación

Para la creación de la barra de navegación, se ha usado el framework [Bootstrap](#). Primero, hemos definido un objeto de la clase `container-fluid` el cual contendrá todo lo referente a la barra de navegación. Por otro lado, hemos añadido atributos a este contenedor para que el aspecto final de la barra de navegación sea el deseado:

- `px-0`: elimina los márgenes horizontales (padding) del elemento, lo que significa que el contenido del elemento se extiende hasta los bordes laterales del contenedor.
- `d-flex`: define el elemento como un contenedor flexible, lo que significa que sus elementos hijos se pueden ajustar dinámicamente para adaptarse a diferentes tamaños de pantalla.
- `align-items-center`: alinea verticalmente los elementos hijos del contenedor, centrándolos en el eje vertical.
- `fixed-top`: fija el elemento en la parte superior de la pantalla, de modo que permanece en su posición incluso cuando el usuario se desplaza hacia abajo en la página.

Una vez sabemos como hemos definido el contendor, vamos con los elementos que se encuentran dentro de este. El principal elemento es un objeto de la clase `navbar navbar-expand-lg`. Esta, se utiliza para crear una barra de navegación responsiva en un sitio web (esto último es lo que indica el `expand-lg`). Dentro de esta, encontramos ya los botones que contienen la barra de navegación:

- **Logo:** se ha realizado mediante la clase `navbar-brand`, perteneciente a Bootstrap. Dentro de este se encuentra la imagen del mismo.
- **Nombre de la página:** el nombre de *Puertos Mallorca* contenido en la barra de navegación, se encuentra dentro del objeto del logo. Este es otro objeto de la clase `navbar-brand`.
- **Botón de navegación:** este botón solo aparece en caso de que la pantalla donde se muestra la web-app no sea lo suficientemente grande como para mostrar todos los botones de la barra de navegación. En caso de suceder esto, aparecerá un botón con el cual podremos acceder a los botones que explicamos a continuación. Esto se ha hecho gracias a la clase `navbar-toggler` de Bootstrap. referenciando al objeto al que influye con el parámetro `aria-controls`. Mediante este, indicamos que el objeto con `id = navbarSupportedContent` se le aplicará la contracción de los botones cuando la pantalla no se alo suficientemente grande.
- **Botones de funcionalidades:** los botones de *Ver Puertos* y *Contacto* son objetos del tipo `nav-item` (items de una barra de navegación) de bootstrap y están contenidos dentro de un objeto `collapse navbar-collapse`. Este objeto permite la contracción de los botones que hemos explicado en el apartado anterior.

A continuació se muestra el codigo referente a la barra de navegacin:

```
<div class="container-fluid px-0 d-flex align-items-center fixed-top">
    <!-- Navegation bar -->
    <nav class="navbar navbar-expand-lg" data-bs-theme="light" id="nvar">
        <div class="container-fluid">
            <!-- LOGO -->
            <a class="navbar-brand ms-3" href="index.html" id="contenedorLogo">
                
                <a class="navbar-brand mb-0 h1" href="#index.html">
                    Puertos mallorca</a>
            </a>
            <!-- END LOGO -->
            <button class="navbar-toggler" type="button"
                data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent"
                aria-controls="navbarSupportedContent"
                aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarSupportedContent">
                <ul class="navbar-nav ms-auto mb-2 mb-lg-0">
                    <li class="nav-item me-3">
                        <a class="nav-link" href="#contenedorMapa">
                            Ver puertos</a>
                    </li>
                    <li class="nav-item me-3">
                        <a class="nav-link" href="sobreNosotros.html">
                            Sobre Nosotros</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
    <!-- END Navegation bar -->
</div>
```

#### 4.1.2 Carousel

El deslizable con fotografias de los puertos, se ha realizado cogiendo como base la tercera de las [plantillas](#) que proporciona la pgina de bootstrap para hacer estos carousels. Partiendo de esa base, nosostros hemos añadido las captions y la flecha que permite al ususario ir ms abajo de la pgina. Las captions se crean con objetos de la clase `carousel-caption` y los botones para ir ms abajo de la pgina, son objetos de la classe `btn btn-down` de bootstrap. Estos nos permiten crear un boton con forma de flecha como la que aparece en la web-app y mediante un `href` en su interior, linkeado con la parte inductoria de la pgina, hacemos que cuando cliquemos encima de este nos lleve ms abajo. A continuacin se muestra el cdigo referente al carrousel de nuestra web-app:

```
<!-- CARROUSEL-->
<div id="carouselInit" class="carousel slide mt-0" data-bs-ride="carouselInit">
    <div class="carousel-indicators">
        <button type="button" data-bs-target="#carouselInit" data-bs-slide-to="0"
            class="active" aria-current="true" aria-label="Slide 1"></button>
        <button type="button" data-bs-target="#carouselInit" data-bs-slide-to="1">
```

```

        aria-label="Slide 2">></button>
    <button type="button" data-bs-target="#carouselInit" data-bs-slide-to="2"
        aria-label="Slide 3">></button>
</div>
<div class="carousel-inner">
    <div class="carousel-item active">
        
        <div class="carousel-caption" id="carousel-text1">
            <h1>Bienvenido a Puertos Mallorca!</h1>
            <p>Una guia completa de todos los puertos de la isla</p>
            <button id="btnDown1" type="button" class="btn btn-down">
                <a href="#intro" id="link1">
                    <i class="fas fa-chevron-down"></i></a></button>
            </div>
        </div>
        <div class="carousel-item">
            
            <div class="carousel-caption" id="carousel-text2">
                <h1>Descubre todo lo que no sabias!</h1>
                <p>Navega entre los puertos mejor valorados de la isla</p>
                <button id="btnDown2" type="button" class="btn btn-down">
                    <a href="#intro" id="link2">
                        <i class="fas fa-chevron-down"></i></a></button>
                </div>
            </div>
            <div class="carousel-item">
                
                <div class="carousel-caption" id="carousel-text3">
                    <h1>Crea tu plan de navegación!</h1>
                    <p>Podras crear tu plan para navegar por las costas
                    mallorquinas</p>
                    <button id="btnDown3" type="button" class="btn btn-down">
                        <a href="#intro" id="link3">
                            <i class="fas fa-chevron-down"></i></a></button>
                </div>
            </div>
        </div>
        <button class="carousel-control-prev" type="button"
            data-bs-target="#carouselInit" data-bs-slide="prev">
            <span class="carousel-control-prev-icon"
            aria-hidden="true">></span>
            <span class="visually-hidden">Previous</span>
        </button>
        <button class="carousel-control-next" type="button"
            data-bs-target="#carouselInit" data-bs-slide="next">
            <span class="carousel-control-next-icon"
            aria-hidden="true">></span>
            <span class="visually-hidden">Next</span>
        </button>
    </div>
<!-- END CAROUSEL-->
```

#### 4.1.3 Sección introductoria

Este apartado contiene dos secciones principales: una sección de introducción con una imagen y un texto descriptivo, y una sección de tres columnas con información destacada sobre los puertos de Mallorca.

En la sección de introducción, hemos hecho un contenedor de HTML div con la clase `site-section` y un identificador `intro`. Dentro de este contenedor, hay otro contenedor de `div` con la clase `container` y un contenedor de fila de div con la clase `row`. Esta fila contiene dos columnas de div con las clases `col-md-6` de Bootstrap. En la primera columna, hay una imagen con la clase `img-fluid`. En la segunda columna, hay un título con la clase `text-black` y un subtítulo con la clase `text-primary`.

En la sección de tres columnas, encontramos un contenedor de div con la clase `py-5`, un contenedor de div con la clase `container` y un contenedor de fila de div con la clase `row`. Dentro de esta fila, hay tres columnas de div con las clases `col-md-6` `col-lg-4` también de bootstrap. Cada columna contiene un contenedor de div con la clase "puertosIncreibles"(clase que luego usamos para referencial algunos objetos en css) que contiene un ícono en un contenedor de `span` con la clase `circulo` y un título de ancla con la clase `tituloIntroduccion`. En resumen, esta sección presenta una breve introducción a la página y destaca tres características principales de la web relacionadas con los puertos de Mallorca.

Para la creación de esta sección introductoria, nos hemos inspirado en una sección similar que encontramos en [esta plantilla](#).

```
<!-- Sección introductoria a la página -->
<div class="site-section" id="intro">
    <div class="container">
        <div class="row">
            <div class="col-md-6">
                
            </div>
            <div class="col-md-6">
                <span class="text-serif text-primary">Sobre nosotros</span>
                <h3 class="text-black">Bienvenido a nuestra pagina</h3>
                <p class="texto">En esta pagina podrás obtener toda la
                    información sobre los puertos de mallorca. Información
                    de contacto, localización, valoraciones, numero
                    de amarres...</p>
                <p class="texto">Además, te damos la posibilidad de crear tu hoja
                    de navegación para poder navegar por las aguas mallorquinas
                    de la forma en que más te guste.</p>
            </div>
        </div>
    </div>
</div>

<div class="py-5">
    <div class="container">
        <div class="row">
            <div class="col-md-6 col-lg-4">
                <div class="puertosIncreibles">
                    <span class="circulo">
                        <i class="fas fa-ship"></i>
                    </span>
                </div>
            </div>
        </div>
    </div>
</div>
```

```

<h3><a class="tituloIntroduccion" href="#contenedorMapa">
Puertos increibles</a></h3>

<p class="texto">Tenemos registrados los principales
puertos de mallorca y subiendo!</p>
</div>
</div>
<div class="col-md-6 col-lg-4">
<div class="puertosIncreibles">
<span class="circulo">
    <i class="fas fa-anchor"></i>
</span>
<h3><a class="tituloIntroduccion" href="#contenedorMapa">
Navega</a></h3>
<p class="texto">Crear tu propio plan de navegación
para disfrutar al máximo de tus calas y lugares
de interés favoritos de mallorca</p>
</div>
</div>
<div class="col-md-6 col-lg-4">
<div class="puertosIncreibles">
<span class="circulo">
    <i class="fas fa-thumbs-up"></i>
</span>
<h3><a class="tituloIntroduccion" href="#contenedorMapa">
Garantizado</a></h3>
<p class="texto">Con nuestro plan seguro que que acabas
descubriendo más de lo que pensabas!</p>
</div>
</div>
</div>
</div>
<!-- END Sección introductoria a la página -->
```

#### 4.1.4 Mapa de puertos

Para la creación del mapa, hemos usado la API que proporciona Google de su servicio Google Maps. Con esta, podemos mostrar un mapa de google maps interactivo, ademas de los marcadores de los puertos que tenemos registrados en la pagina.

El mapa se encuentra contenido dentro de un objeto de la clase `container-fluid` de Bootstrap. Dentro de este, hemos añadido el `div` correspondiente al mapa, el cual le hemos añadido el identificador `map`. En cuanto al HTML, esto sería todo lo que necesitamos para crear el mapa. Siguiendo la documentación que proporciona Google sobre como crear los mapas, hemos implementado el fichero de JavaScript adjuntado en el proyecto llamado `mapa.js`. Este contiene las instrucciones necesarias para la creación del mapa en la web-app (Estas instrucciones concretas se explicaran más adelante en el documento, así como las instrucciones que usamos para la lectura y obtención de la información del fichero json).

El código html con el que generamos el mapa interactivo és el siguiente:

```

<!-- MAP-->
<div class="container-fluid" id="contenedorMapa">
    <div class="container" id="titolMapa">
        <h2 class="text-center">Puertos de Mallorca</h2>
    </div>
    <div id="map"></div>
</div>
<!--END MAP-->

```

#### 4.1.5 Barra de filtros

Para la barra de filtros y ordenación, también hemos usado el framework Bootstrap. Por tanto, todas las clases que se usan para los elementos de la barra de filtros son pertenecientes a este.

Toda la barra de filtros está contenida dentro de un elemento de la clase `container-fluid` para hacer que el contenedor ocupe todo el ancho de la pantalla. Dentro de este contenedor, hay un elemento `row` con la clase `filterSearch` y `align-items-center`, lo que indica que se trata de una fila que contiene elementos centrados verticalmente. Además, hemos utilizado la clase `py-2` para agregar un espacio vertical a la fila. Dentro de la fila, hay cuatro columnas definidas usando la clase `col-xs-12 col-sm-6 col-md-3`. Estas columnas son para los elementos de búsqueda por nombre, valoración, capacidad y ordenar por. Dentro de cada columna, se utiliza la clase `input-group` para agregar estilos a los elementos de entrada de texto y select. Cada uno de estos elementos tiene una etiqueta de texto que describe la entrada que se espera.

El botón de búsqueda está dentro de una columna de 12 columnas y utiliza la clase `d-flex justify-content-center` para centrarlo horizontalmente. Se le da un identificador `botoCerca` (que usaremos posteriormente en los scripts de JavaScript) y se le asigna la clase `btn btn-primary mt-3` para darle un estilo de botón. Además, se utiliza la función `PortSearch()` para cuando se hace clic en el botón. Esta función se explicará junto al contenido del fichero `main.js`.

El código explicado anteriormente es el siguiente:

```

<!-- BARRA DE FILTROS-->
<div class="container-fluid">
    <div class="row filterSearch align-items-center py-2">
        <div class="col-xs-12 col-sm-6 col-md-2" id="textNom">
            <div class="input-group">
                <input type="text" id="form-name" class="form-control"
                    placeholder="Buscar por nombre">
            </div>
        </div>
        <div class="col-xs-12 col-sm-6 col-md-3">
            <div class="input-group">
                <div class="input-group-prepend">
                    <label class="input-group-text"
                        for="filtro-valoracion">Valoración</label>
                </div>
                <select class="custom-select ml-2" id="filtro-valoracion">
                    <option selected>Seleccionar...</option>
                    <option value="5">5 estrellas</option>
                    <option value="4">4-5 estrellas</option>
                    <option value="3">3-4 estrellas</option>
                    <option value="2">2-3 estrellas</option>
                </select>
            </div>
        </div>
    </div>
</div>

```

```

        <option value="1">1-2 estrellas</option>
        <option value="0">Sin valoración</option>
    </select>
</div>
</div>
<div class="col-xs-12 col-sm-6 col-md-2">
    <div class="input-group">
        <div class="input-group-prepend">
            <label class="input-group-text" for="filtro-capacidad">
                Capacidad</label>
        </div>
        <input type="text" class="form-control"
            id="filtro-capacidad-max" placeholder="Máximo">
    </div>
</div>
<div class="col-xs-12 col-sm-6 col-md-3">
    <div class="input-group">
        <div class="input-group-prepend">
            <label class="input-group-text" for="filtro-ordenar">
                Ordenar por</label>
        </div>
        <select class="custom-select" id="filtro-ordenar">
            <option selected>Seleccionar...</option>
            <option value="nombre">Nombre</option>
            <option value="valoracion">Valoración</option>
            <option value="capacidad">Capacidad</option>
        </select>
    </div>
</div>
<div class="col-xs-12 col-sm-6 col-md-3">
    <div class="input-group">
        <div class="input-group-prepend">
            <label class="input-group-text" for="filtro-favoritos">
                Favoritos</label>
        </div>
        <select class="custom-select" id="filtro-favoritos">
            <option selected>Todos</option>
            <option value="Si">Mostrar Favoritos</option>
        </select>
    </div>
</div>
<div class="col-12 d-flex justify-content-center">
    <button id="botoCerca" class="btn btn-primary mt-3"
        onclick="PortSearch()" type="button">Buscar</button>
</div>
</div>
</div>
<!-- END BARRA DE FILTROS--&gt;</pre>

```

#### 4.1.6 Cards de puertos

Justo debajo de la barra de filtros, encontramos los puertos que podemos filtrar e ordenar mediante esta. Esta sección se encuentra encapsulada en un `container-fluid` de Bootstrap, de los cuales

ya hemos hablado en esta documentación. Dentro de este contenedor, encontramos varias filas de objetos de la clase `card`. Las filas las definimos mediante la clase `row equal-width` y las columnas de estas filas con la clase `col-md-3`.

Por lo que respecta a los recuadros de los puertos, el framework de Bootstrap tiene una clase llamada `Card`, con la cual puedes crear recuadros con una foto y una pequeña sección de texto. Este tipo de objetos son los que hemos elegido para mostrar una previsualización de los puertos registrados en la página.

La estructura que siguen todas las `cards` es la siguiente:

- El primer elemento es una etiqueta `a` que envuelve una imagen (`img`) y que sirve como enlace a la página del puerto al que pertenezca la `card`. Mediante este enlace, el usuario podrá navegar y ver la información de diferentes puertos.
- El segundo elemento es un `div` con la clase `card-body` que contiene el contenido principal de la card. Dentro de este, mostraremos la información textual del puerto (nombre, capacidad y valoración).
- Dentro del `div` con la clase `card-body`, hay un elemento `h5` con la clase `card-title` que muestra el título de la card (en este caso será el nombre del puerto).
- También hay una lista no ordenada (`ul`) con un solo elemento (`li`) que muestra la capacidad del puerto.
- Finalmente, hay un `div` con la clase `rating` que contiene estrellas de valoración de la clase `fa fa-star` y un párrafo (`p`) que muestra la valoración promedio del puerto.

Todos estos tipos de objetos de bootstrap, los generamos mediante el fichero JavaScript llamado `port.js`. El contenido de este se mostrará más adelante, pero básicamente generamos el código html del `div` con nombre `contenedorPrepuertos` medida que vamos leyendo el contenido del fichero json. De esta forma, solo creamos los objetos de `bootstrap` necesarios segun el número de puertos que haya en el fichero. Se crearan los objetos html de forma dinámica.

El código html con el que generamos la sección de `cards` de puertos es el siguiente:

```
<!--DISPLAY ALL PORTS-->
<div class="container-fluid" id="contenedorPrePuertos">
    <!-- FILA PUERTOS-->
</div>
<!-- END DISPLAY ALL PORTS-->
```

#### 4.1.7 Footer

Finalmente, en la `homepage` encontramos el footer. Esta sección comienza con una etiqueta `<footer>` que tiene una clase `bg-dark text-light py-5`, lo que significa que el fondo del pie de página será oscuro y el texto será claro. Además, el padding superior e inferior de la sección será de 5 píxeles.

Dentro del pie de página hay un contenedor de la clase `container` que contiene una fila (`row`) con cuatro columnas (`col-md-3 mb-4 mb-md-0`), cada una con información de un miembro del equipo.

Cada columna comienza con un encabezado (`fw-bold mb-2`) que muestra el nombre del miembro del equipo y luego se muestran tres iconos que representan los enlaces a los perfiles de cada miembro en Github, Linkedin e Instagram respectivamente. Los enlaces se abren en una nueva ventana (`target="_blank"`) y no se les permite la seguridad adicional de referencias (`rel="noopener`

noreferrerer").

El código html que general nuestro footer és el siguiente:

```
<!-- Footer -->
<footer class="bg-dark text-light py-5">
  <div class="container">
    <div class="row">
      <div class="col-md-3 mb-4 mb-md-0">
        <h4 class="fw-bold mb-2">Joan Balaguer</h4>
        <a href="https://github.com/Joani13the13the13" target="_blank"
           rel="noopener noreferrer" class="text-light">
          <i class="fab fa-github"></i></a>GitHub<br>
        <a href="https://www.linkedin.com/in/joan-balaguer-llagostera-50839a269/" target="_blank" rel="noopener noreferrer" class="text-light">
          <i class="fab fa-linkedin"></i></a>Linkedin<br>
        <a href="https://twitter.com/JoanBalaguer13" target="_blank"
           rel="noopener noreferrer" class="text-light">
          <i class="fab fa-twitter"></i></a>Joan Balaguer <br>
        <a href="https://www.instagram.com/joan_balaguer/" target="_blank"
           rel="noopener noreferrer" class="text-light">
          <i class="fab fa-instagram"></i></a>Instagram
        </div>
      <div class="col-md-3 mb-4 mb-md-0">
        <h4 class="fw-bold mb-2">Diego Bermejo</h4>
        <ul class="list-unstyled">
          <a href="https://github.com/diegofes18" target="_blank"
             rel="noopener noreferrer"
             class="text-light"><i class="fab fa-github"></i></a>GitHub<br>
          <a href="https://www.linkedin.com/in/diego-bermejo-caba%C3%B1as-742393216/" target="_blank" rel="noopener noreferrer"
             class="text-light"><i class="fab fa-linkedin"></i></a>Linkedin<br>
          <a href="https://twitter.com/diegofes_18" target="_blank"
             rel="noopener noreferrer"
             class="text-light"><i class="fab fa-twitter"></i></a>Twitter<br>
          <a href="https://www.instagram.com/diegofes_18/" target="_blank"
             rel="noopener noreferrer"
             class="text-light"><i class="fab fa-instagram"></i></a>Instagram
        </ul>
      </div>
      <div class="col-md-3 mb-4 mb-md-0">
        <h4 class="fw-bold mb-2">Jaume Adrover</h4>
        <a href="https://github.com/jaaumeadrover" target="_blank"
           rel="noopener noreferrer"
           class="text-light"><i class="fab fa-github"></i></a>GitHub<br>
        <a href="https://www.linkedin.com/in/jaume-adrover-fern%C3%A1ndez-1943411b3/" target="_blank" rel="noopener noreferrer"
           class="text-light"><i class="fab fa-linkedin"></i></a>Linkedin<br>
        <a href="https://twitter.com/jaaumeadrover" target="_blank"
           rel="noopener noreferrer"
           class="text-light"><i class="fab fa-twitter"></i></a>Twitter<br>
        <a href="https://www.instagram.com/jaaumeadrover/" target="_blank"
           rel="noopener noreferrer"
           class="text-light"><i class="fab fa-instagram"></i></a>Instagram
      </div>
    </div>
  </div>
</footer>
```

```

</div>
<div class="col-md-3 mb-4 mb-md-0">
    <h4 class="fw-bold mb-2">Marc Cañellas</h4>
    <a href="https://github.com/MarcCanellasG" target="_blank"
        rel="noopener noreferrer"
        class="text-light"><i class="fab fa-github"></i></a>GitHub<br>
    <a href="https://www.linkedin.com/in/marc-ca%C3%B1ellas-74a629255/" target="_blank" rel="noopener noreferrer"
        class="text-light"><i class="fab fa-linkedin"></i></a>Linkedin<br>
    <a href="https://twitter.com/marccg6" target="_blank"
        rel="noopener noreferrer"
        class="text-light"><i class="fab fa-twitter"></i></a>Twitter<br>
    <a href="https://www.instagram.com/maaroccanellas/" target="_blank"
        rel="noopener noreferrer"
        class="text-light"><i class="fab fa-instagram"></i></a>Instagram
</div>
</div>
</div>
</footer>
<!--END FOOTER-->

```

## 4.2 puerto.html

En este fichero HTML está contenido las estructuras para poder mostrar al usuario la información sobre un puerto en concreto. En las explicaciones siguientes, se obvian la barra de navegación y el footer (ya explicados en la *homepage*). Cabe mencionar que en este fichero la mayoría de atributos que ahora mostraremos no tienen su contenido escrito como tal. Lo que tienen es su estructura html con un `<span></span>` que tiene un identificador. Mediante este id, en el fichero `port.js` le insertamos el contenido cuando leemos la información sobre el puerto correspondiente (en el apartado de archivos JavaScript mostraremos los detalles de la implementación que hace esto posible).

### 4.2.1 Nombre del puerto

El nombre del puerto simplemente de trata de un contenedor con un `h1` que muestra bien en grande como se llama el puerto seleccionado. Además, mediante *CSS* se ha añadido una imagen de fondo para dar un ambiente más marino. Dentro del elemento `h1` se encuentra el elemento `span` ya mencionado.

El código que lo genera es siguiente:

```

<!-- Nombre del Puerto -->
<div class="titulo" id="titulo">
    <h1><span id="port-name"></span></h1>
</div>
<!-- END Nombre del Puerto-->

```

### 4.2.2 Descripción y Video

En esta sección de la página, se muestra la descripción del puerto seleccionado, además de un video que lo muestra. Ambas cosas son atributos que obtenemos del fichero JSON de puertos. El como leemos e insertamos estos atributos en la pagina se explicará en el apartado de ficheros JavaScript.

Los dos atributos se encuentran en un objeto `container-fluid` y dentro de este mostramos una fila con dos columnas. Cada una de ellas es uno de los atributos. La descripción está contenida en una columna de la clase `col-md-8 col-lg-6` y el video en una columna de clase `col-md-4 col-lg-6 mb-4 align-self-center`. El video se encuentra en un `div` de clase `embed-responsive embed-responsive-16by9 custom-embed-responsive`, que nos permite poner un video de YouTube justo al lado de la descripción del puerto, de forma que este se pueda reproducir sin necesidad de abrir una pestaña nueva con el link del video.

```
<!-- Características del Puerto -->
<div id="desc-video" class="container-fluid">
  <div class="row">
    <div class="col-md-8 col-lg-6">
      <!-- Puerto Náutico Descripción -->
      <div class="mb-3 text">
        <p><span id="port-descr"></span></p>
      </div>
    </div>
    <div class="col-md-4 col-lg-6 mb-4 align-self-center">
      <div class="embed-responsive embed-responsive-16by9 custom-embed-responsive">
        <div id="player"></div>
      </div>
    </div>
  </div>
</div>
```

#### 4.2.3 Características i Tiempo

En esta sección, se muestran todas las otras características del puerto junto con el tiempo que hace en este. Las características son todas atributos de dentro del json mientras que el tiempo en ese puerto lo podemos obtener gracias a las coordenadas del puerto (que son dos atributos del puerto en el json).

`<div id="car-tiempo" class="container-fluid">`: Este es un contenedor `<div>` con un ID `car-tiempo` y la clase `container-fluid`. La clase `container-fluid` se utiliza en Bootstrap para crear un contenedor que ocupa todo el ancho de la pantalla, como ya se ha mencionado en ocasiones anteriores en el documento.

`<div class="row">`: con esto, creamos una fila dentro del contenedor. En Bootstrap, las filas se utilizan para organizar y alinear los elementos en una cuadrícula.

`<div class="col-md-8 mb-4" data-aos="fade-right" aos-duration="1000" aos-delay="1500">`: Este `<div>` crea una columna dentro de la fila. La clase `col-md-8` indica que esta columna ocupará 8 columnas de ancho en pantallas medianas y más grandes. La clase `mb-4` se utiliza para agregar un margen inferior de tamaño medio. Los atributos de datos `data-aos`, `aos-duration` y `aos-delay` se utilizan para animar la aparición de la columna utilizando la biblioteca AOS (Animate On Scroll).

`<table class="table">`: Esto crea una tabla con la clase `table`. En Bootstrap, la clase `table` se utiliza para aplicar estilos predefinidos a las tablas. Dentro de la tabla, hay varias filas (`<tr>`) que contienen datos sobre las características de un puerto náutico, como la capacidad de barcos, si permite fumar, horarios, dirección, información de contacto, etc. Cada fila contiene dos celdas (`<td>`): una para el ícono y el título, y otra para el valor correspondiente al atributo del puerto. Los valores se representan dentro de elementos `<span>` con IDs específicos. Estos IDs son los que usaremos para insertar el contenido de estos mediante JavaScript de forma dinámica.

Después de la columna anterior, hay otra columna <div> con la clase col-md-4 mb-4. Esta columna ocupará 4 columnas de ancho en pantallas medianas y más grandes. Dentro de esta columna, hay un contenedor <div> con el ID tiempo que alberga información sobre el clima del puerto. Hay tres bloques (<div class="box">) que muestran información como la temperatura, humedad y velocidad del viento. Estas características se pueden mostrar gracias a que usamos la API OpenWeatherAPI. El como la usuamos, lo mostraremos en el apartado de APIs. Cada valor de la climatología se representa dentro de elementos <span> con IDs específicos. Estos IDs son los que usaremos para insertar el contenido de estos mediante JavaScript de forma dinámica.

El código html que usuamos para mostrar la tabla de características y el tiempo es el siguiente:

```
<!-- Características del Puerto -->
<div id="desc-video" class="container-fluid">
<div class="row">
    <div class="col-md-8 col-lg-6">
        <!-- Puerto Náutico Descripción -->
        <div class="mb-3 text">
            <p><span id="port-descr"></span></p>
        </div>
        </div>
        <div class="col-md-4 col-lg-6 mb-4 align-self-center">
            <div class="embed-responsive embed-responsive-16by9 custom-embed-responsive">
                <div id="player"></div>
            </div>
            </div>
        </div>
    </div>
</div>

<div id="car-tiempo" class="container-fluid">
<!-- Puerto Náutico Características -->
<div class="row">
    <div class="col-md-8 mb-4" data-aos="fade-right" aos-duration="1000"
        aos-delay="1500">
        <table class="table">
            <tbody>
                <tr>
                    <td>
                        <i class="fas fa-ship"></i>
                        <strong>Capacidad de Barcos:</strong>
                    </td>
                    <td><span id="port-capacidad"></span></td>
                </tr>
                <tr>
                    <td>
                        <i class="fas fa-smoking-ban"></i>
                        <strong>Permite Fumar:</strong>
                    </td>
                    <td><span id="port-smoke"></span></td>
                </tr>
                <tr>
                    <td>
                        <i class="far fa-clock"></i>
                        <strong>Horario entre semana:</strong>
                    </td>
```

```

<td><span id="port-horaES"></span></td>
</tr>
<tr>
    <td>
        <i class="far fa-clock"></i>
        <strong>Horario fin de semana:</strong>
    </td>
    <td><span id="port-horaFS"></span></td>
</tr>
<tr>
    <td>
        <i class="fas fa-map-marker-alt"></i>
        <strong>Calle:</strong>
    </td>
    <td><span id="port-calle"></span></td>
</tr>
<tr>
    <td>
        <i class="fas fa-map-marker-alt"></i>
        <strong>Localidad:</strong>
    </td>
    <td><span id="port-locality"></span></td>
</tr>
<tr>
    <td>
        <i class="fas fa-map-marker-alt"></i>
        <strong>Codigo Postal:</strong>
    </td>
    <td><span id="port-addressCode"></span></td>
</tr>
<tr>
    <td>
        <i class="fas fa-envelope"></i>
        <strong>Correo:</strong>
    </td>
    <td><span id="port-correo"></span></td>
</tr>
<tr>
    <td>
        <i class="fas fa-phone"></i>
        <strong>Telefono:</strong>
    </td>
    <td><span id="port-telephone"></span></td>
</tr>
<tr>
    <td>
        <i class="fas fa-star"></i>
        <strong>Valoración:</strong>
    </td>
    <td id="contenedorValoracion">
    </td>
</tr>
</tbody>

```

```

</table>
</div>
<!-- Tiempo -->
<div class="col-md-4 mb-4">
    <!-- Imagen -->
    <div id="tiempo" class="text-center" data-aos="fade-left"
        aos-duration="1000">
        <h3 id="tempCity"></h3>

        <div class="box">
            
            <div class="box2">
                <p>Temperatura:</p>
                <span id="temperature"></span>
            </div>
        </div>

        <div class="box">
            
            <div class="box2">
                <p>Humedad:</p>
                <span id="humidity"></span>
            </div>
        </div>

        <div class="box">
            
            <div class="box2">
                <p>Viento:</p>
                <span id="windSpeed"></span>
            </div>
        </div>
    </div>
</div>
<!-- END Características del Puerto -->

```

#### 4.2.4 Botones de Favoritos

Para que el usuario pueda añadir el puerto que ha seleccionado a sus favoritos, le ofrecemos dos botones para que pueda hacerlo. El código, contiene 2 botones:

- El primer botón tiene el id `addToFavoritesBtn` y cuando se hace clic en él, se invoca la función `addToFavorites()`. También contiene un ícono de corazón representado por la etiqueta `<i class="fas fa-heart"></i>`. El texto dentro del botón dice Añadir a favoritos.
- El segundo botón también tiene el id `addToFavoritesBtn`, lo cual es incorrecto ya que los identificadores deben ser únicos en un documento HTML válido. Al hacer clic en este botón, se invoca la función `removeFavoritePort()`. Este botón contiene un ícono de basura representado por la etiqueta `<i class="fas fa-trash"></i>`. El texto dentro del botón dice Eliminar.

El código explicado es el siguiente:

```
<!-- Botones de Favs -->
<button id="addToFavoritesBtn" onclick="addToFavorites()">
    <i class="fas fa-heart"></i>
</button>
<button id="removeFavoritePort()" onclick="removeFavoritePort()">
    <i class="fas fa-trash"></i>
</button>
<!-- END Botones de Favs -->
```

#### 4.2.5 Galería de Imágenes

En esta sección se muestran las imágenes que se encuentran como atributos en el json de los puertos. Como la mayoría de elementos de este fichero, se generan mediante el fichero `port.js` y en el archivo html solo encontramos el esqueleto de donde se muestra la información que se genera de forma dinámica.

El código html simplemente consiste en un `container-fluid` ya mencionado anteriormente. Este tiene un `id = galeria-img` que usaremos en el fichero `port.js` para añadir contenido HTML de forma dinámica con las imágenes del puerto que el usuario ha seleccionado. Se explicará en el apartado de ficheros JavaScript

El código html del que hablamos es el siguiente:

```
<!-- Galería de Imágenes -->
<div class="container-fluid" id="galeria-img">
</div>
<!-- END Galería de Imágenes -->
```

#### 4.2.6 Lugares de interés cercanos

En la sección de lugares de interés cercanos de un puerto encontramos dos partes principales:

- **Restaurantes y cafeterías:** El código html que nos ayuda a generar el mapa con los restaurantes y cafeterías cercanas es el siguiente:

```
<!-- Restaurantes y Cafeterías-->
<div class="container-fluid" id="contenedorMapa">
    <div class="container" id="titolMapa">
        <h2 class="text-center">Restaurantes y Cafeterías cercanas</h2>
    </div>
    <div id="map"></div>
</div>
<!-- END Restaurantes y Cafeterías-->
```

Mediante este, posteriormente en el fichero `port.js` podremos generar el mapa para mostrar esas cafeterías y restaurantes cercanos.

- **Puertos Cercanos:** En esta sección se muestran los puertos que se encuentran próximos al que el usuario ha seleccionado. La estructura html que usamos para mostrar la previsualización de un puerto es exactamente la misma que usamos para mostrarlos en la *homepage*. Por tanto obviamos la explicación de las clases de `Bootstrap` ya que son idénticas. Eso sí, la generación de los puertos que se tienen que mostrar es un poco diferente que en la *homepage*. Esto

se debe a que solo debemos mostrar un subconjunto de los puertos totales. Los detalles de como implementamos y generamos las `cards` que permiten mostrar los puertos cercanos, los explicaremos en el apartado de los archivos JavaScript cuando expliquemos el archivo `port.js`.

El código html que nos permite mostrar los puertos cercanos es el siguiente:

```
<!-- Puertos cercanos -->
<div class="container" id="titolMapa">
    <h2 class="text-center">Puertos cercanos</h2>
</div>
<div class="container-fluid" id="contenedorPuertos"></div>
<!-- END Puertos cercanos-->
```

### 4.3 sobreNosotros.html

En este último fichero html encontramos la sección de contacto para el usuario ademas de información sobre los creadores de la página. En este último html, también obviaremos la explicación tanto de la barra de navegación como del footer (ya que se han explicado en apartados anteriores).

En cuanto a la parte principal del archivo, enconteramos que todo esta dentro de un objeto de la clase `container-fluid` que a su vez esta dentro de un objeto de tipo `site-section`. Esta última clase mencionada se utiliza comúnmente para definir una sección o bloque de contenido en una página web. Proporciona estilos y características visuales específicas para esa sección en particular. Estos estilos pueden incluir márgenes, rellenos, colores de fondo, bordes u otros atributos visuales que ayudan a estructurar y diseñar la página de contacto. Dentro de este `div` encontramos dos secciones:

- **Video Presentación:** se trata de un video donde los creadores de la web-app nos mostramos y presentamos para darnos a conocer al usuario. Esta parte de la web ha sido generada usando las siguientes clases de `bootstrap`:

- `<div class="row>`: La clase `row` se utiliza en Bootstrap para crear una fila que contiene columnas. Se utiliza para establecer una estructura de cuadrícula en la que se pueden organizar los elementos de forma horizontal. Las columnas dentro de una fila deben sumar un total de 12 unidades para que se distribuyan correctamente en diferentes tamaños de pantalla.
- `<div class="col-md-6 col-sm-6>`: La clase `col-md-6` y `col-sm-6` se utilizan para crear una columna que ocupará la mitad del ancho total de la fila en pantallas medianas (`md`) y pequeñas (`sm`). Esto significa que la columna ocupará 6 unidades de ancho de las 12 unidades disponibles en la fila. Estas clases permiten que el diseño sea responsive y se ajuste a diferentes tamaños de pantalla.

La explicación de las clases que usamos para mostrar el video en la web se explicaran mas adelante en el apartado de inclusión de media en nuestra web-app.

- **Formulario:** para poder mostrar un formulario para que el usuario nos pueda enviar sugerencias, reclamaciones o dudas, hemos usado la API `emailJS`. Por otra parte, para crear los campos donde el ususario tiene que insertar los datos són las siguientes:

- `<div class="col-md-6 col-sm-6>`: Esta clase crea una columna que ocupará la mitad del ancho total de la fila en pantallas medianas y pequeñas. Esto significa que la columna ocupará 6 unidades de ancho de las 12 unidades disponibles en la fila. Esta clase se utiliza para establecer el diseño responsive del formulario en diferentes tamaños de pantalla.
- `<h1>Contáctanos</h1>`: Esto muestra un encabezado que indica el título `Contáctanos`, que se utiliza para indicar que este formulario es para ponerse en contacto.

- `<form id="myForm">`: Esto crea un formulario con el ID `myForm`. El atributo `id` se utiliza para identificar el formulario de manera única, lo cual puede ser útil para manipularlo con el archivo JavaScript `email.js` que exolicaremos en el apartado de ficheros JavaScript.
- `<div class="form-group">`: Esta clase se utiliza para envolver cada grupo de elementos del formulario, lo que ayuda a aplicar estilos y estructurar visualmente el formulario.
- `<label for="firstName">Nom</label>`: Esta etiqueta se utiliza para mostrar un texto descriptivo que indica qué información se espera que se ingrese en el campo de entrada siguiente. El atributo `for` se utiliza para establecer la relación entre la etiqueta y el campo de entrada al vincularlo mediante el atributo `id`.
- `<input type="text" class="form-control" id="firstName" placeholder="Escriu el teu nom aquí">`: Este elemento `<input>` se utiliza para crear un campo de entrada de texto donde los usuarios pueden ingresar su nombre. La clase `form-control` se utiliza en Bootstrap para aplicar estilos específicos a los campos de entrada de formulario. El atributo `placeholder` proporciona un texto de ejemplo que se muestra dentro del campo de entrada antes de que se ingrese cualquier valor. Los siguientes elementos `<div class="form-group">`, `<label>`, y `<input>` siguen el mismo patrón descrito anteriormente, pero se utilizan para el campo de correo electrónico, el campo de selección de categoría y el campo de descripción del mensaje.
- `<button type="submit" class="btn btn-primary" ... onclick="sendEmail(event)"> Envia </button>`: Este elemento `<button>` se utiliza para crear un botón de envío en el formulario. La clase `btn` y `btn-primary` se utilizan en Bootstrap para aplicar estilos predefinidos a los botones. El atributo `onclick` se utiliza para llamar a una función de JavaScript llamada `sendEmail(event)` cuando se hace clic en el botón. La explicación de como se desarrolla esta en el archivo `email.js`, la daremos en el apartado de archivos JavaScript.

El código html que permite generar la pagina de contacto con el video y el formularo es la siguiente:

```
<!-- MIDDLE CONTENT OF THE WEBSITE-->
<!-- Sección introductoria a la página -->
<div class="site-section" id="intro">
  <div class="container-fluid">
    <h1>          </h1>
    <div class="row">
      <div class="col-md-6 col-sm-6">
        <div class="video-container">
          <video width="640" height="360" controls>
            <source src="video.mp4" type="video/mp4">
            <source src="video.webm" type="video/webm">
            Tu navegador no admite la etiqueta de video.
          </video>
        </div>
      </div>
      <div class="col-md-6 col-sm-6">
        <!-- CONTACT WITH US FORM -->
        <h1>Contáctanos</h1>
        <form id="myForm">
          <!-- FIRST NAME ATTRIBUTE-->
          <div class="form-group">
            <label for="firstName">Nombre</label>
            <input type="text" class="form-control"
                   id="firstName" placeholder="Escriu el teu nom aquí">
          </div>
          <div class="form-group">
            <label for="lastName">Apellido</label>
            <input type="text" class="form-control"
                   id="lastName" placeholder="Escriu el teu cognom aquí">
          </div>
          <div class="form-group">
            <label for="email">Correu electrònic</label>
            <input type="email" class="form-control"
                   id="email" placeholder="Escriu el teu correu aquí">
          </div>
          <div class="form-group">
            <label for="message">Missatge</label>
            <input type="text" class="form-control"
                   id="message" placeholder="Escriu el teu missatge aquí">
          </div>
          <div class="form-group">
            <label for="checkbox">Accepto les condicions</label>
            <input type="checkbox" class="form-control"
                   id="checkbox">
          </div>
          <div class="form-group">
            <button type="submit" class="btn btn-primary" ... onclick="sendEmail(event)"> Envia </button>
          </div>
        </form>
      </div>
    </div>
  </div>
</div>
```

```

        </div>
        <!-- EMAIL ATTRIBUTE-->
        <div class="form-group">
            <label for="email">Email</label>
            <input type="email" class="form-control"
                id="email" aria-describedby="emailHelp"
                placeholder="Escriu el teu email">
            <small id="emailHelp" class="form-text text-muted">
                No compartiremos tu correo con nadie</small>
        </div>
        <!-- CATEGORY ATTRIBUTE-->
        <div class="form-group">
            <label for="category">Categoria:</label>
            <select class="form-control" id="category">
                <option value="doubts">Dudas</option>
                <option value="problems">Problemas</option>
                <option value="suggestions">Sugerencias</option>
                <option value="other">Otros</option>
            </select>
        </div>
        <!-- DESCRIPTION ATTRIBUTE-->
        <div class="form-group">
            <label for="description">Descripción</label>
            <textarea class="form-control" id="message" rows="8"
                placeholder="Escriu el que ens vulguis fer saber">
            </textarea>
        </div>
        <div class="text-center">
            <button type="submit" class="btn btn-primary"
                value="Submit" onclick="sendEmail(event)">Envia
            </button>
        </div>
    </form>
</div>
</div>
</div>
<!-- END Sección introductoria a la página -->
<!-- END MIDDLE CONTENT -->
```

## 5 Ficheros JavaScript

En esta sección, explicaremos el contenido de los ficheros JavaScript que hemos desarrollado. Estos realizan 3 funciones principalmente:

- Insertar código de forma dinámica a sus correspondientes ficheros `html`.
- Lectura y tratamiento de ficheros `.json`.
- Utilización de las APIs y añadir datos propios dentro de estas

Sabiendo las principales funciones que realizan los ficheros `.js` pasaremos a explicar cada uno de ellos con mas detalle.

## 5.1 main.js

Este primer javascript va asignado al fichero `index.html`. Este contiene todas las funciones que hacen posible mostrar datos sobre los puertos en la *homepage*.

Lo primero que encontramos en el fichero es el método con el que vamos a leer los ficheros json.

```
let ports = [];
fetch('ports.json')
  .then(response => response.json())
  .then(data => {
    ports = data.itemListElement; // Obtener todos los puertos del array
    updatePorts(ports); // Llamar a la función updatePorts para actualizar
                         // los elementos del DOM
  });
}
```

El código que hemos desarrollado realiza una solicitud HTTP GET a nuestro archivo de puertos llamado `ports.json`. Una vez que se obtiene la respuesta de la solicitud, se convierte en formato JSON utilizando el método `json()`. Después de eso, se utiliza el método `then()` para manejar la promesa devuelta por `response.json()`. En la función de devolución de llamada, se accede a la propiedad `itemListElement` de los datos obtenidos del archivo JSON de puertos y se almacenan en una variable global que hemos llamado `ports`. Por tanto, dentro de esta estarán contenidos todos los puertos del fichero. Esta variable es la que usaremos para pasar por parámetro a las funciones que queremos que se ejecuten cuando el usuario acceda a la *homepage*. Estas son todas la funciones que se encuentran en este fichero de JavaScript:

- `updatePorts(ports)`: La función crea cuatro arrays vacíos: `coordenadasLat`, `coordenadasLon`, `capacidades` y `nombres`, cada uno con una longitud igual a la longitud del array `ports` que recibimos por parámetro. Además, esta función es la encargada de generar las `cards` con la información previsualizada de los puertos en la *homepage*. La función, también realiza la obtención de una referencia al elemento HTML con el id `contenedorPrePuertos` utilizando la instrucción siguiente `document.getElementById(contenedorPrePuertos)`. Luego, inicializa una variable llamada `html` como una cadena vacía y otra variable `items` como 0 para poder contar el numero de `cards` en una fila.

Luego, la función recorre cada elemento del array `ports` utilizando un bucle `for`. En cada iteración, se obtienen diferentes propiedades de cada puerto, como el nombre (`portName`), la ubicación geográfica (`portGeo`), la capacidad (`portCapacitat`), una de las imágenes (`portImage`), y la valoración (`valoracion`). En cada iteración, va almacenando los correspondientes valores obtenidos en los arrays inicializados al principio de la función.

En el bucle, también se construye una estructura de HTML utilizando plantillas literales (`backticks`) para generar tarjetas de puerto (`cards`) de la *homepage*. Se añade la imagen, el nombre del puerto, la capacidad y una sección de valoración utilizando iconos de estrellas. El valor de `valoracion` se compara en diferentes rangos para determinar qué iconos de estrellas se deben mostrar en función de la puntuación. Dependiendo de la valoración, se generará el HTML adecuado con la combinación de estrellas llenas (`fa fa-star checked`) y estrellas medias llenas (`fa fa-star-half checked`) para representar la puntuación. Obviamente, dentro del código `html` generado, se añaden los valores de las variables en esa iteración en concreto del bucle.

Finalmente, se llama a la función `initMap()` con los arrays `coordenadasLat`, `coordenadasLon`, `capacidades` y `nombres` como argumentos. La función `initMap` se encargará de utilizar estos datos para inicializar un mapa con marcadores correspondientes a los puertos y mostrarlos en la interfaz HTML. Además también se asigna el contenido HTML generado y concentrado en la variable `html` a través de `contenedorGeneral.innerHTML`. Esto actualizará el contenido del elemento HTML con el id `contenedorPrePuertos` y mostrará las `cards` de puerto generadas en la web. El código de la función que realiza todo este trabajo es el siguiente:

```

function updatePorts(ports) {
    const contenedorGeneral = document.getElementById("contenedorPrePuertos")
    var html = '';
    var items = 0;
    let coordenadasLat = new Array(ports.length); //Coordenadas de latitud
    let coordenadasLon = new Array(ports.length); //Coordenadas de longitud
    let capacidades = new Array(ports.length); //Capacidades de los puertos
    let nombres = new Array(ports.length); //Array de nombres de los puertos
    // Mostrar la información de cada puerto en el HTML
    for (let i = 0; i < ports.length; i++) {
        const port = ports[i];
        const portName = port.name;
        const portGeo = port.geo;
        const portCapacitat = port.additionalProperty
        && port.additionalProperty maxValue;
        const portImage = port.image[1];
        const valoracion = port.aggregateRating.ratingValue;
        //Localitació
        coordenadasLat[i] = portGeo.latitude;
        console.log(coordenadasLat[i]);
        coordenadasLon[i] = portGeo.longitude;
        console.log(coordenadasLon[i]);
        //Capacitat
        capacidades[i] = portCapacitat;
        //Nom
        nombres[i] = portName;
        const portId=trobarIndex(portName);
        if (items % 4 == 0) {
            html += '<div class="row equal-width">';
        }
        html += `

<div class="col-md-3">
<div class="card">
<a href="puerto.html?portId=${portId}">
</a>
<div class="card-body">
<h5 class="card-title">` + portName + `</h5>
<ul>
<li>Capacidad: ` + portCapacitat + `</li>
</ul>
`;
        html += setStars(valoracion);
        html += `

</div>
</div>
</div>
`;
        items++;
        if (items % 4 == 0) {
            html += '</div>'; // Cerrar la fila después de
            //agregar cuatro tarjetas
        }
    }
}

```

```

        }
    }
    contenidoGeneral.innerHTML = html;
    initMap(coordenadasLat, coordenadasLon, capacidades, nombres);
}

function trobarIndex(nom){

    for (let i = 0; i < ports.length; i++) {
        const port=ports[i];

        if(port.name==nom){
            return i;
        }
    }
    return -1;
}

function setStars(valoracion){

    html='<div class="rating" id="rating">';
    count=valoracion;
    for(let i=0;i<valoracion; i++){
        if(count<=0.5{
            html += '<span class="fa fa-star-half checked"></span>';
        }else{
            html += ' <span class="fa fa-star checked"></span>'
        }
        count--;
    }
    html += `<p class="valoracionPuertoPrev">` + valoracion + `</p>
                </div>`;
    return html;
}

```

La función `initMap()` es la encargada de usar la API de Google Maps y mostrar el mapa interactivo en la página usando los datos que hemos leído del fichero y pasado por parámetro a la función. El contenido de esta se mostrará y explicará en el apartado de APIs de la documentación.

- `PortSearch()`: esta función es la encargada de la barra de filtros que se muestra en la *homepage*. Se ejecuta cuando el usuario clica encima del botón de buscar en la barra de filtros de la *homepage*. Primero, se obtienen los valores de los campos del formulario, como el nombre, la capacidad, la valoración y el criterio de ordenación seleccionado.

A continuación, se verifica si todos los campos de búsqueda están indefinidos o vacíos. Si es así, se muestra una alerta indicando al usuario que debe completar al menos uno de los campos del formulario y se sale de la función.

Después, se realiza el filtrado de los puertos en base a los criterios de búsqueda proporcionados. La función `filter()` se utiliza para iterar sobre la lista de puertos y se aplican las siguientes condiciones de filtrado:

- **nameMatch**: Verifica si el atributo `name` del puerto coincide con el criterio de búsqueda del nombre.
- **capacityMatch**: Verifica si el atributo `capacity` del puerto no excede el valor de capacidad especificado.

- **ratingMatch**: Verifica si el atributo `ratingValue` del puerto coincide con el valor de valoración especificado.
- **favMatch**: Verifica si el puerto está en la lista de puertos favoritos almacenada en el almacenamiento local del navegador.

Si todas las condiciones se cumplen, el puerto se considera un puerto filtrado válido y se incluye en la lista `filteredPorts`.

A continuación, se ordena la lista `filteredPorts` en base al criterio de ordenación seleccionado. Se utilizan las funciones `sortPortsByName()`, `sort()` y `sort()` con comparadores personalizados para ordenar los puertos por nombre, capacidad y valoración respectivamente.

Finalmente, se llama a la función `updatePorts()` para actualizar la visualización de los puertos en la interfaz.

La función `sortPortsByName(ports2)` se utiliza para ordenar la lista de puertos por nombre en orden alfabetico. Se utiliza la función `sort()` con un comparador personalizado que compara los nombres de los puertos en minúsculas.

```
function PortSearch(){
    const nom=document.getElementById("form-name").value;
    const capacity=document.getElementById("filtro-capacidad-max")
                    .value;

    const ratingComponent=document.getElementById("filtro-valoracion");
    const ratingValue=ratingComponent.options[ratingComponent.selectedIndex]
                    .value;

    const sortByComponent=document.getElementById("filtro-ordenar");
    const sortByValue=sortByComponent.options[sortByComponent.selectedIndex]
                    .value;

    const sortByFavourites=document.getElementById("filtro-favoritos");
    const Favourites=sortByFavourites.options[sortByFavourites.selectedIndex]
                    .value;

    var html="";

    //si tots 3 estan indefinitos
    //crea alerta i surt
    if(nom == "" && capacity == "" && ratingValue=="Seleccionar..." && sortByValue == "Seleccionar..." && Favourites == "Seleccionar..."){
        alert("Tienes que llenar al menos uno de los campos del formulario!");
        return;
    }

    var filteredPorts = ports.filter(function (port) {
        var nameMatch = !nom ||
                        nom === "" ||
                        port.name.toLowerCase().includes(nom.toLowerCase());

        var capacityMatch =
            !capacity ||
            capacity === "" ||
            port.additionalProperty maxValue <= capacity;
    })
}
```

```

var ratingMatch =
ratingValue === "Seleccionar..." || (port.aggregateRating.ratingValue
>= ratingValue && port.aggregateRating.ratingValue <= (ratingValue+1));

const favoritePorts = JSON.parse(localStorage.getItem('favoritePorts'))|| []
var favMatch =
Favourites ==="Todos" || favoritePorts.some(function (favoritePort) {
    return favoritePort.name === port.name;
});

return nameMatch && capacityMatch && ratingMatch && favMatch;
});

var sorted=filteredPorts;

if(sortByValue == "nombre"){
    sorted=sortPortsByName(filteredPorts);

}else if(sortByValue == "capacidad"){

    sorted =filteredPorts.sort(function (a, b) {
        return a.additionalProperty maxValue - b.additionalProperty maxValue;
    });

}else if(sortByValue == "valoracion"){

    sorted =filteredPorts.sort(function (a, b) {
        return b.aggregateRating.ratingValue - a.aggregateRating.ratingValue;
    });

}

updatePorts(sorted);
}

function sortPortsByName(ports2){
    return ports2.sort(function(a,b){
        var nameA = a.name.toLowerCase();
        var nameB = b.name.toLowerCase();
        if (nameA < nameB) {
            return -1;
        }
        if (nameA > nameB) {
            return 1;
        }
        return 0;
    });
}

```

## 5.2 port.js

En este fichero JavaScript estan contenidas todas las funciones que hacen posible mostrar la información de un puerto al usuario. Es el fichero que va enlazado al html llamado `puerto.html`. Primero de todo tenemos la función que nos permite leer los ficheros json de puertos, restaurantes

y cafeterías y posteriormente llamamos a todas las funciones que se tengan que ejecutar pasandoles como pámetro el array de puertos, cafeterías o restaurantes que hemos leido llenado las variables `ports`, `restaurantes`, `cafeterias`. También observamos la definición de variables globales. Se definen de forma global porque se deberan utilizar en más de una función que ahora describiremos. la utilidad de cada una, queda descrita en su comentario adjacente en el siguiente código:

```

var port; //puerto que tratamos actualmente en la pagina
let coordenadasLatRestaurantes = []; //Coordenadas de latitud de los restaurantes
let coordenadasLonRestaurantes = []; //Coordenadas de longitud de los restaurantes
let urlsRestaurantes = []; //URL de los restaurantes
let nombresRestaurantes = [] //Array de nombres de los restaurantes

let coordenadasLatCafeterias = []; //Coordenadas de latitud de las cafeterias
let coordenadasLonCafeterias = []; //Coordenadas de longitud de las cafeterias
let urlsCafeterias = [] //URLs de las cafeterias
let nombresCafeterias = [] //Array de nombres de las cafeterias

//Para leer nuestro fichero de puertos
fetch('ports.json')
    .then(response => response.json())
    .then(data => {
        const ports = data.itemListElement; // Obtener todos los puertos del array
        infoPort(ports); //Llamar a la función para mostrar la información
        //de un puerto
        loadPorts(ports);
        loadImgs(ports);
        //Para leer el fichero de restaurantes
        fetch('restaurante.json')
            .then(response => response.json())
            .then(data => {
                const restaurantes = data.itemListElement; // Obtener todos los
                // puertos del array
                actualizaRestaurantes(restaurante,port);
            });
        //Para leer el fichero de cafeterias
        fetch('cafeterias.json')
            .then(response => response.json())
            .then(data => {
                const cafeterias = data.itemListElement; // Obtener todos los
                //puertos del array
                actualizaCafeterias(cafeterias,port);
                console.log("Vaig a ejecutar el mapa");
                initMap(); //ejecutamos el initmap
            });
    });

function initMap() {
    var latitud = port.geo.latitude;
    console.log("Latitud port = "+ latitud);
    var longitud = port.geo.longitude;
    console.log("Longitud port = "+ longitud);
    const init = { lat: latitud, lng: longitud };
    var map = new google.maps.Map(document.getElementById('map'), {

```

```

        zoom: 13,
        center: init
    });
    //Per mostrar els marcadors dels restaurants
    for (let i = 0; i < coordenadasLatRestaurantes.length; i++) {
        var lat = parseFloat(coordenadasLatRestaurantes[i]);
        var lon = parseFloat(coordenadasLonRestaurantes[i]);
        // Define el color del marcador (en este ejemplo, rojo)
        var color = 'blue';

        // Crea un icono personalizado con el color seleccionado
        var icon = {
            path: google.maps.SymbolPath.BACKWARD_CLOSED_ARROW,
            fillColor: color,
            fillOpacity: 1,
            strokeColor: '#ffff',
            strokeOpacity: 1,
            strokeWeight: 1,
            scale: 5
        };
        const marker = new google.maps.Marker({
            position: { lat: lat, lng: lon},
            map: map,
            icon: icon
        });
        console.log("Marker mostrado Lat = "+ lat +" Lon = "+ lon);
        // Agregar evento de clic al marcador
        marker.addListener('click', function () {
            const infoWindow = new google.maps.InfoWindow({
                content: '<a href="'+urlsRestaurantes[i]+'"'+
                    'target="_blank"><strong>$nombresRestaurantes[i]</strong></a> '
            });
            infoWindow.open(map, marker);
        });
    }
}

```

En el anterior código, la función `initMap()` es la encargada de mostrar en la web el mapa de Google Maps con las cafeterías y restaurantes cercanos al puerto seleccionado. Esta realiza exactamente la misma tarea que el `initMap()` de la *homepage*, pero en este caso usando los datos de los restaurantes y cafeterías. El tratamiento de los datos es el mismo y este se explicará en el apartado de la API de Google Maps. Las funciones que se ejecutarán cuando el usuario entre en la página de un puerto serán las siguientes:

- `infoPort(port)`: es una función que se utiliza para mostrar información sobre un puerto en nuestra web-app. A continuación se describen las acciones que realiza la función:
  - Obtiene los parámetros de la URL de la página actual utilizando `URLSearchParams` y obtiene el valor del parámetro `portId` (se lo pasamos al fichero cuando el usuario selecciona un puerto).
  - Actualiza varios elementos del Document Object Model (DOM) de la página con la información extraída del puerto. Estos elementos del DOM se identifican mediante sus atributos `id` y se accede a ellos mediante el uso de `getElementById()`. Los elementos que se actualizan incluyen:

- \* Nombre del puerto.
  - \* Descripción del puerto.
  - \* Horarios de apertura durante la semana y en fines de semana.
  - \* Dirección del puerto, incluyendo calle, localidad y código postal.
  - \* Indicación sobre si se permite fumar en el puerto.
  - \* Valoración media del puerto y número de valoraciones recibidas.
  - \* Capacidad máxima del puerto.
  - \* Número de teléfono de contacto del puerto.
  - \* Correo electrónico del puerto.
- La función también genera dinámicamente un fragmento HTML que muestra la valoración del puerto. La valoración se representa visualmente mediante estrellas estilizadas utilizando la clase de ícono de estrella de Font Awesome. Dependiendo del valor de la valoración media del puerto, se generan estrellas llenas y medias para representar la puntuación de forma precisa. Además, se muestra el número total de valoraciones recibidas.
- El elemento del DOM que se actualiza con el fragmento HTML generado en el paso anterior es el que tiene como `id = contenedorValoracio`. Este contenedor específico se encarga de mostrar la valoración del puerto en la página web.

La función en cuestión es la siguiente:

```
function infoPort(ports) {
    const urlParams = new URLSearchParams(window.location.search);
    const portId = urlParams.get('portId');
    port = ports[portId];
    var puertoJsonString = JSON.stringify(port); //Passam el contingut
                                                //del port a string

    //Atributos a mostrar del puerto
    const portName = port.name; //Nombre del puerto
    const portDesc = port.description //Descripción
    const portHoraES = port.openingHours[0]; //Horario de apertura
                                                //entre semana
    const portHoraFS = port.openingHours[1]; //Horario de apertura en
                                                //fin de semana
    const portAdressStr = port.address.streetAddress; //Calle
    const portAdressLoc = port.address.addressLocality; //Localidad
    const portAdressCod = port.address.postalCode; //Codigo postal
    const portSmoke = port.smokingAllowed; //Se puede fumar?
    const portRating = port.aggregateRating.ratingValue; //Valoración media
    const portRatingCount = port.aggregateRating.reviewCount; //Número de
                                                               //valoraciones
    const portCapacity = port.additionalProperty maxValue; //Capacidad del
                                                               //puerto
    const portTelephone= port.telephone; //Telefono del puerto
    const portCorreo = port.keywords.termCode; //Correo del puerto
    var portVideo = port.subjectOf.video[0]; //Link del video del puerto
    var html = '';

    //Actualizamos el valor del script en el head para la web semántica
    var scriptElement = document.getElementById('port-json');
    scriptElement.textContent = puertoJsonString;
    var videoId = extractVideoId(portVideo);
    var playerDiv = document.getElementById("player");
    playerDiv.innerHTML =
```

```

'<iframe class="embed-responsive-item" src="https://www.youtube.com/embed/' +
+ videoId + '"></iframe>';

//Nombre
const portNameElement = document.getElementById(`port-name`);
portNameElement.textContent = portName;

///Descripción
const portDescrElement = document.getElementById(`port-descr`);
portDescrElement.textContent = portDesc;

//Telefono
const portTelephoneElement = document.getElementById(`port-telephone`);
portTelephoneElement.textContent = portTelephone;

//Correo
const portCorreoElement = document.getElementById(`port-correo`);
portCorreoElement.textContent = portCorreo;

//Horas de apertura entre semana
const portHoraESElement = document.getElementById(`port-horaES`);
portHoraESElement.textContent = portHoraES;

//Horas de apertura fin de semana
const portHoraFSElement = document.getElementById(`port-horaFS`);
portHoraFSElement.textContent = portHoraFS;

//Calle
const portCalleElement = document.getElementById(`port-calle`);
portCalleElement.textContent = portAdressStr;

//Localidad
const portLocalityElement = document.getElementById(`port-locality`);
portLocalityElement.textContent = portAdressLoc;

//Codigo postal
const portPostalCodeElement = document.getElementById(`port-addressCode`);
portPostalCodeElement.textContent = portAdressCod;

//Codigo capacidad
const portCapacityElement = document.getElementById(`port-capacidad`);
portCapacityElement.textContent = portCapacity;

//Permite fumar
const portSmokeElement = document.getElementById(`port-smoke`);
if(portSmoke){
    portSmokeElement.textContent = `Si`;
} else{
    portSmokeElement.textContent = `No`;
}
//Valoración
const portValoracionContenedor =
    document.getElementById(`contenedorValoracion`);
```

```

        html = setStars(portRating);
        html += `<p>Número de valoraciones: ` + portRatingCount + `</p>
            </div>
        </div>
    `;
    portValoracionContenedor.innerHTML = html;
}

function extractVideoId(url) {
    var videoId = url.split('v=')[1];
    var ampersandPos = videoId.indexOf('&');
    if (ampersandPos !== -1) {
        videoId = videoId.substring(0, ampersandPos);
    }
    return videoId;
}

```

- `actualizaRestaurantes(restaurantes, port)`: este método recibe por parámetro un array de restaurantes leidos del fichero .json de restaurantes y el puerto del que obtenemos la información para mostrar en la web. Aquí está lo que hace cada parte de la función:

- `var latitudPort = port.geo.latitude;` y `var longitudPort = port.geo.longitude;`: Se obtienen las coordenadas de latitud y longitud del puerto actual.
- Se utiliza un bucle `for` para iterar sobre la lista de restaurantes.
- `var restaurant = restaurantes[i];`: Se asigna el objeto de restaurante actual a la variable `restaurant`.
- `var latitudRest = restaurant.geo.latitude;` y `var longitudRest = restaurant.geo.longitude;`: Se obtienen las coordenadas de latitud y longitud del restaurante actual.
- `if((sacarDist(latitudPort, longitudPort,latitudRest,longitudRest) < 10))`: Se verifica si la distancia entre el puerto y el restaurante es menor a 10 (se explica como funciona la función más abajo). Si la condición se cumple, significa que el restaurante está dentro de un radio de 10 unidades de distancia del puerto y se procede a procesar la información del restaurante.
- Se almacenan las coordenadas de latitud y longitud del restaurante que estamos tratando en esta iteración en las listas `coordenadasLatRestaurantes` y `coordenadasLonRestaurantes` respectivamente.
- Se obtiene el nombre del restaurante y se agrega a la lista `nombresRestaurantes`.
- Se obtiene la URL del restaurante y se agrega a la lista `urlsRestaurantes`.

```

function actualizaRestaurantes(restaurantes, port){
    var latitudPort = port.geo.latitude;
    var longitudPort = port.geo.longitude;
    // Mostrar la información de cada puerto en el HTML
    for (let i = 0; i < restaurantes.length; i++) {
        var restaurant = restaurantes[i];
        var latitudRest = restaurant.geo.latitude;
        var longitudRest = restaurant.geo.longitude;
        if((sacarDist(latitudPort, longitudPort,latitudRest,longitudRest) < 10)){
            coordenadasLatRestaurantes.push(latitudRest);
            coordenadasLonRestaurantes.push(longitudRest);
        }
    }
}

```

```

        var nom = restaurant.name;
        nombresRestaurantes.push(nom);
        var url = restaurant.url;
        urlsRestaurantes.push(url);
    }
}
}

```

- `actualitzaCafeterias(cafeterias, port)`: se realiza el mismo tratamiento que en la función `actualitzaRestaurantes()` pero en este caso con los datos leido del json de cafeterías (recibidos por parametro). También encontramos diferencias en el destinatario del tratamiento de los datos. Las coordenadas obtenidas de las cafeterías seran almacenadas en las variables globales `coordenadasLatCafeterias` y `coordenadasLonCafeterias`. El código de la función es el siguiente:

```

function actualitzaCafeterias(cafeterias, port){
    var latitudPort = port.geo.latitude;
    var longitudPort = port.geo.longitude;
    // Mostrar la información de cada puerto en el HTML
    for (let i = 0; i < cafeterias.length; i++) {
        var cafeteria = cafeterias[i];
        var latitudCaf = cafeteria.geo.latitude;
        var longitudCaf = cafeteria.geo.longitude;
        if((sacarDist(latitudPort, longitudPort, latitudCaf, longitudCaf) < 10)){
            coordenadasLatCafeterias.push(latitudCaf);
            coordenadasLonCafeterias.push(longitudCaf);
            var nom = cafeteria.name;
            nombresCafeterias.push(nom);
            var url = cafeteria.url;
            urlsCafeterias.push(url);
        }
    }
}

```

- `loadImgs(ports)`: función que se encarga de mostrar las imágenes contenidas como atributos de un puerto. Estas se mostraran en la galeria de imágenes en el contenido mostrado por el archivo `puerto.html`. Esta hace un tratamiento similar a las funciones ya explicadas anteriormente, pero con algunas diferencias sustanciales. En primer lugar, toma el parámetro `ports`, que es un objeto que contiene información sobre los puertos del json. Despues, busca el valor del parámetro `portId` en la URL de la página y lo asigna a la variable `portId`. Luego, utiliza ese valor para obtener el objeto puerto correspondiente del objeto `ports` (que será el puerto seleccionado por el usuario).

La función se encarga de cargar las imágenes de la galería de un puerto específico en una nuestra web. Esto es lo que hace cada parte de la función:

- `const urlParams = new URLSearchParams(window.location.search)`: Se utiliza para obtener los parámetros de la URL de la página actual. Esto es útil para obtener información sobre el puerto seleccionado.
- `const portId = urlParams.get('portId')`: Se obtiene el valor del parámetro `portId` de la URL, que representa el identificador del puerto seleccionado.
- `const puerto = ports[portId]`: Se obtiene el objeto del puerto correspondiente al identificador obtenido. La variable `puerto` contiene información sobre el puerto, incluyendo las imágenes de la galería.

- var images = puerto.image: Se asigna la propiedad `image` del objeto puerto a la variable `images`. Esta propiedad debe contener un array de URLs de imágenes.
- Se inicializan dos variables `html1` y `html2` que almacenarán el código HTML de la galería de imágenes.
- Se crea una estructura de carrusel utilizando las clases de `Bootstrap`. El código HTML generado incluye un título, indicadores de carrusel, y el contenido de las imágenes.
- Se recorre el array de imágenes y se generan los elementos HTML correspondientes para cada imagen. Esto incluye la creación de indicadores de carrusel y elementos de carrusel para cada imagen.
- Finalmente, se asigna el contenido HTML generado al elemento con el identificador `galeria-img` en la página web. La galería de imágenes se actualizará con el nuevo contenido.

La función explicada es la siguiente:

```
function loadImgs(ports) {
  const urlParams = new URLSearchParams(window.location.search);
  const portId = urlParams.get('portId');
  const puerto = ports[portId];
  var images = puerto.image;
  var html1 = '<h2>Galería de Imágenes</h2>' +
  '<div id="carouselInit" class="carousel slide mt-0 w-90%"' +
  '+data-bs-ride="carouselInit">';
  var html2= '<div class="carousel-inner">';
  var items = 0;
  const containerGeneral = document.getElementById("galeria-img");

  html1+="

37


```

```

html2+= '</div><button class="carousel-control-prev" type="button" '+
' data-bs-target="#carouselInit" data-bs-slide="prev">' +
'<span class="carousel-control-prev-icon" aria-hidden="true"></span>' +
'<span class="visually-hidden">Previous</span></button>' +
'<button class="carousel-control-next" type="button" '+
' data-bs-target="#carouselInit" data-bs-slide="next">' +
'<span class="carousel-control-next-icon" aria-hidden="true"></span>' +
'<span class="visually-hidden">Next</span>' +
'</button></div>';//tancam inner content

contenedorGeneral.innerHTML = html1+html2;
}

```

- **loadPorts(ports)**: es la función que se encarga de mostrar los puertos más cercanos al puerto que el usuario ha seleccionado. La estructura de la función es muy similar a la función de **updatePorts()** del archivo **main.js**. Recibimos por parámetro un array de los puertos leídos del archivo json y mediante un bucle **for** los vamos iterando uno por uno.

En cada iteración, vamos obteniendo las características que necesitamos mostrar de cada uno para las **cards** que debemos insertar dentro del código html del fichero **puerto.html**. La principal diferencia se encuentra en que dentro del bucle **for**, encontramos una sentencia condicional con la cual, no se hará nada de todo lo mencionado anteriormente si la distancia des de el puerto seleccionado hasta el puerto sobre el que estamos iterando es menor a 20km.

Cabe destacar que, al estar calculando las distancias en una superficie casi completamente esférica como es la Tierra, debemos utilizar una fórmula más compleja como podría ser la del módulo. El cálculo es realizado mediante la Fórmula de Haversine, que sirve para calcular la distancia entre dos puntos en una superficie esférica.

- **sacarDist(lat1,lon1,lat2,lon2)**: esta función se encara de calcular en km la distancia entre dos puntos de la Tierra. Utiliza la fórmula de Haversine, que, como hemos mencionado anteriormente, sirve para calcular la distancia entre dos puntos de una superficie esférica. Su expresión es la siguiente:

$$d = 2r \arcsin \left( \sqrt{\sin^2 \left( \frac{\phi_2 - \phi_1}{2} \right) + \cos(\phi_1) \cos(\phi_2) \sin^2 \left( \frac{\lambda_2 - \lambda_1}{2} \right)} \right)$$

- $d$ : distancia entre dos puntos
- $r$ : radio de la esfera, en nuestro caso, el radio de la Tierra es de unos 6371km.
- $\phi_1, \phi_2$ : latitud de los dos puntos en radianes
- $\lambda_1, \lambda_2$ : longitud de los dos puntos en radianes.

Las dos funciones explicadas en los dos últimos apartados son las siguientes:

```

function sacarDist(lat1, lon1, lat2, lon2){
  const R = 6371; // Radius of the earth in km
  const dLat = deg2rad(lat2 - lat1);
  const dLon = deg2rad(lon2 - lon1);
  const a =
    Math.sin(dLat/2) * Math.sin(dLat/2) +
    Math.cos(deg2rad(lat1)) * Math.cos(deg2rad(lat2)) *
    Math.sin(dLon/2) * Math.sin(dLon/2);
  const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
  const d = R * c; // Distance in km
}

```

```

        return d;
    }

//Graus a radians
function deg2rad(deg) {
    return deg * (Math.PI/180)
}

//Funció per a obtenir ports més propers
function loadPorts(ports) {
    const contenedorGeneral = document.getElementById("contenedorPuertos");
    const urlParams = new URLSearchParams(window.location.search);
    const portId = urlParams.get('portId');
    const puerto = ports[portId];
    var html = '';
    var items = 0;

    for (let i = 0; i < ports.length; i++) {
        const port = ports[i];
        if((sacarDist(puerto.geo.latitude,puerto.geo.longitude,port.geo.latitude,
            port.geo.longitude) < 20) && (puerto.name != port.name)){
            const portName = port.name;
            const portCapacitat = port.additionalProperty
            && port.additionalProperty.maxValue;
            const portImage = port.image[1];
            const valoracion = port.aggregateRating.ratingValue;

            if (items % 4 == 0) {
                html += '<div class="row equal-width">';
            }
            html += `

<div class="col-md-3">
    <div class="card">
        <a href="puerto.html?portId=${i}">
            
        </a>
        <div class="card-body">

            <h5 class="card-title">` + portName + `</h5>
            <ul>
                <li>Capacidad: ` + portCapacitat + `</li>
            </ul>
        `;
            html += setStars(valoracion);
            html += `

        </div>
    </div>
</div>
`;
            items++;
            if (items % 4 == 0) {
                html += '</div>'; // Cerrar la fila después de agregar cuatro tarjetas
            }
        }
    }
}

```

```

        }
    }
}
contenedorGeneral.innerHTML = html;
}

function setStars(valoracion){
    html='<div class="rating" id="rating">';
    count=valoracion;
    for(let i=0;i<valoracion; i++){
        if(count<=0.5){
            html += '<span class="fa fa-star-half checked"></span>';
        }else{
            html += ' <span class="fa fa-star checked"></span>'
        }
        count--;
    }
    html += `<p class="valoracionPuertoPrev">` + valoracion + `</p>
</div>`;
    return html;
}

```

- **addToFavorites()**: es la función que proporciona la funcionalidad de añadir un puerto a los favoritos del usuario. Se utiliza para agregar un puerto a la lista de favoritos en el almacenamiento local del navegador. La explicación técnica de lo que hace exactamente se encontrará en el apartado de APIs, concretamente en el de WebStorage.
- **removeFavoritePort()**: se encarga de eliminar el puerto, que el usuario haya seleccionado, de la lista de favoritos. La usaremos para eliminar de la lista de favoritos en el almacenamiento local del navegador. Como en la anterior, esta se explicará en el apartado de APIs.

### 5.3 weather.js

Este fichero es el encargado de poner en marcha la API del tiempo (OpenWeatherAPI) que es usado en el fichero `puerto.html`. Como la pone en funcionamiento se explicará en el apartado específico de su API. Por ahora, veremos el resto de funcionalidades del fichero que no incumben a específicamente a la API. Como en todos los demás ficheros, se empieza leyendo el fichero de puertos de la misma forma que lo hacemos en los otros. En esta ocasión, nos interesarán obtener el parámetro de la URL llamado `portId` y con él, obtener el atributo `addressLocality` del puerto que ha seleccionado el usuario (contenido en la variable `port`). En este fichero solo nos interesa leer ese atributo, no es necesario ningún otro. Este atributo obtenido será guardado en la constante llamada `city`. A partir de ahí, todo el código pertenece específicamente a la API. Como ya hemos mencionado, se explicará como se pone en funcionamiento en su debido apartado.

```

fetch('ports.json')
.then(response => response.json())
.then(data => {
    const ports = data.itemListElement; // Obtener todos los puertos del array

    const urlParams = new URLSearchParams(window.location.search);
    const portId = urlParams.get('portId');
    const port = ports[portId];

```

```

const city = port.address.addressLocality;

const apiKey = '7ee27410d43b852ca993e17f18a42e5a';
const apiUrl =
    "http://api.openweathermap.org/data/2.5/weather?q=" +
    city + "&appid=7ee27410d43b852ca993e17f18a42e5a&units=metric";
fetch(apiUrl)
    .then(response => response.json())
    .then(data => {

        const temperature = data.main.temp;
        const city = data.name;
        const humidity = data.main.humidity;
        const windSpeed = data.wind.speed;

        document.getElementById("tempCity").innerHTML = `Meteorología en
            ${city} `;
        document.getElementById("temperature").innerHTML = `

            ${temperature} grados Celsius. `;
        document.getElementById("humidity").innerHTML = `

            ${humidity} %`;
        document.getElementById("windSpeed").innerHTML = `

            ${windSpeed} km/h`;
    })
    .catch(error => console.error(error));
});

```

## 5.4 email.js

En este fichero se encuentra el código de JavaScript que permite al usuario enviar dudas a nosotros, los creadores. Esto se puede hacer gracias al uso de la API EMAIL JS. Por tanto, como el contenido en su totalidad del fichero pertenece a la API, este se explicará en su debido apartado en las APIs.

# 6 APIs utilizadas

## 6.1 Google Maps

Para poder mostrar al usuario un mapa con marcadores de los puertos que puede ver en nuestra web-app, hemos decidido usar la [API de Google Maps](#). Esta, nos permite introducir en nuestra página un mapa de google maps interactivo donde tenemos la posibilidad de introducir marcadores dentro de este con información que nos interesa mostrar al usuario. Nosotros usamos 2 mapas. El primero y más importante, lo encontramos en la *homepage* y el otro en la pagina donde mostramos la información de un puerto. En este apartado, solo mostraremos como ponemos en funcionamiento la API en la *homepage*, ya que para el segundo caso se repetirá el mismo procedimiento que para el primero pero simplemente se cambiarán los datos introducidos en las funciones propias de la API.

Lo primero que hay que decir es que esta lista de acciones para poder poner en funcionamiento la API la hemos creado siguiendo las recomendaciones de la [documentación](#) de la API de Google Maps usando JavaScript. Estas son las acciones que hemos llevado a cabo para que se muestre el mapa de forma correcta en la página:

1. Lo primero que tenemos que hacer para poder usar la API de Google Maps es añadir una serie de **scripts** en el fichero html donde lo queramos insertar. Estos son propios de la API i se

tiene que poner por obligación. Añadiremos estos enlaces a scripts, el primero de ellos en el <head> del fichero y el segundo en algun puto del <body> del fichero html.

```
<!-- SCRIPT DEL HEAD -->
<script src=
">
https://maps.googleapis.com/maps/api/
js?key=AIzaSyDXSDvjej20MV0XjDBpT4rsznMoZeAfKLM
">
</script>

<!-- SCRIPT DEL BODY-->
<script async defer src=
">
https://maps.googleapis.com/maps/api/
js?key=AIzaSyDXSDvjej20MV0XjDBpT4rsznMoZeAfKLM
&callback=initMap
">
</script>
```

Si nos fijamos, en el link de los archivos, podemos ver la palabra `js?key=` (Hemos separado el link en trozos para que se vea de forma mas clara, aunque en el fichero original esta puerto todo seguido). Esta api nos proporciona una API key que hemos generado. Esta, la tenemos que insertar después del `=`. De esta forma, Google podra verificar nuestra clave de API. También, si nos fijamos en el segundo link, vemos al final de todo un `callback=initmal`. El nombre que aparece después del `=` es el nombre que le deberemos de dar a la función que de JavaScript que genere el mapa. De esta forma la API puede reconocer inequívocamente que queremos generar el mapa en la página.

2. En el fichero `main.js` quedó pendiente explicar el funcionamiento de la función `initMap()`. Como ya dijimos, esta es la encargada de generar el mapa en nuestra web y nos servirá de ejemplo en este apartado para mostrar como se genera un mapa de Google Maps. Como ya dijimos en anteriores apartados, la función recibe por parámetro 4 arrays con los atributos de los puertos ya previamente tratados del fichero `json`. Sabiendo esto, la función realiza las siguientes acciones:
  - Definimos una ubicación de referencia llamada `palma` con una latitud y longitud fijas (39.6952635, 3.0175719). Esta variable nos servirá para la vista inicial que tiene que tener el mapa.
  - Crea un nuevo mapa de Google Maps en el elemento HTML con el ID `map`. El mapa se centra en la ubicación de la variable `palma` y se establece un nivel de zoom de 9 (es un parametro nque nos permite establecer la API de Google Maps). Como ya dijimos, este mapa se insertará en el elemento del fichero `index.html` con el `id=map`.
  - Luego, la función itera sobre los arrays `latit`, `longi`, `capa` y `nomb` pasados por parámetro. Estos arrays contienen los datos de ubicación específicos para los marcadores que mostraremos encima del mapa.
  - Para cada conjunto de coordenadas de latitud y longitud en los arrays `latit` y `longi`, se crea un nuevo marcador en esa ubicación y se muestra en el mapa. El parámetro `map: map` se utiliza para asociar el marcador con el mapa creado anteriormente.
  - A cada marcador se le agrega un evento de clic. Cuando se hace clic en un marcador, se crea una ventana de información (`InfoWindow`) que muestra el nombre del puerto (`nomb[i]`) y su capacidad (`capa[i]`). El contenido de la ventana de información incluye también un enlace a una página del puerto que esta contenida dentro del fichero `puerto.html` con un parámetro de consulta `portId` que se establece en el valor de `i`. Esto

permite pasar el índice del marcador seleccionado a la página `puerto.html` para poder saber, una vez estamos en la página del puerto, qué puerto ha seleccionado el usuario.

- Finalmente, se abre la ventana de información asociada al marcador en el mapa cuando se hace clic en él.

```
function initMap(latit, longi, capa, nomb) {
    const palma = { lat: 39.6952635, lng: 3.0175719 };
    var map = new google.maps.Map(document.getElementById('map'), {
        zoom: 9,
        center: palma
    });
    for (let i = 0; i < latit.length; i++) {
        const marker = new google.maps.Marker({
            position: { lat: latit[i], lng: longi[i] },
            map: map,
        });
        // Agregar evento de clic al marcador
        marker.addListener('click', function () {
            const infoWindow = new google.maps.InfoWindow({
                content: `<a href="puerto.html?portId=${i}">
                    <strong>${nomb[i]}</strong></a><br>Capacidad: ${capa[i]}<br>
                `;
            });
            infoWindow.open(map, marker);
        });
    }
}
```

Esta es la forma que tenemos de mostrar el mapa en la web-app. Creemos que aporta mucho dinamismo a la página el hecho de que sea dinámico. recordar que para el segundo mapa de la web se seguirán los mismos pasos que en este, pero en vez de introducir coordenadas de puertos, se insertarán coordenadas de restaurantes y cafeterías. Por tanto, es cuestión de tratamiento de datos y no de las funciones de la API como tal.

## 6.2 OpenWeatherAPI

En esta sección explicaremos el uso y funcionamiento de la función Javascript encargada de solicitar los datos necesarios a la OpenWeatherAPI con el fin de mostrar la información meteorológica de la localidad donde se encuentra el puerto.

- Lo primero que debemos de hacer para poder acceder a los datos meteorológicos del puerto en el que nos encontramos es obtener la ciudad o localidad en la que se encuentra el puerto. Esta información la podemos obtener cargando el json a partir del id que identifica la página actual.
- Lo siguiente que necesitamos a parte de la localidad del puerto, es la `APIKEY` gratuita que podemos obtener registrándonos en la pagina web.

Una vez tengamos guardados estos dos datos en sus respectivas variables podemos crear la cadena que realizará la petición:

```
const apiUrl ="http://api.openweathermap.org/data/2.5/weather?q=
    "+city+"&appid="+APIKEY+"&units=metric";
fetch(apiUrl)
    .then(response => response.json())
```

```
.then(data => {
})
```

Ahora, a partir de data, podremos acceder a las métricas que nos resulten más interesantes a la hora de enseñar al usuario:

```
const temperature = data.main.temp;
const city = data.name;
const humidity = data.main.humidity;
const windSpeed = data.wind.speed;

document.getElementById("tempCity").innerHTML = `Meteorología en
${city} `;
document.getElementById("temperature").innerHTML = ` 
${temperature} grados Celsius. `;
document.getElementById("humidity").innerHTML = ` 
${humidity} %`;
document.getElementById("windSpeed").innerHTML = ` 
${windSpeed} km/h`;
```

### 6.3 EmailJS

EmailJS es una API de coste gratuito que nos permite la comunicación mediante los usuarios que puedan acceder a la web, mediante el envío de correos. La implementación se muestra a continuación:

```
// Initialize EmailJS with your user ID
emailjs.init("esMeuUserID");

// Function to send email
function sendEmail(event) {
    event.preventDefault(); // Prevent default form submission behavior

    // Get form data
    let name = document.getElementById("firstName").value.trim();
    let email = document.getElementById("email").value.trim();
    let category = document.getElementById("category").value.trim();
    let message = document.getElementById("message").value.trim();

    // Check if form fields are not empty
    if (name && email && message) {
        // Set email parameters
        let params = {
            to_name: "admin",
            from_name: name + "(" + email + ")",
            message: "Category:" + category + "\n" + message,
        };

        // Send email using EmailJS
        emailjs.send("PuertosMallorca", "Puerto_plantilla", params)
            .then(function(response) {
                alert("Email sent successfully!"); // Show success message
                document.getElementById("myForm").reset(); // Reset form
            }, function(error) {
```

```

        console.error("Error sending email:", error); // Log error message
        alert("Error sending email!"); // Show error message
    });
} else {
    alert("Please fill out all fields."); // Show error message
}
// Add event listener to form submission
document.getElementById("myForm").addEventListener("submit", sendEmail);

```

Por orden, realizamos las siguientes operaciones:

- **Inicialización:** inicializamos la API con nuestro ID de usuario, sustituyendo "esMeuUserID" por el ID correspondiente.
- **Obtener datos HTML:** posteriormente, obtenemos los valores de Nombre, email, categoría del mensaje y el mensaje.
- **Envío correo:** finalmente, después de asignar los parámetros con los valores obtenidos anteriormente, enviamos el correo y damos feedback al usuario, diciéndole si el email ha sido enviado correctamente o no.

El resultado es recibir un correo en el email que ha sido registrado en la API, con un formato/plantilla preestablecido con todos los parámetros comentados anteriormente.

## 6.4 WebStorage

Para otorgar al usuario la posibilidad de almacenar sus puertos favoritos, hemos usado la api de WebStorage, más concretamente LocalStorage. Hemos decidido usar localStorage en lugar de sessionStorage porque hemos pensado que lo más completo y lo más adecuado para poder seleccionar puertos favoritos y poderlos volverlos a mirar. De esta forma, al volver a meternos a la página desde el mismo navegador, los datos que habíamos guardado en el local storage se han quedado guardados. En cambio, si hubiésemos usado sessionStorage, cada vez que cerramos el navegador perdemos la información guardada.

Para llevar esto a cabo hemos hecho dos funciones en javascript para los puertos. La primera nos sirve para añadir un puerto a favoritos. Lo primero que hacemos en esta función es establecer la conexión con la localStorage y obtener la matriz con los puertos que tenemos guardados en favoritos. Una vez tenemos esto, creamos el objeto puerto que añadiremos al localStorage. Este objeto está definido por una clave y un valor, de forma que la WebStorage API para encontrar el puerto primero busca la clave, y devuelve el valor. En nuestro caso, tanto la clave como el valor serán el nombre del puerto que lo obtenemos de forma dinámica.

A continuación comprobamos que el puerto que queremos meter a favoritos no exista, es decir, no se encuentre en los favoritos. Lo primero que haremos es mirar si hay puertos con el mismo nombre que el puerto que hemos definido previamente y queremos meter. Si no hay puertos con ese nombre, entonces podemos añadirlo mediante la función push de la WebStorage y `setItem()`. También, tanto si lo añadimos como si no, avisamos al usuario mediante un alert con la información de lo que ha sucedido.

La siguiente función que hemos implementado es la de eliminar un puerto de favoritos. El funcionamiento es el mismo que para añadir un puerto a los favoritos. Primero definimos el puerto en el cual estamos, con el nombre de este como clave y valor, y establecemos la conexión con el localStorage y los puertos que tenemos en favoritos. A continuación, si tenemos puertos en favoritos, volvemos a crear el array con todos estos, pero quitando el que queremos eliminar, y volvemos a añadir los

puertos con `setItem`.

Las funciones que se han explicado anteriormente son las siguientes:

```
function addToFavorites() {
    const favoritePorts = JSON.parse(localStorage.getItem('favoritePorts')) || [];
    const puerto = { name: port.name, code: port.name }; // Define el puerto
                                                       // a agregar

    // Verificar si el puerto ya existe en la lista de favoritos
    const portExists = favoritePorts.some(favoritePort =>
        favoritePort.code === puerto.code);

    if (!portExists) {
        favoritePorts.push(puerto);
        localStorage.setItem('favoritePorts', JSON.stringify(favoritePorts));
        alert(`El puerto ${puerto.name} ha sido añadido a favoritos`);
    } else {
        alert(`El puerto ${puerto.name} ya está en la lista de favoritos`);
    }
}

function removeFavoritePort() {
    // Obtener los puertos favoritos almacenados en localStorage
    const puerto = { name: port.name, code: port.name };
    const favoritePorts = JSON.parse(localStorage.getItem('favoritePorts'));
    // Verificar si hay puertos favoritos almacenados

    if (favoritePorts && favoritePorts.length > 0) {
        // Encontrar y eliminar el puerto de favoritos
        const updatedPorts = favoritePorts.filter(p => p.code !== puerto.code);

        // Actualizar los puertos favoritos en localStorage
        localStorage.setItem('favoritePorts', JSON.stringify(updatedPorts));

        // Mostrar mensaje de éxito o realizar alguna acción adicional si es necesario
        alert(`El puerto ${port.name} ha sido eliminado de favoritos.`);
    } else {
        alert('No hay puertos favoritos almacenados');
    }
}
```

## 7 Web Semántica

Es importante hacer que nuestra web-app sea semántica utilizando, ya que esto nos permite enriquecer la información que presentamos en la página web de una manera estructurada y comprensible tanto para los usuarios como para los motores de búsqueda.

En nuestro caso, hemos decidido usar **JSON-LD**, que significa JavaScript Object Notation for Linked Data (Notación de Objetos JavaScript para Datos Enlazados). Es un formato de datos que permite la adición de metadatos y contexto semántico a la información presentada en una página web. Al utilizar **JSON-LD**, podemos especificar la semántica de los elementos en nuestra pá-

gina web, como el tipo de entidad, las propiedades y sus valores. Para ello, hemos añadido en el elemento <head> de cada uno de nuestros archivos html un script de JSON-LD (<script type='application/ld+json'></script>). Aunque tenemos 3 archivos html, hemos generado 2 tipos de scripts, ya que 2 de ellos son idénticos.

- **Script estático:** este primer script lo hemos incluido tanto al archivo `index.html` como al archivo `sobreNosotros.html`. Lo hemos generado mediante [esta](#) página. Esta, se nos ha proporcionado en el sitio web de la asignatura. El script añadido es el siguiente:

```
<script type='application/ld+json'>
{
  "@context": "http://www.schema.org",
  "@type": "WebSite",
  "name": "Puertos Mallorca",
  "alternateName": "puertos de mallorca",
  "url": "puertosmallorca.com"
}
</script>
```

Los atributos que se muestran en el script tienen diversos significados. Aquí explicamos cada uno de ellos:

- `@context`: Especifica el contexto en el que se interpretan las propiedades del objeto. En nuestro caso, hemos utilizado "http://www.schema.org"(ya que el fichero json también lo hemos realizado mediante esa estructura) como el contexto, lo que indica que las propiedades se rigen por el esquema de datos de Schema.org.
- `@type`: Define el tipo de entidad del objeto. En este caso, se especifica que el tipo de entidad es un "WebSite"(sitio web).
- `name`: Indica el nombre del sitio web. En este ejemplo, el nombre es "Puertos Mallorca".
- `alternateName`: Proporciona un nombre alternativo para el sitio web. En este caso, se utiliza "puertos de mallorca" como nombre alternativo.
- `url`: Especifica la URL del sitio web. En nuestro caso es "puertosmallorca.com".

- **Script dinámico:** por lo que respecta al segundo script, este se ha añadido al <head> de del archivo `puerto.html`. Decimos que es dinámico porque este cambia en función del puerto que haya seleccionado el usuario. Lo que hacemos es que el contenido de este sea el trozo del fichero json perteneciente al puerto que se ha seleccionado. Para llevarlo a cabo, en el fichero `port.js`, concretamente en la función `infoPort()` (explicada en apartados anteriores de la documentación), obtenemos el objeto json del puerto y lo transformamos a un string para poder insertarlo en el elemento del archivo html con `id = port-json`. De esta forma, cuando el usuario entra en la pagina de un puerto, en el script de JASON-LD se insertará solo el trozo del fichero json que corresponda con el puerto. Para llevar esto a cabo, se han usado las siguientes intrucciones en la función y fichero anteriormente mencionados:

```
var port = ports[portId]; // Variable que contiene el puerto
                           // leído del fichero json
var puertoJsonString = JSON.stringify(port); //Convertimos
                                              //el puerto a string

//Actualizamos el valor del script en el head para la web semántica
var scriptElement = document.getElementById('port-json');
scriptElement.textContent = puertoJsonString;
```

Por otro lado, lo único que añadimos al fichero html es el siguiente código:

```
<script type="application/ld+json" id="port-json"></script>
```

## 8 Inclusión de media: SVG y video propios

Hemos incluido 2 archivos de media propios sin contar imágenes. Las dos inclusiones han sido las siguientes:

- **Ancla SVG:** com archivo SVG hemos incluido nuestro logo de la página. Esta se muestra en cualquiera de las 3 páginas que hemos desarrollado, concretamente en la barra de navegación en la parte superior izquierda. Para incluirlo simplemente hemos añadido un objeto `img` en el cual su `src` es el archivo `ancla.svg`. La sección en concreto donde incluimos el SVG es la siguiente:

```
<a class="navbar-brand ms-3" href="index.html" id="contenedorLogo">
  
  <a class="navbar-brand mb-0 h1" href="#index.html">Puertos mallorca</a>
</a>
```

El resto de clases de bootstrap ya se han explicado en su respectiva sección del documento.

- **Video presentación:** tal y como requería la práctica, hemos añadido un archivo de video propio. Este es el que se muestra en el archivo `sobreNosotros.html` donde los creadores de la web-app nos presentamos al usuario. Para la inclusión de un video propio, hemos usado las siguientes clases específicas para ello:

- `<div class="video-container">`: La clase `video-container` se utiliza para aplicar estilos específicos al contenedor del video.
- `<video width="640" height="360" controls>...</video>`: El elemento `<video>` es una etiqueta HTML5 que se utiliza para insertar un video en la página. Los atributos `width` y `height` se utilizan para establecer las dimensiones del video en píxeles. El atributo `controls` muestra los controles de reproducción estándar para el video, como el botón de reproducción, pausa y barra de progreso.

A continuación se muestran las líneas de código específicas donde se incluye el archivo de video en el archivo `sobreNosotros.html`.

```
<div class="video-container">
  <video width="640" height="360" controls>
    <source src="video.mp4" type="video/mp4">
    <source src="video.webm" type="video/webm">
    Tu navegador no admite la etiqueta de video.
  </video>
</div>
```

## 9 Evaluación de la Web-App

En esta última sección, de la documentación, veremos diferentes evaluaciones sobre la web-app. Utilizaremos 3 evaluadores que se nos han proporcionado en la página de la asignatura:

## 9.1 nibbler.insites.com

Es una herramienta en línea que proporciona análisis y evaluación de sitios web en función de diferentes aspectos técnicos y de rendimiento. Al ingresar la URL de un sitio web en el campo de búsqueda, Nibbler generará un informe detallado que muestra la puntuación y comentarios sobre varios aspectos clave del sitio.

Una vez realizado la evaluación a nuestra URL, estos han sido los resultados obtenidos: A continu-

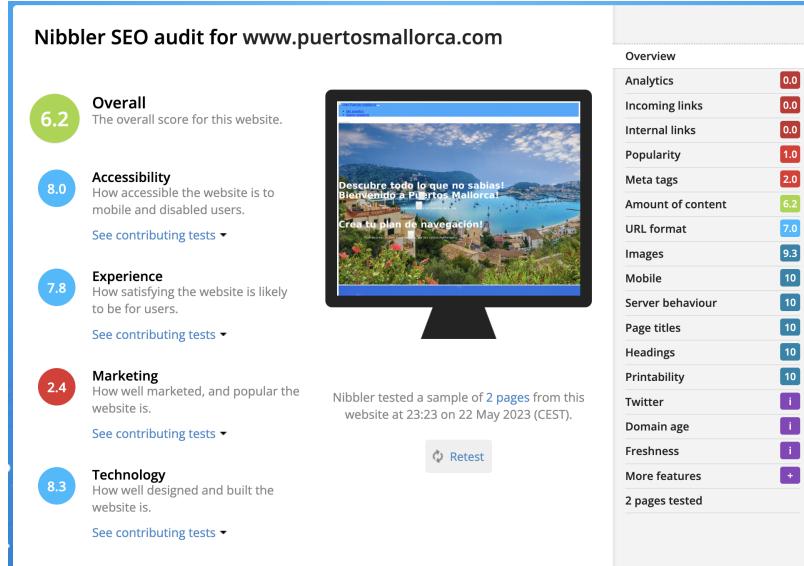


Figura 16: Imagen que muestra la valoración general de nuestra web por parte de Nibbler

ación se explicarán el porqué de cada nota en cada apartado:

### 9.1.1 Accesibilidad

Apartado en el que se valora la accesibilidad del sitio web, tanto en dispositivos móviles como en ordenadores. Si nos fijamos más concretamente en los aspectos que se han evaluado en esta sección, vemos los siguientes resultados: Como podemos ver, todos los apartados tiene una buena califica-

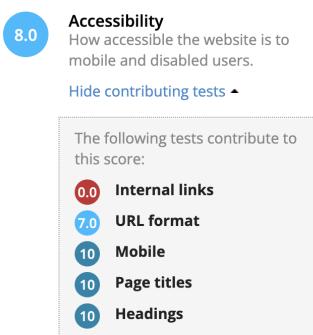


Figura 17: Resultados del apartado de accesibilidad

ción menos el de Internal Links. Esto se debe a que muchos de los links que usamos en nuestra web, no tienen un texto de referencia que indique a donde lleva. Esto se debe principalmente a los links que usamos en los mapas para trasladar al usuario a las páginas web tanto de los puertos

como de los restaurantes y cafeterías. En esos casos es inevitable, ya que tenemos muy poco espacio como para poner un texto suficientemente descriptivo. Por tanto, sabemos el porque de esa calificación pero no se mejora ya que si queremos hacer la web tal y como no gusta que esté, es inevitable. Por otra parte, tenemos un 7.0 y no un 10 por el nombre de la URL una vez entramos a una de las secciones de la web. No debería tener una extensión de fichero pero en este caso la tiene ([www.puertosmallorca.com/sobreNosotros.html](http://www.puertosmallorca.com/sobreNosotros.html)).

### 9.1.2 Experiencia

En este apartado, la web evalúa el grado de satisfacción de la web-app de cara a los usuarios finales que la usarán.

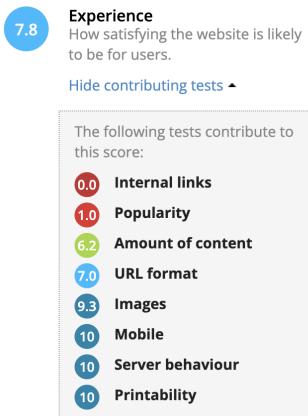


Figura 18: Resultados del apartado de Experiencia

La razón del Internal Links ya la hemos explicado en el anterior apartado. Por otro lado, la popularidad es una cosa que no podemos controlar, ya que la página registra el número de visitas mensuales de la web. Por otra parte, la única otra nota que podría no ser del todo buena es la cantidad de contenido. La web cuenta el número de palabras por página que existe. A partir de aquí, estas pueden ser consideradas muchas o pocas, pero lo importante es que el contenido existente sea de calidad.

### 9.1.3 Marketing

Este apartado es el de menor calificación con diferencia. Esto se debe a que evalúa el marketing de la web y como la creación de la web-app no incluye mucho de eso, por eso tiene una evaluación tan baja.

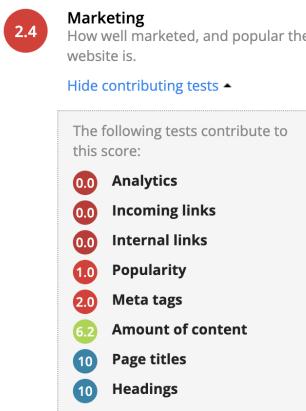


Figura 19: Resultados del apartado de marketing

Obviando los apartados ya explicados, el apartado de Incoming links tiene una calificación tan baja porque no hay ninguna web externa que use nuestro link para redirigir a sus usuarios a la nuestra. Esto es una cosa que no podemos controlar. Por otra parte, el apartado de Analytics tiene tan baja puntuación porque no usamos ningún software de analítica. Finalmente, la popularidad, no la podemos controlar y los Meta tags, es porque del apartado de Web semántica hemos puesto lo mínimo para que estuviera incluido en la práctica, no nos hemos extendido mucho.

### 9.1.4 Tecnología

Finalmente, en el apartado de tecnología, se evalúa cómo de bien diseñada esta nuestra web app. Como se puede observar este es nuestro mejor apartado, ya que creemos que el diseño de esta ha sido nuestro punto fuerte.

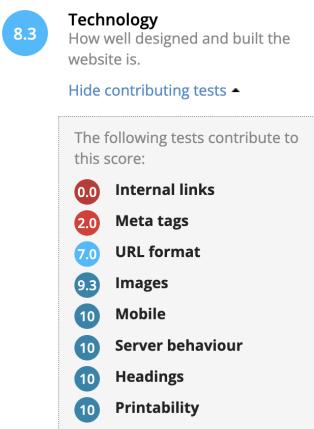


Figura 20: Resultados del apartado de marketing

El motivo de las malas notas obtenidas en este apartado, ya se han explicado anteriormente.

## 9.2 pagespeed.web.dev

Es una herramienta proporcionada por Google que evalúa el rendimiento de un sitio web y ofrece sugerencias para mejorarlo. Está centrado en la velocidad de carga y rendimiento general de un sitio web, lo cual es crucial para brindar una buena experiencia de usuario y mejorar el posicionamiento en los motores de búsqueda.



Figura 21: Imagen que muestra la valoración general por parte de pagespeed en ordenador



Figura 22: Imagen que muestra la valoración general por parte de pagespeed en móviles

En las anteriores imágenes podemos observar el rendimiento que tiene nuestra web app tanto en móviles como en ordenadores. Obviamente el rendimiento es ligeramente superior. Se puede observar una nota digna, aunque somos conscientes de que en algunos ficheros JavaScript, podríamos haber ahorrado bastantes lecturas mediante la optimización del código desarrollado. Igualmente, creemos que no son malos números, por eso los hemos dejado de esta forma.

## 9.3 validator.schema.org

Es una herramienta en línea que permite validar y verificar la estructura y el marcado de datos estructurados (structured data) según las especificaciones del proyecto Schema.org. Esta web nos servirá para verificar que no haya errores en el código que ya hemos explicado en el apartado de Web Semántica de este mismo documento. Una vez realizada la validación, estos han sido los resultados obtenidos:

Como se puede observar, el contenido de Schema.org añadido a nuestra web no contiene errores ninguno (tal y como nos indica la web).

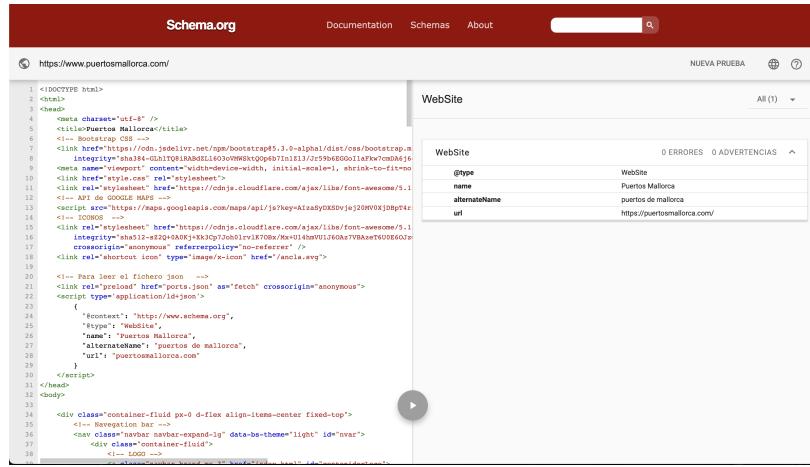


Figura 23: Imagen que muestra el resultado de la validación de Schema.org

## 10 Conclusiones

En conclusión, en esta práctica, hemos creado una web app sobre los puertos de Mallorca utilizando diversas tecnologías y recursos. Mediante HTML, JavaScript y Bootstrap, hemos construido una interfaz web interactiva y atractiva. Además, hemos aprovechado las capacidades de Schema.org para agregar datos estructurados, lo que mejora la visibilidad y comprensión de nuestro contenido por parte de los motores de búsqueda.

Para enriquecer la experiencia del usuario, hemos integrado APIs como Google Maps y OpenWeather API, lo que nos ha permitido mostrar ubicaciones de los puertos en un mapa interactivo y ofrecer información meteorológica en tiempo real. Esto ha proporcionado una funcionalidad adicional y una experiencia más completa para los visitantes de nuestra web app.

En cuanto a la inclusión de medios, hemos utilizado archivos SVG para gráficos vectoriales escalables y videos propios para mostrar contenido multimedia. Estos elementos visuales han enriquecido la presentación de la información y han brindado una experiencia visual atractiva para los usuarios.

Además, hemos utilizado archivos JSON para almacenar y administrar los datos de los puertos y otros tipos de datos (cafeterías y restaurantes externos). Esto nos ha permitido organizar y acceder de manera eficiente a la información relevante, facilitando la actualización y expansión de nuestra web app en el futuro.

En resumen, mediante la combinación de HTML, JavaScript, Bootstrap, Schema.org, APIs y el uso de archivos SVG, videos y JSON, hemos creado una web app sobre los puertos de Mallorca que ofrece una experiencia interactiva, informativa y visualmente atractiva. Esta práctica nos ha brindado la oportunidad de aplicar diversas tecnologías y recursos para desarrollar una aplicación web completa y funcional.