

Documentación Técnica - Entrega Final

Calas de Mallorca



Integrantes:

Juan José Nieto García - 43471537L

Carlos Veny Carmona - 43194073G

Joan Alcover Lladó - 45187596W

Índice

Introducción	3
Página principal - index.html	5
Barra de navegación	5
Carousel	6
Noticias y Comentarios	7
API Twitter	7
Comentarios recientes	7
Covid	8
API Media	8
Vídeo	8
Audio	9
Footer	9
Página listado	10
Filtros	10
Ordenaciones	11
Calas/fila	11
Tarjetas de cala	12
Animaciones	13
Integración otros JSON	13
Página mapa	15
API OpenLayers 3	15
API Geolocalización	15
Calas cercanas	16
API Bing	17
Página producto (cala)	18
Lectura JSON	18
Carousel imágenes	18
Comentarios	19
Mapa	21
API Meteorología	21
Integración otros JSON	22
Integración dinámica Pueblos	24
Página favoritos	25
API WebStorage	25
Web Semántica	26
Estructura de directorios	29
JSON	30
Metadatos	31
Evaluación en dispositivos	32
Lighthouse	32
Herramientas	33
Opinión	35

Introducción

El objetivo de nuestro proyecto es realizar un recopilatorio de calas de la isla en la que nos encontramos, Mallorca. Para ello, hemos construido una aplicación web funcional para una amplia mayoría de navegadores y dispositivos de todos los tamaños la cual incorpora CSS, Javascript y HTML junto con ficheros JSON, que son las fuentes de datos.

Nuestra página web permite a los diferentes usuarios la posibilidad de ver la gran mayoría de las calas de Mallorca clasificadas por diferentes criterios a gusto de los mismos. La aplicación es totalmente gratis y está disponible en todas las plataformas y sistemas operativos.

La web tiene diferentes maneras de visualizar las calas: se pueden filtrar las calas según varios atributos de estas, también hay un mapa de Mallorca con todas las calas marcadas en el mapa, y por último un apartado de favoritos donde cada usuario podrá visualizar aquellas calas marcadas como favoritas (tanto listado como mapa). También hemos realizado integraciones con JSON de otros grupos.

El objetivo de esta documentación técnica es tratar de explicar de manera técnica las tecnologías y los procesos seguidos para llevar a cabo esta webapp. Para estructurar el documento se va a dividir la documentación según las diferentes páginas que componen la página web y al final se hablarán de temas de carácter general que tratan sobre toda la web, como son metadatos, web semántica, JSON...

La aplicación web se puede consultar aquí: <https://calasdemallorca.netlify.app/>

Página principal - index.html

Se ha seguido una plantilla de [Wise](#) para hacer la página principal. La hemos ampliado y adaptado a las necesidades de nuestra webapp, conjuntamente con la grid de Bootstrap.

Barra de navegación

La barra de navegación es común a todas las páginas y se mantiene fija siempre en la parte superior de la pantalla. Se ha usado una plantilla de Bootstrap llamada "Navbar-Fixed" y se ha modificado la ubicación de los objetos para adaptarlos a nuestro gusto.

Hemos añadido transiciones al desplegar el menú cuando la pantalla es demasiado pequeña y hemos puesto el logo de nuestra webapp en el medio. Este logo usa el formato SVG y lo hemos creado mediante una herramienta web de edición de SVGs online.

También se ha modificado el CSS de Bootstrap para que la barra de navegación colapse cuando nosotros queramos ya que en la plantilla tarda demasiado y se sobrescriben nuestros títulos del menú.

```
<!-- menu de navegacion -->
<header>
  <nav class="navbar navbar-expand-xl navbar-light fixed-top bg-blue">
    <!--xl nuevo -->
    <div class="container-fluid">
      <a class="navbar-brand" href="index.html">
        
      </a>
      <button class="navbar-toggler btn-navbar-toggler" type="button" data-bs-toggle="collapse"
        data-bs-target="#navbarCollapse" aria-controls="navbarCollapse" aria-expanded="false"
        aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <!--btn-navbar-toggler nuevo -->

      <div class="collapse navbar-collapse" id="navbarCollapse">
        <ul class="navbar-nav ms-auto">
          <li class="nav-item">
            <a class="nav-link active" aria-current="page" href="#">Inicio</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="listado.html">Listado</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="mapa.html">Mapa</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="favoritos.html">Favoritos</a>
          </li>
          <li class="nav-item">
            <a class="nav-link disabled" href="#" tabindex="-1" aria-disabled="true">Extra</a>
          </li>
        </ul>
      </div>
    </div>
  </nav>
</header>
<!-- fin menu de navegacion -->
```

Carousel

El carousel de imágenes de la página principal se carga de manera dinámica y aleatoria. Las imágenes que aparecen no son siempre las mismas y dependiendo de la hora del día se cargan imágenes nocturnas o diurnas. Esto es posible porque disponemos de un fichero JSON propio que contiene dos arrays de imágenes. El primero contiene unas 20 fotos de calas realizadas de día y el segundo contiene 15 fotos de calas realizadas de noche. Usamos la API de OpenWeatherMap (desarrollada más adelante) para saber a qué hora se pone el sol y luego en Javascript obtenemos la hora actual del usuario para saber de qué array hay que cargar las imágenes. A continuación se puede ver el fichero JSON y el código que selecciona las imágenes según la hora.

```
"index": {
  "titulo": "CALAS DE MALLORCA",
  "subtitulo": "Una webapp para descubrir los rincones más bonitos de la costa mallorquina",
  "carousel": [
    "imagenes/carousel/carousel1.jpg",
    "imagenes/carousel/carousel2.jpg",
    "imagenes/carousel/carousel3.jpg",
    "imagenes/carousel/carousel4.jpg",
    "imagenes/carousel/carousel5.jpg",
    "imagenes/carousel/carousel6.jpg",
    "imagenes/carousel/carousel7.jpg",
    "imagenes/carousel/carousel8.jpg",
    "imagenes/carousel/carousel9.jpg",
    "imagenes/carousel/carousel10.jpg",
    "imagenes/carousel/carousel11.jpg",
    "imagenes/carousel/carousel12.jpg",
    "imagenes/carousel/carousel13.jpg",
    "imagenes/carousel/carousel14.jpg",
    "imagenes/carousel/carousel15.jpg",
    "imagenes/carousel/carousel16.jpg",
    "imagenes/carousel/carousel17.jpg",
    "imagenes/carousel/carousel18.jpg"
  ],
  "carousel-n": [
    "imagenes/carousel/carousel-n1.jpg",
    "imagenes/carousel/carousel-n2.jpg",
    "imagenes/carousel/carousel-n3.jpg",
    "imagenes/carousel/carousel-n4.jpg",
    "imagenes/carousel/carousel-n5.jpg",
    "imagenes/carousel/carousel-n6.jpg",
    "imagenes/carousel/carousel-n7.jpg",
    "imagenes/carousel/carousel-n8.jpg",
    "imagenes/carousel/carousel-n9.jpg",
    "imagenes/carousel/carousel-n10.jpg",
    "imagenes/carousel/carousel-n11.jpg",
    "imagenes/carousel/carousel-n12.jpg",
    "imagenes/carousel/carousel-n13.jpg",
    "imagenes/carousel/carousel-n14.jpg"
  ]
},
}
```

```
($.getJSON("https://api.openweathermap.org/data/2.5/onecall?lat=39.550598&lon=2.657367",
  function(json){
    var sr = new Date(json.current.sunrise * 1000);
    var sunrise = (sr.getHours()+1);
    var ss = new Date(json.current.sunset * 1000);
    var sunset = (ss.getHours()+1);
    var now = new Date().getHours();

    //elegir fotos nocturnas segun la hora
    var path = "carousel-n"; //carousel noche
    if (now > sunrise && now < sunset) { //dia
      path = "carousel" //carousel dia
    }
    var fotosRandom = [];
    while (fotosRandom.length < 5) {
      var r = Math.floor(Math.random() * (obj["index"][path].length));
      if (fotosRandom.indexOf(r) == -1) fotosRandom.push(r);
    }
    for (i = 0; i < 5; i++) {
      $('#carousel-foto-${i}`).attr("src", obj["index"][path][fotosRandom[i]]);
    }
  });
});
```

Noticias y Comentarios

Esta sección contiene un widget de Twitter y una recopilación de los comentarios más recientes que se han puesto en las calas.

API Twitter

Para mostrar los tweets más recientes del Consell de Mallorca hemos usado la API de Twitter, que se configura desde la siguiente web: <https://publish.twitter.com/#> . Simplemente se pega el link que quieres que se muestre en el widget (en nuestro caso el timeline del Consell de Mallorca), seleccionas el tipo de widget y luego se tiene que configurar las dimensiones, el color, idioma... Esto genera un código en HTML que al pegarlo en nuestra web ya se incrusta la API.

```
<div class="col-xl-6 text-center mg-bt-md">
  <a class="twitter-timeline" data-lang="es" data-width="500" data-height="600" data-theme="light"
    href="https://twitter.com/ConselldeMca?ref_src=twsrc%5Etfw">
    Tweets by ConselldeMca</a>
  <script async src="https://platform.twitter.com/widgets.js" charset="utf-8"></script>
</div>
```

Comentarios recientes

Este apartado contiene un Carousel sacado de internet que se rellena de manera dinámica con los 9 últimos comentarios que se han añadido a la webapp (de cualquier cala). Esto es posible porque disponemos de un servidor externo donde almacenamos los comentarios (explicado en detalle en el apartado de comentarios). En este caso simplemente se leen los 9 últimos comentarios y se muestran en cada tarjeta del Carousel, indicando el nombre, el texto, la fecha y la valoración del comentario. La plantilla está sacada de Codepen ([link](#)).

```
//Funcion que rellena el carousel de comentarios de manera dinámica
function escribirComentariosIndex(objComentarios) {
  for (i = 0; i < 9; i++) {
    tarjeta = document.createElement("div");
    tarjeta.setAttribute("class", "carousel_item");
    comentario = document.createElement("div");
    comentario.setAttribute("class", "comment");

    div_img = document.createElement("div");
    div_img.setAttribute("class", "comment-avatar");
    img = document.createElement("img");
    img.setAttribute("src", "imagenes/avatar.png");
    img.setAttribute("alt", "Comentario")
    div_img.appendChild(img);

    caja = document.createElement("div");
    caja.setAttribute("class", "comment-box");

    nombre = document.createElement("div");
```

Covid

El Widget del Covid está sacado de la web del Cercle d'Economia de Mallorca, <https://cerclemallorca.es/covid-data-balearic-islands/>, y desde su propia página te enseñan como usar su API para integrar el widget en tu propia web. Simplemente se genera un código HTML, en el cual se definen las dimensiones y el borde para que quede a nuestro gusto una vez incrustado en la web. En nuestro caso le hemos aplicado nuestro propio CSS para que sea web responsive.

```
<div class="row">
  <iframe src="https://cerclemallorca.es/?covid-widget&source=uh" loading="lazy" class="covid19-widget"></iframe>
</div>
```

API Media

Hemos usado la API Media de HTML para incrustar tanto vídeo como audio en nuestra aplicación web.

Vídeo

Hemos editado un pequeño vídeo de playas y rincones de Mallorca y lo hemos convertido a mp4 y webm para que sea compatible con el máximo número de navegadores. Como nuestro hosting tiene el ancho de banda limitado cargamos el video dinámicamente desde el Javascript para que así cargue después de las imágenes del Carousel. Si no hacíamos esto se intentaba cargar todo al mismo tiempo y al final tardaba varios minutos en terminar de cargar el Carousel y el vídeo. De esta manera primero se cargan las imágenes (que tardan muy poco) y luego ya se empieza a cargar el vídeo (al estar más abajo en la web ya se ha cargado cuando el usuario lo visualiza). El video tiene autoplay y se visualiza en bucle, para que sea todo automático. Además, está siempre muteado y no tiene controles, para tener una interfaz limpia y agradable.

```
<video id="video" autoplay loop muted playsinline class="video-calas">
  <!-- <source src="videos/video-paraiso.mp4" type="video/mp4" /> creado en js para que cargue despues de carousel-->
  <!-- <source src="videos/video-paraiso.webm" type="video/webm" /> creado en js para que cargue despues de carousel-->
  ¡Lo sentimos! :( Tu navegador no soporta mp4 ni webm.
</video>
```

```
//Video se carga despues del carousel
var src = document.createElement("source");
src.setAttribute("src", "videos/video-paraiso.mp4");
src.setAttribute("type", "video/mp4");
var src2 = document.createElement("source");
src2.setAttribute("src", "videos/video-paraiso.webm");
src2.setAttribute("type", "video/webm");
document.getElementById("video").appendChild(src);
document.getElementById("video").appendChild(src2);
```

Audio

En la página principal también tenemos audio. Se trata de un sonido de las olas del mar que dura unos 5 segundos y suena con el volumen bastante bajo para no molestar. El audio empieza automáticamente y tampoco tiene controles. Una vez ha finalizado no se puede volver a reproducir. Debido a la política que tienen ciertos navegadores con el autoplay, el audio solo funciona en Firefox y en Edge, en los demás navegadores hay que interactuar con la página para que suene. El audio está en formato mp3 y ogg.

```
<audio id="index-audio">
  <source src="audio/mar.mp3" type="audio/mpeg">
  <source src="audio/mar.ogg" type="audio/ogg">
  ¡Lo sentimos! :( Tu navegador no soporta mp3 ni ogg.
</audio>
```

Footer

El footer está basado en una plantilla de Bootstrap ([link](#)), que se ha modificado para posicionar los elementos a nuestro gusto, cambiar el texto, añadir enlaces a páginas de interés y también hemos puesto un botón que te lleva a la parte superior de la página. Hemos aplicado CSS propio para que sea Web Responsive de la manera que deseábamos. Este footer es común a todas las pantallas de nuestra web.

```
<div class="col-sm-6 col-md-3 item fix-column">
  <h3>Enlaces de interés</h3>
  <ul>
    <li><a href="https://www.booking.com/index.es.html" target="_blank">Booking</a></li>
    <li><a href="https://okrentacar.es/" target="_blank">Rent a Car</a></li>
    <li><a href="https://getbootstrap.com/" target="_blank">Bootstrap</a></li>
  </ul>
</div>
<div class="col-md-6 item text">
  <h3>Calas de Mallorca</h3>
  <p>Una pequeña recopilación de las mejores Calas de Mallorca hecha por un grupo de estudiantes de Ingeniería Informática en la Universitat de les Illes Balears. El formato de presentación es una webapp que usa HTML, CSS y JavaScript.<del></del></p>
</div>
<div class="col-sm-6 col-md-3 item ir-arriba-titulo fix-column">
  <h3>Ir Arriba</h3>
  <a href="#"><i class="icon ion-arrow-up-c ir-arriba-flecha"></i></a>
</div>
```

Página listado

Filtros

Para poder organizar y mostrar las calas que se desean tenemos un sistema de filtros y ordenaciones. Se puede filtrar por servicios y municipios, y luego estos resultados se pueden ordenar según el nombre o la valoración. El código Javascript para aplicar los filtros está completamente hecho por nosotros, salvo alguna función de ordenar o de manejo de strings, de la cual hemos buscado información por internet. El contenido de los desplegables de filtros y municipios se cargan dinámicamente desde el Javascript y, además, solo aparecen los municipios que contienen alguna cala (se lee todo el JSON para saber qué municipios hay).

Al seleccionar un filtro/municipio se añade ese filtro a la lista de filtros que hay más abajo, la cual puede contener como máximo 5 filtros (hemos decidido este número porque no hay tantas calas como para necesitar más filtros). Si se excede de 5 filtros sale un aviso (plantilla de [Bootstrap](#)) que indica el error. Los filtros aplicados se pueden quitar con la cruz, ya que esto ejecuta un método que busca en el div de los filtros aplicados ese filtro en cuestión y lo elimina. Después de añadir o eliminar filtros siempre se recargan todas las tarjetas de las calas.

```
//MUNICIPIOS
//si no hay filtro de municipios se añaden todas las calas
if (municipios.length == 0) for (t = 0; t < obj.length; t++) calas.push(t);
else {
  for (j = 0; j < obj.length; j++) {
    if (tema == "calas" && municipios.includes(obj[j]["geoposicionamiento1"]["city"])) calas.push(j);
    else if (tema == "actividades" && municipios.includes(obj[j]["tipus"])) calas.push(j);
  }
}

//FILTROS
if (filtros_v.length != 0) {
  for (j = 0; j < obj.length; j++) {
    var añadir = true; //true si una cala cumple todos los filtros
    for (n = 0; n < filtros_v.length && añadir; n++) {
      //$('#prueba').html(filtros_v[filtros_v.length - 1]);
      if (filtros_t[n] == "tipusCala") {
        if (!obj[j]["dadesPropies"]["serveis"]["tipusCala"].toLowerCase().includes(filtros_v[n])) {
          añadir = false;
        }
      }
      else if (obj[j]["dadesPropies"]["serveis"][filtros_t[n]] == "No") añadir = false;
    }
    if (!añadir) { //eliminar del listado de calas a mostrar
      if (calas.includes(j)) calas.splice(calas.indexOf(j), 1);
    }
  }
}
```

```

if (document.getElementById(id) == null) { //si el boton ya existe no lo crea
//revisar que no haya mas de 5 filtros aplicados o ya exista el aviso
if (document.getElementById("selected-filters").childElementCount == 6) {
    if (document.getElementById("alerta-filtros") == null) {
        var alerta = document.createElement("div");
        alerta.setAttribute("id", "alerta-filtros");
        alerta.setAttribute("class", "alert alert-warning alert-dismissible fade show fs-6");
        alerta.setAttribute("role", "alert");
        alerta.innerHTML = "Aviso, el número máximo de filtros aplicados es 5. Por favor, elimine un filtro.";
        var bt = document.createElement("button");
        bt.setAttribute("type", "button");
        bt.setAttribute("class", "btn-close");
        bt.setAttribute("data-bs-dismiss", "alert");
        bt.setAttribute("aria-label", "Close");
        alerta.appendChild(bt);
        document.getElementById("caja-listado").appendChild(alerta);
        //alert("El número máximo de filtros aplicados es 5");
    }
}
}

```

```

//Funcion que elimina el filtro seleccionado
function eliminarFiltro(id) {
    var alerta = document.getElementById("alerta-filtros");
    if (alerta != null) {
        alerta.remove();
    }
    var del = document.getElementById(id);
    del.remove();
    filtrar_ordenar();
}

```

Ordenaciones

Una vez ya se han aplicado los filtros se puede ordenar al gusto del usuario. Para realizar esto tenemos una función que ordena alfabéticamente (en caso de ser orden descendente simplemente se invierte el array con la función "reverse()") y también ordena por valoración, con una línea de código sacada de StackOverflow que ordena un array de números. Una vez tenemos el array con las calas ordenadas se vuelven a recargar todas las calas.

```

switch (txt) {
    case "nombre-down":
        calas_nombre.sort();
        break;
    case "nombre-up":
        calas_nombre.sort();
        calas_nombre = calas_nombre.reverse();
        break;
    case "valoracion-down":
        calas_valoracion.sort((a, b) => b - a); // For descending sort
        nombre = false;
        break;
    case "valoracion-up":
        calas_valoracion.sort((a, b) => a - b); // For ascending sort
        nombre = false;
        break;
}

```

In ES6, you can simplify this with arrow functions:

```

numArray.sort((a, b) => a - b); // For ascending sort
numArray.sort((a, b) => b - a); // For descending sort

```

Calas/fila

Para poder seleccionar la cantidad de tarjetas que se visualizan en cada fila hemos combinado CSS y Javascript. Tenemos una clase en CSS para cada tamaño (1, 2 y 3) y luego con código Javascript hacemos que se modifique la clase de las tarjetas al pulsar los botones. También tenemos una función que se ejecuta al modificar el tamaño de la ventana, para asegurarnos de que cuando la ventana es más pequeña solo se pueden seleccionar las opciones pertinentes (en móviles solo 1 cala/fila, tablets hasta 2 calas/fila y ordenadores hasta 3 calas/fila). Esta función cambia las clases según el tamaño y elimina los botones que no se pueden usar.

```
//Funcion que cambia los objetos por fila segun la anchura de la ventana
function cambiarObjetosPorFila() {
    var w = document.documentElement.clientWidth;
    if (w < 700) {
        objetosPorFila(1);
        document.getElementById("objetos-2").classList.add('btn-delete');
        document.getElementById("objetos-3").classList.add('btn-delete');
        document.getElementById("objetos-2").removeAttribute("onclick");
        document.getElementById("objetos-3").removeAttribute("onclick");
    }
    else if (w < 1400) {
        if (!document.getElementById("objetos-1").classList.contains('btn-seleccionado')) {
            objetosPorFila(2);
        }
        document.getElementById("objetos-3").classList.add('btn-delete');
        document.getElementById("objetos-3").removeAttribute("onclick");
        document.getElementById("objetos-2").classList.remove('btn-delete');
        document.getElementById("objetos-2").setAttribute("onclick", "objetosPorFila(2)");
    }
    else {
        document.getElementById("objetos-2").classList.remove('btn-delete');
        document.getElementById("objetos-3").classList.remove('btn-delete');
        document.getElementById("objetos-2").setAttribute("onclick", "objetosPorFila(2)");
        document.getElementById("objetos-3").setAttribute("onclick", "objetosPorFila(3)");
    }
}
function resizeListado() {
    window.addEventListener("resize", cambiarObjetosPorFila);
}
```

```
<div class="filtrosObjetos-container-item">
    <div id="label-obj-fila" class="caja-filtrosObjetos fondo-azulFuerte"><strong>Calas/fila:&nbsp;</strong>
        <button id="objetos-1" class="btn btn-light btn-dim" type="button" onclick="objetosPorFila(1)">1</button>
        <button id="objetos-2" class="btn btn-light btn-dim btn-seleccionado" type="button"
            onclick="objetosPorFila(2)">2</button>
        <button id="objetos-3" class="btn btn-light btn-dim" type="button" onclick="objetosPorFila(3)">3</button>
    </div>
```

Tarjetas de cala

Las tarjetas de cala se basan en una plantilla de Bootstrap ([card](#)) y las generamos dinámicamente en Javascript leyendo los datos del JSON. Como la descripción es muy larga lo que hacemos es seleccionar solo 170 caracteres de la misma y al final añadimos “(...)” indicando que es más extensa. Para ir a la página de cada cala se puede pulsar en la imagen o en el botón de “Ver más”.

Las estrellas son de la librería Font Awesome y se llaman [fa-star](#). Tenemos una función que redondea la valoración de cada cala para tener un número entero y pintar las estrellas que tocan.

```
municipio = document.createElement("div");
municipio.setAttribute("class", "mg-top-sm-neg mg-bt-xsm fs-5-card-text");
municipio.innerHTML = "Municipio: " + obj[arr[num]]["geoposicionament1"]["city"];

descripcion = document.createElement("p");
descripcion.setAttribute("class", "card-text fs-5-card-text");
descripcion.innerHTML = obj[arr[num]]["descripcion"].substr(0, 170).replace("<br><br>", "") + " (...);

boton = document.createElement("a");
boton.setAttribute("href", "cala.html?" + arr[num]);
boton.setAttribute("class", "btn btn-primary btn-sm float-right");
boton.innerHTML = "Ver más";
```

```
//Funcion que dibuja las estrellas segun la valoracion de cada cala
function pintarEstrellas(n) {
    estrellas = document.createElement("div");
    estrellas.setAttribute("class", "estrellas mg-top-stars");

    stars = [];
    var j = 0;
    for (j; j < Math.round(n); j++) {
        stars[j] = document.createElement("span");
        stars[j].setAttribute("class", "mg-stars fs-5-card-star fa fa-star checked"); //tamaño estrellas en moviles en horizontal
    }
    for (j; j < 5; j++) {
        stars[j] = document.createElement("span");
        stars[j].setAttribute("class", "mg-stars fs-5-card-star fa fa-star");
    }
    for (i = 0; i < 5; i++) {
        estrellas.appendChild(stars[i]);
    }
    return estrellas;
}
```

Animaciones

Al aplicar filtros o cambiar el atributo de ordenación se recargan todas las calas y aparece una animación en las tarjetas. Estas animaciones son de una librería llamada [AOS](#) (Animate on Scroll) que añade una animación al objeto HTML que desees y se ejecuta al hacer scroll hacia abajo. Configuramos la animación para que solo se ejecute una vez (no queremos que si subes y vuelves a bajar se vuelva a ejecutar) y seleccionamos el tipo de animación, el tiempo y el delay. Estas animaciones también las usamos en el apartado del mapa y favoritos.

```
AOS.init({
    once: true,
    easing: "ease-in-out"
});
```

```
aos = document.createElement("div");
aos.setAttribute("data-aos", "fade-up");
aos.setAttribute("data-aos-delay", "0");
aos.setAttribute("data-aos-duration", "800");
```

Integración otros JSON

En este mismo fichero HTML vamos a cargar los JSON de otros compañeros. Hemos realizado la integración con un grupo que recopila actividades de Mallorca y otro grupo que realiza pueblos y lugares de interés. Estas integraciones se pueden visualizar mediante los enlaces de la barra de navegación “Actividades” y “Pueblos”. Al pulsar una de las dos opciones se carga el fichero de listado.html, al igual que con las calas, pero en el Javascript se comprueba a ver si se tienen que mostrar calas, actividades o pueblos. Es decir, se carga esta página dependiendo de lo que se ha pulsado. En el caso de las actividades se puede filtrar por tipo de actividad, y esto se realiza de la misma manera que con las calas. En el caso de los pueblos se puede filtrar por pueblo o lugar de interés, o simplemente visualizar todos los lugares de interés de un determinado pueblo. Además, en ambos casos se puede ordenar por nombre y valoración.

```
var url;
if (tema == "calas") url = "_json/datos.json";
else if (tema == "actividades") url = "https://sweetmallorca.netlify.app/json/actividades.json";
else if (tema == "pueblos") url = "https://beyls.netlify.app/json/pobles.json";
```

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET", url, true);
xmlhttp.send();
xmlhttp.onreadystatechange = function () {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        obj = JSON.parse(xmlhttp.responseText);
        if (tema == "calas") {
            //peticion de GET al servidor de comentarios
            var xmlhttp2 = new XMLHttpRequest();
            xmlhttp2.open("GET", "https://calasdemallorca.pythonanywhere.com/leerComentarios", true);
            xmlhttp2.send();
            xmlhttp2.onreadystatechange = function () {
                if (xmlhttp2.readyState == 4 && xmlhttp2.status == 200) {
                    objComentarios = JSON.parse(xmlhttp2.responseText);
                    escribirListadoCalas();
                }
            };
        }
        else if (tema == "actividades") escribirListadoActividades();
        else if (tema == "pueblos") escribirListadoPueblos();
    }
};
```

Al pulsar en una actividad o en un pueblo se visualiza de manera completa los datos de los JSON integrados. Esto está explicado en detalle en la página de producto.

Página mapa

API OpenLayers 3

Para mostrar el mapa con todas las calas de la web hemos usado la API de OpenLayers 3 que permite crear mapas dinámicos en Javascript pudiendo personalizarlos con gran variedad de opciones. El tipo de mapa está sacado de Mapbox ya que los que vienen por defecto en OpenLayers, que son los de OpenStreetMap no nos terminaban de gustar.

Primero, se recorre el JSON para obtener las coordenadas de todas las calas, el nombre, el municipio y una foto. Después se genera el mapa con todos los marcadores y se crea un popup con toda la información de cada cala (nombre, municipio, foto y botón de “Ver más”). Al pulsar en un marcador se genera un evento que visualiza el popup dependiendo del marcador que se ha pulsado. El código lo hemos sacado de los ejemplos de OpenLayers, de sus foros y de StackOverflow, sobretodo el tema de los popups ya que ha sido lo más complicado de realizar.

```
var iconFeatures = [] //array con todas las calas
for (i = 0; i < obj.length; i++) {
  var f = new ol.Feature({
    geometry: new ol.geom.Point(ol.proj.fromLonLat([obj[i]["geoposicionament1"]["long"], obj[i]["geoposicionament1"]["lat"]])),
    nombre: obj[i]["nom"],
    municipio: "<div class='mapa-popup-mun'>" + "Municipio: " + obj[i]["geoposicionament1"]["city"] + "</div>",
    imagen: "<img class='mapa-popup-img' src='" + obj[i]["imatges"][0] + "'>",
    boton: "<a href='cala.html?' + i + "' class='btn btn-primary btn-sm float-right fs-8 mapa-popup-bt'>Ver más</a>"
  });
  iconFeatures.push(f);
}
```

```
var popup = new ol.Overlay.Popup();
map.addOverlay(popup);

//Muestra el popup al pulsar en un marcador
map.on('singleclick', function (evt) {
  map.forEachFeatureAtPixel(evt.pixel, function (feature) {
    if (feature.get("nombre") != "-1") {
      popup.show(evt.coordinate, "<div class='mapa-popup'><h4>" + feature.get("nombre") + "</h4>" +
        feature.get("municipio") + feature.get("imagen") + feature.get("boton") + "</div>");
    }
  });
});

//Oculta el popup al pulsar en cualquier sitio del mapa que no sea un marcador
map.on('click', function (evt) {
  popup.hide();
});
```

API Geolocalización

Para posicionar la ubicación actual del usuario en el mapa usamos la API de Geolocalización de HTML. Obtenemos las coordenadas del usuario y el nivel de precisión de la muestra y llamamos a una función para actualizar el mapa con la posición del usuario y un círculo azul alrededor indicando la precisión. También hemos usado la función propia

de esta API llamada "watchPosition()" que sirve para definir una función que se ejecutará cuando la posición del usuario cambie. En nuestro caso volvemos a actualizar el mapa con la nueva ubicación del usuario. Si el usuario no acepta el permiso de ubicación o el navegador no soporta Geo simplemente no se muestra la ubicación del usuario.

```
//GEO
if (navigator.geolocation) {
  //navigator.geolocation.watchPosition(escribirMapa); //actualizar cuando el usuario se mueve
  var myLat;
  var myLon;
  navigator.geolocation.getCurrentPosition(function (position) {
    myLat = position.coords.latitude;
    myLon = position.coords.longitude;
    accuracy = position.coords.accuracy;

    //MAPA-CALAS CERCANAS
    getDistanciaCalas(myLat, myLon, accuracy);
    navigator.geolocation.watchPosition(watchGeolocation); //actualizar cuando el usuario se mueve
  });
  //no se ha dado permiso de geo
  cargarMapa(obj, [], 0.0, 0.0); //siempre cargamos el mapa (sin calas cercanas ni ubicacion actual)
  escribirCalasCercanas(false, [[0]]);
}
else { //el navegador no soporta geo
  cargarMapa([], 0.0, 0.0); //siempre cargamos el mapa (sin calas cercanas ni ubicacion actual)
  escribirCalasCercanas(false, [[-1]]);
}
```

Calas cercanas

Si se ha aceptado el permiso de Geo se mostrarán las 5 calas más cercanas a la ubicación actual del usuario con un marcador de color rojo en el mapa. Estas 5 calas también se mostrarán en un listado más abajo del mapa (ordenado por distancia), indicando información relevante de la cala y la distancia a la que está. Estas tarjetas son de Bootstrap y también usan las animaciones de la librería AOS explicada con anterioridad. Si la API de Geo no da la información de la ubicación se muestra un mensaje de error debajo del mapa y no se muestran las calas cercanas de color rojo.

```
//Funcion que escribe las 5 calas más cercanas o muestra una alerta si no hay geo
function escribirCalasCercanas(geo, calasDistancias) {
  var calas = calasDistancias[0];
  var dist = calasDistancias[1];
  if (!geo && calas[0] == 0) { //no se han dado permisos de geo
    if (document.getElementById("alerta-geo") == null) {
      var alerta = document.createElement("div");
      alerta.setAttribute("id", "alerta-geo");
      alerta.setAttribute("class", "alert alert-warning fade show fs-6 mx-auto mg-top-md");
      alerta.setAttribute("role", "alert");
      alerta.innerHTML = "Por favor, acepta los permisos de ubicación para poder ver las calas cercanas a tu ubicación actual.";
      document.getElementById("mapa-cercanas").appendChild(alerta);
    }
    return;
  }
  if (!geo && calas[0] == -1) { //el navegador no soporta geo
    if (document.getElementById("alerta-geo") == null) {
      var alerta = document.createElement("div");
      alerta.setAttribute("id", "alerta-geo");
      alerta.setAttribute("class", "alert alert-warning fade show fs-6 mx-auto mg-top-md");
      alerta.setAttribute("role", "alert");
      alerta.innerHTML = "Lo sentimos, tu navegador no soporta la geolocalización. Considera cambiar de dispositivo.";
      document.getElementById("mapa-cercanas").appendChild(alerta);
    }
    return;
  }
}
```

API Bing

Para poder calcular la distancia por carretera (si lo quisiéramos hacer en línea recta bastaría con aplicar fórmulas matemáticas con la latitud y la longitud) desde la ubicación actual hasta cada cala se usa la API de [Bing](#), la cual nos permite, con una sola llamada, obtener una matriz de distancias MxN desde M puntos de origen hacia N puntos destino. En nuestro caso solo tenemos 1 punto de origen (ubicación actual) y los destinos son todas las calas. La llamada a esta API nos devuelve un JSON con todas las distancias en kilómetros. Metemos todas las distancias en un array, lo ordenamos, y seleccionamos solo las 5 primeras calas. Una vez hecho esto repintamos el mapa con estas 5 calas en rojo y mostramos sus respectivas tarjetas.

```
var xmlhttp = new XMLHttpRequest();
var url = "https://dev.virtualearth.net/REST/v1/Routes/DistanceMatrix?origins=" + lat1 + "," + lon1 + "&destinations=";
for (l=0; l<obj.length; l++) {
    url += obj[l]["geoposicionament1"]["lat"] + "," + obj[l]["geoposicionament1"]["long"];
    if (l < (obj.length-1)) url += ";";
}
url += "&travelMode=driving&key=Aq4r7Sjg24ktC2N-8CfobSzE7NwXA_wD1aZXNKP6NIC12Iuj0b7ad3iYINUgpnSt";
```

```
var myArr = JSON.parse(xmlhttp.responseText);
calasDistancias = [];
for (l=0; l < myArr["resourceSets"][0]["resources"][0]["results"].length; l++) {
    calasDistancias[l] = myArr["resourceSets"][0]["resources"][0]["results"][l]["travelDistance"];
}
calas = getCalasCercanas(calasDistancias); //obtener array con las 5 calas mas cercanas
actualizarMapa(calas[0], lat1, lon1, acc); //actualizar el mapa
escribirCalasCercanas(true, calas); //escribir las tarjetas de las 5 calas cercanas
```

Página producto (cala)

Lectura JSON

Lo principal de esta página es visualizar todos los datos del JSON referentes a la cala seleccionada. Para saber qué cala visualizar hay que obtener el número que aparece al final de la URL. Al pulsar en una tarjeta de cala o en un botón de “Ver más” se redirige a esta página pero al final de la URL se añade el número de la cala para luego poder leerla desde aquí. Los números de calas hacen referencia a la posición del array que ocupan en el JSON. Una vez leído el JSON solo hay que rellenar los campos correspondientes (nombre, municipio, descripción, servicios...). Para rellenar los datos de cada cala usamos la librería [jQuery](#), ya que nos ha facilitado bastante la labor de modificar el contenido o los atributos de ciertos elementos del HTML.

```
function escribirCala(obj, id) {
    $('#title-cala').html("Calas de Mallorca - " + obj[id]["nom"]);
    //Introduccion
    $('#cala-titulo').html("<strong>" + obj[id]["nom"] + "</strong>");
    $('#cala-descripcion').html(obj[id]["descripcion"]);
    $('#cala-municipio').html("<strong>Municipio:</strong> &nbsp;" + obj[id]["geoposicionament1"]["city"]);
    $('#fav-1').addClass("press"); //comentar segun si esta en favoritos o no
    $('#cala-favorito-label').html("Eliminar de favoritos" + $('#cala-favorito-label').html()); //eliminar/añadir

    //Servicios
    $('#cala-tipoCala').html(obj[id]["dadesPropies"]["serveis"]["tipusCala"]);
    $('#cala-tipoAcceso').html(obj[id]["dadesPropies"]["serveis"]["tipusAcces"]);
}
```

Carousel imágenes

Las imágenes también se leen del JSON y se crea el Carousel dinámicamente desde el Javascript de manera que puede haber un número variable de imágenes y siempre se mostrarán todas en el Carousel. Hemos encontrado una plantilla por internet en la cual aparecen las miniaturas de todas las imágenes en el Carousel. La plantilla es [esta](#).

```
//Funcion que carga las imagenes en el carousel
function cargarImagenes() {
    var nFotos = obj[idObjeto]["imatges"].length;
    if (nFotos > 5) nFotos = 5;
    for (var i = 0; i < nFotos; i++) {
        //fotos
        var item = document.createElement("div");
        if (i == 0) item.setAttribute("class", "carousel-item active");
        else item.setAttribute("class", "carousel-item");
        var img = document.createElement("img");
        img.setAttribute("class", "d-block car-img");
        img.setAttribute("src", obj[idObjeto]["imatges"][i]);
        img.setAttribute("alt", "...");
        item.appendChild(img);
        document.getElementById("carousel-fotos").appendChild(item);
    }
}
```

Comentarios

Nuestro hosting no permite ejecutar código servidor por lo que nos hemos montado un back-end en un servidor externo que corre sobre Python para que dé soporte a los comentarios. El servidor en cuestión es [PythonAnywhere](#) y permite crear una pequeña webapp que gestiona peticiones de POST y GET. Además, tenemos un fichero JSON de comentarios guardado en ese servidor que se actualiza según se ejecutan las peticiones POST (publicar un comentario en el servidor) y GET (obtener comentarios del servidor). De esta manera se pueden publicar comentarios desde cualquier lugar y navegador y son visibles para todo el mundo.

Al cargar la página de una cala se realiza una petición GET al servidor de comentarios para visualizar los comentarios que tiene esa cala (si hay) y al escribir un comentario y publicarlo se ejecuta una petición POST con el comentario en cuestión para que se guarde en el JSON de comentarios. Después de publicar un comentario se realiza otra petición GET para que se actualice el listado de comentarios con el comentario que se acaba de publicar.

```
$.post("https://calasdemallorca.pythonanywhere.com/enviarComentarios", {
  cala: idObjeto,
  nombreCala: obj[idObjeto]["nom"],
  nombre: nombre,
  texto: texto,
  valoracion: parseFloat(valoracion),
  fecha: fecha
});
```

```
//petición de GET al servidor de comentarios
var xmlhttp = new XMLHttpRequest();
var url = "https://calasdemallorca.pythonanywhere.com/leerComentarios";
xmlhttp.open("GET", url, true);
xmlhttp.send();
xmlhttp.onreadystatechange = function () {
  if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
    var myArr = JSON.parse(xmlhttp.responseText);
    escribirComentarios(myArr["calas"][idObjeto]["comentarios"], myArr["calas"][idObjeto]["valoracionGlobal"]);
  }
};
```

En el JSON de comentarios externo tenemos un apartado llamado *index* en el que se van añadiendo los comentarios por orden inverso (el comentario más reciente es el primero). Esto nos sirve para mostrar los últimos comentarios en la página principal de nuestra web. Además, también tenemos otro apartado llamado *calas* en el que tenemos un array con todas las calas de nuestra web y sus respectivos comentarios. Cuando se añade un comentario, este se añade tanto al apartado de *index* como al de *calas*, para que así la página principal y la de cada cala queden estén actualizadas. También se actualiza la valoración global de la cala haciendo la media de los comentarios.

```

"index": [
  {
    "idCala": 7,
    "nombreCala": "Cala Agulla",
    "nombre": "Antoni",
    "texto": "Bona feina",
    "valoracion": 3.0,
    "fecha": 1621952922325
  },
  {
    "idCala": 28,
    "nombreCala": "Cala Estellencs",
    "nombre": "Marc",
    "texto": "Me ha encantado",
    "valoracion": 4.0,
    "fecha": 1621950136952
  }
],

"calas": [
  {
    "nombre": "Cala Tuent",
    "valoracionGlobal": 3.2,
    "comentarios": []
  },
  {
    "nombre": "Camp de Mar",
    "valoracionGlobal": 2.3,
    "comentarios": []
  },
  {
    "nombre": "Cala Varques",
    "valoracionGlobal": 4.8,
    "comentarios": [
      {
        "nombre": "Josep",
        "texto": "La caminata no es muy larga y vale totalmente la pena!!",
        "valoracion": 5.0,
        "fecha": 1620938884276
      }
    ]
  }
],

```

```

@app.route("/enviarComentarios", methods=['POST'])
def send():
    #jsdata = request.form['texto']
    #with open('data.json', 'w') as outfile:
    #    #json.dump(request.form['texto'], outfile)

    nComentario = {"nombre": request.form["nombre"], "texto": request.form["texto"], "valoracion": float(request.form["valoracion"])
    nComentarioIndex = {"idCala": int(request.form["cala"]), "nombreCala": request.form["nombreCala"], "nombre": request.form["nombre"], "idCala": int(request.form["cala"])}
    with open("comentarios.json", encoding='utf-8') as file:
        data = json.load(file)
        #print(data)

    data["calas"][cala]["comentarios"].insert(0,nComentario)
    data["index"].insert(0,nComentarioIndex)

    val_global = data["calas"][cala]["valoracionGlobal"]
    lst = [val_global]
    for i in range(len(data["calas"][cala]["comentarios"])):
        lst.append(data["calas"][cala]["comentarios"][i]["valoracion"])

    data["calas"][cala]["valoracionGlobal"] = round((sum(lst) / len(lst)), 1)

    with open("comentarios.json", "w", encoding='utf-8') as file:
        json.dump(data, file, indent=4, ensure_ascii=False)
    return "."

```

```

@app.route("/leerComentarios", methods=['GET'])
def receive():
    with open("comentarios.json", encoding='utf-8') as file:
        #data = json.load(file)
        data = jsonify(json.load(file))
    return data

```

Mapa

El mapa está realizado de la misma manera ya explicada anteriormente, es decir, con la API de OpenLayers 3, pero el proceso es mucho más sencillo ya que solo hay que añadir un marcador en las coordenadas de la cala. En este caso sí que hemos usado la estética del mapa de OpenStreetMap, para cambiar un poco. Hay que configurar el zoom, las coordenadas para centrar el mapa y las coordenadas del marcador.

```
var iconFeature = new ol.Feature({
  geometry: new ol.geom.Point(ol.proj.fromLonLat([lon, lat])),
  name: nombre,
});
```

```
var map = new ol.Map({
  layers: [
    new ol.layer.Tile({source: new ol.source.OSM()}),
    vectorLayer
  ],
  view: new ol.View({
    center: ol.proj.fromLonLat([lon, lat]),
    zoom: 11
  }),
  controls: ol.control.defaults({ attribution: false }),
  target: 'cala-mapa'
});
```

API Meteorología

Para mostrar la meteorología de cada cala hemos usado la API de [OpenWeatherMap](#). Esta API permite hacer llamadas indicando las coordenadas donde se quiere obtener el tiempo, el idioma, y si se quiere saber el tiempo por días, horas o minutos.

En nuestro caso indicamos las coordenadas de la cala, el idioma español y el tiempo en días. La API nos devuelve un fichero JSON con toda la información y nosotros simplemente mostramos la información en nuestra web. Hemos convertido todas las unidades en unidades españolas (Celsius, kmh, 24 horas) para facilitar el entendimiento. El icono del tiempo día a día nos lo proporciona la misma API pero los demás iconos los hemos descargado nosotros de internet (viento, nubes, humedad, UV...). Además, también debemos obtener la fecha de los próximos días en un formato entendible ya que la API nos lo da en segundos.

```
//Proximos 5 dias
for (var i=0; i<5; i++) {
    if (i == 0) {
        $('#dia0-dia').html("Hoy");
    }
    else if (i == 1) {
        $('#dia1-dia').html("Mañana");
    }
    else {
        var fecha = new Date(json.daily[i].dt * 1000);
        $('#dia{i}-dia').html(fecha.getDate() + " de " + getMes(fecha.getMonth()));
    }
    $('#dia{i}-icono').attr("src", "http://openweathermap.org/img/wn/" + json.daily[i].weather[0].icon + "@2x.png");
    var max_t = Math.round(json.daily[i].temp.max - 273.15);
    var min_t = Math.round(json.daily[i].temp.min - 273.15);
    $('#dia{i}-temperatura').html(min_t + " - " + max_t + " °C");
    $('#dia{i}-viento').html((Math.round(json.daily[i].wind_speed * 3.6)) + " kmh");
}
```

```
//Funcion que devuelve el mes actual segun el numero
function getMes(n) {
    switch (n) {
        case 0: return "Enero";
        case 1: return "Febrero";
        case 2: return "Marzo";
        case 3: return "Abril";
        case 4: return "Mayo";
        case 5: return "Junio";
        case 6: return "Julio";
        case 7: return "Agosto";
        case 8: return "Septiembre";
        case 9: return "Octubre";
        case 10: return "Noviembre";
        case 11: return "Diciembre";
    }
}
```

Integración otros JSON

En la página de listado se visualizan todas las tarjetas correspondientes a los datos de los JSON, ya sean integrados o propios. Al pulsar en una tarjeta de un JSON integrado se redirige a esta página de producto, que se rellena como si se tratase de una cala pero con los datos de los JSON integrados. Hemos tenido que realizar pequeños cambios porque hay grupos que usan campos de datos que nosotros no usamos (precio, horarios...). De esta manera, al cargarse esta página de producto se llama a una función u otra para cargar los datos en la página dependiendo de si se trata de una cala, actividad, pueblo o lugar de interés. Todo esto se realiza añadiendo parámetros a la URL al llamar a la página base de cala.html. Después, se revisa la URL y se decide si se muestra una cala, actividad o pueblo.

```
var url;
if (tema == "calas") url = "_json/datos.json";
else if (tema == "actividades") url = "https://sweetmallorca.netlify.app/json/actividades.json";
else if (tema == "pueblos") url = "https://beyls.netlify.app/json/pobles.json";

if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
    obj = JSON.parse(xmlhttp.responseText);

    if (tema == "calas") escribirCala();
    else if (tema == "actividades") escribirActividad();
    else if (tema == "pueblos") escribirPueblo();
}
```

En el caso del JSON de actividades se ha tenido que añadir una etiqueta para el precio, y el apartado de servicios que teníamos en cada cala lo hemos transformado en un campo de información extra, en el que se indican redes sociales, datos importantes de la actividad...

En el caso del JSON de pueblos hemos creado dos páginas de producto distintas (una para pueblos y otra para lugares de interés) ya que los pueblos tienen descripción e historia, pero los lugares de interés también tienen precio, información extra (teléfono, página web, facebook) y horario de apertura.

```
//ACTIVIDADES
function escribirActividad() {
    document.getElementById("bg-cala").setAttribute("class", "bg-cala-verde");
    $("#pueblo-label").remove(); //borrar pueblo mas cercano
    $("#pueblo").remove(); //borrar pueblo mas cercano
    $('#title-cala').html("Calas de Mallorca - " + obj[idObjeto]["nom"]);
    //Introduccion
    $('#cala-titulo').html("<strong>" + obj[idObjeto]["nom"] + "</strong>");
    $('#cala-descripcion').html(obj[idObjeto]["descripcion"]);
    $('#cala-municipio').html("<strong>Municipio:</strong> &nbsp;" + obj[idObjeto]["geoposicionament1"]["city"]);
    var precio = parseFloat(obj[idObjeto]["preu"]["import"].replace("€", ""));
    if (precio == 0) $('#cala-favorito-label').html("<strong>Precio:</strong> &nbsp;" + "Gratis");
    else $('#cala-favorito-label').html("<strong>Precio:</strong> &nbsp;" + obj[idObjeto]["preu"]["import"]);
    $('#cala-favorito-label').removeClass("like-div");
    //escribirFavoritos(idObjeto);

    //Informacion extra
    $('#cala-servicios').html("Información extra");
    $('#cala-tipoCala-1').html("<strong> Duración: </strong>");
    $('#cala-tipoCala').html(obj[idObjeto]["dadesPropies"]["duracio"]);
}
```

```
//PUEBLOS
function escribirPueblo() {
    document.getElementById("bg-cala").setAttribute("class", "bg-cala-verde");
    $("#pueblo-label").remove(); //borrar pueblo mas cercano
    $("#pueblo").remove(); //borrar pueblo mas cercano
    $('#title-cala').html("Calas de Mallorca - " + obj[idObjeto]["nom"]);
    //Introduccion
    $('#cala-titulo').html("<strong>" + obj[idObjeto]["nom"] + "</strong>");
    if (subtema == "pueblo") { //pueblo
        //Web Semantica
        var info = generarJsonLDPueblo(obj[idObjeto]);
        cargarJsonLD(info);

        $('#cala-municipio').html("<strong>Tipo:</strong> &nbsp;Pueblo");
        $('#cala-favorito-label').html("");
        $('#cala-favorito-label').removeClass("like-div");
        $('#cala-descripcion').html(obj[idObjeto]["dadesPropies"]["introduccio"]);

        for (i = 1; i < 6; i++) $('#servicios-fila-${i}`).remove();
        $('#cala-servicios').html("Historia");
        $('#cala-historia').html(obj[idObjeto]["dadesPropies"]["historia"]);
    }
    else { //lugar de interes
    }
}
```

Integración dinámica Pueblos

También hemos diseñado una integración inteligente con el grupo de los pueblos. Hemos pensado que sería interesante ver información sobre el pueblo en el que se encuentra una cala de nuestro JSON. Por ello, al visualizar una cala se revisa en qué pueblo se encuentra y si ese pueblo está definido en el JSON integrado de pueblos, se añade justo debajo del apartado de meteo una tarjeta con el nombre, una foto, la distancia a la que se encuentra y una pequeña descripción del pueblo. Si se pulsa sobre ese pueblo se redirige a la página de producto del pueblo en cuestión.

Para hacer esto simplemente miramos el pueblo de la cala y recorremos todo el JSON de pueblos en busca de algún pueblo con el mismo nombre. Si no hay ningún pueblo que coincida, no se añade ninguna tarjeta a la página.

Para calcular la distancia usamos una función encontrada en [StackOverflow](https://stackoverflow.com/questions/3642048/algorithm-for-efficiently-finding-nearest-city) que dadas dos coordenadas devuelve la distancia entre ellas. Usa matemática básica pero al ya encontrarla hecha en Internet la hemos usado en nuestra aplicación web.

```
//Pueblo cercano
var xmlhttp2 = new XMLHttpRequest();
xmlhttp2.open("GET", "https://beyls.netlify.app/json/pobles.json", true);
xmlhttp2.send();
xmlhttp2.onreadystatechange = function () {
    if (xmlhttp2.readyState == 4 && xmlhttp2.status == 200) {
        pueblos = JSON.parse(xmlhttp2.responseText);
        escribirPuebloCercano(pueblos);
    }
};
```

```
//Funcion que escribe el pueblo más cercano (si hay) cuando se muestra una cala
function escribirPuebloCercano(pueblos) {
    var p = []
    for (i = 0; i < pueblos.length; i++) {
        if (obj[idObjeto]["geoposicionament1"]["city"] == pueblos[i]["geoposicionament1"]["city"] && pueblos[i]["tipus"] == "poble") {
            p.push(i);
        }
    }
    if (p.length != 0) {
```

```
function getDistanceFromLatLonInKm(lat1, lon1, lat2, lon2) {
    var R = 6371; // Radius of the earth in km
    var dLat = deg2rad(lat2-lat1); // deg2rad below
    var dLon = deg2rad(lon2-lon1);
    var 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)
        ;
    var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    var d = R * c; // Distance in km
    return d;
}

function deg2rad(deg) {
    return deg * (Math.PI/180)
}
```

Página favoritos

API WebStorage

La sección de favoritos se basa en la API de WebStorage de HTML. Las calas guardadas en favoritos se almacenan solamente en el navegador del usuario, hasta que éste elimine los datos del navegador o simplemente use navegación privada. Para ello guardamos en un JSON un array con todos los identificadores de las calas que se han añadido a favoritos. Cada vez que se añade o elimina una cala se modifica este JSON. En el caso de no existir este JSON lo generamos de 0 e introducimos el identificador de la primera cala que se va a añadir a Favoritos.

A la hora de mostrar todas las calas se accede a este JSON mediante la API de WebStorage y se generan dinámicamente todas las tarjetas referentes a los identificadores que contiene este JSON. En el caso de no haber ninguna cala en favoritos se muestra una alerta de Bootstrap. También se actualiza la etiqueta que indica cuantas calas hay en favoritos (X resultados).

```
//Funcion que añade o elimina de favoritos. numCala: numero de cala, añadir: true es añadir y false es eliminar
function editarFavoritos(numCala, añadir) {
    var favoritos;
    var favJSON = localStorage.getItem("favoritos");
    if (favJSON === null) favoritos = []; //crear array de favoritos
    else favoritos = JSON.parse(favJSON); //parsear el array del JSON

    if (añadir) {
        favoritos.push(numCala);
    }
    else {
        var i = favoritos.indexOf(numCala);
        favoritos.splice(i, 1);
    }

    localStorage.setItem("favoritos", JSON.stringify(favoritos));
}
```

```
function eliminarFavorito(id) {
    $('#fav-$(id)').toggleClass("press", 1000);

    idCala = parseInt(id);
    var favJSON = localStorage.getItem("favoritos");
    var favoritos = JSON.parse(favJSON);
    var i = favoritos.indexOf(idCala);
    favoritos.splice(i, 1);
    localStorage.setItem("favoritos", JSON.stringify(favoritos));

    //Eliminar tarjeta
    var del = document.getElementById("fav-card-" + id);
    $('#fav-card2-$(id)').fadeOut(400, function(){
        del.remove();

        //Actualizar numero de resultados
        if (favoritos.length == 1) $('#num-resultados').html("<strong>" + favoritos.length + " Resultado</strong>");
        else $('#num-resultados').html("<strong>" + favoritos.length + " Resultados</strong>");

        //Mostrar aviso si no hay tarjetas
        if (favoritos.length == 0) avisoSinFavoritos();
    });
}
```

Web Semántica

Nuestra página está realizada al completo con Web Semántica y para ello hemos elegido la opción de JSON-LD ya que nos ha parecido más conveniente porque se puede hacer todo en Javascript y no hay necesidad de modificar el código HTML. Se han seguido las reglas de marcado de datos vistas en clase y definidas en schema.org.

La página principal está definida como "WebApp" y en ella hemos definido las imágenes del Carousel, los autores de la web (nosotros) y sus respectivas URLs, y los comentarios recientes como "Review". También se define el video y está debidamente etiquetado con la etiqueta "VideoObject".

En el caso de las páginas de Listado, Mapa y Favoritos, todas las tarjetas de cala que aparecen están definidas como "Beach" y en ella se incluye la foto, una pequeña descripción, las coordenadas, el nombre y la valoración. Con la valoración hemos tenido problemas ya que Google decidió dejar de dar soporte para las valoraciones en muchos tipos de datos, entre ellos las playas. Aún así hemos decidido dejar las valoraciones ya que así se indica en schema.org y si que funcionan en otros buscadores que no sean Google.

En la página de cada cala se define todo como un único bloque "Beach" y en él se añade el nombre, descripción, todas las imágenes, el mapa, las coordenadas y los comentarios que tiene la cala. Todo esto se realiza de manera dinámica ya que no sabemos a priori cuantas imágenes tiene una cala o cuantos comentarios tiene.


Las integraciones con otros JSON en nuestra webapp también disponen de web semántica siguiendo el mismo modelo explicado anteriormente y haciendo pequeños cambios para que todo cobre más sentido. En el caso de las actividades, son definidas como "Product" y los pueblos como "Place". Se rellenan los datos con JSON-LD de manera dinámica del mismo modo que las calas.

A continuación se muestran unas cuantas capturas que reflejan cómo se va creando el código de JSON-LD de manera dinámica según se va visualizando cada apartado de nuestra webapp. También se adjunta alguna captura de la herramienta de prueba de datos estructurados de Google ([link](#)).

```
//WEB SEMANTICA
function generarJsonLDIndex() {
  let info = {
    "@context": "https://schema.org",
    "@type": "WebApplication",
    "applicationCategory": "Calas de Mallorca",
    "applicationSubCategory": "Playas, Calas",
    "about": "Una webapp para descubrir los rincones más bonitos de la costa mallorquina",
    "name": "Calas de Mallorca",
    "description": "Una pequeña recopilación de las mejores Calas de Mallorca hecha por un grupo de",
    "author": [
      {
        "@type": "Person",
        "givenName": "Carlos",
        "familyName": "Veny",
        "gender": "Male",
        "image": "imagenes/nosotros_carlos.jpg",
        "url": "https://www.instagram.com/carlos_veny/"
      },
      {
        "@type": "Person",
        "givenName": "Juanjo",
        "familyName": "Nieto",
        "gender": "Male",
        "image": "imagenes/nosotros_juanjo.jpg",
        "url": "https://www.instagram.com/juanjo_nieto9/"
      }
    ]
  }
}
```

```
//WEB SEMANTICA
function generarJsonLDCala(cala, numero) {
  likes = cala["likes"]
  if (likes == 0) likes = 1;
  var url = "calasdemallorca.netlify.app/cala.html?calas&" + numero;
  let info = {
    "@context": "http://www.schema.org",
    "@type": "Beach",
    "name": cala["nom"],
    "geo": {
      "@type": "GeoCoordinates",
      "latitude": cala["geoposicionament1"]["lat"],
      "longitude": cala["geoposicionament1"]["long"]
    },
    "aggregateRating": {
      "@type": "AggregateRating",
      "itemReviewed": "Thing",
      "bestRating": "5",
      "worstRating": "0",
      "ratingValue": objComentarios["calas"][numero]["valoracionGlobal"],
      "reviewCount": likes
    },
    "description": cala["descripcio"].substr(0, 170).replace("<br><br>", "") + " (...)",
    "photo": cala["imatges"][0],
    "url": url
  }
  return info;
}
```

```
function generarJsonLDActividad(act) {
  likes = act["likes"]
  if (likes == 0) likes = 1;
  var precio = act["preu"]["import"].replace("€", "");
  let info = {
    "@context": "http://www.schema.org",
    "@type": "Product",
    "name": act["nom"],
    "geo": {
      "@type": "GeoCoordinates",
      "latitude": act["geoposicionament1"]["lat"],
      "longitude": act["geoposicionament1"]["long"]
    },
    "aggregateRating": {
      "@type": "AggregateRating",
      "itemReviewed": "Thing",
      "bestRating": "5",
      "worstRating": "0",
      "ratingValue": act["puntuacio"],
      "reviewCount": likes
    },
    "description": act["descripcio"],
    "image": [],
    "hasMap": {
      "@type": "Map",
      "mapType": { "@id": "https://schema.org/VenueMap" },
    },
    "offers": {
      "@type": "Offer",
      "price": precio,
      "priceCurrency": "EUR"
    }
  }
}
```

Beach		1 ERROR 0 ADVERTENCIAS ^
@type	Beach	
name	Cala Agulla	
description	Cala Agulla es una playa de arena y rocas situada en el municipio de Capdepera y muy cerca del núcleo urbano de Cala Ratjada. Su arena es blanca y fina, alternand (...)	
url	https://calasdemallorca.netlify.app/cala.html?calas&7	
geo		
@type	GeoCoordinates	
latitude	39.721505051968222	
longitude	3.4528749803644359	
aggregateRating		
@type	AggregateRating	
bestRating	5	
worstRating	0	
ratingValue	0.8	
reviewCount	1	
itemReviewed		
@type	Thing	
name	Thing	
photo		
	Beach no es un tipo de segmentación válido conocido para la propiedad itemReviewed.	
@type	CreativeWork	
name	https://calasdemallorca.netlify.app/imagenes/calas/cala_agulla_1.jpg	

El error que sale es debido a lo que se ha explicado antes de que Google ha dejado de permitir que se puedan añadir valoraciones a ciertos tipos datos (entre ellos playas).














Estructura de directorios

Disponemos de dos directorios principales ya que tenemos nuestra webapp dividida en 2 servidores distintos. Esto se debe a la problemática explicada anteriormente con la imposibilidad de ejecutar código servidor en nuestro hosting.

En nuestro servidor principal ([Netlify](#)) tenemos prácticamente la totalidad de la página web. En el directorio raíz tenemos los HTML y una serie de carpetas organizando el resto de los ficheros (Javascript, CSS, JSON, media y bootstrap). De esta manera nos es mucho más fácil encontrar el fichero que estamos buscando.

En nuestro servidor de comentarios ([PythonAnywhere](#)) simplemente almacenamos nuestro fichero JSON con todos los comentarios, tal como se ha explicado anteriormente.

Para actualizar nuestra webapp simplemente subimos nuestro directorio principal a nuestro hosting y al cabo de unos segundos se actualiza la web de manera pública.

 _css	20/05/2021 8:33	Carpeta de archivos	
 _js	20/05/2021 8:33	Carpeta de archivos	
 _json	20/05/2021 8:33	Carpeta de archivos	
 audio	20/05/2021 8:33	Carpeta de archivos	
 bootstrap5	20/05/2021 8:33	Carpeta de archivos	
 imagenes	20/05/2021 8:33	Carpeta de archivos	
 videos	20/05/2021 8:33	Carpeta de archivos	
 _headers	06/05/2021 17:50	Archivo	1 KB
 cala.html	24/05/2021 11:51	Firefox HTML Doc...	27 KB
 favoritos.html	24/05/2021 11:51	Firefox HTML Doc...	9 KB
 index.html	24/05/2021 14:05	Firefox HTML Doc...	20 KB
 listado.html	24/05/2021 11:51	Firefox HTML Doc...	9 KB
 mapa.html	24/05/2021 11:51	Firefox HTML Doc...	8 KB

JSON

Para realizar el fichero JSON hemos seguido las directrices de normativa de etiquetas que se presentó en clase, rellenando varios ejemplos de calas para tener una primera aproximación inicial al JSON definitivo.

Además de usar las etiquetas vistas en clase hemos añadido datos propios en la correspondiente sección ya que, en nuestro caso, guardamos qué servicios tiene cada cala (duchas, lavabos, socorrista...).

El fichero JSON completo con todas las calas se puede consultar en el siguiente link: https://calasdemallorca.netlify.app/_json/datos.json

```
"dadesPropies": {  
  "serveis": {  
    "tipusCala": "Arena/Rocas",  
    "tipusAcces": "Curvas pronunciadas",  
    "accesMinusvalids": "No",  
    "nudista": "No",  
    "hamacas": "No",  
    "sombrellas": "No",  
    "parking": "No",  
    "animals": "Sí",  
    "dutxes": "No",  
    "lavabos": "No",  
    "alquilerEmbarcacions": "No",  
    "socorrista": "No",  
    "boies": "Sí",  
    "pier": "No",  
    "bar": "Sí"  
  }  
}
```

Metadatos

Para completar un poco más nuestra webapp hemos añadido metadatos en las cabeceras de nuestros documentos HTML. Los datos más básicos que hemos añadido son la codificación (UTF-8), los autores de la web (nosotros mismos), una pequeña descripción y el título de cada página (eso se va actualizando dinámicamente según el contenido que se muestra en la web). Además, también disponemos de un pequeño icono que se muestra en la pestaña del navegador.

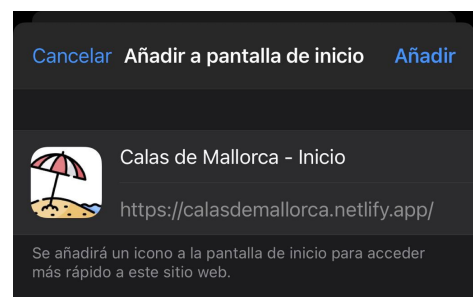
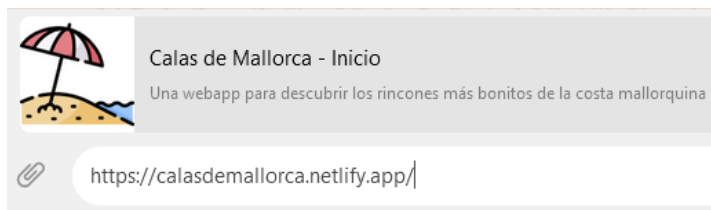
Para facilitar la compartición de nuestra web en redes sociales también hemos añadido metadatos en formato "Open Graph" que sirven para controlar cómo se muestran las URLs cuando se comparten en redes sociales. De esta manera al compartir nuestra webapp, por ejemplo en Whatsapp" sale el logo, el título y la descripción de nuestra webapp.

También hemos añadido soporte para dispositivos Apple, de manera que si se añade nuestra webapp a la pantalla de inicio de un dispositivo iOS o MacOS también se muestra el logo, en lugar de una imagen generada automáticamente. A continuación se adjuntan capturas de todo lo explicado en este apartado.

```
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta name="description" content="Una webapp para descubrir los rincones más bonitos de la costa mallorquina">
<meta name="author" content="Carlos Veny, Juanjo Nieto, Joan Alcover">
<meta name="generator" content="">
<title>Calas de Mallorca - Inicio</title>
```

```
<meta property="og:site_name" content="Calas de Mallorca">
<meta property="og:title" content="Calas de Mallorca - Inicio" />
<meta property="og:description" content="Una webapp para descubrir los rincones más bonitos de la costa mallorquina" />
<meta property="og:image" itemprop="image" content="imagenes/playa-apple.png">
<meta property="og:type" content="website" />
```

```
<link rel="icon" href="imagenes/playa-apple.png" type="image/icon type">
<link rel="apple-touch-icon" href="imagenes/apple-touch-icon.png">
```



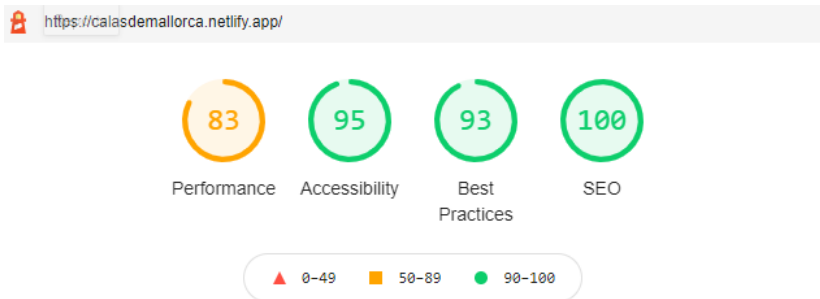
Evaluación en dispositivos

Para probar el funcionamiento de nuestra aplicación web la hemos ido testeando entre los miembros del grupo en busca de fallos o funcionalidades que no tienen un funcionamiento correcto. Al encontrar un error tratábamos de arreglarlo y volvíamos a publicar la web.

Todas estas pruebas las hemos ido realizando en diferentes dispositivos (móviles, tablets, ordenadores, ordenadores con monitor ultrawide, Smart TV...), en varios sistemas operativos (iOS, MacOS, Android, Windows...) y en los navegadores principales (Safari, Firefox, Chrome, Edge y Opera). Hemos podido comprobar que la página es completamente web-responsive y funcional en todos estos dispositivos mencionados anteriormente, siempre y cuando sean compatibles con HTML5.

Lighthouse

Para completar más la labor de evaluar nuestra web hemos usado la herramienta Lighthouse que viene incluida en Google Chrome. Esta herramienta sirve para medir la calidad de las páginas web. Hemos ido realizando diversos tests para así mejorar los aspectos que nos indicaba (web responsive, más opciones de accesibilidad...). La puntuación que hemos obtenido es la siguiente:



Herramientas

Hemos usado muchas herramientas durante el desarrollo de nuestra aplicación web, ya sea de comunicación, programación, APIs o librerías. A continuación detallamos un listado con las mismas:

- **Discord:** Herramienta fundamental para la comunicación entre nosotros (voz y compartición de pantalla).
- **Whatsapp:** Usado principalmente para comunicarnos, repartir las tareas y avisar de errores o funcionalidades nuevas.
- **Google Drive:** Lo hemos usado para subir archivos actualizados de nuestra web e ir realizando la documentación técnica de forma simultánea.
- **Netlify:** Hosting para almacenar toda nuestra aplicación web. Inicialmente usábamos Neocities pero Netlify nos ofrecía mejores servicios así que cambiamos.
- **PythonAnywhere:** Hosting para almacenar nuestro JSON de comentarios y ejecutar código en Python cada vez que se realiza una petición externa para poder actualizar este JSON.
- **Bootstrap:** Prácticamente la base de nuestra webapp. Elementos como la barra de navegación, el carousel, el footer y las tarjetas de cada producto son de Bootstrap. También hemos usado el sistema de Grid para posicionar elementos en la posición deseada.
- **jQuery:** Librería para simplificar las interacciones con los ficheros HTML. Usado a la hora de mostrar los datos JSON en la aplicación web.
- **AOS:** Librería para realizar las animaciones en las páginas de Listado, Mapa, Favoritos y en cada cala. La librería genera una animación en el *div* correspondiente cada vez que se hace scroll y queda visible.
- **Font Awesome:** Recopilación de diversos iconos para poder usar en HTML. Los hemos usado para valoraciones e iconos de redes sociales.
- **Visual Studio Code:** Herramienta principal que hemos usado para programar tanto los HTML, como los CSS, Javascript y JSON.
- **Plugin Live Server:** Plugin de Visual Studio Code que simula que no estamos trabajando en local. Lo hemos usado para probar las peticiones a servidores externos antes de subir la webapp a nuestro hosting.
- **Herramienta prueba de datos estructurados:** Usado para comprobar la correctitud de la web semántica.
- **Lighthouse:** Lo hemos usado para medir la calidad de la página web y así poder mejorar ciertos aspectos de ella.

-
- **StackOverflow:** Nos ha sido de gran ayuda a la hora de solucionar errores, buscar información sobre ciertas funciones o código que no entendíamos, encontrar código útil para nuestras implementaciones...
 - **Codepen:** Página web que recopila fragmentos de código que usan HTML, CSS y Javascript. Hemos usado alguna plantilla, como la del carousel de comentarios, o el botón de incluir a favoritos.
 - **W3School:** Hemos usado esta web para entender el funcionamiento básico de ciertas partes de código en HTML o en Javascript. Tiene tutoriales de todos los elementos básicos de estos lenguajes.

Además, también hay que añadir a esta lista todas las APIs que hemos ido usando y que ya se han explicado durante el desarrollo de esta documentación técnica.

Opinión

Una vez terminada nuestra webapp podemos afirmar que nos ha gustado mucho realizar esta asignatura ya que nos puede servir mucho de cara al futuro y a trabajar en una empresa. Nos hemos tomado esta práctica como un entrenamiento de cara a nuestra vida laboral ya que hemos descubierto que nos gusta bastante este mundo del desarrollo web.

Como aspectos a mejorar hay que comentar el aspecto de Progressive Web App ya que no nos ha dado tiempo a implementarlo para la entrega final, aunque nos habría gustado investigar un poco sobre el tema y así aprender cómo funciona. También queríamos implementar una API de traducción para poder traducir los datos de algunos JSON de compañeros que estaban en catalán, pero tampoco hemos tenido suficiente tiempo para hacerlo. No obstante, pensamos que hemos cubierto todos los demás requisitos que se pedían para la entrega de esta práctica y consideramos que hemos realizado un gran trabajo y que nos hemos esforzado al máximo para conseguir sacar adelante la webapp.

Por todo lo dicho anteriormente hemos decidido valorar nuestra práctica con un 9.