

DOCUMENTACIÓN

Café con Cultura

Práctica Tecnología multimedia

2022-2023



Universitat
de les Illes Balears

Juan Francisco Hernández Fernández

Marco Martínez González

Jordi Sevilla Marí

Índice

1. Prólogo	3
1.1 Historial de Versiones	3
1.2 Propósito del Documento	3
1.3 Estructura del Documento	3
1.4 Integrantes del Proyecto	3
2. Café con Cultura	4
2.1. Descripción	4
2. 2. Funcionalidades de la Web-App	4
3. Proceso de maquetación y diseño gráfico.	6
3.1. Hosting y Dominio	6
3.2. Diseño gráfico.	6
3.3. Maquetación	6
3.4. Formato de Datos	7
3.5. API's.	7
3.6. Librerías.	8
3.7. Inclusión de MEDIA	8
4. Desarrollo de funciones con JavaScript.	8
4.1. main.js	8
4.2. navegacion.js	9
4.3. utilidades.js	11
4.4. maps-api.js	13
5. Distribución de carpetas y ficheros	15
6. Semántica	15
7. Rendimiento	16
8. Web responsive	17
9. Comentarios adicionales	18
10. Autovaloración.	19

1. Prólogo

1.1 Historial de Versiones

- **Versión 1.0 (17/04/2023):** Versión Inicial del documento.
- **Versión 2.0 (15/05/2023):** Actualizadas secciones 2.2-Funcionalidades de la WebApp, 3.3-Maquetación y 3.5-API's. Añadidas secciones 3.7-Inclusión de MEDIA , 4-Desarrollo de funciones con JavaScript, 5-Distribución de carpetas y ficheros; y 6-Comentarios adicionales
- **Versión 3.0 (24/05/2023):** Versión final del documento. Añadidas las secciones, 6-Semántica,7-Rendimiento, 8-Comentarios adicionales, 9-Autovaloración.

1.2 Propósito del Documento

En este documento facilitaremos los conocimientos técnicos necesarios para comprender el código y la estructura de nuestro proyecto de página web (Café con cultura) a todo aquel que tenga como objetivo mantener o evolucionar su código.

1.3 Estructura del Documento

- **Prólogo:** Breve explicación y contextualización del proyecto realizado.
- **Café Con Cultura:** Este apartado recoge los objetivos que se quieren obtener de la práctica y el conjunto de funcionalidades implementadas en nuestro proyecto.
- **Proceso de maquetación y diseño gráfico:** En este apartado se presentan cuestiones relacionadas con, el hosting y dominio, el diseño gráfico, la maquetación, el formato de los datos, las API's escogidas, las librerías utilizadas y la inclusión de Media (SVG's y videos).
- **Desarrollo de funciones con JavaScript:** En este último apartado del documento se realiza una explicación del conjunto de funciones implementadas en cada uno de los archivos JavaScript de nuestro proyecto.
- **Distribución de carpetas y ficheros:** Breve explicación de la estructura del proyecto respecto directorios y archivos
- **Comentarios adicionales:** Comentarios que hemos pensado necesarios o no pertenecen a ninguna otra sección específica de la documentación

1.4 Integrantes del Proyecto

- **Juan Francisco Hernández Fernández**
 - juan-francisco1@estudiant.uib.cat
 - <https://github.com/juanfco95>
- **Marco Martínez González**
 - marco.martinez1@estudiant.uib.cat
 - <https://github.com/MarcoMG2000>
- **Jordi Sevilla Marí**
 - jordi.sevilla1@estudiant.uib.cat
 - <https://github.com/JordiSM>

2. Café con Cultura

2.1. Descripción

Café con Cultura se trata de una Aplicación Web creada para la asignatura de [Tecnología Multimedia](#) con la finalidad de adquirir competencias en:

- Maquetación de páginas web con HTML5.
- Aplicación de estilos mediante CSS3.
- Control del comportamiento de la página con JavaScript.
- Definición de ficheros JSON según la esquemática de [Schema.org](#).
- Carga dinámica de HTML a raíz del contenido almacenado en archivos JSON.
- Diseño web *responsive*.
- Utilización de APIs.
- Integración de contenido multimedia (imágenes, vídeos y audios).

2.2. Funcionalidades de la Web-App

- **Top Cafeterías:** Mostrará al usuario el top 3 de cafeterías con un rating de puntuación más alto. Haciendo clic en una de las cafeterías iremos a la página de la cafetería.
- **Cafeterías cercanas:** Esta funcionalidad permitirá mostrar al usuario que cafeterías se encuentran más próximas a su ubicación, haciendo uso de una API de geolocalización. Al igual que en el apartado anterior, al clicar cargaremos la página de la cafetería.
- **Próximos eventos:** Permite al usuario visualizar los próximos eventos teniendo en cuenta la fecha y hora en la que el usuario esté haciendo uso de la Web-App (mostrará los eventos más cercanos en el tiempo que todavía no han sucedido), además de aparecer los eventos próximos, el usuario podrá clicar en ellos para consultar información adicional del evento.
- **Cafeterías y eventos consultados recientemente:** Tendremos una sección donde, si el usuario ha visitado previamente alguna cafetería o evento, mostrará un carrusel con las últimas cinco cafeterías y/o eventos que se han visitado. Al visitar una cafetería o evento lo guardamos en *Local Storage* para saber cuáles ha visitado. Si el usuario no ha visitado ninguna cafetería ni evento, no aparecerá la sección. Si el usuario solo ha visitado cafeterías o eventos, únicamente se mostrará su respectivo carrusel. Si el usuario ha visitado cafeterías y eventos, aparecerán sendos carruseles.
- **Equipo:** Pequeña sección con imágenes de los desarrolladores de la Web App con enlaces a sus perfiles de [LinkedIn](#) y [GitHub](#), además de un video hablando sobre la Web App
- **Consultar una cafetería:** Al hacer clic sobre una cafetería nos aparecerá la información de la cafetería que hemos clicado:
 - Nombre de la cafetería.
 - Breve descripción de la cafetería.
 - Valoración de los usuarios.

- Horario de apertura.
- Ubicación.
- Rango de precios.
- Información de contacto.

También mostraremos un carrusel con imágenes de esa cafetería y la ubicación de la cafetería con la API de Google Maps. Finalmente, tendremos una lista con todos los eventos que todavía no han sucedido y están planeados para esa cafetería un día en concreto. Haciendo clic en uno de estos eventos iremos a la página descrita en el siguiente apartado.

- **Consultar un evento:** De la misma forma que al hacer clic en una cafetería, al hacer clic sobre un evento veremos su información:

- Nombre del evento.
- Breve descripción del evento.
- Edad recomendada de audiencia.
- Fecha y hora de inicio y fin del evento.
- Precio de la entrada.
- Nombre del anfitrión del evento, en el cual podremos hacer clic para ir a su página personal.
- Aforo máximo del evento.
- Ubicación del evento, es decir, la cafetería donde sucede (se puede clicar para ir a la cafetería) y la calle donde está ubicada tal cafetería.

También dispondremos de un carrusel con imágenes del evento y un mapa con la ubicación de la cafetería donde tiene lugar. Finalmente, y al igual que en las cafeterías, tendremos una sección con todas las cafeterías donde tendrá lugar el evento en cuestión.

- **Reproductor de Música:** Reproductor de Spotify que se ha integrado al final de la página con la API de Spotify. Nos permite reproducir una playlist de Spotify seleccionada por los desarrolladores. El reproductor no se interrumpe durante la navegación de las distintas vistas de la página web debido a la carga dinámica de HTML en el index. Como todo el contenido (quitando el *header*, el *footer*, y el reproductor) están fuera del contenedor <main> del index.html, nos permite mantener estos elementos activos en todo momento mientras navegamos por el resto de páginas (home.html, cafeteria.html, evento.html...). En la versión de Móviles, la Web App solo admite la versión preliminar de la playlist, permitiendo únicamente reproducir 30 segundos por canción

- **Buscador:** Podremos ver un listado de todos los eventos (o cafeterías) dentro de nuestra WebApp. Además, tendremos la opción de aplicar diferentes filtros, en función de los rangos de precio, las categorías, las valoraciones...;
- **Mapa:** Muestra un mapa interactivo de la Isla de Mallorca mediante el uso de la API de Google Maps (más concretamente, Maps Javascript API). Se mostrarán diversos marcadores que representarán cada uno de los puntos de interés recogidos de los diversos JSON (cafeterías, restaurantes, bodegas y monumentos), cada uno representado por un ícono SVG distinto. Cada uno de estos marcadores podrán clicarse para mostrar su dirección en un mensaje emergente en el Mapa y toda la información más detallada en la sección situada a la derecha del Mapa (o debajo si se encuentra en el móvil).

Además, el usuario tendrá la opción de poder filtrar los marcadores para poder elegir cuáles mostrar, o todos a la vez.

3. Proceso de maquetación y diseño gráfico.

3.1. Hosting y Dominio

Tanto para la selección y compra del Dominio, como para el Hosting de la Aplicación Web, se han usado los servicios de la Empresa [DonDominio](#).

- **Dominio:** <https://www.cafeconcultura.com/>
- **JSON:** <https://www.cafeconcultura.com/assets/JSON/cafeterias.json>

3.2. Diseño gráfico.

El diseño gráfico de la página se ha llevado a cabo con la ayuda de una [plantilla](#) <https://bootstrapmade.com/demo/Gp/> con código HTML5, CSS y JavaScript de ejemplo que hemos retocado para adaptarlo a nuestras necesidades. Hemos seleccionado elementos de la plantilla que encajaban con el diseño que teníamos planteado de la Web-App y las hemos adaptado mediante la modificación de su código. Asimismo, también hemos añadido funcionalidades que no estaban ya pensadas en la plantilla (como la búsqueda de cafeterías/eventos, la sección de cafeterías visitadas recientemente, cafeterías guardadas, ...).

En cuanto a nuestro fichero style.css, está dividido por secciones, de manera que nos facilite la modificación del estilo de cualquier elemento, pese a que este se repita en distintas secciones. Hemos conservado los estilos de la plantilla de las secciones que hemos conservado, modificándolas en gran medida para que se ajuste a nuestras necesidades.

3.3. Maquetación

El HTML de nuestra Web-App consiste en un fichero index.html con un *header* y un *footer* y un reproductor con la API de Spotify, que se mantendrán visibles durante la ejecución de cualquier sección, sin embargo, el contenido del contenedor *main*, que aparece vacío inicialmente en index.html, cargará el código correspondiente a la sección en la que se encuentre el usuario de forma dinámica haciendo uso de una función de JavaScript llamada *cargarContenido*, que obtiene como parámetros el nombre del archivo HTML con el contenido que tiene que cargar en el contenedor. Esta función está implementada en el archivo navegacion.js y hemos modificado el archivo main.js para que llame a esta función con el nombre home.html al finalizar la carga de la página para que muestre la página principal. De este modo, evitamos cargar el código del *header* y el *footer* de forma innecesaria. Además, tenemos otras funciones en el archivo de navegacion.js donde cargamos dinámicamente el html con la información recibida del JSON de cafeterías.

El uso de esta plantilla de [BootstrapMade](#) nos ha ahorrado la implementación de los menús para navegar desde un dispositivo móvil.

3.4. Formato de Datos

Para el formato de datos, tanto para las cafeterías como para los distintos tipos de eventos que mostraremos en la Web, serán guardados con formato JSON. Se utilizarán los estándares de **Schema.org** tomando como referencia:

- **Cafetería:** <https://schema.org/CafeOrCoffeeShop>
- **Eventos:** <https://schema.org/Event>

En adición, se ha querido añadir un último atributo “**keywords**” que servirá para filtrar las diferentes cafeterías según diferentes características de la misma, como “**Juegos de Mesa**”, “**Comida y Bebida**”, “**Música**”, etc.

3.5. API's.

Hemos decidido usar 3 API's externas para dar apoyo a las funcionalidades comentadas en la sección 2.2.. Las API's correspondientes son:

- **Web Storage API:** La API proporciona los mecanismos mediante los cuales el navegador puede almacenar información de tipo clave/valor, de una forma mucho más intuitiva que utilizando cookies. Funcionalidades de la Aplicación Web:
 - **Listado de cafeterías/eventos consultados recientemente**
 - https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API
- **Spotify Web API:** La API permite la creación de aplicaciones que pueden interactuar con los servicios de streaming de Spotify, como la recogida de metadatos, obtención de recomendaciones, creación y administración de listas de reproducción, o el control del reproductor. Funcionalidades de la Aplicación Web:
 - **Reproducción de Lista de reproducción de la Aplicación Web CaféConCultura.**
 - <https://developer.spotify.com/documentation/web-api>
- **Google Maps API:** La API permite la creación de aplicaciones que pueden interactuar con los servicios de Google, mediante la creación de mapas dinámicos, identificación de rutas cercanas mediante coordenadas y el cálculo de distancias y los tiempos de viaje para varios orígenes y destinos. Funcionalidades de la Aplicación Web:
 - **Diseño de Mapa personalizado**
 - **Marcadores de mapa dinámicos, obtenidos a través del JSON de cafeterías**
 - **Marcadores de mapa dinámicos, obtenidos a través del JSON de restaurantes, bodegas, monumentos... obtenidas de los ficheros JSON de nuestros compañeros.**
 - **Filtro de Marcadores**
 - **Listener de Marcadores (Función que se ejecuta al clickar sobre un marcador)**
 - <https://developers.google.com/maps/documentation/javascript?hl=es-419>

3.6. Librerías.

- **FontAwesome:** [FontAwesome](#) es una biblioteca de iconos y conjunto de herramientas de Internet, utilizado por millones de diseñadores, desarrolladores y creadores de contenidos. Los iconos de esta librería están en formato SVG, por lo que consigue con poco espacio de memoria mostrar una amplia lista de iconos re-escalables sin pérdida de detalle.
- **JQuery:** [jQuery](#) es una biblioteca multiplataforma de JavaScript, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.
- **Bootstrap:** [Bootstrap](#) es un framework de desarrollo web front-end que facilita la creación de sitios estilizados y responsive. Proporciona una colección de plantillas, componentes y estilos predefinidos que agilizan el proceso de diseño y optimizan la experiencia del usuario.
- **AOS:** [aos](#) es una librería que nos permite, de forma sencilla, agregar animaciones de aparición a los distintos elementos de nuestra página web, lo que le da un aspecto más fresco y dinámico a nuestra página.

3.7. Inclusión de MEDIA

Además de los SVG que incorpora la librería de FontAwesome, también hemos creado un SVG con el logo de CaféConCultura, que se muestra en el *header* de la página, y un ícono de una taza de café que utilizamos a modo de Favicon en la pestaña del navegador. En cuanto a los iconos de los marcadores en la sección de Mapa-CafeConCultura de las cafeterías, restaurantes, bodegas,...; los hemos obtenido a través de este servicio gratuito. <https://freesvg.org/>

Respecto a videos, tenemos un carrusel con vídeos de algunas de las cafeterías y eventos incluidos en nuestro fichero JSON, por ejemplo tenemos vídeos de presentación de la cafetería Capuccino de Valldemossa y trailers de algunos eventos como Pirates Adventure, u Ohalá.

Además, hemos grabado un vídeo donde contamos resumidamente el objetivo de la página web y el propósito de su desarrollo.

4. Desarrollo de funciones con JavaScript.

Para llevar a cabo todas las funcionalidades que tiene nuestra aplicación, hemos tenido que desarrollar los siguientes ficheros de código en JavaScript.

4.1. main.js

En main.js tenemos las funciones implementadas por la plantilla de BootstrapMade, únicamente nos hemos ocupado de eliminar el código inútil al eliminar algunas de las librerías con las que estaba implementada la plantilla y, también hemos añadido una llamada a "cargarContenido("home.html");", para que cargue la página principal en el contenedor <main> una vez que la página haya terminado de cargar.

4.2. navegacion.js

En navegacion.js hemos desarrollado aquellas funciones relacionadas con la carga dinámica de código html en nuestra página web. Las funciones implementadas son las siguientes:

- **function cargarContenido(nombreArchivo, nombreCafeteria, nombreEvento):** Esta función es la encargada de cargar el html de los distintos archivos en el contenedor, el primer parámetro es el nombre del archivo que queremos cargar, véase home.html, cafeteria.html o buscador.html, y el segundo y tercer parámetro son necesarios para cargar algunos archivos, por ejemplo, para cargar un evento necesitamos saber el nombre del evento y el nombre de la cafetería donde tiene lugar dicho evento según el enlace clicado.
- **function cargarHome():** Se encarga de hacer el XMLHttpRequest de nuestro fichero JSON de cafeterías para luego llamar a las diversas funciones que generan código html dinámico para la página principal, basándonos en los datos obtenidos.
- **function clickCafeteria(nombreCafeteria):** Función que se ejecuta al hacer clic sobre una cafetería. Lee la cafetería del JSON, llama a la función que carga el html de la cafetería y guarda la cafetería en *LocalStorage*.
- **function clickEvento(nombreCafeteria, nombreEvento):** Función homóloga a clickCafeteria, pero obteniendo y cargando la información de un evento en lugar de una cafetería.
- **function cargarBuscador(nombre, bool):** Función encargada de cargar el archivo buscador.html. El parámetro nombre nos permite saber si el buscador que se ha seleccionado es de cafeterías o de eventos, y el booleano se utiliza para que al generar el código html desde js no se genere de nuevo al aplicar los filtros todo el código de la selección de filtros, de esta manera si el booleano vale true se cargará el código de la selección de filtros y si vale false simplemente se cargan las cafeterías que cumplen con los filtros. Esta función realiza una *request* para obtener la lista de cafeterías de nuestro JSON, recoge los filtros seleccionados mediante la función filtrosSeleccionadosCaf() si nombre es igual a Cafeterías o filtrosSeleccionadosEv() si nombre es igual a eventos, y realiza una llamada a cargarBusquedaCafe(listaCafeterias, filtros, bool) o cargarBusquedaEvent(listaCafeterias, filtros, bool) respectivamente.
- **function cargarCafeteriasPorValoracion(listaCafeterias):** Genera la sección de top-cafeterias, ordenando las cafeterías por valoración descendente y mostrando las tres primeras.
- **async function cargarCafeteriasPorCercania(listaCafeterias):** Genera la sección de closest-cafeterias, calculando la distancia del usuario desde su ubicación (obtenida mediante geolocalización) a cada una de las cafeterías, luego ordenando por distancia de forma ascendente y mostrando las tres primeras.

- **function cargarEventos(listaCafeterias):** Genera la sección de events con los cuatro próximos eventos desde la fecha actual, sin mostrar los eventos que ya han sucedido y ordenando los que aparecen por orden de fecha.
- **function cargarRecientes(listaCafeterias):** Genera dos, uno o ningún carrusel de cafeterías y/o eventos en función de la información disponible en el Local Storage mostrando, en el caso de haber visitado previamente alguna cafetería o evento, las cinco últimas cafeterías y/o eventos que el usuario ha visualizado.
- **function cargarCafeteriaClickada(listaCafeterias, nombreCafeteria):** Genera el html de la cafetería que se ha clicado. Recibe como parámetro el nombre de la cafetería clicada.
- **function cargarEventoClickado(listaCafeterias, nombreCafeteria, nombreEvento):** Genera el html del evento que se ha clicado. Recibe como parámetro el nombre del evento clicado y la cafetería donde tuvo o tendrá lugar.
- **async function cargarBusquedaCafe(listaCafeterias, filtros, primeraVez):** Función encargada de generar el código html que será insertado en el archivo buscador.html, primeraVez es una variable booleana que nos permite controlar la generación del código de selección de filtros en función de si se accede por primera vez o se está accediendo tras haber aplicado filtros, en cuyo caso no será necesario generar de nuevo ese código. Esta función utiliza la función cumpleFiltrosCaf para comprobar si el elemento i de la lista que recibe por parámetro cumple con los filtros y de ser así genera el código necesario para que dicha cafetería aparezca en el buscador. Además se comprueba si la cafetería está abierta comprobando el horario y se muestra si está abierta o cerrada a través del html generado. También cabe mencionar que el rango del *slider* del filtro distancia, toma la geolocalización del usuario y establece el rango [0, Distancia a la cafetería más lejana]. El filtraje se realiza de modo que basta que una cafetería cumpla con uno de los filtros para que aparezca.
- **async function cargarBusquedaEvent(listaCafeterias, filtros, primeraVez):** Función análoga a async function cargarBusquedaCafe(listaCafeterias, filtros, primeraVez), pero para eventos. La diferencia en este caso, aparte de la selección de filtros, es que en este caso para que un evento aparezca debe de cumplir con todos los filtros seleccionados.
- **function cargarContenidoMap():** Función análoga a cargarContenido, pero en adición se creará un Elemento <script> necesario para la llamada a la API de GoogleMaps.
- **function vaciarJSONLD():** Elimina todas las etiquetas <script> que tengan type="json-ld" de la cabecera del html.
- **function addElementoJSONLD(elemento):** Añade un elemento <script type="json-ld"> a la cabecera de la página web con contenido de un objeto de JavaScript en formato texto.
- **function addListaJSONLD(lista):** Utiliza la función addElementoJSONLD para añadir un array de objetos a la cabecera de nuestra página, cada uno de ellos con su etiqueta <script type="json-ld">.

4.3. utilidades.js

En utilidades.js hemos codificado las funciones de apoyo para cambiar de formatos, realizar cálculos, ordenar listas... Las funciones de este fichero son:

- **function ordenarLista(lista, atributo, orden = "ascendente"):** Ordena la lista del primer parámetro por el atributo que elijamos de forma ascendente o descendiente, dependiendo del tercer parámetro (por defecto ascendente).
- **function obtenerValor(objeto, ruta):** función auxiliar de ordenarLista que nos da el atributo seleccionado de un objeto para poder compararlo.
- **function getMaxDistance(listaCafeterias):** Devuelve la máxima distancia de todas las cafeterías al usuario, empleada para establecer el valor máximo para el *slider* del buscador por distancia.
- **function comprobarEstadoDeNegocio(openingHours):** Comprueba, según la fecha y hora actual, si la cafetería está abierta o cerrada en función de su horario de apertura.
- **function comprobarSiEstaAbiertoEnEsteDia(diaInicio, diaFin):** Función auxiliar de comprobarEstadoDeNegocio que comprueba si el día concuerda con el horario para saber si la cafetería está abierta.
- **function comprobarSiEstaAbiertoEnEstaHora(horaInicio, horaFin, horaActual):** Función auxiliar de comprobarEstadoDeNegocio que comprueba si la hora concuerda con el horario para saber si la cafetería está abierta.
- **function storeCafeteria(nombre):** Función encargada de guardar en el Local Storage las 5 últimas cafeterías clicadas por el usuario. Esta función recibe el nombre de la cafetería clicada y tras leer la lista de cafeterías llama a buscarCafeteriaPorNombre que lo que hará es buscar la cafetería cuyo atributo name sea igual a nombre y devolverla, de esta manera podremos guardar el objeto cafetería completo en el local storage.
- **function storeEvento(nombre):** Función análoga a la anterior pero para eventos, en este caso se usa buscarEventoPorNombre como función auxiliar para encontrar el evento cuyo atributo name sea igual a nombre.
- **function filtrosSeleccionadosCaf():** Esta función recoge los filtros que han sido seleccionados desde el html los recoge todos dentro de un array y lo devuelve para que después puedan ser comprobados a la hora de cargar las cafeterías en buscador.html.
- **function filtrosSeleccionadosEv():** Función análoga a la anterior pero para eventos.
- **function cumpleFiltrosCaf(cafeteria, filtros):** Esta función recibe por parámetro los filtros que han sido seleccionados y una cafetería del JSON, si esta cafetería cumple con los filtros que el usuario ha seleccionado desde el html, entonces se generará el código para mostrar dicha cafetería.

- **function cumpleFiltrosEv(evento, cafeteria, filtros):** Función análoga a la anterior pero para eventos. En este caso necesitamos también la cafetería en la que se realiza el evento para que se pueda comprobar el filtro de distancia.
- **function traducirHorarioApertura(horario):** Interpreta el horario de las cafeterías del JSON y nos devuelve en un lenguaje más natural los rangos de días y horas en los que una cafetería está abierta.
- **function obtenerUbicacionUsuario():** Nos da la latitud y longitud del usuario según la geolocalización realizada por el navegador (es necesario conceder los permisos de acceso a la ubicación).
- **function calcularDistancia(latitud1, longitud1, latitud2, longitud2):** Calcula la distancia que hay entre un usuario y una cafetería, siendo las coordenadas del usuario la latitud1 y longitud1, mientras que las coordenadas de la cafetería vienen indicadas por la latitud2 y longitud2.
- **function gradosARadianes(grados):** Convierte un valor de grados sexagesimales a radianes.
- **async function obtenerDistanciasCafeterias(listaCafeterias):** Llama a obtenerDistanciasCafeterias con las coordenadas del usuario y cada una de las coordenadas de las cafeterías, almacenando el valor de esta función en el atributo distancia de la listaCafeterias.
- **function obtenerListaEventos(listaCafeterias):** Recorre la lista de cafeterías, generando una lista con todos los eventos que todavía no han sucedido.
- **function obtenerListaTotalEventos(listaCafeterias):** Recorre la lista de cafeterías, generando una lista con todos los eventos que todavía no han sucedido y también los que ya han sucedido.
- **function getCurrencySymbol(priceCurrency):** Devuelve el símbolo asignado al código ISO de las 10 monedas más utilizadas (USD, EUR, GBP, JPY...)
- **function buscarEventoPorNombre(listaCafeterias, nombreEv):** Realiza un recorrido por la lista de cafeterías y por cada una de ellas realiza otro recorrido por sus eventos para devolver aquel evento cuyo atributo name sea igual a nombreEv.
- **buscarCafeteriaPorNombre(listaCafeterias, nombreCaf):** Función análoga a la anterior pero en este caso se realiza un recorrido y se devuelve aquella cafetería cuyo atributo name sea igual a nombreCaf.

4.4. maps-api.js

En maps-api.js tenemos las funciones necesarias para inicializar y controlar el comportamiento de nuestra sección Mapa-CafeConCultura. En ella tendremos unas variables que nos ayudarán con las funciones explicadas más adelante

Variables:

- **map:** Estancia de un objeto [google.maps.Map](#), Se trata de la estancia del Mapa que se mostrará en la sección. Este mapa se le asignarán marcadores [google.maps.Marker](#).
- **listaCafeterias, listaRestaurantes, listaBodegas, listaMonumentos:** se tratan la lista de “itemListElement” de cada uno de los JSON's. Se guardan en una variable para no tener que realizar consultas cada vez que se interactúa con un marcador, y poder recogerlas con mayor eficiencia. Estas listas se resetean cada vez que se recarga la sección del Mapa.
- **coffeMarkers, restaurantMarkers, bodegaMarkers, monumentosMarkers:** Array de cada uno de los marcadores [google.maps.Marker](#), inicializados en las funciones **load**. Estos marcadores serán guardados con el propósito de obtener una mayor eficiencia en la función de **filterMap(tipoMarcador)**, explicada más adelante.

Funciones:

- **function initMap():** Función “CallBack” de la llamada a la API de GoogleMaps, en caso de que la conexión haya sido un existo, se creará una nueva estancia de “map” y se llamarán a cada una de las funciones de **loadJSON**.
- **function loadCafeterias():** Función que se encarga de recoger el archivo **cafeterias.json** y crear un nuevo marcador [google.maps.Marker](#), con su icono SVG correspondiente, ubicación y título (Nombre de la cafetería). A cada marcador se le asignará un [google.maps.InfoWindow](#), que mostrará la dirección completa de la cafetería; y un *listener*, para llamar a funciones cuando se clique sobre el marcador; y se guardarán en la lista **listaCafeterias**.
- **function loadRestaurantes():** Función que se encarga de recoger el archivo **restaurantes.json** y crear un nuevo marcador [google.maps.Marker](#), con su icono SVG correspondiente, ubicación y título (Nombre del restaurante). A cada marcador se le asignará un [google.maps.InfoWindow](#), que mostrará la dirección completa del restaurante; y un *listener*, para llamar a funciones cuando se clique sobre el marcador; y se guardarán en la lista **listaRestaurantes**.
- **function loadBodegas():** Función que se encarga de recoger el archivo **bodegas.json** y crear un nuevo marcador [google.maps.Marker](#), con su icono SVG correspondiente, ubicación y título (Nombre de la bodega). A cada marcador se le asignará un [google.maps.InfoWindow](#), que mostrará la dirección completa de la bodega; y un *listener*, para llamar a funciones cuando se clique sobre el marcador; y se guardarán en la lista **listaBodegas**.

- **function loadMonumentos():** Función que se encarga de recoger el archivo **monumentos.json** y crear un nuevo marcador **google.maps.Marker**, con su icono SVG correspondiente, ubicación y título (Nombre del monumento). A cada marcador se le asignará un **google.maps.InfoWindow**, que mostrará la dirección completa del monumento; y un *listener*, para llamar a funciones cuando se clique sobre el marcador; y se guardarán en la lista **listaMonumentos**.
- **function iconMapClickCafeteria(marker):** Función que se llama al interactuar con un marcador del Mapa. Obtendremos la cafetería de la lista **listaCafeterias**, gracias al índice guardado en el marcador (ZIndex) inicializado durante la función **loadCafeterías**; y mostraremos la información de la cafetería en la sección lateral (o inferior si es en el móvil) de la APP.
- **function iconMapClickRestaurante(marker):** Función que se llama al interactuar con un marcador del Mapa. Obtendremos el restaurante de la lista **listaRestaurantes**, gracias al índice guardado en el marcador (ZIndex) inicializado durante la función **loadRestaurantes**; y mostraremos la información del restaurante en la sección lateral (o inferior si es en el móvil) de la APP.
- **function iconMapClickBodega(marker):** Función que se llama al interactuar con un marcador del Mapa. Obtendremos la bodega de la lista **listaBodegas**, gracias al índice guardado en el marcador (ZIndex) inicializado durante la función **loadBodegas**; y mostraremos la información de la bodega en la sección lateral (o inferior si es en el móvil) de la APP.
- **function iconMapClickMonumento(marker):** Función que se llama al interactuar con un marcador del Mapa. Obtendremos el monumento de la lista **listaMonumentos**, gracias al índice guardado en el marcador (ZIndex) inicializado durante la función **loadMonumentos**; y mostraremos la información del monumento en la sección lateral (o inferior si es en el móvil) de la APP.
- **function filterMap(tipoMarcador):** Función encargada de filtrar los marcadores que se ven en el mapa a partir del parámetro **tipoMarcador** de la función. Tendremos la opción de mostrar **individualmente** las cafeterías, restaurantes, bodegas y monumentos, o mostrar **todos los marcadores** al mismo tipo. No será necesario crear los marcadores de nuevo, sino llamar a la función **Marker.setMap(mapa)**, para asignar el marcador al mapa; o eliminarlo (visualmente) llamando a **Marker.setMap(null)**; a cada uno de los marcadores que queramos mostrar, u ocultar, almacenados en los *arrays* de Marcadores explicados anteriormente.

En las sección donde se muestran la información de la cafetería, restaurante, bodega o monumento seleccionado, Se mostrará el **nombre** del lugar, la **descripción** del lugar (Detallada en el JSON correspondiente), su **dirección completa** y un link de “**Más información**”, que nos llevará a la sección de la cafetería correspondiente; o en caso de que sea un restaurante, bodega o monumento, nos llevará a la sección individual del lugar en la Aplicación Web: “monumentosmallorca.com”, “mllcarestaurantes.com” o “mallorcabodegas.com”

5. Distribución de carpetas y ficheros

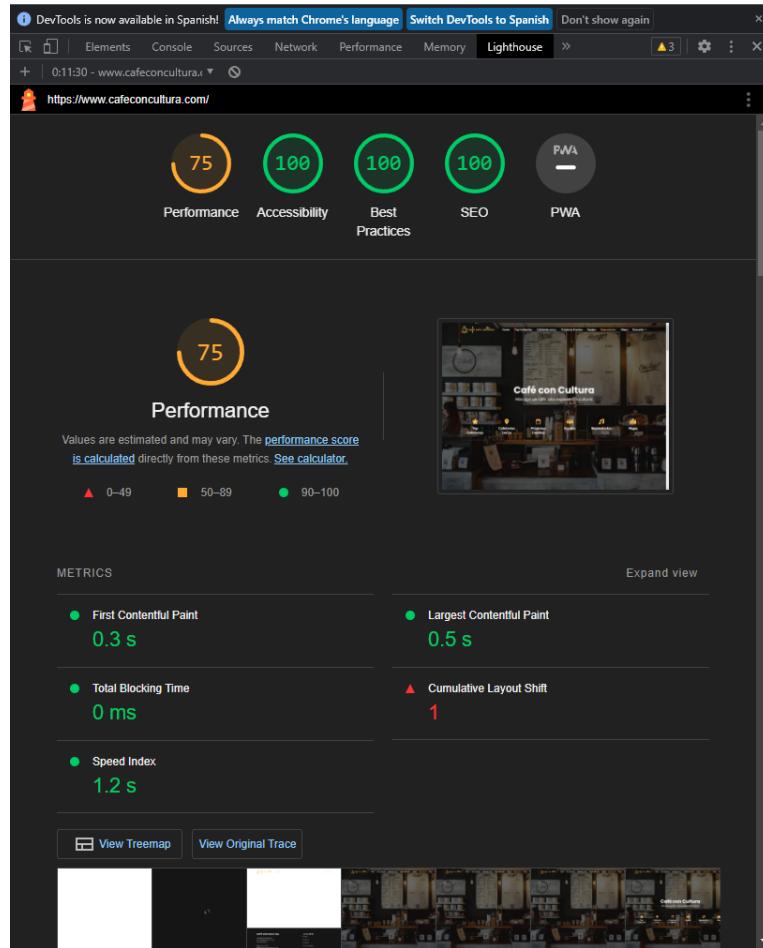
A continuación explicaremos la distribución de las carpetas y ficheros de la Aplicación:

- **index.html:** Archivo principal en el cual se irá modificando el “main” tal como se explicó en el apartado **3.3.**
- **“Secciones HTML”:** No se tratan de una carpeta, sino de una agrupación de archivos situados en la misma ruta que **index.html**. Estos archivos se tratan de segmentos de código html los cuales serán cargados en el archivo principal **index.html**. Cada uno de estos archivos representa una sección de la APP.
 - **buscador.html**
 - **cafeteria.html**
 - **evento.html**
 - **home.html**
 - **map.html**
- **assets**
 - **css:** Directorio con todos los archivos CSS de la APP
 - **favicon:** Directorio donde se encuentra el Favicon de la APP
 - **img:** Directorio que almacena todas las imágenes de eventos y cafeterías relacionadas con el JSON, iconos SVG propios e imágenes auxiliares usadas en el *index.html* y *home.html*
 - **js:** Directorio con los ficheros de código JavaScript.: Directorio donde se recogen todos los archivos de JavaScript
 - **JSON:** Directorio para los distintos ficheros JSON utilizados.
 - **vendor:** Directorio con todas las. Librerías externas ya proporcionadas por la Plantilla utilizada, utilizado principalmente para los iconos SVG de la APP, bootstrap para diseño flexible de rows y columns, aos para la carga de la visualización, y jQuery para algunas consultas sobre elementos del html.
 - **video:** Directorio con todos los vídeos que aparecen en la página, es decir, el carrousel de vídeos de las cafeterías y eventos de nuestro JSON y el vídeo propio explicando la práctica.

6. Semántica

Para aplicar semántica a la página web hemos utilizado JSON-LD (JavaScript Object Notation for Linked Data) aprovechando que nuestro JSON sigue el esquema de Schema.org. Esto nos ha permitido desarrollar tres simples funciones que nos permiten cargar dinámicamente el html de la página con sus respectivas etiquetas `<script type="json-ld"> ... </script>`, donde hemos introducido el texto con los atributos de cada objeto presente en la página. Los objetos del JSON ya tienen por defecto las etiquetas `@type`, `@context...` para que las etiquetas de la cabecera permitan al navegar obtener información del contenido de nuestra página y optimizar las búsquedas. En todo momento solo están las etiquetas de los elementos que se están mostrando por pantalla en ese preciso instante.

7. Rendimiento



La página web queda obtiene una valoración de **100%** en **accesibilidad, buenas prácticas y SEO**, en cuanto al **rendimiento** hemos alcanzado el **75%** gracias a emplear formatos de imagen webp, formato de vídeo webm, iconos svg, ajustar la resolución de las imágenes y vídeos al tamaño que se muestra, minimizar los archivos tanto de vendor como nuestro html, la carga inteligente de funciones javascript y ficheros css, la utilización de la etiqueta preload de vídeos y lazy load en imágenes e iframes...

Sin embargo, la página no nos ha sido posible optimizarla para una performance de 100 ya que al utilizar una única página no tenemos back/forward caché cuando navegamos entre cafeterías o eventos, tampoco hemos podido configurar la caché de los archivos estáticos y, finalmente, nos valora negativamente el CLS al no reservar espacio para los DIV donde cargamos dinámicamente HTML con nuestras funciones JavaScript. Tampoco podemos cargar los archivos minimizados de Bootstrap, AOS y nuestro css, ya que la sección Hero (la que se muestra al principio de la página no se mostraría hasta que estas tres estén descargadas, sin embargo, el tiempo que tardan en cargar lo podríamos considerar “despreciable” y por lo tanto, hemos decidido mantener la animación de AOS en el Hero pese a ser “menos óptima”.

8. Web responsive

Gracias a la librería BootStrap, hemos logrado hacer la página web 100% *responsive*, a continuación mostramos unas imágenes de la página web desde navegador y celular.



RATING

CAFETERÍAS MEJOR VALORADAS

 Goblintrader Mallorca ● Abierto ★★★★★ Carrer de Rosselló i Cazador, 7
 Cafetería Parabellum ● Abierto ★★★★★ Carrer del Bisbe Rafael Josep Verger, 6
 Literanta ● Abierto ★★★★★ Carrer del Call, 4



9. Comentarios adicionales

Al cargar todas las páginas sobre el contenedor `<main>` del `index.html`, solo es necesario añadir los `<link =...>` y los `<script src=...>` en `index.html`, de este modo, la carga de los *assets* se realiza una única vez al abrir la página. **ÚNICAMENTE**, para la sección de Mapa, se creará un “script” necesario para la API de GoogleMaps cada vez que se llame a la función `CargarContenidoMapa()`.

El *footer* de nuestra página tiene enlaces a términos de servicio, políticas de privacidad, un formulario para suscribirse a un boletín de noticias... Estos enlaces no están activos y se han conservado de la plantilla por motivos estéticos, dado el alcance de esta práctica, excedería los objetivos que esta plantea.

La plantilla disponía de funciones que nos muestran el elemento del *navbar* donde nos encontramos en todo momento y además incluía una función `scrollto` que nos desplaza al elemento que hemos clicado en el *navbar* o en el *hero*, hasta el elemento con dicho ID con un desplazamiento suave y con un `offset` para lo que ocupa nuestra barra de navegación.

Las funciones `async` son aquellas donde tenemos que llamar a la función que ordena cafeterías u obtiene sus distancias, ya que al llamar a una función el código JavaScript continúa ejecutando código y necesitamos que las funciones terminen sus cálculos (`await`).

La página web es completamente responsive gracias a la librería bootstrap. Nos hemos encargado expresamente en asegurarnos de que todo el contenido es visualizable y accesible desde cualquier dispositivo y para cualquier tamaño de ventana. Gracias a la plantilla que hemos utilizado, nos hemos ahorrado la preocupación de desarrollar el botón que despliega el menú de la barra de navegación cuando esta es muy pequeña.

10. Autovaloración.

Pese a las posibles mejoras que podríamos haber implementado, como guardar cafeterías en favoritos, el back/forward caché, mejoras en el rendimiento, filtros de búsqueda más detallados, etc. Hemos realizado funcionalidades similares como las **cafeterías/eventos recientemente visitados**, búsqueda de **cafeterías cercanas** mediante la Ubicación del usuario, y ser los únicos (o los primeros en proponer la idea) del uso de **API's de Spotify** para la reproducción de una Playlist durante la navegación de cualquier sección de la APP; o toda la sección de Mapa, mediante el uso de la **API de GoogleMaps with JavaScript** (no solo usando un <iframe>, sino toda la creación de objetos y marcadores ya explicadas en la **sección 4.4 map-api.js.**)

Consideramos que dados los objetivos establecidos para la práctica de Tecnología Multimedia, cumplimos con todos los requisitos necesarios, además de tener un diseño de la página web con un aspecto muy profesional. Y valoraríamos nuestro trabajo realizado con un **9,5 sobre 10**, ya que pese a tener margen de mejora, hemos cumplido satisfactoriamente todos los objetivos.