



Proyecto DAW

“Aplicación web de consulta de establecimientos próximos”



Alumno: Mateo Fortea Dugo 2º DAW

Tutor del proyecto: Alfonso Jiménez Vílchez

Índice

1 - Descripción del proyecto	4
2 - Objetivos del proyecto	4
3 - Análisis previo	4
4 - Justificación de la elección	12
5 - Recursos necesarios	13
5.1 - Hardware	13
5.2 - Software	14
5.3 - Servicios online	14
6 - Desarrollo del proyecto	15
6.1 - Modelado de la información	15
6.2 - Programación	16
6.2.1 - Guía de estilo de codificación	16
Código PHP	16
Código JavaScript	19
Plantillas Twig	20
6.2.2 - Organización física del código	20
6.2.3 - Clases y funciones	28
Parte servidor (PHP)	28
Clases del controlador y sus funciones	28
Clases de entidades (Modelo)	32
Clases de formularios	34
Clases del servicio de recogida de datos	37
Parte cliente (JavaScript)	41
Funciones de llamada al servidor	41
Fichero de utilidades (utiles.js)	45
6.2.4 - Pruebas	47
6.3 - Desarrollo web	50
6.3.1- Estructura del sitio web	50
6.3.2 - Maqueta del sitio web	51
6.3.3 - Manual de diseño	64
Logotipos	64
Colores	64
Imágenes	66
Tipografías	67
Frameworks de diseño	68
Iconografía	69

6.3.4 - Usabilidad	70
6.3.5 - Accesibilidad	73
Tests de Accesibilidad	74
6.3.6 - Proceso de despliegue	76
Plataforma de despliegue	76
Despliegue de la base de datos	78
7 - Dificultades encontradas	78
8 - Conclusiones personales	80
9 - Posibles mejoras	81
10 - Fuentes de información	82
10.1 - Webgrafía	82
10.2 - Recursos utilizados	84

1 - Descripción del proyecto

Este proyecto consiste en el desarrollo de una **aplicación web que permita al usuario buscar y obtener información sobre lugares y establecimientos** a partir de su ubicación, almacenar y editar dichos lugares, de forma de que se pueda acceder a ellos en cualquier momento.

El hecho de ser una aplicación web, garantiza su accesibilidad en multitud de dispositivos, ya sean smartphones, tablets o equipos de escritorio.

El espectro de usuarios que pueden utilizar esta aplicación es muy amplio, desde personas que viajan con frecuencia y les gustaría ubicar sus destinos favoritos en un mismo lugar, pudiendo acceder a información sobre ellos en cualquier momento, a usuarios corrientes que quieran tener almacenados en una misma aplicación los lugares y establecimientos que suele frequentar.

2 - Objetivos del proyecto

El **objetivo principal** de este proyecto, es el **desarrollo de una aplicación web que pueda ofrecer al usuario información sobre los establecimientos cercanos y su ubicación**. A partir de este objetivo principal, distinguimos diferentes objetivos que podemos denominar **parciales**:

- Mostrar al usuario una lista de establecimientos cercanos e información sobre estos y su distancia respecto al usuario.
- Mostrar los establecimientos ubicados en un mapa.
- Permitir al usuario registrarse, modificar sus datos y darse de baja.
- Permitir al usuario registrado almacenar, editar y consultar una lista de sus establecimientos favoritos.
- Desarrollar teniendo en cuenta la limpieza del código, la seguridad, la accesibilidad y la usabilidad.
- Ofrecer una interfaz preparada tanto para ordenador como para dispositivos móviles.

3 - Análisis previo

En base a los objetivos planteados en el punto anterior, podemos realizar un análisis previo al desarrollo, donde tener en cuenta las diferentes herramientas, tecnologías y métodos que podemos utilizar, para más adelante, encontrar aquellos que sean más adecuados para nuestro proyecto:

Desarrollo Back End

Para el desarrollo del backend de la aplicación web, ya sea para el desarrollo de una API REST integrada en la aplicación, de forma separada o simplemente para generar las distintas páginas de forma dinámica, podemos hacer uso de algunos de los tres lenguajes siguientes complementados de algún framework:

Utilizar el **lenguaje PHP** para el backend, haciendo uso de algunos de los frameworks siguientes:

- **Symfony.** Algunas de sus características más destacadas son:

- Hace uso del patrón de diseño Modelo-Vista-Controlador, que separa de una forma más sencilla los datos de una aplicación, la interfaz de usuario, y la lógica de negocio.
- Usa Twig como motor de plantillas por defecto que permite herencia de plantillas, utilizar variables, se integra con HTML, hace uso de estructuras de control entre otras funciones.
- Hace uso de Doctrine como ORM (Mapeo objeto-relacional), que funciona de forma independiente al SGBD (Sistema Gestor de Base de Datos) que utilicemos.
- Dispone de sistemas de autenticación y gestión de credenciales, que permiten la restricción de secciones y una gestión segura de usuarios.
- Es compatible con Composer, un sistema de gestión de paquetes que permite manejar de forma sencilla una gran variedad de dependencias y librerías.
- Dispone de una potente línea de comandos que automatiza la generación de mucho código: asistentes para generar clases, la creación de tablas automáticas en la base de datos o el despliegue del proyecto de forma local entre otras funciones.

- **Laravel.** Sus funciones más destacadas son las siguientes:

- Al igual que Symfony, hace uso del patrón de diseño Modelo-Vista-Controlador.
- Utiliza el motor de plantillas Blade, integrado con HTML, basado en el uso de anotaciones (que permiten variables, estructuras de control...) y herencia completa y parcial de plantillas entre otras funciones.
- Gran soporte para el desarrollo de test unitarios y de integración, utilizando su propio componente para tests o bien utilizando PHPUnit, con el cuál es compatible.

- Posee gran cantidad de herramientas de seguridad sobre la base de datos, además de disponer de diferentes algoritmos de encriptación como Bcrypt Hashing.
- Usa Eloquent como ORM, que permite realizar sentencias SQL de forma integrada con el lenguaje PHP, ofreciendo independencia sobre el SGBD utilizado.
- Usa componentes de Symfony como Console (consola de Symfony), Debug (sistema de debugging de Symfony), Routing o CssSelector entre otros,
- Es compatible con Composer para la gestión de dependencias y librerías.

Hacer uso del **lenguaje Java**. Donde se puede combinar el uso de los dos frameworks siguientes:

- **Spring Framework.** Que tiene las características siguientes:

- Hace uso del patrón de diseño Modelo-Vista-Controlador. Facilita el uso de este patrón haciendo uso de anotaciones (@Service, @Controller o @Model), ahorrando gran parte del código.
- Por defecto utiliza la API JDBC para realizar operaciones sobre las bases de datos. Es compatible con bases de datos relacionales y noSQL.
- Permite una fácil gestión de transacciones de bases de datos. Permitiendo cambiar la transaccionalidad de cualquier operación sobre la base de datos (cambiar su nivel de propagación, controlar bloqueos...)
- Destaca su ligereza en su proporción de tamaño y funcionalidad respecto a otros frameworks.
- Es compatible con proyectos Maven o Gradle, por lo que permite añadir cualquiera de las numerosas dependencias disponibles de forma muy sencilla.
- Gran integración con otros frameworks y librerías. Por ejemplo, para el mapeo objeto-relacional puede integrarse perfectamente con iBatis o Hibernate.
- Dispone de gran cantidad de herramienta de seguridad dentro de Spring Security para autenticación y autorización, securización de APIs, etc.

- Junto a la base Spring, podría utilizarse **JSF (Java Server Faces)** para el desarrollo de la interfaz. JSF destaca en lo siguiente:
 - Utiliza Facelets como sistema de controlador de vista por defecto.
 - Tiene integración con expresiones del lenguaje Java, por lo que es muy sencillo comunicarse con la lógica de negocio mediante la gestión de “beans”, (haciendo uso de las anotaciones).
 - Ofrece herramientas para la internacionalización de la aplicación: soporte para el uso de diferentes idiomas dentro de la aplicación.
 - Es compatible con numerosas plantillas y cualquier librería que pueda ser utilizada en HTML.
 - Soporta AJAX de forma nativa para realizar peticiones asíncronas.

Utilizar **JavaScript** para el backend, utilizando el framework Express JS, que funciona bajo el **entorno de ejecución Node JS**. Algunas de sus funciones más destacadas son:

- Al utilizar la mayoría de características de NodeJS, es framework muy ligero y rápido.
- Tiene un sistema de rutas muy eficiente y completo.
- Es compatible con varios tipos de motores de plantillas como Pug, Mustache o EJS.
- Hace uso de middlewares, es decir, funciones que pueden ejecutarse antes o después del manejo de una ruta. Permiten el manejo de errores, validaciones de datos, verificar niveles de acceso...etc.
- Hace uso de NPM, el gestor de paquetes de Node JS, que es compatible con numerosas dependencias y librerías.

Desarrollo Front End

En el caso del frontend, disponemos de un amplio espectro de alternativas de bibliotecas o frameworks de diseño:

- **Bootstrap.** Tiene las características siguientes:
 - Tiene estilos por defecto para casi todos los elementos HTML, que se pueden personalizarse.
 - Tiene un diseño responsive (se adapta a diferentes tamaños de pantalla).

- Utiliza el modelo de rejilla (Grid), que permite distribuir los elementos de forma muy sencilla.
- Hace uso de la librería JQuery.
- Existe mucha documentación oficial para todos sus componentes.

- **Materialize CSS.** Se caracteriza por:

- Utilizar las líneas de diseño de Material Design de Google, que aplican a sus elementos.
- Tener un diseño responsive.
- Utilizar la librería JQuery para algunos elementos de la interfaz.
- Ser Open Source.

- **Pure.CSS:**

- Está compuesto por diferentes módulos responsive.
- Ofrece funciones que ayudan al SEO de la aplicación (posicionamiento en los buscadores).
- Es muy ligero. La versión minimizada no llega a los 4KB.

O frameworks de frontend completos como:

- **Angular:**

- Está mantenido por Google.
- Está desarrollado en TypeScript (JavaScript con tipado).
- Está destinado para crear aplicaciones web SPA (aplicaciones de página única) y PWA (Aplicaciones Web Progresivas).
- Es muy universal, permite ejecutar la vista sobre servidores Node JS, .NET o PHP.
- Tiene una sintaxis para las plantillas muy potente
- Es compatible con una gran variedad de IDEs (entornos integrados de desarrollo).

- **React:**

- Está mantenido por Facebook.
- Dispone de un DOM Virtual propio gestionable de forma más eficiente.

- Utiliza componentes, los cuales pueden definirse en diferentes estados del mismo (trabajar con diferentes “instantáneas de cada componente”).
- Se basa en el uso de propiedades (props) que son los atributos de configuración de cada componente.
- Utiliza una sintaxis similar a HTML denominada JSX, que permite escribir código más legible con una experiencia similar a HTML.

- **Svelte:**

- Permite generar aplicaciones más pequeñas debido a la compresión que realiza en un mismo fichero JavaScript.
- Se basa en componentes reactivos.
- Tiene una sintaxis mucho más simple respecto a otros frameworks.
- Permite utilizar estilos CSS por componentes o de forma global.
- Permite el “Hot reloading”, que permite visualizar los cambios que se vayan realizando inmediatamente en el navegador, sin necesidad de recargar la página.

- **Vue:**

- Es un framework completamente modular, por lo tanto más ligero, ya que en un proyecto, se utilizarán sólo los módulos que se necesiten.
- Ofrece propiedades reactivas, es decir si cambia el valor de una variable en una parte de la página, se cambia de forma automática en todas partes.
- Se basa en componentes web.
- Incluye su propio DOM virtual.

O simplemente utilizar VanillaJS, es decir, el desarrollo del código JavaScript sin hacer uso de frameworks.

Representación y uso de mapas

Por la naturaleza del proyecto, en relación al uso de mapas y representación de los mismos, es necesario utilizar APIs u otras herramientas de terceros que nos puedan brindar la información que necesitamos. Para ello, tenemos alternativas como:

- APIs de la Google Maps Platform (incluyendo Maps, Routes y Places), que permiten utilizar sus mapas, su sistema de rutas y su base de datos de

lugares y sitios. El uso de estas APIs es de pago (con posibilidad de realizar una prueba gratuita).

- Mapas Open Source de OpenStreetMaps que incluyen únicamente la cartografía.

Para representar la información de estas APIs o mapas, necesitamos hacer uso de librerías para GIS (Sistemas de información geográfica). Podemos utilizar:

- **OpenLayers:**

- Se basa en HTML5, WebGL y CSS3.
- Permite crear mapas web, e integrarle capas de OpenStreetMaps, Bing, Google Maps...etc.
- Permite incorporar otras librerías para mapas.
- Es Open Source.

- **Leaflet:**

- Es una librería muy ligera, no llega a los 33 KB.
- Permite personalizar todos sus elementos (mapa, marcadores, botones...).
- Dispone de numeroso plugins para darle más funcionalidades: para usar pantalla completa, medir distancias o indicar rutas entre otros.

- **Carto.js:**

- Dispone de diversas API pertenecientes de la plataforma CARTO.
- Permite realizar capas de mapas muy variadas que pueden ser integradas en otras librerías.
- Nos permite crear los mapas online sin necesidad de escribir código.
- El uso de algunas de sus APIs requiere una suscripción de pago.

Persistencia de datos

Por último, debemos optar por algún sistema de persistencia de datos, en este caso para los sitios guardados, usuarios o cualquier información adicional que la aplicación requiera. Para ello se puede hacer uso de una base de datos, existen varias alternativas como:

- **MariaDB (derivado libre de MySQL):**

- Sistema Gestor de Bases de Datos (SGBD) relacionales.

- Es software libre bajo una licencia GNU GPL gratuita.
- Es compatible con todas las librerías, conectores y aplicaciones de MySQL.

- **MongoDB:**

- Base de datos NoSQL basado en documentos.
- Es opensource y gratuito.
- Toda la información se almacena en JSON.
- Tiene un alto rendimiento y disponibilidad.
- Ofrece escalado horizontal: pueden distribuirse los datos por diferentes cluster de máquinas.

- **Oracle Database:**

- SGBD relacional.
- Software privativo con un elevado coste de licencia. (Oracle)
- Es multiplataforma.
- Tiene soporte para transacciones .
- Tiene una alta disponibilidad y rendimiento en el ámbito empresarial.
- Tiene un sistema de control de acceso avanzado.
- Hace uso de PL/SQL (lenguaje de programación) destinado a la creación de procedimientos almacenados o funciones.

- **Microsoft SQL Server:**

- SGBD relacional.
- Software privativo con un elevado coste de licencia.
- Incluye un potente entorno gráfico para su administración: pueden ejecutarse comandos de DDL (lenguaje de definición de datos) y DML (lenguaje de manipulación de datos) de forma gráfica.
- Permite trabajar en modo cliente-servidor.
- Tiene soporte para transacciones.
- Hace uso de Transact-SQL para creación de procedimientos almacenados, funciones, triggers...

4 - Justificación de la elección

Una vez presentadas las diferentes alternativas que podemos utilizar para poder llevar a cabo el desarrollo del proyecto, se han tomado las siguientes decisiones:

Desarrollo Back End

Para desarrollar el backend de la aplicación se va a utilizar el framework Symfony en su versión 5. Los motivos por los que va a ser utilizado en este proyecto son:

- Utiliza el sistema de gestión de paquetes Composer, que es compatible con numerosas librerías disponibles (como las disponibles en <https://packagist.org/>). Que además incluyen las necesarias para hacer funcionar la API de Google Maps que va a ser utilizada.
- Tiene un entorno de pruebas muy completo, que permite localizar errores de forma muy sencilla durante el desarrollo durante su despliegue de forma local.
- Dispone de componentes de autenticación para generar logins de forma sencilla (y gestionarlos según el rol, la página en la que se encuentra...).
- Personalmente, he trabajado más con este framework en clase que sobre el resto, por lo que me puedo desenvolver con más soltura.

Desarrollo Front End

En el frontend se utilizará Bootstrap 4 como framework de diseño, por su diseño responsive integrado, por la gran cantidad de información que hay sobre él (al ser tan utilizado) y porque puedo personalizar y crear estilos nuevos fácilmente para adaptarlos a las líneas de diseño que se quieren seguir en la aplicación.

Para realizar peticiones asíncronas y aportar funcionalidad a la aplicación se va a utilizar VanillaJS (JavaScript puro sin uso de frameworks) y JQuery (requerido por Bootstrap en algunas situaciones).

De forma complementaria, se utilizarán librerías como “Micromodal.js” para generar ventanas modales simples y personalizables, y la iconografía de “FontAwesome”.

Representación y uso de mapas

Para obtener la información de los lugares cercanos se va a hacer uso de la API de Google Maps Places, que cumple exactamente con los criterios del proyecto: mostrar información sobre los lugares cercanos (ubicación, datos de dirección,

horarios...) pasándole información como nuestra ubicación y un criterio de búsqueda.

La herramienta que se encargará de plasmar esa información obtenida en un mapa será la librería Leaflet. He optado por esta librería por su gran personalización (permite cambiar los iconos de los marcadores, incorporar las capas de mapas que deseemos, añadir elementos gráficos, cambiar la escala y zoom del mapa...etc), y por su ligereza respecto otras librerías.

En dicha librería se utilizarán dos capas de mapas. La capa predeterminada hará uso los mapas de OpenStreetMaps tipo “callejero” para que de esta forma se limiten un poco las peticiones a la API de Google Maps que ya es usada en otros puntos. Además, el uso de estos tipos de mapas de forma predeterminada (tipo “callejero”) garantiza que el tráfico de red sea más reducido (respecto de un mapa de vista de satélite), lo que hace que la aplicación sea más rápida y evite (si el usuario dispone de él) un consumo excesivo en un plan de datos.

Como alternativa, si el usuario lo prefiere se podrán utilizarán los mapas de Google Maps de vista de satélite, ya que estos mapas concretamente hacen uso de imágenes de satélite más actuales y de mayor calidad.

Persistencia de datos

Para finalizar, para la persistencia de los datos, al utilizarse Symfony, éste hace uso de Doctrine como ORM, que realizará operaciones sobre la base de datos, en este caso MariaDB. Se va a utilizar MariaDB como SGBD por las siguientes razones:

- Su licencia es gratuita, y no tiene coste alguno.
- Para el desarrollo de este proyecto, la carga que soporta es suficiente.
- Es compatible con Symfony, framework que va a ser utilizar.
- Soporta relaciones, necesarias para poder desarrollar el proyecto.
- Al utilizar SQL como lenguaje, es sencillo migrar el proyecto a otros SGBD más potentes como SQL Server u Oracle si se requiere en un futuro.

5 - Recursos necesarios

5.1 - Hardware

En este proyecto, serán utilizados como hardware implicado en el desarrollo, dos equipos que presentan las siguientes características:

- Un equipo portátil, marca Apple. Con un procesador Intel Core i5 5250U y 8 Gigabytes de memoria DDR3. Con sistema operativo macOS Catalina 10.15.

- Un equipo de sobremesa, marca MSI. Con un procesador Intel Core i5 8600K y 16 Gigabytes de memoria DDR4. Con doble sistema operativo: Microsoft Windows 10 y Linux Mint 19.3 (basado en Debian).

La causa de utilizar dos equipos, además de permitir versatilidad en el desarrollo del proyecto, es debido a que al utilizar equipos con sistemas operativos distintos (los tres principales del mercado) y formatos diferentes de pantalla, el posterior testeo de la aplicación puede ser más completo, y puede ayudar a localizar posibles fallos de compatibilidad.

Además de ambos equipos, para el ámbito móvil se utilizarán dos smartphones:

- Un dispositivo marca Xiaomi, con pantalla de 5,15 pulgadas, procesador Qualcomm Snapdragon 835 y 6 Gigabytes de memoria LPDDR4X. Con sistema operativo Android 9.0 Pie.
- Un dispositivo marca Apple, con una pantalla de 4 pulgadas, y procesador Apple A7. Con sistema operativo iOS 12.

De forma complementaria, se podrán utilizar terminales disponibles de familiares y conocidos ajenos al proyecto para llevar a cabo también pruebas de compatibilidad.

5.2 - Software

La herramienta principal de desarrollo del proyecto es el entorno de desarrollo integrado Microsoft Visual Studio debido a su compatibilidad con una inmensa de lenguajes de programación, librerías, extensiones ...etc gracias a su extenso “marketplace” de extensiones. Además de ser multiplataforma.

Se utilizarán los siguientes sistemas operativos durante el desarrollo: Windows 10, macOS Catalina 10.15 y Linux Mint 19.3 como sistemas operativos de escritorio, y Android e iOS para smartphones.

De forma complementaria se utilizarán cuatro navegadores para la visualización, depuración y testeo de la aplicación: Google Chrome, Mozilla Firefox, Apple Safari y Microsoft Edge (versión Chromium).

Para el control de versiones, se hará uso de la herramienta Git desde el terminal nativo de Linux (y terminal propio desde Windows) para poder subir y bajar cambios desde un repositorio.

5.3 - Servicios online

Para el desarrollo del proyecto, se han utilizado las siguientes herramientas online:

- **Github**, que hace uso del control de versiones de Git, el cual permite subir y bajar cambios remotamente. De esa forma siempre existe una copia del

proyecto en la que pueden verse de forma muy visual y sencilla todos los cambios realizados.

- **Documentos de Google (G Suite)**, que ha permitido elaborar la memoria de este proyecto, pudiendo acceder a ella desde cualquier dispositivo. Además ofrece funciones muy útiles, como el autoguardado automático, un historial de versiones, y el trabajo colaborativo, que ha permitido a los tutores del proyecto, realizar un seguimiento del mismo a tiempo real.
- **Para el despliegue** se utilizará la plataforma gratuita **Heroku**, que permite desplegar proyectos que utilicen diferentes lenguajes o frameworks, a cambio de utilizar dominios gratuitos cedidos por ellos. Además, soporta la sincronización con proyectos de Github, lugar donde se encuentra alojado nuestro proyecto.

6 - Desarrollo del proyecto

6.1 - Modelado de la información

Basándonos en los objetivos del proyecto, podemos enunciar el siguiente “dominio del problema”:

A través de nuestra aplicación, un usuario registrado se autentica. Al autenticarse, se le concede acceso a la aplicación completa. Mediante la aplicación, el usuario puede añadir a sus sitios favoritos, lugares que haya buscado con anterioridad. El usuario puede tener varios sitios favoritos, y cada sitio favorito puede serlo de otro usuario.

Tomando lo anterior, podemos deducir las entidades que implementaremos más adelante en el código mediante clases:

La entidad Usuario

La entidad usuario representa al usuario que utiliza la aplicación. Cada usuario dispondrá de los siguientes atributos:

- **Id**: necesitará de un identificador único dentro de la aplicación para poder distinguirse de otro usuario de forma interna.
- **Email**: es necesario para conocer de qué usuario se trata. Sería un valor definido por el usuario, que no podrá repetirse.
- **Password (contraseña)**: junto al email, sería la forma con la que el usuario podrá autenticarse dentro de la aplicación.
- **Roles**: indica de qué tipo de usuario se trata y sus privilegios dentro de la aplicación.

- **Lista de favoritos:** contiene una lista con los sitios favoritos del usuario. Se relaciona con la entidad Favorito.

La entidad Favorite

La entidad favorito representa un sitio del cual queremos guardar su información. Los favoritos tendrán los siguientes atributos:

- **Id:** identificador único dentro de la aplicación, que distingue a un sitio favorito de otro.
- **Id de sitio:** es el identificador real del sitio. Distingue el sitio (lugar físico) en sí de otros. Puede ser utilizado por el proveedor de sitios para saber de qué sitio se trata. Es único.
- **Nombre:** nombre real del sitio favorito.
- **Atributos Latitud y Longitud:** ambos atributos juntos definen la ubicación exacta del sitio en un mapa.
- **Icono:** un símbolo o logotipo representativo sobre el sitio favorito.
- **Dirección:** dirección descrita del sitio dentro de una ciudad o pueblo: calle, número ...
- **Lista de usuarios:** contiene la lista de aquellos usuarios que tienen a ese sitio en sus favoritos. Se relaciona con la entidad Usuario.

6.2 - Programación

El código fuente de este proyecto al que se hace referencia a continuación, puede encontrarlo adjuntado junto a este documento o en el siguiente repositorio de GitHub: <https://github.com/mfortea/SiteMe>

6.2.1 - Guía de estilo de codificación

Durante el desarrollo, se han llevado a cabo unas pautas de codificación, para de esta forma, tener un código limpio, consistente y ordenado.

En este proyecto se han utilizado diferentes lenguajes y motores de plantillas, por lo que se especifica una guía de estilo para cada uno de ellos:

Código PHP

Para el lenguaje PHP, se ha seguido la guía de estilo oficial PSR-2 (que extiende PSR-1), utilizando el formateador de código “phpfmt - PHP Formatter” del desarrollador “kokorin” disponible en el Marketplace de Visual Studio Code.

Algunas de las reglas más destacadas de la guía de estilo PSR-2 son:

- **El código debe utilizar 4 espacios para la indentación, no tabulación.**
- **No debe existir un límite estricto para la longitud de una línea. El límite permisivo debe ser de 120 caracteres como máximo, pero se recomienda que sea de 80 caracteres.**
- **La visibilidad debe de ser declarada en todas las clases y métodos.**
- **Los nombre de las clases deben declararse en “StudlyCaps” y los de los métodos en “camelCase”.**
- **Modificadores:**
 - Los modificadores “abstract” y “final” deben declararse antes de la visibilidad.
 - El modificador “static” debe declararse después de la visibilidad.
- **Para las declaraciones de “Namespaces” y “Use” (si están presentes):**
 - Debe existir una línea en blanco detrás de la declaración de un “namespace”.
 - Las declaraciones “Use” deben ir tras los “namespaces” separados por una línea en blanco.
 - Sólo debe existir una palabra “Use” para cada declaración.
 - Después de un bloque de declaraciones “Use” debe existir un espacio en blanco.
- **Llaves (en clases y métodos):**
 - Las llaves de apertura deben comenzar en una línea nueva tras la declaración del método o la clase.
 - Las llaves de cierre deben ir en una nueva línea tras el cuerpo del código del método o la clase.
- **Llaves (en estructuras de control):**
 - Las llaves de apertura deben estar en la misma línea.
 - Las llaves de cierre deben ir en una nueva línea tras el cuerpo.
- **Paréntesis:**

- Los paréntesis de apertura de los métodos no deben llevar un espacio antes, y los de cierre no pueden llevar un espacio después.
- En cambio, en las estructuras de control, los paréntesis de apertura pueden llevar espacios antes de ellos pero no después, y los paréntesis de cierre no deben tener espacios antes.

- **Atributos de clase:**

- La visibilidad debe ser declarada en todos los atributos.
- No se debe utilizar la palabra “var” para declarar un atributo de clase.
- No debe haber más de un atributo de clase por declaración.
- No se recomienda añadir un guión bajo como prefijo para indicar que el atributo es “protected” o “private”.
- Las constantes deben ser escritas en mayúsculas, con guiones bajos como separador.

Ejemplo:

```
<?php
namespace Vendor\Paquete;

use MiInterfaz;
use ClaseB as Cb;
use OtroVendor\OtroPaquete\ClaseC;

class ClaseD extends Cb implements MiInterfaz
{
    public $atributo1 = null;
    protected $atributo2;
    public const MI_CONSTANTE = "SiteMe";

    public function miMetodo($a, $b = null)
    {
        if ($a === $b) {
            ejemplo();
        } elseif ($a > $b) {
            $foo->ejemplo($arg1);
        } else {
            ClaseC::bar($arg2, $arg3);
        }
    }
}
```

```
        }
    }

    final public static function ejemplo()
    {
        // Cuerpo del método
    }
}
```

Código JavaScript

Para el código escrito en JavaScript se ha utilizado el formateador de código “Prettier” disponible desde el Marketplace de Visual Studio Code.

Dicho formateador, no sigue una guía de estilo concreta, por lo que a partir de su uso, podemos definir una guía de estilo aproximada, aportando reglas propias. Algunas de las reglas más relevantes son las siguientes:

- **Todas las sentencias acaban en punto y coma.**
- **La indentación se realiza con una tabulación.**
- **Los nombres de los métodos se escriben en “camelCase”.**
- **Todas las variables se declaran con la palabra “var”.**
- **Llaves (clases y métodos):**
 - Las llaves de apertura deben estar en la misma línea.
 - Las llaves de cierre deben ir en una nueva línea tras el cuerpo.
- **Paréntesis:**
 - Los paréntesis de apertura de los métodos no deben llevar un espacio antes.
 - En cambio, en las estructuras de control, los paréntesis de apertura deben llevar espacios antes de ellos pero no después, y los paréntesis de cierre no deben tener espacios antes pero sí después.

Ejemplo:

```
var variable = "SiteMe";

function miFuncion(texto) {
```

```
if (variable === "SiteMe") {  
    alert("Bienvenido");  
}  
}
```

Plantillas Twig

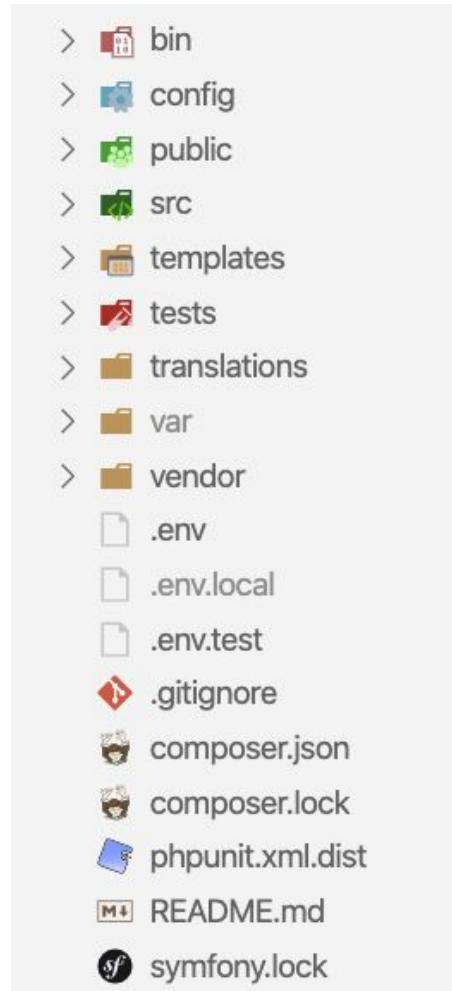
Las plantillas utilizadas en la aplicación, hacen uso de el motor de plantillas Twig, que se combina con HTML. Para formatear dichas plantillas, se ha utilizado la extensión de VS Code “Twig Language 2”. Al no disponer de guía de estilo, se describen algunas de las reglas que sigue este formateador:

- **Las declaraciones de bloques de apertura y cierre se escriben en líneas diferentes.**
- **Las declaraciones de bloques de apertura y cierre que no tienen contenido, se escriben en la misma línea.**
- **Las indentaciones se realizan con una tabulación.**
- **Las declaraciones apertura y cierre de los bucles y estructuras de control van en líneas distintas.**

```
{% block form_errors %}  
    {% if errors|length > 0 %}  
        {% for error in errors %}  
            <p>{{ error.message }}</p>  
        {% endfor %}  
    {% endif %}  
{% endblock form_errors %}  
  
{% block otroBloque %}{% endblock %}
```

6.2.2 - Organización física del código

El proyecto ha sido desarrollado utilizando el framework Symfony, por lo que la mayor parte de su estructura de directorios ha sido generada automáticamente por este framework utilizando sus propios criterios.



Vista general de la raíz del proyecto

De forma más detallada, las funciones de los directorios más relevantes y sus ficheros son la siguientes:

Directorio /bin



Este directorio, contiene los binarios de Symfony, en este caso el ejecutable de la consola de Symfony y referencias a dependencias como las de phpunit.

Directorio /config



Dentro de este directorio, se encuentran todos los ficheros de configuración del proyecto. Symfony suele utilizar ficheros con extensión .yaml para esta función.

Algunos de los ficheros más relevantes de este directorio son:

- Fichero `routes.yaml` : fichero donde se definen las rutas de los endpoints, que llaman a métodos del controlador.

```
index:
  path: /
  controller: App\Controller\MainController::index

registro:
```

```

path: /registro
controller: App\Controller>MainController::registro

busqueda:
    path: /busqueda
    controller: App\Controller>MainController::busqueda

sitios:
    path: /sitios
    controller: App\Controller>MainController::sitios

ajustes:
    path: /ajustes
    controller: App\Controller>MainController::ajustes

# [ ... ]

```

- Fichero `security.yaml` : fichero que contiene configuraciones sobre los componentes de seguridad de Symfony, como la entidad usuario utilizada, el tipo de algoritmo de codificación, el autenticador utilizado para el login y la restricciones de los endpoints por roles entre otras.

```

security:
    encoders:
        App\Entity\Usuario:
            algorithm: auto

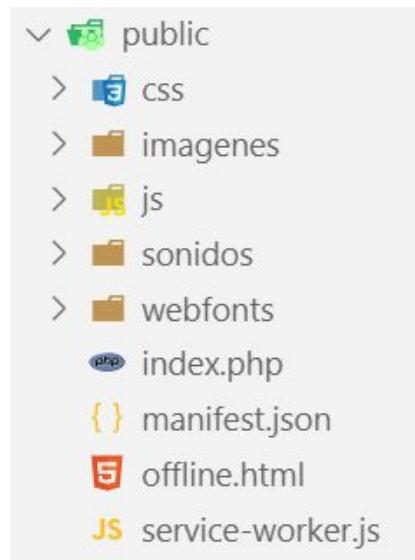
    providers:
        app_user_provider:
            entity:
                class: App\Entity\Usuario
                property: email
    firewalls:
        dev:
            pattern: ^/(_profiler|wdt)|css|images|js/
            security: false
        main:
            anonymous: lazy
            provider: app_user_provider
            guard:
                authenticators:
                    - App\Security\LoginFormAuthenticator
            logout:
                path: app_logout
                target: app_login

    access_control:
        - { path: ^/busqueda, roles: "ROLE_USER" }
        - { path: ^/sitios, roles: "ROLE_USER" }
        - { path: ^/ajustes, roles: "ROLE_USER" }

# [ ... ]

```

Directorio /public



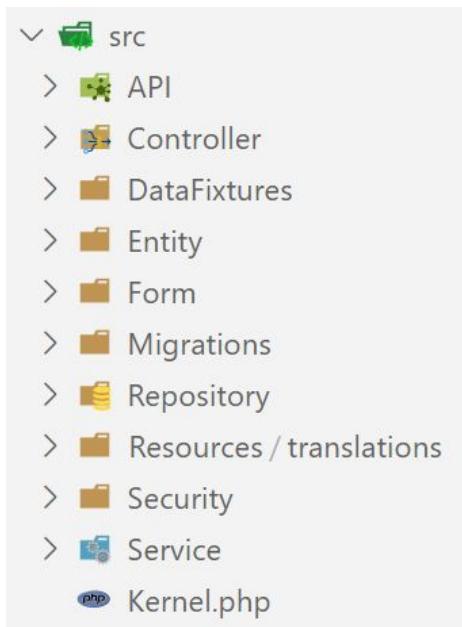
Este directorio contiene todos los “archivos web” que utiliza el proyecto: hojas de estilo CSS, ficheros JavaScript y “assets” como imágenes y sonidos.

En el caso de nuestro proyecto, se han organizado los elementos en carpetas según el tipo de archivo:

- **/css** para todos los ficheros de hojas de estilo.
- **/imagenes** para los ficheros de imágenes (png principalmente).
- **/js** para todos los ficheros JavaScript.
- **/webfonts** para las fuentes utilizadas por FontAwesome.

Además existen ficheros complementarios como el manifest.json, service-worker.js u offline.html en la raíz de /public, que en conjunto son utilizados para definir nuestro proyecto como una Aplicación Web Progresiva (PWA).

Directorio /src



En este directorio es donde se almacena todo el código de nuestra aplicación.

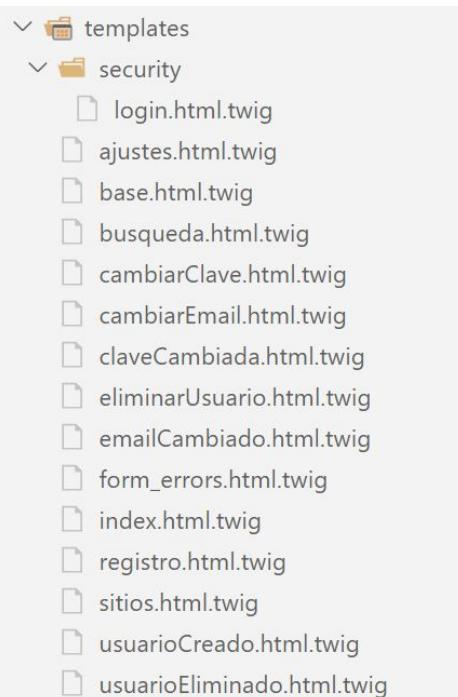
Symfony hace uso del patrón de diseño Modelo-Vista-Controlador, por lo que dicho patrón se ve reflejado en la estructura de directorios, donde “**Entity**” sería el Modelo, y “**Controller**” el Controlador.

La función del resto de directorios es la siguiente:

- **/API** es un directorio de utilidades creado concretamente para este proyecto, donde se incluyen clases con métodos que realizan las llamadas a la API de Google Maps o un “objeto simulado” que contiene una respuesta cacheada de la API.
- **/DataFixtures** incluye clases que definen objetos que pueden ser añadidos a una base de datos directamente, normalmente para cargar datos iniciales o de prueba.
- **/Form** incluye clases y métodos necesarios para la generación de distintos formularios como “cambiar clave”, “cambiar email” o “eliminar cuenta.”
- **/Migrations** es un directorio generado por Doctrine donde se lleva un registro de los cambios realizados en la última migración en la base de datos, así como las acciones que deben realizarse en cada migración.
- **/Repository** es otro directorio generado por Doctrine, donde se definen las clases que se utilizarán para crear instancias de las diferentes entidades de nuestro proyecto.

- **/Resources/translations** este directorio contiene los ficheros relacionados con la traducción de mensajes, en este caso, la traducción a español de los mensajes mostrados en la ventana de login.
- **/Security** contiene clases necesarias utilizadas por el componente Symfony Security. Es este caso, incluye una clase que define el autenticador del login.
- **/Service** es el directorio donde se almacenan las clases que se han utilizado como servicios, siguiendo el patrón DAO (Data Access Object), que hace uso de la inyección de dependencias.

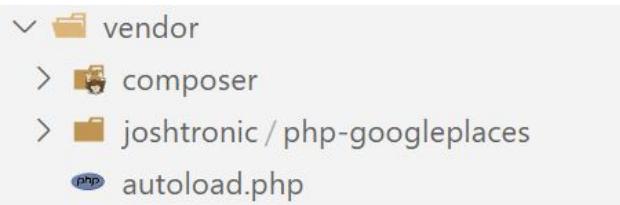
Directorio /templates



Es el directorio que contiene todas las plantillas utilizadas en la aplicación. Todos los ficheros de este directorio tienen el formato .twig que pertenece al motor de plantillas Twig que utiliza Symfony. De forma general, la gran mayoría heredan de base.html.twig.

Este directorio corresponde a la **Vista** del Modelo-Vista-Controlador mencionado anteriormente.

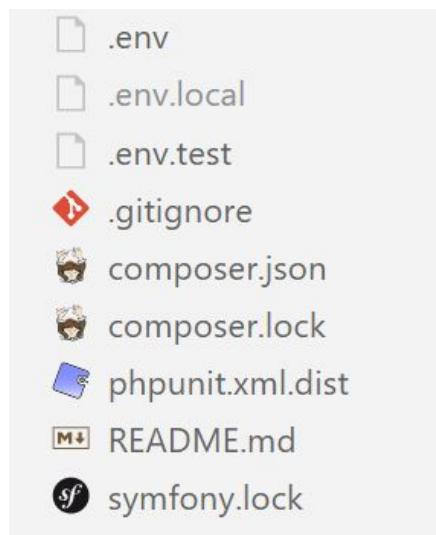
Directorio /vendor



Es el directorio que contiene todas las dependencias de Composer que hayamos indicado en el fichero composer.json de la raíz del proyecto.

Incluye ficheros como autoload.php que se encarga de gestionar las dependencias que se deben de cargar automáticamente al desplegar el proyecto.

Ficheros del directorio raíz /



En la raíz del proyecto, existen diferentes ficheros que tienen gran relevancia dentro del proyecto, como son:

- **Los ficheros .env** : son los ficheros que almacenan las variables de entorno que utiliza Symfony de forma global. Se distingue su extensión según su uso, .test (para un entorno de pruebas), .local (para un espacio de trabajo concreto)...

Las variables de entorno utilizadas en este proyecto son las credenciales de la base de datos y la API Key de Google Maps.

- **Fichero composer.json** : es el fichero donde se definen los requisitos del proyecto: la versión de PHP utilizada, las dependencias o las rutas donde deben cargarse, entre otros. Con dicho fichero se indica a composer las

dependencias que debe descargar o actualizar para que éstas funcionen en el proyecto.

- **Fichero .gitignore:** es un fichero que es interpretado por el control de versiones Git, utilizado en este proyecto, que indica que ficheros o carpetas no deben subirse al repositorio donde se aloja el proyecto.

Algunas de esas carpetas son /vendor y /var, que pueden ser generadas más tarde por Composer.

- **Fichero README.md :** es un fichero en formato Markdown, que suele incluirse en los proyectos para indicar ciertas instrucciones básicas y detalles sobre un proyecto. Es utilizado por el sitio web GitHub para mostrar las descripciones de sus repositorios.

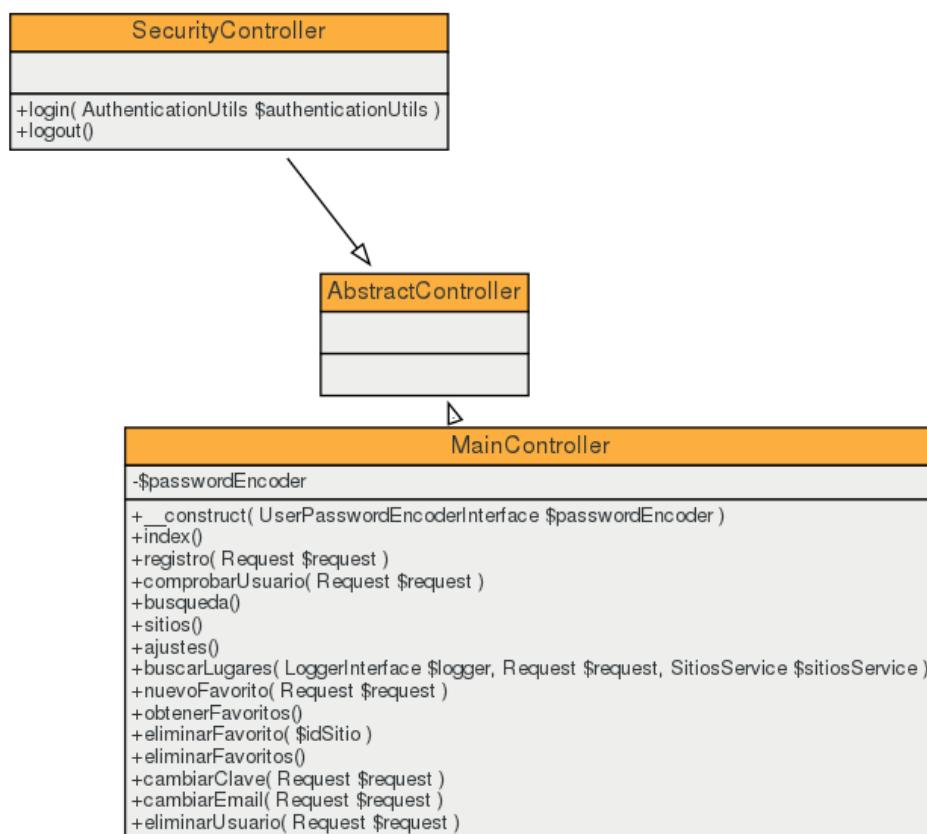
6.2.3 - Clases y funciones

Toda la lógica de la aplicación la encontramos en las clases y funciones. Las principales y más relevantes utilizadas son las siguientes:

Parte servidor (PHP)

Clases del controlador y sus funciones

Dentro de la aplicación, encontramos dos clases utilizadas como controlador:



- **Clase MainController**

Esta clase contiene los métodos principales del Controlador. La mayoría de ellos se referencian desde el fichero “routes.yaml”. Es una clase que extiende la clase AbstractController.

Únicamente existe un atributo en esta clase, que es inicializado en el constructor: **\$passwordEncoder**. Dentro de este atributo, desde el constructor, se almacena una instancia de la interfaz “UserPasswordEncoderInterface”, necesaria en alguno de los métodos del controlador.

Los métodos que podemos encontrar son los siguientes:

- **index()**: este método se encarga de renderizar la página de inicio (index) a través de su plantilla twig “index.html.twig”.
- **registro(Request \$request)**: método que se encarga de generar el formulario de registro de nuevo usuario (“registro.html.twig”). Tiene como parámetro una variable de tipo Request, que recoge la información que el usuario ha introducido en la web.

En primer lugar se comprueba si el formulario ha sido enviado, si es así, se comprueba su validez (que el correo no esté en uso o que las contraseñas coinciden), y si es válido, renderiza la plantilla “usuarioCreado.html.twig”.

En el caso de que no sea válido, se le muestra un mensaje al usuario indicando de que no se ha podido crear el usuario.

- **comprobarUsuario(Request \$request)**: con este método, se comprueba que el email introducido por el usuario no esté en uso. Para ello, recoge la petición del usuario y se compara con los emails almacenados en la base de datos y se informa al usuario.
- **busqueda()**: este método renderiza la página Búsqueda utilizando la plantilla “busqueda.html.twig”.
- **sitios()**: es el método encargado de renderizar la página de “Mis sitios” a través de la plantilla “sitios.html.twig”.
- **ajustes()**: renderiza la página de Ajustes con la plantilla “ajustes.html.twig”.
- **buscarLugares(LoggerInterface \$logger, Request \$request, SitiosService \$sitiosService)**: este método se encarga de buscar los sitios más cercano al usuario, recibiendo como parámetro su ubicación (latitud y longitud), su criterio de búsqueda, y un radio en metros.

En primer lugar, se crea una instancia del tipo de fuente de datos que se va a utilizar y se le pasa al objeto del servicio SitiosService. Se

Llama al método `getSitios()`, pasando como parámetros los recogidos de la petición del usuario, y se almacena su respuesta en una variable.

A continuación, se comprueba si alguno de los sitios encontrados ya se encuentra en la lista de favoritos del usuario actual, para ello, se recorre el resultado, y se comparan con los sitios favoritos del usuario; si se encuentra un sitio ya almacenado por el usuario, se cambia el valor de su propiedad “favorito” a true.

Finalmente se devuelve el objeto de la respuesta como JSON al usuario.

- **nuevoFavorito(Request \$request)**: con este método se añade un sitio como favorito del usuario.

Este método recoge los datos de la petición del usuario correspondientes con el sitio (id, nombre, latitud, longitud...) y crea un nuevo objeto con ellos. Comprueba que ese objeto no se encuentre ya en su lista comparando sus “`idSitio`”. Si el sitio no está entre sus favoritos lo añade, y si no, se le informa al usuario de que ya se encuentra entre sus favoritos.

- **obtenerFavoritos()**: es el método encargado de recoger todos los sitios favoritos del usuario y devolverlos.

En primer lugar, se obtiene la lista completa de los favoritos, y se almacena en una variable. De dicha lista se comprueba que haya más de un sitio, si no es así, se le devuelve una respuesta al usuario, que indica que se encuentra vacía.

Si hay más de un sitio, se recorre la lista y se almacena dentro de un nuevo objeto, que se le devuelve al usuario en formato JSON.

- **eliminarFavorito(\$idSitio)**: este método permite eliminar un sitio concreto de la lista de favoritos del usuario. Para ello se recoge el parámetro “`idSitio`” que manda el usuario. A continuación se comprueba que el sitio se encuentra en los favoritos del usuario, y si es así, se elimina; en caso contrario, se le devuelve una respuesta al usuario, indicando de que el id proporcionado no existe.
- **eliminarFavoritos()**: al llamar a este método, se eliminan toda la lista de sitios favoritos de un usuario. Simplemente, se obtiene la lista de todos los sitios favoritos del usuario, y recorre de forma de que se eliminan todos.
- **cambiarClave(Request \$request)**: este método es el encargado de generar el formulario (“`cambiarClave.html.twig`”) para que el usuario pueda cambiar su clave.

El usuario pasa como parámetros su clave actual y la nueva clave. Si el formulario es correcto, se cambia la contraseña del usuario por la

nueva, y se renderiza la página “claveCambiada.html.twig”. Si ocurre algún fallo durante el cambio, se le informa al usuario.

- **cambiarEmail(Request \$request)**: este método permite al usuario cambiar su email. En primer lugar, se renderiza el formulario (“cambiarEmail.html.twig”). El usuario pasa como parámetros el nuevo email que quiere cambiar, y su contraseña. Se comprueba que la contraseña sea correcta, y que el email no esté en uso, y si es correcto, se cambia el email del usuario, y se renderiza la página de “emailCambiado.html.twig”.
- **eliminarUsuario(Request \$request)**: y el último método de este controlador, es el encargado de dar de baja al usuario. Se genera un formulario de confirmación al usuario (“eliminarUsuario.html.twig”), donde debe escribir su contraseña como método de confirmación. Si la contraseña es correcta, se elimina al usuario, y se renderiza la página “usuarioEliminado.html.twig”. Si este proceso falla, se le muestra un mensaje al usuario indicándose.

- **Clase SecurityController**

Esta clase que actúa como controlador, es creada por el sistema de seguridad Symfony Controller, a través de su asistente. Al ser un controlador, extiende a la clase AbstractController. La principal función de esta clase es contener los métodos necesarios para llevar a cabo las operaciones de autenticación (iniciar sesión y cerrar sesión). Los métodos que contiene son los siguientes:

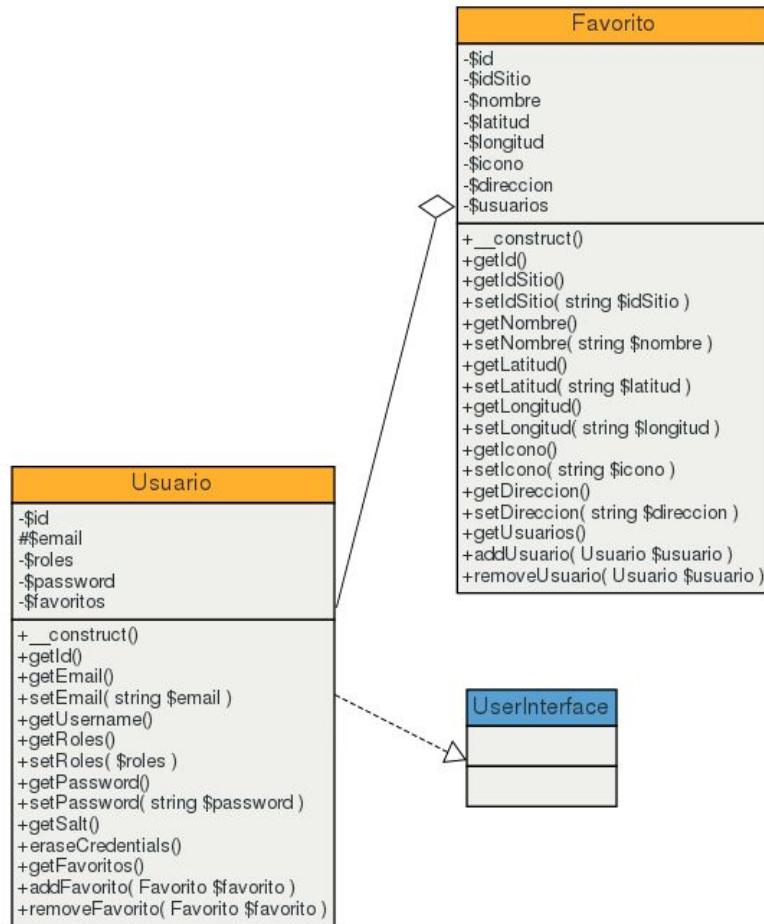
- **Método login(AuthenticationUtils \$authenticationUtils)**: es el método encargado de generar el formulario y la lógica de la página de Login. Se renderiza la plantilla “login.html.twig” que contiene el formulario, y si los datos son correctos (email y contraseña), redirige al usuario ruta correspondiente de la página “Búsqueda”. Si el formulario no es correcto, objeto \$authenticationUtils de la clase AuthenticationUtils, comprueba el tipo de fallo producido, y se lo notifica al usuario.

En las anotaciones de este método se indican el “endpoint” y nombre de la ruta correspondiente, en este caso “/login” y “app_login”.

- **Método logout()**: al crear este método, mediante sus anotaciones se le indica el endpoint (“/logout”) y el nombre (“app_logout”) de la ruta correspondiente que hace que el usuario cierre su sesión y vuelva al Login. Ésta última lógica no se define en este método, si no en el fichero “security.yaml” nombrado en el apartado de la “Organización física del código”.

Clases de entidades (Modelo)

Para la codificación de las entidades desarrolladas durante el modelado de la información, se han creado las siguientes clases modelo:



- **Clase Usuario**

Esta clase representa a cada usuario dentro de la aplicación. Es utilizada por Symfony para la autenticación de nuevos usuarios. Tiene los siguientes atributos:

- **\$id:** es un número entero que identifica al usuario dentro de la aplicación. Es único y no puede repetirse.
- **\$email:** cadena de texto que representa el correo electrónico del usuario. Es único.
- **\$roles:** contiene una lista de los roles que puede tener el usuario. En este caso, todo usuario autenticado, tendrá el rol ROLE_USER para poder acceder a las páginas restringidas de la aplicación.

- **\$password:** es una cadena de texto encriptada que representa la contraseña del usuario, junto al email, se utiliza para que el usuario inicie sesión dentro de la aplicación.
- **\$favoritos:** contiene una colección de objetos de la clase “Favorito”. Representan los sitios favoritos de un usuario. La relación entre esta entidad y “Favorito” es de “Many to Many” (muchos a muchos).

Excluyendo el constructor y los getters y setters de todos los atributos, podemos distinguir los siguientes métodos:

- **Método getUsername():** este método devuelve el atributo del usuario que se haya configurado como un “identificador visual” (lo que en algunas ocasiones es el nombre de usuario), en este caso sería el atributo email.
 - **Método getSalt():** es el método encargado de devolver la “sal” de la contraseña. Es decir, unos bits aleatorios que se añaden a la misma para que de esta manera, aunque ésta se descifre, no se pueda obtener la contraseña correcta.
 - **Método eraseCredentials():** habilitando este método (al descomentar las líneas generadas por defecto), se permite eliminar cualquier dato temporal sensible del usuario. Este método, junto a los dos anteriores, provienen de la interfaz “UserInterface”.
 - **Método addFavorito(Favorito \$favorito):** este método, añade (si el sitio no está) a la colección “favoritos” un nuevo sitio, y añade el usuario actual a la colección “usuarios” del sitio favorito en cuestión.
 - **Método removeFavorito(Favorito \$favorito):** se encarga de eliminar un sitio de la colección “favoritos” (si está en la lista), y de eliminar al usuario actual de la colección “usuarios” del sitio al que nos referimos.
- **Clase Favorito**

Con esta clase, se representa un sitio del cual queremos almacenar su información. Contiene los atributos de clase siguientes:

- **\$id:** contiene un número entero que Symfony utiliza para distinguir un sitio de otro dentro de la aplicación.
- **\$idSitio:** es una cadena de texto, que representa el identificador real del sitio. Es el número de identificación que utiliza la API de Google Maps para distinguir un sitio de otro.
- **\$nombre:** es una cadena de texto que representa el nombre del sitio (o una descripción representativa).

- **\$latitud:** cadena de texto que representa la latitud del sitio en un mapa.
- **\$longitud:** es una cadena de texto que representa la longitud del sitio en un mapa. Junto a la latitud, se utilizan para geolocalizar el sitio en un mapa.
- **\$icono:** es una cadena de texto que representa la URL de una imagen representativa del sitio (icono o logotipo).
- **\$direccion:** contiene una cadena de texto con los datos completos de la dirección del sitio.
- **\$usuarios:** contiene una colección de usuarios que consideran el sitio como favorito.

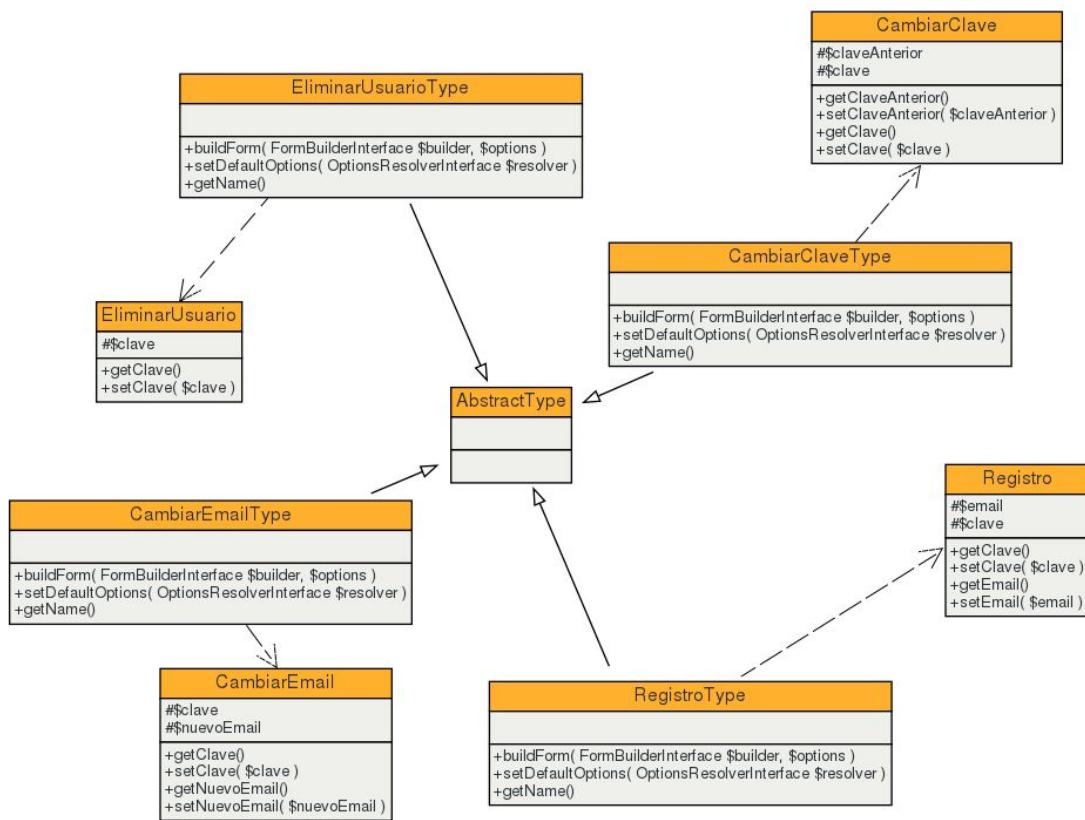
Esta clase, además del constructor y getters y setters correspondientes, incluye los siguientes métodos:

- **Método addUsuario(Usuario \$usuario):** este método permite añadir un nuevo usuario a la colección “usuarios” si éste no existe. Además, añade el actual favorito a la colección “favoritos” del usuario.
- **Método removeUsuario(Usuario \$usuario):** con este método, podremos eliminar un usuario de la colección “usuario” siempre que ya se encuentre en ella. Y además elimina al actual favorito de la colección “favoritos” del usuario.

Clases de formularios

Para la creación de formularios, Symfony requiere de la creación de dos clases, una que actúe como modelo (por ejemplo “Registro”), y otra que contendrá los métodos necesarios para construir el formulario (por ejemplo “RegistroType”).

En este caso, se han necesitado crear cuatro formularios diferentes: “cambiar clave”, “cambiar email”, “eliminar usuario” y “registro”.



Las clases modelo definen los atributos y métodos que van a ser utilizados en el formulario. Esas clases son:

- **Clase CambiarClave**: es el modelo del formulario que permite cambiar la clave del usuario. Contiene dos atributos de clase:
 - **\$claveAnterior**: es la cadena correspondiente a la contraseña del usuario que tiene antes de ser cambiada.
 - **\$clave**: contiene la cadena de la nueva contraseña que ha definido el usuario.

Como métodos de esta clase tenemos los métodos getter y setter de los atributos anteriores.

- **Clase CambiarEmail**: esta clase corresponde al modelo del formulario que permite al usuario cambiar su email. Sus atributos son:
 - **\$clave**: contiene la clave del usuario actual. Se utiliza como confirmación del cambio de email.

- **\$nuevoEmail:** contiene la cadena que corresponde al nuevo email al que quiere cambiarse el usuario.

Igual que la clase anterior, sus métodos son los getter y setter de los atributos.

- **Clase EliminarUsuario:** corresponde al modelo del formulario que confirma la eliminación de la cuenta de un usuario.

Contiene un único atributo: **\$clave**, que corresponde a la contraseña actual, que se usará como confirmación. Sus métodos son los getter y setter de este atributo.

- **Clase Registro:** esta clase es el modelo del formulario de registro de un nuevo usuario. Tiene los atributos siguientes:

- **\$email:** este campo será el email del usuario.
- **\$clave:** este campo corresponde a la contraseña que tendrá el usuario.

De igual forma, sus métodos son los getter y setter de los atributos de clase.

Las clases para construir los formularios (las clases Type) siguen todas la misma estructura. Poniendo de ejemplo la clase “CambiarClaveType”:

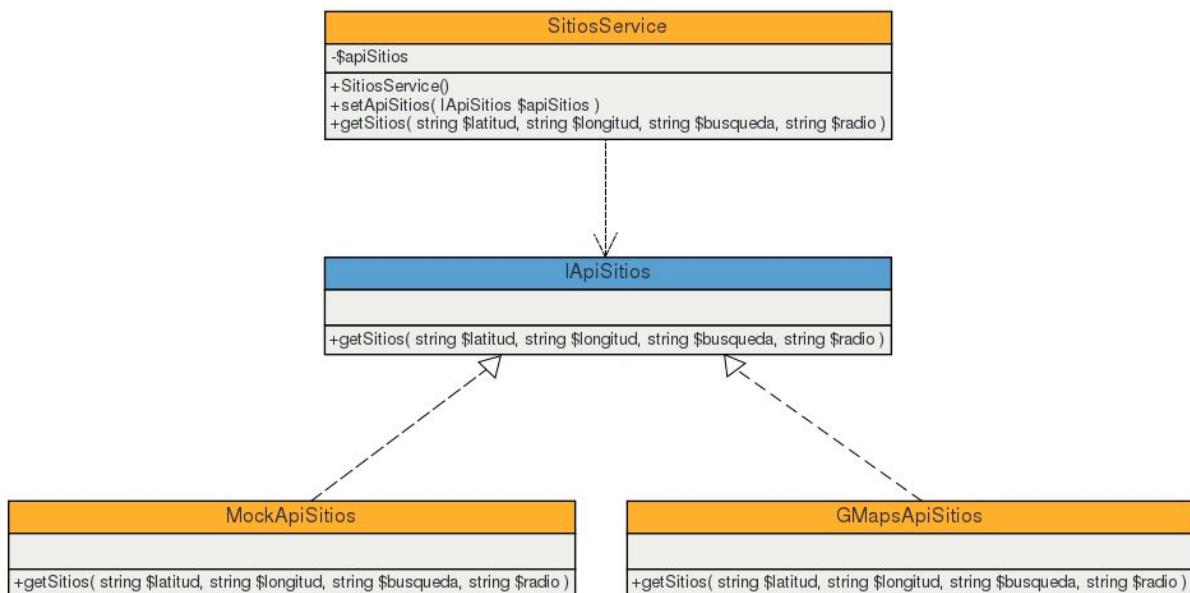
Es una clase que extiende de la clase AbstractType. Contiene tres métodos:

- **Método buildForm(FormBuilderInterface \$builder, array \$options):** es el método que define los campos del formulario (etiqueta, estilos, textos ...). Utiliza un objeto de la interfaz FormBuilderInterface como constructor del formulario. En el ejemplo de CambiarClaveType se crearían los campos de contraseña actual, nueva contraseña, etc.
- **Método setDefaultOptions(OptionsResolverInterface \$resolver):** este método establece mediante un objeto de la interfaz OptionsResolverInterface, a qué clase modelo corresponde este formulario. En este caso, sería la clase “CambiarClave”.
- **Método getName():** este método simplemente devuelve el nombre que se le quiere dar al formulario, que se muestra luego en la página (id del formulario en el DOM). En este caso es “change_passwd”.

Clases del servicio de recogida de datos

A la hora de obtener los datos de los sitios dentro de nuestra aplicación, tenemos dos posibles fuentes de datos: obtener los datos reales a través de la llamada de la API de Google Maps, o utilizando un objeto simulado con los datos necesarios para realizar pruebas.

Para asegurar una escalabilidad futura (una nueva versión de la API por ejemplo) o la aparición de una nueva fuente de datos, se ha hecho uso de **la inyección de dependencias** y del el patrón DAO, esto permite utilizar una fuente de datos (repositorio) u otro sin necesidad de modificar código ya implementado.



Las clases implicadas son las siguientes:

- **Clase SitiosService**

Es la clase que actúa como **servicio para determinar qué fuente de datos utilizar**. Hace uso de la inyección de dependencias. Tiene un atributo de clase llamado **\$apiSitios**, que será el que contenga el objeto del tipo de fuente de datos que se quiera usar. Dispone de tres métodos:

- **Método constructor SitiosService()**: es el constructor de la clase, establece a null por defecto el valor del atributo \$apiSitios.

- **Método setApiSitios(IApiSitios \$apiSitios):** es el método que da un valor al atributo \$apiSitios. Recibe un objeto que trabaja contra la interfaz IApiSitios.
- **Método getSitios (string \$latitud, string \$longitud, string \$busqueda, string \$radio):** este método llama al método getSitios() del objeto \$apiSitios pasándole los parámetros: \$latitud, \$longitud, \$busqueda y \$radio.

- **Interfaz IApiSitios**

Es la interfaz que implementan las clases de fuentes de datos. Incluye un único método llamado getSitios(string \$latitud, string \$longitud, string \$busqueda, string \$radio) que deberán de incluir en sus clases.

- **Clase GMapsApiSitios**

Es la clase que obtiene los datos desde la API de Google Maps según el criterio de búsqueda y la ubicación del usuario. Implementa la interfaz IApiSitios. Hace uso de la librería [joshtronic/php-googleplaces](#), que hace de cliente para la API de Google Maps.

Tiene implementado el método getSitios() de la interfaz. Cuyo funcionamiento es el siguiente:

En primer lugar, haciendo uso de la clase Dotenv, se recoge el valor de la variable de entorno “API_KEY”, dentro del fichero .env.local. Esta clave, es un identificador único que ofrece Google cuando autoriza a un usuario a utilizar su servicio de la API, con él se nos permite realizar las peticiones.

A continuación, creamos un objeto de la clase GooglePlaces, (disponible dentro de la librería), al que le pasamos como parámetro la API_KEY, y seguidamente se asignan los siguientes valores a sus propiedades:

- A la **propiedad location** del objeto, se le pasa un array con la latitud y la longitud.
- A la **propiedad radius**, se le pasa el radio de acción de la búsqueda en metros. Este valor sólo es reconocido por la API si se introduce como cadena (String) y no como entero (int).
- Y la **propiedad keyword**, a la que se le pasa el criterio de búsqueda que ha introducido el usuario.

Después, se almacena en la variable \$results, el resultado de la llamada al método “nearbysearch()”, la cual contiene un objeto con los resultados de la búsqueda.

Dicho objeto contiene un array asociativo con todos los resultados, que se almacenan en la variable \$arrayResultados para poder tratarlos.

```
$dotenv = new Dotenv();
$dotenv->load('../.env');
$API_KEY = $_ENV['API_KEY'];

$google_places = new joshtronic\GooglePlaces($API_KEY);
$google_places->location = array($latitud, $longitud);
$google_places->radius = $radio;
$google_places->keyword = $busqueda;
$results = $google_places->nearbysearch();

$arrayResultados = $results['results'];
```

A continuación, se crea un objeto genérico al que llamamos \$resultado.

Se recorre el array de resultados, almacenando cada valor del resultado de la API que nos interesa como atributo de otro objeto llamado \$objeto.

Finalmente, se almacena cada objeto \$objeto dentro de un array \$sitios[] que se guarda en la variable \$resultado, que es la que se devuelve.

```
$resultado = new stdClass();

$numSitios = count($arrayResultados);

for ($i = 0; $i < $numSitios; $i++) {

    $sitio = $arrayResultados;

    $id = $sitio[$i]['id'];
    $latitud = $sitio[$i]['geometry']['location']['lat'];
    $longitud = $sitio[$i]['geometry']['location']['lng'];
    $nombre = $sitio[$i]['name'];
    $icono = $sitio[$i]['icon'];
    $direccion = $sitio[$i]['vicinity'];
```

```
$objeto = new stdClass();
$objeto->id = $id;
$objeto->latitud = $latitud;
$objeto->longitud = $longitud;
$objeto->nOMBRE = $nOMBRE;
$objeto->iCONO = $iCONO;
$objeto->dIRECCION = $dIRECCION;
$objeto->fAVORITO = false;

$resultado->sITIOS[$i] = $objeto;
}

return $resultado;
```

- **Clase MockApiSitios**

Esta clase se encarga de **devolver un mock (objeto simulado) al usuario** de una búsqueda anterior ya realizada mediante la API de Google Maps. Implementa la interfaz IApiSitios.

El objetivo de esta clase es **evitar las continuas llamadas a la API** de Google Maps en el momento del desarrollo.

Al igual que la clase anterior, esta clase implementa el método getSitios() de la interfaz. En este método se almacena una variable llamada \$json, que contiene el resultado de una búsqueda ya realizada a la API en formato JSON, almacenado como una cadena de texto plano.

Dicha cadena, se decodifica y se convierte a objeto. Se recorre, y se crea un objeto \$resultado a partir de sus datos, que es el que el método devuelve.

Parte cliente (JavaScript)

Funciones de llamada al servidor

Desde el cliente, se realizan peticiones asíncronas (utilizando tecnología AJAX) para la comunicación con el servidor. Estas peticiones realizan operaciones como la recogida de una lista de datos, modificación, borrado ...

Según cada fichero, podemos distinguir los diferentes métodos que realizan llamadas al servidor:

- **Fichero búsqueda:**

- **Método buscar()**

Este método es el encargado de **realizar la búsqueda de un nuevo sitio**, y muestra los resultados sobre un mapa.

En primer lugar, el método comprueba si el mapa ya tiene contenido (lo que indicaría que ya se ha realizado una búsqueda), y si este lo tiene, elimina su contenido, borra la caché almacenada y vuelve a generar otro mapa.

Más adelante, se declaran e inicializan las variables que utiliza la librería “Leaflet” para los iconos de los marcadores: azul para las nuevas búsquedas, y amarillo para indicar los sitios que ya tenemos entre nuestros favoritos.

A continuación, se realiza la **recogida de datos del usuario**: se recoge el valor que contiene el cuadro de búsqueda, el deslizador que indica el radio y las variables globales “lat” (latitud) y “lng” (longitud) que indican la ubicación actual del usuario, y con esta información, se crea un objeto llamado “datos” que se incluye en formato JSON dentro de la variable “json”.

Una vez recogidos los datos del usuario, se realiza la **petición POST** utilizando la API fetch de JavaScript. La petición se realiza al servidor, llamando al endpoint “/buscarLugares”; como cuerpo de la petición se incluyen la variable “json” nombrada anteriormente.

Una vez responda el servidor, se almacena su respuesta en una variable global denominada “json_cache”, que actuará como caché de la respuesta para ahorrar peticiones a la API, y además de ofrecer un mejor rendimiento y accesibilidad dentro del código.

Los datos recogidos, se recorren utilizando un bucle for (según la cantidad de sitios devueltos) y se hace el montaje de una cadena de texto del contenido del popup en HTML con los datos devueltos.

Dependiendo de si el sitio devuelto está o no en los sitios favoritos del usuario, se añade a la cadena anterior el ícono correspondiente (azul o

amarillo) y se añaden al array de los marcadores “arrayMarcadores”, se añaden al mapa, y se adaptan para que se centren en la pantalla.

- **Método accionPopup(posicion)**

Este método es llamado desde el botón de la estrella situado en cada uno de los “pop up” de cada sitio del mapa.

Este método llama a los dos métodos siguientes: nuevoFavorito() o eliminarFavorito() según si el sitio ya está en favoritos o no; esto se consigue obteniendo el valor del campo “favorito” del sitio según su posición dentro de la “variable caché json_cache”.

- **Método nuevoFavorito(posicion)**

Con este método, **se añade a favoritos** aquel sitio del mapa que no tengamos entre nuestros sitios favoritos.

Para ello, se construye un objeto con los datos del sitio con únicamente el dato de su posición dentro de la variable “json_cache” (dato que se pasa a través del método accionPopup()).

Con el objeto creado y “parseado” a JSON, se realiza una **petición POST** al endpoint “/nuevoFavorito” del servidor, y se le pasa el objeto en el cuerpo de la petición.

Si se recibe una respuesta correcta del servidor, se le informa al usuario ejecutando una breve vibración (en los dispositivos compatibles) y un sonido que indica que la operación ha sido exitosa.

Al mismo tiempo, se cambia el aspecto visual del marcador al de un sitio ya agregado a favoritos, para ello, se realizan los siguientes pasos:

- Se cambia a “true” el valor “favorito” de ese sitio dentro de la variable caché “json_cache”.
- Se elimina la estrella vacía, y se cambia por una rellena, iniciando una animación al usuario durante dicha transición.
- Pasado un segundo (tiempo que toma la animación), se ejecuta la **función callback “comprobarEstado()”**, que comprueba el nuevo estado del sitio (si es favorito o no) dentro de la caché, y cambia el contenido HTML del pop up, con la nueva información.

Este paso es necesario debido a las **restricciones de la librería “Leaflet”** a la hora de fijar cambios en la información del interior del pop up, el cual debe de estar abierto para poder

modificar su contenido, y dicho contenido debe ser reescrito completamente (la cadena de texto completa).

- **Método eliminarFavorito(posicion)**

Este método se encarga de **eliminar los sitios favoritos ya añadidos** anteriormente, y que han aparecido en la búsqueda.

Mediante la posición del sitio en la caché (pasada en la llamada del método), se calcula el ID del sitio. Dicho ID se pasa como parámetro al servidor realizando, en este caso, **una petición DELETE**.

Si la respuesta del servidor es correcta, al igual que el método “nuevoFavorito()”, se le indica al usuario mediante vibración y sonido.

En este caso, al contrario que el método anterior, el sitio cambia el valor de “favorito” dentro de la caché a “false”, la estrella pasa a estar vacía, y de igual forma, se llama al método “comprobarEstado()” como se ha mencionado anteriormente.

- **Fichero sitios:**

- **Método obtenerFavoritos()**

Con este método se **obtienen todos los sitios favoritos del usuario** y se sitúan en el mapa.

En primer lugar se realiza una **petición GET** al endpoint “/obtenerFavoritos” del servidor. Si éste responde, se recorre el objeto de los sitios recibido y al igual que el método buscar(), se generan los marcadores, y se sitúan en el mapa.

- **Método eliminarFavorito(posicion)**

Este método permite **eliminar un sitio del mapa** de favoritos pulsando sobre el icono de la estrella.

Utilizando la posición del sitio (pasada durante la llamada al método), se calcula el ID del sitio mediante los datos almacenados en caché, y se realiza una **petición DELETE** al endpoint “eliminarFavorito/id”.

Si se ha eliminado el sitio correctamente, se muestra una respuesta al usuario mediante una vibración y un sonido, y el marcador de ese sitio desaparece del mapa.

Tras el aviso al usuario, se comprueba si el sitio eliminado es el último del mapa, y si es así, el icono de “Eliminar todos mis sitios” situado bajo el mapa, desaparece.

En caso de que el servidor devuelva un error, se le muestra al usuario una ventana modal informando de que no se ha podido eliminar el sitio.

- **Método eliminarFavoritos()**

Con este método, el usuario puede **eliminar todos sus sitios favoritos** del mapa.

Esta función es llamada al pulsar el botón “Eliminar todos mis sitios”, situado tras el mapa.

Se realiza una **petición DELETE** al endpoint “/eliminarFavoritos” del servidor. Si la respuesta del servidor es correcta, se le informa al usuario mediante una ventana modal, acompañada de una vibración rápida y un sonido. El botón de “Eliminar todos mis sitios” se elimina de la página, y la lista que contenía los marcadores de los sitios, se vacía.

Si el servidor responde con un error, se le informa al usuario con otra ventana modal, indicando que no se han podido eliminar los sitios.

- **Fichero útiles :**

- **Método comprobarUsuario(email)**

Este método **comprueba que el email introducido por el usuario tenga un formato correcto, y que no esté ya registrado en la base de datos**. Es llamado cada vez que el usuario escriba una letra en el “input” de un nuevo email.

En primer lugar, se comprueba que el email pasado como parámetro cumpla con la expresión regular. Si no se cumple, se le muestra un mensaje al usuario (bajo el campo input) indicando que el formato no es válido. Si el formato es correcto, se realiza **una petición POST** al endpoint “/comprobarUsuario” del servidor, pasando en el cuerpo de la petición el parámetro que contiene el email.

Si el servidor responde con una respuesta positiva, se le indica al usuario que el email introducido se encuentra disponible y se colorea el mensaje de color verde.

En cambio, si el servidor responde con una respuesta negativa, se le indica que el email introducido ya está en uso, y se colorea el mensaje de color rojo.

Fichero de utilidades (utiles.js)

Dentro de este fichero podemos distinguir dos tipos de funciones:

- **Funciones para la gestión de los mapas**

- **Método generarMapa(seccion)**

Esta método es el encargado de **generar el mapa inicial con la ubicación del usuario** utilizando objetos y funciones de la librería Leaflet.

Para comenzar, se comprueba que la ubicación esté activa, accediendo a la variable global “estado_ubicacion”. Si la ubicación no está activa, se le informa al usuario mediante una ventana modal.

En caso contrario, si está activa, se crea el objeto map, con las coordenadas latitud y longitud, del usuario (también variables globales) y un valor que indica la cantidad de zoom inicial que habrá sobre dicha ubicación (altura en la vista).

Tras esto, se crean los controles de pantalla completa, y se añaden al mapa.

A continuación se crean los objetos que contienen las distintas capas de mapas:

- En primer lugar, se crea la capa base mediante una URL, en este caso de OpenStreetMaps, y se añade al mapa.
- Despues, se crea un nuevo objeto llamado “osm” que contiene la URL de OpenStreetMaps (mismo que el anterior) y un texto para mostrar en la leyenda del mapa.
- También se crea un objeto “googlesat”, que contiene la URL de Google Maps para las imágenes de satélite y, al igual que el anterior, el texto que se mostrará en la leyenda del mapa.
- Tras esto, se crean los botones que permitirán seleccionar una capa u otra dentro del mapa, a los que se les referencia los objetos anteriores (“osm” y “googlesat”), con los nombres a mostrar: “Callejero” y “Satélite”.
- Finalmente, se añaden los botones al mapa.

Una vez añadidas las capas al mapa, se crean los distintos objetos necesarios para poder mostrar en el mapa la ubicación del usuario:

- Se crea y añade al mapa la variable “circle”, que indica el tamaño, color y forma del círculo que rodea la ubicación del usuario.
- A continuación, se crea el icono del marcador que indicará la ubicación del usuario, en este caso de color rojo.
- Junto al ícono anterior y las coordenadas del usuario (latitud y longitud), se crea el marcador del usuario, que finalmente se añade al mapa.

- **Funciones auxiliares (utilidades)**

- **Método obtenerCoordenadas(seccion)**

Con este método, **se calculan las coordenadas del usuario** al momento de entrar en la página.

Si el usuario tiene la ubicación activada se llama al método “getCurrentPosition()”. Si se ha podido obtener la ubicación, se almacena su valor en las variables globales “lat” y “lng”, y el valor de la variable “estado_ubicación” pasa a ser true.

Mediante la variable “sección” pasada como parámetro, dependiendo de si estamos en la sección “Búsqueda” o en “Mis sitios”, se habilitan diferentes campos, como el input de búsqueda, el botón de buscar, o el mensaje del pie de página.

Tras obtener la ubicación, se pasa a llamar automáticamente al método “generarMapa(sección)” nombrado anteriormente.

En el caso de que la obtención de la ubicación haya fallado, se contemplan mediante un “switch”, los diferentes mensajes de error que se deben mostrar al usuario a través de una ventana modal.

Dichos mensajes son diferentes dependiendo del tipo de error relacionado con la ubicación que se ha recibido: permiso denegado, posición no disponible, “timeout” en la obtención de la ubicación o un error desconocido.

- **Método medirDistancia(lat1, lon1, lat2, lon2)**

Este método realiza un **cálculo aproximado de la distancia entre la ubicación del usuario y un sitio concreto**.

Se llama a esta función desde los popups de los marcadores de los sitios del mapa.

Se le pasan como parámetros la latitud y longitud del usuario, y la de un sitio.

Mediante diversas operaciones matemáticas, utilizando el radio de la Tierra como referencia, se calcula la distancia aproximada en kilómetros entre ambos puntos.

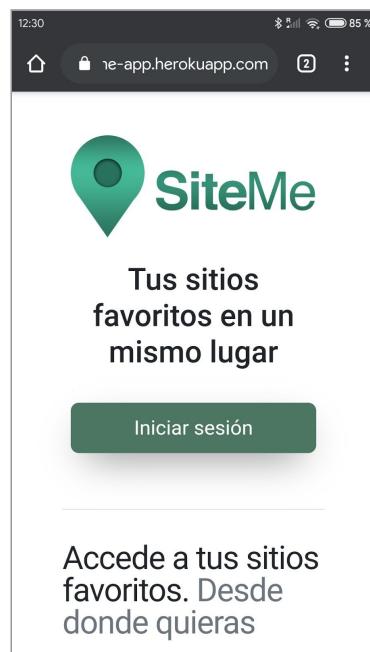
6.2.4 - Pruebas

Para comprobar el correcto funcionamiento de la aplicación se han realizado diferentes pruebas, que corresponden a diferentes casos de uso.

Se ha simulado ser un nuevo usuario que llega a la aplicación y quiere registrarse. Algunas acciones que se han comprobado son:

- En el registro, que se comprueben correctamente los campos email (que tenga un formato correcto y no esté en uso) y que las contraseñas introducidas coincidan.
- Dentro de la aplicación se ha comprobado que al buscar sitios aparezcan en el mapa, y estos se ajusten a la pantalla.
- Al añadir un sitio favorito, éste debe aparecer en la sección de “Mis sitios” en el mismo lugar y con los mismos datos.
- Al realizar una búsqueda, si algún sitio de los resultados ya se encuentra entre nuestros sitios favoritos, debe aparecer marcado de color amarillo.
- Se deben poder eliminar los sitios ya añadidos a favoritos desde la sección “Búsqueda” y desde “Mis sitios”.
- El botón de “Borrar todos mis sitios” debe desaparecer cuando no tenemos sitios en el mapa, y debe eliminar todos los sitios del mapa si se confirma su mensaje de la ventana modal.
- Desde ajustes, el cambio de email y contraseña debe realizarse correctamente, comprobando que el nuevo email no esté en uso y que las nuevas contraseñas coinciden.
- La opción de “Eliminar cuenta” debe comprobar correctamente la contraseña antes de eliminar la cuenta. Y una vez eliminada, si se intenta iniciar sesión se debe recibir el mensaje de “El correo no existe”.

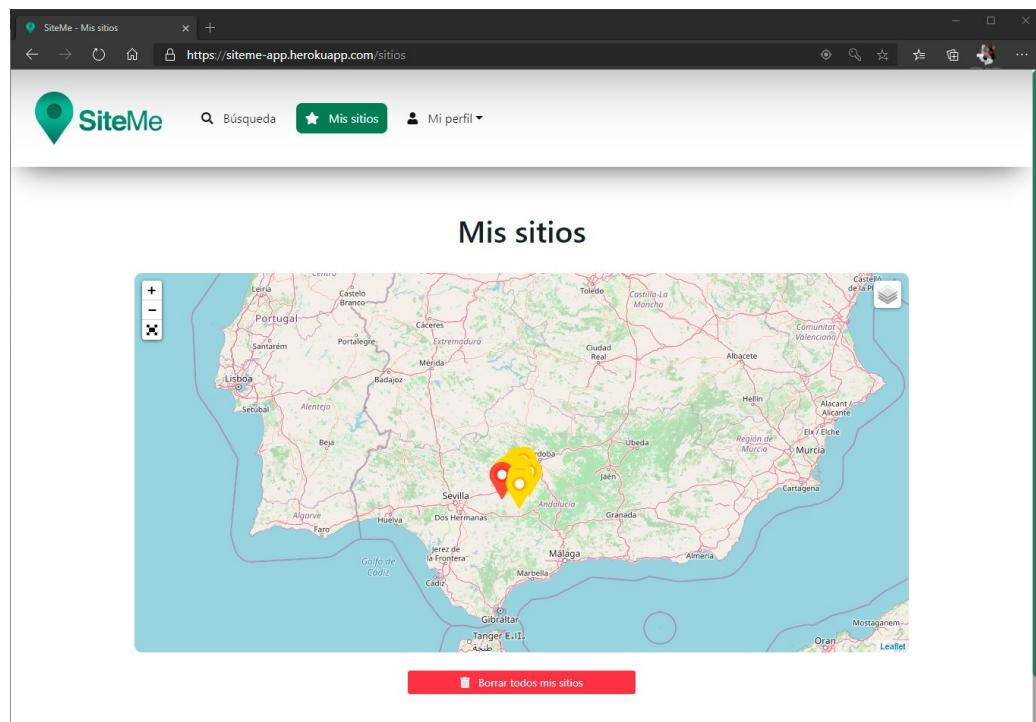
En el ámbito de la compatibilidad, se ha probado la aplicación en diferentes dispositivos: ordenadores de escritorio, portátiles, tablets y smartphones que utilizan diferentes plataformas: Windows, Android, macOS, iOS y distintas distribuciones de Linux, con diferentes navegadores: Google Chrome, Mozilla Firefox, Apple Safari y Microsoft Edge, navegadores más extendidos actualmente:



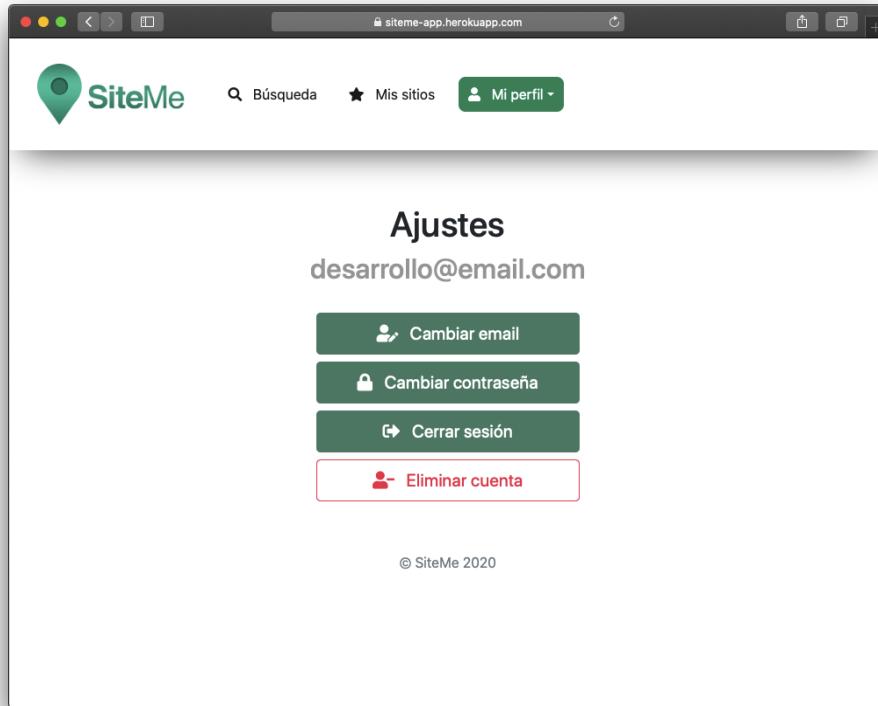
Google Chrome en Android



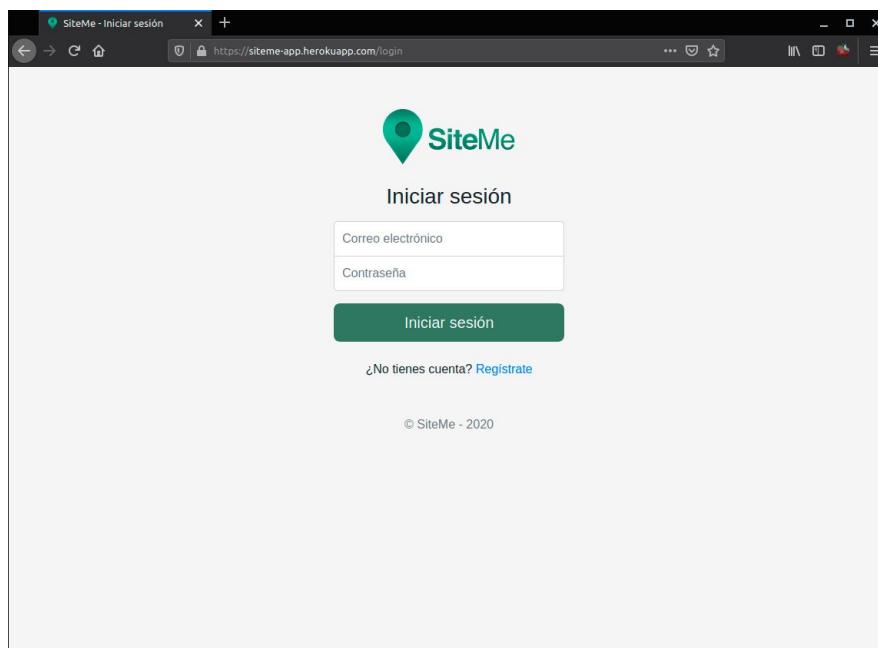
Apple Safari en iOS



Microsoft Edge en Windows



Apple Safari en macOS



Mozilla Firefox en Linux (distro Linux Mint)

Además, como prueba de accesibilidad y usabilidad, se ha instalado la aplicación en smartphones de amigos y familiares ajenos al proyecto para que la probaran y dieran una opinión: dificultades encontradas, estética, posibles funcionalidades...

6.3 - Desarrollo web

6.3.1- Estructura del sitio web

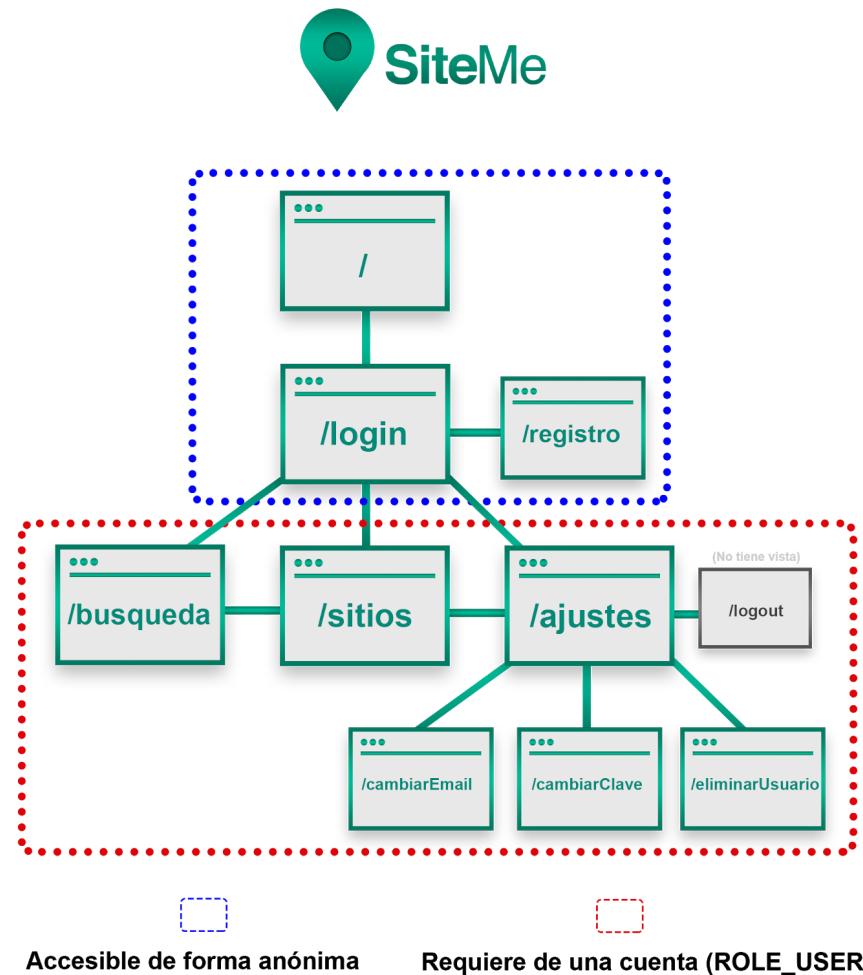
En este proyecto, se ha seguido una estructura jerárquica para representar las diferentes páginas. Al tratarse de una MPA (Multiple-Page-Application), cada página representa una ruta en la URL (los distintos endpoints).

La organización de dichas rutas es la siguiente:

- En primer lugar, al acceder a la URL **raíz “/”**, accedemos a la página de Inicio, una web donde se describen las características principales de la aplicación.
- De la raíz, depende directamente la URL **“/login”**, donde el usuario puede iniciar sesión con su cuenta de SiteMe.
 - Desde el Login, podremos acceder a la URL **“/registro”**, donde el usuario podrá crear una nueva cuenta si no dispone de una.
- Si el usuario inicia sesión correctamente, podrá acceder al contenido de la aplicación que queda restringido a usuarios registrados (internamente a usuarios con rol **ROLE_USER**). Todas las URL siguiente son accesibles entre ellas mediante el uso del menú de navegación:
 - La URL **“/busqueda”** es la mostrada por defecto una vez que el usuario inicia sesión. Desde este apartado, el usuario podrá realizar la búsqueda de nuevos sitios y añadirlos a favoritos.
 - La URL **“/sitios”** muestra los sitios favoritos guardados por el usuario en un mapa y ofrece la posibilidad de eliminarlos.
 - Desde la URL **“/ajustes”** podemos acceder a los ajustes de la cuenta del usuario actual. Desde aquí pueden llevarse a cabo diferentes operaciones como “Eliminar todos los sitios favoritos” o “Cerrar sesión” (que se procesa a través de la URL **“/logout”**).

Además, pueden realizarse operaciones más complejas que requieren de una autenticación adicional desde las siguientes URLs:

- La URL **“/cambiarEmail”**, que permite al usuario cambiar el email asociado a su cuenta.
- La URL **“/cambiarClave”** donde el usuario puede cambiar la contraseña de su cuenta.
- Y la URL **“/eliminarUsuario”** donde puede darse de baja como usuario de SiteMe.



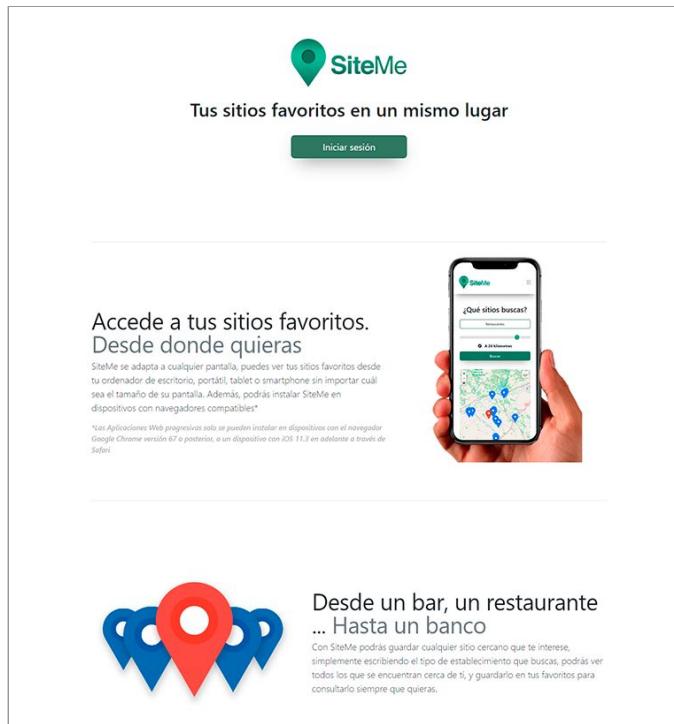
6.3.2 - Maqueta del sitio web

En el paso previo a la codificación de la aplicación, se planteó mediante el uso de “wireframes” (maquetas del sitio web), el concepto de diseño de cada una de las páginas que componen la web, teniendo en cuenta la versión de escritorio y la versión móvil.

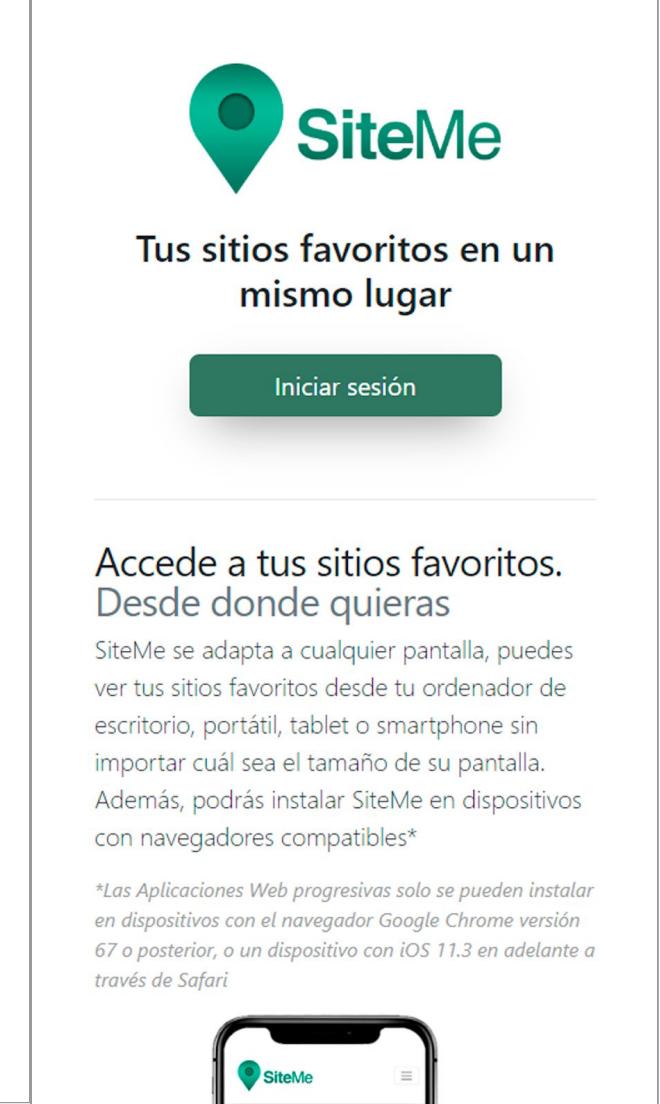
En las siguientes imágenes, se muestra una comparación entre la maqueta y una captura real para cada una de las páginas:

Página de Inicio

Vista de escritorio



Vista móvil



LOGO

Lore ipsum dolor sit amet

Botón

IMAGEN

Tus sitios favoritos en un mismo lugar

Iniciar sesión

Accede a tus sitios favoritos. Desde donde quieras

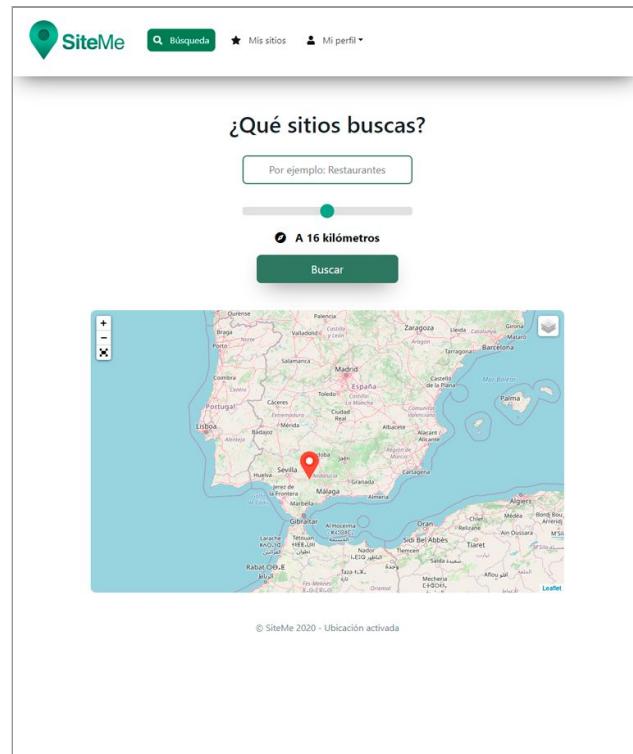
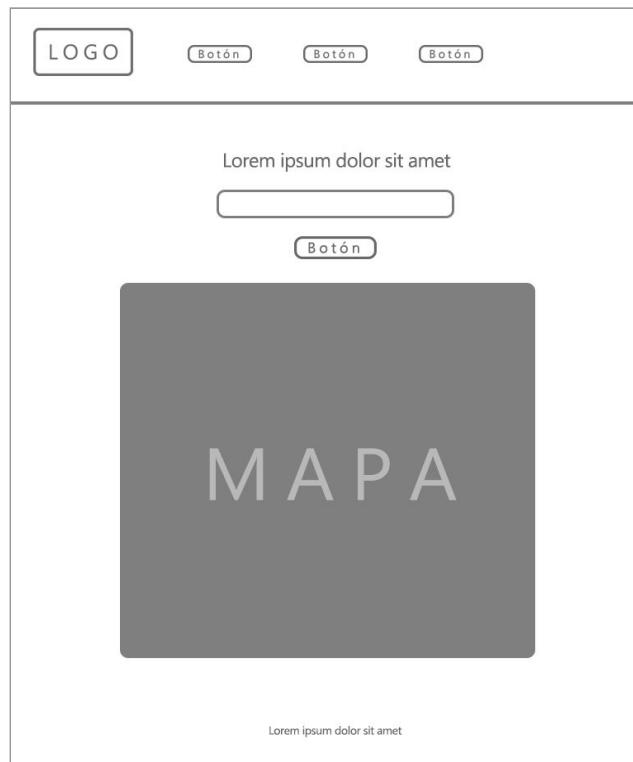
SiteMe se adapta a cualquier pantalla, puedes ver tus sitios favoritos desde tu ordenador de escritorio, portátil, tablet o smartphone sin importar cuál sea el tamaño de su pantalla. Además, podrás instalar SiteMe en dispositivos con navegadores compatibles*

*Las Aplicaciones Web progresivas solo se pueden instalar en dispositivos con el navegador Google Chrome versión 67 o posterior, o un dispositivo con iOS 11.3 en adelante a través de Safari



Página “Búsqueda”

Vista de escritorio



Vista móvil

The image shows a side-by-side comparison of SiteMe's mobile and desktop websites.

Mobile View (Left):

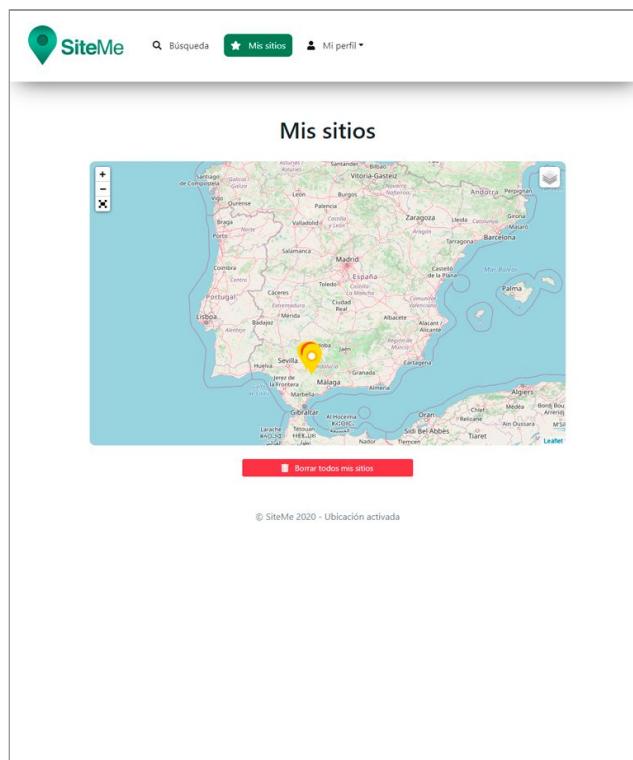
- Header:** Contains a placeholder "LOGO" box and an empty square box.
- Text Area:** Displays placeholder text "Lorem ipsum dolor sit amet" above a large empty input field.
- Call-to-Action:** A button labeled "Botón".
- Background:** A large dark gray rectangular area containing the word "MAPA" in white.

Desktop View (Right):

- Header:** Features the SiteMe logo with a teal location pin icon and a three-line menu icon.
- Section:** "¿Qué sitios buscas?" (What sites are you looking for?)
- Text Input:** Placeholder "Por ejemplo: Restaurantes" (For example: Restaurants).
- Search Radius:** A slider set to "A 16 kilómetros" (Within 16 kilometers).
- Search Button:** A green button labeled "Buscar" (Search).
- Map:** A map of the Iberian Peninsula and surrounding regions. It shows major cities like Madrid, Barcelona, and Lisbon, along with provincial boundaries and labels. A red location pin marks a specific point in Andalucía, Spain. Map controls (+, -, X) are visible in the top-left corner of the map area.

Página “Mis sitios”

Vista de escritorio



Vista móvil

LOGO

Vista móvil

+

-

MAPA

+

-

×

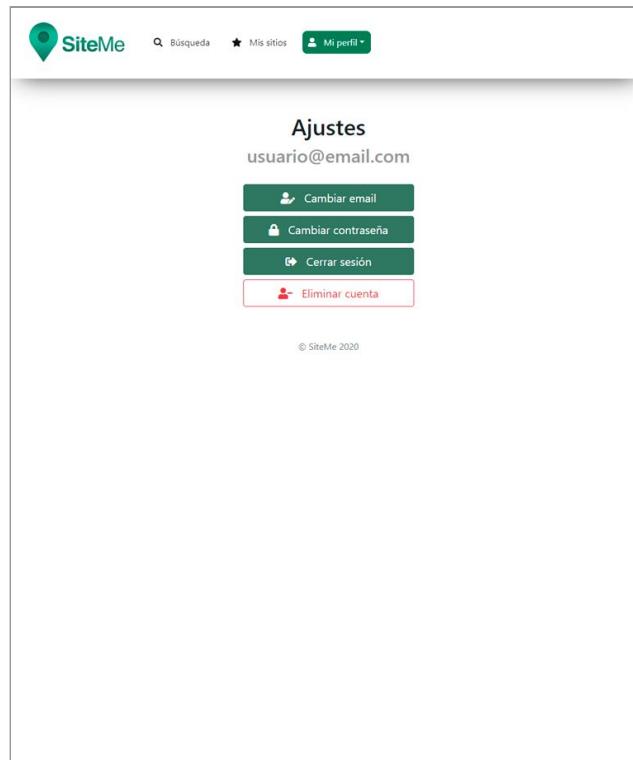
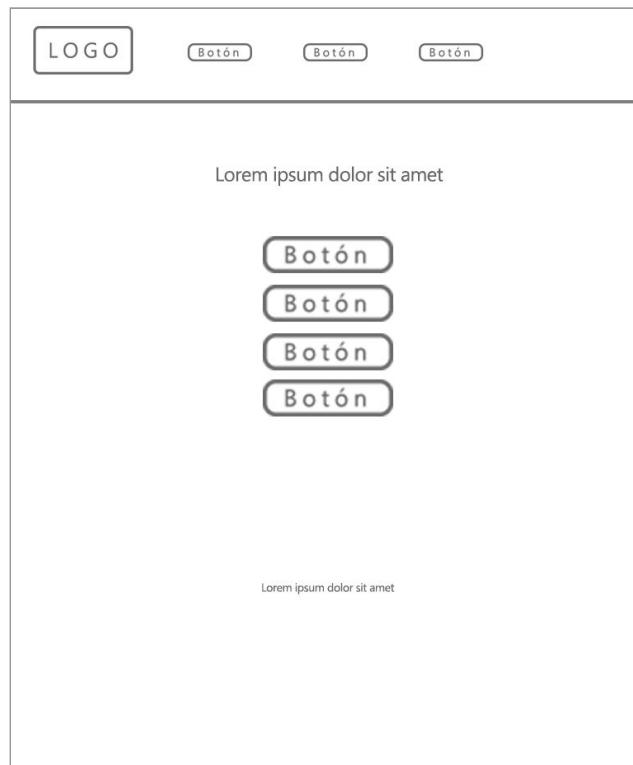
SiteMe

Mis sitios

Borrar todos mis sitios

Página “Ajustes”

Vista de escritorio



Vista móvil

The image shows a comparison between a mobile view (left) and a desktop view (right) of a SiteMe website.

Mobile View (Left):

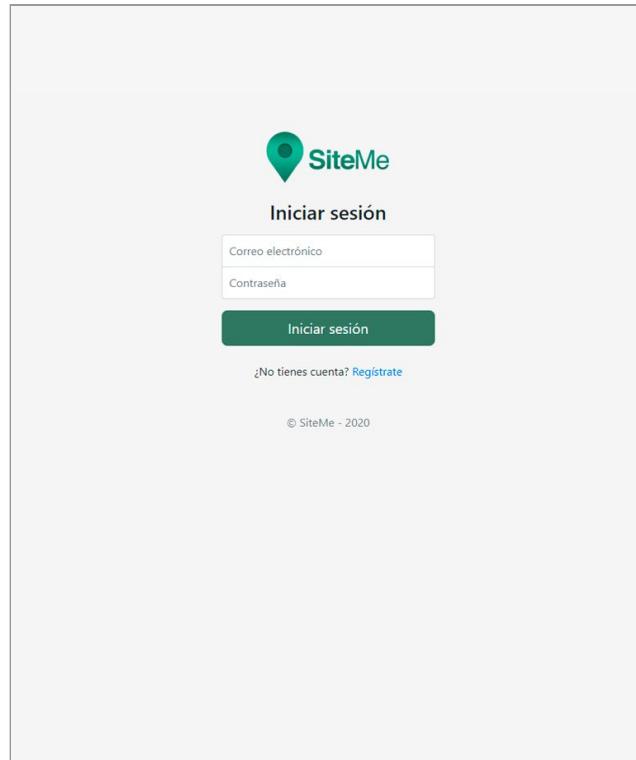
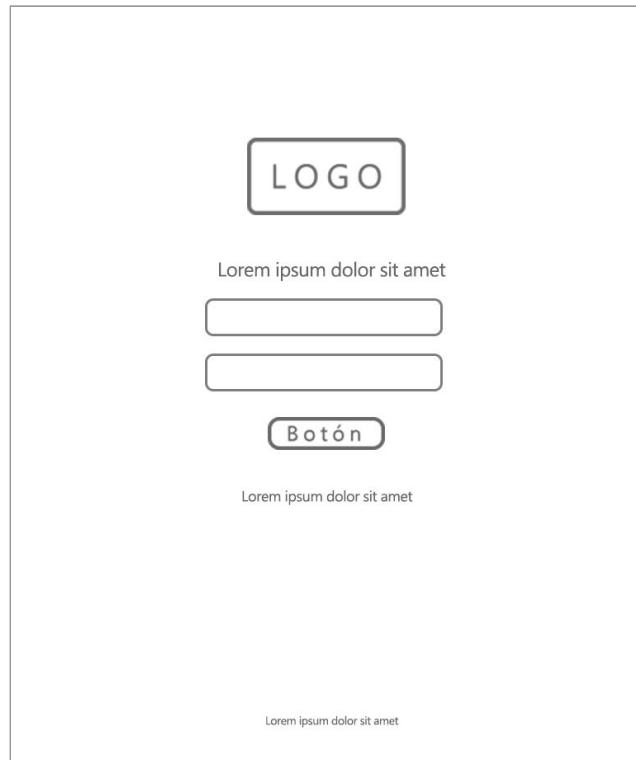
- Header: A placeholder box labeled "LOGO" and an empty square placeholder.
- Main Content: Placeholder text "Lorem ipsum dolor sit amet" and four rounded rectangular buttons labeled "Botón".
- Footer: Placeholder text "Lorem ipsum dolor sit amet".

Desktop View (Right):

- Header: SiteMe logo with a green location pin icon and a three-line menu icon.
- Title: "Ajustes" (Settings) and "usuario@email.com".
- Buttons:
 - "Cambiar email" (Change email) with a person icon.
 - "Cambiar contraseña" (Change password) with a lock icon.
 - "Cerrar sesión" (Close session) with a right-pointing arrow icon.
 - "Eliminar cuenta" (Delete account) with a person minus icon, highlighted with a red border.
- Footer: © SiteMe 2020.

Páginas con autenticación (login, registro, cambio de email...)

Vista de escritorio



Vista móvil

LOGO

Lorem ipsum dolor

Botón

Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet



Iniciar sesión

Correo electrónico

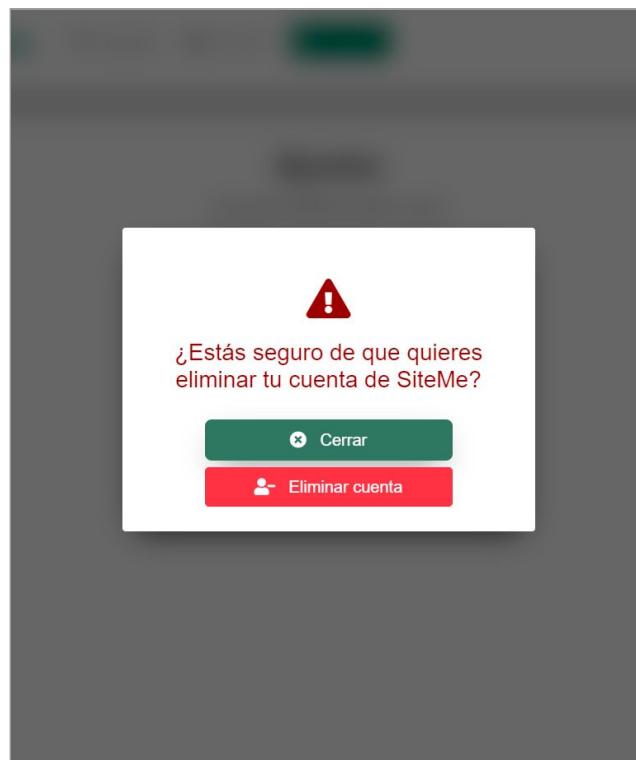
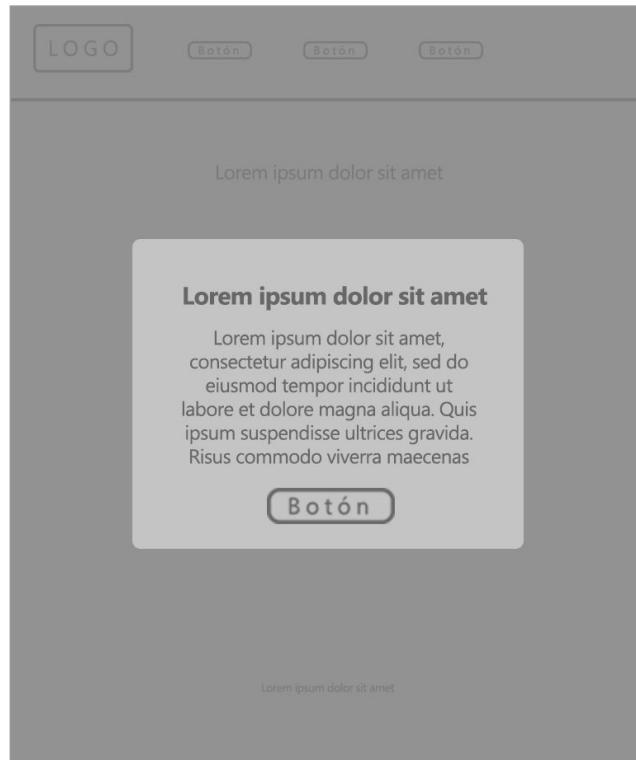
Contraseña

[¿No tienes cuenta? Regístrate](#)

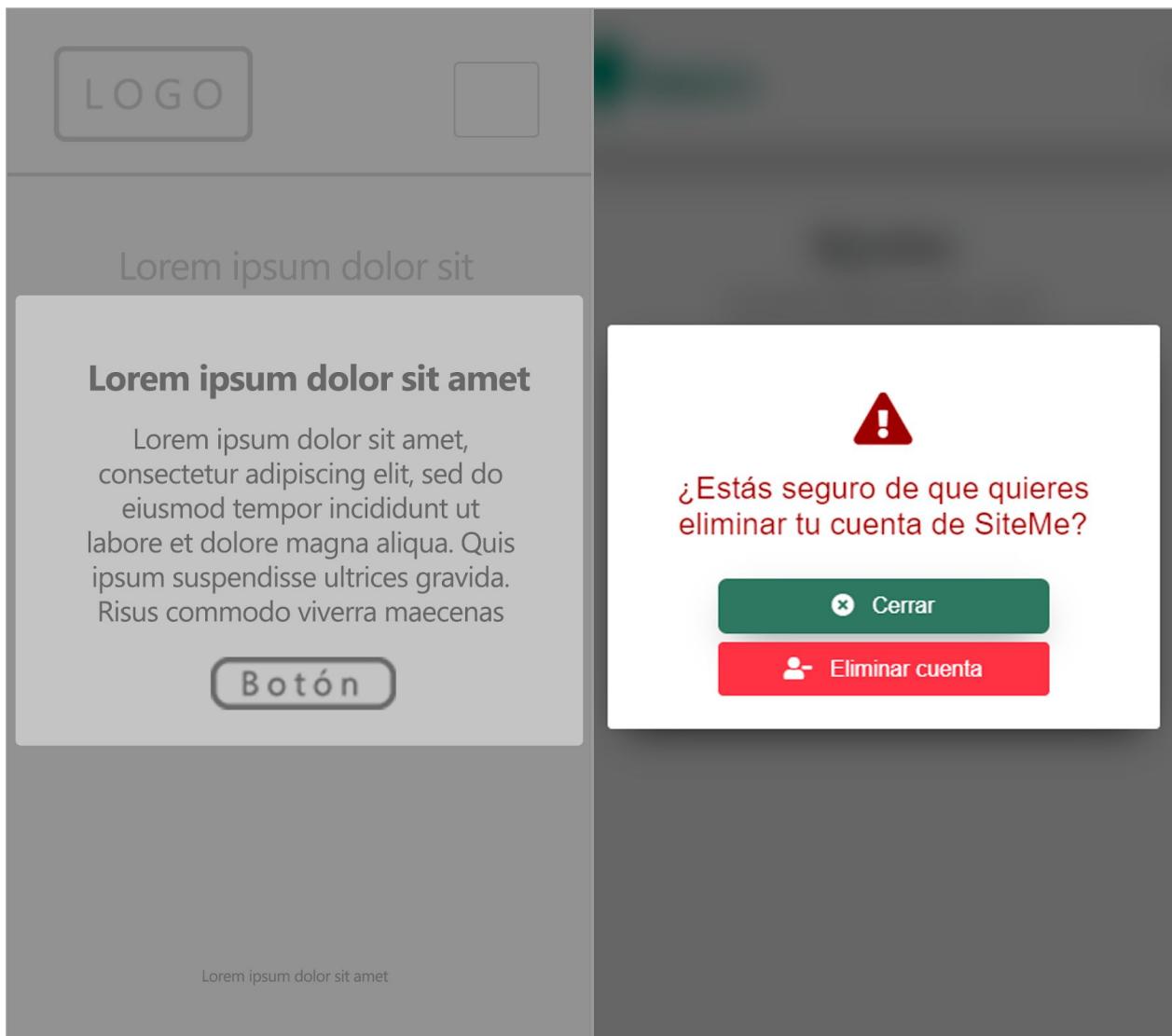
© SiteMe - 2020

Estilo de las ventanas modales

Vista de escritorio



Vista móvil



6.3.3 - Manual de diseño

Para conseguir una consistencia visual en el diseño de la aplicación, se ha seguido una guía de estilo propia, que es visible en toda la aplicación:

Logotipos

Podemos distinguir dos clases de logotipos: uno simplificado (utilizado en el favicon) y otro más completo, visible la gran mayoría de secciones de la web.



Logo simplificado



Logo completo

El logo de SiteMe representa una chincheta, utilizada de forma habitual por aplicaciones de trazado de rutas, visualización de mapas, etc; pero de una forma más simplificada.

Colores

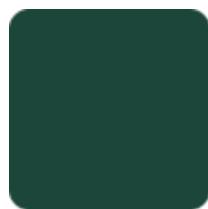
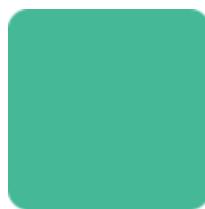
El color predominante en la aplicación es el verde, con diferentes gradientes. Este color es visible en los logotipos, botones y demás elementos interactivos de la web.

La representación hexadecimal de los colores principales utilizados son:

#4d7662

#75b797

#30483c



Dependiendo del contraste de color con el fondo de ciertos elementos dentro de la aplicación, los colores anteriores han tomado diferentes matices algo más claros o algo más oscuros. Como por ejemplo en el menú de navegación, donde el paso del

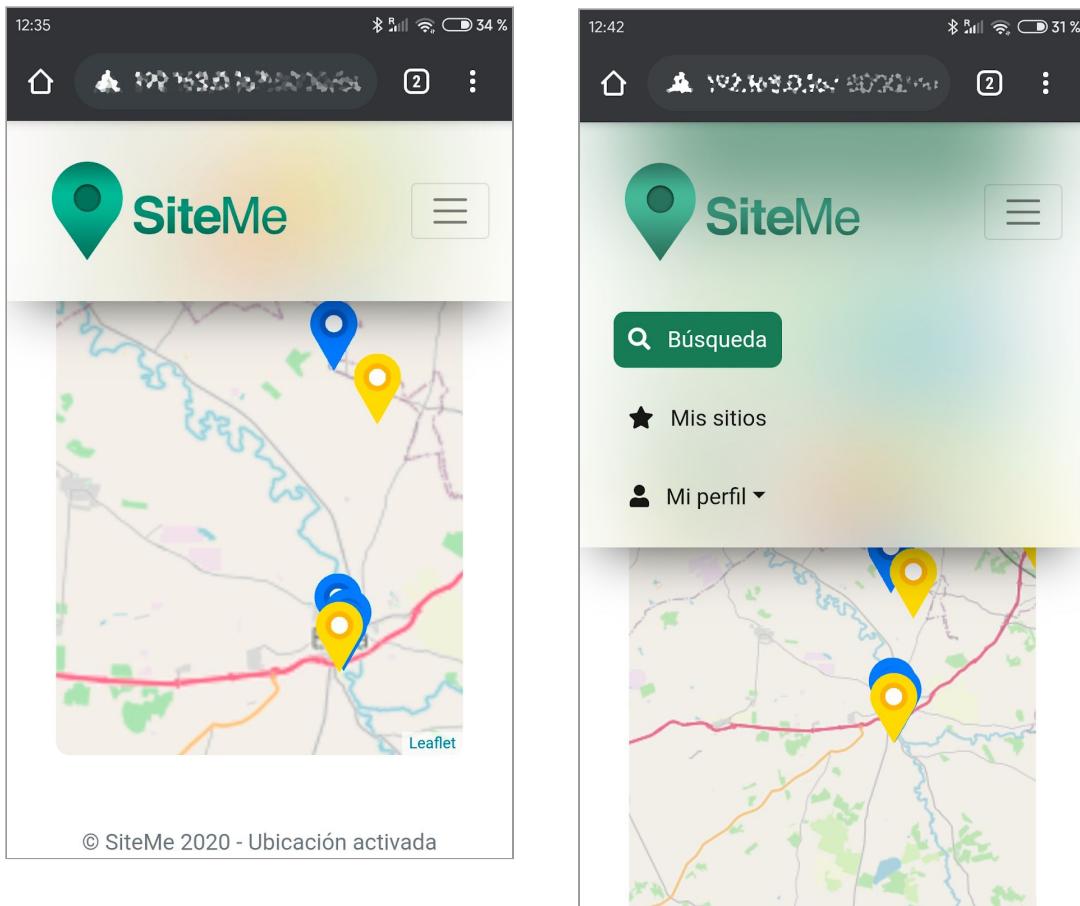
ratón sobre los distintos elementos, muestra una variante de color diferente respecto los colores principales: #1fa874



Además de los colores basados en el verde, predominan colores claros, casi blancos (tonalidades de gris claro) en los fondos y en el menú, y el negro y gris en las tipografías.



En los navegadores compatibles, se añade un efecto translúcido al menú con respecto a los elementos que se sitúan tras él, lo que da una sensación de inmersión al usuario y deja apreciar parte del contenido. Esto es especialmente visible al navegar a través desde el mapa de sitios desde un dispositivo móvil:



Capturas de pantalla realizadas desde Chrome en Android

Imágenes

La página de inicio contiene varias imágenes que simbolizan algunas de las características que tiene SiteMe. Éstas imágenes son de autoría propia, con algunas modificaciones realizadas a través del software de edición Adobe Photoshop.

Para la imagen que muestra la sección de búsqueda en un smartphone, se ha utilizado un “mockup” de un iPhone 11 como plantilla, dónde se ha colocado la captura de pantalla real.



Tipografías

En la totalidad de la aplicación, se ha utilizado la fuente **Segoe UI**, una fuente muy utilizada, que garantiza una gran compatibilidad con la mayoría de navegadores. Es una fuente que se adapta perfectamente a diferentes tamaños de pantalla, lo que asegura su legibilidad.

Es la fuente utilizada por ejemplo, en versiones actuales del sistema operativo Microsoft Windows, disponible en millones de dispositivos diferentes por todo el mundo, lo que da una visión de lo extendida que se encuentra.

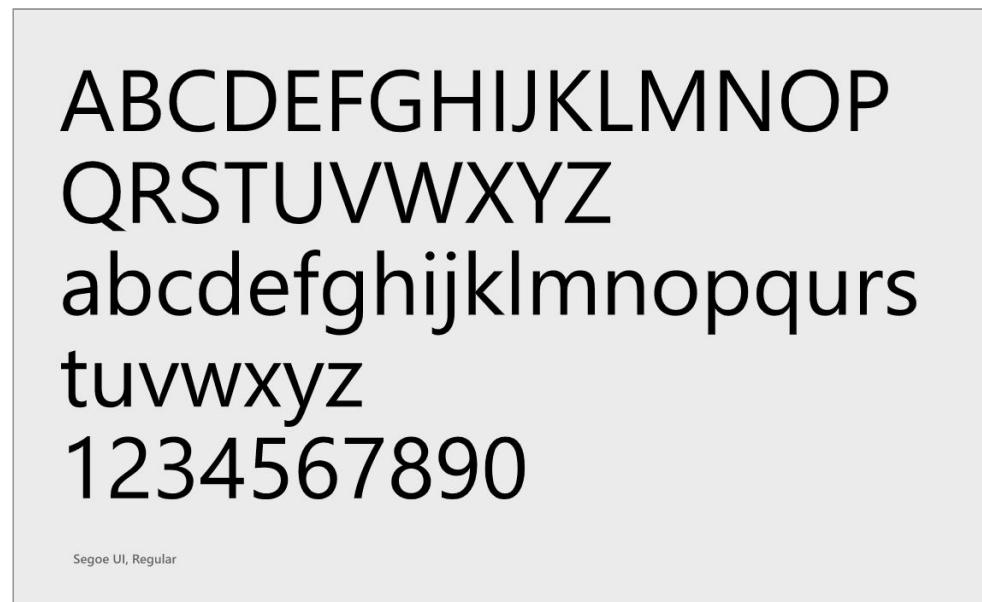


Imagen extraída de la documentación oficial de Microsoft

Sólo existe una excepción respecto a la fuente utilizada, y se encuentra en el logotipo, dónde se ha utilizado **Helvetica**. Esto se debe a que durante su diseño, se buscaba una fuente más compacta y recta, que no incluyera demasiada separación en sus letras, pero que a su vez, fuera legible en pantallas más pequeñas, lo que hacía que Segoe UI, no encajase del todo.



Logo con Helvetica



Logo con Segoe UI

Frameworks de diseño

En el desarrollo del proyecto, se ha utilizado el framework de CSS Bootstrap. Esto ha ayudado a que márgenes, espacios, tamaños de letra, menús, formularios y otros elementos, integren un diseño responsive por defecto en toda la aplicación.

Sobre los diseños de Bootstrap se han hecho modificaciones para hacerlos más acordes con las líneas de diseño de SiteMe, como son el color, fuentes, distribución de los elementos, etc.

El menú de navegación, los botones, “inputs” y la sección de ajustes, son un claro ejemplo de elementos de Bootstrap muy modificados y adaptados al nuevo diseño.

A screenshot of the SiteMe search interface. At the top, there's a header with the SiteMe logo, a search bar labeled 'Búsqueda', and user navigation links for 'Mis sitios' and 'Mi perfil'. Below the header is a search form with a placeholder 'Por ejemplo: Restaurantes', a range slider set to 'A 16 kilómetros', and a large green 'Buscar' button. At the bottom is a map of Spain and parts of Portugal and the Mediterranean, showing city names like Madrid, Barcelona, and Lisbon. A red marker is placed on the map near Valencia.

En cambio, páginas como las de login, registro, cambio de email, cambio de contraseña y ciertos elementos de la página de inicio, conservan un diseño más cercano al Bootstrap nativo.



Iconografía

Para dar un mayor apoyo visual al usuario y mostrar un diseño más sencillo y limpio, se ha hecho uso de la librería de iconos “Font Awesome”, que dispone de una amplia variedad de iconos gratuitos.

Dichos iconos se han utilizado en múltiples sitios dentro de la aplicación, como son: el menú de navegación, los ajustes o las ventanas modales.





6.3.4 - Usabilidad

Para ofrecer una mejor experiencia al usuario más sencilla y cómoda, se han seguido diversas pautas de accesibilidad:

- **Utilizar un diseño responsive**

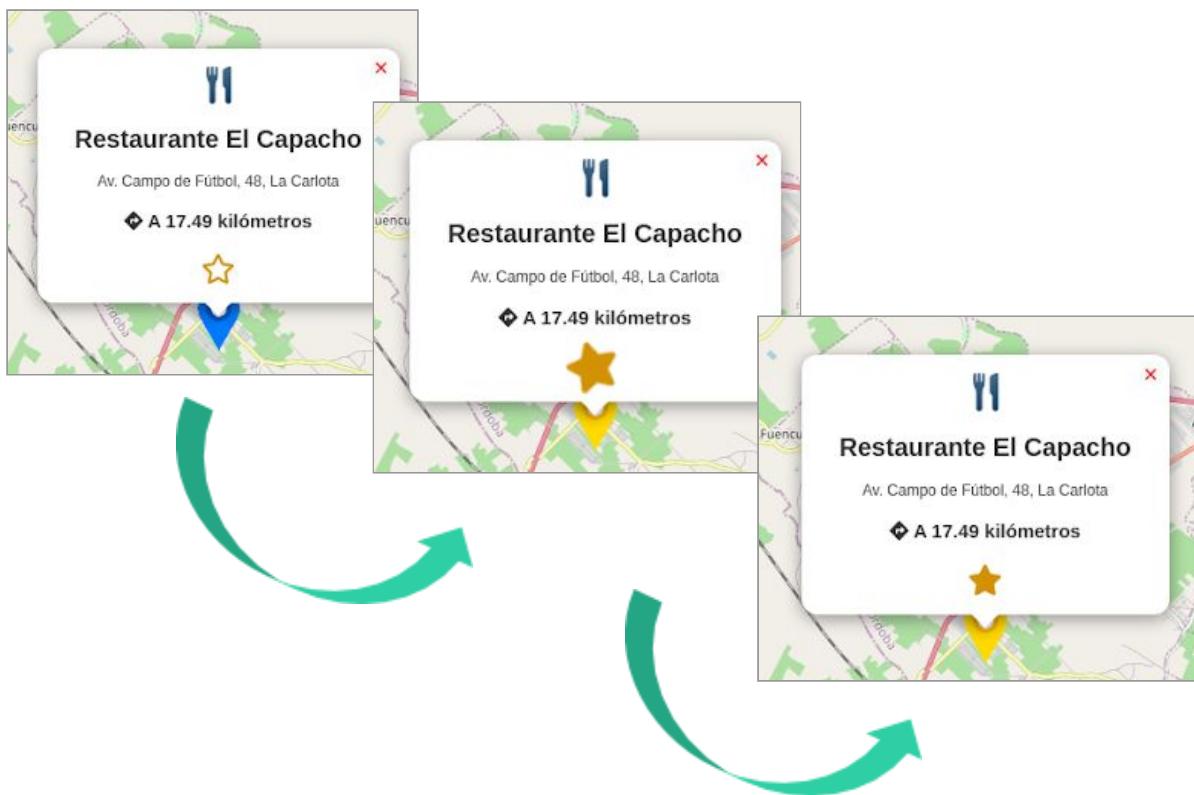
Al haber utilizado un diseño responsive en toda la aplicación, el usuario puede disponer de los botones, menús y otros elementos de interacción dispuestos de una forma más cómoda y adaptada según el tamaño de su pantalla, disposición, método de entrada, etc.

- **Botón de favoritos**

El botón mostrado en el mapa utilizado para añadir un nuevo sitio a favoritos es un gran ejemplo de usabilidad dentro de la aplicación. Al utilizar el icono de una estrella amarilla, el usuario lo relaciona rápidamente con el concepto “favorito”, “preferido” o “destacado”, lo que lo hace un **elemento muy intuitivo**.

La estrella rellena o vacía, es también otro ejemplo donde simplemente el aspecto del ícono, nos indica si está añadido o no a nuestros favoritos.

En este botón además, se muestra una animación donde de forma muy visual, el usuario puede apreciar el cambio entre estrella vacía o estrella rellena, lo que le asegura de que el cambio se ha realizado realmente.



- **Uso de vibración y sonidos**

Otra función de usabilidad relevante dentro de la aplicación es el uso de vibración y sonidos (en aquellos dispositivos que lo soportan) como respuesta a diferentes acciones del usuario, lo que supone una aclaración extra de que la acción que ha realizado, se ha llevado a cabo correctamente.

Algunos ejemplos donde se han utilizado son:

- Al pulsar sobre el botón de buscar de la página de Búsqueda, se recibe una pequeña vibración para informar al usuario que se ha pulsado el botón.
- Al mostrarse una ventana modal, se combina una vibración larga con un sonido, para alertar al usuario de que debe atender la alerta.
- Al añadir o eliminar un sitio a favoritos, también se hace uso de la vibración y de sonidos diferentes para distinguir una acción de otra.

- **Selección de iconos y colores**

Otro punto importante que ayuda a la accesibilidad, son los iconos y los colores utilizados. Por ejemplo, al hacer referencia a nuestros sitios favoritos siempre se utiliza el ícono de la estrella como símbolo representativo, y el

color amarillo, ya sea para los marcadores del mapa o los botones. Esto hace que el usuario siempre relacione “amarillo” o “estrella” con “sitios favoritos”.

También es importante el uso de los iconos y colores para remarcar avisos, errores o advertencias.

Un claro ejemplo es el menú de ajustes, donde la opción “Eliminar mi cuenta” resalta de un color rojo, indicando así al usuario que la acción tiene consecuencias. O durante el registro de un nuevo usuario, donde una zona de notificación aparece de color rojo si cometemos algún fallo, y de color verde si por ejemplo, el email que estamos escribiendo está disponible.

The screenshot shows a registration form titled "Registro de nuevo usuario". At the top is the SiteMe logo. Below it, there is a text input field containing "siteme@email.com". Directly beneath this field is a green button with the word "Disponible" (Available) written on it in white. The entire interface has a clean, modern design with a light gray background.

The screenshot shows a registration form titled "Registro de nuevo usuario". At the top is the SiteMe logo. Below it, there is a text input field containing "email.com". Directly beneath this field is a red error message box containing the text "El email no tiene un formato válido" (The email does not have a valid format). The entire interface has a clean, modern design with a light gray background.

- **Consistencia**

Por último, otro aspecto destacable de la usabilidad de la web es la consistencia. Todos las secciones y páginas de la aplicación utilizan la misma fuente tipográfica, mismos iconos, tonos de colores, etc para referirnos los mismos elementos, además, siempre se utiliza un único término para cada concepto.

Esto evita confusiones al usuario como: en qué parte de la aplicación se encuentra, qué función realiza un elemento, el significado de los colores o sonidos ...

Por ejemplo, si se utiliza un sonido para mostrar al usuario una ventana modal, se debe utilizar siempre el mismo sonido para la misma función. Si se utilizara ese sonido por ejemplo al eliminar un sitio del mapa, el usuario puede confundir una acción con la otra y no saber si realmente se ha llevado a cabo o no.

6.3.5 - Accesibilidad

Además de usable, también es importante que nuestra aplicación sea accesible, que cualquier tipo usuario pueda acceder a ella sin ningún tipo de problemas. Para conseguir esto, se han tenido en cuenta los puntos siguientes:

- **Diseño responsive**

El diseño responsive, además de usable, hace que sea más accesible, ya que se adapta a cientos de combinaciones de tamaños de pantallas en diferentes dispositivos, lo que la convierte en una aplicación universal.

- **Uso de texto alternativo en todas las imágenes**

El utilizar un texto alternativo para todas las imágenes de la web ofrece una mejora sustancial en la accesibilidad:

- Disponemos de un texto descriptivo de la imagen si ésta no puede cargarse por algún fallo de conexión.
- Ofrece una descripción extra para las personas con discapacidades visuales que requieran de un software de lectura para navegar por la web.
- Permite a los motores de búsqueda situar mejor la web.

- **Colores contrastados**

El ofrecer colores contrastados del fondo, como el verde con el blanco, permite que la aplicación sea más accesible, tanto en diferentes condiciones de iluminación (brillo de la pantalla por ejemplo), como en personas con discapacidades visuales, que no puedan distinguir correctamente tonalidades de colores.

Tests de Accesibilidad

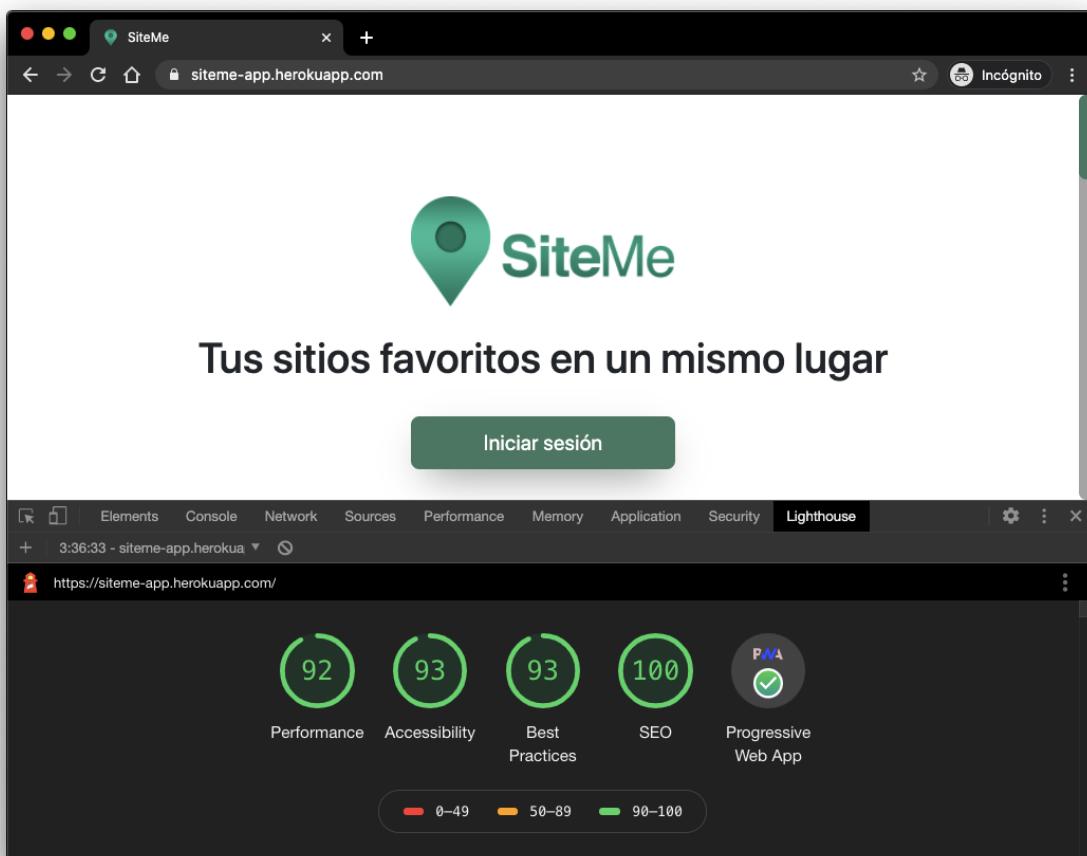
Para determinar el nivel de accesibilidad llevado a cabo en nuestra web, se han realizado algunos test de accesibilidad para comprobarlo.

En primer lugar, se ha llevado a cabo una auditoría a través de la herramienta gratuita Google Lighthouse que realiza un informe de los tiempos de carga, el posicionamiento SEO y la accesibilidad.

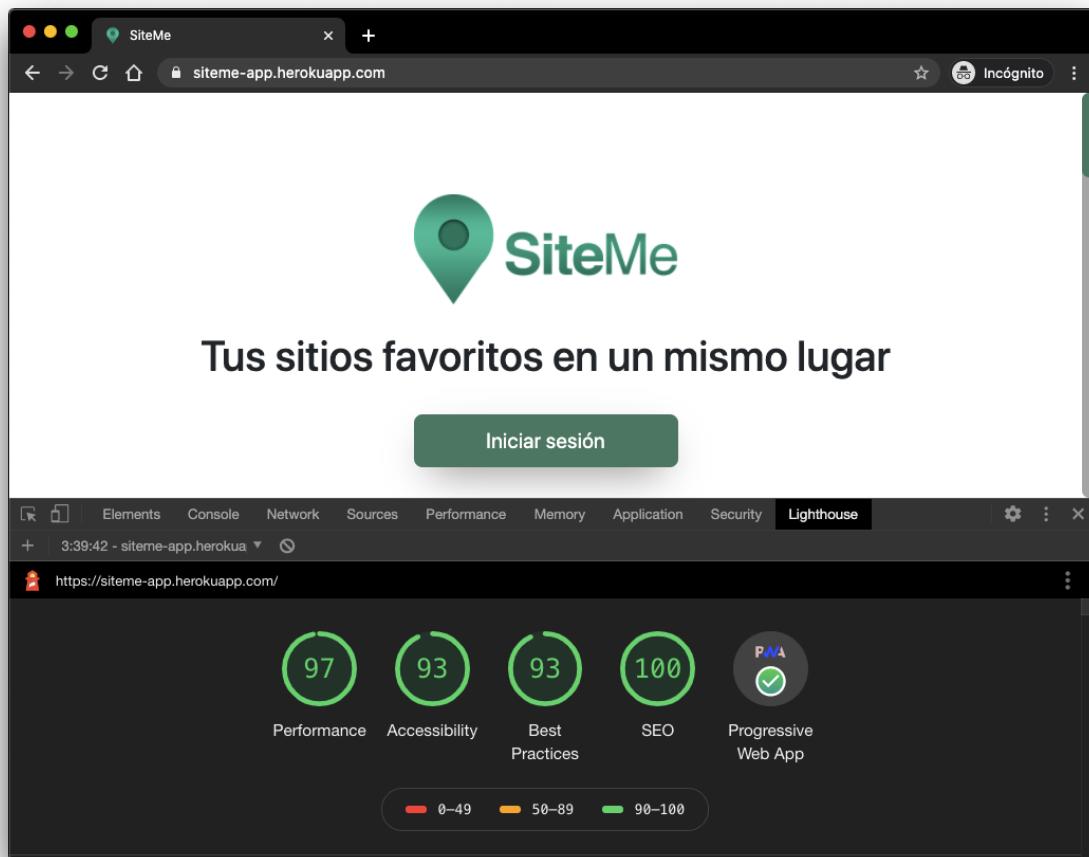
Analizando la página principal, obtenemos los siguientes resultados:

Las pruebas se han realizado sin extensiones adicionales (a través del modo incógnito de Chrome).

Para la versión de escritorio:



Para la versión móvil:



Como podemos observar, la aplicación aprueba con nota en la totalidad de las pruebas realizadas, superando el 90% en todos los casos, y obteniendo unos resultados algo superiores en la versión móvil.

Para contrastar estos datos, se ha realizado un test de sólo accesibilidad desde la web <http://examinator.ws/check/> cuyo resultados han sido los siguientes:

The screenshot shows a browser window with the title 'Informe - examiner'. The page displays a large score of 7.8 in a blue circle. Below the score, there is a summary of the analysis: URI: Entrada directa, Título: SiteMe, Elementos: 55, Tamaño: 4.4 KB (4465 bytes), and Fecha/Hora: 28/05/2020 - 11:16 GMT. A note at the bottom states 'Los resultados de la validación (X)HTML no están incluidos.' Below this, it says 'Resultados generales de 8 pruebas:' with categories: Excelente (6), Regular (1), Mal (1), and Tablero. Further down, it shows 'Se usan 4 elementos de encabezado' with a note 'G141: Organizar una página usando encabezados' and a score of 10. A note at the bottom right says 'Encabezados (h1~h6): 4'.

Se ha obtenido una **puntuación de 7,8 sobre 10**, una nota bastante positiva, destacando como punto negativo relevante la ausencia de enlaces () en la página, esto es debido a que, por limitaciones del framework de CSS se utilizan elementos <button> en lugar de <a> para acceder a otras partes de la página, por lo que el analizador no es capaz de determinar si realmente se puede navegar con ellos o no.

6.3.6 - Proceso de despliegue

Plataforma de despliegue

El proyecto ha sido desplegado en la plataforma gratuita Heroku, que es compatible con proyectos desarrollados con Symfony como es el caso de nuestra aplicación. Es gratuito siempre que el proyecto no supere los 500 Megabytes de capacidad.

Heroku nos proporciona un dominio (siempre con el subdominio *.herokuapp.com) gratuito con el que podemos acceder desde cualquier parte. En dicho dominio, hemos sincronizado nuestro proyecto, alojado en un repositorio de Github. El enlace

al repositorio de Github donde se aloja el código fuente de la aplicación es el siguiente:



<https://github.com/mfortea/SiteMe>

Y la dirección para acceder al despliegue de la aplicación en Heroku es esta:



<https://siteme-app.herokuapp.com/>

Entre otras opciones, Heroku nos permite desde su “dashboard” gestionar nuestro proyecto, podemos ejecutar comandos, reiniciar la aplicación, ver los logs, configurar certificados SSL, editar las variables de entorno...

Esto último nos permite realizar cambios en ellas en tiempo real, sin necesidad de volver a desplegar la aplicación, con ello podemos por ejemplo, cambiar a otra base de datos o editar la API Key proporcionada por la API de Google Maps de forma muy sencilla.

A screenshot of the Heroku dashboard showing the configuration variables (Config Vars) section. It displays five environment variables with their current values and edit/delete icons. The variables are: API_KEY (value: a colorful emoji string), APP_ENV (value: prod), DATABASE_URL (value: a colorful emoji string), TIPO_API (value: mock), and KEY (value: empty). There is also a 'Value' field and an 'Add' button.

Despliegue de la base de datos

Al haber utilizado una **base de datos MariaDB** (derivado de MySQL) se reducen las posibilidades de encontrar una base de datos online que sea gratuita y sin limitaciones.

Debido a esto, y a que la base de datos utilizada no contiene grandes cantidades de información, se ha desplegado de forma casera mediante un dispositivo Raspberry Pi (Modelo 3b +), con un servidor MariaDB versión 10.3 bajo el sistema operativo Raspbian (basado en Debian).

Servidor de base de datos

- Servidor: Localhost via [UNIX socket](#)
- Tipo de servidor: [MariaDB](#)
- Versión del servidor: [10.3.22-MariaDB-0+deb10u1 - Raspbian 10](#)
- Versión del protocolo: [10](#)
- Usuario: 
- Conjunto de caracteres del servidor: [UTF-8 Unicode \(utf8\)](#)

Servidor web

- Apache/[2.4.38 \(Raspbian\)](#)
- Versión del cliente de base de datos: [libmysql - mysqlnd 5.0.12-dev - 20150407 - \\$Id: b5c5906d452ec590732a93b051f3827e02749b83 \\$](#)
- extensión PHP: [mysqli](#)  [mbstring](#) 
- Versión de PHP: [7.0.33-0+deb9u7](#)

Para poder acceder desde el exterior se ha configurado una IP pública fija para la red, y se han abiertos los puertos utilizados por MariaDB para establecer la conexión (3306) redirigidos al servidor (la Raspberry). Dichos datos, se han introducido en la variable de entorno de Heroku que utiliza el proyecto para la conexión. (DATABASE_URL).

7 - Dificultades encontradas

Algunos aspectos del proyecto que han dificultado su desarrollo son los siguientes:

- **La limitación de la versión gratuita de la API de Google Maps Places.**

La API de Google Maps dispone de un sistema de cobro donde, según las peticiones que se realizan a la misma, se deberá pagar una cantidad proporcional. Al ser un proyecto pequeño y que, por el momento, no va a ser

lanzado al mercado, se optó por la versión de prueba gratuita de la API cuyas peticiones están limitadas a unas 1000 peticiones aproximadamente. Debido a esto, a la hora de testear la aplicación, se tenía el riesgo de superar este límite y que se produjese algún cobro.

Por ello, en el principio del desarrollo, se hicieron varias búsquedas de prueba para comprobar que la API respondía con los datos que se necesitaban, dichos datos se almacenaron en formato JSON, y a partir de ellos, se ha creado un “mock” (objeto simulado), que devuelve todas las respuesta como si se tratara de la API de Google Maps. De esta forma no se supera el límite de peticiones.

En algunas situaciones del desarrollo, debido a algunos cambios importantes que se han realizado, se ha habilitado la API de nuevo, y se ha ido comprobando que la comunicación continuaba funcionando correctamente.

- **Limitación de seguridad de los navegadores respecto a la geolocalización.**

Otro problema destacable durante el desarrollo, ha sido respecto a los permisos de geolocalización que se utilizan dentro de la aplicación para situar al usuario en el mapa.

La mayoría de navegadores actuales (Google Chrome, Mozilla Firefox, Apple Safari y Microsoft Edge), disponen de ciertas medidas de seguridad a la hora de permitir la geolocalización. Si la web a la que se accede no dispone de una conexión HTTPS segura, no se permite habilitar la ubicación de ninguna forma, excepto si el acceso se realiza desde el mismo equipo (localhost) (a excepción de Safari).

Esto ha impedido poder probar la aplicación completa en otros dispositivos de la misma red que no son el equipo que actúa de servidor de la aplicación (tales como smartphones, tablets, portátiles...), lo que limita las pruebas.

Como solución temporal antes del despliegue de la aplicación en un servidor seguro verificado (HTTPS), se ha desplegado el proyecto de forma local habilitando TLS desde Symfony, lo que ha permitido probar la aplicación en más dispositivos, aunque con algunas excepciones en algunos navegadores, debido a que el certificado que se ha utilizado no está verificado por ninguna CA (entidad certificadora).

- **Limitación con el sistema de rutas de Leaflet.**

La librería Leaflet que se ha utilizado para mostrar los mapas, dispone de un sistema de rutas llamado “Leaflet Routing Machine”. Haciendo llamadas a un backend de sistema de rutas, permite hacer cálculos de rutas entre dos puntos indicando de forma detallada el itinerario a seguir. Este sistema ha

sido probado y ha funcionado correctamente en el proyecto, pero el problema es que la mayoría de los backend puestos en línea requieren de suscripciones de pago por lo que no se ha implementado finalmente.

Para probar este sistema se ha utilizado el OSRM (Open Source Routing Machine), del que hay implementaciones gratuitas online, pero son demos muy inestables que tienen un límite de peticiones muy bajo, por lo que no es recomendable utilizarlo en una aplicación final.

8 - Conclusiones personales

Con este proyecto, he podido conocer de primera mano la implicación que necesita el desarrollo completo de una aplicación web: la planificación requerida, los numerosos cambios que se realizan, el amplio abanico de alternativas que existen o la continua evolución de las herramientas y librerías implicadas en su desarrollo.

El uso de tecnologías como la de búsqueda de sitios de Google Maps Places o los diferentes tipos de mapas que se han incorporado han sido puntos claves en el desarrollo de esta aplicación, que dejan claro que el uso de servicios de APIs de terceros potencia cualquier pequeño proyecto que necesite de nuevas funcionalidades fáciles, rápidas y seguras.

Aunque lamentablemente, dichos servicios dejan cada vez menos alternativas libres, con código abierto y de acceso universal para pequeños proyectos o de ámbito educativo, al necesitar para la gran mayoría, una suscripción de pago, que en cierta medida, ha dificultado el desarrollo de este proyecto.

Otro ámbito que he podido conocer más a fondo es el de la seguridad. Actualmente, los navegadores siguen estrictos estándares seguros que hacen que cualquier proyecto necesite de los mecanismos de seguridad necesarios para poder ser desplegados en internet de forma confiable. Esto ha dado resultado en una dificultad algo mayor en el uso y testeo de ciertas características en algunos navegadores web, como la geolocalización, la vibración o la reproducción de sonidos dentro de la aplicación, que suelen ser funciones altamente restringidas por los navegadores si no se cumplen con los requisitos de seguridad necesarios.

Por otro lado, gracias a sistemas como el de Symfony Security, he podido solventar dicho problema haciendo uso de herramientas de seguridad como: mecanismos de encriptación de claves, login seguros, uso de roles dentro de la aplicación o la habilitar el acceso seguro sobre TLS de forma local.

Como conclusión, pienso que las aplicaciones web son una gran alternativa frente las aplicaciones nativas; ofrecen una gran versatilidad, escalabilidad y rendimiento sin necesidad de centrarse en una plataforma concreta; con el desarrollo del

proyecto de una sola aplicación web, puedes llegar a millones de dispositivos diferentes. Aunque, si bien es cierto, cuentan con ciertas limitaciones con el uso de diversos tipos de sensores y funciones nativas de los dispositivos, creo que si continúan su evolución al ritmo actual, serán el futuro más próximo del mercado de las apps.

9 - Posibles mejoras

Tras concluir el desarrollo, podemos determinar una serie de posibles mejoras que podrían incluirse en la aplicación para hacerla más completa:

- **Usar autenticación mediante OAuth.**

Una posible mejora podría ser utilizar el sistema de autorización OAuth, que permite iniciar sesión en nuestra aplicación desde un servidor de autorización de acceso externo, esto se traduce en que podríamos iniciar sesión en nuestra aplicación mediante una cuenta de Google, de Facebook, Instagram y otras muchas redes sociales compatibles.

- **Calcular rutas entre los sitios.**

Una forma de añadirle más utilidad a SiteMe, sería poder habilitar de forma completa el sistema de rutas mencionado anteriormente, que nos permitiría obtener las indicaciones para llegar hasta un sitio del mapa que tengamos guardado en favoritos sin necesidad de hacer uso de otras apps.

- **Mostrar una mayor información sobre los sitios.**

Haciendo uso de nuevas APIs y aprovechando la totalidad de datos que nos ofrece la API de Google Maps Places, se podría añadir una mayor cantidad de información sobre los sitios, tales como horarios de apertura, opiniones, valoraciones, contacto, etc.

- **Opciones de personalización para el usuario.**

Para dotar al usuario de una experiencia más personalizada, se podrían añadir nuevas opciones ajustables como por ejemplo:

- Poder activar/desactivar sonidos y vibraciones.
- Ofrecer diferentes estilos de marcadores (con diferentes colores, formas...).
- Poder clasificar los sitios por categorías: restaurantes, bares, supermercados, bancos ...

- Posibilidad de habilitar un “Modo oscuro”.
- **Una opción para compartir**

Por último, sería interesante añadir un botón “compartir” que permitiera a los usuarios recomendar sitios a sus amigos y familiares (mediante URL, email, redes sociales, etcétera), o en forma de mensaje entre usuarios de SiteMe, lo que habilitaría una bandeja de entrada o zona de notificaciones dentro de la aplicación.

10 - Fuentes de información

10.1 - Webgrafía

Principales características de Symfony:

<https://www.coriaweb.hosting/symfony-principales-caracteristicas/>

Principales características de Laravel:

<https://www.techcronus.com/blog/go-php-laravel-framework/>

Componentes de Symfony utilizados en Laravel:

<https://rimorsoft.com/componentes-symfony-en-laravel>

Principales características de Spring Framework:

<https://dzone.com/articles/why-spring-framework-is-popular>

Principales características de Java Server Faces:

<https://www.educba.com/what-is-jsf/>

Principales características de Express JS:

<http://www.softxml.com/3003/Why-Are-Express-JS-Developers-Valuable?>

Qué es un middleware en Express JS:

<https://medium.com/@aarnlpezsosa/middleware-en-express-js-5ef947d668b>

Frameworks de frontend alternativos a Bootstrap:

<https://guiadev.com/alternativas-a-bootstrap/>

Características de Bootstrap:

<https://www.sitesbay.com/bootstrap/bootstrap-features-of-bootstrap>

Características de MaterializeCSS:

<https://ampersandacademy.com/tutorials/materialize-css/materialize-css-features>

Características de PureCSS:

<https://www.javatpoint.com/what-is-pure-css>

Características de Angular:

<https://angular.io/features>

Qué es React JS:

<https://es.reactjs.org/tutorial/tutorial.html#what-is-react>

Características de Vue:

<https://codingpotions.com/que-es-vue>

Características de Svelte:

<https://sumatd.com/blog/svelte-caracteristicas/>

Información sobre las APIs de la Google Maps Platform:

<https://cloud.google.com/maps-platform?hl=es>

Comparativa de librerías para GIS:

<https://mappinggis.com/2015/03/las-mejores-apis-javascript-para-webmapping/>

Documentación sobre el funcionamiento de las APIs de Google Maps:

https://developers.google.com/maps/documentation/?hl=es&_ga=2.144280885.1047118153.1585686087-1834421712.1585490101

Cómo crear un mapa con Leaflet y OpenStreetMaps:

<https://asmaloney.com/2014/01/code/creating-an-interactive-map-with-leaflet-and-opensestreetmap/>

Documentación sobre MariaDB

<https://www.hostinglatam.cl/caracteristicas-de-mariadb-un-proyecto-derivado-de-mySQL/>

Documentación sobre MongoDB

<http://www.manualweb.net/mongodb/que-es-mongodb/>

Documentación sobre Oracle Database

<https://www.netec.com/que-es-oracle>

Información de uso de la librería Micromodal.js

<https://micromodal.now.sh/#usage>

Documentación de Symfony para su sistema de seguridad.

<https://symfony.com/doc/current/security.html>

Documentación de Symfony para crear datos de prueba con Doctrine.

<https://symfony.com/doc/current/testing/database.html#doctrine-fixtures>

Documentación de Symfony para la creación de un formulario de Login.

https://symfony.com/doc/current/security/form_login_setup.html

Documentación de uso de los iconos de FontAwesome

<https://fontawesome.com/how-to-use/on-the-web/referencing-icons/basic-use>

Documentación del sistema de rutas de Leaflet

<https://www.liedman.net/leaflet-routing-machine/>

Estructura de directorios en Symfony 4

<https://symfony.es/noticias/2017/04/10/symfony-4-estructura-de-directorios/>

Guía de estilo de PSR-2 para PHP

<https://www.php-fig.org/psr/psr-2/>

Ejemplos de código utilizando la guía de estilo PSR-2

<https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-1-basic-coding-standard.md>

Formateador de código PHP utilizado en el proyecto

<https://marketplace.visualstudio.com/items?itemName=kokorin.vscode-phpfmt>

Formateador de código JavaScript utilizado en el proyecto

<https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode>

Formateador para plantillas Twig utilizado en el proyecto

<https://marketplace.visualstudio.com/items?itemName=mblode.twig-language-2>

Guía sobre la tipografía en aplicaciones de Windows (Segoe UI)

<https://docs.microsoft.com/es-es/windows/uwp/design/style/typography>

Códigos de respuesta HTTP

<https://developer.mozilla.org/es/docs/Web/HTTP/Status>

Herramienta de auditoría Google Lighthouse

<https://developers.google.com/web/tools/lighthouse>

10.2 - Recursos utilizados

Iconos de los marcadores de ubicación de la aplicación

https://www.iconfinder.com/icons/299087/map_marker_icon.

Sujetos a una licencia CC BY 3.0. Han sido modificados para su adaptación dentro de la aplicación.

Mockup iPhone 11

<https://www.freepng.es/png-r5ny13/>

Según el autor, se permite su utilización con fines editoriales, personales o educativos.

Sonido de notificación por defecto

<https://notificationsounds.com/notification-sounds/just-maybe-577>

Sujetos a una licencia CC BY 4.0

Sonido de notificación al añadir un sitio a favoritos

<https://notificationsounds.com/notification-sounds/definite-555>

Sujetos a una licencia CC BY 4.0

Sonido de notificación al eliminar un sitio a favoritos

<https://notificationsounds.com/notification-sounds/case-closed-531>

Sujetos a una licencia CC BY 4.0