UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Desarrollo de una aplicación web para la visualización geo-posicionada de capas de servicios administrativos para la Oficina de Transparencia de la CARM

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA TELEMÁTICA

Autor: Ignacio Ballesta Giménez

Director: Esteban Egea López

Director externo: Ignacio Ballesta Germán

Cartagena, 6 de julio de 2021

Índice de contenidos

1. Introducción 2
2. Tecnologías empleadas 3
3. Desarrollo del proyecto 5
   3.1. Funcionamiento de la aplicación 5
   3.2. *Back-end* (PHP, Twig y HTML) 9
      3.2.1. Implementación del controlador PHP (Symfony) 9
      3.2.2. Generación de la vista HTML con Twig 11
   3.3. *Front-end* (JavaScript y CSS) 12
      3.3.1. Implementación JavaScript (Leaflet) 12
         3.3.1.1. Inicialización del mapa y sus variables de control 12
         3.3.1.2. Implementación de las funcionalidades del mapa 16
            3.3.1.2.1. Funcionalidad para seleccionar área 17
            3.3.1.2.2. Filtrado dinámico de capas 20
            3.3.1.2.3. Panel lateral para informar de las ubicaciones contenidas 21
            3.3.1.2.4. Enlace para obtener toda la información disponible de una ubicación 22
            3.3.1.2.5. Funcionalidad para generar informe 23
      3.3.2. Apariencia y estilo con CSS (Bootstrap) 24
4. Conclusiones y líneas futuras 25
   4.1. Ejemplo de uso de la aplicación 25
      4.1.1. Vista de la aplicación principal 25
      4.1.2. Vista inicial del mapa 26
      4.1.3. Vista focalizada del mapa 27
      4.1.4. *Popup* de un marcador 29
      4.1.5. Vista de la información completa de una ubicación 29
      4.1.6. Dibujo de superficies 30
      4.1.7. Generación de informe 31
   4.2. Conclusiones 31
   4.3. Líneas de desarrollo futuras 32
5. Bibliografía y referencias 34

1. Introducción

La cultura de datos abiertos (más conocida por su anglicismo *Open Data*) pretende poner a disposición de la ciudadanía los datos gestionados por las administraciones públicas en formato reutilizable para que pueda generar valor económico y social a ciudadanos, empresas y otras administraciones.

La Oficina de Transparencia y Participación Ciudadana de la Comunidad Autónoma de la Región de Murcia (OTPC) es un órgano administrativo dependiente de la Dirección General de Gobierno Abierto y Cooperación. Destacan entre sus funciones, con carácter transversal, la transparencia, la participación ciudadana en la vida pública y la política de datos abiertos.

En materia de datos abiertos le corresponde, en particular, el ejercicio de la coordinación y supervisión de la política en materia de datos abiertos en la Administración Regional, el impulso del acceso abierto a la documentación de carácter científico-técnico elaborada o encargada por la misma, así como las actividades de fomento de los datos abiertos en el conjunto de la Administraciones Públicas de la Región de Murcia.

Dentro de este contexto, la OTPC ha comenzado un proyecto con el fin de ofrecer a la ciudadanía una herramienta que permita la visualización y, en cierta medida, el análisis de datos referentes a servicios administrativos. Dicha herramienta consiste en una aplicación web cuya primera y principal funcionalidad es leer dinámicamente datos publicados en diferentes portales de datos abiertos (a nivel mundial) con el fin de permitir a los usuarios tanto el acceso como la posibilidad de agruparlos según su criterio en lo que se denominarán capas de datos.

Dado que esta aplicación tiene diversas funcionalidades pero no es en sí misma el objetivo de este Trabajo de Fin de Grado (TFG), a lo largo de esta memoria únicamente se hará referencia a ella cuando se considere imprescindible, ya sea para aportar contexto o para justificar alguna decisión tomada durante el desarrollo del Trabajo.

A partir de este entorno, se plantea la necesidad de crear una aplicación adicional que permita la visualización e interacción geográfica de los datos ya

seleccionados mediante la aplicación principal. Esta segunda aplicación, cuyo desarrollo es el objetivo de este Trabajo, puede considerarse como un módulo de la aplicación principal, dado que, como se verá más adelante en esta memoria, se trata de una aplicación que depende enteramente de los datos recibidos de la aplicación principal.

Para concluir esta introducción, cabe señalar algunos de los principales objetivos de este Trabajo:

- En primer lugar, como ya se ha mencionado, la visualización geográfica (esto es, mediante un mapa) de los datos preseleccionados por el usuario (los cuales provienen de los portales de datos abiertos).
- En segundo lugar, la implementación de ciertas funcionalidades que permitan al usuario modificar (aunque sea levemente) la forma en la que se visualizan los datos, así como la navegación de estos.
- Por último, siendo este quizá el apartado con mayor complejidad del proyecto, ofrecer una herramienta al usuario para poder, mediante el dibujo de superficies sobre el mapa, generar un informe con datos y estadísticas útiles sobre los puntos contenidos en dichas superficies.

Por último, procede destacar que esta aplicación puede experimentar una evolución considerable a partir del desarrollo realizado y expuesto para este TFG. Pese a que esta memoria solamente expondrá esto último, al final de esta se incluirán ideas más o menos concretas relativas a posibles líneas de desarrollo futuras.

2. Tecnologías empleadas

En este apartado se detallarán las diferentes tecnologías de las que se ha hecho uso en el desarrollo de este proyecto:

- **Symfony** [1]: un *framework* PHP empleado para la implementación de toda la lógica de servidor (también conocida como *back-end*). Esta herramienta, como la mayoría en su terreno, sigue el patrón de arquitectura software MVC (Modelo Vista Controlador). Concretamente,

los controladores se implementan mediante clases PHP y toda su lógica se encapsula en métodos de dichas clases (funciones). Tanto estos últimos como cualquier controlador a nivel general pueden ser mapeados a rutas específicas para su llamada con HTTP. En lo referente a las vistas, Symfony [1] ofrece un lenguaje ad-hoc para permitir parametrizar el contenido HTML generado: Twig. Este lenguaje aporta la posibilidad de implementar estructuras de control, bucles y uso de variables locales (entre otras utilidades) para dinamizar la generación de la vista HTML que los controladores devuelven como respuesta HTTP a los clientes (aunque las respuestas no tienen por qué ser necesariamente de este tipo de contenido).

- **Leaflet** [2]: una librería JavaScript para el tratamiento de datos geográficos y la generación del mapa donde se posicionan las ubicaciones contenidas en dichos datos. Toda la lógica *front-end* de la aplicación (implementada con JavaScript) es dependiente en gran medida de objetos y/o métodos de Leaflet [2], ya que se relaciona de alguna manera con el mapa de la vista. Pese a que una parte del código JavaScript desarrollado se integra en la vista HTML (con etiquetas *script*) principalmente por conveniencia, la mayor parte de este es cargado por el navegador a partir de un fichero externo.

  Asimismo, para añadir ciertas funcionalidades al mapa (a detallar en el siguiente apartado) se ha hecho uso de diversos plugins de Leaflet [2]. Estos han sido desarrollados y publicados por diferentes desarrolladores y/o entidades y, al igual que Leaflet [2], son de libre uso.
- **Bootstrap** [3]: una librería CSS para la configuración del estilo de la vista. Para esta aplicación en particular, se han empleado versiones de esta librería publicadas de forma abierta en la plataforma Bootswatch [4] que aportan temáticas basadas en diversos grupos de colores (claros, oscuros…). El uso de esta librería ha sido especialmente útil para la generación y el diseño de botones, paneles o diálogos (entre otros) cuya funcionalidad ha sido implementada con JavaScript.

3. Desarrollo del proyecto

Lo primero a tratar en este punto es, sin duda, el funcionamiento de la aplicación; tanto a nivel individual como a nivel comunicativo con la aplicación principal (la cual le provee de los datos necesarios). Para ello, en el próximo subapartado pasará a explicarse, con el debido detenimiento, el papel que desempeña cada una de las partes involucradas y cómo, de manera coordinada, todas estas partes aportan la funcionalidad que se busca para esta aplicación.

En lo concerniente al desarrollo del proyecto, al tratarse de una aplicación web, es conveniente hacer una primera distinción entre dos partes de esta: la parte de *back-end* y la parte de *front-end*. La primera de ellas abarca todo lo relativo a la lógica de servidor, lo que en este caso incluye tanto la implementación del controlador en Symfony [1] (mediante una clase PHP y sus funciones) como la programación de la vista (trabajando para ello con los lenguajes HTML y Twig). La segunda de dichas partes, a su vez, engloba toda la implementación del cliente web (usando para ello fundamentalmente JavaScript). Más adelante, se tratará con más detalle cada una de las partes ya mencionadas y, para cada una de ellas, se enunciarán y expondrán las distintas herramientas de las que se ha hecho uso.

3.1. Funcionamiento de la aplicación

En este apartado se proporcionará una explicación concreta acerca del funcionamiento conjunto de todas y cada una de las partes que conforman la aplicación, así como una descripción específica del funcionamiento individual de cada una de ellas. Para ello, en primer lugar se mostrará un diagrama representativo de dichas partes y las interacciones entre ellas, a partir del cual se explicará cada una paso a paso.

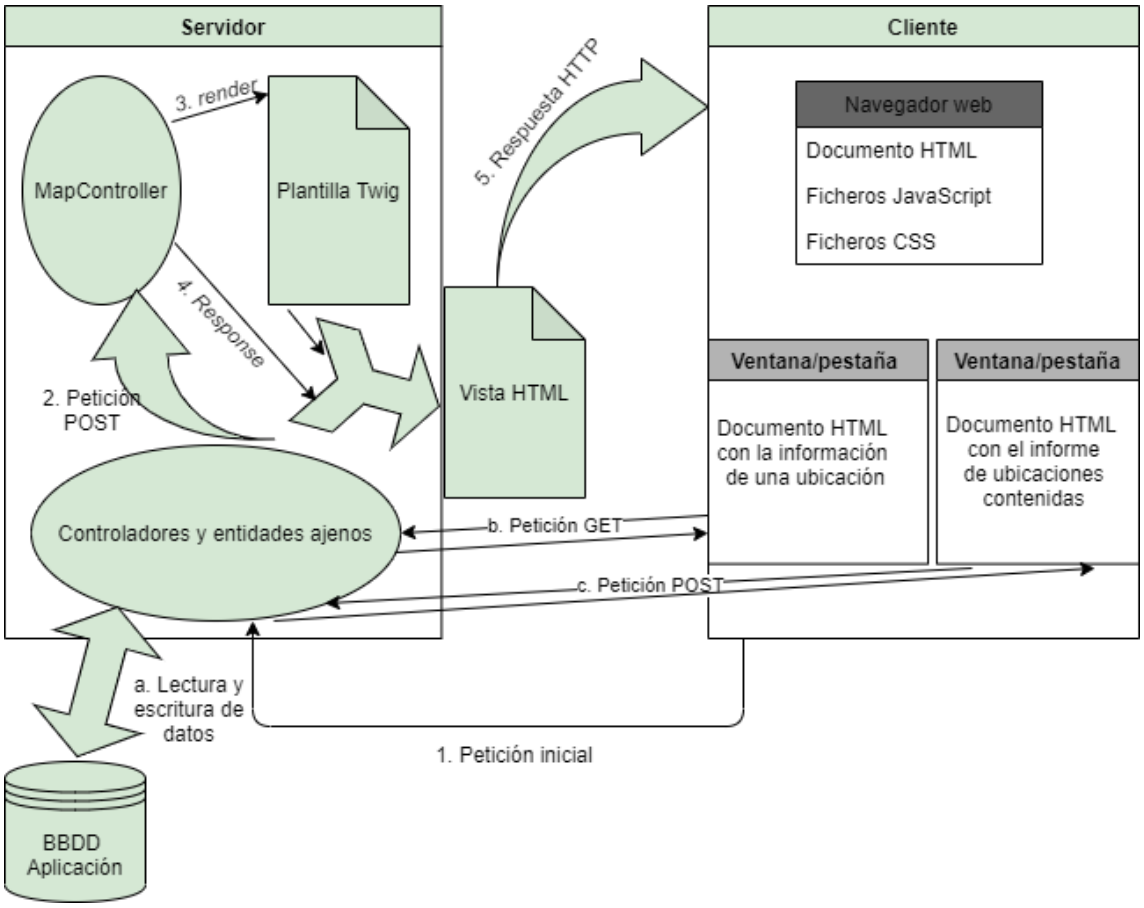

Figura 1: Diagrama del funcionamiento general de la aplicación (elaboración propia)

Como se puede apreciar en el diagrama, la mayoría de las interacciones indicadas en él aparecen numeradas; es el orden que sigue el flujo de aplicación, y es el que se va a seguir para explicar, uno a uno, los elementos involucrados en este. En lo que respecta a las interacciones marcadas con una letra, son relativamente independientes del flujo y se tratarán más adelante.

1. **Petición HTTP inicial**: esta es la interacción necesaria para iniciar todo el flujo de la aplicación; el cliente web, dentro de la aplicación principal, solicita visualizar determinadas capas de datos en el mapa. Nótese que esta petición al servidor no se dirige directamente al controlador PHP del mapa, sino que uno de los controladores de la aplicación principal es quien atiende dicha petición.
2. **Petición POST**: una vez el controlador anteriormente mencionado recibe la petición del cliente, determina los datos a enviar al controlador del mapa y se los hace llegar por medio de una petición POST (se trata de una redirección interna entre controladores, ambos alojados en el mismo

servidor); los datos en cuestión se incluyen en el cuerpo de dicha petición en formato JSON, y especifican tanto las capas de datos que se han de representar como las ubicaciones pertenecientes a cada una de ellas, además de otros parámetros de control.

3. **Llamada a la función *render***: el *MapController*, una vez recibidos y procesados los datos contenidos en la petición POST, procede a renderizar la vista HTML (esto es, interpretar el código Twig dentro de la plantilla especificada para así generar el contenido HTML resultante). Más adelante (véase Implementación del controlador PHP (Symfony)) se proporcionará una explicación algo más detallada acerca del uso de esta función PHP de Symfony [1].
4. **Construcción del objeto *Response***: dado que es el controlador PHP el que ejecuta la llamada a la función *render*, una vez este proceso ha concluido la vista HTML obtenida se devuelve al primero. Dicha vista se encapsula directamente dentro de un objeto *Response* de Symfony [1], el cual puede emplearse a nivel PHP para introducir un punto de retorno de una función. Este punto de retorno, extrapolado al contexto HTTP, implica el envío de una respuesta.
5. **Envío de respuesta HTTP**: este punto supone la respuesta dada al cliente a su petición inicial (primer evento listado). El contenido de dicha respuesta no es más que un documento HTML a interpretar por parte del cliente web (el navegador).

A partir de aquí, el usuario ya dispone en su navegador de la vista del mapa. Cuando más adelante (apartado Implementación JavaScript (Leaflet)) se explique la implementación de *front-end* (JavaScript), se distinguirá entre una primera fase inicial y otra que aporta las funcionalidades del mapa. La vista que el cliente tiene en este instante se corresponde con la primera de esas fases y, por medio de los ficheros JavaScript y CSS referenciados en esta para ser cargados por el navegador, se incorporará seguidamente la segunda fase de la implementación *front-end*.

Para concluir este apartado, han de explicarse aquellas interacciones del diagrama que no han sido incluidas como tal en el flujo de aplicación anteriormente descrito:

a. **Lectura y escritura en la base de datos**: la aplicación principal cuenta con una base de datos propia donde, entre otros, se almacenan datos relativos a los usuarios de la aplicación (para autenticación, por ejemplo) y a los portales de datos abiertos. Concretamente, centrando la cuestión en lo que compete a la aplicación del TFG, resulta imprescindible mantener un control de los datos que se cargan de diferentes portales mediante la aplicación principal, y estos deben ser inequívocamente identificables. Esto se debe a que la implementación JavaScript que permite obtener en cualquier momento todos los datos relativos a una ubicación cualquiera del mapa (se explicará en el apartado Enlace para obtener toda la información disponible de una ubicación) no es factible si la aplicación principal, a la cual se llama desde el cliente con este fin, no es capaz de identificar de alguna manera una ubicación específica. Es por esto que la base de datos cuenta con una tabla en la que se mantiene un registro de las URLs de los ficheros que han sido cargados en la aplicación principal (téngase en cuenta que todo dato pasado al mapa ha debido de ser obtenido antes de un fichero alojado en un portal de datos, por lo que conviene controlar mediante esta tabla todos los ficheros que se hayan leído con la aplicación principal). Es importante resaltar que en ningún caso se almacenan ficheros en la base de datos, únicamente sus URLs asociadas a un identificador único. Asimismo, para cada uno de estos ficheros las ubicaciones detalladas en estos se referencian también con su propio identificador único (dentro del contexto del fichero), que no es otro que el número de la línea del fichero en la que se encuentra la información de la ubicación en cuestión. Los identificadores de las ubicaciones leídas no se almacenan en base de datos, solamente se generan para poder ser enviados al controlador del mapa junto con los identificadores de los ficheros involucrados (en la estructura JSON de la petición POST). Teniendo presente esta explicación, puede pasarse a explicar las otras dos interacciones restantes.
b. **Petición GET para obtener la información de una ubicación**: tanto esta como la siguiente interacción que se detalla se originan siempre en el lado del cliente (por elección del usuario). Para obtener la información al completo de una ubicación cualquiera, el cliente envía una petición GET

a un controlador PHP de la aplicación principal con dos parámetros: los identificadores de fichero y ubicación (el número de línea, este último). De esta manera el controlador llamado, mediante una consulta a la base de datos, obtiene la URL que se precisa para leer remotamente el fichero necesario y, más específicamente, la línea dada por el segundo parámetro de la petición. Una vez obtenida la información, se presenta en un formato HTML y se devuelve al cliente (el cual la muestra en una nueva ventana, como indica el diagrama). La funcionalidad descrita en este punto se explica en el apartado Enlace para obtener toda la información disponible de una ubicación.

c. **Petición POST para solicitar el informe de ubicaciones contenidas**: esta funcionalidad, también explicada posteriormente en su correspondiente apartado (Funcionalidad para generar informe), es ofrecida (en lo referente al lado de servidor) por un controlador PHP de la aplicación principal. Este se limita a recibir y procesar los datos de las ubicaciones contenidas en las superficies que se hubieran dibujado (siempre en referencia a la capa a la que pertenecen) y, al igual que la funcionalidad descrita en el punto anterior, devuelve el informe como un documento HTML (que también es mostrado en una nueva ventana del cliente). Los datos enviados en el cuerpo de la petición POST siguen la estructura JSON.

En los próximos apartados se explicarán con más detalle los distintos elementos de la aplicación que se han mencionado en este apartado.

3.2. *Back-end* (PHP, Twig y HTML)

3.2.1. Implementación del controlador PHP (Symfony)

Como ya se ha mencionado, el controlador PHP implementado (*MapController*) es responsable de atender las peticiones HTTP recibidas desde la aplicación principal y, acto seguido, generar la respuesta HTTP que proceda (a partir de la renderización de una plantilla Twig). Dicho controlador, implementado como una clase PHP, tiene una serie de métodos que pueden ser mapeados a diferentes rutas para ser llamadas a través de una petición HTTP. Además, pueden especificarse los parámetros a esperar en la URL (útil para las peticiones GET,

por ejemplo) e incluso pueden especificarse los tipos de petición permitidos para llamar a un método en concreto.

En el caso de *MapController*, la mayoría de los métodos implementados no son de elaboración propia, puesto que competen a la aplicación principal (son las funciones PHP que garantizan el enlace entre esta última y la aplicación del TFG). No obstante, el controlador tiene dos métodos de elaboración propia que, como es lógico, tienen su papel tanto en el procesado de los datos recibidos en la petición como en la generación de la vista (esto es, llamar a la función *render* de Symfony pasándole los parámetros a emplear con Twig).

Concretamente, el primero de estos métodos se emplea localmente para procesar y reestructurar los datos recibidos en la petición HTTP. Esta reestructuración se hace en un array asociativo que es lo que posteriormente se pasa a la vista (llamando al método *render*, como ya se ha dicho). Conviene señalar, también, que esta función PHP realiza asimismo una validación de los datos recibidos, principalmente en lo respectivo al formato de estos (por ejemplo, se comprueba que los valores de latitud y longitud se encuentren dentro de los rangos apropiados).

Por otro lado, el otro método implementado es el responsable de extraer en un primer lugar los datos del cuerpo de la petición POST (el único tipo de petición contemplado para ejecutar esta función) y, a continuación, llamar al método anteriormente explicado para obtener el array asociativo con los datos ya procesados. Por último, esta función genera la respuesta HTTP a la petición del cliente mediante la generación de la vista HTML, para lo cual se renderiza una plantilla Twig. Esta plantilla trabaja con los datos recibidos en la llamada a *render*, los cuales se referencian con nombres de variables (dicha función recibe dos parámetros de entrada, la URL relativa de la plantilla a cargar y un array asociativo donde las claves son los nombres de las variables para Twig y los valores los elementos a asociar con dichas variables). Esta renderización de la plantilla Twig, que resulta en una vista HTML, se encapsula en un objeto genérico *Response* de Symfony [1] que no es más que la respuesta HTTP que espera el cliente a su petición inicial.

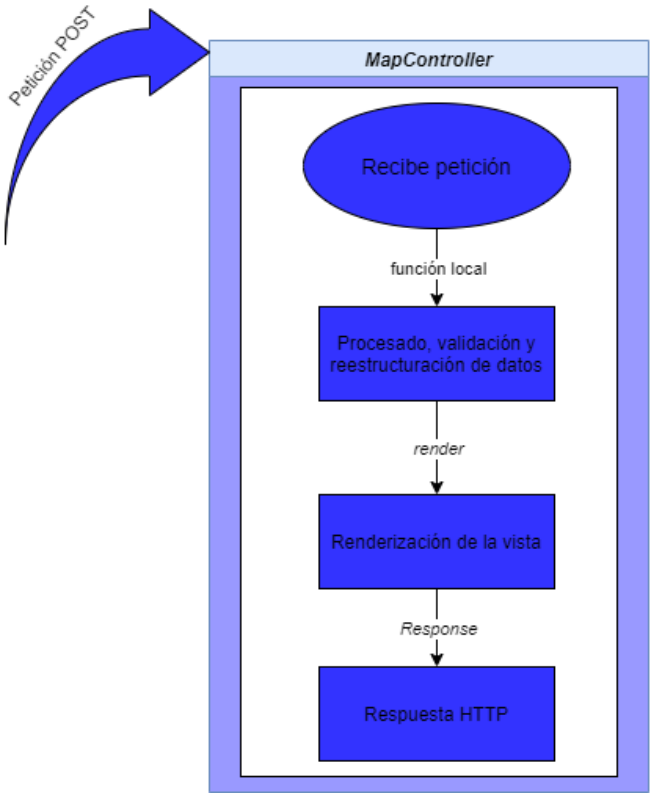

Figura 2: Diagrama del controlador PHP MapController

3.2.2. Generación de la vista HTML con Twig

Como se acaba de explicar en el punto anterior, la generación de la vista HTML se lleva a cabo mediante la renderización de una plantilla Twig de Symfony [1]. Cada plantilla está compuesta esencialmente por código HTML y estructuras Twig que aportan, como se ha mencionado, el uso de variables locales, estructuras de control y bucles iterativos, principalmente. Asimismo, cabe destacar que Twig proporciona una considerable variedad de funciones para el tratamiento de los datos recibidos y/o el contenido HTML que se genera.

Una funcionalidad muy relevante de Twig que se emplea en la implementación que se está describiendo es la posibilidad de crear una plantilla base, la cual puede ser extendida por cualquier otra plantilla. Dado que todo el código (tanto HTML como Twig) de las vistas debe agruparse y estructurarse en bloques Twig, disponer de una vista base donde determinar los nombres y la jerarquía de los bloques a implementar resulta de gran ayuda. De igual manera, si en un futuro se quisieran desarrollar distintas vistas con la misma base (al menos en lo referente a la apariencia), sería tan simple como crear una nueva plantilla Twig que extienda en primer lugar la plantilla base ya existente.

Así se ha hecho en este proyecto; existe una plantilla base de Twig que define la estructura de bloques a seguir y, además, contiene el código HTML correspondiente a los elementos básicos de la vista de la aplicación de este TFG. Esta plantilla es extendida por la plantilla Twig específica para codificar la vista relativa al mapa y a diferentes elementos para la interfaz de usuario. Además, pese a que no es objeto de explicación en este apartado, cabe mencionar que esta vista incluye también código JavaScript (añadido mediante etiquetas *script*) para la inicialización del mapa (usando Leaflet [2]), así como la creación de diversas variables de control para su posterior uso. Esto se describirá con más detalle en el apartado de *front-end* (Front-end (JavaScript y CSS)).

La vista generada conjuntamente a partir de la vista base y la vista del mapa se devuelve al controlador en formato HTML y, como ya se ha dicho, este se ocupa de enviarla al cliente como respuesta HTTP.

3.3. *Front-end* (JavaScript y CSS)

3.3.1. Implementación JavaScript (Leaflet)

JavaScript ha sido, con diferencia, el lenguaje más empleado en el desarrollo de este TFG, dado que la mayor parte de la lógica a implementar pertenece al lado del cliente web y, más concretamente, al elemento del mapa controlado con Leaflet [2] (librería JavaScript).

En primer lugar, conviene señalar que toda la implementación realizada puede dividirse en dos partes o “etapas”:

- Una primera fase de inicialización del objeto del mapa con Leaflet y sus complementos (se detallarán más adelante) y, de la misma manera, determinados elementos HTML y variables de control cuyos atributos dependen en alguna medida de los datos del mapa.
- La segunda etapa, por su parte, abarca toda la lógica que se ejecuta a posteriori, generalmente en respuesta a un evento o una acción del usuario, para alterar y/o recabar datos del mapa y los elementos HTML circundantes (los cuales muestran información relativa al mapa).

3.3.1.1. Inicialización del mapa y sus variables de control

Toda la funcionalidad de Leaflet [2] empleada en este proyecto parte de la instanciación de un objeto JavaScript de la clase *L.Map*, la clase que proporciona

Leaflet [2] para crear los mapas. Por ende, lo primero que se hace a nivel JavaScript es dicha instanciación, cuyo resultado se vuelca en una variable. Esto nos permitirá controlar en todo momento los elementos añadidos al mapa (de cualquier tipo entre los contemplados por Leaflet [2]) y también comprobar en cualquier momento el estado de este.

Para explicar con más claridad el resto de las acciones llevadas a cabo con JavaScript tras la creación del objeto del mapa (en el contexto de esta primera fase), procede entender la manera en que Leaflet [2] interpreta los elementos añadidos a un mapa: principalmente y de forma muy genérica, existen dos tipos de elementos a tener en cuenta; los de control (*L.Control*) y los de capa (*L.Layer*). Los primeros comprenden elementos útiles como el control de *zoom* (mediante botones), el control de escala (no interactivo, únicamente visual), el control de selección de mapas (el del IGN [5], el de OpenStreetMap [6]…) o, más particularmente para este proyecto, el control de dibujo de superficies. Por otro lado, la clase genérica de capas incluye prácticamente cualquier elemento que se pueda añadir al mapa; en concreto para este proyecto, cabe destacar los marcadores (ubicaciones), los polígonos (que representan superficies en el mapa, bien porque se han dibujado o bien porque se trata de áreas específicas como por ejemplo las áreas municipales) y los *popups* y *tooltips* (mensajes emergentes relativos a otras capas tales como marcadores o superficies). Tanto estos elementos como los de control tienen sus respectivas clases dentro de Leaflet [2] y heredan atributos y métodos de la superclase que corresponda (*L.Control* o *L.Layer*).

Respecto a los elementos de control del mapa, todos ellos están presentes en todo momento exceptuando el control de dibujo; este último solo se muestra cuando se ha hecho clic sobre el botón de “Seleccionar área” (la rutina concerniente a esto se explica con más detalle en el apartado Funcionalidad para seleccionar área). A continuación se incluye una breve descripción para cada uno de los elementos de control de los que dispone el mapa:

- **Control de *zoom* (parte superior derecha):** un pequeño panel de dos botones (“+” y “-“) para aumentar o disminuir el nivel de *zoom* aplicado al mapa. Al margen de esto, también puede controlarse el *zoom* usando la

rueda del ratón cuando el puntero se encuentra sobre la superficie del mapa. Esta funcionalidad es propia de Leaflet [2].

- **Control de escala (parte superior derecha):** un simple elemento meramente informativo que muestra la escala a la que se visualiza el mapa en cada instante (se actualiza dinámicamente). Esta funcionalidad se ha conseguido incorporando un plugin de Leaflet [7] desarrollado por un tercero (se incluye referencia a la fuente en Bibliografía y referencias).
- **Control de mapas (parte inferior derecha):** inicialmente se muestra como una pequeña barra horizontal que, al ubicarse el puntero del ratón sobre ella, se despliega hacia abajo permitiendo al usuario elegir entre diferentes opciones para la fuente del mapa que se muestra. Asimismo, este elemento permite añadir opciones secundarias de datos de mapas las cuales pueden combinarse a gusto del usuario. Esta funcionalidad se ha conseguido incorporando un plugin de Leaflet [8] desarrollado por un tercero (se incluye referencia a la fuente en Bibliografía y referencias).
- **Control de posición (parte inferior izquierda):** una barra horizontal que actualiza dinámicamente la posición actual del cursor expresada en coordenadas de latitud y longitud. Esta funcionalidad se ha conseguido incorporando un plugin de Leaflet [9] desarrollado por un tercero (se incluye referencia a la fuente en Bibliografía y referencias).
- **Control de dibujo de superficies (parte superior derecha):** cuando se muestra, permite al usuario dibujar áreas circulares, rectangulares o poligonales en general; la aplicación se ocupa, para cada evento de adición o eliminación de una superficie, de comprobar los puntos contenidos en las superficies restantes. Esta funcionalidad se ha conseguido incorporando un plugin de Leaflet [10] desarrollado por un tercero (se incluye referencia a la fuente en Bibliografía y referencias).

En cuanto a los elementos de capa, se instancia un objeto *L.Marker* (marcador para ubicación), un objeto *L.Popup* (*popup* con la información de la respectiva ubicación) que a su vez se añade al objeto *L.Marker* correspondiente y, en último lugar, un objeto *L.Icon* el cual también se añade al objeto *L.Marker* con tal de asignar la imagen de icono para el marcador en cuestión. Lo anteriormente descrito se lleva a cabo iterativamente para cada punto especificado en los datos

recibidos en la plantilla Twig (téngase en cuenta que esta parte de la implementación se encuentra en código JavaScript integrado en la vista HTML que recibe el cliente).

Considerando el último párrafo, se dispone de un conjunto de marcadores (cada uno de ellos representando una ubicación) que a priori se añadirían directamente al mapa para su representación en este. Sin embargo, para mejorar la calidad de la vista en lo que a marcadores se refiere, se ha empleado otro plugin de Leaflet [11] (desarrollado también por un tercero, la referencia a la fuente constará en Bibliografía y referencias) que permite agrupar dinámicamente los marcadores teniendo en cuenta la proximidad entre ellos y el nivel de *zoom* aplicado. Visualmente, esto resulta especialmente útil para el usuario cuando se trata con cantidades de ubicaciones relativamente grandes. Para cualquier grupo de puntos se muestra un elemento circular que varía su color en función del número de ubicaciones que alberga y con dicho número indicado en el centro del círculo.

Además, la vista incluye en el lateral izquierdo del mapa un panel en el que se listan las categorías de ubicaciones existentes en el mapa. Como se ha explicado con anterioridad, estas categorías vienen dadas por los datos recibidos en la petición POST al servidor web. Cada una de ellas se muestra en el panel (titulado como “Capas”) como un elemento *checkbox* que se puede marcar y desmarcar a voluntad. Esto permite al usuario elegir en cada momento qué categorías quiere representadas en el mapa y cuáles no. En esta fase de la implementación JavaScript, lo único que se hace es generar el panel a nivel HTML y rellenar las consiguientes variables de control para su uso posterior.

Por último, y en referencia a las mencionadas variables de control, es aquí donde la mayoría de ellas se inicializan; casi todas son colecciones *Map* que serán claves para simplificar la lógica correspondiente a la segunda fase (la cual depende en gran medida de las acciones del usuario). Por ilustrar algo más la utilidad de estas colecciones JavaScript, considérese un objeto *Map* que tenga tantas entradas como categorías de ubicaciones (capas) existan; supóngase ahora que, para cada una de estas entradas, se asigna como clave de esta el identificador de la capa (análogo al identificador de fichero en la aplicación principal) y, como valor, una colección *Set* que incluya todos los objetos *L.Marker*

correspondientes a las ubicaciones de la capa en cuestión. Resulta relativamente sencillo instanciar una variable así en la vista y esta, a su vez, facilita considerablemente la identificación de ubicaciones según la capa (en este caso en concreto), algo muy aprovechable en la implementación de muchas de las funciones JavaScript de la otra fase de la implementación.

Por resumir muy brevemente este apartado, la mayoría de las acciones realizadas a nivel JavaScript en esta etapa generan tanto el mapa y sus elementos como otros elementos HTML en su configuración inicial y, paralelamente, inicializan variables de control para estos que resultarán útiles posteriormente.

3.3.1.2. Implementación de las funcionalidades del mapa

En este apartado se van a tratar las diferentes implementaciones llevadas a cabo para dotar al mapa de la funcionalidad requerida para cumplir los objetivos con los que parte este TFG. Además, todo el código necesario se encuentra almacenado en un archivo JavaScript alojado, a su vez, en el servidor web. Su URL relativa está referenciada en un elemento *script* de la vista, con tal de que el navegador cargue el contenido del fichero. Cabe destacar que, dado que la mayor parte del código hace uso de elementos HTML y JavaScript instanciados en la vista HTML, se fuerza al navegador a ejecutar una carga diferida (*deferred*), esto es, el contenido del fichero no se carga hasta que el documento HTML ha sido cargado por el navegador en su totalidad. De esta forma, se garantiza que cualquier código del fichero que se ejecute directamente como consecuencia de la carga de este no encuentre problemas relativos a la inexistencia de determinados elementos debido a que no se ha terminado de cargar la vista (que es quien los instancia).

En cuanto a la estructura del fichero, está mayoritariamente compuesto por funciones JavaScript que luego son llamadas dependiendo de las acciones del usuario. El resto de este son, por un lado, variables y constantes útiles para referenciar determinados parámetros y elementos HTML de la vista; por el otro, asignaciones de manejadores de eventos concernientes tanto a elementos HTML como al mapa (lo cual es un caso particular puesto que para ello se emplea Leaflet). Cabe señalar también que en algunas ocasiones se ha hecho uso de AJAX [12] con el fin de ejecutar llamadas asíncronas al servidor (los casos

más relevantes se detallan en los apartados Enlace para obtener toda la información disponible de una ubicación y Funcionalidad para generar informe).

La principal funcionalidad que se pretende conseguir es la de permitir al usuario dibujar una o más superficies sobre el mapa con tal de analizar el total de ubicaciones contenidas en la superficie global que resulte de la unión de todas las superficies individuales dibujadas. Por otra parte, se incluyen funcionalidades secundarias (que también se describirán con más detenimiento a continuación), tales como:

a) El filtrado dinámico de los marcadores a mostrar según su categoría (algo ya mencionado anteriormente).
b) Un segundo panel lateral donde se actualiza dinámicamente información básica relativa a las ubicaciones contenidas en las superficies dibujadas.
c) Ofrecer al usuario la posibilidad de, haciendo clic sobre un enlace presente en el *popup* de cada marcador, obtener en una nueva ventana toda la información relativa a la respectiva ubicación (para lo cual se llama a un controlador de la aplicación principal).
d) La opción de solicitar un informe conteniendo información y estadísticas detalladas sobre las ubicaciones contenidas en las superficies dibujadas (también se llama a la aplicación principal para esto).

3.3.1.2.1. Funcionalidad para seleccionar área

Esta funcionalidad parte de la existencia de un botón en la esquina superior izquierda del mapa (“Seleccionar área”); al pulsarlo, se suceden los siguientes eventos:

- Se añaden dos nuevos botones a su derecha: el de cancelar y el de generar informe. El primero simplemente deshace la acción de haber pulsado el botón de seleccionar área; el segundo, siempre que haya al menos una superficie dibujada (o una capa geográfica seleccionada), permite ser pulsado y solicita al servidor la creación del informe en formato HTML para posteriormente mostrarlo al usuario en una nueva ventana.
- En la parte superior derecha se muestra el control de dibujo (ya descrito anteriormente); con él, el usuario puede dibujar tantas figuras como guste, así como eliminar las que desee. Gracias a la funcionalidad aportada por

este plugin en particular, se controlan los eventos correspondientes a haber dibujado una figura o haberla eliminado, y de esta manera resulta posible actualizar dinámicamente el panel lateral con la información relativa a las ubicaciones contenidas (se actualiza cada vez que se dibuja o elimina una figura).

- Los elementos existentes sobre el mapa (paneles, controles…) ven reducida su opacidad (mediante CSS), lo cual ofrece al usuario una visión algo más limpia del mapa sobre el que va a dibujar. Esto sucede cada vez que se pincha en una de las figuras del control de dibujo que permiten dibujar (esta implementación se ha logrado también gracias a eventos incluidos en el plugin).

En lo referente a la rutina empleada para determinar si una ubicación dada se encuentra dentro de alguna de las superficies dibujadas, a continuación se explica con algo de detalle el procedimiento que se sigue:

1. Al iniciarse la rutina en cuestión (algo que sucede cada vez que se añade o se elimina una figura del mapa, como ya se ha explicado), lo primero a realizar es un filtrado de las categorías de ubicaciones existentes; la rutina selecciona las capas (y todos los marcadores pertenecientes a estas en consecuencia) que el usuario tiene marcadas en el presente instante.
2. De forma iterativa para cada capa seleccionada, se considera cada marcador incluido en esta y para cada uno de ellos se evalúan todas las figuras dibujadas; si un marcador resulta encontrarse dentro de una o varias superficies, se almacena internamente como tal (es decir, basta con que un marcador esté dentro de una única superficie del conjunto de ellas). Ahora bien, el algoritmo empleado para dicha determinación varía en función del tipo de figura que se esté tratando:
    a. **Si la figura es un rectángulo**: para cualquier objeto de Leaflet [2] que represente un polígono, existe un método que permite obtener una representación del área rectangular en la que este está contenido. Como puede deducirse, para el caso del rectángulo dicha área coincidirá con la de este último, lo cual resulta ideal dado que el objeto de Leaflet [2] devuelto por el método al que se ha hecho referencia tiene a su vez otro método que permite

comprobar si un marcador se encuentra dentro de esa área rectangular. De esta manera, se puede comprobar fácilmente si un rectángulo dado contiene en su área cualquier marcador que se quiera considerar.

b. **Si la figura es un círculo**: este caso cuenta también con una solución relativamente sencilla; pese a que en esta ocasión la metodología utilizada en el anterior punto no es del todo eficiente (puesto que se estaría empleando como referencia el área rectangular conteniendo al círculo en cuestión), hay otra opción que sí aporta una efectividad plena: usando Leaflet [2], se obtiene primeramente el radio del círculo; acto seguido, se extraen de igual manera las coordenadas de su centro (representadas mediante un objeto *L.LatLng* de Leaflet [2]). Finalmente, basta con computar (también mediante Leaflet [2]) la distancia modular del centro al marcador que se quiera evaluar, y en caso de que dicha distancia resulte ser menor o igual al radio del círculo puede asumirse que el marcador está dentro del área del círculo.

c. **Si la figura es un polígono irregular**: para este caso más genérico, Leaflet [2] no ofrece implícitamente ninguna solución para el problema planteado. Sin embargo, ha sido posible encontrar un algoritmo conocido como Ray Casting [13] (antiguamente utilizado en motores gráficos) que determina si un punto dado está dentro de un polígono irregular en concreto. Particularmente, el algoritmo hallado era específico de Leaflet [2] (es decir, implementa la rutina empleando atributos y métodos propios de Leaflet [2]). La página web donde está publicado este algoritmo se referenciará en Bibliografía y referencias.

3. Una vez determinados los marcadores contenidos en las superficies dibujadas, se almacenan internamente (mediante colecciones) y se procede a actualizar la información del segundo panel lateral. Dicha información indica el total de ubicaciones encontradas y el número de capas implicadas. Asimismo, se muestra una enumeración de estas capas incluyendo, para cada una de ellas, la cantidad de ubicaciones contenidas (también se expresa esta información de una forma más

gráfica con una barra de progreso que aporta una idea del peso relativo de cada capa en el total de ubicaciones).

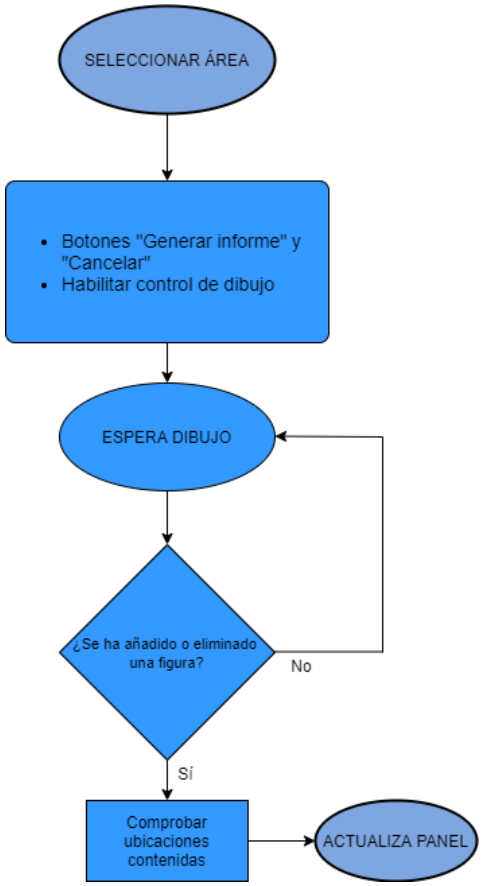

Figura 3: Flujograma de Seleccionar área (elaboración propia)

3.3.1.2.2. Filtrado dinámico de capas

El propósito de esta implementación es bastante sencillo: mediante una lista de elementos *checkbox* (HTML) que enumera las capas existentes (es decir, las categorías en que se clasifican las ubicaciones del mapa), permitir al usuario elegir en cada momento cuáles de esas capas deben ser o no mostradas. Igualmente, procede señalar que este filtro afectará también a las capas que se tengan en cuenta para la implementación descrita en el punto anterior.

En cuanto a la rutina JavaScript desarrollada, esta hace uso de la colección *Map* que agrupa los marcadores por capas (ya se ha hablado de ella con anterioridad). Todos los elementos *input* HTML (correspondientes a los *checkbox*) tienen asignado un manejador para el evento de hacer clic sobre cualquiera de ellos (ya sea para marcar o desmarcar). Dicho manejador identifica, gracias a los atributos del elemento HTML, qué capa se ha marcado o

desmarcado y, dependiendo de cuál de estas dos acciones ha disparado el evento, incorpora o elimina los marcadores que procedan.

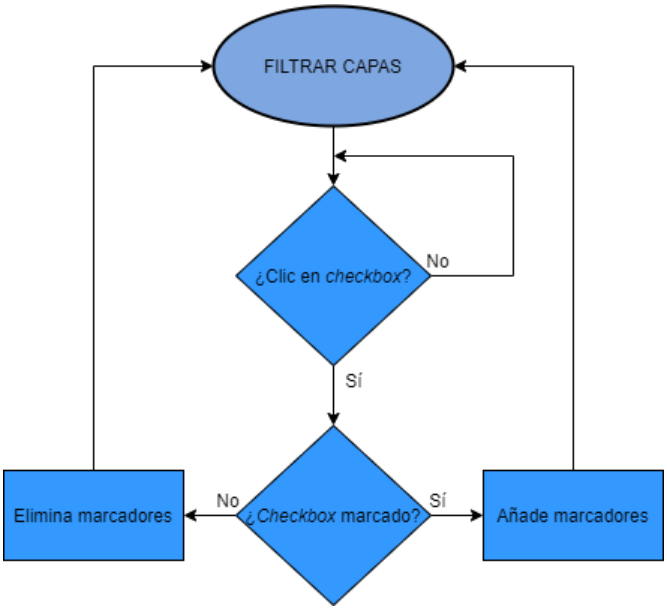

Figura 4: Flujograma de Filtrar capas

3.3.1.2.3. Panel lateral para informar de las ubicaciones contenidas

Este elemento de la interfaz ya se ha nombrado en más de una ocasión previamente. Su función es únicamente informativa; resulta útil para tener una primera perspectiva del resultado de la selección de área(s) realizada. No obstante, como también se ha explicado, no ofrece información específica ni detallada, para lo cual conviene hacer uso de la funcionalidad de generar el informe asociado a dicha selección.

Este elemento actualiza su contenido cada vez que se ejecuta la rutina de comprobar las ubicaciones contenidas, lo cual implica que se actualiza cada vez que el usuario añade o elimina una figura del mapa.

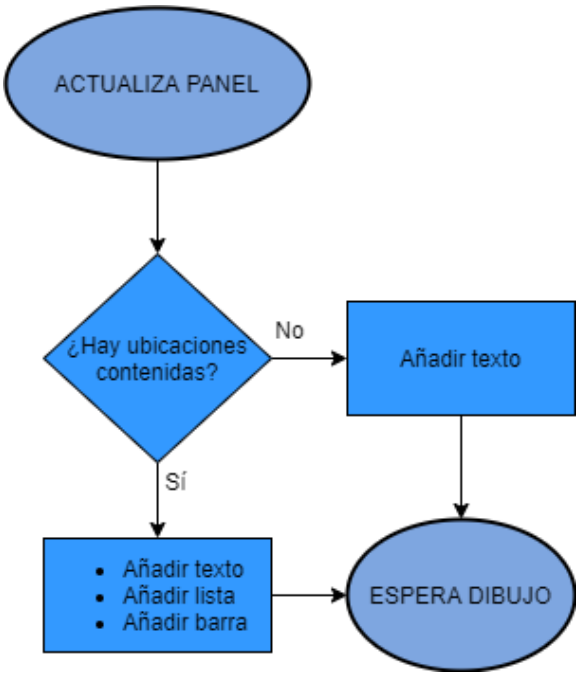

Figura 5: Flujograma de Actualizar panel

3.3.1.2.4. Enlace para obtener toda la información disponible de una ubicación

Esta implementación trabaja sobre el *popup* existente para cada marcador del mapa (mensaje emergente al hacer clic sobre el marcador en cuestión). El contenido de este puede ser cualquiera que se desee; para esta aplicación, este contenido no es más que la información relativa a cada ubicación que el usuario ha elegido visualizar en el mapa (esto se hace en la aplicación principal). Dicha información se envía al cliente en la petición POST inicial. No obstante, todos los marcadores tienen en sus *popups* un elemento adicional: un enlace (*anchor* HTML) que, al ser pinchado, envía una petición a un controlador de la aplicación principal (usando AJAX [12]) el cual, a modo de respuesta, envía un documento HTML. JavaScript se ocupa de abrir una ventana nueva para la visualización de dicho documento.

Para informar debidamente al servidor de la ubicación de la que se desea obtener la información, en la URL de la petición (de tipo GET) se incluyen dos parámetros: el identificador de capa (en lo que a la aplicación principal se refiere, esto equivale al identificador de fichero en su base de datos, de esta forma sabrá qué archivo debe leer para encontrar la ubicación) y el identificador de ubicación; este último es en realidad la fila del fichero en la cual se halla la información relativa a la ubicación buscada.

Como se habrá podido deducir, en el marco de esta implementación cada ubicación del mapa tiene asociada una URL única dada por sus identificadores de capa y ubicación (o lo que es lo mismo, sus identificadores de fichero y de fila, respectivamente). Pese a que esto implica que se podría generar la URL para la ubicación conforme se va a enviar la solicitud (es decir, concatenar a la parte fija de esta los parámetros que procedan con el formato adecuado), el hecho de estar utilizando Symfony [1] en el servidor web ofrece otra opción más fiable: en la fase de inicialización de variables de control, crear y rellenar una colección *Map* que relacione cada marcador con su URL (para esta implementación). La ventaja de hacerlo de esta manera reside en que, al estar incluyendo el código JavaScript necesario en la vista (esto es, en la plantilla Twig que Symfony [1] usará para generar la vista HTML), puede hacerse uso de una de las muchas funciones de Twig, *path*, la cual permite obtener la URL completa para llamar a cualquier método de cualquier controlador, incluyendo si se necesitasen los parámetros. Así, llegado el momento de enviar la petición con AJAX [12] solo se necesita extraer la URL de la colección *Map* ya creada en la primera fase.

3.3.1.2.5. Funcionalidad para generar informe

Esta funcionalidad permite al usuario obtener información contextual más detallada acerca de las ubicaciones contenidas en las superficies dibujadas. Por tanto, la posibilidad de usarla depende de si existe alguna superficie dibujada sobre el mapa (y si esta(s) contiene(n) mínimamente una ubicación en su conjunto). El usuario tiene a su disposición el botón de “Generar informe” para emplear esta funcionalidad, la cual despliega una vista HTML independiente a la del mapa en una nueva ventana. Pese a que la lógica de servidor relativa a esta implementación (el controlador PHP) pertenece al perímetro de la aplicación principal, para llamar a esta última, en este caso, se precisa de una petición POST que incluya en su cuerpo los datos relativos a las ubicaciones contenidas clasificadas según la capa a la que pertenecen (las categorías de las ubicaciones). Esto último compete a la implementación del cliente web, que es quien recopila y formatea los datos (en una estructura JSON).

Adicionalmente, se ha añadido a la vista del mapa, en la esquina superior derecha, un selector de capas; este no tiene nada que ver con las capas

equivalentes a las categorías en que se distribuyen las ubicaciones, son únicamente capas geográficas (cuyos datos se obtienen del servidor en formato GeoJSON y se procesan usando Leaflet [2]) que en su conjunto conforman una determinada superficie (dicho de otro modo, las capas mostradas en el selector son subdivisiones, en términos de superficie, de un área mayor). Esta es una funcionalidad con gran potencial para futuras líneas de desarrollo en lo respectivo a esta aplicación; por ahora, esta última puede cargar únicamente dos grupos distintos de capas (cuál se carga es una decisión tomada en el marco de la aplicación principal): las áreas municipales de la CARM o sus áreas de Salud. Se trata de dos criterios de subdivisión territorial muy relevantes a nivel administrativo.

El selector del que se ha hablado solamente permite marcar o desmarcar las diferentes capas una a una (aquellas marcadas aparecerán resaltadas) y, a efectos del mapa, las capas marcadas se resaltarán en este con un color diferente al resto (inicialmente todas las capas se muestran en el mapa con un mismo color, y cuando se mueve el ratón por encima de cualquiera de ellas se indica su nombre y su perímetro queda resaltado). Adicionalmente, también puede hacerse clic sobre cualquiera de las capas para marcarla o desmarcarla (en lugar de usar el selector).

Sin embargo, el hecho de disponer de este conjunto de capas adicional (independientemente de cuál sea) sí aporta algo más a nivel funcional; cuando se inicia la rutina para generar el informe, esta última tiene en cuenta estas capas y también envía al servidor una relación de las ubicaciones contenidas en función de la capa en que se encuentren, lo cual, a su vez, será considerado por el controlador PHP a la hora de generar el informe solicitado.

3.3.2. Apariencia y estilo con CSS (Bootstrap)

Para la implementación del estilo a mostrar en la vista del cliente, se ha trabajado con CSS, tanto a nivel primitivo (añadiendo normas ad-hoc al documento HTML) como con el uso de una librería (Bootstrap [3]) para propósitos más generales.

Respecto a la librería Bootstrap [3], debe indicarse ante todo que no se ha hecho un uso totalmente directo de esta, puesto que lo que realmente se ha empleado han sido diferentes versiones de esta publicadas de forma abierta en la

plataforma web Bootswatch [4] que, valiéndose principalmente del cambio de los colores establecidos como primario y secundario respectivamente, aportan a la vista una temática muy distinta. Dicha temática afecta prácticamente al estilo de cualquier elemento HTML de la vista, e incluso se ha aprovechado la ya mencionada declaración de colores primario y secundario para implementar ciertas normas específicas con CSS.

En cuanto a la adición de normas CSS con un propósito más específico, la mayoría de ellas se encuentran almacenadas en un fichero CSS alojado en el servidor web de la aplicación; este fichero está referenciado en la vista HTML (mediante un elemento *link*) con tal de que el navegador web lo cargue. No obstante, en determinadas ocasiones también se ha hecho uso de CSS a través de JavaScript, principalmente para alterar dinámicamente el estilo y/o apariencia de ciertos elementos de la vista.

4. Conclusiones y líneas futuras

A continuación, se expondrá un ejemplo de uso de la aplicación desarrollada, así como las conclusiones finales de este Trabajo Fin de Grado y sus posibles líneas de desarrollo futuras.

4.1. Ejemplo de uso de la aplicación

Para mostrar el uso de la aplicación, se va a relatar, paso por paso, un posible caso de uso de esta, incluyendo para ello imágenes de la interfaz. Todas estas imágenes son capturas de pantalla de la interfaz web (de elaboración propia, por tanto).

4.1.1. Vista de la aplicación principal

Inicialmente, es necesario acceder a la aplicación principal y, desde el módulo de mantenimiento de capas (accesible en el menú superior “Mantenimientos”), crear como mínimo una nueva capa (en caso de que no se disponga de ninguna aún) con datos provenientes de diferentes fuentes (estas últimas se corresponderán con las capas de datos en el mapa). Una vez se dispone de la capa que se pretende visualizar geográficamente, basta con pinchar en el icono del mapa en el extremo derecho de la fila de la tabla donde se ubica la capa en cuestión.

Figura 6: imagen de la vista "Mantenimiento de capas" de la aplicación principal

4.1.2. Vista inicial del mapa

Una vez realizado el punto anterior, se generará la vista correspondiente al mapa como resultado del flujo de aplicación descrito en el apartado Funcionamiento de la aplicación, con todas las funcionalidades descritas en Implementación de las funcionalidades del mapa ya cargadas a nivel JavaScript.

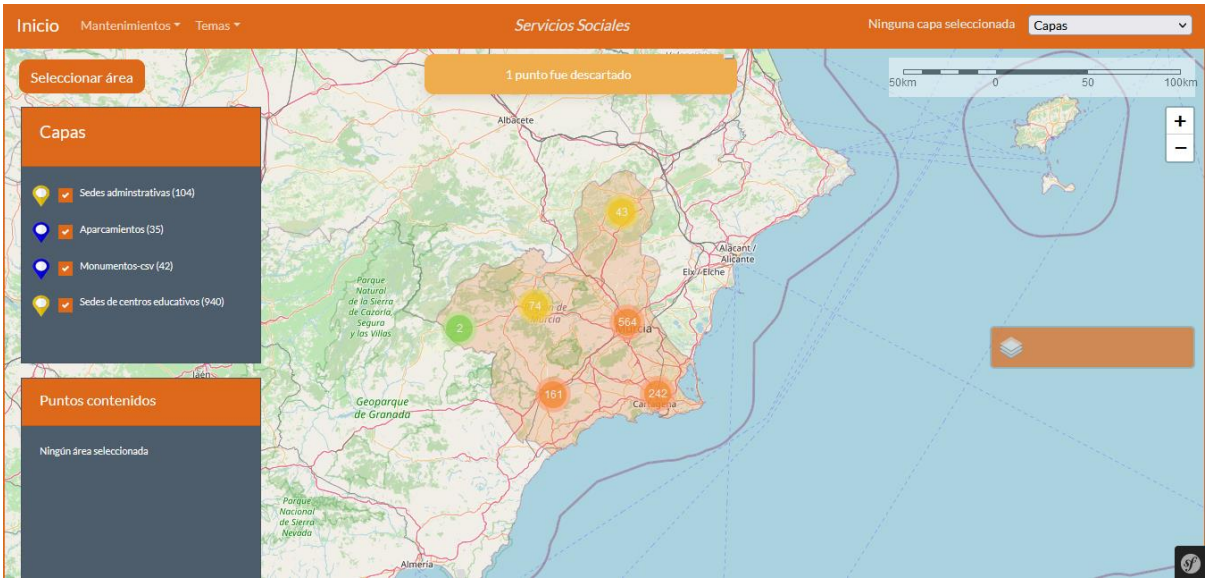

Figura 7: imagen de la vista inicial del mapa

Como se puede apreciar en la imagen, se muestra al usuario un pequeño mensaje informándole de los puntos que se han descartado para su representación del total de los que fueron recibidos desde la aplicación principal (es decir, el número de ubicaciones que no han pasado la validación mencionada

en Implementación del controlador PHP (Symfony)). Este mensaje puede cerrarse manualmente; si no se hace desaparecerá al cabo de unos segundos.

4.1.3. Vista focalizada del mapa

Seguidamente se mostrarán otras vistas del mismo mapa, pero en esta ocasión con el zoom de este aumentado con el propósito de poder apreciar determinadas características mencionadas en esta memoria.

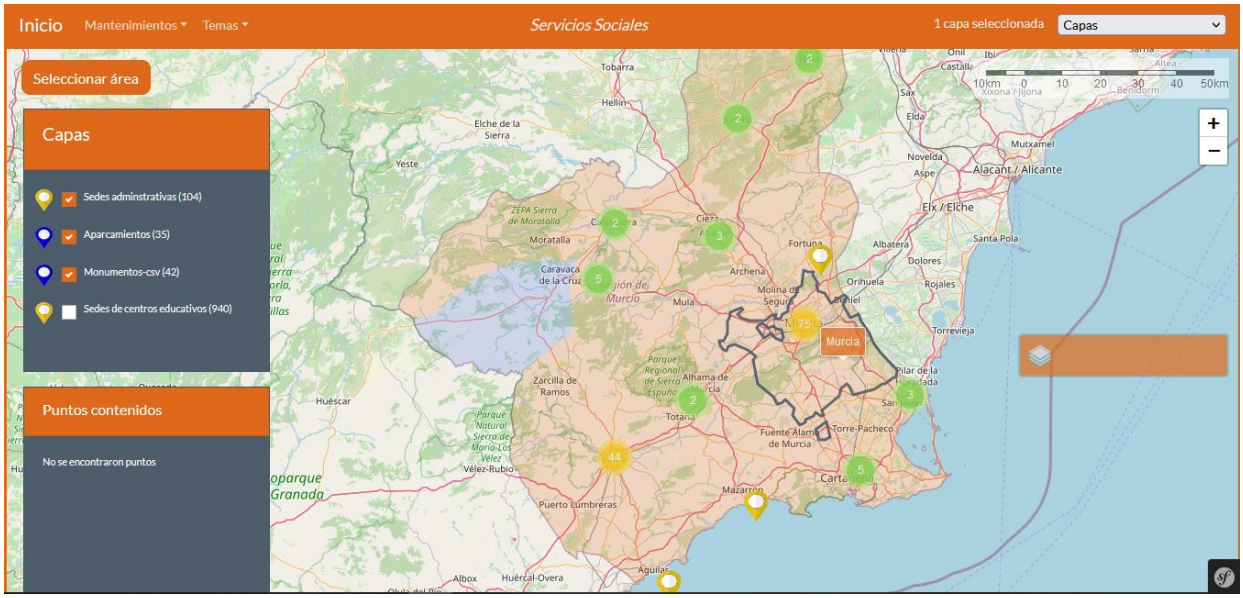

Figura 8: imagen del mapa mostrando una capa geográfica

En esta primera imagen se puede ver cómo, al colocar el puntero del ratón sobre cualquiera de las capas geográficas cargadas por el mapa, los límites de esta se resaltan y aparece su nombre en un pequeño cuadro de texto.

Además, se puede ver igualmente cómo se ha seleccionado una capa geográfica (usando el selector). El texto informativo junto a este último así lo refleja y, consecuentemente, la capa correspondiente ha quedado resaltada en el mapa con un color azulado (en este caso representa el municipio de Caravaca de la Cruz).

Asimismo, tanto en esta imagen como en la anterior se pueden observar diferentes elementos de la interfaz tales como los dos paneles laterales (para selección de capas de datos e información de ubicaciones contenidas), el botón de seleccionar área, los controles del mapa (la mayoría de ellos en el lateral derecho) o el selector de capas geográficas (esquina superior derecha).

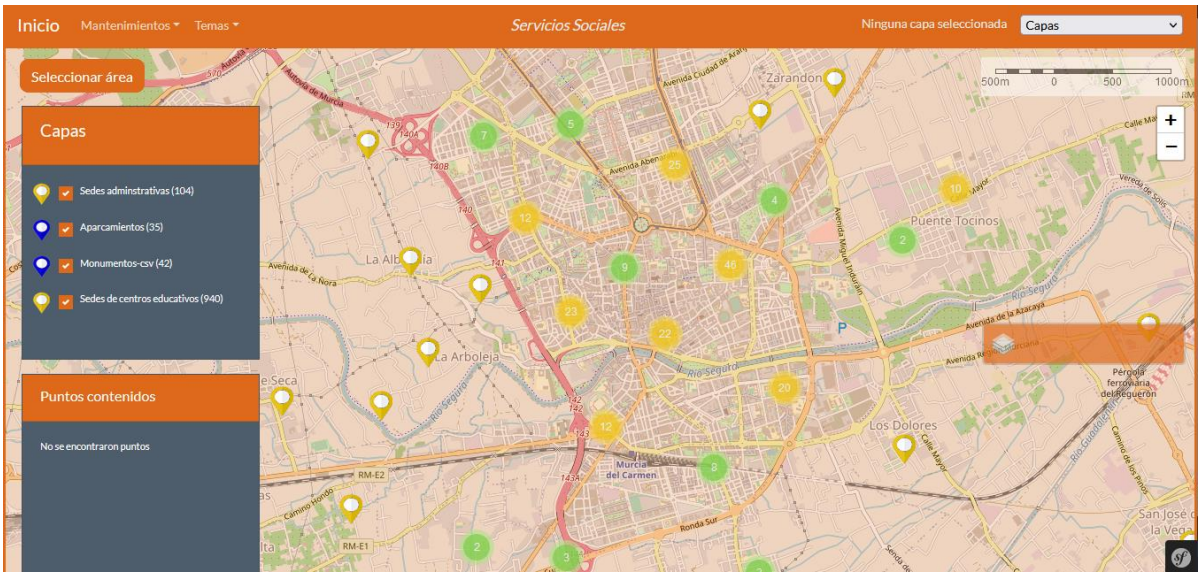

Figura 9: imagen del mapa con zoom en una ciudad (Murcia)

Esta otra imagen se ha tomado con el mapa enfocado en la ciudad de Murcia. El principal objetivo de esto es que se pueda apreciar el funcionamiento de los marcadores agrupados por zonas; si se compara la distribución de estos en esta imagen con su distribución en la imagen previa se podrá notar cómo los marcadores se agrupan o separan en función del zoom aplicado.

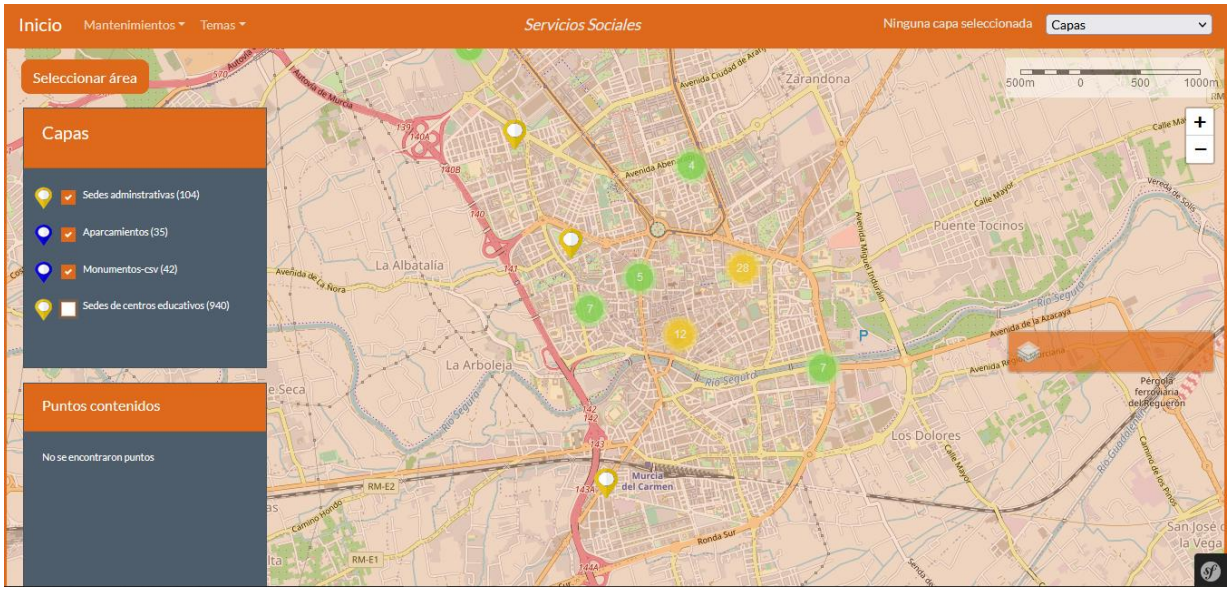

Figura 10: imagen del mapa con una capa filtrada

En esta otra imagen se puede ver el resultado de filtrar (desmarcar) una de las capas de datos; respecto a la imagen anterior, muchas de las ubicaciones que aparecían en la ciudad de Murcia ya no se muestran.

4.1.4. *Popup* de un marcador

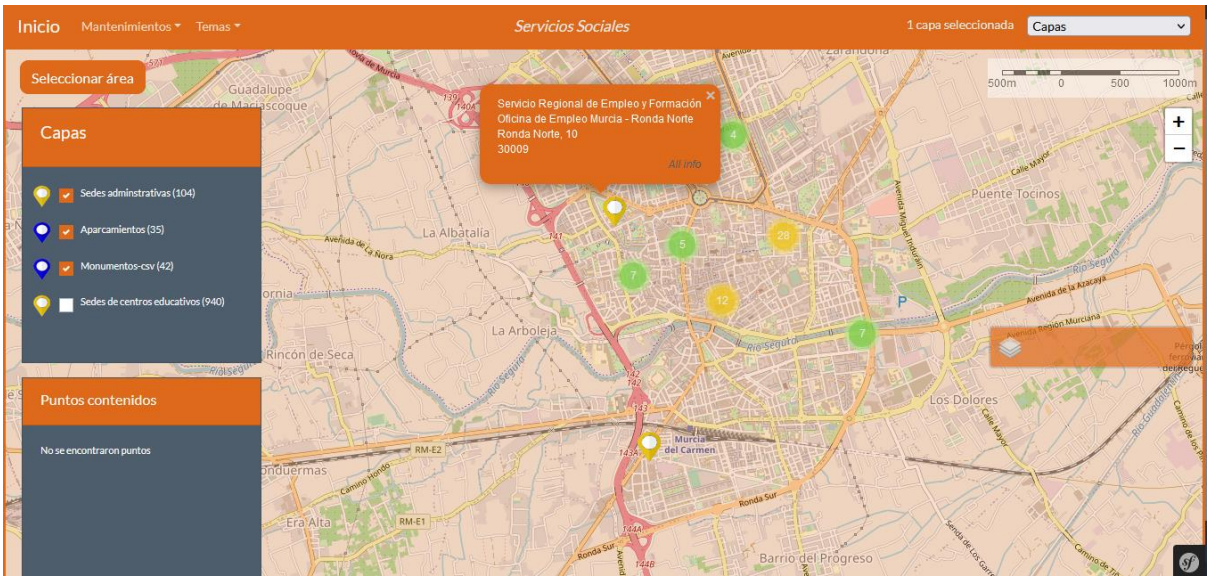

Figura 11: imagen del mapa con popup de una ubicación

Aquí se observa el *popup* de una ubicación sobre la que se ha hecho clic; como también puede apreciarse, el *popup* incluye el enlace "*All info*" para solicitar toda la información sobre la correspondiente ubicación.

4.1.5. Vista de la información completa de una ubicación

Origen: Portal Regional de Datos Abiertos de la Región de Murcia
Fichero: sedes adminstrativas (Fila 98 de 104 registros.)

TIPO	Servicios Públicos
SUBTIPO	Servicio Regional de Empleo y Formación
AGRUPACION	Oficinas de Empleo
CONSEJERIA_ORGANISMO	Servicio Regional de Empleo y Formación
UNIDAD	Oficina de Empleo Murcia - Ronda Norte
DIR3	
DIRECCION	Ronda Norte, 10
CP	30009
LOCALIDAD	
MUNICIPIO	Murcia
TELEFONO	968294060
CORREO	sef-murcia-rnorte@listas.carm.es
ENLACE	
PROPIEDADES	
LAT	37.9924598
LON	-1.1401763
COORDENADAS	37.9924598,-1.1401763

Figura 12: imagen de la vista con la información de una ubicación

Este es un ejemplo de la vista que se obtiene al hacer clic en el enlace del *popup* de cualquier marcador. La información mostrada corresponde, como es lógico, a la ubicación de la que se ha solicitado esta vista.

4.1.6. Dibujo de superficies

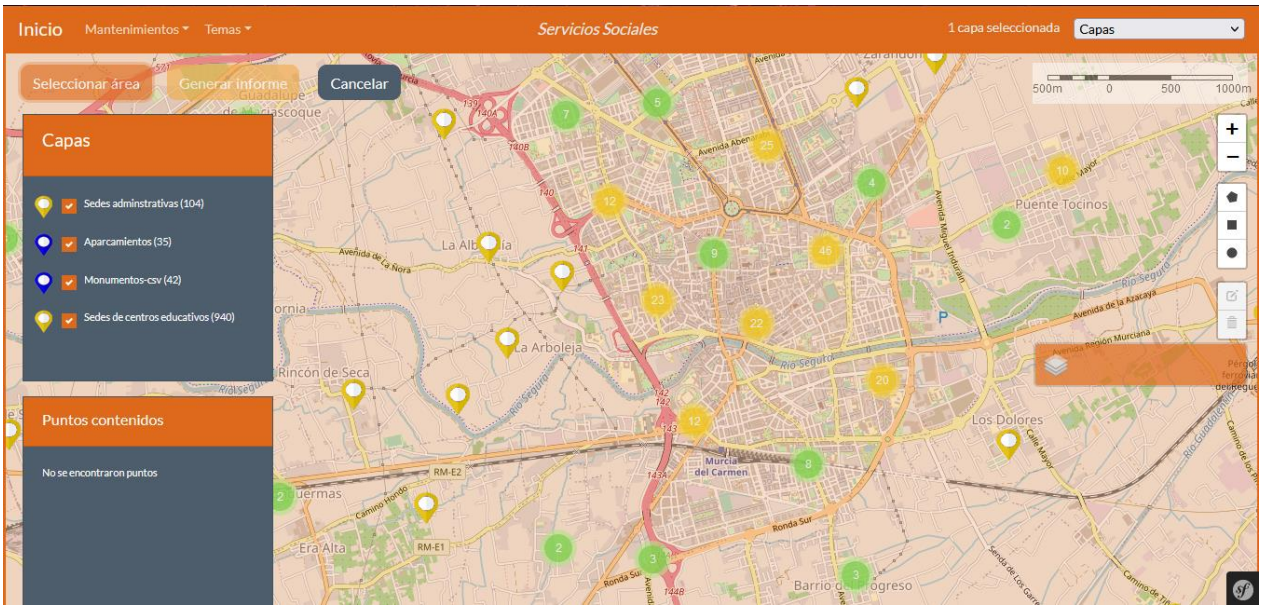

Figura 13: imagen del mapa con la funcionalidad de seleccionar área

Una vez se ha pinchado en el botón de seleccionar área, como se aprecia en esta imagen, se añaden al mapa los botones de cancelar y generar informe, así como los controles de dibujo (debajo del control de zoom). Estos últimos permiten dibujar rectángulos, círculos o polígonos irregulares, al igual que eliminar cualquier figura ya dibujada. Seguidamente se muestra una imagen con un ejemplo de dibujo de cada una de estas superficies, así como el efecto que esto tiene en el panel lateral inferior.

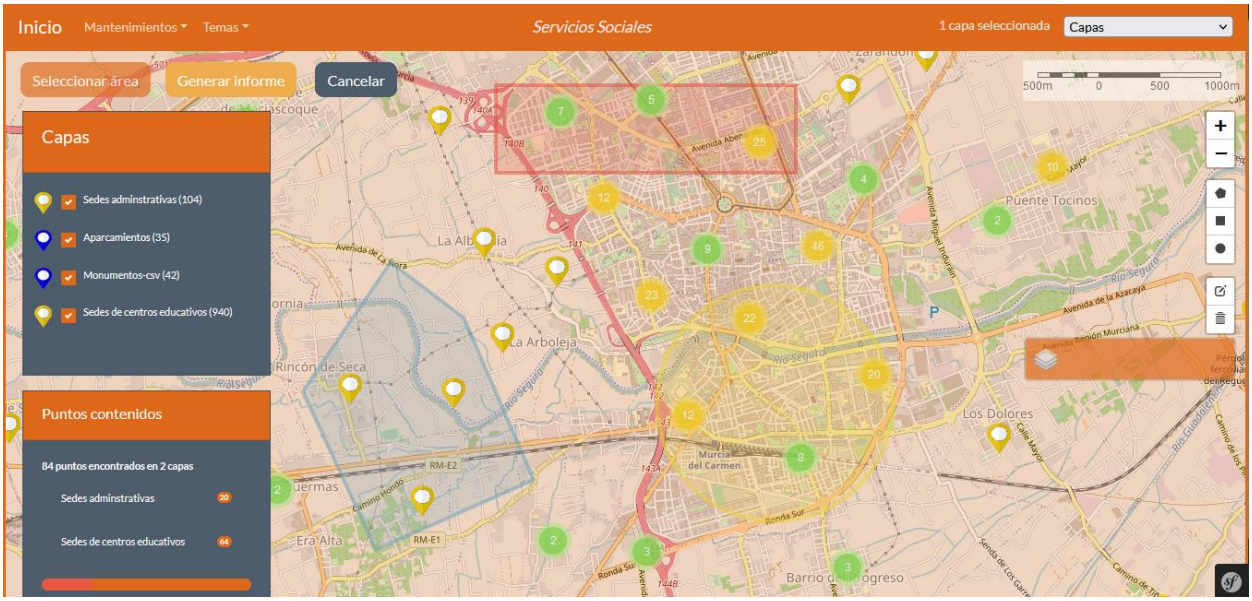

Figura 14: imagen del mapa con superficies dibujadas

Como se ha dicho, se han dibujado en el mapa un rectángulo (en rojo), un círculo (en amarillo) y un polígono irregular (en azul). El panel lateral inferior (“Puntos contenidos”) muestra un resumen de las ubicaciones encontradas en estas superficies, incluyendo un listado por capas de datos y la barra de progreso con el peso relativo de cada capa sobre el total.

4.1.7. Generación de informe

Figura 15: imagen de la vista del informe generado

En la imagen se puede observar una parte de la vista generada con el informe de las ubicaciones contenidas. Este informe tendrá en cuenta tanto las superficies dibujadas como las capas geográficas seleccionadas previamente a pinchar sobre el botón de generar informe.

4.2. Conclusiones

Al término de este Trabajo Fin de Grado, la aplicación desarrollada dispone de las principales funcionalidades que se marcaron como objetivo al comienzo de este, si bien es cierto que algunas de ellas se han replanteado a lo largo del proceso de implementación para adaptarlas de la mejor manera posible a determinados requerimientos que iban surgiendo durante este periodo.

La aplicación permite, en primer lugar, visualizar conjuntos de datos relativos a ubicaciones de cualquier ámbito que se preste, categorizadas según el criterio determinado previamente por parte del usuario. Asimismo, el mapa generado ofrece una interfaz relativamente sencilla y limpia incluso cuando se trata con

grandes cantidades de puntos, algo posible en gran medida a complementos software de la librería principal (Leaflet [2]) empleada.

También ha sido posible implementar una herramienta de dibujo que permita al usuario trazar sus propias superficies sobre el mapa, y contemplar tanto estas últimas como otras superficies generadas a partir de datos externos (datos GeoJSON) en la determinación de qué ubicaciones se encuentran dentro de los perímetros establecidos y cuáles no. Se ofrece tanto un resumen escueto de los resultados como la posibilidad de generar un informe mucho más completo de los mismos.

Por otra parte, se permite también visualizar información concreta de cualquier ubicación representada, así como toda la información disponible de esta, algo concerniente al propósito de ofrecer al usuario una visualización más atractiva y amigable de los datos abiertos.

Por último, conviene remarcar que todo el desarrollo de este TFG ha sido posible gracias al software libre, tanto de entidades como de terceras personas. Personalmente, considero que no cabe mejor manera de culminar un proyecto en el marco de la cultura de datos abiertos que expresando el reconocimiento que merecen todos aquellos desarrolladores que han hecho de la realización de este Trabajo algo mucho más sencillo, liviano e incluso plausible de lo que a priori podía aparentar.

4.3. Líneas de desarrollo futuras

La aplicación desarrollada en este TFG supone un primer paso en el contexto del proyecto emprendido por la Oficina de Transparencia y Participación Ciudadana de la CARM para ofrecer a la ciudadanía un conjunto de herramientas efectivas para la visualización de datos abiertos de servicios administrativos.

Como se podrá imaginar, las aplicaciones que puede llegar a tener un mapa interactivo como el desarrollado en este proyecto son inmensas, y la cantidad de funcionalidades adicionales que podrían plantearse en un futuro lo son incluso en mayor medida. Téngase en cuenta que muchas de estas posibles funcionalidades estarían probablemente supeditadas al ámbito de los datos a visualizar. Por poner un ejemplo concreto (de considerable actualidad), podrían

emplearse datos del contexto sanitario para generar un mapa con estadísticas geo posicionadas del proceso de vacunación del COVID-19.

Por otro lado, durante todo este proyecto se ha planteado la aplicación a desarrollar como un módulo de la aplicación principal. Sin embargo, no es nada inverosímil contemplar la posibilidad de utilizarla como si de una API se tratase, ya que los mecanismos implementados para ejecutar las llamadas a esta son fácilmente replicables por otras aplicaciones, en caso de que estas últimas quisieran hacer algún uso de la aplicación desarrollada. Esta idea, además, va en la línea de la cultura de datos abiertos en la que se enmarca el TFG.

Una de las funcionalidades adicionales que sería interesante plantear en un futuro para esta aplicación sería la opción de permitir a los usuarios incorporar al mapa (y por ende al contexto geográfico que se visualiza) ubicaciones de cualquiera de las categorías mostradas en este y, gracias a la aplicación, obtener una idea del impacto que el cambio planteado supondría sobre ese contexto (dicho impacto podría cuantificarse mediante indicadores estadísticos calculados por la aplicación, por ejemplo).

Para finalizar este apartado, y en la línea del párrafo anterior, podría también explotarse la funcionalidad de la carga de capas geográficas a partir de datos GeoJSON; los tipos de capas que se pueden cargar son infinitos y dependen únicamente de disponer de los datos necesarios para ello, ya sean municipios, provincias en el caso de una Comunidad pluriprovincial, áreas de Salud, distritos educativos, entre otros.

5. Bibliografía y referencias

[1] Symfony SAS, «Symfony,» [En línea]. Available: https://symfony.com/.

[2] V. Agafonkin, «Leaflet,» [En línea]. Available: https://leafletjs.com/.

[3] Bootstrap, «Bootstrap,» [En línea]. Available: https://getbootstrap.com/.

[4] T. Park, «Bootswatch,» [En línea]. Available: https://bootswatch.com/.

[5] Ministerio de Transportes, Movilidad y Agenda Urbana, «Instituto Geográfico Nacional,» [En línea]. Available: https://www.ign.es/web/ign/portal.

[6] © OpenStreetMap contributors, «OpenStreetMap,» [En línea]. Available: https://www.openstreetmap.org.

[7] E. Escoffier, «Leaflet-Graphicscale,» [En línea]. Available: https://github.com/nerik/leaflet-graphicscale.

[8] P. Soriano y I. Sánchez Ortega, «Leaflet.Spain.WMS,» [En línea]. Available: https://github.com/sigdeletras/Leaflet.Spain.WMS.

[9] A. Lukianto, C. Rosen y J. Pedraza, «Leaflet.MousePosition,» [En línea]. Available: https://github.com/ardhi/Leaflet.MousePosition.

[10] M. Guild, «Leaflet.Draw,» [En línea]. Available: https://github.com/Leaflet/Leaflet.draw.

[11] D. Leaver, «Leaflet.Markercluster,» [En línea]. Available: https://github.com/Leaflet/Leaflet.markercluster.

[12] MDN Web Docs, «AJAX,» [En línea]. Available: https://developer.mozilla.org/es/docs/Web/Guide/AJAX.

[13] Stack Overflow, «Determine if a point reside inside a leaflet polygon,» [En línea]. Available: https://stackoverflow.com/questions/31790344/determine-if-a-point-reside-inside-a-leaflet-polygon.

[14] W3Schools, «W3Schools,» [En línea]. Available: https://www.w3schools.com/default.asp.

[15] Stack Overflow, «StackOverflow,» [En línea]. Available: https://stackoverflow.com/.