UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

VISOR WEB-GIS AVANZADO PARA LA VISUALIZACIÓN DE REDES DE AGUA

Trabajo de Fin de Máster

Autor: Eduardo Farinós Rijo
Tutor 1: Agustín Trujillo Pino
Tutor 2: Pablo Fernández Moniz

Curso 2016/2017 - Convocatoria Septiembre - extraordinaria
Máster en Ingeniería Informática - Tecnología Web y Negocio Digital

Tabla de contenido

1. Agradecimientos:

Me gustaría que estas líneas sirvieran para expresar mi más profundo y sincero agradecimiento a todas aquellas personas que con su ayuda han colaborado en la realización del presente trabajo, en especial a mis tutores Agustín Trujillo Pino y a Pablo Fernández Moniz, por la orientación, el seguimiento y la supervisión continúa de la misma, pero sobre todo por la motivación y el apoyo recibido.

Quisiera hacer extensiva mi gratitud a mis compañeros de la Universidad de Las Palmas de Gran Canaria y al Instituto Universitario de Microtecnólogia Aplicada por su amistad y apoyo.

Finalmente, un agradecimiento muy especial a la comprensión, paciencia y el ánimo recibidos de mi familia que ha estado ahí para mí desde el principio y sin cuyos ánimos habría resultado mucho más complicado finalizar este proyecto.

2.1 Resumen:

Antes cuando trabajamos con datos geoespaciales había que instalar programas cargantes e incómodos. Ahora gracias a las nuevas tecnologías es posible crear una herramienta de visualización de elementos geográficos ligera y flexible, accesible desde cualquier parte y dispositivo.

Resolvemos este dilema con un Visor Web-GIS Avanzado, constituido con el conjunto de tecnologías Openlayers, Geoserver, PostgreSQL/PostGIS, JavaScript HTML en su mayoría. Se expondrá el desarrollo de la aplicación, su funcionamiento y la interacción con el usuario.

Aunque el proyecto esté orientado a la visualización de redes de aguas, la aplicación tiene el potencial suficiente como para emplearse a la gestión de los recursos geoespaciales de cualquier campo.

2.2 Abstract:

In the past when working with geospatial data we had to install weighty and uncomfortable programs. Now thanks to new technology it is posible create a tool to display geographic elements lightweight and flexible, accessible from anywhere and any device.

We solve this dilemma with an Advanced Web-GIS Viewer, constituted with the set of technologies Openlayers, Geoserver, PostgreSQL/PostGIS, JavaScript and HTML mostly. Will be described the development of the application, the operation, and interaction with the user.

Although the project is oriented to the visualization of water networks, the application has the potencial to be used in the management of geospatial resources in any other field.

3. Introducción:

La tecnología de los SIG (Sistema de Información Geográfica), o GIS en inglés, puede ser utilizada para investigaciones científicas, la gestión de los recursos, la gestión de activos, la arqueología, la evaluación del impacto ambiental, la planificación urbana, la cartografía, la sociología, la geografía histórica, el marketing, la logística por nombrar unos pocos.

Por ejemplo, un SIG podría permitir a los grupos de emergencia calcular fácilmente los tiempos de respuesta en caso de un desastre natural, o encontrar los humedales que necesitan protección contra la contaminación, o pueden ser utilizados por una empresa para ubicar un nuevo negocio y aprovechar las ventajas de una zona de mercado con escasa competencia. En este caso, se implementará esta tecnología para mejorar las redes de agua y su administración.

3.1 Estado del Arte

En las antiguas herramientas y programas, los datos geográficos (y los índices asociados) se almacenan en archivos del sistema, utilizando formatos propietarios. Por lo tanto, la herramienta SIG es el único capaz de interpretarlas y manipularlas.

La última generación de herramientas de desarrollo SIG proporciona una integración completa de datos geográficos en los sistemas gestor de base de datos. Estos están diseñados como módulos que amplían las bases de datos existentes con nuevos tipos de datos espaciales, operadores e índices. Dado que la base de datos entiende información nativa espacial, se hace responsable de todas las tareas de gestión (transacciones, seguridad, consulta, optimización, etc.).

A continuación, podemos ver las ventajas e inconvenientes del uso de una aplicación web contra una instalada o de escritorio.

Ventajas

- Ahorra tiempo: se pueden realizar tareas sencillas sin necesidad de descargar ni instalar ningún programa.
- No hay problemas de compatibilidad: basta tener un navegador actualizado para poder utilizarlas.
- No ocupan espacio en nuestro disco duro.
- Actualizaciones inmediatas: como el software lo gestiona el propio desarrollador, cuando nos conectamos estamos usando siempre la última versión que haya lanzado.
- Consumo de recursos bajo: dado que toda (o gran parte) de la aplicación no se encuentra en nuestra computadora, muchas de las tareas que realiza el software no consumen recursos nuestros porque se realizan desde otra computadora.
- Multiplataforma: se pueden usar desde cualquier sistema operativo porque solamente es necesario tener un navegador.
- Portables: es independiente de la computadora donde se utilice (PC de sobremesa, portátil) porque se accede a través de una página web (solamente es necesario disponer de acceso a Internet). La reciente tendencia al acceso a las aplicaciones web a través de teléfonos móviles requiere sin embargo un diseño específico de los ficheros CSS para no dificultar el acceso de estos usuarios.

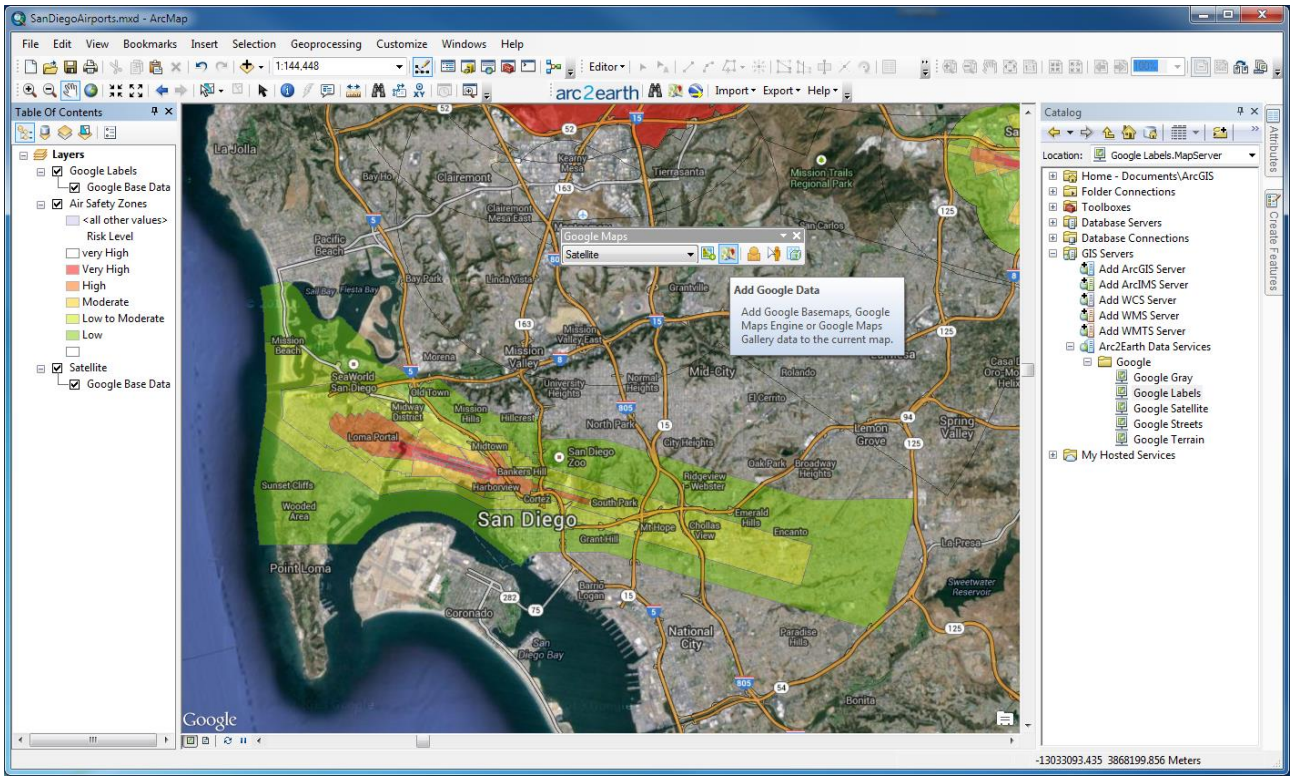
- La disponibilidad suele ser alta porque el servicio se ofrece desde múltiples localizaciones para asegurar la continuidad del mismo.
- Los virus no dañan los datos porque están guardados en el servidor de la aplicación.
- Colaboración: gracias a que el acceso al servicio se realiza desde una única ubicación es sencillo el acceso y compartición de datos por parte de varios usuarios. Tiene mucho sentido, por ejemplo, en aplicaciones en línea de calendarios u oficina.

Inconvenientes

- Habitualmente ofrecen menos funcionalidades que las aplicaciones de escritorio. Se debe a que las funcionalidades que se pueden realizar desde un navegador son más limitadas que las que se pueden realizar desde el sistema operativo.
- La disponibilidad depende de un tercero, el proveedor de la conexión a internet o el que provee el enlace entre el servidor de la aplicación y el cliente. Así que la disponibilidad del servicio está supeditada al proveedor.

Muchos son los softwares de SIG existentes en el mercado que permiten realizar análisis de datos alfanuméricos asociados a una componente espacial, los más importantes son:

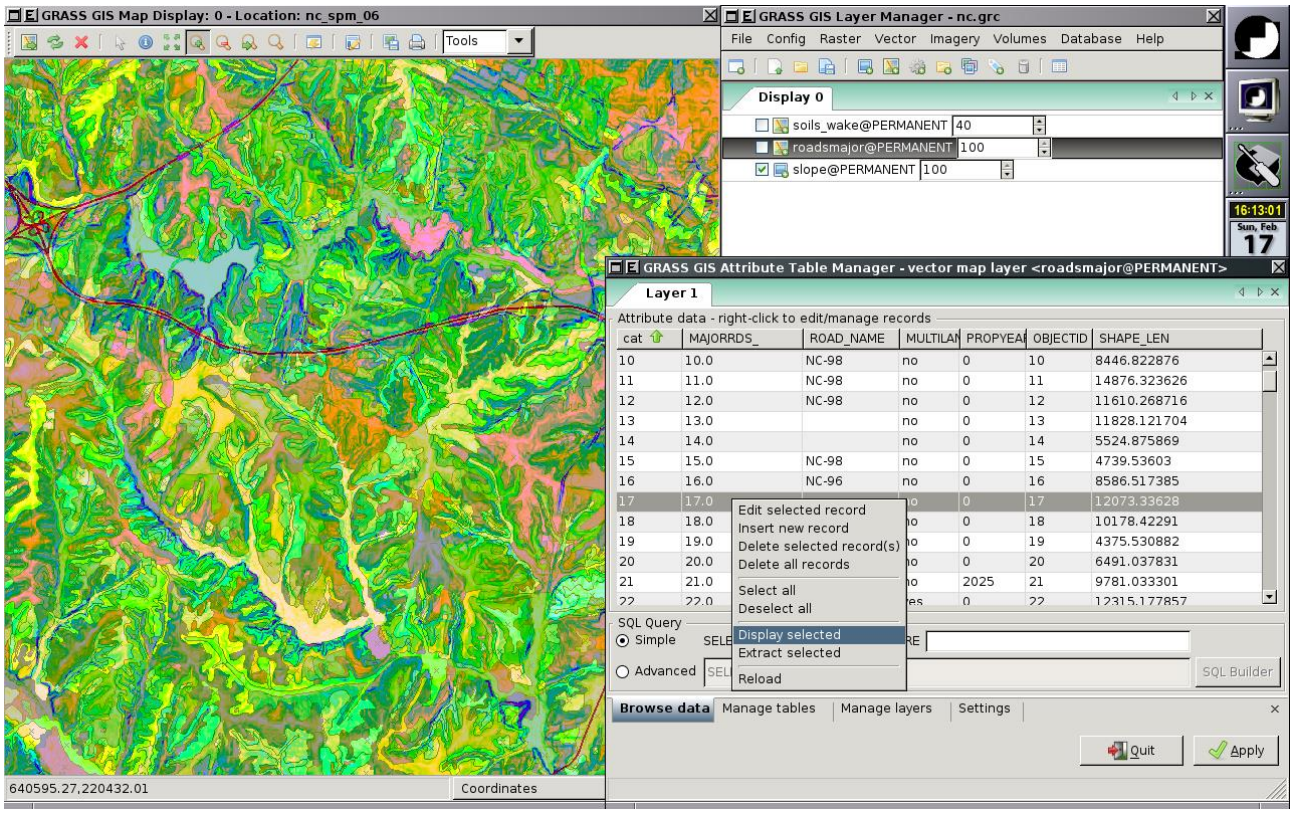
El sistema **ArcGIS** constituye un sistema integrado completo, que comparte la misma arquitectura de componentes con el fin de poder manipular, distribuir, crear y analizar la información geográfica. Tiene una licencia comercial, por lo que hay que pagar por el uso de sus productos.



ArcGIS

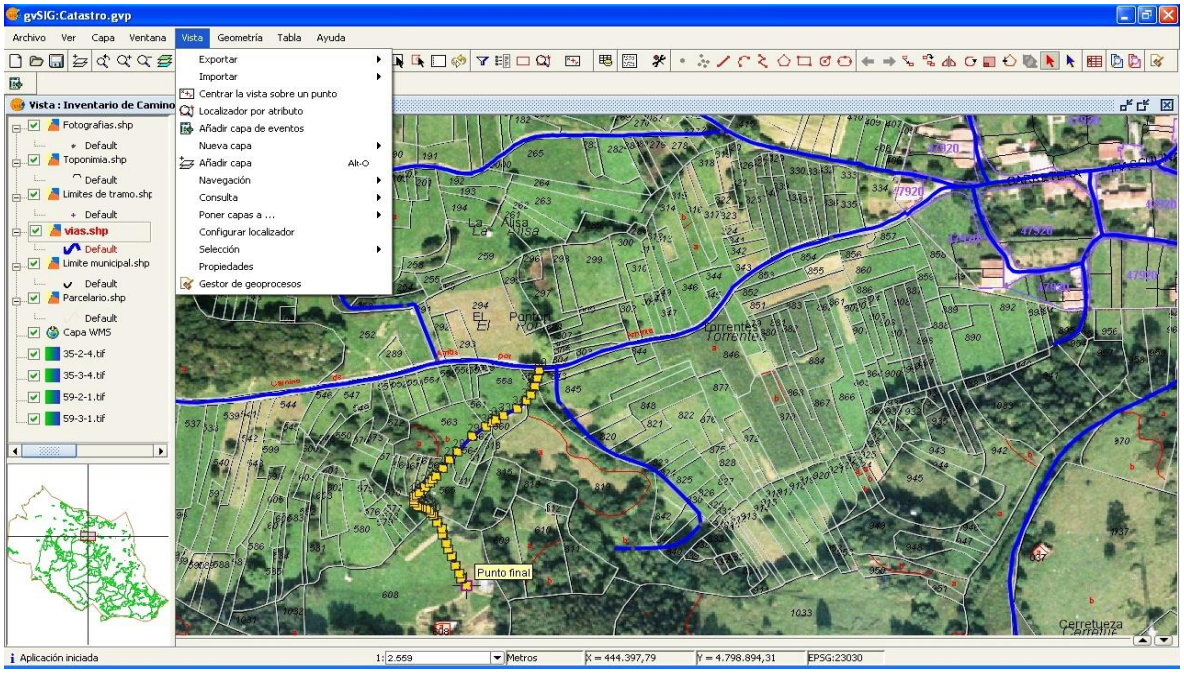
El Sistema de Soporte de Análisis de Recursos Geográficos, comúnmente conocido como **GRASS** GIS, es un Sistema de Información Geográfica (SIG) utilizado para la gestión de datos, procesamiento de imágenes, producción de gráficos, modelado espacial y visualización de muchos

tipos de datos. Es Libre (Libre) Software / Open Source publicado bajo GNU General Public License (GPL. GRASS GIS es un proyecto oficial de la Open Source Geospatial Foundation.



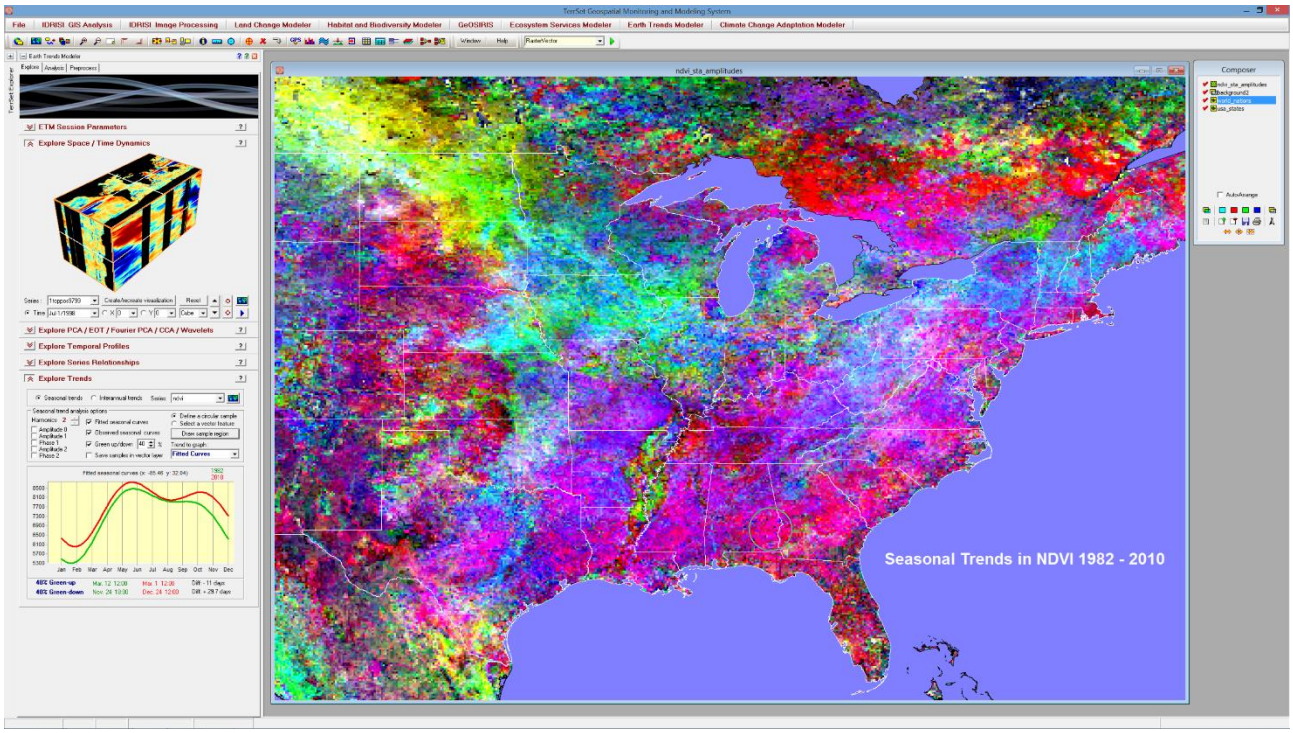
GRASS GIS

gvSIG Desktop es un programa informático para el manejo de información geográfica con precisión cartográfica que se distribuye bajo licencia GNU GPL v3. Permite acceder a información vectorial y rasterizada así como a servidores de mapas que cumplan la especificaciones del OGC.



gvSIG

TerrSetes un sistema integrado de información geográfica (GIS) y software de teledetección desarrollado por Clark Labs en la Universidad Clark para el análisis y visualización de información geoespacial digital. TerrSet es un sistema basado en C++, que ofrece herramientas para investigadores y científicos dedicados al análisis de la dinámica de sistemas de tierra para una toma de decisiones efectiva y responsable para la gestión ambiental.



TerrSetes

3.2 Objetivos

Los usuarios quieren ser capaces de hacer uso de los mapas con los modernos dispositivos electrónicos, como los smartphone y las tabletas, o los simples navegadores web, e integrarlos directamente en las aplicaciones de uso más extendido. Abriendo la puerta a la creación de un ambiente participativo donde las herramientas SIG sean más accesibles y flexibles.

Se pretende desarrollar una herramienta para añadir comentarios de puntos estratégicos y visualizar la información de redes de agua en terrenos geográficos, sobre una aplicación web con una base de datos alfanuméricos estáticos. Como resultado se obtendrá un mapa interactivo y responsivo que permitirá visualizar con facilidad la información de redes de agua, elementos geográficos, cargar los servicios WMS y una búsqueda por atributos

4. Competencias:

A continuación, se justificarán las competencias específicas cubiertas a lo largo del proceso de este proyecto. Se listarán las distintas competencias, su definición, y el lugar en las que se ven realizadas.

C01

Capacidad para proyectar, calcular y diseñar productos, procesos e instalaciones en todos los ámbitos de la ingeniería informática.

Esta competencia ha sido desarrollada en la sección Diseño e implementación. En ese capítulo se muestran las decisiones tomadas con respecto al diseño y desarrollo implementado. Se contempla todos los puntos del proceso desde la captura de requisitos hasta la obtención del producto final.

DG3

Capacidad para la dirección de proyectos de investigación, desarrollo e innovación, en empresas y centros tecnológicos, con garantía de la seguridad para las personas y bienes, la calidad final de los productos y su homologación.

Mediante la metodología iterativa, y a través de las pruebas y evaluaciones del software, se garantizaron la seguridad, calidad y robustez del producto final.

TI01

Capacidad para modelar, diseñar, definir la arquitectura, implantar, gestionar, operar, administrar y mantener aplicaciones, redes, sistemas, servicios y contenidos informáticos.

Como se puede ver en el aparatado de desarrollo, se consiguió implantar una aplicación web en su totalidad en un servidor virtual. A lo largo de este documento se justifica la competencia especifica.

TI02

Capacidad de comprender y saber aplicar el funcionamiento y organización de Internet, las tecnologías y protocolos de redes de nueva generación, los modelos de componentes, software intermediario y servicios.

Esta competencia se cumplió mediante las peticiones para obtener la información de los mapas, cargar los servicios WMS, y los filtros para la política del mismo origen. Ver en conceptos CORS.

TI05

Capacidad para analizar las necesidades de información que se plantean en un entorno y llevar a cabo en todas sus etapas el proceso de construcción de un sistema de información.

En la sección de desarrollo de la aplicación se creó un base de datos geoespacial para modificar los campos, ya estos requerían características especiales.

TI11

Capacidad para conceptualizar, diseñar, desarrollar y evaluar la interacción persona–ordenador de productos, sistemas y servicios informáticos.

Como se puede ver en la sección Diseño e implementación, se desarrolla una interfaz intuitiva al usuario, cómoda de usar y con el apoyo de OpenLayers para realizar tareas complejas de SIG.

TI12

Capacidad para la creación y explotación de entornos virtuales, y para la creación y distribución de contenidos multimedia.

En la sección de desarrollo del entorno de prueba, se describe y explica la implementación de una máquina virtual que alojara la aplicación.

5. Aportaciones:

El planteamiento inicial de este trabajo se estableció como una solución a las necesidades de tener una aplicación ligera y flexible, que pudiera ser usada en cualquier dispositivo, para la visualización de redes de agua. Cualquier usuario puede hace uso de esta herramienta y está diseñada de forma que se pueda seguir mejorando y añadiendo funcionalidades en un futuro y según las necesidades de sus usuarios.

La aplicación ofrece las siguientes características:

- Poder consultar la información mostrada sobre el mapa desde cualquier lugar y cualquier dispositivo.
- Cargar servicios WMS de cualquier fuente y seleccionar las capas que quieran ser mostradas en el mapa.
- Los usuarios pueden añadir comentarios en los puntos de interés marcados por ellos mismos.
- Tiene la posibilidad de hacer diferentes tipos de zoom
- Medir áreas y longitudes sobre el mapa
- Dibujar figuras con diferente grosor y color.
- Mostrar las coordenadas en el punto marcada en el mapa.
- Buscar por calle y portal la dirección deseada.
- Una selección de mapas temáticos con los que comparar la información ya por defecto o nueva añadida por el usuario.
- Estilos diferentes en las redes de agua para facilitar al usuario la observación.
- Cada capa ya sea introducida por el servicio WMS o por defecto posee las siguientes propiedades.
    - Ocultar y mostrar la capa en el mapa
    - Modificar la transparencia de la misma
    - Obtener la información general de la capa.
- Cambiar la vista del mapa base entre mapa político y satélite.

Todas estas características se implementaron en una interfaz cómoda e intuitiva. A demás, gracias al diseño modular y abierto es posible seguir mejorando y añadiendo funciones a la aplicación.

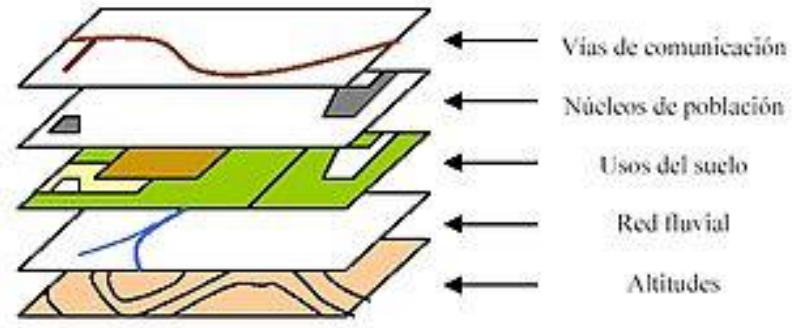
6. Conceptos de la geomántica:

Para facilitar la comprensión de este documento, se explicarán varios conceptos:

Base de datos espacial

Es una base de datos que maneja datos existentes en un espacio o datos espaciales. Se utilizan para definir la localización y relación entre objetos, ya que este tipo de dato poseen un valor relativo y no absoluto. Para referenciar los objetos normalmente se utiliza del tipo georreferenciado (aquellos que se establecen sobre la superficie terrestre), aunque existen excepciones.

La construcción de una base de datos geográfica implica un proceso de abstracción para pasar de la complejidad del mundo real a una representación simplificada que pueda ser procesada por el lenguaje de las computadoras actuales. Generalmente en capas; en esta fase, y dependiendo de la utilidad que se vaya a dar a la información a compilar, se seleccionan las capas temáticas a incluir.



Información del mundo real realizado en capas

Datos Geográficos

Los datos geográficos, presentan la información en representaciones subjetivas a través de mapas y símbolos, que representan la geografía como formas geométricas, redes, superficies, ubicaciones e imágenes, a los cuales se les asignan sus respectivos atributos que los definen y describen.

Capa

La idea de capa permite dividir la información espacial referida a una zona de estudio en varios niveles, de tal forma que, pese a coincidir sobre un mismo emplazamiento, información sobre distintas variables se encuentra recogida de forma independiente. Es decir, en función de la componente temática se establecen distintos bloques de datos espaciales.

Proyecciones

Cuando hablamos de proyecciones en SIG nos referimos a un sistema de representación gráfica que establece una relación ordenada entre los puntos de la superficie curva de la Tierra y

los de una superficie plana (mapa). Estos puntos se localizan auxiliándose en una red de meridianos y paralelos, en forma de malla.

Existen diversas proyecciones dependiendo de la zona, cuál sea el punto que se considere el centro, el tipo de geometría, etc. La más usada es la proyección de Mercator (EPSG:3857) junto con EPSG:4326 que, aunque no son demasiado precisas en los detalles, son esencialmente útiles para ver la superficie de la Tierra completa. Otra de las proyecciones que veremos en el documento es EPSG:32628, que se centra sobre las Islas Canarias, dando así mayor exactitud a las medidas.

Servicio WMS

El servicio Web Map (WMS) definido por el OGC (Open Geospatial Consortium) produce mapas de datos referenciados espacialmente, de forma dinámica a partir de información geográfica. Las operaciones pueden ser invocadas usando un navegador estándar realizando peticiones en la forma de URLs (Uniform Resource Locators).

Al solicitar un mapa, la URL indica qué información debe ser mostrada en el mapa, qué porción de la tierra debe dibujar, el sistema de coordenadas de referencia, y la anchura y la altura de la imagen de salida. Ejemplo:

http://idecan1.grafcan.es/ServicioWMS/MTL?request=GetCapabilities&service=WMS

Después del carácter '?' viene las peticiones, no importa el orden en el que se escriban. En el ejemplo vemos una petición básica donde se pide las propiedades y el tipo de servicio. Es posible demandar más atributos, pero en esta aplicación no es necesario ya que lo hace de forma automática.

Coordenadas geográficas

Las coordenadas de posición horizontal utilizadas son la latitud y longitud, un sistema de coordenadas angulares esféricas o esferoides cuyo centro es el centro de la Tierra y suelen expresar en grados sexagesimales. Por ejemplo '28° N -15° E' (Las Palmas de Gran Canaria).

Coordenadas UTM

El sistema de coordenadas UTM (en inglés Universal Transverse Mercator) es un sistema de coordenadas basado en la proyección cartográfica de Mercator. A diferencia del sistema de coordenadas geográficas, expresadas en longitud y latitud, las magnitudes en el sistema UTM se expresan en metros únicamente al nivel del mar, que es la base de la proyección del elipsoide de referencia, con eje X e Y. Ejemplo 'X:456590 Y:3109939' (Las Palmas de Gran Canarias)

CORS

La política del mismo origen es una medida importante de seguridad para scripts en la parte cliente (casi siempre JavaScript). Previene que un documento o script cargado en un "origen" pueda cargarse o modificar propiedades del documento desde un "origen" diferente.

7. Herramientas utilizadas:

A lo largo de este proyecto se hicieron uso de diversas herramientas para el desarrollo de la misma. A continuación, se especificarán detalles sobre su uso, funcionalidades, características particulares y cualquier otro aspecto que facilite la comprensión.

OpenLayers

Es una biblioteca para mostrar mapas interactivos y dinámicos en cualquier página web. Puede mostrar mosaicos de mapa, datos vectoriales y marcadores cargados desde cualquier fuente.

Está basado en el leguaje JavaScript de código abierto, con una licencia BSD. Se caracteriza de otras herramientas de su entorno por tener una gran comunidad de usuarios y desarrolladores que día a día mejoran esta biblioteca. Ofrece una amplia gama de opciones para los servicios de mapas más conocidos, haciendo que su uso sea simple e incluso permite la mezcla de información obtenida desde distintos servidores.

VirtualBox

Es un software de virtualización para arquitecturas x86/amd64. Por medio de esta aplicación es posible instalar sistemas operativos adicionales, conocidos como «sistemas invitados», dentro de otro sistema operativo «anfitrión», cada uno con su propio ambiente virtual.

Posee una licencia GPL, es decir libre, compatible con la mayoría de los sistemas operativos, un programa estable con mucha documentación y se puede emplear con múltiples configuraciones muy fáciles de utilizar.

PostgreSQL

Es un Sistema de gestión de bases de datos relacional orientado a objetos. Como servidor de base de datos, sus funciones principales son almacenar datos de forma segura y devolverlos en respuesta a solicitudes de otras aplicaciones de software. Puede manejar cargas de trabajo que van desde pequeñas aplicaciones de una sola máquina hasta grandes aplicaciones orientadas a Internet con muchos usuarios simultáneos.

PostGIS

Es un módulo que añade los tipos de datos GIS, que aportan soporte de objetos geográficos al sistema de gestión de base de datos relacionales PostgreSQL. Almacena la información geográfica en una columna del tipo GEOMETRY, que es diferente del homónimo “GEOMETRY” utilizado por PostgreSQL, donde se pueden almacenar la geometría

GeoServer

GeoServer pretende operar como un nodo a través de una Infraestructura de Datos Espaciales libre y abierta para ofrecer datos geoespaciales. Es de código abierto (GPL), y permite a los usuarios compartir y editar datos geoespaciales. Diseñado para la interoperabilidad, publica datos de las principales fuentes de datos espaciales usando estándares abiertos.

HTML y CSS

HTML, sigla en inglés de HyperText Markup Language (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros.

Hojas de estilo en cascada (o CSS, siglas en inglés de Cascading Stylesheets) es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Está diseñado principalmente para marcar la separación del contenido del documento y la forma de presentación de este, características tales como las capas o layouts, los colores y las fuentes. Esta separación busca mejorar la accesibilidad del documento, proveer más flexibilidad y control.

PHP

PHP es un lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico. Fue uno de los primeros lenguajes de programación del lado del servidor que se podían incorporar directamente en el documento HTML en lugar de llamar a un archivo externo que procese los datos.

JavaScript y jQuery

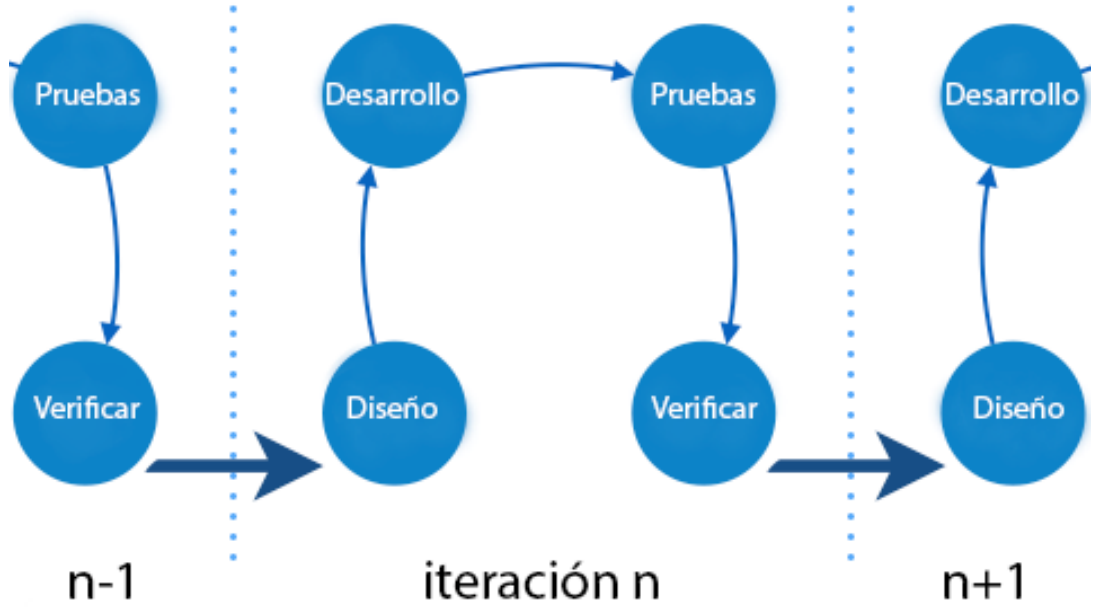
JavaScript es un lenguaje de programación interpretado orientado a objetos, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.

jQuery es una biblioteca de JavaScript que contiene las funcionalidades comunes de DOM, eventos, efectos y AJAX. La característica principal de la biblioteca es que permite cambiar el contenido de una página web sin necesidad de recargarla, mediante la manipulación del árbol DOM y peticiones AJAX. Para ello utiliza las funciones '$()'o 'jQuery()'.

8. Análisis y diseño:

8.1 Metodología

En este proyecto se empleó el modelo de desarrollo iterativo y creciente, que no es más que un conjunto de tareas agrupadas en pequeñas etapas repetitivas (iteraciones). Debido a que, al tratarse de un prototipo, cuando se cubrieron los requerimientos mínimos deseados se empezó a añadir funcionalidades, testearlas, verificar el correcto funcionamiento de todo y vuelta a la adición de funcionalidades.



Metodología iterativa

8.2 Plan de trabajo

Las etapas a cubrir durante la realización del proyecto fueron las siguientes:

- Análisis: Se planteó las necesidades de la aplicación para visualizar las redes de agua. Se llevó a cabo un estudio para implementar el proyecto mediante software libre y compatible entre sí. También hubo un periodo de pruebas con el software y lectura de documentación para familiarizarse con este.

- Desarrollo: En un principio se implementó la herramienta con las primeras funcionalidades, con una interfaz básica. Luego se fueron añadiendo más funciones hasta que se estuvo satisfecho con la aplicación. Seguidamente se desarrolló por completo la interfaz final.

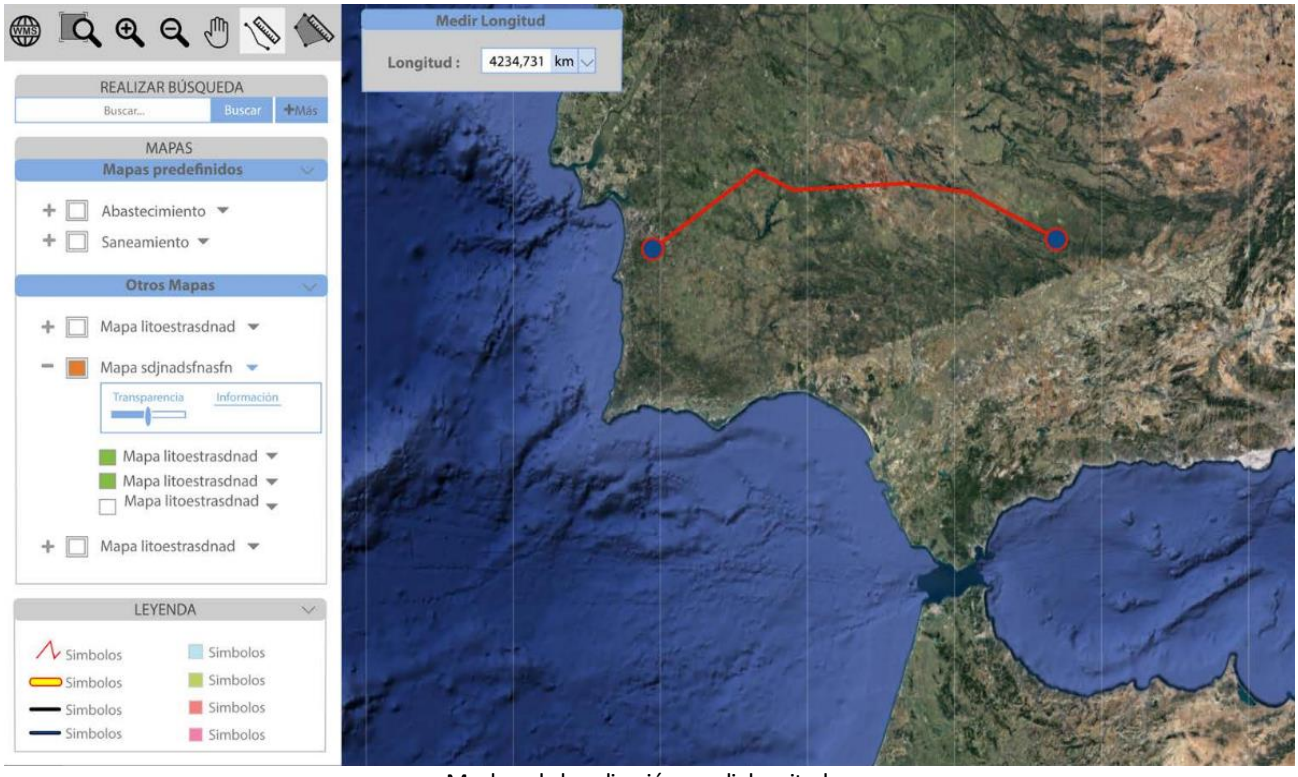
- Evaluación: Cada vez que se terminaba una funcionalidad se probaba y evaluaba por separado y en conjunto a la interfaz. Además, para probar su flexibilidad, las pruebas finales se realizaron en varios dispositivos con distintos exploradores.

- Documentación: Una vez terminado la aplicación se empezó la confección de este documento junto con la preparación de la defensa.

8.3 Diseño de la aplicación

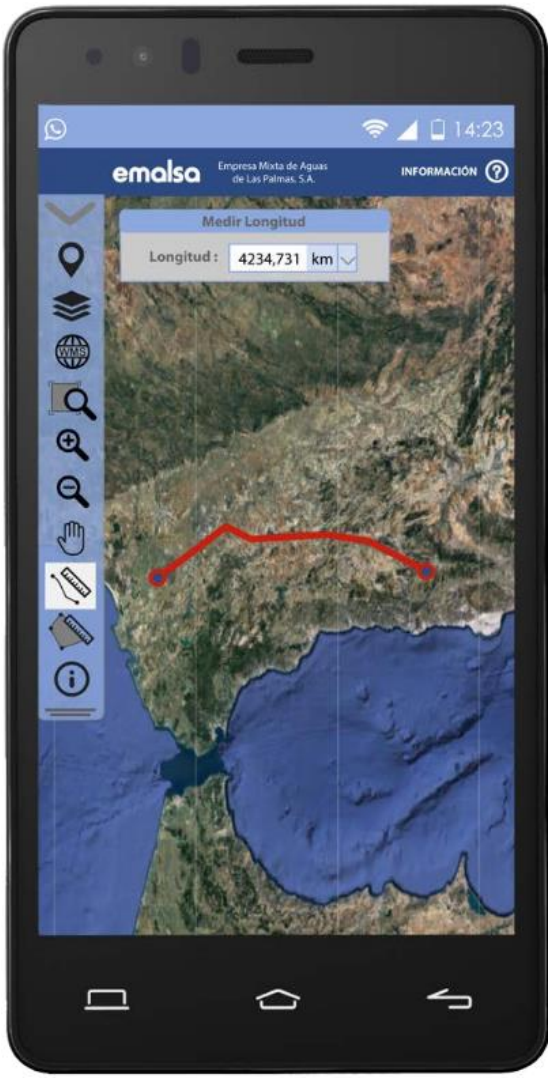
La interfaz debe ser simple, facilitándoles una vista sencilla, clara y concisa. Dividir en dos partes la web es una buena idea para juntar las funciones y centra su atención en pocos objetivos. También hay que tener en cuenta la experiencia de manejar otros visores. Hay símbolos y acciones que tienen una herencia, se ha hecho así y se ha mantenido durante los años. Por ejemplo, aumentar el zoom es una lupa con un símbolo más dentro. Hay que tener en cuenta las animaciones de la web, dando al usuario una impresión dinámica de la aplicación. Incorporar detalles es otro método para que el usuario se sienta cómodo y se dé cuenta de lo completa que es la web.

Para que un diseño web sea efectivo, debe lograr que los usuarios del sitio puedan acceder con facilidad a los contenidos, interactuar con eficacia con todos los componentes y sentirse cómodo en forma permanente. Se hicieron unos mockups con la idea de que la comunicación con el usuario se desarrolle de la forma más fácil y cómoda posible para las características del usuario que utiliza el servicio.



Mockup de la aplicación, medir longitudes

La aplicación se cambió durante su desarrollo debido a que los requisitos alteraban la interfaz de esta. Cuando se planteó la herramienta no había tantas funciones por lo que había menos botones en el menú. Al ir añadiendo estas modificaciones se consideró que lo mejor era agrupar los botones por su funcionalidad y mostrarlos en un desplegable. Debido a que se pueden añadir mapas, una leyenda no es suficiente, se introdujo en la información general de cada capa su propia leyenda. El mensaje con las medidas en un principio solo mostraba el último dibujo que se realizaba. Dando un paso más, se implanto un a función que añadía cada medida con su dibujo al mapa, visualizado todas, sin perder ningún tipo de información.



Mockup de la aplicación móvil.

El diseño de la aplicación de móvil quedo casi igual en el planteamiento del diseño. Esto se debe a que, aunque se cambiaran funciones, botones, paneles, información, etc. Hay que tener en cuenta que estas tienen unas propiedades más restringidas, debido a que su pantalla posee una resolución menor. Por ejemplo, cuando se muestra información de los mapas se despliega un panel con esos datos, es la mejor forma de plantearlo ya que no hay espacio suficiente para ampliar la interfaz.

En resumen, se modificaron muchas de las herramientas para que la interfaz se adaptara al diseño final, otras no se alteraron en absoluto. Incluso algunas no estuvieron tenidas en cuenta en el diseño previo por lo que se tuvieron que adaptar en el momento.

9. Desarrollo:

9.1 Implementación del entorno de prueba

En primer lugar, instalamos el programa VirtualBox con el que creamos una máquina virtual con el sistema operativo Ubuntu 16.04. En este entorno desarrollaremos y pondremos a prueba la aplicación web que llevaremos a cabo. Gracias a esto tenemos un entorno controlado, y en caso de que tengamos que restaurar por algún motivo se podrá hacer fácilmente. También podemos aumentar la flexibilidad del servicio, ya que es muy fácil administrar los recursos usados por la máquina virtual, modificándolos para su óptimo funcionamiento.

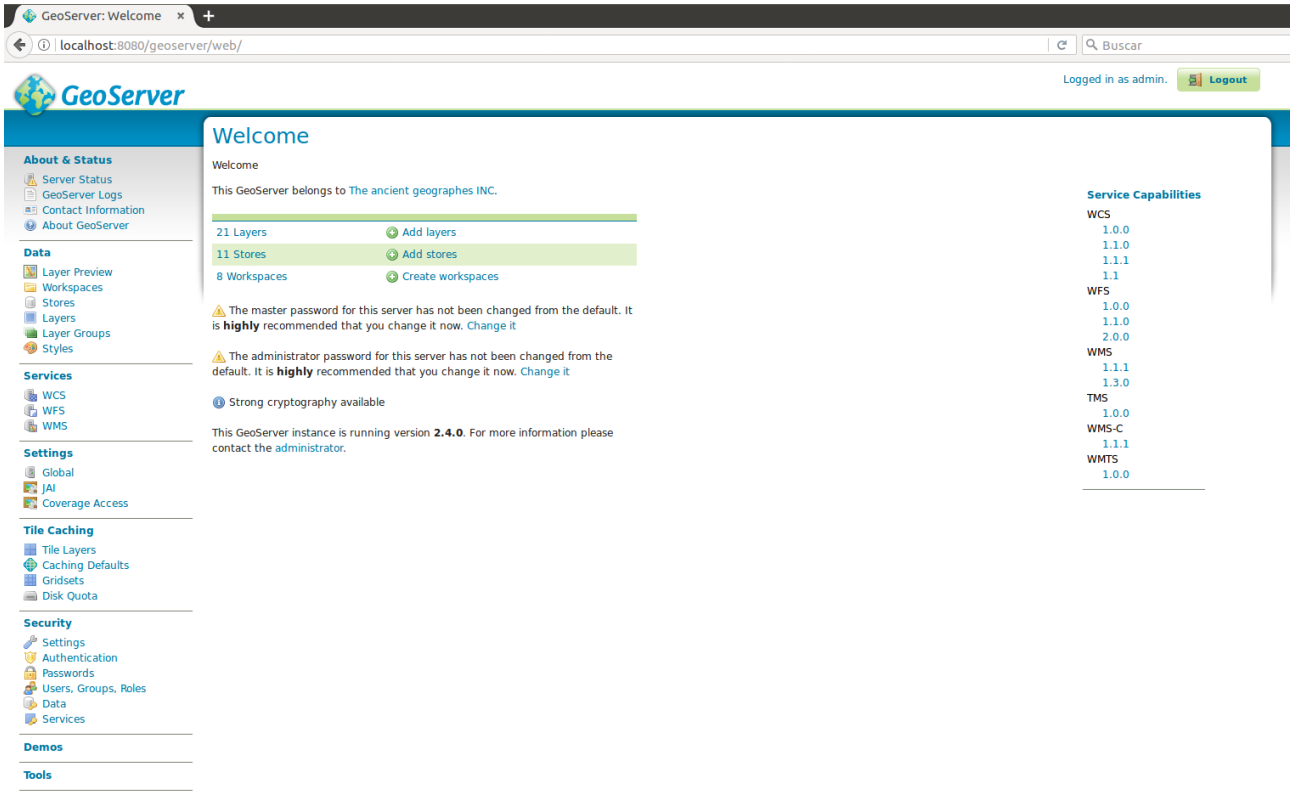
El siguiente paso es obtener información sobre las versiones más recientes de los paquetes y sus dependencias, e instalar PostgreSQL nuestra base de datos orientado a objetos. Ahora se instala el PostGIS, esto añade a la base de datos PostgreSQL soporte de objetos geográficos convirtiéndola en una base de datos espacial. Se harán peticiones de POST y GET a la base de datos, por lo que instalar la última versión de PHP en nuestro servidor web para resolver estas peticiones.

Para habilitar las dependencias entre PHP y PostgreSQL hay que habilitar una extensión de PHP. Con esto podremos conectarnos a una base de datos PostgreSQL. Instalaremos el Apache Tomcat, ya que es el servidor HTTP que usa GeoServer, y es donde alojaremos nuestra aplicación web. El siguiente filtro no se descubrió que se requería hasta más adelante, pero para no desordenar la composición ni estructura de este documento es mejor describirlo y explicarlo aquí. Debido a que estamos intentando cargar un documento desde un origen diferente debemos definir la política del mismo origen en Apache y Tomcat. Politica del mismo origen (CORS)

Por defecto Apache tiene habilitadas las cabeceras por lo solo hay que crear un archivo .htaccess en la raíz de nuestra aplicación web /var/www/html especificando la política del mismo origen.

Habilitamos un módulo de Apache para hacer accesibles las diferentes aplicaciones de Tomcat al exterior a través del puerto estándar HTTP (80). Este mapeo de direcciones se hará mediante la extensión *proxy_ajp* de Apache. Reiniciamos el Apache para que los cambios efectuados tengan efecto.

Comprobamos que todo funciona por ahora entrando en http://localhost:8080/geoserver/web/
En el GeoServer viene por defecto con el usuario: admin y la contraseña: geoserver.



Interfaz gráfica de GeoServer

Los datos de la red de aguas están almacenados en una copia de seguridad, que se nos ha sido entregada en un archivo *.backup*. Antes creamos la base de datos 'tfm', el esquema 'abastecimiento'. Para añadir la extensión PostGIS a nuestra base de datos PostgreSQL usamos el siguiente comando, ahora la base de datos 'tfm' posee tipos de datos espaciales, índices espaciales y funciones que operan sobre ellos. Ya tenemos la tabla creada, ahora procedemos a insertar los datos de la copia ya mencionada antes. Se insertan en la base de datos tfm en la tabla tubería.

Una de las funciones de la aplicación es buscar una calle y centrar la vista en ella. Para ello se nos ha proporcionado otras tablas con la información necesaria, por lo que tendremos que restaurar la copia igual que la última vez.

Esta vez la copia de seguridad ya viene con el código necesario para crear la tabla, debido a esto no hace falta crear con anterioridad la tabla. Como el nombre de las calles puede tener acentos y caracteres especiales hay que relacionar estos con sus caracteres regulares. Obteniendo en las búsquedas las palabras con y sin condiciones especiales. *Unaccen*t ya viene definida por defecto en PostgreSQL, por lo que solo hay que extenderla a nuestras tablas de callejero.

Para que los usuarios puedan colocar comentarios propios en los puntos de interés crearemos la tabla comments. A continuación, publicaremos las tablas tuberías y comments en el GeoServer, así Openlayers podrá acceder a los datos y administrar las configuraciones de las

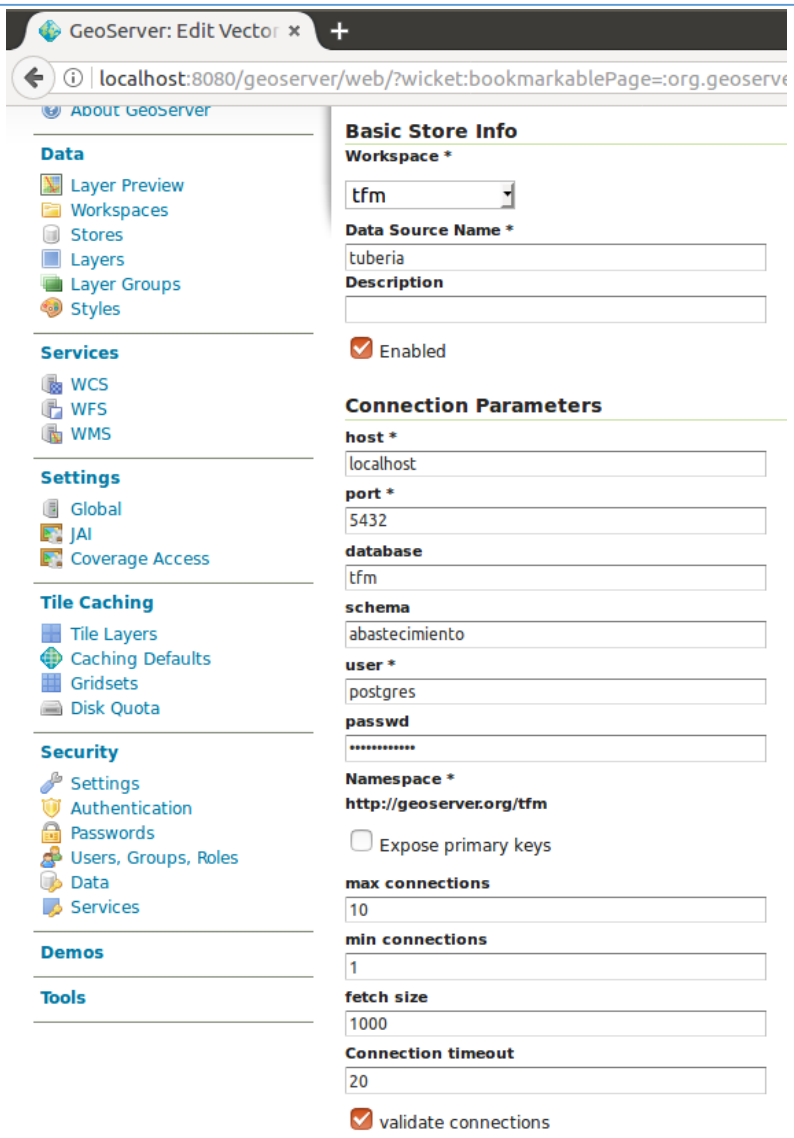
respectivas tablas. Estos pasos hay que realizarlos por cada una de las publicaciones que hagamos, una para tubería y otra para comments.

```
                                    Table "abastecimiento.tuberia"
        Column         |            Type             |                            Modifiers
-----------------------+-----------------------------+-------------------------------------------------------------------
 geom                  | geometry(LineString,32628)  |
 ipid                  | bigint                      | not null default nextval('abastecimiento.tuberia_seq'::regclass)
 estado_validacion     | text                        | not null default 'pendiente_validacion'::text
 fecha_digitalizacion  | timestamp with time zone    | default now()
 fecha_instalacion     | timestamp with time zone    |
 fecha_modificacion    | timestamp with time zone    |
 ignorar_validaciones  | boolean                     | not null default false
 revalidar             | boolean                     | not null default false
 usuario_digitalizacion | text                       |
 usuario_modificacion  | text                        |
 borrado               | boolean                     | not null default false
 contrato_explotacion  | text                        | not null default 'Las Palmas de GC'::text
 depurada              | boolean                     | not null default false
 estado_conservacion   | text                        | not null default 'desconocido'::text
 estado_servicio       | text                        |
 estilo                | text                        |
 fuente_datos          | text                        | not null default 'desconocido'::text
 incendio              | boolean                     | not null default false
 link_descripcion      | text                        |
 observaciones         | text                        |
 privada               | boolean                     | not null default false
 riego                 | boolean                     | not null default false
 sector                | text                        | not null default 'desconocido'::text
 tipo_red              | text                        | not null
 transporte            | boolean                     | not null default false
 id_activo             | integer                     |
 altura_seccion        | real                        |
 diametro_equivalente  | real                        |
 diametro_mayor        | text                        |
 diametro_menor        | text                        |
 gravedad              | boolean                     |
 longitud              | real                        |
 material              | text                        | not null default 'desconocido'::text
 pasatubo              | boolean                     | not null default false
 presion_nominal       | text                        | not null default 'desconocido'::text
 tipo_seccion          | text                        | not null default 'circular'::text
 virtual               | boolean                     | not null default false
Indexes:
    "tuberia_pkey" PRIMARY KEY, btree (ipid)
    "spatial_index_tuberia" gist (geom)
```

Tabla tubería en PostgreSQL

Nos dirigimos a *Data –> Workspace*, y hacemos clic en *Add new workspace*. Rellenamos los datos de nombre del workspace y nombre URI (Uniform Resource Identifier), es normalmente una URL asociada al proyecto.

Habrá que crear un store en *Data –>Stores*, aquí nos da a elegir una serie de opciones para el tipo de datos en el nuevo almacén. Nuestro caso es *PostGIS Database*, es decir, el tipo de las tablas en la base de datos. Ahora hay que rellenar una información básica, seleccionamos el espacio de trabajo que creamos anteriormente, el nombre de la tabla que queremos publicar y una breve descripción de los datos. En los parámetros de conexión se pondrán los siguientes datos, dejando los otros por defecto.

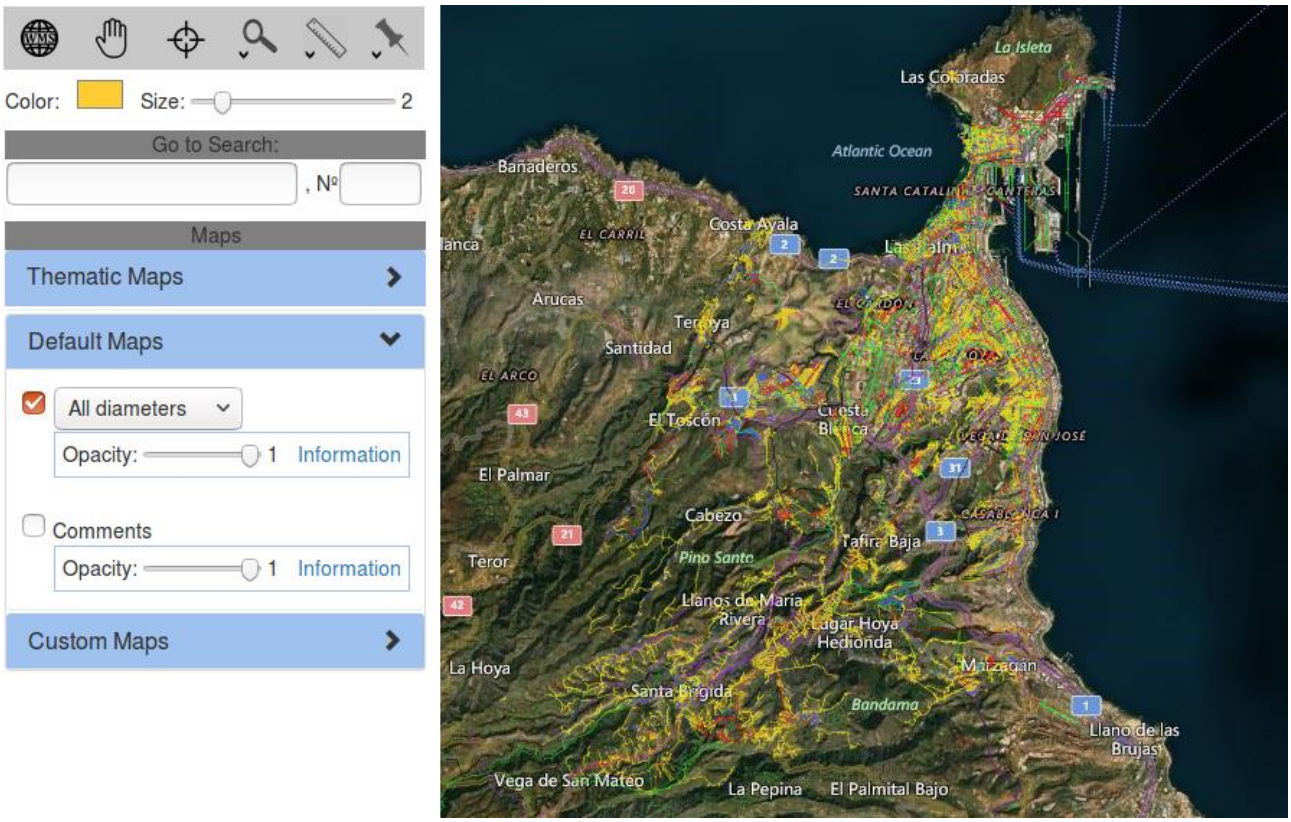


Añadir un workspace en GeoServer

Por último, añadiremos una nueva capa en *Data–>Layers–>Add new resource*. En el selector elegiremos la *Workspace/Store* que acabamos de crear y hacemos clic en *Edit Layer*. En la pestaña *Data* hay mucha información que se puede rellenar, la mayoría son opcionales, lo importante es la opción *Bounding Boxes*. Podemos introducir a mano la información, pero es recomendable usar el enlace *Compute from data,* esta analiza la geometría de la base de datos introducida y calcula el cuadro a delimitar para la capa. En la pestaña *Publishing* podemos elegir el tipo de estilo con el que se representa la capa, normalmente GeoServer escoge la mejor dependiendo de la geometría, más adelante se añadirán algunos.

Una vez terminado, si queremos comprobar que todos los pasos se han realizado correctamente podemos ver una vista anticipada de la capa. En el menú principal vamos a *Layer Preview* y buscamos la capa que deseamos visualizar, hacemos clic en el enlace Openlayers. Si es correcto veremos nuestra capa sobre un fondo blanco y unas opciones que trae por defecto Geoserver.

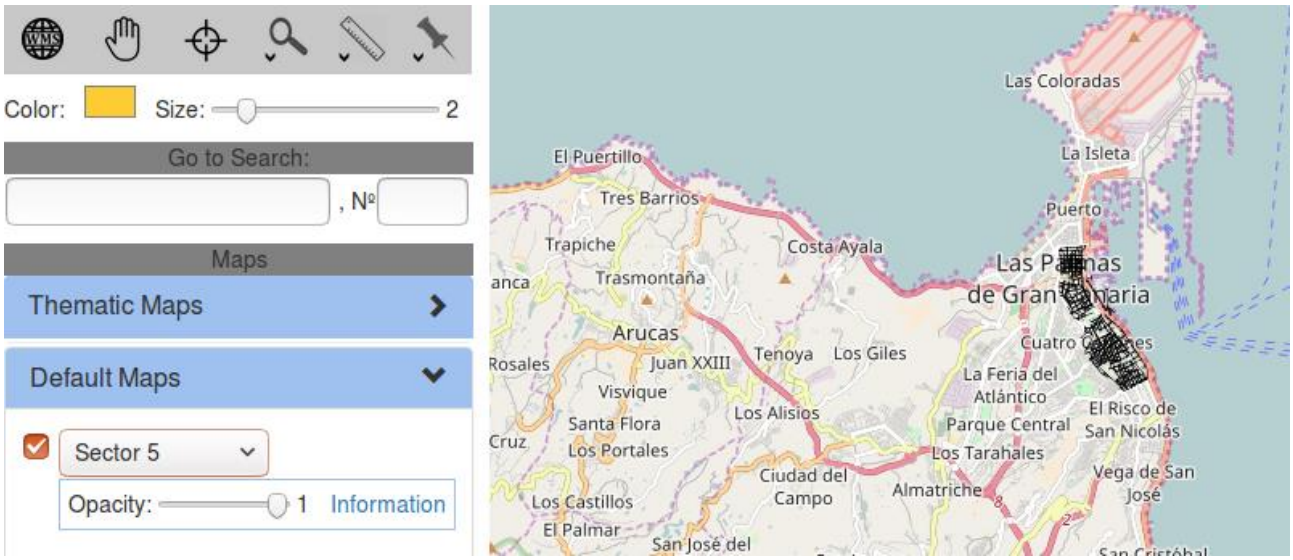
Ahora se añaden estilos o reglas a la capa tubería, para que el usuario pueda observar la información de forma fácil y eficiente. Para agregar estos estilos hay que dirigirse a *Data –>Styles* y hacemos clic en el enlace *add a new style.* Introducimos el nombre del estilo, su espacio de trabajo, en este caso *tfm,* y se escribe el código para que tubería tenga las propiedades deseadas. Como ejemplo se explica la regla para ver las tuberías que tengan de diámetro entre 175 y 250, que estén en servicio activo.



Estilo de tubería con todos los diámetros a la vez

En primer lugar, se definen una serie de propiedades, esquemas y definiciones necesarias para el código. A continuación, introducimos el nombre de la capa a la que se aplica la regla, junto con su nombre. Añadimos un filtro con las reglas que la propiedad con el nombre 'diámetro_equivalente' tiene que tener un valor mayor a 175 y ser menor de 250. También se especifica que la tubería tiene que tener el 'estado_servicio' en 'en_servicio'. Por último, solo queda establecer el estilo con el que se expondrán en el mapa las tuberías seleccionadas con las reglas anteriores. *MinScaleDenominator,* determina hasta que zoom mínimo se pueden ver estos estilos. Las unidades de las líneas se disponen en metros, color verde, un grosor de 0.3, cuando se unen dos líneas se inserta una equina en diagonal, y cuando las líneas terminan se introduce un cuadrado del mismo ancho. Así con todos los diámetros que se quieran

También se creó estilos por sectores. Se observan las tuberías de cada sector diferente de la ciudad.

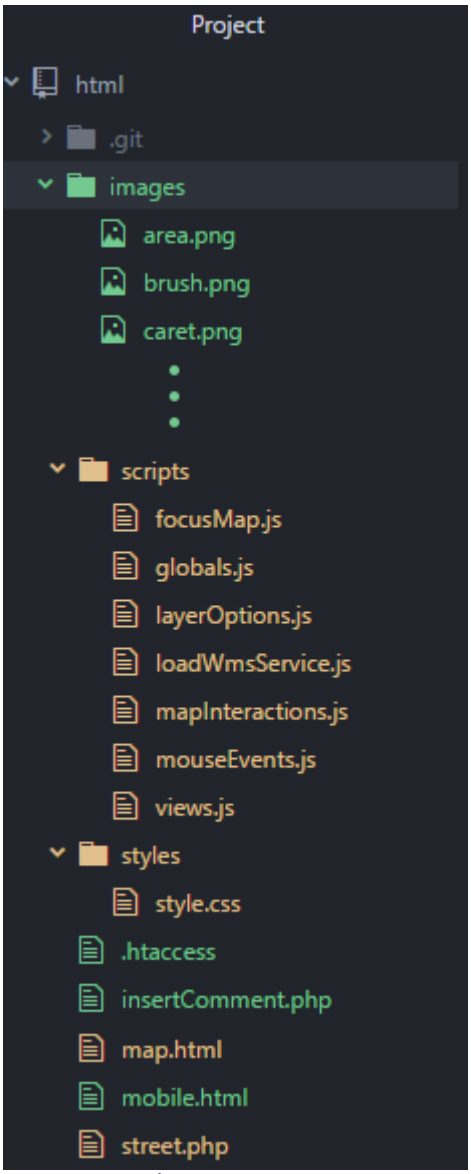


Estilo de tubería, pertenecientes al sector cinco

Se han creado 21 estilos diferentes para tubería, si tiene mayor interés puede observar el resto en el siguiente repositorio.
https://github.com/Eduardofarinos/WebViewerTFM/tree/master/tuberia-styles

9.2 Desarrollo Aplicación Web

La web está alojada en /var/www/html, dividida en varias subcarpetas para ordenar los ficheros según su extensión. Las imágenes que usaremos estará en la carpeta 'images', los archivos JavaScript en 'scripts', los estilos CSS en 'styles', y en la raíz de nuestra web 'map.html', 'mobile.html', 'street.php' e 'insertComment.php'.



Árbol del proyecto

Carpeta *images*:

En esta carpeta almacenaremos las imágenes que la aplicación web usa. Todas son para los botones del menú que activan las distintas funciones.

Carpeta *scripts*:

Aquí colocaremos las funciones en código JavaScript. Según su cometido se han organizado en siete ficheros *focusMap.js, layerOptions.js, mapInteractions.js, mouseEvents.js, views.js, loadWmsService.js* y *globals.js*.

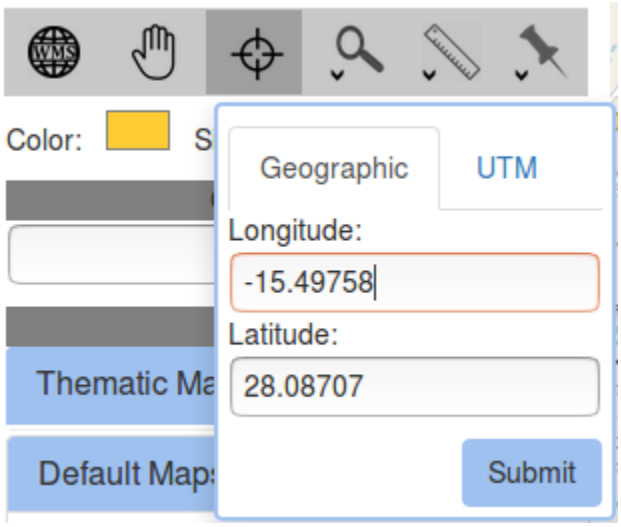
focusMap.js

El archivo *focusMap.js* contiene las funciones que se centrarán en ajustar la vista del mapa, es decir, el zoom, la localización del centro del mapa o puntos específicos en el mapa.

centerGeo y *centerUtm,* centran la vista del mapa en las coordenadas que el usuario introduce mediante un formulario.

- *centerGeo* es una variable que registra el evento cuando se hace clic en el botón identificado con 'GoTo_geo'. La función recoge las variables *longitude* y *latitude* que ha introducido el usuario, se comprueba que los valores son correctos con *isEmpty*. Si no lo son, se mostrará un mensaje de alerta que los parámetros son incorrecto, si son correctos hay que transformar las coordenadas dadas por el usuario a la proyección del mapa. Haciendo hincapié en el uso de *parseFloat* ya que es posible que en el transito la variable pueda ser modificada, atribuyendo así un resultado erróneo. Por último, situamos el centro del mapa en las coordenadas con un zoom de 18 por defecto.

- *centerUtm* tiene el mismo empleo y funcionamiento que *centerGeo*, pero en lugar de las coordenadas geográficas, son UTM 'x' e 'y'. Trasformadas a la proyección correcta.



Formulario para ir a las coordenadas especificas

- El mapa de Openlayer por defecto tiene la opción de si dejas apretado el botón *Shift* del teclado y arrastras con el ratón saldrá un recuadro, indicando donde se acercará el mapa. Con la función *zoomDrag* modificamos esta interacción para que solo haya que arrastrar con el ratón.

  Primero eliminamos cualquier vínculo que haya con el mapa, por si quedara alguno. A continuación, si la función estaba activa y se hace clic otra vez quiere decir que el usuario

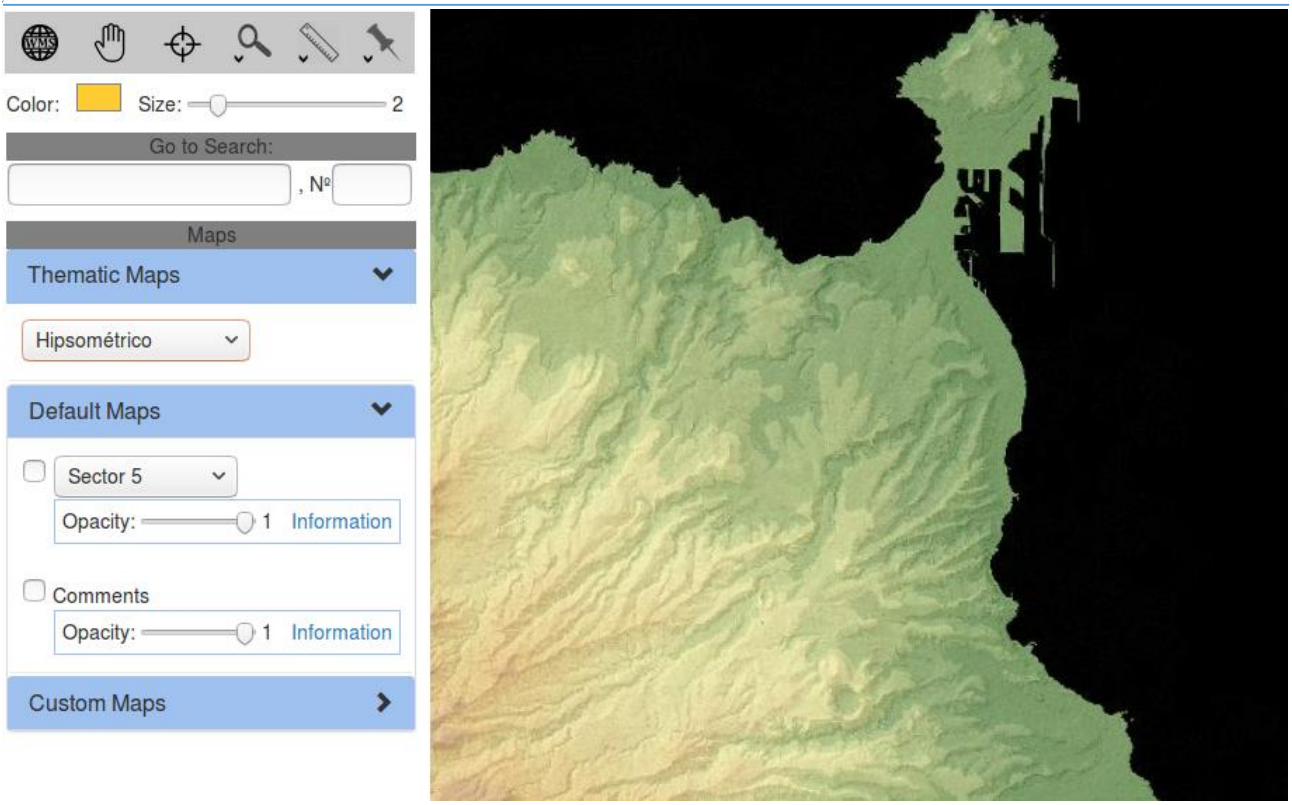
quiere desactivar esta opción, por lo que establecemos la interacción a falso, terminando la función. Por otro lado, si se quiere activar, cambiamos la imagen del curso por una lupa y especificamos que DragZoom no tenga tecla de modificación. Añadimos la nueva interacción al mapa y finalizamos verificando botones y funciones.

- *centerStreet* se le pasan las coordenadas (longitud y latitud) donde centrar la vista del mapa y un tercer parámetro indicando si se busca un portal o número de bloque concreto. Debido a que nuestra base de datos (callejero) está en una proyección diferente del mapa, las coordenadas se transforman a 'EPSG:3857'. El siguiente paso es preguntar si existe el tercer parámetro, en tal caso, añadiremos un circulo de cinco metros de radio para resaltar el lugar específico del portal. Para finalizar la función situamos el centro del mapa en las coordenadas, con un zoom lo suficiente cercano como para que se distinga la calle.

- *showHint* buscará en la base de datos la calle y portal que el usuario haya introducido. Es posible que la lista este oculta, debido a eso hay que volver a mostrar, cambiando el estilo a 'inline-block'. Como queremos que la información se vaya actualizando a medida que el usuario escriba, hay que tener en cuenta que si no se escribe nada o haya menos de tres letras no busque. Cuando se escriba más de tres letras se le envía una petición 'GET' al archivo *street.php* que se encargara de buscar y mostrar los resultados.

- *insertComm.* Cuando el usuario rellena el formulario para insertar un nuevo comentario esta recoge los datos, y se envía una petición 'POST' a *insertComment.php* con la nueva información a introducir. Para que el usuario tenga un feedback, hay que actualizar el mapa con el nuevo punto de interés insertado. También hay que borrar el formulario y el dibujo temporal del mapa.

- *selecThematic* cambia la capa base del mapa entre varias opciones. Cada vez que el usuario cambia entre estas capas en un selector, se llama a la función para que oculte todas las capas, y por ultimo solo hacemos visible la base con el nombre que se nos pasa por parámetro. Si no existe ese parámetro simplemente se ocultan todas las capas para que volver a la vista base.



Vista hipsométrica de los mapas temáticos

layerOptions.js

El fichero *layerOptions.js* son funciones que ayudan a manejar las opciones de visibilidad de las capas y sus datos correspondientes.

- *changeOpacity* cambiará la transparencia de la capa dada con el valor seleccionado por el usuario entre 0 y 1, dando pasos de 0,1. En primer y segundo lugar preguntamos si la capa seleccionada es alguna de las que tenemos por defecto (tubería o comments). Si es así entonces obtenemos el valor correspondiente que se le quiere dar para representarlo en el mapa. Sino entonces es una de las capas personalizadas, y hay que determinar el número de la capa, cortando los últimos caracteres de la Id, como está ordenado el mismo número de la posición del vector es el mismo que el identificador. Finalmente establecemos en el mapa la transparencia que marca el selector.

- La función *Visibility* se encarga esconder o enseñar la capa seleccionada en el mapa. Si el usuario marca el checkbox, la capa se hará visible, si lo desmarca se esconderá. Los dos primeros casos son para 'tubería' y 'comments'. En caso que sea otro Id, se trocea los dos últimos dígitos y fijamos la propiedad *setVisible* a verdadero o false según el usuario este marcando o desmarcando.

- *showInfo* inserta la información general de la capa selecciona en el modal, para luego mostrársela al usuario. En este caso también preguntamos por las capas por defecto, mostrando sus datos ya definidos, si es una capa personalizada buscamos en la matriz *info*. La información de esta matriz se obtiene al cargar la capa en *loadWmsServcie*. En el primer

campo de la matriz es el identificador de la capa, que al estar ordenado es el mismo que el de la capa. El segundo campo es cada uno de los datos, por ejemplo, el '0' es el título de la capa. Por ultimo solo queda insertar la información específica en cada elemento identificado en el HTML.

- *setStyle* es una función que maneja el estilo de la capa tubería, en función seleccione el usuario. Se pasa una variable por parámetro que es el nombre del estilo elegido. Al asignar este a tubería se tiene que actualizar su fuente con los nuevos parámetros para que los cambias se aprecien.

mapInteracions.js

El siguiente archivo es *mapInteractions.js* donde se manipulan los mensajes y dibujos que se hacen sobre el mapa.

- *addDraw* delimita, esboza y traza dibujos en el mapa dependiendo del tipo de figura que el usuario elija. Primero eliminamos cualquier vínculo que haya con el mapa, y dejamos el cursor por defecto. Si se ha llamado a la función y su checkbox correspondiente no está marcado, quiere decir que el usuario quiere dejar de dibujar por lo que restablecemos el cursor y no se hace nada. Sino, definimos una nueva interacción de tipo Draw con sus características:
  - *clickTolerance* es el número máximo de pixeles que considera como evento de clic.
  - *source* es la capa donde se almacenarán las propiedades de los dibujos.
  - *type* define el tipo de dibujo que se realizará, en nuestro caso habrá:
    - *lineString:* es una línea recta.
    - *polygon:* un polígono regular o irregular.
    - *circle:* un circulo.
    - *point:* un punto, que es lo mismo que un circulo, pero rellenado con un color.
  - *style* determinamos las formas y colores que tendrán las figuras mientras se dibuja
    - *fill:* rellenamos sin color a la figura mientras se hace la traza.
    - *stroke:* definimos que la silueta sea de color negro punteada.
    - *image:* creamos un circulo con un radio de cinco pixeles, negro transparente que reemplazar el cursor.

  Ahora que se ha definido el dibujo, añadimos la interacción al mapa y empezamos creando un mensaje donde se inicia la figura. Cuando se empieza a dibujar se llama la función *drawstart* que sigue la trazada del cursor en el dibujo. Se obtienen las propiedades de la figura y sus coordenadas, dependiendo de la figura que el usuario haya seleccionado se calcula su medida llamado a la función adecuda (*formatArea, formatLength* o *formatCircle*) y guardamos las últimas coordenadas del dibujo. Mostramos el cartel con las medidas, junto con un código HTML, esto es un enlace que llama a la función que borrará, en caso de que el usuario quiera, el esbozo. Representado por una 'x' en la parte superior derecha. Finalmente situamos el mensaje final en las últimas coordenadas guardadas anteriormente sobre el mapa.

  Cuando se termina de dibujar se llama a *drawend,* que añade en uno la variable *tooltipid* que utilizaremos más adelante para identificar el mensaje con la medida. Situamos

el mensaje en su posición final sobre el mapa, reiniciamos la variable *sketch* y *measureToolipElement* para dejarlas listas en el siguiente dibujo. Llamamos a la función que crea estos carteles para que la desvincule y se pueda utilizar en el próximo dibujo. Establecemos *featureID* como un identificador de la figura final, así podremos reconocer la figura más adelante. Por último, se revisan los botones y funciones con la función *toggleButtons.*

Tres dibujos básicos con sus medidas sobre el mapa

- *createMeasureToolip* tiene el cometido de crear el recuadro en el mapa con las medidas del dibujo creado por el usuario. Se comprueba si ya existe, si es así se desvincula y se crean el elemento que insertaremos en el mapa con el tipo, clase e identificador. Finalmente creamos una capa superpuesta con *ol.Overlay* en el mapa con las coordenadas del elemento creado anteriormente, añadiéndola en el mapa.

- *changeUnits,* el usuario tiene un selector para cambiar las unidades en que se ha realizado la medida, a metros o kilómetros. Primero obtenemos la Id del mensaje para saber cuál cambiar. Luego diferenciamos si se cambia para una longitud, el identificador empiezaria por 'l' de 'length', o un área. Si el valor del selector es una 'm' de metros se redondea y se divide, ya que el valor que se desea cambiar estaba en kilómetros. Si cambiamos a kilómetros se

redondea y se multiplica para obtener el resultado en metros. Para modificar las unidades en el área se realiza de forma similar, calculando de forma correcta para cada caso.

- Cuando se quiere calcular la longitud de una línea recta dibujada en el mapa, se llama a la función *formatLength.* Esta, redondea el resultado y lo pasa a metros, luego pone un identificador y se escribe el código HTML que irá en el mensaje con la medida, los Ids y el selector de las unidades.

- *formatArea* es similar a la anterior función solo que esta calcula el área de un polígono.

- *formatCircle* igual que las anteriores, dado el radio de un circulo calcula el área de este.

- La función *freeDraw* permite al usuario dibujar de forma libre, es decir, mientras tenga el botón izquierdo del ratón y lo mueva, se dibujará el recorrido de este. Removemos cualquier interacción anterior con el mapa y la clase del mapa vuelve a ser con el cursor normal. Si su checkbox no está marcado dejamos el cursor por defecto. Añadimos la interacción al mapa de una línea recta, especificando que el parámetro *freehand* sea verdadero, esto permite el dibujo libre. Comprobamos los botones y preguntamos si la función es llamada desde un dispositivo móvil. Si es así, hay que desplegar el menú de color y grosor del pincel. Si está escondido, se despliega, y viceversa. Por último, esconder el menú si el usuario desmarca el botón.

- *deleteFeature* borra un dibujo especifico del usuario en el mapa. Se obtienen todas las propiedades de todos los dibujos (todos almacenados en la capa *source*). Si existen dibujos, se busca que el identificador sea el mismo que el elegido. En caso de que se encuentre borra todas las propiedades relacionadas con esa Id. Escondemos el mensaje con la medida, utilizando la misma identificación que se nos dio. Y terminamos el bucle con un *break*, ya que se ha encontrado el dibujo a borrar y no hace falta buscar más.

mouseEvents.js

En *mouseEvents.js* encontramos las funciones que se encargan de manejar los eventos que el usuario hace con el ratón.

clickEvents recogerá todos los eventos de clic que se hagan con el ratón sobre el mapa. Primero se guardan todos los posibles checkbox que puedan intervenir en este tipo de evento, y también las coordenadas. El checkbox que esté marcado será las instrucciones que se han de hacer. *toggleButtons* se asegura que no haya más de un checkbox marcado a la vez.

- *pto*: El usuario demanda las coordenadas de un punto concreto. Transformamos las coordenadas en la proyección adecuada y las pasamos a horas, minutos y segundo. Creamos el mensaje y dentro insertamos su Id, las coordenadas, y la función para borrar todo el conjunto y lo introducimos en su lugar. Como *measureTooltipElement* es una variable global

hay que limpiarla por seguridad, terminamos de desvincular el mensaje y el observador del evento.

- *comment:* Se crea el formulario que el usuario rellenará para insertar un comentario en el punto donde es usuario ha hecho clic. Teniendo en cuenta el Id del mensaje y figura para que puedan ser borrados. Las coordenadas e información a insertar en la base de datos. Se añade un botón de cancelar que borrará todo el conjunto sin insertar la información en la base de datos. Posicionamos el formulario y la figura, y hacemos lo mismo que en el anterior apartado para finalizar.
- *zin:* Centrará la vista en las coordenadas del clic y recopilamos el zoom actual que hay en el mapa y le añadimos uno para acercar la vista. El zoom máximo por defecto en Openlayers es de 28.
- *zou:* Tiene la misma función que *zin,* excepto que, en lugar de acercar alejamos en uno. Zoom mínimo puede ser menor que cero, Openlayers repite la imagen del mapa.
- *feat:* Mostrará un modal de la información en las coordenadas donde se ha hecho clic, de la capa o capas seleccionadas. Guardamos la resolución del mapa y definimos unas variables. *url* será la fuente de la información a mostrar. *code* es donde introducimos código HTML que se mostrará en el modal. *flag* es una bandera indicándonos si hay alguna capa seleccionada.

Las dos primeras peguntas son para comprobar si la información requerida es de las capas por defecto. Como hay capas seleccionadas *flag* es verdadero, obtenemos el enlace de los datos a mostrar, ubicándolos en un iframe, que su vez insertaremos en el modal. Recorremos el vector de las capas personalizadas, comprobando cuales de ellas están visibles en el mapa para enseñar la información.

A partir de aquí es igual que con las capas por defecto salvo que la proyección se encuentra en el vector *info.* Se comprueba la bandera en caso de que no hay capas seleccionadas, en tal caso se muestra un mensaje al usuario. Finalmente enseñamos el modal con todos los datos de las capas que se han marcado. Ahora que hemos definido la función hay que activarla en el mapa. *clickEvents* se accionará cuando el usuario haga un solo clic en el mapa.

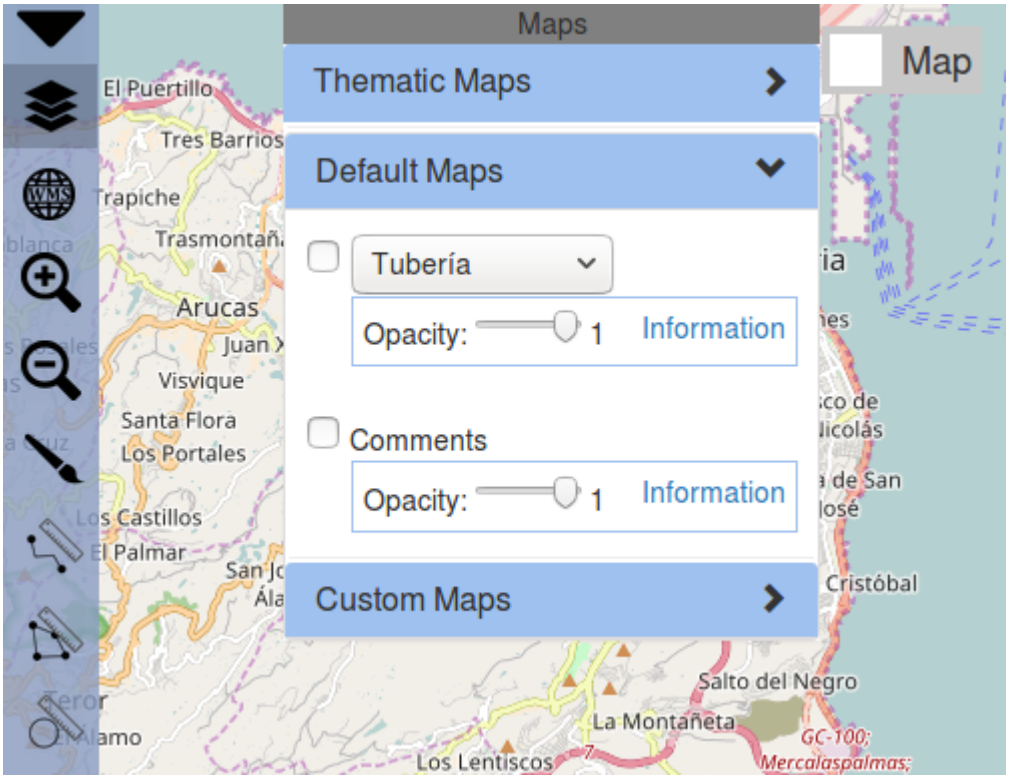
view.js

El archivo *views.js* es donde están agrupadas las funciones que manejan la mayoría de la estética de la aplicación web.

- En la función t*oggleMenu* se utiliza en la versión móvil del servicio, esta despliega o esconde el menú de la derecha con los botones. Tenemos dos elementos un botón escondido y otro con el menú desplegado. Se pregunta si el menú está desplegado, si se ha llamado a la función significa que el usuario quiere esconderlo, por lo que le añadimos el estilo 'none' para que el menú no se vea en pantalla. Y al botón que está solo, se le añade 'inline-block' para que se muestre. En caso contrario, para desplegar el menú, se intercambian los estilos.

- *displayMenu* enseña o esconde algunos menús en la versión de móvil, tendrá el mismo modo de funcionamiento que la anterior función con respecto a los estilos. En primer lugar, por si antes de llamar a esta función se estaba haciendo otra actividad, se sitúa el cursor por defecto y se desvincula la integración de dibujar. Si el bloque del menú está escondido lo

desplegamos, sino lo escondemos. En caso de que se haya apretado el botón hay que reiniciar los otros botones del menú, sino escondemos el menú seleccionado.



Despliegue del menú de mapas en la aplicación móvil

- *changeSymbol* cambia el símbolo de los grupos de mapas en el menú. Se trocea el identificador del grupo para el numero después del guion bajo. Ya que es el mismo número para identificar el símbolo que se muestra. Si se llama a la función y es un '-', significa que está desplegado y hay que plegarlo, cambiándolo por un '+'. Si no, entonces es el caso contrario e intercambiamos símbolos.

- Para la funcionalidad de mover por el mapa se usa *mHand*. Se ha creado un estilo para el mapa el cual transforma el cursor en una mano si pasa por encima de este, y cuando se hace clic se transforma en un puño. Dando la sensación al usuario que agarra el mapa para moverlo. Por último, asegurarse de que todos los botones estan correctos.

- La función *toggleButtons* ayuda a finalizar otras funciones. En un principio había instrucciones que se repetían en muchas de las funciones, se decidió que juntando esas instrucciones el código estaría mejor optimizado. Se divide en cinco partes:

    - Recogemos todos los checkbox con el nombre *myCheckbox* y los recorremos poniéndolos todos a falso. Para terminar, solo dejamos marcado el que sea igual a la Id que se pasa a la función.
    - Para las etiquetas (*mylabel*) es igual, colocamos todos a falso excepto el que tenga la marca. Esto se hace para que el botón tenga el estilo de que esta pulsado.

- Para los bloques de menú (*mydiv*) del móvil tiene la misma estructura que los anteriores. Solo si se le pasa un identificador del bloque se enseña el seleccionado.
- Si queremos activar otra funcionalidad después de *zoomDrag* tenemos que desactivar la interacción con el mapa y comprobar que está desmarcado.

De cara a la interfaz el usuario, este solo verá que puede activar una funcionalidad a la vez, si pulsa en otra, las demás se desactivan y si pulsa en la misma se desactiva esa.

```
function toggle_buttons (checkboxId, labelId, divId){
  //For Checkbox
  var myCheckbox = document.getElementsByName("myCheckbox");
  Array.prototype.forEach.call(myCheckbox,function(mc){
    mc.checked = false;
  });
  checkboxId.checked = true;
  //For Label
  var mylabel = document.getElementsByName("mylabel");
  Array.prototype.forEach.call(mylabel,function(ml){
    ml.classList.remove('active');
  });
  //For Menus
  var mydiv = document.getElementsByName("mydiv");
  Array.prototype.forEach.call(mydiv,function(md){
    md.style.display = "none";
  });
  if (divId) {
    //Only display this div
    var mobileMenu = document.getElementById(divId);
    mobileMenu.style.display = "inline-block";
  }
  //For Zoom-Drag
  if(document.getElementById(labelId)){
    document.getElementById(labelId).classList.add('active');
    if ((boxInteraction)&&(checkboxId.id!='zd')){
      //Deactivate the interaction
      boxInteraction.setActive(false);
    }
  }
}
```

Función *toggle_buttons*

- *setCursor* cambia la imagen del cursor según la funcionalidad elegida. Si el checkbox está desmarcado quiere decir que se desactiva la funcionalidad, por lo que dejamos el cursor por defecto cambiando de clase del mapa.
  - *zi:* se coloca un más dentro de una lupa, expresando que se hará un aumento del zoom cuando se haga clic.
  - *zo:* un menos dentro de una lupa, para expresar la disminución del zoom.
  - *fid:* lo dejamos por defecto para buscar la información en las coordenadas especificas del clic.

Finalmente comprobamos que los botones y funciones están como deben.

- La función *setCoord* se creó debido a que el ultimo botón del menú es un dropmenu y no funciona como los demás. Se sitúa el cursor por defecto y removemos la interacción del dibujo en el mapa. Si la el dropmenu está abierto, significa que la etiqueta tiene que estar activa, sino llamamos a la función *toggleButtons* para que se asegure de que los botones están como tienen que estar.

- *pickColo*r cambia el estilo, en concreto el color de la capa donde se guardan los dibujos. Se crea una nueva variable de estilo Openlayers. Para que se guardara el grosor hay que almacenar el dato en una variable global, entonces se especifica que el color ahora es el pasado por parámetro con el grosor guardado. Aplicamos el estilo a la capa *vector* y ahora el nuevo color es el base, ya que más adelante puede que haya que recordarlo.

- *pickWeight* sigue el mismo procedimiento que la función anterior, en vez de cambiar el color, se modifica el grosor del pincel. En la última línea se actualiza el número que se ha seleccionado mostrándolo al usuario.

- La función *changeBase* intercambia la capa base del mapa entre una vista por satélite o la geográfica. Si se activa la vista por satélite se cambia el nombre de la etiqueta y la capa se hace visble. Sino, se activa la vista geográfica, cambiamos la etiqueta y ocultamos el satélite.

globals.js

globals.js define variables globales, el mapa por defecto de la aplicación, y algunas funciones. Estas variables para dibujar son globales porque se usan en distintas funciones. Las que se usan para borrar figuras del mapa lo son, debido a que siempre hay que tener en cuenta el número de identificación de cada una de ellas.

baseWidth y *baseColor:* Se usan para recordar el grosor del pincel y el color, ya que cuando se cambia cualquiera de las dos la otra se sitúa por defecto, cuando es posible que el usuario lo haya modificado.

Variable *source*: es la fuente donde se guardan las propiedades de las figuras dibujadas.
Vector: es la capa donde se representa los dibujos con la fuente en *source* y un estilo. El estilo que definimos aquí es cuando se termina finalmente la figura, situando el borde amarillo.
setZIndex define en que en posición del índice Z se encuentra la capa, esta tiene que estar por encima de todas ya que es lógico pensar que si se quiere dibujar será sobre las otras capas.

De la variable *sketch* a *tooltipid* son variables para dibujar y borrar en el mapa. Estas tienen que estar accesibles para varias funciones y en cualquier momento.

Pipeline y *comments* definen las capas que se han puesto por defecto, detallando la fuente como un enlace al Geoserver que creamos antes y la capa a mostrar. Al definir el mapa tenemos las capas de fondo un mapamundi, de dibujo, tubería y comments. Como la aplicación está especializada en visualizar red de agua de Las Palmas de Gran Canaria, centramos la vista en esa

localidad con el suficiente zoom para que se vea toda la red de tuberías. En un principio *pipeline* y *comments* las escondemos, si el usuario quiere verlas solo tiene que marcarlas.

También se define la capa por satélite (*bingAerialLayer*), debido a que se usa una API de BingMaps es un poco diferente. En *preload* asignamos *infinity* para que el mapa se repita en caso que el zoom sea menor que 1. Para la fuente se introduce la clave de la aplicación, y el nombre de la capa deseada. Por último, bajamos un nivel en el índice Z para que las otras capas añadidas tengan prioridad, y este oculto por defecto.

Ahora creamos los mapas temáticos, que son: *hipsometrico, clinometro, sombras* y *temp*. Estos actúan de capa base igual que el satélite, pero las vistas solo están disponibles en las Islas Canarias. La diferencia de esta definición con respecto a las anteriores es que, el servicio se obtiene de Grafcan, y hay que especificar el formato de salida y la proyección en el mapa.

Definimos *map*:

- *layers*: tendremos la capa principal *ol.source.OSM* que es el mapa del mundo, y un grupo de capas, por defecto definidas con anterioridad.
- *target*: el mapa se despliega en el bloque de HTML que se identifique como '*map*'.
- view: Cuando se cargue el mapa, se sitúa en esas coordenadas (Las Palmas de Gran Canaria), con un zoom de doce.

Para cargar el servicio WMS son obligatorias algunas variables globales, debido a que es necesario recordar cuantas capas se han cargado, la última posición y el número de grupos de mapas. También son necesarias guardar la información de las capas para cuando sean necesarias mostrarla.

```
var map = new ol.Map({
 layers: [
   new ol.layer.Tile({
     source: new ol.source.OSM(),
     zIndex: '-2'
   }), vector, pipeline, comments, bingAerialLayer, hipsometrico,
       clinometrico, sombras, temp
 ],
 target: 'map',
 view: new ol.View({
   center: [-1722584.517212906, 3257823.5051733414],
   zoom: 12
 })
});
```

Declaración del mapa principal

Para que no quede ninguna información de otras peticiones de las coordenadas especificas en el modal hay que borrarlas cada vez que se oculte. Como ya se ha comentado antes la propiedad de un dropmenu es que se oculta cuando se hace clic dentro o fuera de este. Ya que se ha puesto pestañas dentro, tenemos que evitar la interacción con ellas cierre el dropmenu. Aquí definimos la función *isEmpty*, a esta se le pasa una cadena de caracteres y esta comprobará que no esté vacía o llena de espacios.

loadScript carga los scripts que definen las nuevas proyecciones. La función añade a la cabecera de nuestra aplicación web, la declaración de la proyección que es necesaria para la capa personalizada del usuario. Crea un elemento tipo '*text/javascript*' y cuando esté listo y caragado, se inserta en la cabecera. Hay que tener en cuenta si es para Internet Explorer o para los otros exploradores.

Window.onload re-escala el mapa para que sea de las dimensiones de la pantalla del usuario, pero se realiza solo al cargar la aplicación móvil. Esto se debe a que algunas de los estilos distorsionan la visión del mapa. Como es posible que al refrescar la aplicación quede en cache algún botón marcado, se ponen todos a falso.

Cuando se escribe un enlace en el modal para añadir un servicio WMS, si el usuario pulsa el botón 13 que en el teclado es el Enter, llamamos a la función *checkLayers*. Se añade la facilidad al usuario de hacer clic en el botón correspondiente en el modal o simplemente apretar la tecla intro.

La lista con los resultados de las calles se ocultará si el usuario hace un clic o toca (dispositivo táctil) fuera del bloque HTML asignada a esta lista.

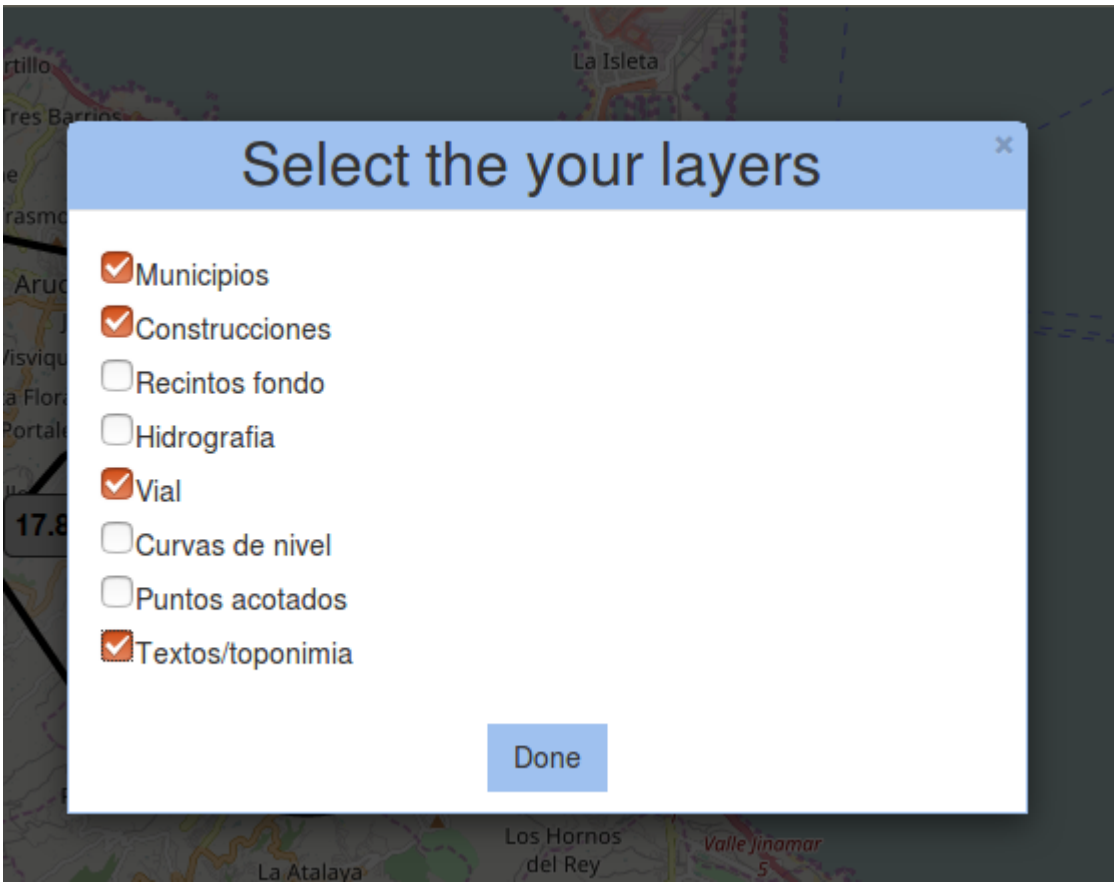
loadWmsService.js

loadWmsService.js tiene el cometido de cargar el servicio WMS (Web Map Service) que el usuario ha introducido mediante una URL, en el mapa de nuestra aplicación.

La función *checkLayers* en primer lugar comprueba que el enlace es correcto, y en segundo lugar muestra todas las capas del servicio, para que el usuario elija cuales se han de cargar en la aplicación web.

Escondemos el modal donde se introduce la URL, creamos la variable de petición XMLHTTP, y almacenamos el enlace. Nos aseguramos que la dirección no tiene ningún defecto con la función *isEmpty*. Ahora debemos comprobar que la sintaxis de la petición se hace correctamente. Si no existe la palabra 'request', significa que el usuario solo hay introducido la URL sin petición. Se añade a la cadena la solicitud dependiendo si ya tiene el carácter '?' o no.

Esperamos a que la solicitud esté lista y sea correcta. Openlayers tiene la función *ol.format.WMSCapabilities* para que el resultado de la petición tenga un formato fácilmente manejable. *nlayers* calcula el número capas que tiene el servicio, borramos el modal por si quedara algo de otra llamada. Luego, presentamos un modal con una lista creada de las capas y un checkbox asignado a cada una. Esto se hace para que el usuario seleccione las que desea cargar. En caso de que de un error haciendo la solicitud, mostrara una alerta con el mensaje de error adecuado.



Selección de capas al cargar un servicio WMS

layerLoader es la segunda parte para cargar el servicio WMS, a partir de las capas seleccionadas por el usuario en la función anterior. Primero escondemos el modal con las capas seleccionadas y se captura el enlace del servicio, luego obtenemos todos los checkbox con el nombre 'layer_check'. Creamos un vector que almacena el nombre de las capas que se quieren cargar, llamado *layerSelected*, y nos aseguramos que las capas por defecto no estén en el *customLayers*, que es el vector final de las capas personalizadas.

Se guarda la vista del mapa para que al cargar el nuevo, el usuario no note diferencia. Borramos el bloque con el mapa por defecto, y creamos otro para más adelante insertar las capas. Se crean unas variables para el control de los bucles, esto es importante, ya que controla las iteraciones de esta inserción de capas y las pasadas si hubiera.

- *lnames*: es un vector con todos los nombres de las capas que se encuentren seleccionadas.
- *nlayers*: almacena el número de capas totales en el servicio WMS.
- *i*: se utiliza para recorrer todas las capas del servicio.
- *x*: se recorre el vector de las capas seleccionadas de la anterior función.
- *projec*: se guarda la proyección de la capa actual en el vector. De tipo *projection.*
- *projections*: es el vector donde guardamos las proyecciones que se han definido ya en la cabecera de la aplicación.
- *layerCrs:* se guarda la proyección de la capa actual en el vector. De tipo literal.
- *lastLayer:* es una variable global que recuerda la posición del vector *customLayers*, así si se añaden más capas estarán ordenadas.

Ahora se crea un grupo de capas nuevo con el título del servicio, donde se insertará las capas que el usuario ha seleccionado, el grupo de puede plegar o desplegar con un clic. Recorremos todas las capas de servicio, si coincide con las seleccionadas, se almacenan el formato de la imagen y la proyección, comprobando que exista. Si no se despliega un mensaje de error. Se guarda toda la información de la capa para mostrarla en el futuro.

Después de mucho ensayo y error, se llegó a la conclusión de que definir la proyección adecuada en la capa es esencial para que se muestre. Si la capa que intentamos cargar no está en 'EPSG:4326', que es la más común, entonces hay que definirla y almacenarla para especificarla más adelante. Pero si es 'EPSG:32628' solo hay que almacenarla, ya que está definida en un principio.

Para definiría comprobamos que no esté ya buscando en el vector *projections*, en caso negativo creamos un Script cuya fuente es una web que contiene todas las proyecciones definidas. Añadiéndola en la cabecera de nuestra aplicación, estaría definida. Definimos las propiedades de la capa para añadirla al mapa:

- *url*: es el enlace del servicio, sin la sintaxis de la petición.
- *params*: establecemos los parámetros importantes para la capa, el nombre por el que se define y su formato.
- *projection*: la proyección en la que se encuentra.

Se crea el bloque de HTML donde están las funciones de visibilidad, transparencia e información con sus identificadores. Todo tiene que estar ordenado, es decir, la posición en el vector es la misma que su Id, para que en el futuro si se llaman a estas funciones puedan reconocer la capa correcta. Si se ha insertado una capa, incrementamos las posiciones de los vectores 'lastLayer' y 'x'. También se suma en uno la variable 'i' en cualquier caso. Para pasar a la siguiente llamada, si existe, el identificador del grupo de capas se añade uno.
Con el motivo de que estén siempre localizables insertamos las capas por defecto al final del vector. Definimos una nueva variable mapa con las propiedades recogidas antes:

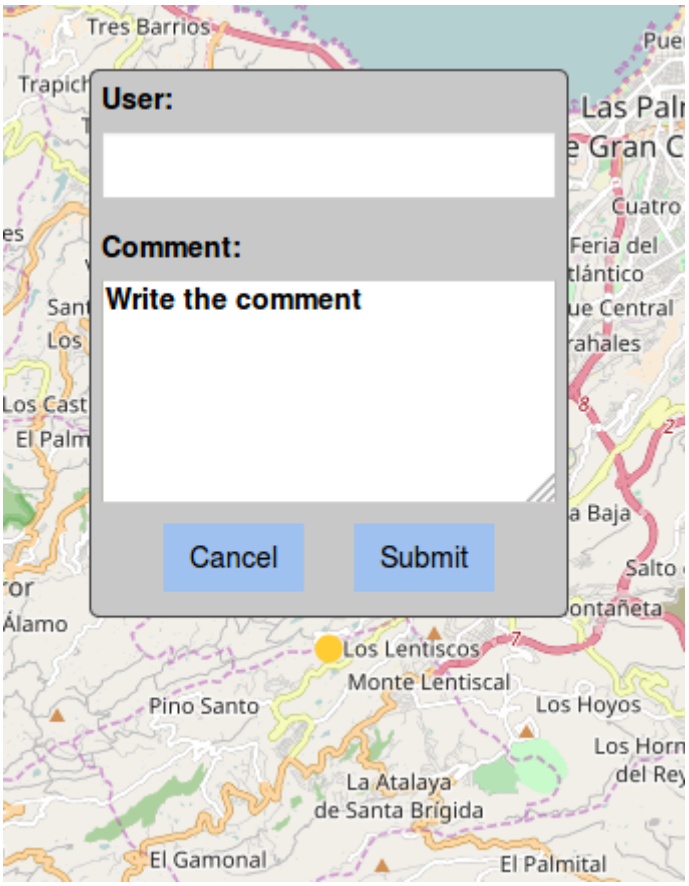
- *layers*: tendremos la capa principal *ol.source.OSM* que es el mapa del mundo, y un grupo de capas, el vector *customLayers* con todas las que se quieran mostrar.
- *target*: el mapa se despliega en el bloque de HTML que se identifique como '*map*'.
- view: Cuando se cargue el mapa, se sitúa en la vista que almacenamos al principio de la función.

Redimensionamos el alto y el ancho del mapa para que se ajuste a la pantalla del usuario, solo si es para un dispositivo de pantalla pequeña. Se activa en el mapa el evento de solo un clic. Como ahora hay capas en el menú de '*customLayers*', se despliega para que el usuario pueda ver que hay cambios. Hay que borrar el vector de las capas seleccionadas, en caso de que la siguiente llamada no se vuelvan a insertar. Por ultimo comprobamos que los botones del menú estén correctamente.

insertComment.php

Archivo *insertComment.php* insertará los datos de un nuevo comentario que el usuario ha introducido en el formulario. Primero conectamos con la base de datos, se hay algún error se le informa al usuario. Este es posible que se le olvide o no esté interesado en rellenar el campo de *user*,

por lo que, si no está vacío, se insertan todos los datos. Pero si el parámetro no se encuentra, por defecto en la base de datos se rellena como *Anonymous*.

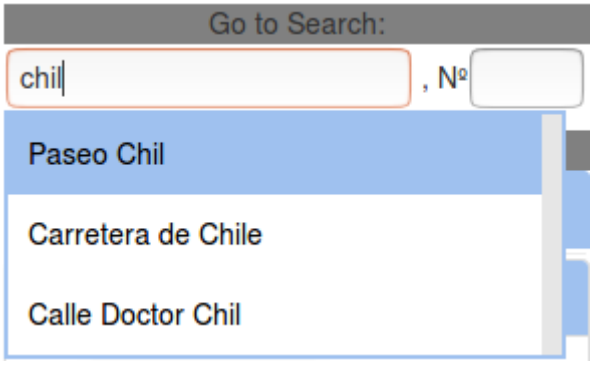


Formulario para insertar un comentario en el mapa

street.php

street.php realiza la petición a la tabla para buscar la calle introducida por el usuario. Se crea la ristra de con la información, y se intenta conectar con la base de datos con *pg_connect*, en el caso de que falle el, se le comunicará al usuario mediante un mensaje en la pantalla. Almacenamos los datos a buscar, y creamos una serie de variables que nos ayudará para desglosar las coordenadas facilitadas en el resultado de la búsqueda. Se crea el bloque donde irán los resultados. Ahora dividiremos las clases de búsquedas en dos:

- Si se pregunta por la calle y el portal, buscaremos en la tabla *portales*, ya que posee los dos datos que necesitamos. Para la calle buscaremos la palabra parecida o que este contenida con insensibilidad de mayúsculas, y para el portal el número exacto. También es insensible a los acentos y otros caracteres como la diéresis, gracias a la extensión *unaccent* que creamos con anterioridad. Si no hay coincidencias muestra el mensaje sin resultados. Por cada fila extraemos las coordenadas, con esto hacemos código HTML creando un enlace con la función *centerStreet*, el nombre de la calle y número de portal.
- Si solo se requiere la calle, se busca en la tabla vial, ya que tiene el tipo de vía y podemos añadir más sugerencias a la búsqueda. El procedimiento es el mismo que el anterior, solo que no investigamos por el número del portal. Para las coordenadas obtenemos la latitud y longitud del principio de la calle. Y formamos el enlace para centrar la vista en la vía deseada.



Búsqueda por calle

map.html

El archivo *map.ht*ml es donde el usuario empieza a interactuar con la aplicación. En la cabecera definimos el tipo de contenido y la escala del dispositivo, dependiendo de la resolución de la pantalla se transporta a la web de móvil o no. También determinamos los estilos y scripts que nos ayudaran a dar forma a la web.

La web se divide en dos, en primer lugar, un bloque con las funciones a la izquierda. Compuesto de un menú con seis botones, los últimos tres son dropmenus para mostrar más funciones. Después se halla el selector de color y grosor del pincel. La búsqueda por calle está compuesta por un formulario de calle y portal, y los resultados de esta. El siguiente bloque Maps está formado por tres subsecciones:

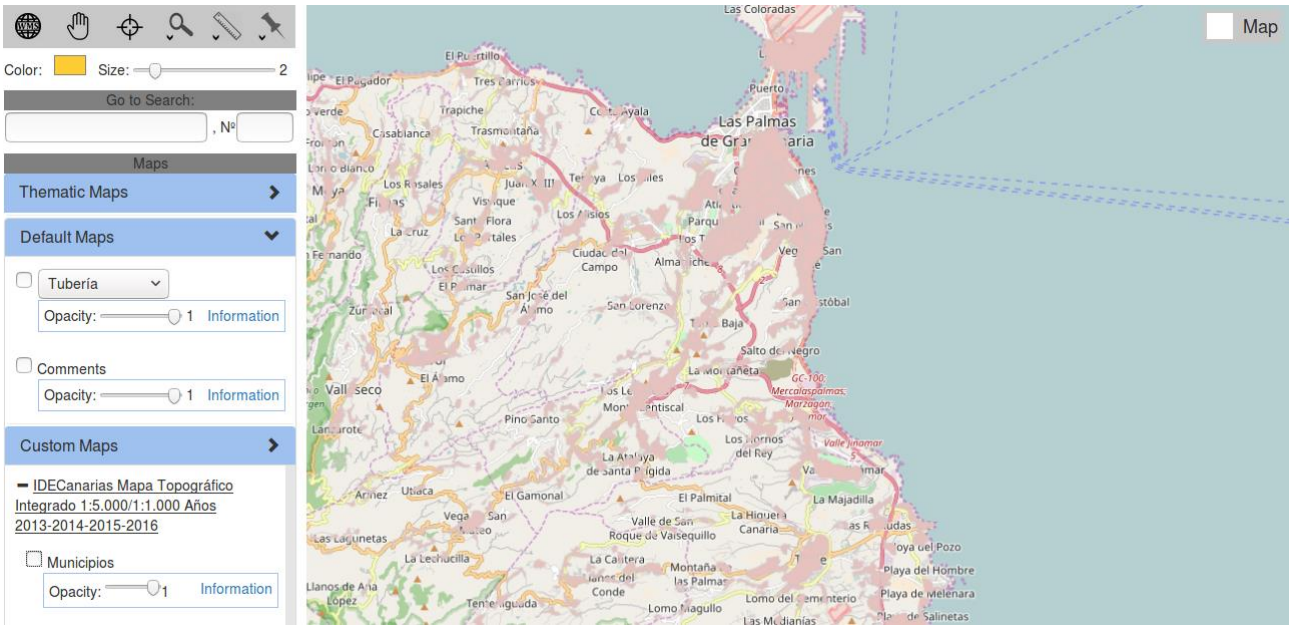
- *Thematic maps,* donde el usuario puede indicar el mapa temático a partir de un selector.
- *Default maps,* aquí se encuentra las opciones de los mapas por defecto (tubería, comments), con sus estilos. Cada capa tiene un checkbox de visibilidad, una entrada de rango que especifica la trasparencia, y un enlace que despliega un modal con información de la capa.
- *Custom maps,* inicialmente está vacío, si el usuario añade un servicio WMS, las opciones de las capas se añaden aquí. Estas propiedades son iguales que la de los mapas por defecto.

En la segunda parte arriba a la derecha, se sitúa un interruptor para intercambiar entre la vista por satélite o el mapa general. Y el mapa que ocupa toda la pantalla del dispositivo.

A parte, se puntualizan los modales que permanecerán ocultos hasta que se les llame.

- *WMSurl*: Aquí el usuario inserta el enlace del servicio WMS
- *infoModal*: Muestra la información general de la capa seleccionada
- *PointInfo*: Expone los datos de las capas seleccionadas en las coordenadas donde el usuario ha hecho clic.
- *LayerSelection*: Muestra todas las capas que posee el WMS para elegir las que se mostraran en la aplicación.

Por último, declaramos los scripts propios al final porque el navegador carga el HTML en el orden en que aparece en el archivo. Si se cargara primero el Javascript y este afecta al HTML que tiene debajo, podría no funcionar, ya que ha sido cargado antes que el HTML.



Interfaz final de la aplicación

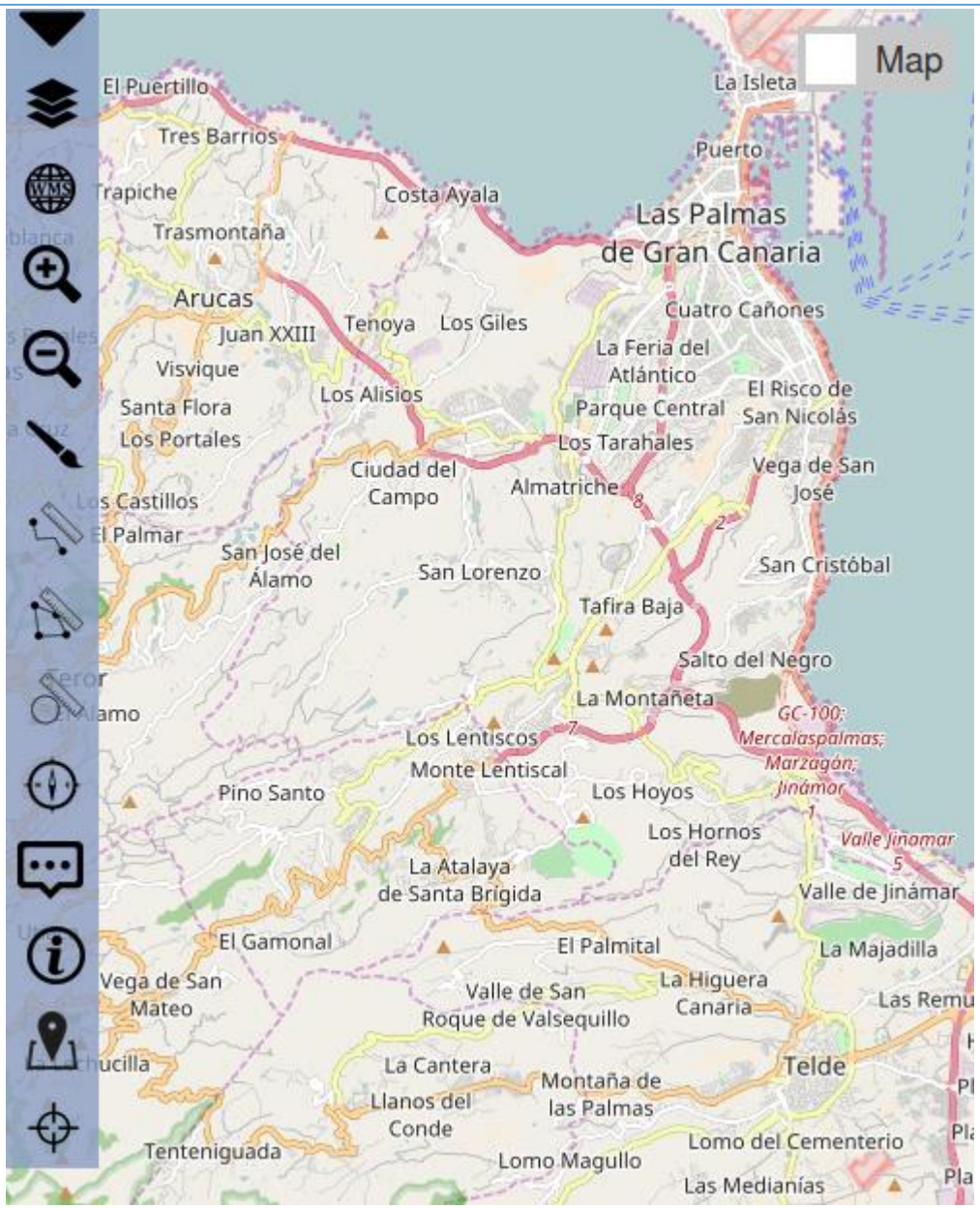
mobile.html

Cuando la pantalla del dispositivo tiene un ancho menor de 700 pixeles se cargar la web *mobile.html*. En la cabecera definimos el tupo de contenido y la escala del dispositivo, dependiendo de la resolución de la pantalla se transporta a la web de escritorio o no. También determinamos los estilos y scripts que nos ayudaran a dar forma a la web.

A la derecha hay un menú con las funciones de la aplicación, algunos de estos botones, aparte de realizar su función correspondiente despliegan bloques de contenido ya que poseen opciones adicionales. Por defecto están ocultos, y cuando se llama a la función correspondiente se enseñan en la parte superior en el centro.

- *searchMobile:* muestra el formulario para buscar la calle y portal, con los resultados de esta
- *coordMobile:* expone dos pestañas con otro formulario donde se introducen las coordenadas a que se quieran visitar.
- *drawMobile:* al dibujar enseña el selector con los colores y el tamaño del pincel.
- *mapsMobile:* aquí se muestra las capas que hay en el mapa, organizado en *Thematic maps, Default maps* y *Custom maps.* Tal y como explique en *map.html*

Esta última parte también es igual que su versión para escritorio, se crea el interruptor para el cambio de vista por satélite, los modales, y las declaraciones de los scripts propios.



Interfaz final de la aplicación móvil

10. Conclusiones y líneas futuras:

10.1 Conclusiones

El desarrollo de una aplicación web ligera y flexible que permita la visualización de redes agua en GIS se adaptó positivamente para la mayoría de los dispositivos modernos. Debido a su modularidad la herramienta basada por completo en Software libre, es fácilmente restaurable y sencilla en actualizar, tanto el Front-End como el Back-End

Al realizarse este proyecto se extrae la conclusión de que profundizar en conocimientos de GIS (Geographic Information System) que, a priori, puede que no tengamos es importante. Y también tener un mayor grado de entendimientos sobre las comunicaciones web y protocolos, a fin de conseguir la mayor integración posible entre las herramientas usadas.

Otra parte muy importante, es el diseño de la interfaz de usuario, se ha conseguido de manera eficaz. El usuario pueda desenvolverse sobre la aplicación de una forma intuitiva y con un control efectivo de la misma.

Debido a que la aplicación web no se pudo probar en todos los navegadores que existen, y en todos los distintos dispositivos del mercado, no es posible afirmar que el desempeño y el aprovechamiento del visor web sea total.

10.2 Líneas futuras

Existen algunas recomendaciones y mejoras que volverían la aplicación más robusta y eficiente en base a los resultados y conclusiones a que se llegó al finalizar el proyecto.

- GeoServer es una herramienta poderosa a la que se pudo haber sacado un mejor rendimiento. Optimizando la cache de los espacios de trabajo es posible disminuir los tiempos de respuesta, obteniendo así la visualización de los mapas y capas más rápido.

- Implementar unas optimizaciones en el servidor Apache. Comprimir las imágenes de los botones y minificar el código JavaScript y HTML reducirían los tiempos de carga.

- En la búsqueda de calles, se podría añadir la opción de tipo de calle, diferenciando así entre un paseo, avenida, calle o carretera. Permitiendo al usuario un uso más cómodo y eficiente de la aplicación.

- A través de Geolocalización, OpenLayers permite indicar la ubicación del usuario en el mapa. De este modo si el usuario está en la calle deseada, no tiene que buscarla, la aplicación le marcaría el lugar.

- Ya que el mapa a efectos es un lienzo (canvas), se pueden implementar muchas de las funciones de este. Por ejemplo, añadir un texto rápido, un gradiente, imágenes, videos, animaciones, entre otras.

- Implementar de una sesión de usuario. Al iniciar sesión el usuario aumentamos la seguridad sabiendo que usuario hizo que, y permitiendo la personalización. Por ejemplo, al introducir un comentario en un punto de interés, no haría falta rellenar el campo *user.*

11. Bibliografía:

Documentación oficial de OpenLayers
http://docs.openlayers.org/

OpenLayers CookBook
http://www.acuriousanimal.com/Openlayers-Cookbook/

OpenLayers Beginner's Guide
https://openlayersbook.github.io/

Documentación PostgreSQL
http://www.postgresql.org/docs/9.0/interactive/index.html

Manual de PostGIS
http://postgis.net/docs/manual-2.0/

Manual de usuario de Geoserver
http://docs.geoserver.org/2.0.2/user/

W3school
http://www.w3schools.com/

Stackoverflow
http://stackoverflow.com/

CSS cheat sheet
http://overapi.com/css/

jQuery API Documentation
http://api.jquery.com/

Manual de PHP
http://www.php.net/manual/es/

IDECanarias
http://www.idecan.grafcan.es/idecan/

Base de datos espacial
https://es.wikipedia.org/wiki/Base_de_datos_espacial

Sistema de información geográfica (SIG)
https://es.wikipedia.org/wiki/Sistema_de_informaci%C3%B3n_geogr%C3%A1fica

Proyecciones cartográficas

https://es.wikipedia.org/wiki/Proyecci%C3%B3n_cartogr%C3%A1fica

Servicio WMS

https://es.wikipedia.org/wiki/Web_Map_Service

Política del mismo origen (CORS)

https://es.wikipedia.org/wiki/Pol%C3%ADtica_del_mismo_origen

Filtros para el control de acceso

https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS

Peticiones XMLHttpRequest

https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Synchronous_and_Asynchronous_Requests

VirtualBox

https://www.virtualbox.org/manual/UserManual.html

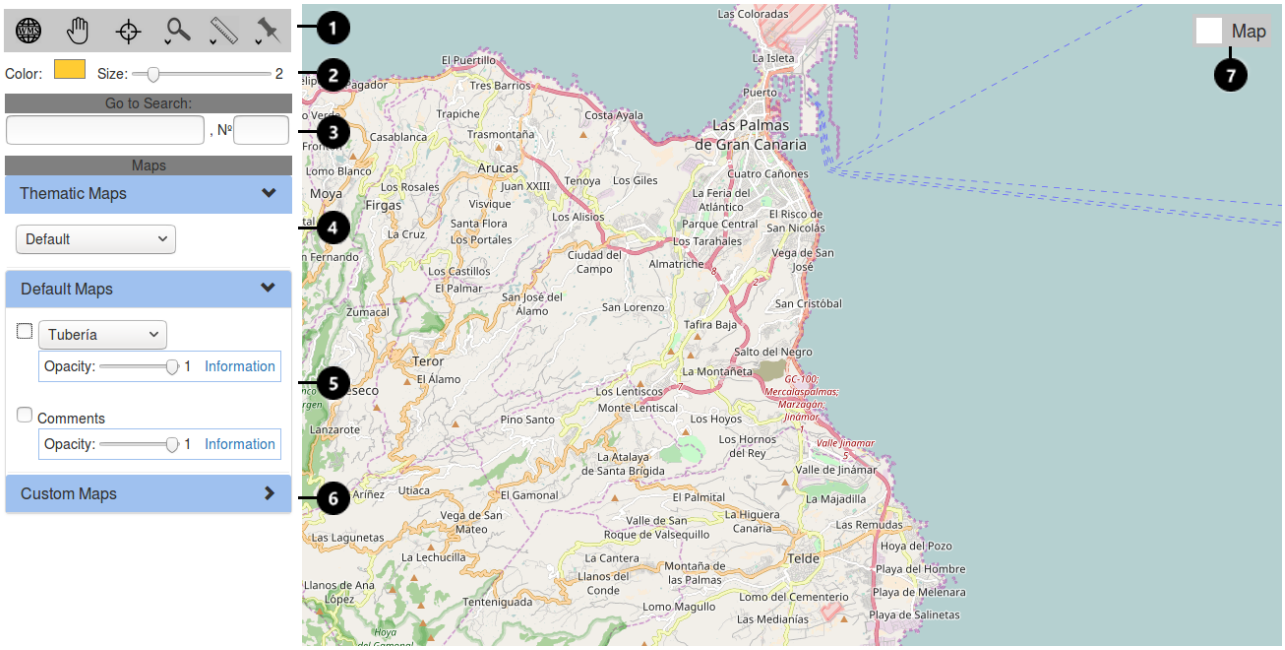
IDECanarias

http://www.idecan.grafcan.es/idecan/

12. Manual de usuario:

En este apartado se explicará el uso de las distintas funciones de la aplicación. Aunque es bastante intuitiva es recomendable leer este manual para su uso correcto y eficiente.

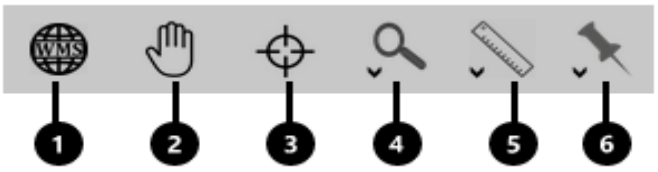
12.1 Interfaz de escritorio



La interfaz se compone de siete secciones diferente, la mayoría están en un menú a la izquierda, excepto la última que está en la esquina superior derecha.

1. Menú con distintas funciones.
2. Color y tamaño del pincel y figuras
3. Buscador por calle y portal
4. Selector de mapas temáticos
5. Mapas por defecto
6. Mapas personalizados
7. Interruptor cambio de mapa base

1. Menú con funciones variadas



Este menú activa las distintas funciones que de un modo u otro interactúan con el mapa.

1.1 Cargar servicio WMS

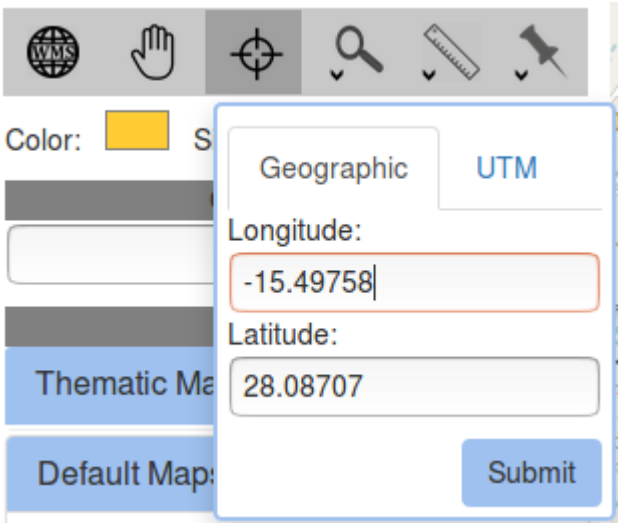
Cuando se activa, despliega un modal para que los usuarios introduzcan el enlace del servicio WMS. Y a continuación, se seleccionan las capas que se desean mostrar en el mapa.

1.2 Desplazamiento del mapa

Ahora el cursor del ratón es una mano, cuando se hace clic se cambia a un puño. Dando la sensación los usuarios de que agarran el mapa y lo arrastran para moverlo.

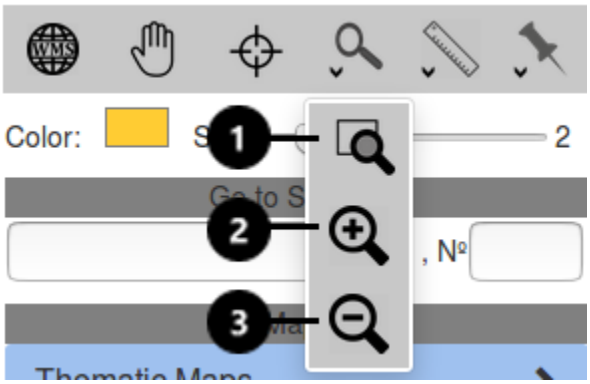
1.3 Ir a las coordenadas

Al hacer clic aparecerá un menú con dos pestañas (Coordenadas geográficas y UTM), cuando se introducen las coordenadas la vista del mapa se centra en estas.



1.4 Diferentes tipos de zoom

Es un desplegable con tres botones para los distintos tipos de zoom.



1.4.1 Zoom arrastrar

Manteniendo pulsado el botón izquierdo del ratón y arrastrando sobre el mapa, aparecerá un recuadro azul. Este señala hasta donde se tiene que hacer zoom para que las dimensiones del mapa se queden de la forma más parecida posible.

1.4.2 Aumentar zoom

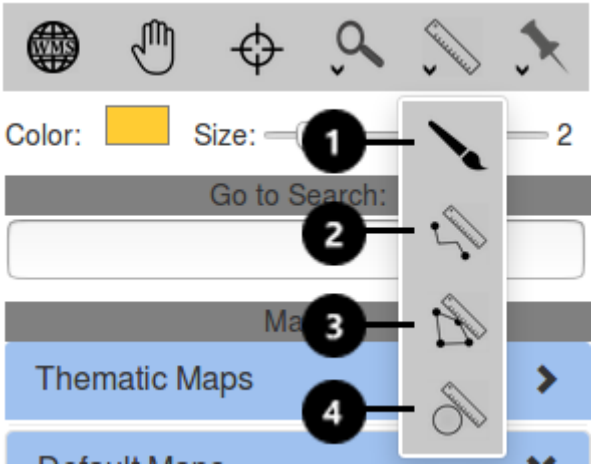
Pulsado el botón izquierdo del ratón sobre el mapa, aumentará el enfoque, centrando el mapa donde el usuario haya hecho clic.

1.4.3 Alejar zoom

Es la misma función que el apartado anterior, excepto que este disminuye el enfoque.

1.5 Dibujos en el mapa

El botón agrupa las funciones que añaden figuras y mediciones al mapa. Estas medidas aparecen por defecto en metros, pero se puede cambiar a kilómetros en un selector.



1.5.1 Dibujo Libre

Se puede añadir trazos al mapa de manera libre y sin restricciones. Esta función no mide la distancia o área alguna.

1.5.2 Medir distancias

El usuario dibuja trazos de líneas rectas que se van midiendo cuando el cursor se mueve. Si se hace clic izquierdo se añade un punto en la ruta. Para terminar de medir se ha de hacer doble clic izquierdo con el ratón.

1.5.3 Medir áreas, polígonos

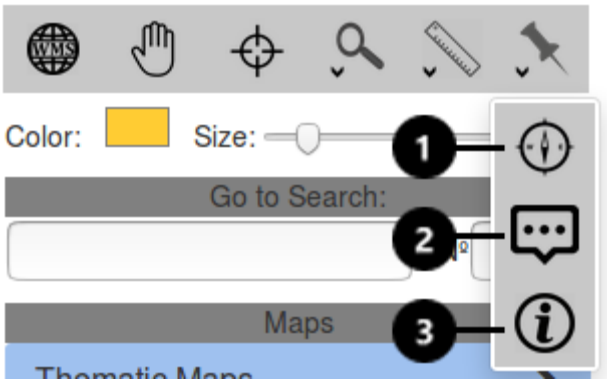
Añadiendo como en el apartado anterior más de dos puntos se despliega una medida del área, es posible añadir tantos puntos en la ruta de la figura como se prefiera.

1.5.4 Medir áreas, circunferencia

Haciendo clic y moviendo el cursor se dibuja un circulo con la medida de su área.

1.6 Puntos en el mapa

Aquí se agrupan las funciones que tiene que ver con las coordenadas específicas que el usuario ha hecho clic en el mapa.



1.6.1 Obtener coordenadas

Haciendo clic en el mapa aparece un recuadro con las coordenadas geográficas exactas de ese punto, en grados, minutos y segundo.

1.6.2 Insertar comentario

Se pude insertar un comentario en el punto de interés haciendo clic en el mapa. Aparecerá un formulario para rellenar con los datos que el usuario quiera especificar.

1.6.3 Ver información de las capas

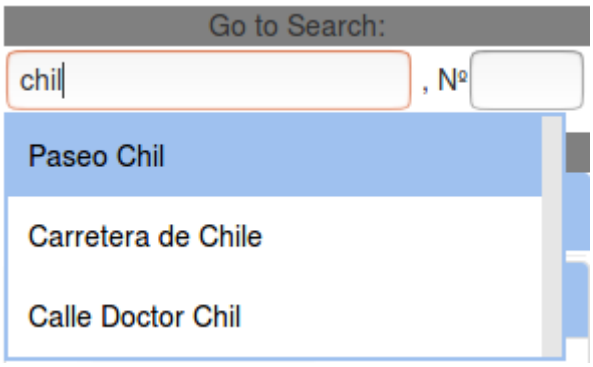
Al hacer clic en un punto del mapa, se muestra un modal con iframes que contienen la información de ese punto. Aparecerá un iframe por cada capa seleccionada, es decir, que tenga el tic activado en la visibilidad.

2. Color y tamaño del pincel y figuras

Haciendo clic en el botón amarillo por defecto, enseña un selector que muestra los colores, e incluso especificar en hexadecimal. Este cambia todo el color de las figuras y dibujos. Para cambiar el grosor hay que mover el rango de la entrada, a la izquierda hay un número del uno al diez para calibrar la referencia entre grosor y número.

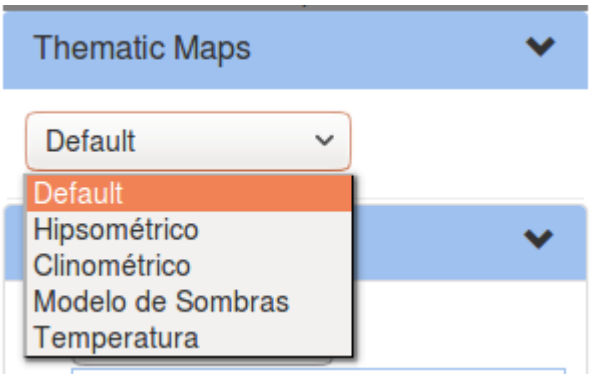
3. Buscador por calle y portal

A partir de la tercera letra, se despliega un panel con los resultados que coincidan. A medida que se escriba o borra estos resultados cambian dando al usuario un ayuda a la hora de elegir el destino. Si se introduce un numero en la entrada del portal, se buscará si existe ese portal en la calle. Cuando el usuario hace clic en el resultado, el mapa centra la vista en la dirección.



4. Mapas temáticos

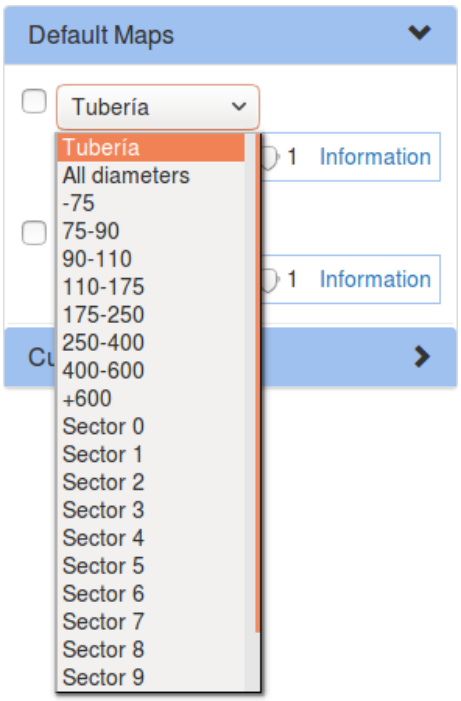
En este desplegable hay un selector con varios mapas con diferente información de las Islas Canarias.



- *Default*: Establece el mapa base que viene por defecto.
- *Hipsométrico*: Mapa topográfico que refleja el relieve mediante curvas de nivel que conectan puntos de la misma altura; cuanto menos espacio hay entre líneas, mayor es la pendiente relativa.
- *Clinométrico*: Mapa que muestra el grado de inclinación de la superficie de la Tierra mediante el uso de varios colores o sombreado para rangos críticos de pendiente.
- *Modelo de sombras*: Muestran la orientación de la pendiente y mediante su espesor y densidad proporcionan una sensación general de inclinación.
- *Temperatura*: Se muestra la temperatura media en colores de la zona según la temperatura durante todo el año, la radiación solar recibida y su potencia fotovoltaica.

5. Mapas por defecto

Contiene los dos mapas por defecto en la aplicación, junto con las opciones de cada una de las capas. Con tic marcado la capa se muestra en el mapa, desmarcado se oculta. Se puede modificar la transparencia de una capa moviendo la entrada de rango. El enlace *information* presenta la información general de la capa.



- *Tubería* muestra la red de agua de Las Palmas de Gran Canaria, además posee distintos estilos diferentes. Podemos diferenciar por código de colores según el diámetro de las tuberías, o por el sector al que pertenece.

- La capa *Comments* es aquella con los comentarios en los puntos de interés que ha insertado cualquier usuario.

6. Mapas personalizados

En esta sección se muestran las opciones de los mapas insertados a través del servicio WMS. Cada vez que se añaden mapas se agrupan en menús desplegables. Las opciones de estas capas son las mismas que se explicaron en los mapas por defecto. Es posible añadir todos los servicios que se quieran.

7. Interruptor cambio de mapa base

El mapa base por defecto es un mapa político, es posible cambiar la vista a uno por satélite haciendo clic en el interruptor.

12.2 Interfaz móvil

La interfaz móvil se compone de quince secciones diferentes, la mayoría de los botones son iguales a la aplicación de escritorio.

1. **Menú móvil**
   Oculta o muestra el menú entero.
2. **Menú mapas**
   Muestra un panel con los mapas, *Thematic*, *default* y *custom*. Así como las opciones que se explicaron en el apartado 5 de la interfaz de escritorio.
3. **Cargar servicio WMS**
   Despliega el modal para introducir el enlace del servicio WMS.
4. **Aumentar zoom**
   Aumenta el enfoque del mapa con un toque.

5. **Disminuir zoom**

   Disminuye el enfoque del mapa con un toque.

6. **Dibujo libre**

   Empieza a dibujar sin restricciones. También aparece un panel con los selectores de color y grosor.

7. **Medir distancias**

   El usuario dibuja trazos de líneas rectas que se van midiendo cuando el cursor se mueve.

8. **Medir áreas, polígono**

   Se dibuja un polígono y la aplicación devuelve el área de este.

9. **Medir área, circunferencia**

   Se dibuja un circulo y la aplicación devuelve el área de este.

10. **Obtener coordenadas**

    Al hacer un toque en el mapa, se obtiene las coordenadas geográficas del punto.

11. **Insertar comentario**

    Se añade un comentario escrito en las coordenadas del punto.

12. **Ver información de las capas**

    A partir de las capas que puedan verse se muestra la información específica del punto en un modal.

13. **Buscar calle y portal**

    Se despliega un panel con un formulario para buscar el nombre de la calle y el portal

14. **Ir a las coordenadas**

    Aparece una tabla en el que se introducen las coordenadas geográficas o en UTM para que el mapa centra la vista en ese punto.

15. **Interruptor cambio mapa base**

    Cambia la base del mapa entre político y satélite.