APLICACIÓN *WEB MAPPING* PARA DAR A CONOCER LOS SENDEROS DE LA ISLA DE GRAN CANARIA

Grado en Ingeniería Informática

Autor: Néstor Santana Hernández

Tutor: Dr. Agustín R. Trujillo Pino

Co-tutor: D. Salvador J. Hernández García

Las Palmas de Gran Canaria, Mayo 2019

“Nadie que enciende una lámpara, la tapa con una vasija o la mete debajo de la cama, sino que la pone en un candelero para que los que entren vean la luz.”

(Lc 8,16)

Agradecimientos

Me gustaría aprovechar estas líneas para agradecer a Agustín Trujillo la labor de dirigir este proyecto, así como la paciencia que ha tenido conmigo. Además, también merece mención especial Salvador Hernández, que nos ha facilitado los caminos y la información necesaria para llevar a cabo este trabajo.

Por otra parte, merecen mi agradecimiento y admiración mi familia, mis compañeros y amigos, en particular Alejandro, Ángel, Cristian, Jesús, y como no, Kimberley, que me han acompañado en los momentos buenos y no tan buenos de estos años de carrera. Gracias de corazón.

Finalmente, también me gustaría tener una especial mención a los profesores del *Dpto. de Cartografía y Expresión Gráfica* de la *Universidad de Las Palmas de Gran Canaria* (en especial a mi padre) por su predisposición a ayudar y colaborar en lo que fuera necesario, así como a los diferentes técnicos y profesionales del mundo de la *Topografía* y los *SIG* que han aportado su granito de arena en este proyecto.

Índice

Resumen

Con este proyecto se pretende dar a conocer los diversos caminos y senderos de la isla de Gran Canaria, así como la divulgación de toda la información cultural, etnográfica, medioambiental y gastronómica vinculada a ellos. Por tanto, se ha realizado una aplicación *web mapping* que permita, no solo, mostrar la red de senderos de la isla, sino también la posibilidad de crear cuántos caminos se desee a partir de los puntos contenidos en dicha red. Para ello, el uso de una base de datos espacial ha sido fundamental para poder desempeñar esta funcionalidad.

Abstract

This project aims to publicize the various paths and trails of the island of Gran Canaria, as well as the divulgation of all cultural, ethnographic, environmental and gastronomic information linked to them. For this reason, a *web mapping* application has been made that allows, not only, to display the network of trails on the island, but also the possibility of creating the tracks that you want from the points contained this network. Therefore, the use of a spatial database has been fundamental to be able to perform this functionality.

1. Estado actual y objetivos iniciales

1.1. Situación actual

Los sistemas de información geográfica (*SIG*), también denominados *GIS* (*Geographic Information System*), se encuentran dentro del extenso campo de los sistemas de información. Un sistema de información se puede definir como un determinado conjunto de elementos organizados que, una vez procesados y almacenados, se usan para tomar decisiones y controlar una organización. Resulta lógico, por tanto, que en los sistemas de información, debe existir una estrecha unión entre información y sistemas informáticos.

En esta línea aparecen los sistemas de información geográficos. Lo que caracteriza a los *SIG*, con respecto a los sistemas de información, es que su eje central se concentra en el procesamiento y almacenamiento de información geográfica. Por tanto, *se puede afirmar que un SIG es un sistema de información capaz de integrar, almacenar, editar, analizar, compartir y mostrar la información geográfica referenciada* [1].

En la década de los años 1950 y 1960 aparecen los SIG, pero no comenzaron a desarrollarse por completo hasta los años 1980 y 2000, gracias a los avances tecnológicos y al abaratamiento de los recursos informáticos. Como consecuencia de ello, los medios tecnológicos comenzaron a aparecer tanto las entidades públicas como las educativas, así como las empresas privadas.

Actualmente, los *SIG* se encuentran bastante desarrolladas, pero como todas las tecnologías, siguen en constante crecimiento. En muchos campos, como por ejemplo, la gestión forestal, navegación, la gestión del transporte público, el registro de la propiedad, tanto pública como privada, se han logrado avances que permiten, no solo, que se mejore la gestión de la información en estos sectores, sino que también se fomente el uso de las tecnologías *SIG* en otros campos. Es importante destacar, que han surgido nuevas herramientas vinculadas los *SIG*, como pueden ser los *Internet-GIS*, la tecnología *WMS* o el uso de los *SIG* en los GPS.

No obstante, aunque el objetivo de este proyecto no consiste en desarrollar un *SIG*, es importante comprender en qué consiste esta tecnología de la información, ya que la *web mapping* elaborada para este proyecto emplea, principalmente, una base de datos espacial, que es uno de los elementos más importantes y determinantes de un *SIG*.

Las bases de datos espaciales no dejan de ser una base de datos convencional que permite almacenar información. La peculiaridad de las bases de datos espaciales es que no solo posibilitan guardar datos georreferenciados, sino que también permiten guardar información relacionada a esos datos espaciales, posibilitando crear así capas temáticas que facilitan a los usuarios llevar una gestión rápida y sencilla de esta información.

Por otra parte, resulta interesante comentar en este apartado algunos de los sitios web que, actualmente en Canarias, emplean este tipo de tecnologías, y que al mismo tiempo, han servido de referente en el desarrollo de este proyecto. El primero de ellos (primera imagen de la *Ilustración 1*) es https://lagomera.forwalk.org/es/, cuya funcionalidad principal consiste en crear senderos a partir de puntos geográficos concretos que el usuario puede ir combinando mediante un formulario. El otro sitio web (segunda imagen de la *Ilustración 1*) es

http://senderosdelagomera.com/, que a diferencia del anterior, posibilita la creación de senderos interactuando en el mapa con los nodos de la red.

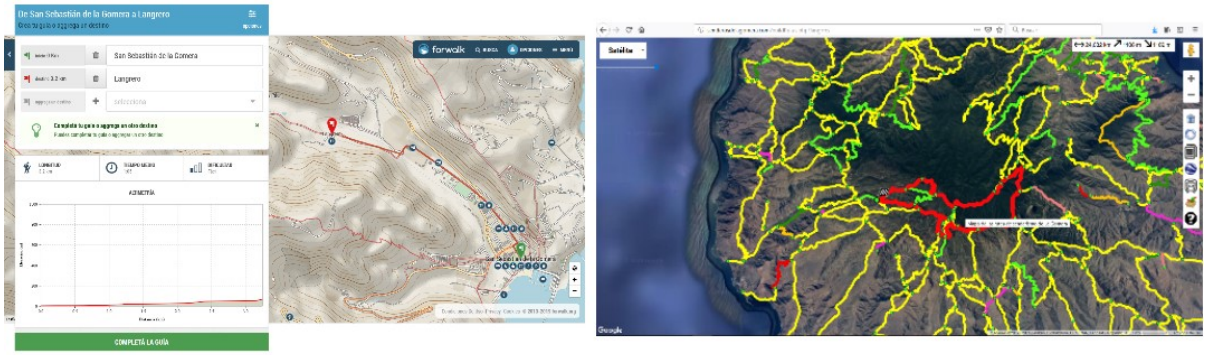


Ilustración 1.- Comparativa entre las dos páginas de la Gomera

1.2. Motivación

El proyecto surge a partir de una propuesta publicada en la página de la Escuela de Ingeniería Informática. Se trata de una idea que planteaba Salvador, el co-tutor del proyecto. Tras realizar las prácticas curriculares del grado sobre los sistemas de información geográficos, me resultó interesante la propuesta sobre este proyecto, con lo cual decidí escogerlo, no solo porque me resultaba una propuesta interesante, sino porque también me parecía un trabajo de fin de título donde poder profundizar y conocer más sobre este tema.

1.3. Objetivos

En relación a los objetivos, inicialmente se especificaron en la solicitud del trabajo de fin de título los siguientes:

- Ofrecer al usuario una aplicación que le permita crear o elegir senderos de la isla que se encuentran registrados en la base de datos. Estos senderos estarán organizados por municipios y por temas.
- Mostrar al usuario una ficha informativa con los datos técnicos de la misma (horario estimado, distancia, tiempo estimado para completar la ruta, entre otros).
- Elaborar y diseñar la base de datos en colaboración con otro compañero.

Es importante señalar como aclaración a los objetivos arriba descritos, que estos fueron marcados al comienzo del desarrollo del trabajo de fin de título y que responden a las necesidades planteadas en la propuesta de trabajo. Tras el desarrollo, y debido a que se proporcionó un pequeño subconjunto de la información con la que se quiere realizar el proyecto, solo se han podido organizar los senderos por municipio.

2. Justificación de las competencias específicas cubiertas

Las competencias desarrolladas en este trabajo son las siguientes:

- **ISO1.- Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software.**

  Con este proyecto se responde a las necesidades de un usuario, que es el que propone la idea para este trabajo de fin de título. Con lo cual, se llevan a cabo todos los requisitos que este propone y se desarrollan de tal manera que permita que la aplicación sea robusta y fiable, pero sobre todo que le genere utilidad.

- **IS02.- Capacidad para valorar las necesidades del cliente y especificar los requisitos software para satisfacer estas necesidades, reconciliando objetivos en conflicto mediante la búsqueda de compromisos aceptables dentro de las limitaciones derivadas del coste, del tiempo, de la existencia de sistemas ya desarrollados y de las propias organizaciones.**

  Como bien se comentaba anteriormente, se trata de un proyecto donde los requisitos del usuario juegan un papel fundamental. Una buena especificación de los mismos es crucial para poder generar una aplicación informática que permita satisfacer con éxito las expectativas que el usuario, en este caso propietario, desea.

  Asimismo, el contacto con el usuario ha sido importante en este proyecto, no solo para verificar que los requisitos que plantea se han implementado correctamente, sino también para acordar con él los tiempos y comentarle las limitaciones tecnológicas que se encuentran en el desarrollo.

- **IS04.- Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales.**

  Esta competencia se ha trabajado bastante en este trabajo de fin de título. La información proporcionada y las tecnologías empleadas dificultaban el desarrollo de los requisitos del usuario, con lo cual era frecuente el análisis de los problemas que se iban encontrando, así como el diseño, desarrollo, implementación y verificación de soluciones que dieran respuesta a estas dificultades.

  Además, uno de los propósitos de esta memoria consiste documentar formalmente las soluciones software adoptadas para solventar los problemas encontrados en el desarrollo de este proyecto.

- **TFG01.- Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas.**

Aunque el desarrollo de este trabajo se ha realizado inicialmente por dos alumnos, cabe destacar que cada uno desarrollará funcionalidades diferentes para este proyecto. Por tanto, se cumple esta competencia en lo que se refiere a la realización individual del ejercicio.

También, la realización de esta memoria tiene como objetivo, tal y como se comentó anteriormente, la documentación del proyecto realizado para que un tribunal pueda evaluarlo. Para ello será necesario una presentación del trabajo realizado ante dicho tribunal.

3. Aportaciones

3.1 Aportaciones al entorno socio-económico y social

La aplicación *web mapping* desarrollada tiene como finalidad ofrecerle un recurso digital al senderista para que pueda descubrir y conocer con detalle los caminos, y todo lo vinculado a ello, que existen en la isla de Gran Canaria, con lo cual, no se trata de un aporte al ámbito socio-económico, sino más bien al social y cultural, más concretamente, al patrimonio medio ambiental de nuestra isla.

En esta línea, uno de los aportes más significativos que este proyecto realiza a la divulgación y promoción del senderismo en Canarias, es que la aplicación web desarrollada emplea una capa de información geográfica, que en este caso es la red de senderos, y que tiene enlazada a ella información relacionada con los caminos de la red. Gracias a esta red, y a las herramientas que permiten *enrutar*, se pueden crear tantos senderos como el usuario desee. Esta funcionalidad apenas se puede apreciar en las páginas web existentes que abordan esta temática en Canarias.

3.2 Aportación personal

En lo que a la formación académica y profesional respecta, el proyecto me ha aportado una buena experiencia en el desarrollo de software. El contacto con la persona interesada en el proyecto para obtener así los requisitos ha hecho que esta experiencia fuera más enriquecedora, pues me ha acercado a lo que realmente supone trabajar en el desarrollo software.

Al mismo tiempo, trabajar con elementos propios de los *SIG* ha significado otra fuente de aprendizaje, pues me ha ayudado a profundizar, no solo en la forma de trabajar de los sistemas de información geográfica, sino también el uso de herramientas potentes que se emplean en este campo. Asimismo, el manejo y empleo de estas herramientas en el desarrollo de una aplicación web me ha permitido aprender a enlazar los *SIG* con las tecnologías informáticas.

4. Recursos utilizados

4.1 Tecnologías hardware empleadas

Las tecnologías hardware empleadas en este proyecto han sido:

- Ordenador portátil *Acer Aspire V5-572PG*, Intel Core i7-3537U 2.0 GHz with Turbo Boost up to 3.1 GHz, 8 GB DDR3 Memory, NVIDIA GeForce GT 750M with 4GB Dedicated VRAM.
- Monitor *HP 27es* de 27' y FullHD de 1920x1080 pixels.
- Teléfono móvil *Huawei Mate 10* con EMUI 9.0.0
- Servidor de la ULPGC, situado en la Escuela de Ingeniería Informática, donde se encuentra alojada la página *web mapping* creada en este trabajo de fin de título.

4.2 Análisis de las tecnologías software existente

4.2.1 Puntos a favor y en contra de Drupal y WordPress

Los sistemas de gestión de contenido que se han analizado para este proyecto han sido *Drupal* y *WordPress*, pues son los más usados en el desarrollo de sitios web.

Actualmente, *Drupal* supone un 4.6% de la participación en el mercado de los gestores de contenido, mientras que un 59.8% de esta participación se le debe a *WordPress*. Estas cifras han hecho pensar y reflexionar, en un primer momento, sobre estas tecnologías, así como en sus ventajas y desventajas [2].

Si analizamos en profundidad a *WordPress* podemos ver que destaca por su facilidad de uso y de obtención de ayuda, su extensibilidad y por sus costes bajos en desarrollo.

La facilidad de uso se debe, principalmente, no solo a la sencilla configuración que tiene, sino que además cuenta con una interfaz gráfica que permite gestionar el sitio web de forma sencilla y asequible, sobre todo para aquellas personas que no son desarrolladores.

Por otra parte, al ser una tecnología usada por mucha más gente, existe una comunidad a la que se le puede pedir ayuda cuando comienzan a florecer los problemas. En esta línea, la existencia de una comunidad hace posible que existan una serie de plugins y temas que se puedan incorporar con facilidad sitio web.

Además, con *WordPress* se pueden adquirir soluciones a determinados problemas que ya han sido realizadas por otros desarrolladores, con lo cual disminuye los costes de desarrollo.

No obstante, *Drupal* es más potente en todo lo relacionado a la cuestión de la seguridad. *Drupal* tiene un sistema de control de acceso que permite crear nuevos roles de usuario con permisos individuales, mientras que *WordPress* viene con cinco roles de usuario básico. Asimismo, *Drupal* tiene un soporte lingüístico más potente y un sistema de taxonomía (forma de agrupar las publicaciones) mejor desarrollado, por lo que permite manejar grandes cantidades de información.

Como bien se comentaba anteriormente, *Drupal* tiene como punto fuerte la seguridad. En el pasado 2016 sólo un 2% de los sitios web que usan *Drupal* sufrió ataques cibernéticos, con lo cual es una cifra importante que demuestra la inmunidad de esta tecnología ante estas situaciones.

Otro aspecto a tener en cuenta es la facilidad de uso y la curva de aprendizaje. *WordPress* es más fácil de usar y de aprender. La creación de contenido con *Drupal* es mucho más compleja que con *WordPress*. Todos los temas que usa *Drupal* son perfectamente codificados y personalizados por desarrolladores, por lo que se va a necesitar un programador con cierta experiencia en *Drupal* para lograr que el contenido del sitio web se vea bien [3].

Sin embargo, con *WordPress* es mucho más fácil empezar sin nada en un sitio web y tener una página con un contenido bien organizado, agradable a la vista y con unas bonitas vistas.

Después de realizar este análisis y tras poner en marcha ambos gestores de contenido se encontraron diferentes dificultades que llevaron a adoptar otra decisión. El primer problema encontrado al configurar *Drupal* fue la estructuración de los directorios y la filosofía de programación que tiene este gestor de contenido. La comprensión de cómo se crean módulos de contenido o como se cargan ficheros *HTML* para mostrar información resultaba bastante complejo, no solo para los desarrolladores, sino también para los usuarios de la aplicación que no tienen conocimientos de programación.

El segundo problema surgió al intentar conectar la base de datos *MySQL* que usa *WordPress* con *QGIS*. Tal y como se comentará posteriormente, *QGIS* es la herramienta que permitirá almacenar en la base de datos los senderos y rutas con una componente espacial. Esta dificultad puede ser solventada con un plugin, pero no permitía su instalación debido a errores internos.

Finalmente, se optó por desarrollar las páginas *HTML* editando el código manualmente usando un entorno de desarrollo. De esta manera se evitan los problemas comentados, con el único inconveniente de que el mantenimiento de la aplicación web la tendrá que llevar una persona con conocimientos de programación.

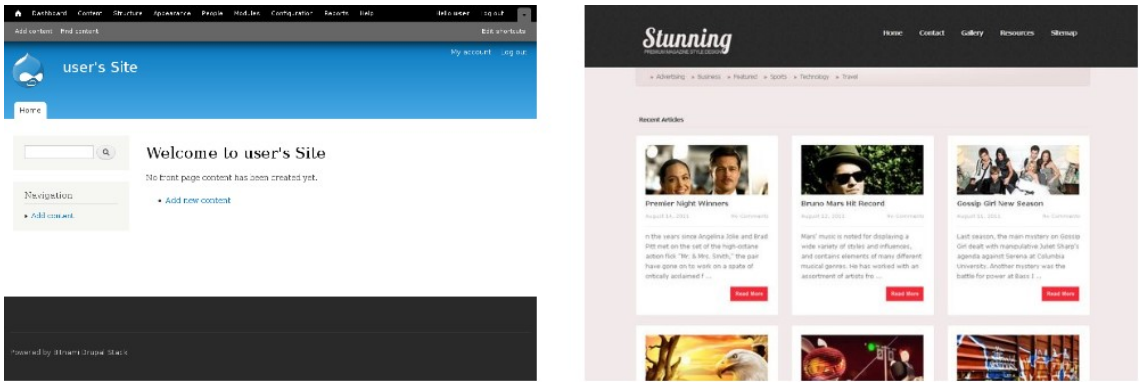


Ilustración 2.- Drupal (izquierda) y WordPress (derecha)

4.2.2 PostgreSQL, PostGIS y QGIS

Tanto *PostgreSQL* como *PostGIS* son herramientas indispensables para almacenar los senderos, poder representarlos y *enrutarlos* en la aplicación web.

PostgreSQL es un gestor de bases de datos, mientras que *PostGIS* es una extensión de *PostgreSQL* que permite añadir una componente geográfica a la base de datos, y así poder representar en las tablas puntos, líneas y polígonos.

Por otra parte, *QGIS* es un software libre que permite crear archivos vectoriales y exportarlas a tablas espaciales. Este programa resulta bastante útil para convertir los ficheros *KML* que se han proporcionado para realizar el proyecto en archivos *SHAPEFILE*, y así poder generar las tablas necesarias para la aplicación.

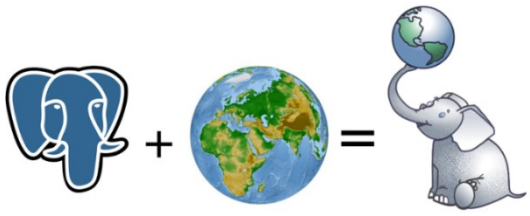


Ilustración 3.- Imagen ilustrativa de PostGIS

4.2.3 pgRouting y Leaflet

pgRouting es una extensión de *PostgreSQL*, que junto con *PostGIS*, pueden *enrutar* y analizar redes de una base de datos espacial. Por otra parte, *Leaflet* es una biblioteca de *JavaScript* que permite representar mapas en entornos web. *Leaflet* permite la representación de rutas de una forma muy sencilla.

Por tanto, ambas tecnologías nos aportan las herramientas necesarias para desarrollar las funcionalidades principales del proyecto: la representación gráfica de un sendero en un mapa y el enrutamiento entre diferentes nodos de la red de senderos.

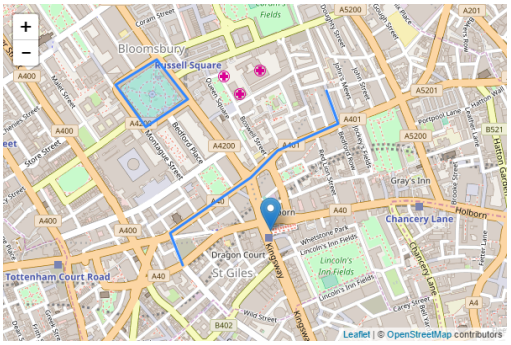


Ilustración 4.- Leaflet

4.2.4 MAMP y XAMPP para el servidor local

Lo que está claro es que para poder alojar, al menos localmente, la aplicación web era necesario emplear un servidor. Se han barajado estos dos opciones: *MAMP* y *XAMP*. Ambas hacen uso de Apache como servidor web y de *MySQL* como gestor de bases de datos.

La elección de una u otra se ha basado principalmente en la facilidad de instalación y de configuración. Con *MAMP*, no solo es más sencillo arrancar el servidor y alojar en él los ficheros de la web, sino que también conlleva un proceso de instalación muy corto y poco costoso.

4.2.5 Tecnologías de programación

Al tratarse de un sitio web, los lenguajes principales de programación serán *PHP*, *JavaScript* y *CSS*. También se empleará la biblioteca multiplataforma de *Bootstrap* para el diseño de la aplicación.

El motivo por el que se han seleccionado estás tecnologías ha sido principalmente a que el servidor instalado emplea *PHP* como lenguaje de programación del lado del servidor. Los otros lenguajes son una buena opción para aplicarle un buen diseño y unas buenas funcionalidades al sitio web.

Asimismo, se ha empleado *JQuery* para modificar y seleccionar los elementos creados en las páginas *HTML*, así como para crear alguna animación o efecto sobre los mismos. *JQuery* es una biblioteca creada para JavaScript que manipula de forma simplificada el código *HTML*.

Por otra parte, para realizar las consultas a la base de datos se ha hecho uso de *AJAX*, que es una tecnología que se utiliza para realizar peticiones desde *JavaScript* mediante protocolos *HTTP/HTTPS*.

4.2.6 Otras herramientas y bibliotecas

En cuanto al entorno de programación, se ha usado *NetBeans*. Se trata de un entorno de desarrollo gratuito que permite la creación de todo tipo de aplicaciones *Java*. También dispone de paquetes que posibilitan la creación de proyectos en *C/C++* y *PHP*. En este caso en particular se ha empleado para un proyecto creado en *PHP*.

Por otro lado, se ha hecho uso de *pgAdmin 4*, que es una herramienta libre para gestionar las bases de datos *PostgreSQL*, así como sus extensiones. En lo que se refiere a este proyecto, no solo se ha empleado para administrar una base de datos local, sino también una base de datos instalada y configurada en el servidor donde se ha alojado el proyecto.

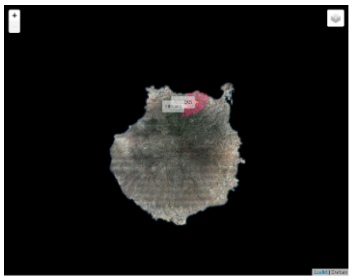
También, se han usado los programas *Filezilla* (en su versión de cliente) y *PuTTY* para realizar el despliegue de la aplicación. *Filezilla* es un software libre que permite realizar transferencias de ficheros a un servidor usando los protocolos *FTP*, *SFTP* y *FTP* sobre *SSL/TLS* de forma interactiva. Por otra parte, *PuTTY* es otro software libre que posibilita realizar conexiones mediante protocolos *SSH*, *Telnet*, *rlogin*, entre otros.

Para representar gráficos se ha empleado d3.js, una biblioteca de *JavaScript* que permite la representación de información mediante gráficos de diferentes tipos (de barras, de líneas, de áreas, circular, etc.). Resulta bastante útil porque se puede obtener el gráfico como una imagen gráfico vectorial escalable (*SVG*) e incrustarlo en la página *HTML*.

4.2.7 Web Map Service

Otra de las tecnologías usadas para poder representar el mapa han sido los servicios web de mapas. Un servicio web de mapas o *Web Map Service (WMS)* es un protocolo estandarizado definido por el *Open Geospatial Consortium* (OGC) que proporciona imágenes de mapas a partir de una determinada información geográfica. Principalmente, los mapas que proporcionan los *WMS* vienen en formato *JPEG*, *PNG*, *GIF* e incluso en *SVG* [4]. En este proyecto se emplean las fotos ofrecidas por *GrafCan*, *openstreetmap.org* y por *stamen.com*.

Para poder consumir estos servicios con *Leaflet* simplemente hay que invocar una función a la que se le pasa la *URL* del *WMS* y una serie de propiedades, tal y como se verá en las siguientes imágenes:



```
function getOrtoExpress(){
    return L.tileLayer.wms('https://idecan3.grafcan.es/ServicioWMS/OrtoExpress?', {
                layers: 'ortoexpress',
                format: 'image/jpeg',
                transparent: true,
                attribution: "Grafcan",
                crs: L.CRS.EPSG4326,
                maxZoom: 22,
                minZoom:10
            });
}
```

Fotografía aérea (Ortofoto express de GrafCan)

```
function getCallejero(){
    return L.tileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png',{
        attribution: "OpenStreerMap",
        maxZoom: 22,
        minZoom:10
    });
}
```

Callejero (OpenStreetMap)

Ilustración 5.- Fotografía aérea y callejero junto a la invocación a su WMS

```
function getLidar(){
    return L.tileLayer.wms("http://idecan3.grafcan.es/ServicioWMS/MTL?", {
            layers: 'LIDAR_MTL',
            format: 'image/jpeg',
            transparent: true,
            attribution: "Grafcan",
            crs: L.CRS.EPSG4326,
            maxZoom: 22,
            minZoom:10
        });
}
```

Modelo de terreno (Modelo de Terreno LIDAR de GrafCan)

```
function getTopografico(){
    return L.tileLayer.wms("http://idecan3.grafcan.es/ServicioWMS/MTI?", {
            layers: 'WMS_MTI',
            format: 'image/png',
            transparent: true,
            attribution: "Grafcan",
            crs: L.CRS.EPSG4326,
            maxZoom: 22,
            minZoom:10
        });
}
```

Mapa topográfico (Mapa topográfico de GrafCan)

Ilustración 6.- Modelo de alturas y mapa topográfico junto a la invocación a su WMS

En la imagen anterior, si se observa la captura que carga la *fotografía aérea*, se puede ver como la *ortofoto*[1] obtenida proviene del *WMS* de *GrafCan*. La petición se hace a *https://idecan3.grafcan.es/ServicioWMS/OrtoExpress?* y se le pasan parámetros como el nombre de la capa, el formato de la imagen, el sistema de referencia (*crs*), entre otros.

[1] Representación fotográfica de una zona del globo terráqueo.

5. Requisitos

Un requisito es *una condición o capacidad que un usuario necesita para resolver un problema o lograr un objetivo* [5]. Esta definición que sostiene el estándar *IEE* sobre los requisitos se aproxima bastante a la motivación de este proyecto, pues se trata de una idea que un usuario quiere llevar a cabo. Por ello, el contacto con este usuario es vital para el buen desarrollo del mismo, y así lograr que la aplicación sea cien por cien útil.

Tras una reunión inicial con Salvador, la persona interesada en el proyecto, se comenzaron a identificar los requisitos que la aplicación web debe tener. Asimismo, conforme se fueron completando las diferentes iteraciones del proyecto, se iban concertando nuevas reuniones con Salvador.

El objetivo de estas reuniones consistió en fijar la atención en todo lo que afecta al usuario, concretamente con la persona interesada. No solo se trataba de la identificación de los requisitos, sino también que éstos quedaran completamente definidos. De esta manera, se consiguió que el usuario se implicara en el proyecto desde el principio.

La especificación de los requisitos se ha recogido en una plantilla, que se mostrará a continuación. Además, para facilitar el modelado y como técnica de requisitos se ha realizado un diagrama de casos de uso, con su especificación, con el fin de representar la realidad a la que nos enfrentamos.

5.1 Descripción de los requisitos

A continuación se detallarán los requisitos funcionales y no funcionales que se han desarrollado en esta aplicación web.

ID	TIPO	DESCRIPCIÓN	JUSTIFICACIÓN	INTERESADO	ROL	CRITERIOS DE VALIDACIÓN	IMPORTANCIA	PRIORIDAD	DEPENDENCIAS
RQ001	Funcional	El sistema deberá permitir el *enrutamiento* entre dos puntos clicados en la red	Para mostrar el sendero más óptimo.	Salvador Hernández	Cliente	Se dibuja el sendero en el mapa.	Alta	Alta	
RQ002	Funcional	El sistema deberá posibilitar el *enrutamiento* concatenando varios puntos cualesquiera clicados en la red	Para mostrar el sendero más óptimo.	Salvador Hernández	Cliente	Se dibuja el sendero en el mapa.	Alta	Alta	
RQ003	Funcional	El sistema deberá permitir el *enrutamiento* entre dos puntos de enlace de la red	Para mostrar el sendero más óptimo.	Salvador Hernández	Cliente	Se dibuja el sendero en el mapa.	Alta	Alta	
RQ004	Funcional	El sistema deberá mostrar un perfil topográfico al obtener el sendero	Para mostrar las diferentes altitudes del sendero.	Salvador Hernández	Cliente	Se muestra el perfil de altura.	Alta	Media	RQ001,RQ002, RQ003
RQ005	Funcional	El sistema deberá mostrar otro sendero cuando se haya clicado un punto inicial y otro final	Para mostrar una alternativa.	Salvador Hernández	Cliente	Se muestra otro sendero en el mapa.	Alta	Alta	RQ001,RQ002, RQ003
RQ006	Funcional	El sistema deberá mostrar información del sendero construido	Para que el usuario pueda conocer las características del mismo.	Salvador Hernández	Cliente	Se muestra un panel con la información.	Media	Baja	RQ001,RQ002, RQ003
RQ007	Funcional	El sistema deberá permitir la visualización de la red en el mapa	Para que el usuario pueda visualizarla.	Salvador Hernández	Cliente	Se muestra la red en el mapa.	Alta	Media	
RQ008	Funcional	El sistema deberá permitir la visualización de los puntos de enlace de la red	Para que el usuario pueda visualizarlos.	Salvador Hernández	Cliente	Se muestran los puntos de la red en el mapa.	Alta	Media	

RQ010	Funcional	El sistema posibilitará la visualización de los puntos de interés	Para que el usuario pueda situarlos en el mapa.	Salvador Hernández	Cliente	Los puntos de interés son dibujados en el mapa.	Media	Baja	
RQ011	Funcional	El sistema posibilitará cambiar la capa base del mapa	Para que el usuario pueda ver la imagen del mapa.	Salvador Hernández	Cliente	Se visualiza la capa en el espacio de la página reservado para el mapa.	Media	Baja	
RQ011	Funcional	El sistema permitirá añadir capas transparentes	Para que el usuario pueda combinarlas en el mapa.	Salvador Hernández	Cliente	Se visualiza la capa transparente sobre la capa base.	Media	Baja	
RQ012	Funcional	El sistema posibilitará fijar el nivel de opacidad a las capas transparentes	Para que el usuario pueda visualizar varias capas transparentes simultáneamente.	Salvador Hernández	Cliente	Se visualizan varias capas transparentes al mismo tiempo.	Media	Baja	RQ011
RQ013	Funcional	El sistema permitirá visualizar información sobre los caminos registrados	Para que el usuario conozca el detalle de los mismos.	Salvador Hernández	Cliente	Se muestra la información del camino seleccionado.	Media	Baja	
RQ014	Funcional	El sistema permitirá la visualización de información sobre los puntos de interés registrados	Para que el usuario conozca el detalle de los mismos.	Salvador Hernández	Cliente	Se muestra la información del punto de interés seleccionado.	Media	Baja	
RQ015	Funcional	El sistema permitirá la visualización de información sobre los servicios registrados.	Para que el usuario conozca el detalle de los mismos.	Salvador Hernández	Cliente	Se muestra la información del servicio seleccionado.	Media	Baja	
RQ016	Funcional	El sistema posibilitará la identificación del administrador	Para que realice las funcionalidades que le corresponden.	Salvador Hernández	Cliente	Se inicia sesión tras hacer la identificación en el sistema.	Media	Media	
RQ017	Funcional	El sistema permitirá al administrador acceder a la página de configuración	Para que pueda modificar los parámetros de	Salvador Hernández	Cliente	Los parámetros de configuración quedan	Media	Media	RQ016

			configuración[2] de la página.			guardados tras la modificación.			
RQ018	Funcional	El sistema deberá mostrar la separación entre los puntos clicados en el mapa y los puntos extremos del sendero	Para que el usuario pueda apreciar la distancia que hay entre ambos.	Néstor Santana	Desa-rrollador	Se muestra una línea discontinua entre el punto inicial clicado y el punto inicial del sendero, así como entre el punto final clicado y el punto final del sendero.	Media	Media	RQ001,RQ002, RQ003
RQ019	No funcional	El sistema deberá tener conexión a Internet	Para poder funcionar.	Néstor Santana	Desa-rrollador	La aplicación web funciona.	Alta	Alta	
RQ020	No funcional	El sistema podrá ejecutarse en todos los navegadores	Para que pueda verse en dispositivos móviles.	Néstor Santana	Desa-rrollador	Se puede acceder mediante el navegador de un teléfono móvil.	Media	Media	
RQ021	No funcional	El sistema deberá tener acceso a los *CDN* de las bibliotecas necesarias	Para que se puedan realizar las funcionalidades que se han desarrollado.	Néstor Santana	Desa-rrollador	La aplicación web funciona.	Alta	Alta	Todos

Tabla 1.- Especificación de requisitos

[2] Los parámetros de configuración son los colores de las líneas que trazan los senderos y caminos, la anchura de las mismas, el nivel de opacidad, así como las imágenes de los diferentes iconos que se pueden visualizar en el proyecto.

5.2 Casos de uso

En las siguientes líneas se mostrarán los casos de uso que representan los requisitos arriba expuestos. Estos casos de uso han ayudado a comprender la realidad a la que nos enfrentamos, ya que la muestran de forma simplificada y razonada.

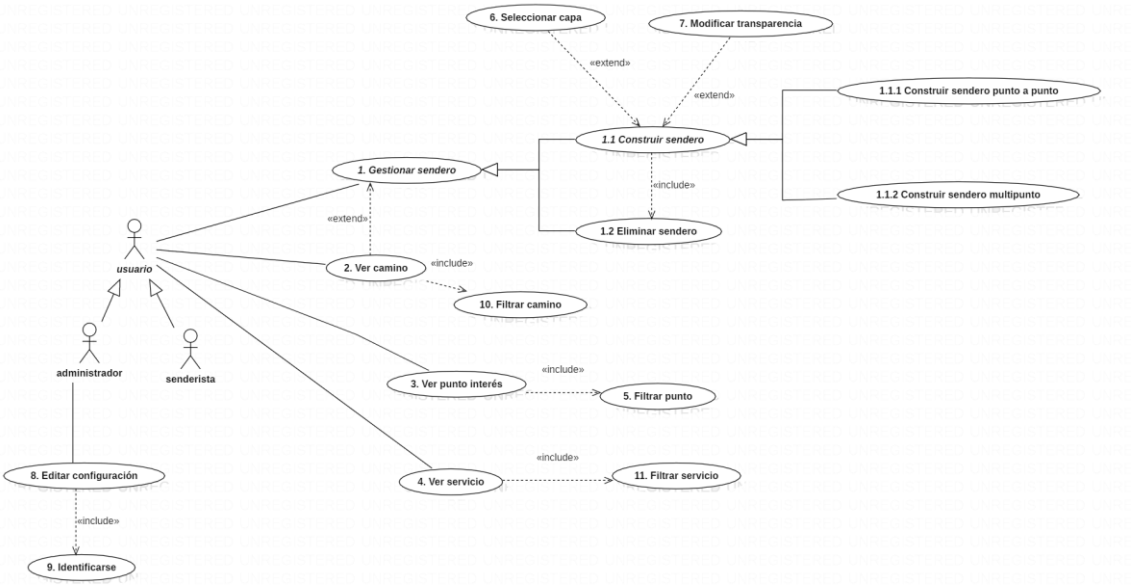


Ilustración 7.- Diagrama de casos de uso

En este diagrama, se puede observar que los casos de uso más importante son el 1.1, pues representa la funcionalidad más relevante del proyecto: *Construir sendero*. Esta funcionalidad se puede desarrollar de dos maneras diferentes, o bien construimos un sendero entre dos puntos o lo construimos a partir de muchos puntos. Para poder realizar esta funcionalidad es necesario tener otra que permita eliminar los senderos construidos del mapa. Los casos de uso 6 y 7 son los que permiten modificar las capas del mapa y modificar su transparencia.

Los casos de uso 8 y 9 se encargan de la configuración de la página y de la autentificación del usuario administrador, mientras que los casos de uso 2, 3 y 4 tienen la función de mostrar la información sobre los caminos, puntos de interés y los servicios (centros de salud, policía, etc.).

A continuación se mostrará la especificación de los casos de uso más relevantes:

CASO DE USO	1.1.1	Construir sendero punto a punto		
Descripción		El usuario crea un sendero entre dos puntos.		
Actores		Usuario		
Precondiciones		-		
Flujo normal		**Paso**	**Acción**	
		1	El sistema elimina del mapa los senderos creados con anterioridad. (CU 1.2)	
		2	El usuario selecciona el punto inicial.	
		3	El usuario selecciona el punto final.	
		4	El sistema obtiene el sendero más óptimo entre los dos puntos.	
		5	El sistema dibuja el sendero en el mapa.	
		6	El sistema muestra el perfil topográfico del sendero.	
Postcondiciones		-		
Variaciones		**Paso**	**Acción**	
Extensiones		**Paso**	**Condición**	**Caso de Uso**
Excepciones				
Observaciones				

Tabla 2.- Especificación del caso de uso 1.1.1

CASO DE USO	1.1.2	Construir sendero multipunto		
Descripción		El usuario crea un sendero entre muchos puntos.		
Actores		Usuario		
Precondiciones		-		
Flujo normal		**Paso**	**Acción**	
		1	El sistema elimina del mapa los senderos creados con anterioridad. (CU 1.2)	
		2	El usuario un punto de partida en el mapa.	
		3	El usuario selecciona otro punto.	
		4	El sistema obtiene el sendero más óptimo entre los dos puntos.	
		5	El sistema dibuja el sendero en el mapa.	
		6	El sistema muestra el perfil topográfico del sendero.	
		7	Se vuelve al paso 3 mientras el usuario no haya confirmado la finalización del sendero.	
Postcondiciones		-		
Variaciones		**Paso**	**Acción**	
Extensiones		**Paso**	**Condición**	**Caso de Uso**
Excepciones				
Observaciones				

Tabla 3.- Especificación del caso de uso 1.1.2

6. Diseño

6.1 Arquitectura

La aplicación *web mapping* desarrollada para este proyecto ha seguido una arquitectura en tres capas. Se trata de una de las arquitecturas más usadas en la gran mayoría de los sistemas informáticos, como por ejemplo una tienda *online* o aplicaciones que manipulen o empleen unos ciertos datos, como es el caso de esta aplicación. Se distingue, por como su nombre indica, dividir el software en tres partes: la capa de presentación, la capa de negocio y la capa de datos.

Como bien es sabido, todo sistema que maneja datos dispondrá de una base de datos donde estos se encuentren alojados, así como de una interfaz de usuario que permita a la persona que va a usar la aplicación interaccionar con estos datos. También, una parte del sistema se encargará de procesar los datos, así como gestionar lo que se hace con ellos.

Por tanto, en términos generales, se pueden definir las diferentes capas de la siguiente forma [6]:

- **Capa de presentación**: Se trata de la capa que ve el usuario, y que por ello también se le denomina *capa de usuario*. Su objetivo es proporcionarle al usuario una vista amigable de la aplicación para que este pueda interaccionar con ella y visualizar la información.
- **Capa de negocio**: Se trata de la capa intermedia que se comunica con la capa de presentación y con la capa de datos. En ella se encuentra toda la lógica de la aplicación, porque es en ella donde se fijan las reglas que debe cumplir el sistema. En esta aplicación en concreto, la capa de negocio se encargará de obtener y ordenar la información solicitada a la base de datos, ya que se trata de una *web* que principalmente muestra información al usuario.
- **Capa de datos** o **persistencia**: Se trata de la capa que almacena los datos. La conforman uno o varios gestores de bases de datos cuya función consiste en gestionar el almacenamiento y acceso a los datos.

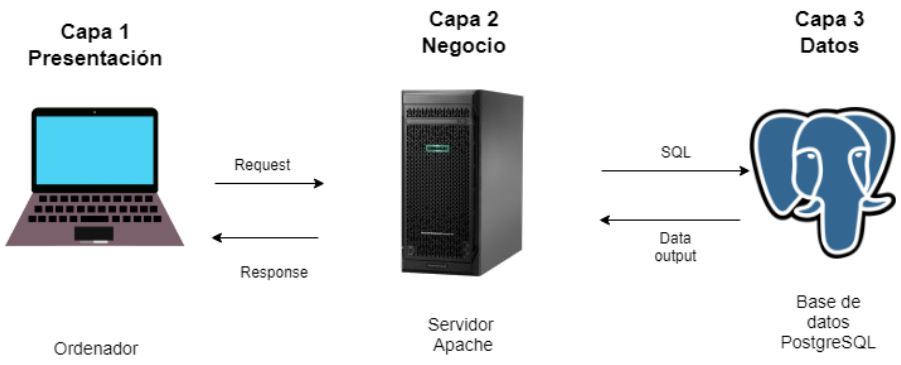


Ilustración 8.-Diagrama de la arquitectura 3 capas

6.1.1 Capa de presentación

La capa de presentación es la más visible de cara al usuario. Por tanto, es en esta capa donde se le dará la posibilidad al usuario de emplear las funcionalidades de este proyecto. Como se trata de una aplicación web, los lenguajes empleados, tal y como se indicaba anteriormente, han sido *HTML*, *JavaScript* y *CSS*. El uso de *Bootstrap* ha facilitado enormemente el diseño de la página web. Asimismo, para la representación de los mapas se ha empleado *Leaflet*.

6.1.1.1 Diseño estándar de la página

Las páginas que conforman la página web han sido desarrolladas en ficheros *PHP*, con el fin de aprovechar al máximo las prestaciones que este lenguaje nos brinda. Como la estructura de la página será siempre la misma, se han creado funciones con los componentes comunes a todas las páginas y así poder invocarlas cuando sea necesario.

La estructura de las páginas es muy sencilla. Arriba se muestra un menú donde se puede acceder a las diferentes páginas que dispone la web, así como el inicio de sesión para el usuario administrador. Luego, se presenta el cuerpo de la página, donde se muestra el mapa y/o la información que se desee mostrar o visualizar. Finalmente se puede ver un pie página con información sobre el sitio web.

A continuación se muestra un ejemplo de cómo se hace la llamada a la función programada en *PHP* que devuelve el contenido *HTML* a mostrar. En la primera imagen se puede ver la llamada a la función y en la segunda lo que esta función devuelve.

```
        </div>
        <?php
            OpacidadMapa::rangoOpacidad();
            VistaEnrutamiento::enrutamiento("mapa");
        ?>
```

Ilustración 9.- Llamada a la función que devuelve código HTML

```
<?php

    class OpacidadMapa{
        public static function rangoOpacidad(){
            echo '<div id="accordion">
                    <div class="card">
                        <div class="card-header" id="headingTwo">
                            <h5 class="mb-0">
                                <a class="btn btn-link" data-toggle="collapse" data-target="#collapseTwo" aria-expanded="false" aria-controls="collapseTwo">
                                    Transparencia del mapa &darr;
                                </a>
                            </h5>
                        </div>
                        <div id="collapseTwo" class="collapse" aria-labelledby="headingTwo" data-parent="#accordion">
                            <div class="card-body">
                                <div id="rango-opacidad">
                                    <label for="opacidad-topografico">Transparencia del mapa topográfico </label><br>
                                    <input type="range" class="custom-range" id="opacidad-topografico" oninput="setOpacidadTopografico()"><br>
                                    <label for="opacidad-lidar">Transparencia del modelo de terreno </label><br>
                                    <input type="range" class="custom-range" id="opacidad-lidar" oninput="setOpacidadLidar()"><br>
                                    <label for="opacidad-callejero">Transparencia del callejero </label><br>
                                    <input type="range" class="custom-range" id="opacidad-callejero" oninput="setOpacidadCallejero()"><br>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>';
        }
    }
```

Ilustración 10.- Función que devuelve código HTML

6.1.1.2 Vinculación de *Leaflet* con *HTML*

Constantemente en la aplicación se muestran mapas, con el fin de que el usuario pueda interactuar con él. La herramienta empleada para visualizar e interaccionar con los mapas es *Leaflet*. Por tanto, en este apartado se explicará cómo se ha combinado *Leaflet* con *HTML*.

En primer lugar, se crea una sección * * y le asignamos un *id*. En este espacio que hemos reservado dentro de la página será donde *Leaflet* construye el mapa.

```
<div id="mapa" class="mapa"></div>
```

Ilustración 11.- Sección donde se incluye el mapa

Seguidamente, se crea una función *JavaScript* donde se realizará la vinculación de la sección reservada en la página *HTML* con el mapa. Para ello, se emplea una función de *Leaflet* llamada *L.map*, a la cual se le pasa el *id* de la sección creada y se le asignan una serie de parámetros de configuración, entre los que destacan las coordenadas del centro del mapa, el zoom inicial, así como las capas que tendrá inicialmente el mapa.

```
mapa = L.map('mapa',{ center: [28.0, -15.6], zoom: 10,layers:[orto,redSendero,infoRed,puntosPrincipales]});
```

Ilustración 12.- Configuración del mapa

De la ilustración anterior hay que destacar que la primera capa que se le asigna al mapa es la *ortofoto* de la isla de Gran Canaria.

Finalmente, para que se pueda visualizar el mapa en la página, se crea una función *JavaScript* donde se ejecuten todas estas funciones que se han comentado. Esta nueva función es asignada a un evento (*onload*) que se dispara cuando la página ha cargado todo su contenido.

```
<body onload="cargaMapa()">
    <div class="container">
        <?php
            Header::barraMenu();
        ?>
        <div class="main-wrapper">
            <div id="seccionFormulario">
                <div id="alerta"></div>
```

Ilustración 13.- Ejecución de la función que carga el mapa.

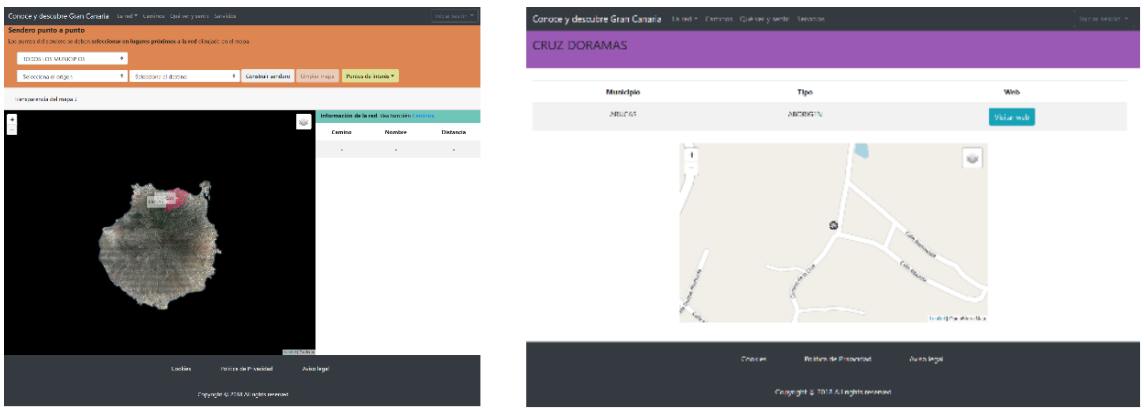


Ilustración 14.- Resultado de la vinculación de Leaflet con HTML

6.1.1.3 Manejo de los eventos

Otro de los aspectos interesantes de comentar en este apartado es el manejo de los eventos. Tanto el mapa como los componentes *HTML* de las diferentes páginas responden a eventos.

Los más sencillos de explicar son los eventos asociados a componentes *HTML,* como los botones o los *select* de los formularios. Estos componentes pueden capturar eventos, y a este evento se le asocia una función *JavaScript* que se encarga de realizar el procedimiento pertinente. Los eventos más frecuentes en este tipo de componente son *onclick* y *onchange*.

```
<div id="formularioBusqueda">
    <form name="busquedaRuta" class="form-inline" method="POST" id="formularioBusquedaRuta">
        <select name="municipio" id="input-municipio" class="custom-select" onchange="buscaMunicipioFormulario(true)">
            <option value='TODOS LOS MUNICIPIOS'>TODOS LOS MUNICIPIOS</option>
        <?php
```

Ilustración 15.- Ejemplo de manejo y captura de eventos con HTML y JavaScript

Sin embargo, para capturar los eventos en el mapa hay que realizar algo parecido. *Leaflet* permite asociar una serie de eventos, no solo al propio mapa, sino a las capas que se cargan sobre él. Los eventos usados para lograr la interacción con el mapa han sido los de *zoom*, *click* y *mousemove*. Es importante decir que *Leaflet* tiene su propia gestión de los eventos, por lo que para capturar un evento con esta herramienta es necesario emplear la función *on* e indicar en ella el evento que se desea capturar.

```
function eventosRed(){
    infoRed.on('click', function(event){
```

Ilustración 16.- Captura de eventos con Leaflet

6.1.2 Capa de negocio

Como bien se adelantaba anteriormente, la capa de negocio de esta aplicación tiene el objetivo de interactuar con la base de datos y de ordenar la información solicitada para poder representarla. En alguna ocasión se hace alguna orden de *UPDATE*, pero por lo general se realizan consultas con *SELECT*, con lo cual, en la mayoría de los casos se trata de pedir información y ordenarla para su correcta representación, ya sea en el mapa o en la propia página *HTML*.

Por ello, todo lo relativo a esta capa se hará *PHP*, que es el lenguaje empleado para la programación del lado del servidor.

6.1.2.1 GeoJSON

Antes de comenzar a detallar como se establece la conexión con la base de datos, así como los controladores que interactúan con ella, sería conveniente explicar que es el formato *GeoJSON*, pues habrá consultas que tendrán como salida información geográfica. Aunque se explicará más adelante, se parte de un conjunto de puntos y caminos que conforman una red, así como de cierta información vinculada a ellos. Para poder representar y obtener los puntos

y las líneas de los caminos, se ha hace necesario usar el estándar *GeoJSON* que a continuación se detallará.

GeoJSON es un formato estándar, que se puede usar libremente, creado no solo pare representar elementos geográficos sencillos, sino también sus atributos no espaciales. Este formato está basado en *JavaScriptObjectNotation*. En nuestro caso, se han representado *LineString* (línea), *MultiPoint* (conjunto de puntos) y *Point* (punto).

Para comprender mejor este formato, se plantea el siguiente ejemplo donde se explica de forma más clara las partes de un *GeoJSON*.

```
{"type": "FeatureCollection", "name": "prueba","crs": { "type": "name", "properties": {
"name": "urn:ogc:def:crs:OGC:1.3:CRS84" } } (1) ,

"features": [

        { "type": "Feature", "properties": { "id": 1, "nombre": "Cruz_1",
"Anno_Construccion": 1875 }(2), "geometry": { "type": "Point", "coordinates": [ -
15.524104267764388, 28.036224329847339 ](3) } },

        { "type": "Feature", "properties": { "id": 2, "nombre": "Cruz_2",
"Anno_Construccion": 1866 }(2), "geometry": { "type": "Point", "coordinates": [ -
15.483697463104081, 28.045827740820414 ](3) } },

        { "type": "Feature", "properties": { "id": 3, "nombre": "Cruz_3",
"Anno_Construccion": 1923 }(2), "geometry": { "type": "Point", "coordinates": [ -
15.508340178053864, 28.021366222304088 ](3) } },

        { "type": "Feature", "properties": { "id": 4, "nombre": "Cruz_4",
"Anno_Construccion": 1945 }(2), "geometry": { "type": "Point", "coordinates": [ -
15.574476876264677, 27.994549150152853 ](3) } },

        { "type": "Feature", "properties": { "id": 5, "nombre": "Cruz_5",
"Anno_Construccion": 1908 }(2), "geometry": { "type": "Point", "coordinates": [ -
15.548203393413806, 28.019735454402998 ](3) } }

        ]

}
```

Ilustración 17.- Ejemplo de formato GeoJSON

Se trata de la representación de un conjunto de puntos ubicados en la isla de Gran Canaria. En primer lugar, el primer elemento y al comienzo de la estructura del *GeoJSON* se debe asignar el sistema de referencia en el cual se encontrarán las coordenadas que se quieren representar (1). Seguidamente, se definen los objetos, especificando primero las propiedades de cada uno (2), y luego, la geometría, que son las coordenadas de este objeto (3). Es importante señalar que antes de especificar las coordenadas hay que indicar el tipo de elemento geográfico que corresponde, en este caso un punto (*Point*).

La estructura de un *GeoJSON* es igual que la del formato *JSON*, la única salvedad es que con *GeoJSON* aparece el elemento *geometry* para indicar las coordenadas, tal y como se explicaba arriba.

6.1.2.2 Conexión con la base de datos

Se ha creado en el proyecto un fichero donde se encuentran todas las funciones que gestionan las conexiones con la base de datos, así como la ejecución de las consultas.

```php
class Persistencia{
    private static $conn=null;

    public static function getConexion(){
        $host='localhost';

        $port='5432';
        $dbname='Conoce_y_descubre_Gran_Canaria';

        $user='nestor';
        $pass='nestortft';
        if(self::$conn===null){
            self::$conn= pg_connect("host=$host port=$port dbname=$dbname user=$user password=$pass");

        }
        return self::$conn;
    }

    public static function cerrar(){
        pg_close(self::$conn);
    }
```

Ilustración 18.- Conexión con la base de datos

Como se puede ver en la imagen, la conexión se realiza a la base de datos que se encuentra en *localhost* y se accede a ella por el puerto 5432. Se especifica el nombre de la base de datos, así como el nombre de usuario y contraseña. Para establecer la conexión se usa *pg_connect* y para cerrarla *pg_close*, que forman parte de un conjunto de funciones que hay que descargar e instalar en el servidor y que posibilitan la interacción con la bases de datos *PostgreSQL* al usar *PHP* como lenguaje del lado del servidor.

6.1.2.3 Ejecución de comandos y consultas

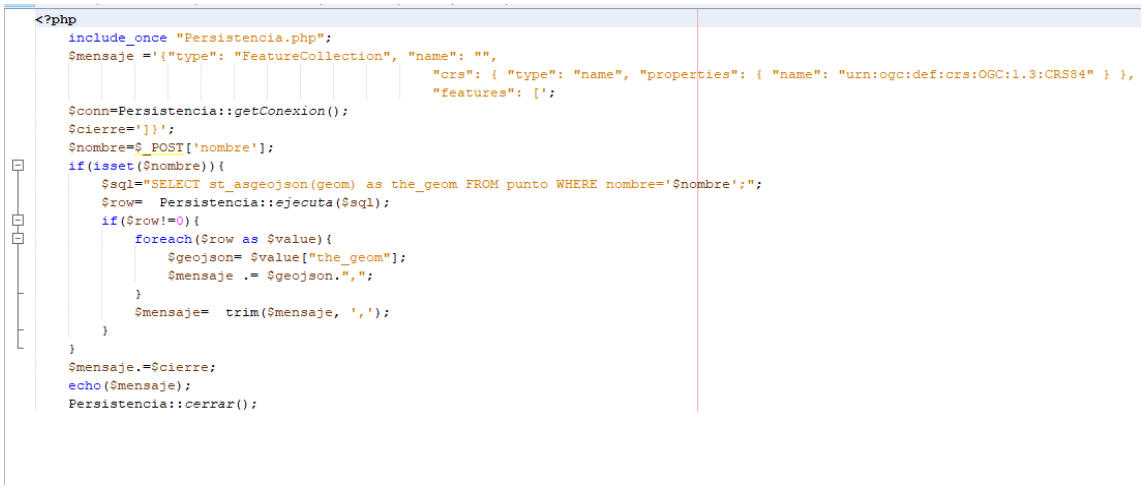
La ejecución de los comandos y de las consultas son lanzadas desde una petición que se realiza mediante *AJAX*. La petición tiene tres pasos:

1) Creación de un objeto *XMLHttpRequest* para realizar la petición.
2) Sobre este objeto se invoca la función *open*, que se encarga de establecer la petición. A esta función se le pasa por parámetro el método con el que se efectúa la petición (*GET* o *POST*), la *URL* y si la petición es síncrona o asíncrona.
3) Luego, se realiza la petición con el método *send*. Si la petición se realiza con el método *POST*, se le puede pasar a esta función los parámetros de la petición.

En el desarrollo de esta aplicación, las peticiones usarán el método *POST* y se harán peticiones síncronas.

```php
function consultaAjax(fichero, data){
    var ajax= new XMLHttpRequest();
    ajax.open("post",fichero,false);
    ajax.setRequestHeader("Content-type","application/x-www-form-urlencoded");
    if(data!=null){
        ajax.send(data);
    }else{
        ajax.send();
    }

    return ajax.responseText;
}
```

Ilustración 19.- Función que lanza peticiones AJAX



```php
<?php
    include_once "Persistencia.php";
    $mensaje ='{"type": "FeatureCollection", "name": "",
                                             "crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" } },
                                             "features": [';
    $conn=Persistencia::getConexion();
    $cierre=']}';
    $nombre=$_POST['nombre'];
    if(isset($nombre)){
        $sql="SELECT st_asgeojson(geom) as the_geom FROM punto WHERE nombre='$nombre';";
        $row=  Persistencia::ejecuta($sql);
        if($row!=0){
            foreach($row as $value){
                $geojson= $value["the_geom"];
                $mensaje .= $geojson.",";
            }
            $mensaje=  trim($mensaje, ',');
        }
    }
    $mensaje.=$cierre;
    echo($mensaje);
    Persistencia::cerrar();
```

Ilustración 20.- Ejemplo de ficheros que ejecuta consultas u órdenes

Primero se reciben los parámetros, luego se ejecuta la consulta o el comando. Si se trata de una consulta, la información se estructura siguiendo el formato *GeoJSON* o *JSON* según interese. En este caso en concreto se organizan los datos siguiendo un formato *GeoJSON*.

Las consultas se ejecutan en la función que se muestra a continuación. Para ello se hace uso de la función *pg_query*, que es otra de las funciones de la *API* que interacciona con bases de datos *PostgreSQL* desde *PHP*.

```php
public static function ejecuta($sql){
    $result = pg_query(Persistencia::getConexion(),$sql);
    if(pg_num_rows($result)>0){
        $row= pg_fetch_all($result);
        return $row;
    }else{
        return 0;
    }
}
```

Ilustración 21.- Función para ejecutar consultas

6.1.3 Capa de datos

Antes de comenzar a detallar todo lo relacionado con esta capa, es importante destacar que esta parte del proyecto se ha realizado conjuntamente con Alejandro Herrera, otro compañero del grado.

6.1.3.1 Bases de datos espaciales

Una base de datos espacial no deja de ser una base de datos relacional que incorpora datos espaciales. La única salvedad es que presenta una complejidad añadida que requiere de prácticas diferentes que permitan utilizar la base de datos de la misma manera que cuando se emplea con tipos de datos habituales [7]. Esta complejidad radica en que ahora hay que pensar en que una determinada información geográfica, referenciada con coordenadas, que tendrá asociada la información que le representa.

Por ejemplo, imaginemos que queremos realizar una capa geográfica de la Escuela de Ingeniería Informática. Para ello, tendremos una tabla donde estará almacenado, por un lado, el campo que guarda la componente geográfica, que en este caso será el polígono que representa la superficie de la escuela, y por otro lado, los diferentes campos de la tabla con la información asociada a dicha componente geográfica, como pueden ser las titulaciones que se imparten, el número de alumnos que estudian en la escuela, etc.

Además, las bases de datos especiales han experimentado una interesante evolución debido principalmente a la importancia que han adquirido los *SIG* en los últimos años, usuarios más destacados de este tipo de bases de datos [7].

Por tanto, una base de datos espacial permite la ejecución de diferentes funciones y procedimientos específicos que permitan una gestión eficiente (índices de la base de datos) y una correcta manipulación de la información georreferenciada. El gestor más conocido que trabaja con este tipo de bases de datos es *PostgreSQL*, comentado anteriormente, que con su extensión *PostGIS*, dota a la base de datos de un numeroso abanico de funciones que permiten hacer operaciones propias del campo de la *geomática*, como por ejemplo la unión de poli líneas o polígonos.

6.1.3.2 Modelo entidad-relación

El modelo entidad-relación es un diagrama que representa las entidades y las relaciones que existen entre ellas en un sistema informático. Para comprender mejor la realidad que intentamos representar en este proyecto es necesario realizar un diagrama de estas características, para así, lograr una estructura en la base de datos que permita una buena interpretación de la información que ella contiene.

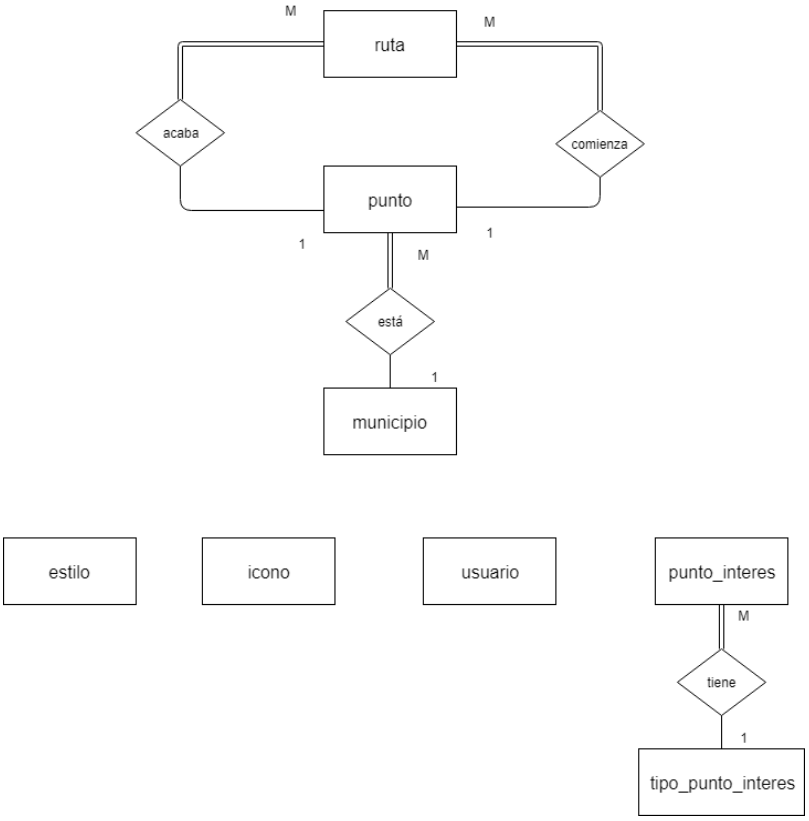


Ilustración 22.- Modelo entidad-relación

Si se observa el modelo entidad-relación, se puede ver que el centro lo ocupan las entidades *ruta* y *punto*. No cabe la menor duda de que la entidad ruta es la más relevante, pues en esta tabla es donde se encuentran todas las rutas o caminos que Salvador nos proporcionó, así como toda la información vinculada a cada una.

La relación entre ruta y punto se debe a que cada *ruta* tiene un punto de origen y un punto final. Estos puntos también fueron proporcionados por Salvador, a los que él denomina como *puntos de enlace de la red*. Por este motivo se han creado dos relaciones diferentes: una que indica el punto que da comienzo al camino y otro que pone fin al mismo. Respecto a la entidad *municipio*, cada punto debe estar asociado a un municipio.

También, en la parte inferior se pueden apreciar una serie de tablas que no tienen relación, salvo la de *punto_interes*. Esto se debe a que son entidades creadas para almacenar información que permiten funcionalidades ajenas a todo lo relacionado con el enrutamiento de los caminos.

La tabla *usuario* registra los usuarios que pueden tener acceso al sistema como administradores del mismo, mientras que las tablas *estilo* e *icono* simplemente almacenan la información relativa al diseño de los componentes del mapa (iconos de los puntos de interés, grosor, color, opacidad y anchura de las líneas que se dibujan sobre el mapa, entre otros).

La tabla *punto_interes*, alberga no solo los lugares interesantes para visitar en cada municipio, sino también los servicios que se prestan en cada uno de ellos (servicios de urgencias, policía, aparcamiento, etc.). La relación con la tabla *tipo_punto_interes* se debe a que cada punto de interés pertenece a un tipo determinado de puntos. Por tanto, es en esta tabla de

tipo_punto_interes donde hay almacenada una lista controlada de los distintos tipos de puntos que debe representar la aplicación.

6.1.3.3 Atributos de las tablas y sus relaciones

Como la algunas tablas tienen muchos atributos, se prefirió representar y detallar los atributos de cada tabla por separado para que la *Ilustración 22* pudiera quedar lo más legible posible.

Antes de comenzar a detallar los campos de cada tabla, sería conveniente explicar el tipo de dato *geometry*. El tipo de dato *geometry* es aquel que permite cargar en una tabla datos espaciales como puntos, líneas o polígonos. Este tipo de datos además permite que una misma tabla se puedan cargar diferentes elementos geométricos, es decir, una tabla puede tener almacenada en una fila una línea, y en la siguiente fila, un polígono.

Una vez aclarado este detalle, se puede seguir con la descripción de los campos:

Tabla ruta

Nombre	Tipo de dato	Clave primaria	Not null	Descripción [3]
id	integer	Sí	Sí	
the_geom	geometry	No	Sí	
nombre	character varying	No	Sí	
numero	integer	No	Sí	
clave	character varying	No	Sí	
referencia_promotor	character varying	No	No	
circular	boolean	No	Sí	Determina si un camino es circular.
enlace	boolean	No	Sí	Determina si un camino es un enlace entre dos o más caminos.
datos				Descripción del camino.
punto_inicial	integer	No	Sí	
punto_final	integer	No	Sí	
recorrido	character varying	No	No	Otros puntos del camino
kilometros	numeric	No	Sí	Distancia en kilómetros del camino.

[3] Solo se detallarán aquellos campos que creen confusión.

sendero	numeric	No	No	Número de kilómetros que tiene el camino en sendero.
pista	numeric	No	No	Número de kilómetros que tiene el camino en asfalto.
asfalto	numeric	No	No	Número de kilómetros que tiene el camino en asfalto.
altura_inicio	numeric	No	No	
altura_máxima	numeric	No	No	
altura_minima	numeric	No	No	
altura_final	numeric	No	No	
desnivel	numeric	No	No	
ascenso	numeric	No	No	
pendiente_media	numeric	No	No	
media_altura	numeric	No	No	
nivel	integer	No	No	
calificacion_camino	integer	No	No	
espacio_natural_protegido	Integer	No	No	Número de espacio natural protegido.
punto_biosfera	boolean	No	No	
lugar_interes_comunitario				
zona_proteccion_aves				
avistamiento_aves				
bien_interes_cultural				
lugar_arqueologico				
bien_etnografico				
baño				
zona_acampada				
espacio_alojativo				
area_recreativa				
espacio_uso_publico				
jardin_parque				
source	integer	No	No	Campo necesario para el enrutamiento.
target	integer	No	No	
variante	boolean	No	Sí	Determina si el camino es variante de otro.

horario_estimado	character varying	No	Sí	Horario en el que se puede hacer el camino.
tiempo	numeric	No	Sí	Tiempo estimado en minutos en el que puede realizar el camino.
tiempop	numeric	No	No	Tiempo estimado en minutos en el que puede realizar el camino en sentido contrario.

Tabla 4.- Campos de la tabla ruta

Tabla punto

Nombre	Tipo de dato	Clave primaria	Not null	Descripción
id	integer	Sí	Sí	
geom	geometry	No	Sí	
nombre	character varying	No	Sí	
descripcion	character varying	No	No	Enlace web del sitio.
zona	character varying	No	Sí	Zona de la isla a la que pertenece el punto.
municipio	integer	No	Sí	
linea_guagua	character varying	No	Sí	Líneas de guagua que pasan por el punto.
cercania_nucleo_urbano	integer	No	Sí	Numeración que indica la proximidad al núcleo urbano.

Tabla 5.- Campos de la tabla punto

Tabla municipio

Nombre	Tipo de dato	Clave primaria	Not null	Descripción
id	integer	Sí	Sí	
geom	geometry	No	Sí	
nombre	character varying	No	Sí	

Tabla 6.- Campos de la tabla municipio

Tabla punto_interes

Nombre	Tipo de dato	Clave primaria	Not null	Descripción
id	integer	Sí	Sí	
geom	geometry	No	Sí	
nombre	character varying	No	Sí	
descripcion	character varying	No	No	Enlace web del sitio.
municipio	integer	No	Sí	
tipo	Integer	No	Sí	

Tabla 7.- Campos de la tabla punto_interes

Tabla tipo_punto_interes

Nombre	Tipo de dato	Clave primaria	Not null	Descripción
id	integer	Sí	Sí	
tipo	character varying	No	Sí	

Tabla 8.- Campos de la tabla tipo_punto_interes

Tabla usuario

Nombre	Tipo de dato	Clave primaria	Not null	Descripción
id	integer	Sí	Sí	
nombre	character varying	No	Sí	
contraseña	character varying	No	Sí	
tipo	integer	No	Sí	Tipo de usuario.

Tabla 9.- Campos de la tabla usuario

Tabla estilo

Nombre	Tipo de dato	Clave primaria	Not null	Descripción
id	integer	Sí	Sí	
nombre	character varying	No	Sí	

color	character varying	No	Sí	Color de la línea a dibujar en el mapa.
anchura	numeric	No	Sí	Anchura de la línea a dibujar en el mapa.
opacidad	numeric	No	Sí	Opacidad de la línea a dibujar en el mapa.

Tabla 10.- Campos de la tabla estilo

Tabla icono

Nombre	Tipo de dato	Clave primaria	Not null	Descripción
id	integer	Sí	Sí	
url	character varying	No	Sí	Ruta del servidor donde se encuentra el icono.
nombre	character varying	No	Sí	
sizex	numeric	No	Sí	Ancho del icono.
sizey	numeric	No	Sí	Alto del icono.
trasladox	numeric	No	Sí	Traslado vertical con respecto al puntero del ratón.
trasladoy	numeric	No	Sí	Traslado horizontal con respecto al puntero del ratón.

Tabla 11.- Campos de la tabla icono

Relaciones entre las tablas

Asimismo, siguiendo el modelo entidad-relación, se dan las siguientes relaciones:

- Los campos *punto_inicial* y *punto_final* hacen referencia al campo *id* de la tabla *punto*.
- El campo *municipio* hace referencia al campo *id* de la tabla *municipio*.
- El campo *tipo* hace referencia al campo *id* de la tabla *tipo_punto_interes*.

6.2 Interfaz de usuario

A continuación, se presentará el diseño de las diferentes vistas que tiene la aplicación web. Se mostrarán en un diagrama de flujo para que se pueda apreciar mejor la navegabilidad de la interfaz de usuario.

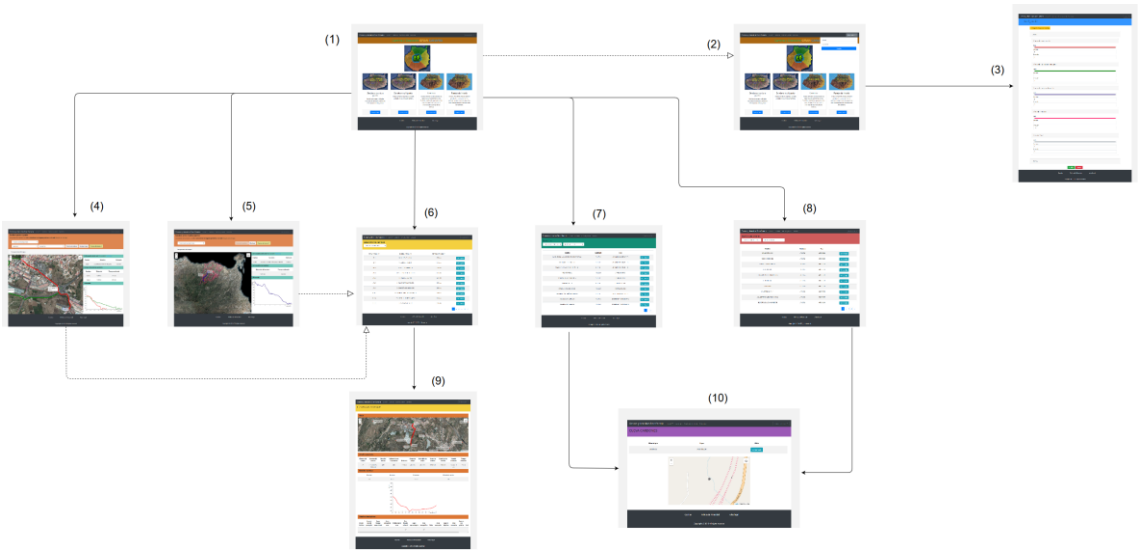


Ilustración 23.- Diagrama de flujo

De la página inicial (1) parten todas las vistas de la aplicación web. Asimismo, en la página inicial, el usuario administrador puede iniciar sesión (2) y acceder a la página de configuración (3).

El resto de páginas muestran las diferentes funcionalidades que ofrece este proyecto:

- Creación de senderos entre dos puntos (4).
- Creación de senderos multipunto (5).
- Visualización de los caminos y sus detalles (6) y (9).
- Visualización de los servicios y de sus detalles (7) y (10).
- Visualización de los puntos de interés y sus detalles (8) y 10).

Como se puede ver, también desde las vistas 4 y 5 se puede acceder a la visualización de los caminos (6), así como a los detalles de los mismos (9).

7. Desarrollo

En este aparatado de la memoria se detallarán los diferentes aspectos relacionados con el desarrollo del proyecto. En lo referente a la metodología a seguir, se fueron realizando las diferentes tareas que los tutores iban proponiendo y se tenían reuniones con ellos para revisar y probar las funcionalidades desarrolladas. En lo relativo al desarrollo, este se detallará en los siguientes apartados. La planificación prevista inicialmente se encuentra reflejada en la solicitud del proyecto y se ha cumplido con éxito.

7.1 Instalación y uso de las tecnologías

Antes de comenzar a desarrollar el proyecto, se realizó con otro compañero, Alejandro Herrera, un estudio de las tecnologías que se podrían usar. Este análisis se ha detallado en el apartado *4.2 Análisis de las tecnologías software existente* del capítulo *4. Recursos utilizados* de esta memoria.

En este apartado solo vamos a detallar los aspectos relacionados con la instalación y el uso de estas tecnologías. Los ejecutables de *NetBeans*, *QGis, MAMP, PostgreSQL* y sus extensiones *Postgis* y *pgRouting*, así como *pgAdmin 4,* fueron descargados de las páginas oficiales correspondientes e instaladas en el ordenador. Es importante destacar que *PostgreSQL*, *PostGIS* y *pgRouting* se pueden descargar conjuntamente en un paquete que ofrecen los desarrolladores.

Por otro lado, las bibliotecas *Leaflet*, *Bootstrap* y *d3.js* son usados mediante su enlace *CDN*. Este enlace permite acceder a las funciones de estas tecnologías, y así poder usarlas con total normalidad sin tener que tener los ficheros en el proyecto.

7.2 Preparación de la base de datos

Antes de comenzar a desarrollar el proyecto, se procedió a configurar la base de datos y a la creación de la red de senderos en la misma. El primer paso que se llevó a cabo fue la inserción de los caminos en la tabla *ruta*, así como todos los datos vinculados a ellos. Para insertarlos se empleó el programa *QGis*. Mediante la interfaz gráfica de *QGis* (ver *Ilustración 24*) se insertaron los caminos a partir de los ficheros *KML* que se nos proporcionó. Es importante destacar que los caminos proporcionados pertenecen, principalmente, a los municipios de Arucas y Firgas.

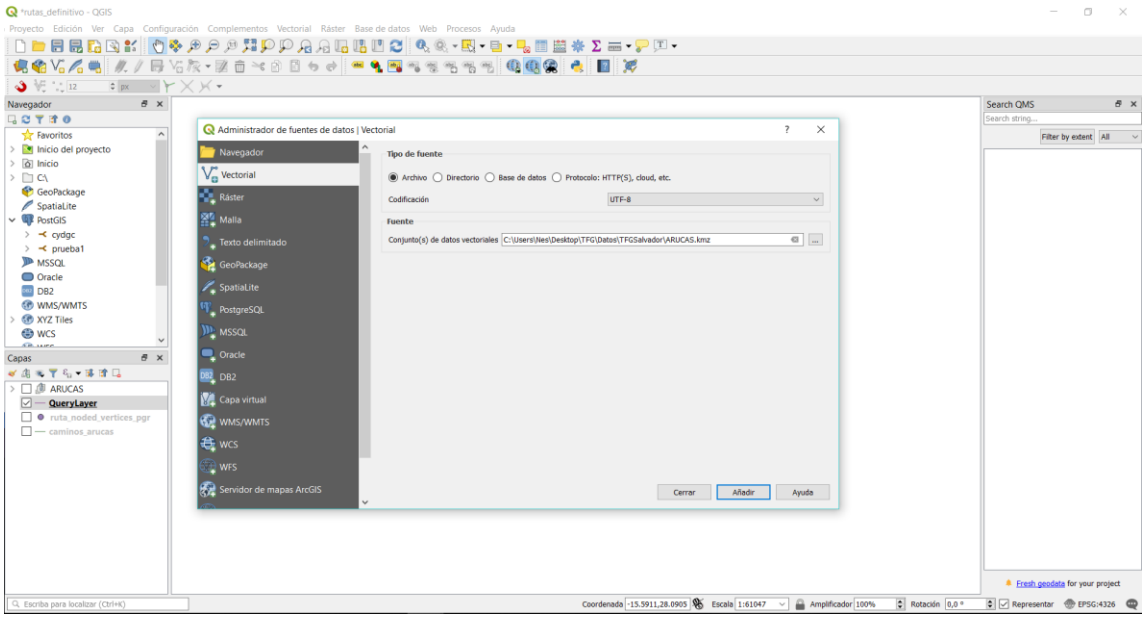


Ilustración 24.- Importación de los caminos en QGis

Una vez importados los caminos en *QGis*, se insertaron en la base de datos. Para ello, hubo que establecer una conexión entre *QGis* y la base de datos. En la imagen que se muestra a continuación se puede ver la conexión establecida, así como el procedimiento de inserción en la base de datos. En la ventana de configuración del programa, se puede ver una pestaña con las diferentes bases de datos. Lo que se despliega de *PostGIS* indica las conexiones disponibles con la base de datos. La conexión usada ha sido *cydgc*. En la ventana de la configuración aparece los parámetros de la inserción: nombre que tendrá la nueva tabla con los caminos, quién será la clave primaria, la columna con la geometría, sistemas de referencia, etc.

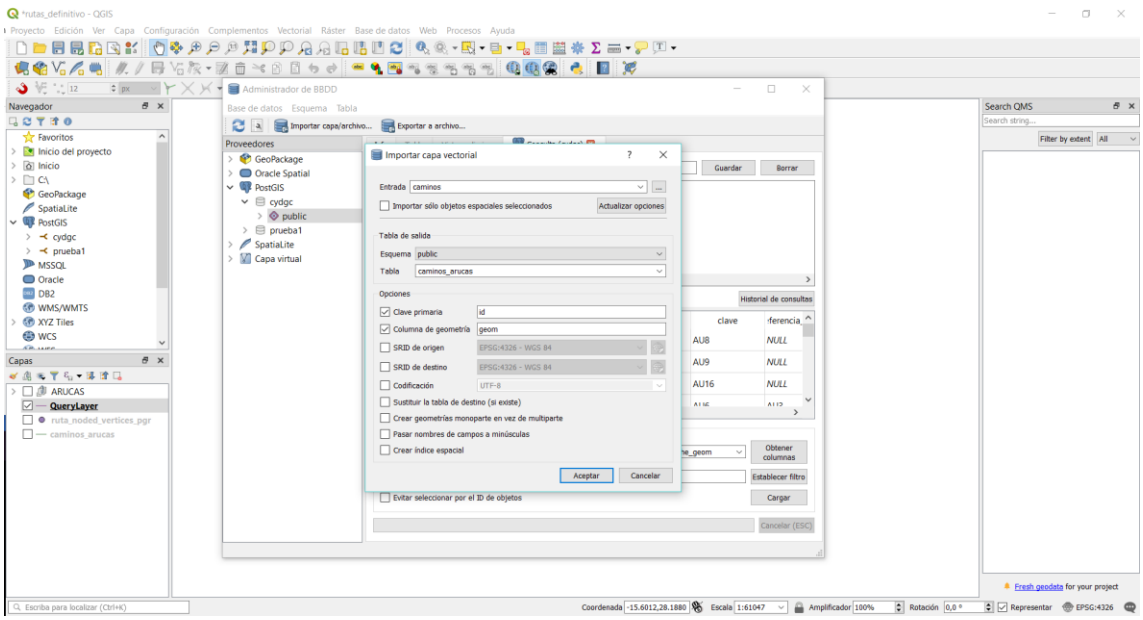


Ilustración 25.- Inserción de los caminos en la base de datos

Una vez insertados los caminos, se insertó toda la información vinculada a ellos, que se encuentra en un fichero *EXCEL*. Tras normalizar toda esa información en un fichero *CSV*, se insertó mediante la interfaz gráfica de *pgAdmin*, que permite la inserción de información en las tablas mediante ficheros *CSV*.

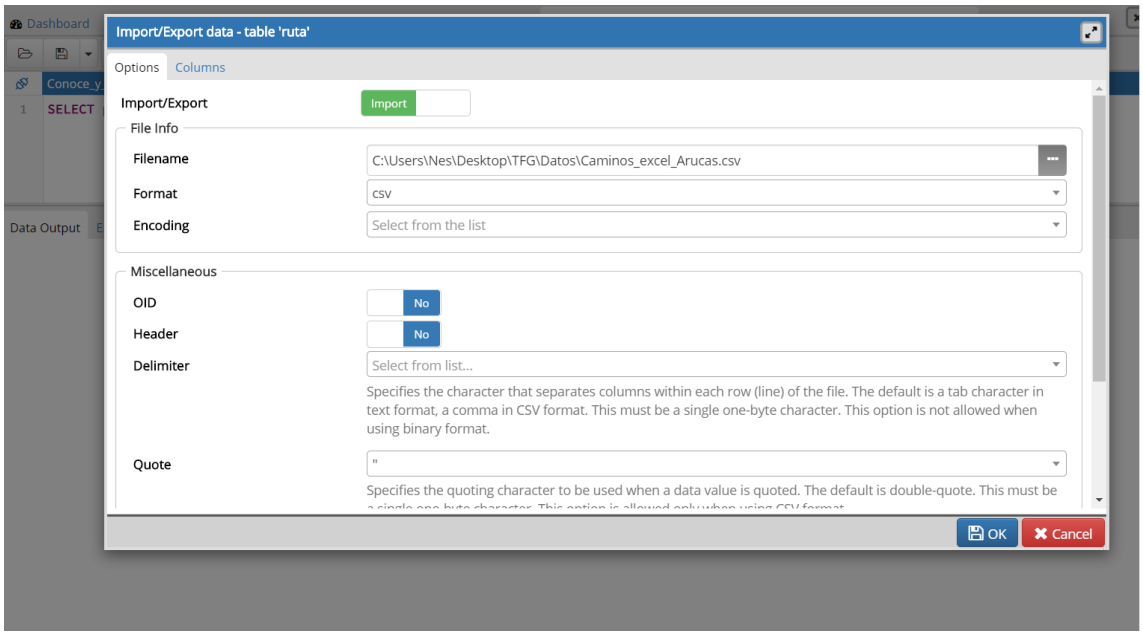


Ilustración 26.- Inserción de los datos asociados a los caminos desde un fichero CSV

```
1,271,AU1,,true,false,false,,1,1,,1.1,,,,,,,250,282,250,250,0,48,48,8,37,,,,,,,,true,true,true,,,,,,true
2,272,AU2,AU3,false,false,false,,1,2,,1,,,,,,,250,254,222,223,-27,19,47.5,21,,,,,,,,true,,true,,,,,,true
3,273,AU3,S12,false,false,false,,1,3,,2.7,,,,,,,250,271,177,191,-59,70,128,7,35,,,,,,,,true,true,true,,,,,true
```

Ilustración 27.- Aspecto del fichero CSV

Del mismo modo se insertaron los puntos de la red, así como los puntos de interés y los servicios. Cada uno de estos puntos tiene componente geográfica e información asociada. Cada punto se insertó en su tabla correspondiente.

Seguidamente, una vez estuvieran todos los datos insertados en sus tablas, se creó la red de senderos a partir de la información contenida en la tabla *ruta*. Para ello se emplearon dos funciones de *pgRouting*: *pgr_nodeNetwork* y *createTopology*.

En primer lugar se ejecuta la función *pgr_nodeNetwork*. Esta función divide cada intersección, *encontrada en la tabla ruta*, en una línea-segmento separada. Por tanto, crea una nueva tabla con las aristas de la red. La función se ejecuta en *pgAdmin* en una consulta, especificando el nombre de la tabla sobre la que se va a crear la red y una tolerancia para detectar nodos o tramos de la red no conectados para alargar y completar el tramo. La nueva tabla creada se denomina *ruta_noded* y tendrá los diferentes tramos de la red.

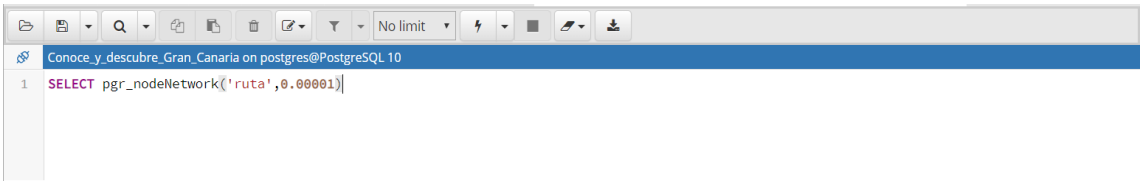


Ilustración 28.- Ejecución de la función pgr_nodeNetwork

Acto seguido se crea la topología con la función *pgr_createTopology*. Una topología es la forma con que está diseñada la red, sea en el plano físico o lógico. La función *pgr_createTopology* crea una topología y una nueva tabla con los vértices de la red. Al igual que con la función explicada anteriormente, esta se ejecuta en *pgAdmin* y se le pasa como parámetros la tabla *ruta_noded*, y de nuevo, una tolerancia. La nueva tabla creada con los vértices se denomina *ruta_noded_vertices_pgr*.

Ilustración 29.- Ejecución de la función pgr_createTopology

Es importante señalar que esta función necesita que la tabla que contiene los caminos, en nuestro caso, la tabla *ruta*, contenga dos campos llamados *source* y *target*. En cada fila de la tabla, estos campos no deberán tener información. Una vez ejecutada la función *pgr_nodeNetwork*, en la nueva tabla (*ruta_noded*) el campo *source* tendrá el *id* del vértice que inicia el tramo y *target* tendrá el id del vértice que finaliza el tramo.

Una vez creada la red y la topología, se puede visualizar en *QGis* cuál es el aspecto de la red con sus vértices.

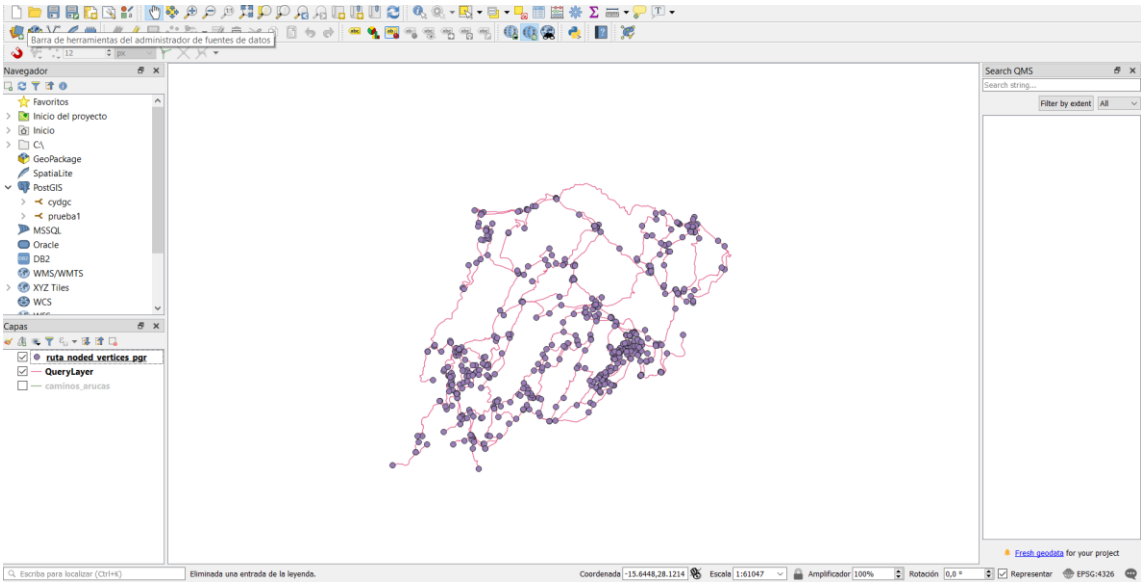


Ilustración 30.- Visualización de la red y sus vértices

Con el fin de poder obtener el camino más corto, se hace necesario obtener para cada tramo su distancia. Esta distancia será el peso de cada arista del grafo. Para ello, lo primero que se hace es crear una nueva columna en la tabla *ruta_noded* llamada *distancia*. El tipo de datos de esta columna será de tipo *float*.

Para obtener la distancia de cada tramo, se emplea la función *ST_Length*, que es una función de *PostGIS* que obtiene la distancia de una geometría. A esta función se le pasará como parámetro la geometría, con el sistema de referencia 4326, en coordenadas geográficas, debido a que las coordenadas geográficas proporcionan mayor precisión. Finalmente, se pasará el resultado a kilómetros, ya que en este caso la función *ST_Length* devuelve la distancia en metros (*Ilustración 25*).

Conoce_y_descubre_Gran_Canaria on postgres@PostgreSQL 10

```
1  ALTER TABLE ruta_noded ADD COLUMN distancia float;
2  UPDATE  ruta_noded SET distancia = ST_Length(ST_Transform(the_geom,4326)::geography) / 1000;
3
4
```

Ilustración 31.- Obtención de distancias

Con el objetivo de aclarar el procedimiento seguido, se exponen el siguiente diagrama de actividades con las acciones realizadas para crear la red.

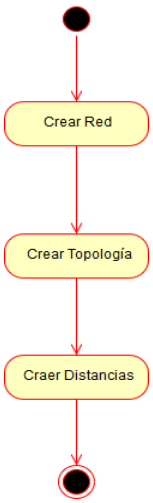


Ilustración 32.- Diagrama de actividades sobre la creación de la red

7.3 Consultas para el cálculo de la ruta óptima

7.3.1 Algoritmo de Dijkstra

Para poder obtener el camino más corto entre dos puntos se ha empleado una función de *pgRouting* que obtiene el camino más corto a partir de la red creada en la base de datos. Esta función se denomina *pgr_dijkstra*, y como su propio nombre indica, emplea el algoritmo de *Dijkstra* para obtener el camino más corto.

Este algoritmo consiste en obtener la ruta más corta desde un nodo inicial hasta cualquiera de los nodos del grafo, iterando hasta encontrar la ruta solución. El algoritmo emplea un grafo con pesos para poder obtener el camino más corto. Se trata de ir mirando que camino va tiendo un menor valor acumulado de los costes. El camino que tenga el menor valor acumulado será la ruta más óptima.

Con el objetivo de comprender mejor este algoritmo, se plante ahora un caso práctico:

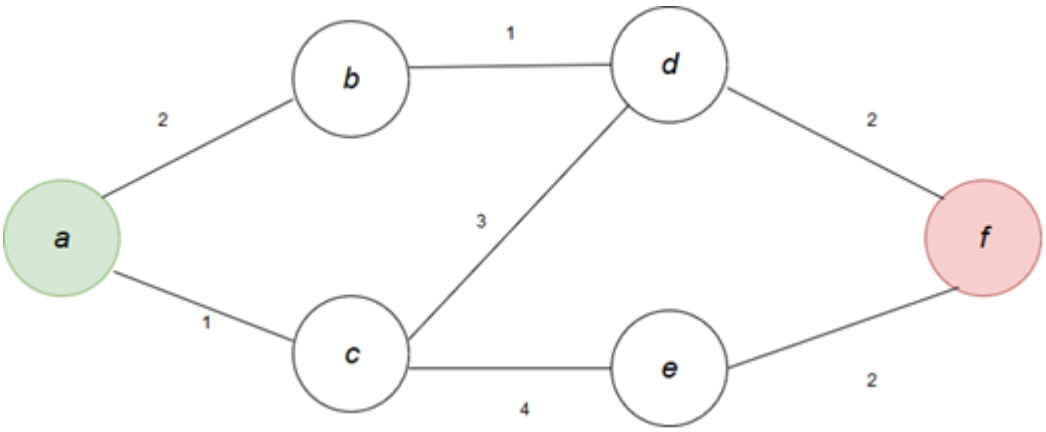


Ilustración 33.- Ejemplo de algoritmo de Dijkstra

Como la ruta parte del vértice *a*, lo marcamos como definitivo. Como es el primero, su coste acumulado es 0 y parte de él mismo (*a*).

Vértice	Iteración 1	Iteración 2	Iteración 3	Iteración 4	Iteración 5	Iteración 6
a	(0, *a*)	*	*	*	*	*
b	(2, *a*)					
c	(1, *a*)					
d						
e						
f						

Tabla 12.- Primera iteración del algoritmo de Dijkstra

Ahora, en la segunda iteración, escogemos del resto de vértices de la iteración 1 el que menor coste acumulado tenga, o sea, el vértice *c*. A partir de este se obtiene el coste acumulado de los nodos adyacentes y se marca el vértice de procedencia. Se marca el vértice *c* como definitivo.

Vértice	Iteración 1	Iteración 2	Iteración 3	Iteración 4	Iteración 5	Iteración 6
a	(0, *a*)	*	*	*	*	*
b	(2, *a*)	(2, *a*)				
c	(1, *a*)	(1, *a*)	*	*	*	*
d		(4, *c*)				
e		(5, *c*)				
f						

Tabla 13.- Segunda iteración del algoritmo de Dijkstra

Realizamos el mismo procedimiento. Se obtiene el vértice con menor peso acumulado y se calcula el peso acumulado de los vértices adyacentes. En este caso, se marca el vértice *b* como definitivo.

Vértice	Iteración 1	Iteración 2	Iteración 3	Iteración 4	Iteración 5	Iteración 6
a	(0, *a*)	*	*	*	*	*
b	(2, *a*)	(2, *a*)	(2, *a*)	*	*	*
c	(1, *a*)	(1, *a*)	*	*	*	*
d		(4, *c*)	(3, *b*)			
e		(5, *c*)	(5, *c*)			
f						

Tabla 14.- Cuarta iteración del algoritmo de Dijkstra

En esta iteración, como los vértices *e* y *f* tienen el mismo peso acumulado, se coge en primer lugar el vértice *e*, pero al obtener el peso acumulado de los vértices adyacentes, se obtiene un peso acumulado mayor que 5 (peso acumulado para llegar a *f* desde *d*) para el vértice f, con lo cual, no interesa. Se marca el vértice *e* como definitivo y se deja el peso acumulado para llegar a *f* como estaba.

Vértice	Iteración 1	Iteración 2	Iteración 3	Iteración 4	Iteración 5	Iteración 6
a	(0, *a*)	*	*	*	*	*
b	(2, *a*)	(2, *a*)	(2, *a*)	*	*	*
c	(1, *a*)	(1, *a*)	*	*	*	*
d		(4, *c*)	(3, *b*)	(3, *b*)	*	*
e		(5, *c*)	(5, *c*)	(5, *c*)	(5, *c*)	*
f				(5, *d*)	~~(7, *e*)~~ (5, *d*)	(5, *d*)

Tabla 15.- Quinta y sexta iteraciones del algoritmo de Dijsktra

Llegados a la iteración 6 ya se puede obtener la ruta más corta. Si se retrocede desde el vértice *f*, observando el vértice procedente, se puede deducir que la ruta más corta, ordenada inversamente es *f*, *d*, *b* y *a*, es decir, que la ruta mínima es *a*, *b*, *d* y *f* con una distancia de 5.

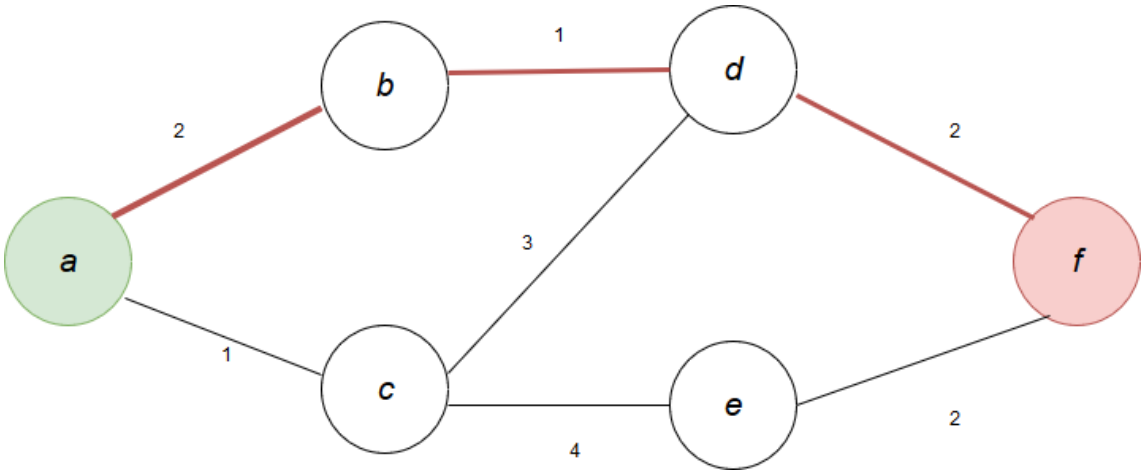


Ilustración 34.- Ejemplo de algoritmo de Dijkstra resuelto

7.3.2 Consulta para obtener la ruta más corta

Una vez entendido el funcionamiento del algoritmo de *Dijkstra* se puede comprender el funcionamiento de la función *pgr_dijkstra* de *pgRouting*. Esta función es importante, pues permite obtener el camino más corto entre dos vértices. A esta función se le van a pasar cuatro parámetros que a continuación se van a detallar:

- *sql*.- Consulta *sql* que obtiene todos los tramos (aristas del grafo) con su *id*, *source*, *target* y *cost* de la tabla *ruta_noded*.
  - *id*.- Identificador de la arista del grafo.
  - *source*.- Identificador del vértice inicial de la arista del grafo.
  - *target*.- Identificador del vértice final de la arista del grafo.
  - *cost*.- Distancia de la arista. Esta distancia es la que fue calculada tras crear la red y la topología.
- *source*.- Identificador del vértice de partida sobre el que se quiere calcular la ruta más corta.
- *target*.- Identificador del vértice final sobre el que se quiere calcular la ruta más corta.
- *directed*.- Este parámetro es para indicar si el grafo sigue direcciones.

Asimismo, esta función devuelve una tabla con seis campos:

- *seq*.- Se trata del número de iteraciones llevadas a cabo en el algoritmo.
- *path_seq*.- Indica la posición que tiene el tramo en la ruta resultante.
- *node*.- *id* del vértice visitado.
- *edge*.- *id* de la arista del grafo.
- *cost*.- Coste o peso que tiene la arista que conforma la ruta óptima.
- *agg_cost*.- Coste acumulado de la ruta óptima.

Conoce_y_descubre_Gran_Canaria on postgres@PostgreSQL 10

```
SELECT * FROM pgr_dijkstra('
                        SELECT id, source, target,
                            distancia as cost FROM ruta_noded',
                            1, 20, false)
```

Data Output | Explain | Messages | Query History

	seq integer	path_seq integer	node bigint	edge bigint	cost double precision	agg_cost double precision
1	1	1	1	29	0.35113290403823	0
2	2	2	12	30	0.505760371425682	0.35113290403823
3	3	3	13	31	0.434360534882061	0.856893275463912
4	4	4	14	32	0.307979514913655	1.29125381034597
5	5	5	15	202	0.743648612772708	1.59923332525963
6	6	6	132	203	0.231730413240635	2.34288193803234
7	7	7	133	204	0.211983509598062	2.57461235127297
8	8	8	112	440	0.122756593864685	2.78659586087103
9	9	9	274	439	0.311490606815338	2.90935245473572
10	10	10	273	438	0.109435531339096	3.22084306155106
11	11	11	272	437	1.056968325242	3.33027859289015
12	12	12	271	436	0.108694076755568	4.38724691813215
13	13	13	21	232	0.155711466593634	4.49594099488771
14	14	14	20	-1	0	4.65165246148135

Ilustración 35.- Ejemplo de ejecución de la función pgr_dijkstra

En la *Ilustración 35* se puede ver un ejemplo de ejecución de la función *pgr_dijkstra* de *pgRouting*. Se trata de un ejemplo en el que se quiere obtener la ruta más óptima entre los

vértices 1 y 20. Como bien se comentaba antes, esta función devuelve una tabla, y es por ello por lo que se necesita realizar una consulta para visualizar la información. En la pestaña *Data Output* del *pgAdmin* se puede visualizar el resultado.

Hasta ahora se ha detallado como obtener la ruta más óptima, pero para poder representarlo en el mapa necesitamos conocer sus coordenadas. Para ello, se ha elaborado la siguiente consulta, y que a continuación, se explicará su funcionamiento.

```
SELECT r.id, ST_AsGeoJson(r.the_geom) AS the_geom, cost
FROM pgr_dijkstra('
                SELECT id, source, target,
                distancia as cost FROM ruta_noded',
                $source, $target, false) AS d, (1)
     ruta_noded AS r
WHERE d.edge=r.id (2)
```

Ilustración 36.- Consulta para obtener la ruta óptima

Con el fin de hacer que la explicación sea lo más sencilla posible, se ha dividido la consulta en dos partes. El objetivo es explicar estas dos partes, pues son las más interesantes de esta consulta.

En la primera parte (1) se obtiene la tabla con los identificadores de los vértices y de las aristas del grafo que conforman la ruta óptima, tal y como se explicó anteriormente. Por otro lado, en la segunda parte (2), la condición *d.edge=r.id* filtra en la tabla solo aquellos tramos de la tabla *ruta_noded* que sean iguales a los que devuelve la función *pgr_dijkstra*. De esta manera se obtienen las coordenadas, y para que puedan ser dibujadas en el mapa, se hace uso de la función *ST_AsGeoJson*, que tiene como finalidad representar el campo con la geometría como un *GeoJson*. Es importante señalar que los parámetros *$source* y *$target* son los vértices sobre los que se quiere obtener la ruta óptima.

Esta consulta se encuentra en un fichero *PHP* que se encarga de enviársela a la base de datos, obtener su respuesta y enviar la información a las funciones de *JavaScript* que tratan las coordenadas y las representan en el mapa.

```php
<?php

  include_once "Persistencia.php";

   $mensaje ='{"type": "FeatureCollection", "name": "",
                                          "crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" } },
                                          "features": [';
   $conn=Persistencia::getConexion();
   $cierre=']}';
   $source=$_POST['source'];
   $target=$_POST['target'];
   $sql="";
   $coste=0;
   if(isset($source) && isset($target)){
       if(($source!=$target)){
           $sql="SELECT r.id, ST_AsGeoJson(r.the_geom) AS the_geom, cost
           FROM pgr_dijkstra('
                   SELECT id, source, target,
                       distancia as cost FROM ruta_noded',
                       $source, $target, false) AS d,
               ruta_noded AS r
               WHERE d.edge=r.id";
           $row=  Persistencia::ejecuta($sql);
           if($row!=0){
               foreach($row as $value){
                   $mensaje.='{ "type": "Feature", "properties": {';
                   $mensaje.='"id": "'.$value['id'].'" },';
                   $mensaje.=' "geometry": '.$value['the_geom'].'}';
                   $mensaje .= ",";
                   $coste+=$value['cost'];
               }
               $mensaje=  trim($mensaje, ',');
               $mensaje.='], "coste":'.$coste.'}';

           }

       }else{
           $mensaje.='], "coste":0}';
       }
   }

   echo($mensaje);
   Persistencia::cerrar();
```

Ilustración 37.- Fichero PHP con la consulta que obtiene la ruta óptima

7.3.3 Consulta para obtener un vértice cercano

Otra consulta necesaria para este proyecto es la que obtiene el vértice cercano a un punto. Se trata de, a partir de una coordenada, obtener el vértice de la red más cercano a él. La consulta tiene el siguiente aspecto:

```
SELECT v.id, st_AsGeoJson(v.the_geom) as the_geom

FROM ruta_noded_vertices_pgr AS v, ruta_noded AS e

WHERE e.source=v.id OR e.target=v.id (1)

ORDER BY v.the_geom <-> ST_SetSRID(ST_MakePoint($x,$y),4326)

LIMIT 1; (2)
```

Ilustración 38.- Consulta para obtener el vértice más cercano a un punto

Al igual que en la consulta explicada anteriormente, esta se ha divido en dos partes. La primera de ellas (1) es una condición que filtra el resultado si el id de la tabla *ruta_noded_vertices_pgr* es igual al *source* o *target* de los tramos contenidos en la tabla ruta_noded. En segundo lugar (2), se ordena el resultado según la proximidad al punto pasado como parámetro, limitando a un solo registro la respuesta de la consulta. Para poder hacer la comparativa y saber si un punto está cercano al vértice, se usa el operador <->. También, se emplea la función *ST_MakePoint* para construir un punto mediante la longitud y la latitud de una coordenada, así como la función *ST_SetSRID* que asigna el sistema de referencia, en este

caso el 4326, al punto creado. Se debe mencionar que *$x* e *$y* son los parámetros con la longitud y la latitud del punto.

Esta consulta, al igual que la consulta anterior, se encuentra en un fichero *PHP*, tal y como se puede observar en *la Ilustración 33*.

```php
<?php
    include_once "Persistencia.php";

    $mensaje ='{"type": "FeatureCollection", "name": "",
                                            "crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" } },
                                            "features": [';
    $conn=Persistencia::getConexion();
    $x=$_POST['x'];
    $y=$_POST['y'];
    $cierre=']}';

    if(isset($x) && isset($y)){
        $sql="SELECT
            v.id,
            st_AsGeoJson(v.the_geom) as the_geom
        FROM
            ruta_noded_vertices_pgr AS v,
            ruta_noded AS e
        WHERE
            e.source=v.id OR e.target=v.id
        ORDER BY v.the_geom <-> ST_SetSRID(ST_MakePoint($x,$y),4326) LIMIT 1;";
        $row=  Persistencia::ejecuta($sql);
        if($row!=0){
            foreach($row as $value){
                $mensaje.='{ "type": "Feature", "properties": {';
                $mensaje.='"id": "'.$value['id'].'" },';
                $mensaje.=' "geometry": '.$value['the_geom'].'}';
                $mensaje .= ",";
            }
            $mensaje=  trim($mensaje, ',');
        }

    }
    $mensaje.=$cierre;

    echo($mensaje);
    Persistencia::cerrar();
```

Ilustración 39.- Fichero PHP con la consulta que obtiene el vértice más próximo a un punto dado

7.4 Obtención de la ruta óptima

En este apartado de la memoria se detallará las diferentes formas de lanzar la consulta que obtiene la ruta óptima, así como el tratamiento que hay que realizar sobre la misma para que pueda ser visualizada en el mapa con normalidad.

7.4.1 Obtención de la ruta óptima entre dos puntos mediante la interacción con el mapa

Esta aplicación tiene dos formas de interactuar con el mapa. La primera de ellas, y la más destacada, consiste en clicar en el mapa dos puntos cualesquiera. En la segunda se puede crear una ruta clicando en dos de los puntos que se muestran en el mapa.

La creación de un sendero a partir de dos puntos clicados en el mapa consiste, en términos programáticos, en manejar un evento de clicado en el mapa. Se controla que se clique dos veces en el mapa, y tras el segundo clic, se llama a las funciones que se encargan de ejecutar la consulta de la base de datos y de tratar la respuesta que da la misma. Una vez tratadas las coordenadas del sendero, se representan en el mapa.

```
function eventosRed(){
    infoRed.on('click', function(event){
        muestraInformacion(event);
        if(interaccionMapa===false && interaccionFormulario===false){
            if(mapa.getZoom()>=13){
                interaccionSenderos=true;
                if(vertices.length===0 && primerTramo==null){
                    reseteaValores();
                }
                var punto;
                if(puntos.length===0){
                    $("#boton-form-busqueda-limpiar").attr('disabled',false);
                    punto= new L.marker(event.latlng, {icon:
                    new L.icon({iconUrl:iconoInicio.url,iconSize:[iconoInicio.sizex,iconoInicio.sizey],iconAnchor:[iconoInicio.trasladox,iconoInicio.trasladoy]})});
                }
                if(puntos.length===1){
                    punto=new L.marker(event.latlng, {icon:
                    new L.icon({iconUrl:iconoFin.url,iconSize:[iconoFin.sizex,iconoFin.sizey],iconAnchor:[iconoFin.trasladox,iconoFin.trasladoy]})});
                }
                puntos.push(punto);
                mapa.addLayer(punto);
                if(puntos.length===1){
                    guardaVerticeCercano(event.layer.feature.properties.id);
                }
                if(puntos.length===2){
                    guardaVerticeCercano(event.layer.feature.properties.id);
                    pintaTramos();
                    vertices=[];
                    primerTramo=null;
                    interaccionSenderos=false;
                }
            }else{
                pararTimer(1);
                setTimeout(function(){
                    activaAlerta("Acérquese a la red para seleccionar un punto");
                },1000);

            }
        }else{
            pararTimer(1);
                setTimeout(function(){
                    activaAlerta("Debe terminar el sendero que comenzó");
                },1000);
        }
    });
```

Ilustración 40.- Detalle de la función que maneja el evento de clicado en senderos punto a punto

En la explicación de las consultas se comentaba que para obtener el camino más corto se necesita pasar como parámetro los vértices inicial y final. En este caso, no se tienen dos vértices, sino cuatro, que al combinarlos se obtiene el sendero resultante. Ambos puntos (inicial y final) estarán cercanos a un tramo de la red que tendrá un vértice inicial y otro final.

Asimismo, los cuatro vértices se obtienen a partir de una capa que está por encima de la red y que no se visualiza. Esta capa contiene información de los tramos, de tal manera, que al clicar sobre la red o sobre algún punto cercano a ella, se pueda obtener, no solo la información relativa al tramo de la red más próximo al punto que se ha clicado, sino también, los vértices que lo definen. El proceso a seguir se explicará detalladamente en la sección *7.4.6 Ajuste del sendero*.

Por otra parte, la creación de los senderos a partir de los puntos del mapa sigue otro procedimiento. Cada punto, al clicar sobre él, despliega un *popup* con información del mismo y un botón que da la posibilidad al usuario de añadir el punto a un sendero. Ese botón, al ser clicado, ejecuta una función que se encarga de recolectar las coordenadas de ese punto y almacenarlas. Cuando ya se tienen las coordenadas inicial y final, se busca el vértice más cercano (ver *7.3.3 Consulta para obtener un vértice cercano de ambas*) y se ejecuta la consulta para obtener el sendero más corto.

```
function botonPunto(nombre){
    if(interaccionFormulario===false && interaccionSenderos===false){
        if(interaccionFormulario===false && interaccionMapa===false){
            reseteaValores();
        }
        if(puntos[1]==null && puntos.length>0){
            if(nombreBoton!==nombre){
                var geomDestino=buscaGeomPunto('nombre='+nombre);
                getTarget(geomDestino.features[0].coordinates[0],true);
                dibujaPuntoFinal(geomDestino,false,false);
                dibujaTramoInteraccion(vertices[vertices.length-2],vertices[vertices.length-1]);
                origenSeleccionado=false;
                destinoSeleccionado=false;
                interaccionMapa=false;
                nombreBoton="";
            }else{
                activaAlerta("Elija otro destino");
            }

        }
        if(puntos[0]==null){
            interaccionMapa=true;
            var geomOrigen=buscaGeomPunto('nombre='+nombre);
            nombreBoton=nombre;
            getSource(geomOrigen.features[0].coordinates[0],true);
            dibujaPuntoInicial(geomOrigen,false);
        }
        $("#boton-form-busqueda-limpiar").attr('disabled',false);
    }else{
        pararTimer(1);
            setTimeout(function(){
            activaAlerta("Debe terminar el sendero que empezó");
            },1000);
    }
}
```

Ilustración 41.- Función que crea el sendero más corto a partir de los puntos del mapa

7.4.2 Obtención de la ruta óptima entre dos puntos mediante un formulario de búsqueda.

En este caso, el enrutamiento es más simple. Al comienzo de la página destinada a la creación de senderos punto a punto se puede observar un formulario que posibilita la creación de senderos a partir de puntos conocidos. El procedimiento que se sigue en este caso es similar al que se comentaba en el apartado anterior en la creación de los senderos a partir de los puntos del mapa.

El usuario selecciona un punto de origen, se representa en el mapa y se buscan sus coordenadas, y de igual modo para el punto final. Cuando el usuario pulsa el botón que construye el sendero más corto, se buscan los vértices más cercanos a los puntos inicial y final (ver *7.3.3 Consulta para obtener un vértice cercano de ambas*) y se ejecuta la consulta de enrutamiento.

```
function dibujaInteraccion(){
    if(interaccionMapa===false && interaccionSenderos===false){
        if(compruebaCampos($("#input-origen").val(),$("#input-destino").val())){
            limpiaCaminos();
            if(interaccionFormulario===false && (origenSeleccionado===false || destinoSeleccionado===false)){
                dibujaPuntoInicial(null,true);
                dibujaPuntoFinal(null,true,false);
            }
            interaccionFormulario=true;
            var geomOrigen=buscaGeomPunto('nombre='+$("#input-origen").val());
            var geomDestino=buscaGeomPunto('nombre='+$("#input-destino").val());
            var source=getSource(geomOrigen.features[0].coordinates[0],true);
            var target=getTarget(geomDestino.features[0].coordinates[0],true);
            dibujaTramoInteraccion(source, target);
            interaccionFormulario=false;
            origenSeleccionado=false;
            destinoSeleccionado=false;
            primerTramo=null;
        }
    }else{
        pararTimer(1);
            setTimeout(function(){
                activaAlerta("Debe terminar el sendero que empezó");
            },1000);
    }
    $("#boton-form-busqueda-limpiar").attr('disabled',false);
}
```

Ilustración 42.- Función que obtiene el camino más corto a partir de un formulario

7.4.3 Obtención del camino óptimo a partir de varios puntos

Esta funcionalidad da la posibilidad al usuario de crear un sendero a partir de muchos puntos clicados en el mapa. Al comienzo de la página destinada para ello, hay un botón que activa la construcción del sendero y otro que la desactiva.

Se trata de que el usuario clique en el mapa los puntos que considere oportuno para que el sistema le vaya mostrando el sendero más corto comprendido entre esos puntos. Por lo tanto, se trata del mismo procedimiento que en el apartado *7.4.1 Obtención de la ruta óptima entre dos puntos mediante la interacción con el mapa*, en lo que se refiere a la creación de un sendero a partir de dos puntos clicados en el mapa. La única salvedad consiste en que primero se obtiene el sendero más corto entre el primer punto y el segundo, luego se concatena el sendero más corto entre el segundo y el tercer punto, y así sucesivamente.

```
function eventosRed(){
    infoRed.on('click', function(event){
        muestraInformacion(event);
        if(marcaPuntos===true){
            if(mapaDiseño.getZoom()>=13){
                var punto= new L.marker(event.latlng, {icon:
                    new L.icon({iconUrl:iconoFin.url,iconSize:[iconoFin.sizex,iconoFin.sizey],iconAnchor:[iconoFin.trasladox,iconoFin.trasladoy]})});

                puntos.push(punto);
                console.log(event.layer.feature.properties.id);
                mapaDiseño.addLayer(punto);
                var array;
                if(puntos.length===1){
                    guardaVerticeCercano(event.layer.feature.properties.id);
                }
                if(puntos.length>=2){
                    guardaVerticeCercano(event.layer.feature.properties.id);
                    if(vertices.length>=2){
                        pintaTramos();
                    }
                }
            }else{
                pararTimer(1);
                setTimeout(function(){
                    activaAlerta("Acérquese a la red para seleccionar un punto");
                },1000);
            }
        }
    });
```

Ilustración 43.- Detalle de la función que maneja el evento de clicado en senderos multipunto

7.4.4 Cálculo de la distancia

El cálculo de la distancia entre dos puntos es una de las herramientas matemáticas fundamentales en el desarrollo de este proyecto, pues resulta necesario para poder ordenar las coordenadas o para conocer la distancia que hay entre dos puntos de un camino o sendero.

Debido a que las coordenadas vienen en geográficas, hay que resolver el desarrollo del arco de un triángulo esférico. En la *Ilustración 44* se puede ver la representación de este

triángulo, siendo **A** el punto inicial, λ*A* en la longitud de *A*, φ*A* es la latitud de *A*, *B* es el punto final, λ*B* en la longitud de *B*, φ*B* es la latitud de *B*.

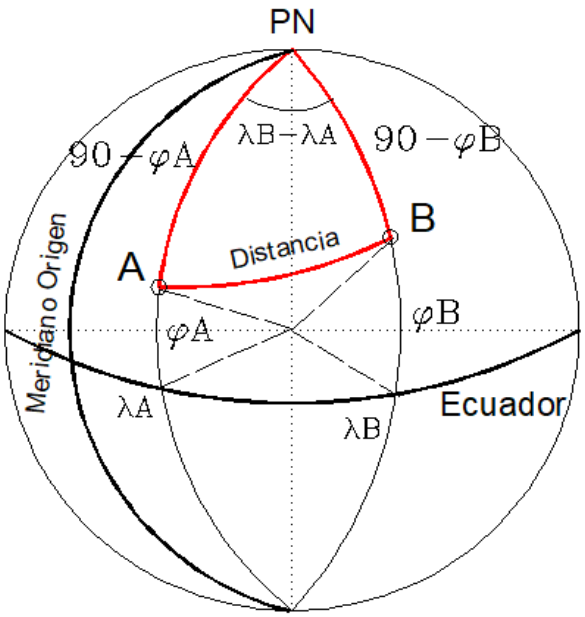


Ilustración 44.- Triángulo esférico

Para el cálculo de la distancia angular entre dos puntos se ha obtenido la siguiente expresión mediante la *Relación del coseno* [8]:

$$\cos\alpha = \cos(90° - \varphi B)\cos(90° - \varphi A) + \sin(90° - \varphi B)\sin(90° - \varphi A)\cos\Delta\lambda$$

Siendo α la distancia angular.

Para obtener la distancia lineal entre un punto y otro, se aplica la siguiente fórmula:

$$Distancia\ lineal = \mathrm{R} \cdot \cos^{-1}\alpha$$

Siendo *R* el radio de la tierra en kilómetros. También, es importante indicar que los ángulos deben estar en radianes.

7.4.5 Proyección de un punto sobre una recta

El cálculo de la proyección de un punto sobre una recta es otra herramienta matemática importante en el desarrollo de este trabajo. Esta proyección permite ajustar los senderos a los puntos que el usuario clica en el mapa. Como el cálculo de la proyección se debe hacer sobre coordenadas *de un plano local*, se explicará primero como se realiza la conversión de coordenadas geográficas a coordenadas *de un plano local*, y seguidamente se detallará el procedimiento que se llevó a cabo para realizar la proyección.

Conversión de coordenadas geográficas a coordenadas de un plano local

La diferencia entre coordenadas geográficas y coordenadas *de un plano local* radica en que las geográficas representan la latitud y longitud del punto, mientras que las *de un plano local* representan coordenadas rectangulares X e Y. Por esta razón, la proyección se debe calcular en coordenadas *de un plano local*, pues las geográficas vienen dadas por ángulos.

Para realizar la conversión se ha seguido las siguientes expresiones [9]:

$$x = R(\lambda - \lambda_0)\cos\varphi_0$$

$$y = R(\varphi - \varphi_0)$$

Siendo λ_0 la longitud inicial del plano sobre el que se va a realizar la conversión, λ del punto a realizar la conversión, φ_0 la latitud del plano sobre el que se va a realizar la conversión, φ la latitud del punto sobre el que se va a realizar la conversión y *R* el radio de la tierra en kilómetros.

Estas ecuaciones tratan de realizar la transformación partiendo de un plano en la esfera terrestre, situado en las islas Canarias, en lugar de hacerlo con toda la esfera. Las coordenadas iniciales que definen este plano son φ_0 =28.2° y λ_0 = 16.6°.

Para realizar el proceso inverso, se despeja de las ecuaciones anteriores los valores de φ y λ.

Proyección de un punto sobre una recta

A continuación se detallará el proceso a seguir para realizar la proyección de un punto sobre una recta [10]. Lo primero que hay que realizar es definir la ecuación paramétrica de la recta, que llamaremos *r*, que permita obtener cualquier punto contenido en esta recta. Para comenzar este procedimiento, es necesario pasar las coordenadas geográficas a coordenadas *de un plano local*.

$$x = x_1 + \lambda(x_2 - x_1)$$

$$y = y_1 + \lambda(y_2 - y_1)$$

Siendo x_1 una longitud de un punto del sendero, x_2 otra longitud de un punto del sendero, y_1 una latitud de un punto del sendero e y_2 otra longitud de un punto del sendero, todas en coordenadas *de un plano local*. En este caso, λ no representa a la longitud, pues como bien se indicaba, ahora se está trabajando en coordenadas *de un plano local*. Ahora, λ representa al factor lambda de la ecuación paramétrica de la recta. A partir de ahora, la diferencia entre los puntos *x* e *y* se representarán de la siguiente manera: Δx y Δy .

Seguidamente, se define un plano π que pasa por el punto que se quiere proyectar, que llamaremos $P(x_p,y_p)$, y perpendicular a la recta definida anteriormente. Esto es, el vector normal del plano será el vector de dicha recta ($\overrightarrow{v_n}(\pi) = \overrightarrow{v_r}$).

Sabiendo que la ecuación continua de π es $Ax + By + Zc + D = 0$ y que el vector director de la recta es ($\Delta x, \Delta y$), obtenemos *D* sustituyendo *A* y *B* por el vector director, ya que como se comentaba antes, el vector director de la recta y el vector normal del plano es el mismo. Además, comosabemos que el punto que se quiere proyectar es $P(x_p,y_p)$, se sustituye en la ecuación del plano *x* por x_p e *y* por y_p.

$$D = -\Delta x \cdot x_p - \Delta y \cdot y_p$$

Una vez obtenido el valor de *D* y sabiendo que el vector director de la recta *r* es ($\Delta x, \Delta y$), la ecuación del plano quedaría de la siguiente manera:

$$\Delta x \cdot x + \Delta y \cdot y - \Delta x \cdot x_p - \Delta y \cdot y_p = 0$$

Ahora, se corta el plano π con la recta *r*. Para ello, se realiza $r \cap \pi$, que consiste en introducir la ecuación paramétrica de la recta dentro de la ecuación del plano:

$$\Delta x \cdot (x_1 + \lambda\Delta x) + \Delta y \cdot (y_1 + \lambda\Delta y) - \Delta x \cdot x_p - \Delta y \cdot y_p = 0$$

Para poder obtener el punto proyectado en la recta, necesitamos obtener el valor de lambda, para una vez calculado, sustituir en la ecuación paramétrica y obtener el valor del punto. Por esta razón, obtenemos λ despejando:

$$\lambda = \frac{\Delta y \cdot y_p + \Delta x \cdot x_p - \Delta y \cdot y_1 - \Delta x \cdot x_1}{\Delta x^2 - \Delta y^2}$$

Obtenidas las coordenadas del punto proyectado, se realiza la conversión a coordenadas geográficas para que el punto pueda ser representado con el resto de coordenadas que conforman el sendero.

```
function ecuacionParametrica(lng1,lng2,lat1,lat2,l){
    var valores=[];
    var lat=lat1+l*diferencia(lat1,lat2);
    var lng=lng1+l*diferencia(lng1,lng2);
    valores[0]=lng;
    valores[1]=lat;
    return valores;
}

function diferencia(valor1,valor2){
    return valor2-valor1;
}

function getL(lng1,lng2,lat1,lat2,lngp,latp){
    var diferenciaLat=diferencia(lat1,lat2);
    var diferenciaLng=diferencia(lng1,lng2);
    var numerador=diferenciaLat*latp+diferenciaLng*lngp-diferenciaLat*lat1-diferenciaLng*lng1;
    var denominador=Math.pow(diferenciaLng, 2)+Math.pow(diferenciaLat, 2);
    return numerador/denominador;
}

```

Ilustración 45.- Funciones JavaScript que realizan la proyección de un punto sobre una recta

7.4.6 Ajuste del sendero

Cuando se construye un sendero clicando en el mapa los puntos inicial y final, se deben tratar las coordenadas obtenidas tras la consulta a la base de datos, pues hay que conseguir que la representación sea lo más precisa posible. Este problema solo surge cuando se construye el sendero de esta manera.

Siguiendo la línea de lo que se comentaba en la *sección 7.4.1 Obtención de la ruta óptima entre dos puntos mediante la interacción con el mapa,* es importante aclarar que para cada punto clicado (inicial y final) se seleccionan los dos vértices del tramo más cercano en cada caso, obteniendo así cuatro vértices, dos por cada punto clicado. De esta manera conocemos los vértices necesarios que se necesitan para construir el sendero que el usuario quiere. Solo falta combinarlos entre sí para obtener, de la forma más exacta posible, el sendero que empiece y termine en los puntos que se han seleccionado en el mapa.

Tras analizar el problema se detectaron tres posibles casos que afectan a los cuatro vértices, así como a la combinación de los mismos. A continuación se detallarán estos tres casos, así como la forma de abordarlos.

- ❖ De los cuatro vértices obtenidos, dos son iguales.
  - Por ejemplo: Los vértices del tramo en el que se encuentra el *Punto Inicial* (ver *Ilustración 47*) son 283 y 282, mientras que los vértices del tramo en el que se encuentra el *Punto Final* (ver *Ilustración 47*) son 282 y 278.
- ❖ De los cuatro vértices obtenidos, todos son distintos.
  - Por ejemplo: Los vértices del tramo en el que se encuentra el *Punto Inicial* (ver *Ilustración 52*) son 488 y 487, mientras que los vértices del tramo en el que se encuentra el *Punto Final* (ver *Ilustración 52*) son 413 y 491.
- ❖ De los cuatro vértices obtenidos, son iguales dos a dos.
  - Por ejemplo: Los vértices del tramo en el que se encuentra el *Punto Inicial* (ver *Ilustración 58*) son 430 y 431, mientras que los vértices del

tramo en el que se encuentra el *Punto Final* (ver *Ilustración 58*) son 430 y 431.

Para resolver estos problemas encontrados, se llevaron a cabo las siguientes soluciones:

Vértices comunes

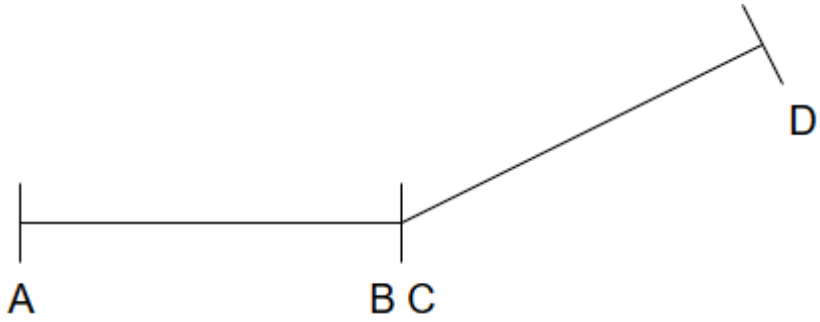


Ilustración 46.- Vértices comunes

Para hacer la explicación lo más sencilla posible, y así poder facilitar su comprensión, se detallarán los pasos que se han seguido para llegar a la solución buscada, acompañándolos de un ejemplo ilustrativo.

En la siguiente imagen se puede ver el ejemplo sobre el que se va a basar la explicación. En la primera imagen se pueden observar los puntos que ha clicado el usuario, y en la segunda, los tramos de la red más próximos estos puntos clicados. Los vértices de la red son, para el primer punto, 283 y 282, mientras que para el segundo punto, 282 y 278.

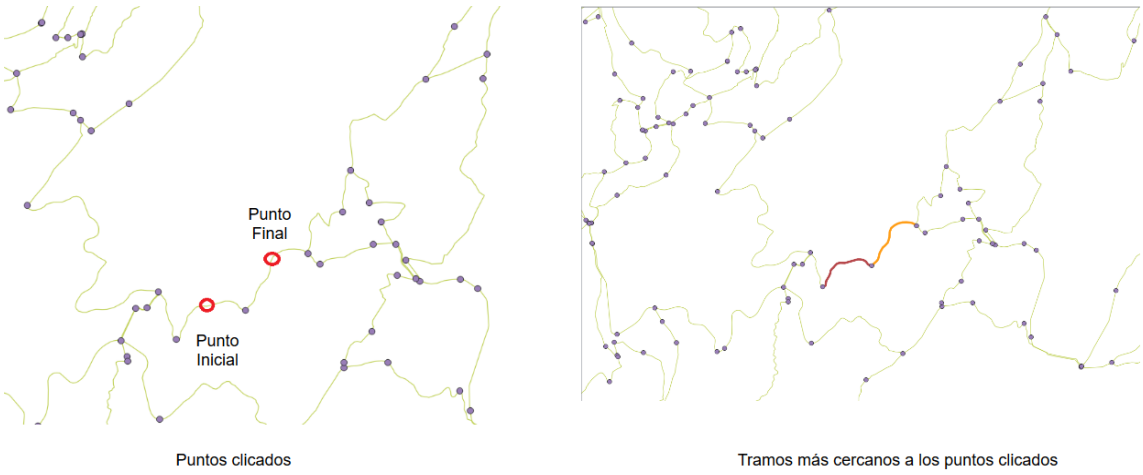


Ilustración 47.- Ejemplo del caso de los vértices comunes

1) Se realiza cuatro veces la consulta que obtiene el sendero más corto con los vértices. El motivo por el que se realiza cuatro veces es porque se ejecuta la consulta con las cuatro combinaciones de los vértices obtenidos, es decir, los dos vértices obtenidos del punto inicial y los dos del punto final.

   Las combinaciones realizadas siguen el siguiente orden:

   1. Primer vértice del punto inicial (vértice 283) – Primer vértice del punto final (vértice 282). Esta combinación corresponde a la línea verde que se puede ver en la *Ilustración 48*.

2. Primer vértice del punto inicial (vértice 283) – Segundo vértice del punto final (vértice 278). Esta combinación corresponde a la línea malva que se puede ver en la *Ilustración 48*.
3. Segundo vértice del punto inicial (vértice 282) – Primer vértice del punto final (vértice 282). Esta combinación no se puede ver en la *Ilustración 48*, pues la distancia del sendero resultante entre los vértices es 0.
4. Segundo vértice del punto inicial (vértice 282) –Segundo vértice del punto final (vértice 278). Esta combinación corresponde a la línea naranja que se puede ver en la *Ilustración 48*.

2) De las cuatro consultas realizadas, se obtienen de los tramos obtenidos los dos que tengan menor distancia y que estos sean distintos de 0.

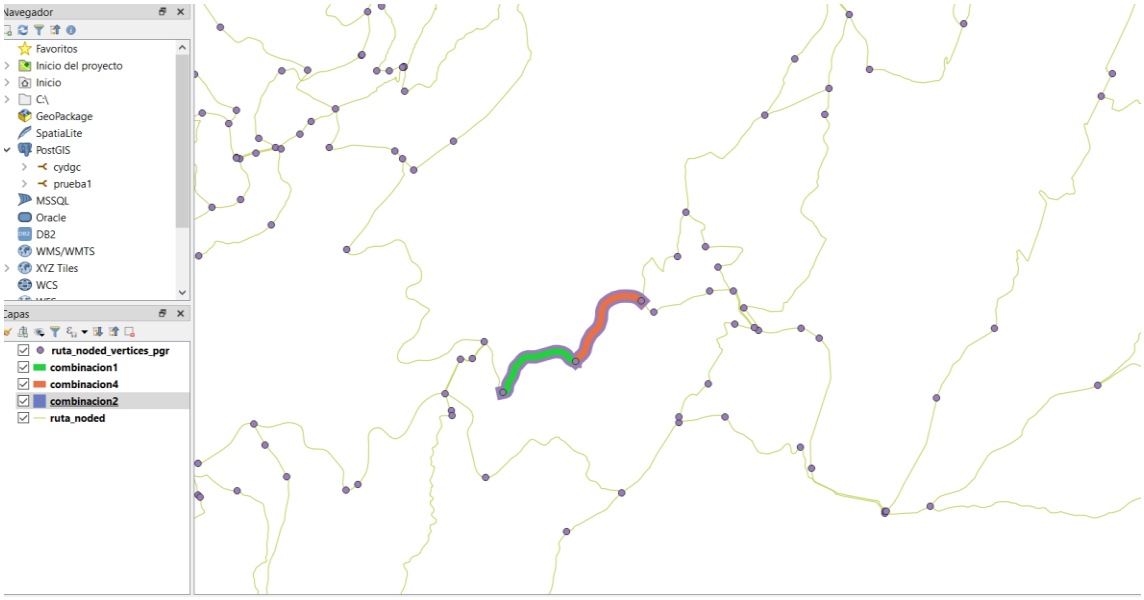


Ilustración 48.- Representación de los tramos del caso de los vértices comunes

Tal y como se ven representados los tramos en la *Ilustración 48*, se escogen la combinación primera (verde) y la cuarta (naranja), pues la tercera tiene distancia 0 y la segunda es el más largo.

3) Cada tramo se almacena ordenadamente en un vector diferente. Seguidamente, se averigua cuál es el vértice común.
4) Una vez localizado el vértice común, en cada vector se averiguan las coordenadas del punto que corresponde al vértice común.
5) Luego, en ambos vectores, se busca cuál es el punto más cercano al punto clicado y se recorre el vector desde el punto que corresponde al vértice común hasta este punto encontrado. De esta manera se evita que los tramos se alargue o se quede más corto.
6) Se proyectan los puntos clicados sobre el sendero.
7) Se preparan las coordenadas para ser dibujadas en el mapa.

Ilustración 49.- Resultado final del caso de los vértices comunes

```
if(rutas[0][0].coste===0 && rutas[1][0].coste===0){
    arrayTramo1=[pasaArrayOrdenado(rutas[2],true),rutas[2][1].properties.vertice,rutas[2][2].properties.vertice];
    arrayTramo2=[pasaArrayOrdenado(rutas[3],true),rutas[3][1].properties.vertice,rutas[3][2].properties.vertice];
    indices=compruebaExtremos(arrayTramo1[0],arrayTramo2[0]);
}else{
    if(rutas[0][0].coste===0){
        arrayTramo1=[pasaArrayOrdenado(rutas[1],true),rutas[1][1].properties.vertice,rutas[1][2].properties.vertice];
        arrayTramo2=[pasaArrayOrdenado(rutas[2],true),rutas[2][1].properties.vertice,rutas[2][2].properties.vertice];
        indices=compruebaExtremos(arrayTramo1[0],arrayTramo2[0]);
    }else{
        arrayTramo1=[pasaArrayOrdenado(rutas[0],true),rutas[0][1].properties.vertice,rutas[0][2].properties.vertice];
        arrayTramo2=[pasaArrayOrdenado(rutas[1],true),rutas[1][1].properties.vertice,rutas[1][2].properties.vertice];
        indices=compruebaExtremos(arrayTramo1[0],arrayTramo2[0]);
    }
    if(rutas[3][0].features.length===1){
        idActualizacion.push(rutas[3][0].features[Math.round(rutas[3][0].features.length/2)-1].properties.id);
    }else{
        idActualizacion.push(rutas[3][0].features[Math.round(rutas[3][0].features.length/2)].properties.id);
    }
    dibujaUnVertice(arrayTramo1,arrayTramo2, indices);
}

}
```

Ilustración 50.- Fragmento de código que trata el caso de los vértices comunes

Vértices distintos

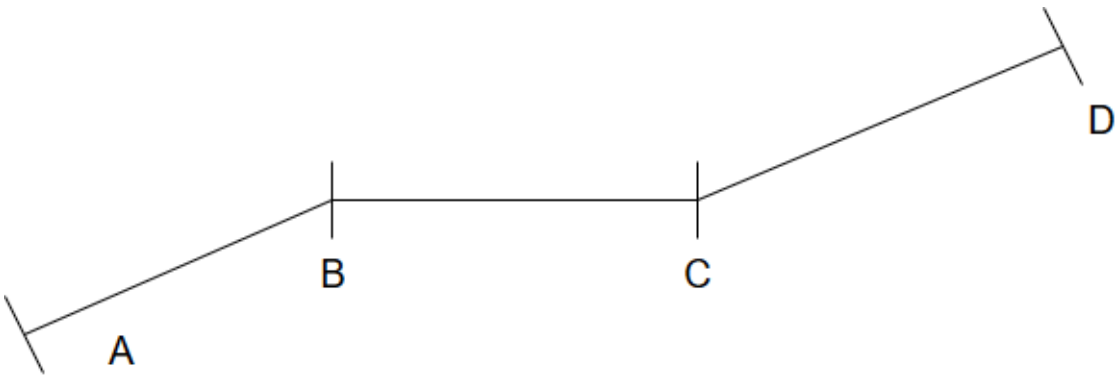


Ilustración 51.- Vértices distintos

Al igual que en el caso anterior, se detallarán los pasos que se siguieron para abordar este problema y se ilustrarán con un ejemplo.

En la siguiente *Ilustración*, la primera imagen muestra los puntos que ha clicado el usuario, y la segunda, los tramos de la red más próximos estos puntos clicados. Los vértices de la red son, para el primer punto, 488 y 487, mientras que para el segundo punto, 413 y 491.

Puntos clicados

Tramos más cercanos a los puntos clicados

Ilustración 52.- Ejemplo del caso de los vértices distintos

1) Se realiza cuatro veces la consulta que obtiene el sendero más corto con los vértices. Se sigue el mismo razonamiento que en el caso anterior.

   Las combinaciones realizadas siguen el siguiente orden:

   1. Primer vértice del punto inicial (vértice 488) – Primer vértice del punto final (vértice 413). Esta combinación corresponde a la línea marrón de la *Ilustración 53*.
   2. Primer vértice del punto inicial (vértice 488) – Segundo vértice del punto final (vértice 491). Esta combinación corresponde a la línea azul de la *Ilustración 53*.
   3. Segundo vértice del punto inicial (vértice 487) – Primer vértice del punto final (vértice 413). Esta combinación corresponde a la línea roja de la *Ilustración 53*.
   4. Segundo vértice del punto inicial (vértice 487) –Segundo vértice del punto final (vértice 491). Esta combinación corresponde a la línea amarilla de la *Ilustración 53*.

2) De las cuatro consultas realizadas, se obtiene el tramo que tenga menor distancia y que sea distinto de 0. Una vez detectado este tramo, se ordenan sus coordenadas en un vector y se dibuja.

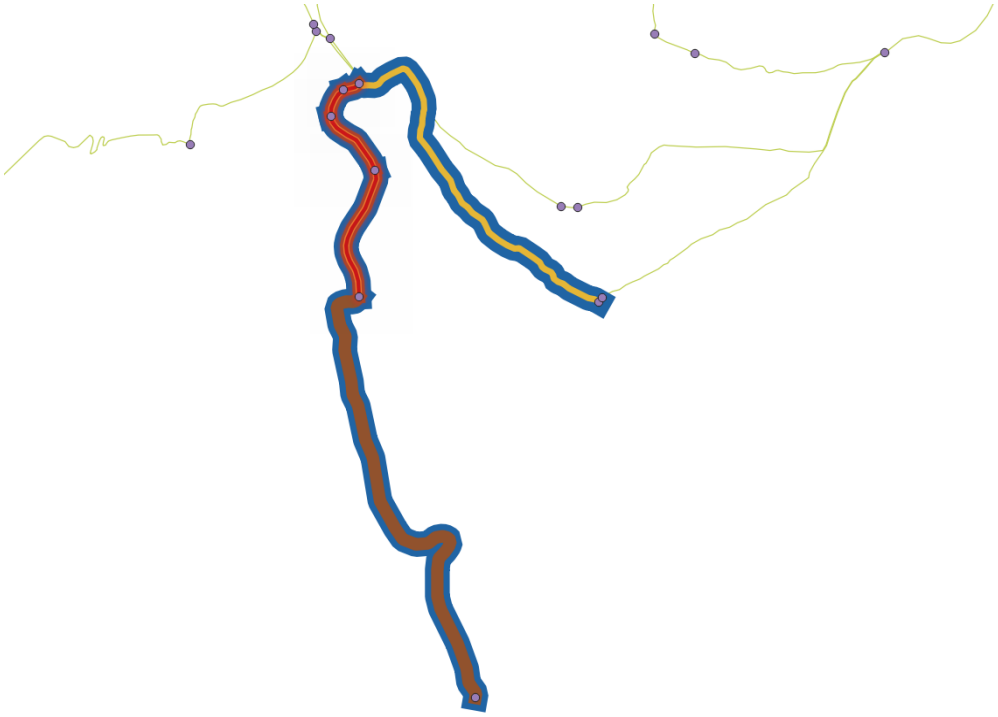


Ilustración 53.- Representación de los tramos del caso de los vértices distintos

Tal y como se ven representados los tramos en la *Ilustración 53*, se escoge el tramo rojo porque es el más corto de los cuatro.

3) Una vez obtenido la parte central del sendero, se obtienen los extremos. Para ello, se ejecuta de nuevo la consulta que obtiene la ruta óptima. El primer extremo se consigue obteniendo la ruta óptima entre los vértices del primer punto clicado (ruta más corta entre los vértices 488 y 487), y el segundo obteniendo la ruta óptima entre los vértices del segundo punto clicado (ruta más corta entre los vértices 413 y 491). En los dos casos se almacenan de forma ordenada las coordenadas en un vector.
4) Seguidamente, se tratan los extremos:
    a. Se analizan los puntos inicial y final de cada tramo extremo para unirlos al tramo central.
    b. Se busca, en cada tramo extremo, el punto más próximo al punto clicado, así evitar que los tramos extremos se alargue o se quede más corto.

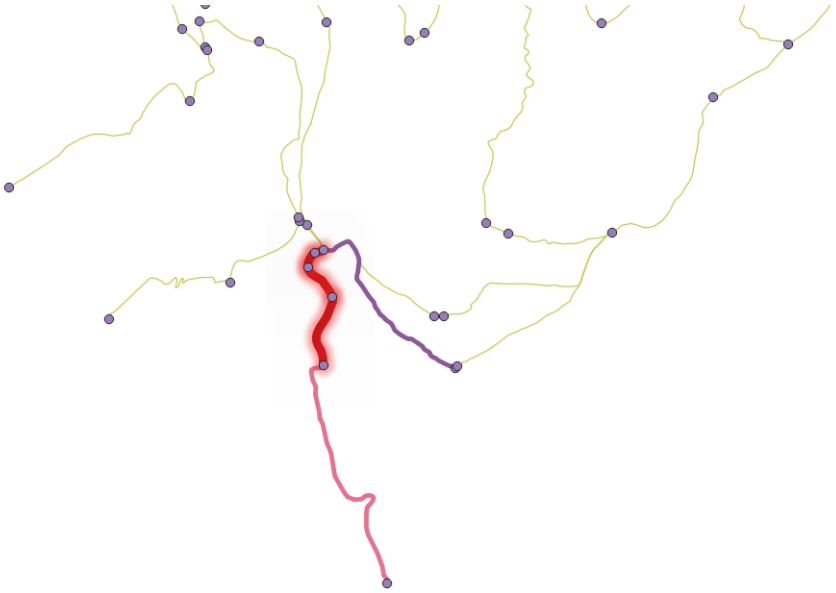


Ilustración 54.- Representación de los tramos finales del caso de los vértices distintos

En la *Ilustración 54* se puede ver como se unen al tramo más corto los tramos extremos que completan el sendero. En *la Ilustración 55* se podrá visualizar el resultado final del sendero con los extremos recortados.

5) Se proyectan los puntos clicados sobre el sendero.
6) Se preparan las coordenadas para ser dibujadas en el mapa.

Ilustración 55.- Resultado final del caso de los vértices distintos

```
function verticesDistintos(){
    var datas=[];
    var ruta=null;
    var indices=null;
    var arrayTramo1=null;
    var arrayTramo2=null;

    datas[0]=[vertices[vertices.length-2].features[0],vertices[vertices.length-1].features[0]];
    datas[1]=[vertices[vertices.length-2].features[0],vertices[vertices.length-1].features[1]];
    datas[2]=[vertices[vertices.length-2].features[1],vertices[vertices.length-1].features[0]];
    datas[3]=[vertices[vertices.length-2].features[1],vertices[vertices.length-1].features[1]];

    for (var i = 0; i < datas.length; i++) {
        rutas[i]=[consultaTramos(datas[i][0].properties.vertice,datas[i][1].properties.vertice),datas[i][0],datas[i][1]];
    }
    rutas.sort(ordenaMenorMayor);
    console.log(rutas);
    var verticeIgual=compruebaVerticesComunes();
    if(verticeIgual==null){
        var tramoPrincipal;
        var arrayTramoPrincipal;
        if(rutas[0].coste===0 && rutas[1].coste===0){
            tramoPrincipal=rutas[2][0];
            arrayTramoPrincipal=pasaArrayOrdenado(rutas[2],true);
        }else{
            if(rutas[0].coste===0){
                tramoPrincipal=rutas[1][0];
                arrayTramoPrincipal=pasaArrayOrdenado(rutas[1],true);
            }else{
                tramoPrincipal=rutas[0][0];
                arrayTramoPrincipal=pasaArrayOrdenado(rutas[0],true);
            }
        }
        if(getDistanciaGeometria(puntos[puntos.length-2].getLatLng().lat,puntos[puntos.length-2].getLatLng().lng,puntos[puntos.length-1].getLatLng().lat,puntos[puntos.length-1].get
            idActualizacion.push(tramoPrincipal.features[Math.round(tramoPrincipal.features.length/2)-1].properties.id);
            idActualizacion.push(tramoPrincipal.features[Math.round(tramoPrincipal.features.length/2)].properties.id);
        }else{
            if(rutas[3][0].features.length===1){
                idActualizacion.push(rutas[3][0].features[Math.round(rutas[3][0].features.length/2)].properties.id);
            }else{
                idActualizacion.push(rutas[3][0].features[Math.round(rutas[3][0].features.length/2)-1].properties.id);
            }

        }

        dibujaDosVertices(tramoPrincipal,arrayTramoPrincipal);
```

Ilustración 56.- Fragmento de código que trata el caso de los vértices distintos

Vértices iguales dos a dos

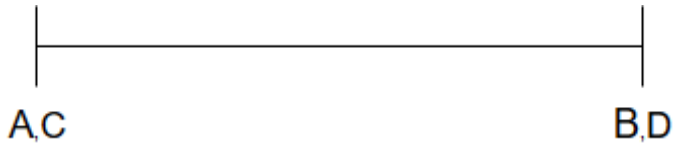


Ilustración 57.- Vértices iguales dos a dos

Este es el caso más simple de todos. Siguiendo la misma dinámica que en los casos anteriores, se detallarán, acompañados de un ejemplo, los pasos que se han seguido para tratar esta situación. Se parte de la precondición de que se ha detectado que los vértices son iguales dos a dos, tal y como se puede ver en la *Ilustración 57*.

Como en anteriores ocasiones, podemos ver en la *Ilustración 58* que la primera imagen representa los puntos clicados por el usuario, y la segunda, el tramo de red más próximo a estos puntos clicados. Los vértices de la red son, únicamente, 430 y 431.

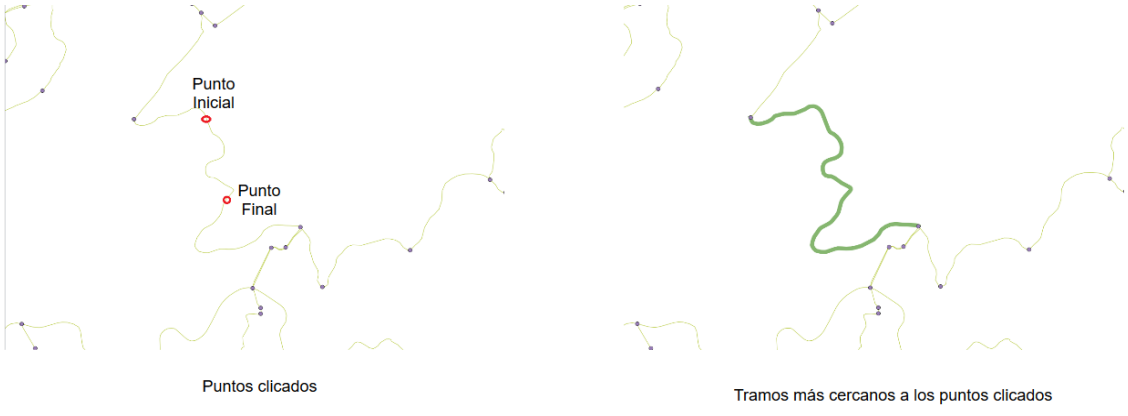


Ilustración 58.- Ejemplo del caso de los vértices iguales dos a dos

1) Se obtiene el sendero más corto entre ambos vértices.

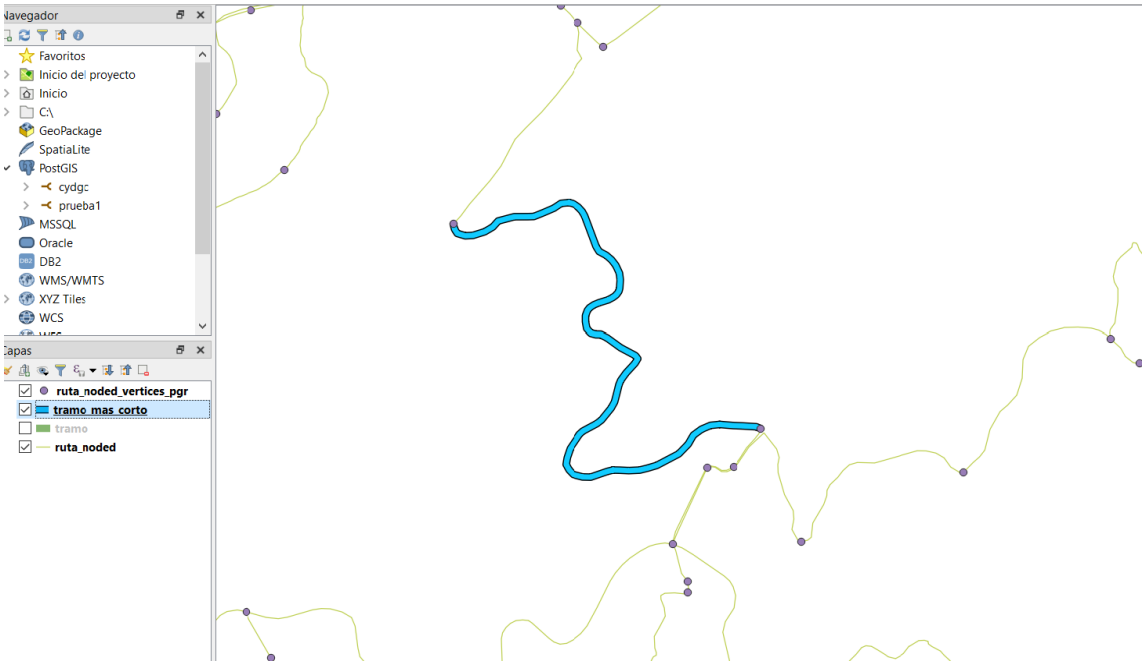


Ilustración 59.-Representación de los tramos finales del caso de los vértices iguales dos a dos

En la imagen se puede ver el tramo final, ya que en este caso no ha hecho falta realizar las cuatro combinaciones que se ejecutan en los casos anteriores, pues los vértices del tramo más próximo al punto inicial son los mismos que los vértices del tramo más próximo del punto final.

2) Se colocan las coordenadas en un vector.
3) Al igual que en los casos anteriores, se obtienen los puntos más cercanos a los puntos clicados por el usuario para evitar que el sendero se alargue o se quede más corto.
4) Se proyectan los puntos clicados sobre el sendero.
5) Se preparan las coordenadas para ser dibujadas en el mapa.

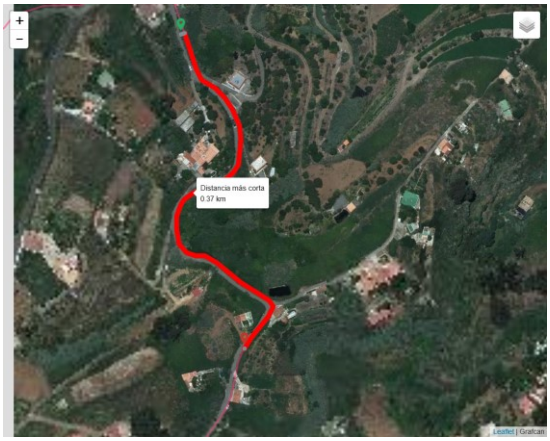


Ilustración 60.- Resultado final del caso de los vértices iguales dos a dos

```
function verticesIguales(){
    var tramo=consultaTramos(vertices[vertices.length-1].features[0].properties.vertice,vertices[vertices.length-1].features[1].properties.vertice);
    var arrayTramo=pasaArray(tramo);
    var indicePrimerPunto=buscaCoordenadaTramo(arrayTramo,puntos[puntos.length-2].getLatLng().lat,puntos[puntos.length-2].getLatLng().lng,0);
    var indiceSegundoPunto=buscaCoordenadaTramo(arrayTramo,puntos[puntos.length-1].getLatLng().lat,puntos[puntos.length-1].getLatLng().lng,0);
    var jsonTramo=JSON.parse(construyeJsonRecortadaTramos(arrayTramo,indicePrimerPunto,indiceSegundoPunto,puntos[puntos.length-2].getLatLng(),puntos[puntos.length-1].getLatLng()));
    dibujaTramo(jsonTramo);
}
```

Ilustración 61.- Fragmento de código que trata el caso de los vértices iguales dos a dos

Para ilustrar mejor estos casos, se muestra a continuación un diagrama de actividades que recoge el procedimiento seguido.

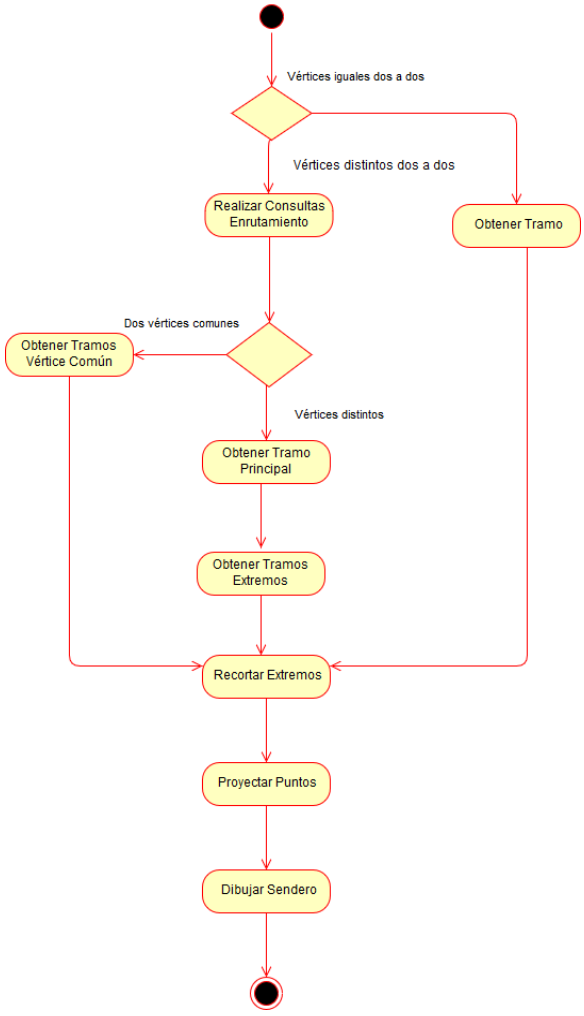


Ilustración 62.- Diagrama de actividades que representa la obtención de los senderos

7.4.7 Líneas discontinuas

En algunas ocasiones, cuando se interacciona con el mapa clicando puntos en él, queda cierta distancia entre el punto clicado y el comienzo o final del sendero. Con el fin de guiar al usuario, se dibujan en el mapa unas líneas discontinuas entre el punto clicado y el comienzo o el final del sendero.

Ilustración 63.- Líneas discontinua gris entre el inicio del sendero y el punto clicado por el usuario

7.5 Obtención de la ruta alternativa

La aplicación permite obtener otro sendero alternativo para que el usuario pueda visualizar otro sendero entre los puntos que ha escogido, así como realizar una comparación con el original.

Para realizar esta funcionalidad, simplemente se ejecutan las funciones que realizan el enrutamiento y que se han explicado arriba. Una vez completado el cálculo del sendero original, se modifican temporalmente los pesos (distancias de los tramos) de las aristas del grafo de la red que se encuentran en el medio del sendero. Una vez terminado el cálculo del sendero alternativo se actualizan los valores de los pesos para que vuelvan a tener su valor original.

La modificación de los pesos consistirá en pasar el valor de la distancia, que inicialmente está en kilómetros, a metros. De esta manera se fuerza a la función que obtiene la ruta óptima a que no pase por esas aristas y, como consecuencia de ello, se obtenga otro sendero pero de mayor distancia.

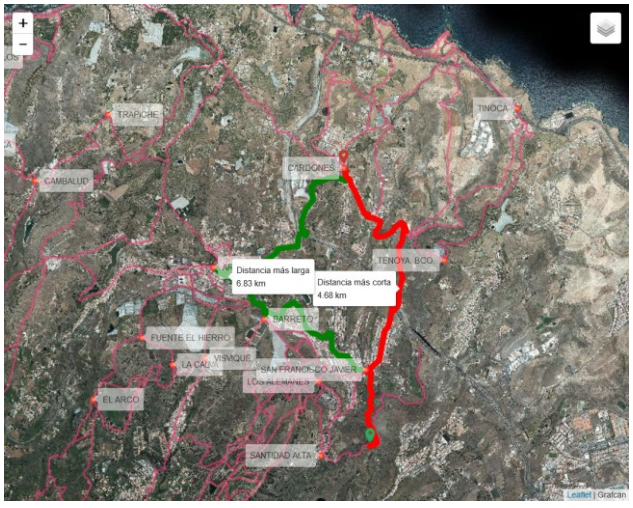


Ilustración 64.- Visualización de la ruta alternativa

```
461  function actualizaCostes(json){
462      if(json.features.length>1 && json.features!==null){
463          for(var i=0;i<idActualizacion.length;i++){
464              consultaAjax("ActualizaCostes.php","id="+idActualizacion[i]+"&modo=actualiza");
465          }
466      }
467  }
468
469  function reseteaCostes(){
470      for(var i=0;i<idActualizacion.length;i++){
471          consultaAjax("ActualizaCostes.php","id="+idActualizacion[i]+"&modo=resetea");
472      }
473      idActualizacion=[];
474  }
475
```

Ilustración 65.- Funciones JavaScript que actualizan y resetean los pesos del grafo de la red

```
<?php

    include_once "Persistencia.php";
    $conn=Persistencia::getConexion();
    $id=$_POST['id'];
    $modo=$_POST['modo'];
    $row=0;

    if(isset($id) && isset($modo)){
        if(strcmp($modo, "actualiza")===0){
            $sql="UPDATE  ruta_noded SET distancia = ST_Length(ST_Transform(the_geom,4326)::geography) WHERE id = $id;";
        }else{
            $sql="UPDATE  ruta_noded SET distancia = ST_Length(ST_Transform(the_geom,4326)::geography)/1000 WHERE id = $id;";
        }
        $row=  Persistencia::ejecuta($sql);
    }
    echo($row);
    Persistencia::cerrar();
```

Ilustración 66.- Fichero PHP que ejecuta la sentencia SQL que actualiza los pesos

7.6 Perfil topográfico

El *perfil topográfico* o *gráfica de altimetría* es otro de los elementos relevantes de la aplicación. Se trata de una gráfica que representa la altura de cada una de las coordenadas que conforman el sendero.

Tal y como se nos han proporcionado los datos, las coordenadas nos disponen de la altura, por lo que la creación del perfil se ha tenido que desarrollar de forma independiente al enrutamiento, usando para ello una imagen de alturas.

La imagen de alturas consiste en una imagen que sigue el *modelo raster* [1]. El modelo *raster* representa un espacio en una imagen. Esta imagen está dividida en píxeles de igual tamaño que contienen un número, que en este caso es la altura del espacio que representa.

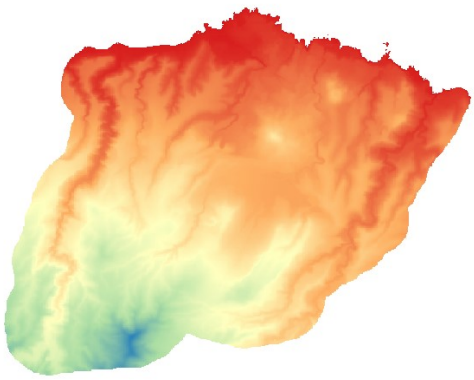


Ilustración 67.- Visualización de la imagen de altura desde QGis. Esta imagen de alturas es solo de la zona de estudio.

Esta imagen de alturas, que se ha obtenido del *Centro de Descargas del CNIG*[4], se ha importado en *QGis* para extraer la información de los píxeles en formato *ASC*. Una vez obtenidos los datos, se pasaron a formato *CSV*, con el objetivo de poder cargar la información con mayor facilidad en la aplicación. El fichero *CSV* resultante se subió al servidor. Para obtener la información de las alturas en la aplicación se realiza una petición *AJAX* a ese fichero y se cargan los datos en una matriz.

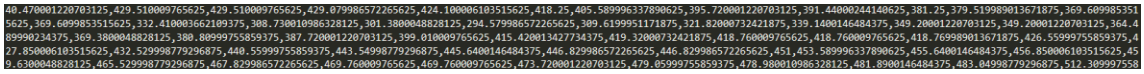


Ilustración 68.- Aspecto del fichero CSV con las alturas

```
function carga(data){
    var allRows = data.split(/\r?\n|\r/);
    for (var i = 0; i < allRows.length-1; i++) {
        var rowCells = allRows[i].split(',');
        table.push(rowCells);
    }
}
```

Ilustración 69.- Función JavaScript que carga los datos de la imagen de alturas en una matriz

Es importante destacar que para formatear el fichero con las alturas a *CSV* se desarrolló un programa en *Java*. El funcionamiento de este programa consiste en leer el fichero *ASC* y escribir en uno nuevo las alturas separadas por comas.

[4] http://centrodedescargas.cnig.es/CentroDescargas/index.jsp

```
    public void formatea() throws IOException{
        try {
            String linea;
            while((linea=br.readLine())!=null){
                bw.write(linea.trim().replaceAll(" ", ",")+"\n");
                lineas++;
            }

        }
        catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if( null != fr ){
                fr.close();
                }
                bw.close();
            }catch (Exception e2){
                e2.printStackTrace();
            }
        }
    }
```

Ilustración 70.- Función Java que formatea el fichero ASC.

Otro de los aspectos que se llevaron a cabo en el desarrollo del perfil ha sido la interpolación. Para poder obtener la altura de un punto en el mapo es imprescindible interpolar. La interpolación desarrollada consiste en, conociendo las coordenadas de cada esquina de la imagen y el número de pixeles que tiene la imagen de ancho y de alto, resolver las siguientes ecuaciones:

$$\text{Para la longitud: } C = \frac{\lambda - \lambda_0}{\lambda_1 - \lambda_0} W$$

$$\text{Para la latitud: } F = \frac{\varphi - \varphi_0}{\varphi_1 - \varphi_0} H$$

Con el fin de hacer la explicación más sencilla, se explicará el razonamiento seguido para el cálculo del índice de la columna de la matriz, pues el cálculo del índice de la fila es análogo. Siendo λ la longitud del punto sobre el que se quiere obtener la altura, λ_0 la longitud del margen izquierdo de la imagen, λ_1 la longitud del margen derecho y *W* el ancho de la imagen en pixel.

Conocido el nombre de las variables de la fórmula, se puede deducir que (ver *Ilustración 63*):

- Cuando λ esté cercana a λ_0, la columna C de la matriz estará cercana a la 0.
- Cuando λ esté cercana a λ_1, la columna C de la matriz estará cercana a la W.
- Cuando λ esté a mitad entre λ_0 y λ_1, la columna de la matriz estará cercana a $\frac{W}{2}$.

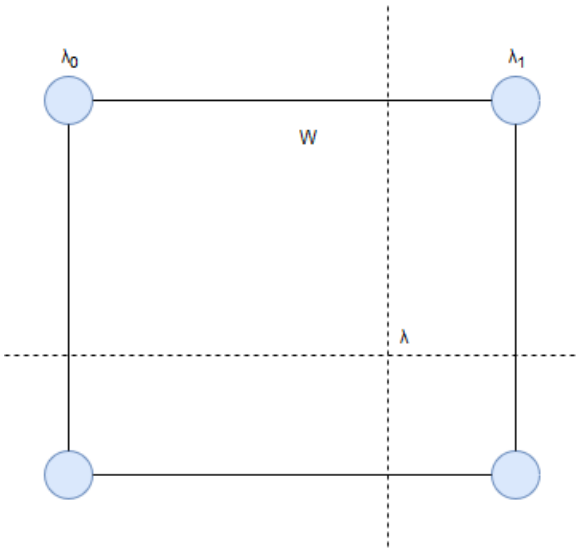


Ilustración 71.- Gráfico representativo del cálculo de la interpolación en la imagen de alturas

```
161  function getColumna(longitud){
162      return interpolacion(longitud, -15.5864985659999995,-15.4762375849999998,725);
163  }
164
165  function getFila(latitud){
166      return interpolacion(latitud,28.1626316989999985,28.0693209520000018, 686);
167  }
168
169  function interpolacion(x,x0, x1, factor){
170      return Math.round((x-x0)/(x1-x0)*factor);
171  }
```

Ilustración 72.- Funciones JavaScript que se encargan de la interpolación

Finalmente, para crear el gráfico, se almacena en un vector, por una parte, el valor acumulado de la distancia entre el punto sobre el que se quiere averiguar la altura y su anterior, y por otra las altura de cada punto del sendero. La distancia del primer punto a analizar será 0, pues es el comienzo del sendero o camino.

```
117  function creaValores(coordenadas, alturas, distancias,datos){
118      for (var i = 0; i < coordenadas.length; i++) {
119          var fila=getFila(coordenadas[i][1]);
120          var columna=getColumna(coordenadas[i][0]);
121          alturas[i]=table[fila][columna];
122          if(i===0){
123              distancias[0]=0;
124              datos.push({distancia: +0, altura: +alturas[i]});
125          }else{
126              var distancia=distancias[i-1]+getDistanciaGeometria(coordenadas[i-1][1],coordenadas[i-1][0],coordenadas[i][1],coordenadas[i][0]);
127              distancias[i]=distancia;
128              datos.push({distancia: +distancias[i], altura: +alturas[i]});
129          }
130      }
131  }
132
```

Ilustración 73.- Función JavaScript que obtiene los valores de la altura para cada coordenada

Posteriormente, se dibuja el gráfico en la página donde se encuentra el mapa o la información del camino. El gráfico se forma en una imagen *SVG*, empelando para ello la librería *d3.js*.

```
function dibujaGrafico(datos,id,color){
    var svgWidth = 600, svgHeight = 400;
    var margin = { top: 20, right: 20, bottom: 40, left: 50 };
    var width = svgWidth - margin.left - margin.right;
    var height = svgHeight - margin.top - margin.bottom;
    var svg = d3.select(id)
        .attr("viewBox",'0 0 600 400')
        .attr("preserveAspectRatio", "xMinYMid");
    var g = svg.append("g")
    .attr("transform",
      "translate(" + margin.left + "," + margin.top + ")"
    );
    var x = d3.scaleLinear().rangeRound([0, width]);
    var y = d3.scaleLinear().rangeRound([height, 0]);
    var line = d3.line().curve(d3.curveBasis).x(function(d) { return x(d.distancia)}).y(function(d) { return y(d.altura)});
    x.domain(d3.extent(datos, function(d) { return d.distancia }));
    y.domain([d3.min(datos,function(d){return d.altura}),d3.max(datos,function(d){return d.altura})+20]);

    g.append("g")
    .attr("transform", "translate(0," + height + ")")
    .call(d3.axisBottom(x))
    .append("text")
    .attr("fill", "#000")
    .attr("transform", "rotate(0)")
    .attr("x", 520)
    .attr("dy", "2.7em")
    .attr("text-anchor", "end")
    .text("Distancia (km)");
    g.append("g")
    .call(d3.axisLeft(y))
    .append("text")
    .attr("fill", "#000")
    .attr("transform", "rotate(-90)")
     .attr("x", -20)
     .attr("dy", "-3.5em")
    .attr("text-anchor", "end")
    .text("Altura (m)");
    g.append("path")
     .datum(datos)
     .attr("fill", "none")
     .attr("stroke", color)
     .attr("stroke-linejoin", "round")
     .attr("stroke-linecap", "round")
     .attr("stroke-width", 3)
     .attr("d", line);
}
```

Ilustración 74.- Función que dibuja el perfil

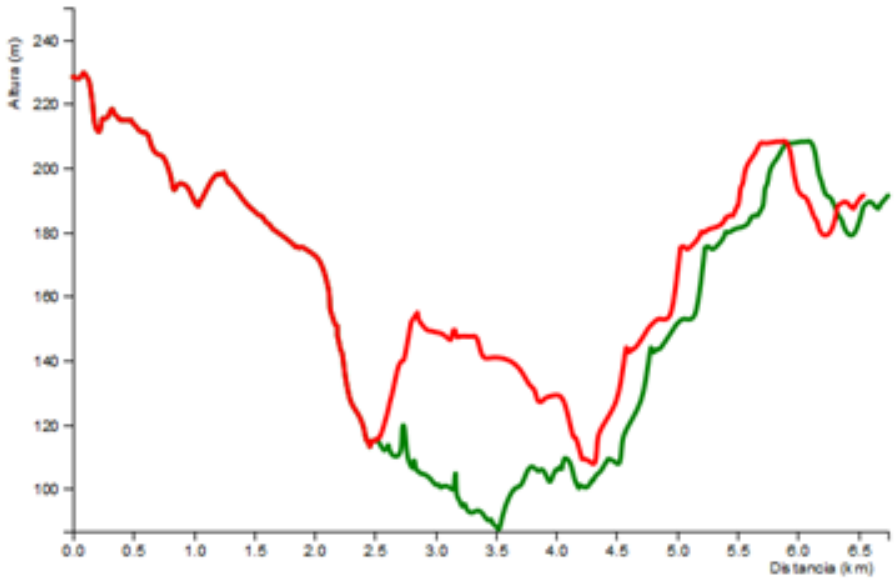


Ilustración 75.- Detalle del perfil topográfico

7.7 Capas en el mapa

En este apartado se expondrán las diferentes capas de información que se muestran en el mapa. Estas capas, no solo tratan de mostrarle información al usuario, sino que también pretenden guiarle en la interacción con el mismo. Además, se detallarán las diferentes capas que se pueden cargar en el mapa.

7.7.1 La red

El objetivo de esta capa es mostrar todos los caminos registrados en la base de datos para permitir al usuario poder crear sus propios senderos. De esta manera, el usuario sabe sobre qué zonas del mapa puede construir sus propios senderos, pues el mapa queda marcado con todos los caminos.

Sobre esta capa, se carga una capa transparente con todos los caminos, lo que estos no se visualizan en el mapa. Mediante esa capa se puede conocer la información de cada camino, así como los vértices inicial y final de cada tramo de la red. Esta capa tiene gran importancia en el desarrollo del proyecto, pues gracias a ella se pueden obtener los vértices cuando el usuario clica sobre el mapa los puntos de un sendero que desea construir. Sobre esta capa se programa el evento que se encarga de comenzar el procedimiento del *enrutamiento*. Es importante señalar que la aplicación permite filtrar tanto los caminos y puntos de la red, como los puntos de interés, por municipios.

```
112  function cargaCaminos(municipio){
113      estiloRed=getDiseño(4);
114      var data='rutas=true&municipio='+municipio;
115      var respuesta=consultaAjax("MuestraRed.php",data);
116      var estiloCaminos = {"color": estiloRed.color,"weight": estiloRed.anchura,"opacity": estiloRed.opacidad};
         return L.geoJSON(JSON.parse(respuesta),{style:estiloCaminos});
118  }
119
120  function cargaInformacionCaminos(municipio){
121      var data='rutas=false&municipio='+municipio;
122      var respuesta=consultaAjax("MuestraRed.php",data);
123      var estiloCaminos = {"color": "black","opacity": 0,"weight": 50};
         return L.geoJSON(JSON.parse(respuesta),{style:estiloCaminos});
125  }
```

Ilustración 76.- Funciones JavaScript que muestran la red

```
13  if(isset($rutas) && isset($municipio)){
14      if(strcmp($rutas, "false")===0){
15          if($municipio==0){
16              $sql="SELECT rn.id ,rn.old_id ,r.clave, st_asGeoJson(rn.the_geom) as the_geom, rn.distancia,r.nombre,r.numero,r.kilometros as distancia_tramo
17                  FROM ruta_noded as rn
18                  INNER JOIN ruta as r ON rn.old_id=r.id
19                  WHERE rn.source!=rn.target
20                  ORDER BY rn.id;";
21          }else{
22              $sql="SELECT rn.id ,rn.old_id ,r.clave,st_asGeoJson(rn.the_geom) as the_geom, rn.distancia,r.nombre,r.numero,r.kilometros as distancia_tramo,pi.municipio,pf.municipio
23                  FROM ruta_noded as rn
24                  INNER JOIN ruta as r ON rn.old_id=r.id
25                                  INNER JOIN punto as pi ON r.punto_inicial=pi.id INNER JOIN punto as pf ON r.punto_final=pf.id
26                  WHERE rn.source!=rn.target
27                                  AND pi.municipio=$municipio OR pf.municipio=$municipio
28                  ORDER BY rn.id;";
29          }
30
31      }else{
32          if($municipio==0){
33              $sql="SELECT st_asGeoJson(st_union(the_geom)) AS the_geom FROM ruta";
34          }else{
35              $sql="SELECT st_asGeoJson(st_union(r.the_geom)) AS the_geom
36                  FROM ruta as r
37                                  INNER JOIN punto as pi ON r.punto_inicial=pi.id INNER JOIN punto as pf ON r.punto_final=pf.id
38                  WHERE pi.municipio=$municipio OR pf.municipio=$municipio;";
39          }
40
41      }
```

Ilustración 77.- Fragmento del fichero PHP que obtiene los caminos de la red

7.7.2 Puntos de la red

También, otro de los elementos que se muestra en el mapa son los puntos de red. Estos puntos se van activando o desactivando según el nivel de zoom en el que se encuentra el mapa. Cuanto mayor sea el nivel de zoom, mayor número de puntos se visualizarán. El criterio seguido para dibujar los puntos es el nivel de cercanía al núcleo urbano, un campo de la tabla *punto*. Asimismo, de cada punto se puede visualizar su información asociada mediante un *popup* que se activa al clicar sobre él.

```
function eventoZoomCargaPuntos(){
    mapa.on('zoomend', function (e) {
        if(tooltips.length!==0){
            borraArray(tooltips);
        }
        compruebaCapas(mapa);
        if(arrayCoordenadas.length!==0 && arrayDistancia.length!==0 && (interaccionSenderos===false || interaccionMapa===false || interaccionFormulario===false)){
            muestraDistanciaOriginal();
        }
        if(arrayCoordenadas[1]!==undefined && arrayDistancia[1]!==undefined && (interaccionSenderos===false || interaccionMapa===false || interaccionFormulario===false)){
            muestraDistanciaAlternativa();
        }
    });
}
```

Ilustración 78.- Función JavaScript que maneja el evento de zoom

```
function compruebaCapas(mapa){
    console.log(mapa.getZoom());
    if(mapa.getZoom()>=10 && mapa.getZoom()<12){
        if(puntosCapaAlta!=null){
            eliminaCapaAlta(mapa);
        }
        if(puntosCapaMedia!=null){
            mapa.removeLayer(puntosCapaMedia);
            visibleCapaMedia=false;
        }

    }
    if(mapa.getZoom()<=13 && visibleCapaAlta===true){
        if(puntosCapaAlta!=null){
            eliminaCapaAlta(mapa);
        }
    }

    if(mapa.getZoom()>=14 && mapa.getZoom()<16){
        if(visibleCapaAlta===false){
            cargaCapaAlta(mapa);
        }
    }
    if(mapa.getZoom()<=15 && visibleCapaMedia===true){
        if(puntosCapaMedia!=null){
            eliminaCapaMedia(mapa);
        }
    }
    if(mapa.getZoom()>=16 ){
        if(visibleCapaMedia===false){
            cargaCapaMedia(mapa);
            if(visibleCapaAlta===false){
                cargaCapaAlta(mapa);
            }
        }
    }

}
```

Ilustración 79.- Función JavaScript que carga y elimina los puntos

7.7.3 Opacidad de las ortofotos

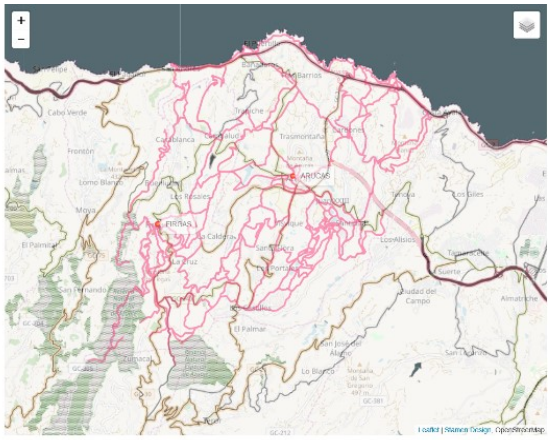
La aplicación web permite al usuario poder visualizar al mismo tiempo varias capas, así como poder cambiar la capa base del mapa. Por tanto, la vista del mapa cuenta con cuatro capas base (fotografía aérea, fotografía urbana, mapa base y capa en blanco) y tres capas transparentes (mapa topográfico, mapa de terreno y callejero).

Para poder interactuar con las capas, se ha creado un selector que permite al usuario cambiar la capa base y combinarla con las distintas capas transparentes, pudiendo modificar su nivel de trasparencia. Cuando se selecciona la capa en blanco, el nivel de opacidad se las capas transparentes se pone al máximo.

```
228  function setOpacidadTopografico(){
229      topo.setOpacity(($("#opacidad-topografico").val())/100);
230  }
231  function setOpacidadLidar(){
232      lidar.setOpacity(($("#opacidad-lidar").val())/100);
233  }
234  function setOpacidadCallejero(){
235      callejero.setOpacity(($("#opacidad-callejero").val())/100);
236  }
237
238  function ponerOpacidadMaxima(){
239      $("#opacidad-topografico").val(100);
240      $("#opacidad-lidar").val(100);
241      $("#opacidad-callejero").val(100);
242      topo.setOpacity(1);
243      lidar.setOpacity(1);
244      callejero.setOpacity(1);
245  }
246  function ponerOpacidadNormal(){
247      $("#opacidad-topografico").val(50);
248      $("#opacidad-lidar").val(50);
249      $("#opacidad-callejero").val(50);
250      topo.setOpacity(0.5);
251      lidar.setOpacity(0.5);
252      callejero.setOpacity(0.5);
253  }
```

Ilustración 80.- Funciones JavaScript que controlan las capas del mapa

Fotografía aérea más mapa topográfico y modelo del terreno



Mapa base más callejero

Ilustración 81.- Transparencia de las capas

7.7.4 Puntos de interés

También, se pueden ver en el mapa los puntos de interés, cada uno con un icono que representa la temática del punto. En el menú que se presenta sobre el mapa se puede encontrar un selector que permite al usuario activar o desactivar los puntos de interés que desee.

Asimismo, se puede obtener información clicando sobre el icono del punto de interés. Al igual que con los puntos de la red, aparece un *popup* con la información vinculada al punto en cuestión.

7.8 Visualización de la información

7.8.1 Páginas con contenido

La visualización de todo el contenido relacionado con los puntos de interés y los caminos es otro de los aspectos que se desarrollan en este proyecto. Simplemente se trata de mostrar por pantalla de forma organizada la información alojada en la base de datos, con el objetivo de que el usuario pueda consultarla. La programación de filtros de búsqueda y de la paginación han sido elementos claves para garantizar que la información se muestra de forma clara y sencilla.

```
<nav>
    <ul class="pagination justify-content-end">
      <?php
        if($pagina!==1){
            $pagina1=$pagina-1;
            if($pagina===2){
                $pagina0=$pagina1;
            }else{
                $pagina0=$pagina1-1;
            }
            echo "<li class='page-item'>
                    <a class='page-link' href='VerCaminos.php?page=$pagina0' aria-label='Previous'>
                    <span aria-hidden='true'>&laquo;</span>
                    <span class='sr-only'>Anterior</span>
                    </a>
                  </li>"
                    . "<li class='page-item'><a class='page-link' href='VerCaminos.php?page=$pagina1'>$pagina1</a></li>";
        }
        echo "<li class='page-item active'><a class='page-link ' href='VerCaminos.php?page=$pagina'>$pagina</a></li>";
        if($pagina!=$paginas){
            $siguiente=$pagina+1;
            if($siguiente<$paginas){
                echo "<li class='page-item'><a class='page-link' href='VerCaminos.php?page=$siguiente'>$siguiente</a></li>";
            }
            $siguiente=$pagina+2;
            if($siguiente<$paginas){
                echo "<li class='page-item'><a class='page-link' href='VerCaminos.php?page=$siguiente'>$siguiente</a></li>";
            }
            $siguiente=$pagina+3;
            if($siguiente<$paginas){
                echo "<li class='page-item'><a class='page-link' href='VerCaminos.php?page=$siguiente'>...</a></li>";
            }
            echo "<li class='page-item'><a class='page-link' href='VerCaminos.php?page=$paginas'>$paginas</a></li>";
            $pagina2=$pagina+1;
            echo "<li class='page-item'>
                    <a class='page-link' href='VerCaminos.php?page=$pagina2' aria-label='Next'>
                      <span aria-hidden='true'>&raquo;</span>
                      <span class='sr-only'>Posterior</span>
                    </a>
                  </li>
                </ul>";
        }
      ?>
</nav>
```

Ilustración 82.- Detalle del código que se encarga de la paginación

```php
public static function filtroBusquedaCaminos(){
    echo "<form name='busquedaCaminos' class='form-inline' method='POST' id='formularioFiltro'>
                <select name='municipio' id='input-municipio' class='custom-select' onchange='filtraMunicipioCaminos()'>
                    <option value='0'>TODOS LOS MUNICIPIOS</option>";

                    $sql="SELECT DISTINCT m.id,m.nombre FROM municipio as m INNER JOIN punto as p ON m.id=p.municipio ORDER BY m.nombre";
                    $municipio=  Persistencia::ejecuta($sql);
                    if($municipio!=0){
                        foreach($municipio as $value){
                            echo "<option value='".$value["id"]."'>".$value["nombre"]."</option>";
                        }
                    }
    echo "</select>

            </form>";
}
```

Ilustración 83.-Función PHP que crea el filtro de la página que muestra los caminos

7.8.2 Visualización de información a tiempo real

Por otra parte, en las páginas relacionadas con la red se puede observar como se muestra información a tiempo real, es decir, que al pasar el ratón sobre la red se va actualizando una tabla con información. Esto se logra gracias a la programación del manejo de un evento llamado *mousemove*, que se encarga de obtener la información de la capa oculta de la red y mostrarla en una tabla.

```
185    infoRed.on('mousemove',function(event){
186        muestraInformacion(event);
187    });
```

Ilustración 84.-Detalle de la función que gestiona el evento mousemove

7.8.3 Información de los senderos construidos por el usuario

Cada vez que el usuario crea un sendero se muestra el perfil topográfico y una tabla informativa con datos que pueden ser de utilidad para el usuario, como la distancia del sendero y el tiempo estimado para recorrerlo.

El cálculo del tiempo estimado para recorrer un sendero se ha obtenido a partir de la fórmula del cálculo de la posición de un movimiento rectilíneo uniforme, suponiendo que la velocidad de una persona al caminar es constante:

$$e = e_0 + vt$$

Siendo e y e_0 la distancia, v la velocidad y t el tiempo. El cálculo se realiza suponiendo que la velocidad media de una persona al caminar es de 5.3 *km/h* [11].

```
function muestraInformacionSenderoSimple(distancia){
    var tiempo=obtenTiempo(distancia);
    var tabla='<div id="informacion-sendero"><div class="textoInformacion"><h6><strong>Información del sendero</strong></h6>'+
            '</div><div class="table-responsive"><table class="table table-striped" id="tabla-detalles" style><thead><tr><th>Distancia del sendero</th><th>Tiempo estimado</th></tr>'+
            '</thead><tbody><tr><td>'+(Math.round(distancia*1000)/1000).toFixed(2)+' km</td><td>'+Math.round(tiempo)+' min</td></tr></tbody></table></div></div>';
    $("#seccion-informacion-sendero").html(tabla);
}

function muestraInformacionSenderos(distanciaCorto, distanciaLargo){
    var tiempoCorto=obtenTiempo(distanciaCorto);
    var tiempoLargo=obtenTiempo(distanciaLargo);
    var tabla='<div id="informacion-sendero"><div class="textoInformacion"><h6><strong>Comparativa de los senderos</strong></h6></div>'+
            '<div class="table-responsive"><table class="table table-striped" id="tabla-detalles"><thead><tr><th>Sendero</th><th>Distancia</th><th>Tiempo estimado</th></tr></thead>'+
            '<tbody><tr><td>Corto</td><td>'+(Math.round(distanciaCorto*1000)/1000).toFixed(2)+' km</td><td>'+Math.round(tiempoCorto)+' min</td></tr><tr><td>Largo</td><td>'
            +(Math.round(distanciaLargo*1000)/1000).toFixed(2)+' km</td><td>'+Math.round(tiempoLargo)+' min</td></tr></tbody></table></div></div>';
    $("#seccion-informacion-sendero").html(tabla);
}

function obtenTiempo(distancia){
    var velocidad=5.3; //km/h
    return (distancia/velocidad)*60;
}
```

Ilustración 85.- Funciones JavaScript que muestran los detalles del sendero construido

7.9 Sesión de usuario y página de configuración

El motivo del desarrollo de una página de configuración parte de la necesidad del usuario administrador del sitio web. Como solo es una funcionalidad que podrá desempeñar este usuario, ha resultado necesario crear una pestaña de inicio de sesión y programar todo lo relativo a la sesión de usuario en *PHP*.

7.9.1 Sesión de usuario e inicio de sesión

El inicio de sesión es un proceso bastante sencillo, que consiste en que al rellenar los campos del formulario de inicio de sesión, se compruebe en la base de datos si las credenciales que el usuario ha introducido son válidas.

```php
class Usuario{
    public static function session_start(){
        if(session_status () === PHP_SESSION_NONE){
            session_start();
        }
    }
    public static function login($usuario, $pass){
        self::session_start();
        $sql="SELECT * FROM usuario WHERE nombre='$usuario' AND contraseña=MD5('$pass')";
        $result = pg_query(Persistencia::getConexion(),$sql);
        if(pg_num_rows($result)>0){
            $row= pg_fetch_all($result);
            $_SESSION['user']=$row[0];
            return true;
        }
        return false;
    }

}
```

Ilustración 86.- Clase PHP que se encarga de todo lo relacionado con el usuario, especialmente del inicio de sesión

Una vez hecho este proceso, se activa la pestaña de usuario (siempre y cuando este usuario identificado sea administrador), donde se le da acceso al usuario a la página de configuración. Es importante destacar, que mientras la sesión de usuario esté iniciada, se le mostrará al usuario la pestaña de su perfil gracias a la variable *$_SESSION*.

```php
if(isset($_SESSION['user'])&&$_SESSION['user']['tipo']==1){
    $barra.='<button type="button" id="dropdownMenu1" data-toggle="dropdown" class="btn btn-outline-secondary dropdown-toggle">'
            .$_SESSION['user']['nombre'].'<span class="caret"></span></button><ul class="dropdown-menu dropdown-menu-right mt-2"><a class="dropdown-item" href="Configuracion.php">Configuración</a>'
            . '<a class="dropdown-item" href="CerrarSesion.php">Cerrar sesión</a></li></ul>';
```

Ilustración 87.- Detalle de la activación de la pestaña de usuario

Como es lógico, también se tiene programada la función de cerrar sesión, que no es más que desvincular el usuario que ha iniciado la sesión de la variable *$_SESSION*.

7.9.2 Página de configuración

La página de configuración pone al usuario administrador dos paneles, perfectamente diferenciados, en los que se le permite modificar todo lo relacionado con la visualización de las

líneas de los senderos y caminos, así como los iconos. En términos programáticos, se trata de un formulario que recoge los campos necesarios para modificar esos parámetros.

Tanto en la base de datos, tal y como se comentó anteriormente, existen dos tablas que almacenan la información relacionada a estas cuestiones de diseño y estética de la página. Todo cambio que realice el usuario administrador, en aspectos relativos al diseño y estética, supondrá una actualización en las tablas correspondientes de la base de datos.

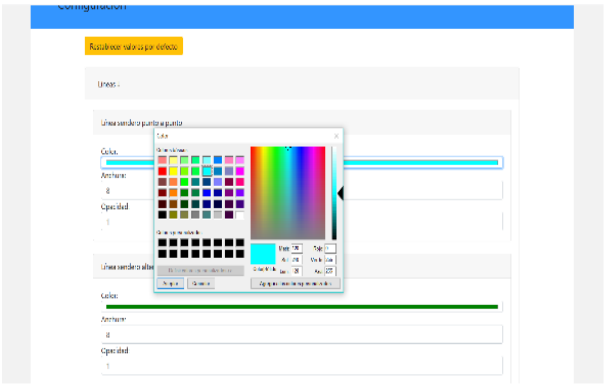
En lo relativo a los iconos, la tabla que se encarga de este asunto simplemente se ocupa de almacenar la dirección del servidor donde se encuentra la imagen del icono. En definitiva, cuando el administrador sube una imagen, esta se sube al servidor, pero es en la base de datos donde se queda registrado el lugar donde esta se aloja.

En este apartado, resulta interesante destacar el uso de la variable *$_FILES* de *PHP*, pues es la que permite la carga de ficheros al servidor mediante formularios *HTML*.

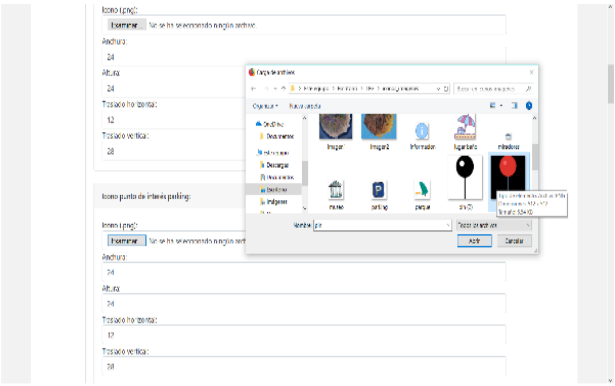
Como en la mayoría de las páginas de configuración, se da la posibilidad al administrador a retornar los parámetros de configuración a los valores por defecto.

```php
<?php
    include_once "Persistencia.php";
    include_once "funciones/FuncionesConfiguracion.php";
    $conn=Persistencia::getConexion();
    $rutas=array();
    if(isset($_POST['defecto'])){
        FuncionesConfiguracion::defectoEstilo("#FF0000", 8, 1, 1);
        FuncionesConfiguracion::defectoEstilo("#008000", 8, 1, 2);
        FuncionesConfiguracion::defectoEstilo("#230D84", 8, 1, 3);
        FuncionesConfiguracion::defectoEstilo("#FF3377", 3, 0.5, 4);
        FuncionesConfiguracion::defectoEstilo("#808B96", 7, 1, 5);
        FuncionesConfiguracion::defectoIcono();
        Persistencia::cerrar();
        header("Location: Configuracion.php?status=1");
    }

    for($id=1;$id<=FuncionesConfiguracion::getCountIconos();$id++){
        $rutas[$id]=FuncionesConfiguracion::compruebaDestinoFichero($id);
    }

    if(FuncionesConfiguracion::compruebaFichero()){
        if(FuncionesConfiguracion::compruebaFormato() && FuncionesConfiguracion::compruebaVariablesEstilo() && FuncionesConfiguracion::compruebaVariableIcono()
            $valores;
            for($id=1;$id<=FuncionesConfiguracion::getCountEstilo();$id++){
                $valores=FuncionesConfiguracion::obtenValoresEstilo($id);
                FuncionesConfiguracion::actualizaEstilo($valores[0],$valores[1],$valores[2],$id);
            }
            for($id=1;$id<=FuncionesConfiguracion::getCountIconos();$id++){
                $valores=FuncionesConfiguracion::obtenValoresIconos($id);
                FuncionesConfiguracion::actualizaIcono($rutas[$id],$valores[0],$valores[1],$valores[2],$valores[3],$id);
                if(isset($_FILES["imagen$id"])){
                    move_uploaded_file($_FILES["imagen$id"]['tmp_name'], $rutas[$id]);
                }
            }
            Persistencia::cerrar();
            print_r($_FILES['imagen1']);
            header("Location: Configuracion.php?status=1");
        }else{
            Persistencia::cerrar();
            header("Location: Configuracion.php?status=2");
        }

    }else{
        Persistencia::cerrar();
        header("Location: Configuracion.php?status=0");
    }
```

Ilustración 88.- Fichero PHP que se encarga de la actualización de los parámetros de configuración



Configuración de las líneas



Configuración de los iconos

Ilustración 89.- Configuración de las líneas y los iconos

7.10 Visualización e interacción en dispositivos móviles

Para que la página se pudiera visualizar en los navegadores de los dispositivos móviles, se tuvieron que crear reglas *CSS* para determinadas resoluciones de pantalla y se añadieron las configuraciones necesarias para que se pudiera interactuar con los mapas.

7.10.1 Responsive

Se denomina *responsive* a la técnica de diseño web que se ocupa de que las vistas de una página web se puedan ver correctamente en diferentes dispositivos. En lo que se refiere a este proyecto, se ha realizado el *responsive* teniendo en cuenta la siguiente estructura [12]:

- Resoluciones menores de 640 píxeles de ancho.
- Resoluciones de menores de 1024 píxeles de ancho, pero mayores que 640 píxeles.
- Resoluciones mayores de 1024 píxeles.

```
@media screen and (max-width:850px) {
    #logo{
        width: 267px;
        height: 325px;
    }
   /*Footer menu*/
    .li-footer{
        margin-right: 10px;
        margin-left: 10px;
        list-style: none;
        display:block;
    }

    /*Formulario login*/
    #formulario-login{
        width:250px;
    }

    .footer-menu {
        background-color: #343a40;
        padding-top: 5px;
        padding-bottom: 20px;
    }

    .dropdown-menu dropdown-menu-right mt-2 show{
        width: 300px;
    }

    /*Formulario sendero punto a punto*/
    #formularioBusquedaRuta{
        width:300px;
    }
    #input-municipio{
        width:250px;
    }
    #input-origen{
        width:250px;
        margin-bottom:1%;
    }
    #input-destino{
        width:250px;
        margin-left:3%;
        margin-bottom: 1%;
    }
```

Ilustración 90.- Detalle de las reglas CSS dedicadas al responsive

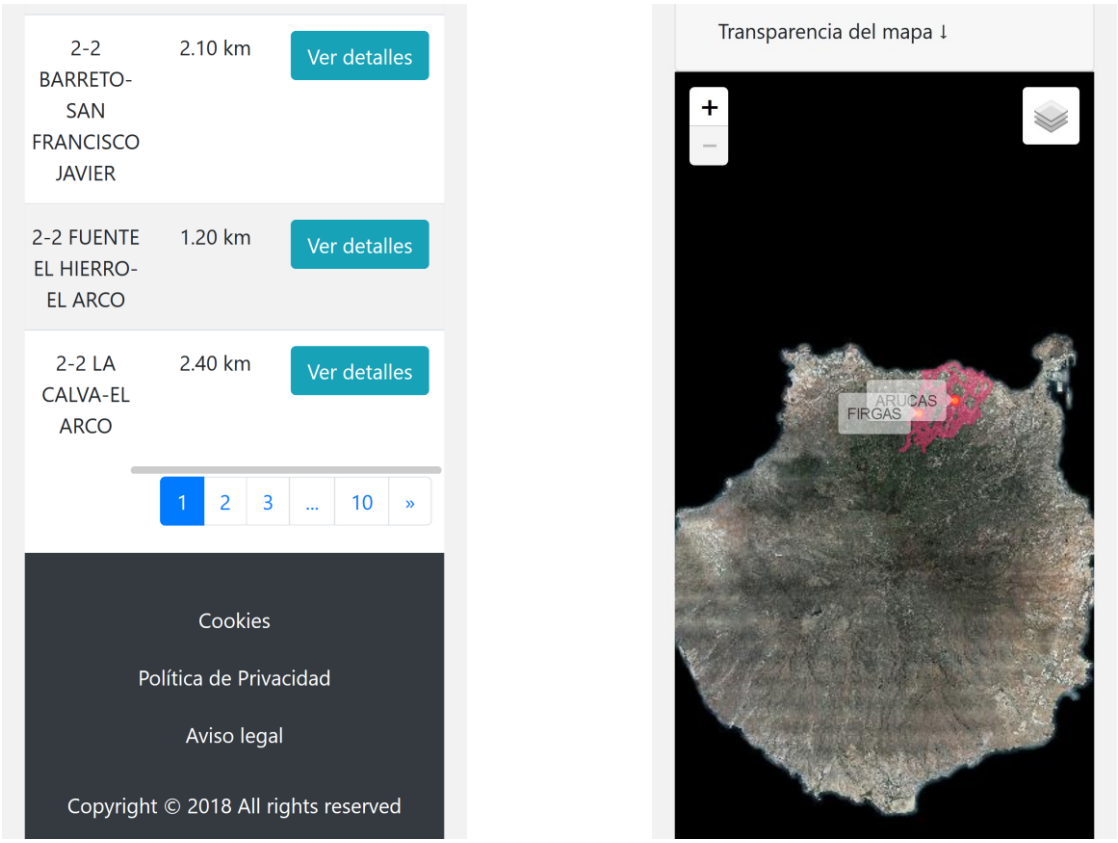


Ilustración 91.- Vista del responsive en un dispositivo móvil

7.10.2 Interacción con el mapa

La interacción con el mapa es un elemento fundamental en esta aplicación web, por lo que se debe garantizar que en los navegadores de los distintos dispositivos móviles también se pueda desempañar. Por tanto, en la sección *head* de cada página se ha añadido la siguiente línea [12]:

```
<meta name="viewport" content="initial-scale=1, content="width=device-width">
```

7.11 Despliegue y pruebas

El despliegue de la página se realizó en uno de los servidores de la ULPGC, situado en la Escuela de Ingeniería Informática. La cuenta de usuario fue creada por el tutor del proyecto para que pudiese desplegar con normalidad la aplicación. Esta fase de despliegue tuvo dos partes. La primera de ellas se centró en la instalación de una base de datos *PostgreSQL* con las extensiones necesarias (*PostGIS* y *pgRouting*). La segunda parte consistió en la transferencia de los ficheros al servidor. Asimismo, esta fase de despliegue está relacionada con la fase de pruebas, pues tanto el tutor como el co-tutor pudieron probar el producto.

7.11.1 Información sobre la cuenta de usuario en el servidor

Para comprender mejor lo que se va a detallar a continuación, conviene especificar algunos detalles relacionados con la cuenta de usuario creada en el servidor:

- El directorio donde se ubicará el proyecto es `/var/www/html/nestor`.
- Este directorio tendrá un enlace en el directorio `/home/chano` .

7.11.2 Instalación de PostgreSQL, PostGIS y pgRouting

Una vez conectados el servidor mediante *SSH* usando *Putty*, se ejecutaron una serie de comandos para instalar PostgreSQL y crear la base de datos [13]. En primer lugar, se instaló PostgreSQL:

```
sudo apt-get update

sudo apt-get install –y postgresql postgresql-contrib
```

Ilustración 92.- Instalación PostgreSQL

Seguidamente, se procedió a crear la base de datos y un usuario que tenga acceso a ella:

```
sudo –u postgres createuser –P nestor

sudo –u postgres createdb –O nestor TFTDB
```

Ilustración 93.- Creación de usuario y base de datos

Luego, se comprobó que la base de datos fue creada correctamente:

```
psql –h localhost –U nestor TFTDB
```

Ilustración 94.- Test de conexión con la base de datos

En cuarto lugar, se instaló la extensión *PostGIS* y se añadió a la base de datos:

```
sudo apt-get install –y postgis postgresql-9.3-postgis-2.1

sudo –u postgres psql –c “CREATE EXTENSION postgis;CREATE
EXTENSION postgis_topology;” TFTDB
```

Ilustración 95.- Instalación y creación de la extensión PostGIS

A continuación, se instaló y añadió la extensión *pgRouting*:

```
sudo apt-get install wget ca-certificates

wget –quiet –O
https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo
apt-key add –sudo apt-get update

sudo apt-get install postgresql-9.3-pgrouting
```

Ilustración 96. Instalación de pgRouting

Otro aspecto a tener en cuenta fue la descarga de las funciones de *PHP* para conectar con *PostgreSQL*:

```
sudo apt-get install php5-pgsql

sudo service apache2 restart
```

Ilustración 97.- Instalación de las funciones PHP para PostgreSQL

Finalmente, se cargó el último *backup* en la base de datos creada en local. Este proceso se realizó mediante la interfaz gráfica de *pgAdmin.* Para ello, se creó una conexión a la base de datos recién instalada en el servidor mediante un *SSH Tunnel.*

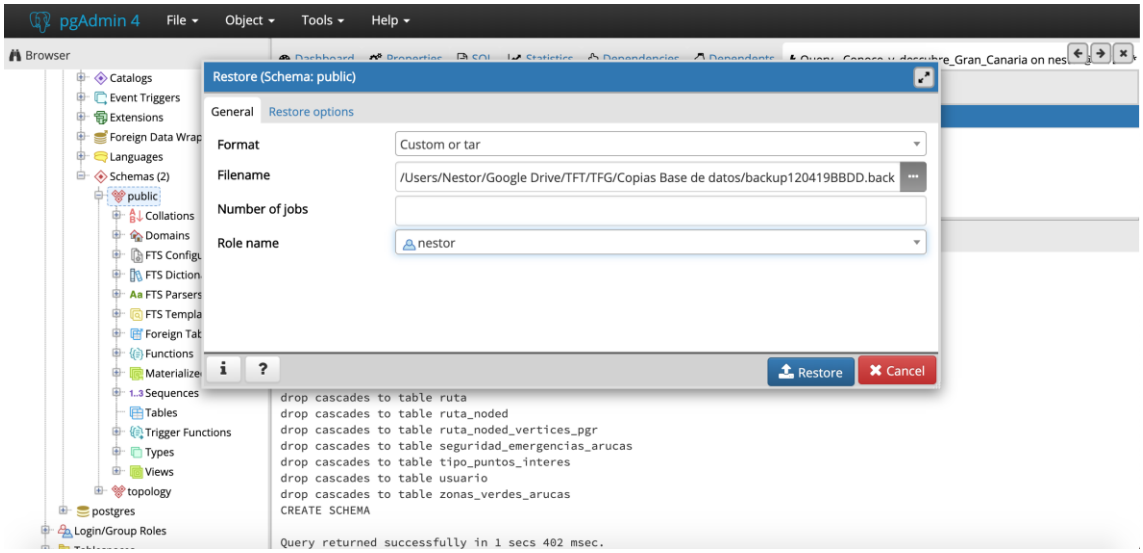


Ilustración 98.- Carga del último backup en pgAdmin

7.11.3 Transferencia de los ficheros al servidor

Esta segunda parte ha sido más sencilla de realizar, pues *Filezilla* realiza todo el trabajo. Se establece la conexión con el servidor en *Filezilla*. El programa muestra el árbol de directorios del cliente y del servidor. Solo hay que arrastrar el directorio que contiene todos los ficheros de la aplicación y esperar a que se complete la trasferencia.

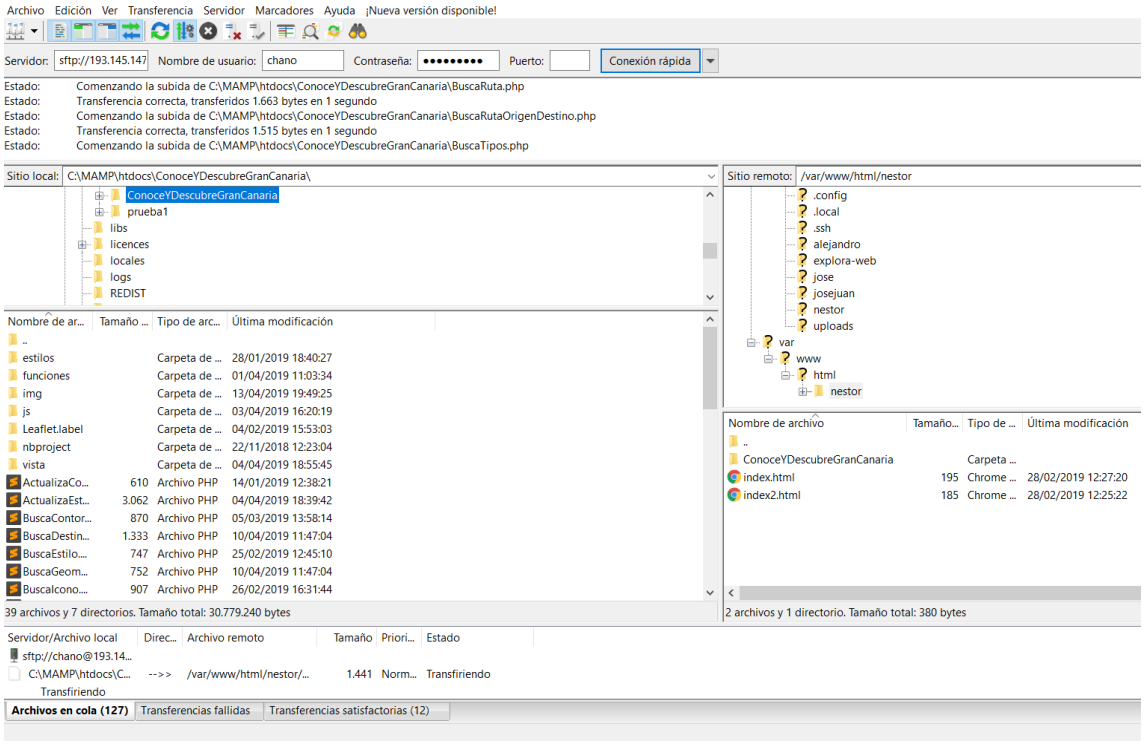


Ilustración 99.- Transferencia de ficheros con Filezilla

7.11.4 Pruebas

Una vez desplegado el proyecto en el servidor, tanto el tutor como el co-tutor pudieron acceder a él y realizar las pruebas pertinentes. Esto no quita que durante el desarrollo no se realizaran las pruebas necesarias para verificar el correcto funcionamiento de la aplicación.

Al mismo tiempo, el despliegue en un servidor permitió comprobar que se puede acceder a los datos y a la aplicación en sí de forma remota.

8 Conclusiones y trabajos futuros

8.1 Conclusiones

El desarrollo de este trabajo ha supuesto para mí una buena experiencia donde, sin lugar a dudas, no solo se han puesto en práctica muchos conocimientos estudiados en el grado universitario, sino que también me ha permitido aprender lo interesante y la utilidad que tienen hoy en día los *SIG*, así como las distintas herramientas potentes que existen hoy en día para trabajar esta realidad.

Al mismo tiempo, los objetivos se han cumplido de forma satisfactoria, pues se consiguió resolver la necesidad que más importancia le suponía a Salvador, el artífice de la idea y la persona que ha trazado todos los caminos que conforman la red. Esta necesidad consistía en poder crear una herramienta informática capaz de *enrutar* senderos dentro de una red de caminos.

También, y en la línea de lo expuesto anteriormente, ha sido bastante positivo realizar un trabajo y llevar a cabo una idea de una persona. El contacto con el cliente como principal fuente de requisitos ha supuesto además una fantástica experiencia, no solo de conocimiento, sino también profesional. En este sentido, al tratarse de un proyecto real, se ha podido trabajar la toma de decisiones en función de la prioridad de las funcionalidades y del tiempo disponible para llevarlas a cabo.

No obstante, durante el desarrollo del proyecto se llegó a la conclusión de que la red proporcionada no responde a las características de un grafo, ya que existen duplicidades en los caminos. En la *Ilustración 80* se puede ver un ejemplo de ello.

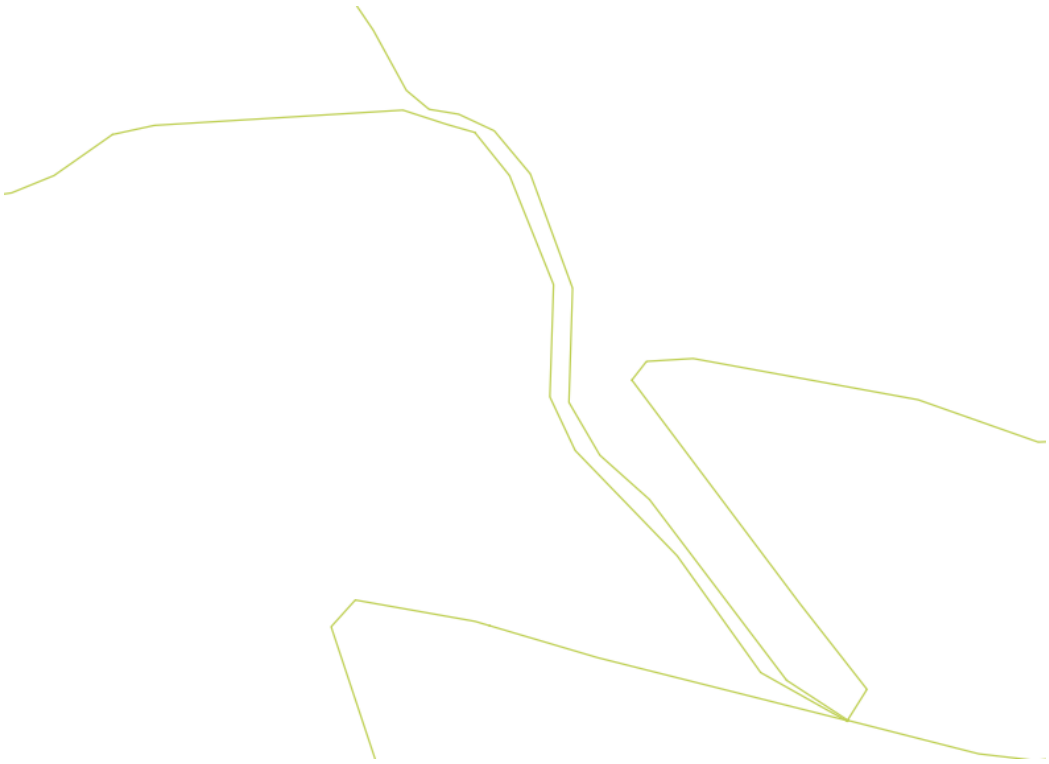


Ilustración 100.- Ejemplo de caminos duplicados

Pero a pesar de ello, la herramienta creada para la creación de senderos funciona correctamente. Lo único que se debería modificar en la aplicación *web mapping* es la estructuración y trazado de los caminos, y así eliminar los duplicados.

8.2 Trabajos futuros

La idea que abarca este proyecto es bastante grande, y en lo que se refiere este proyecto, se han desarrollado las funcionalidades más destacadas y que cubriesen las horas estipuladas para el desarrollo del trabajo de fin de título. Por lo tanto, se proponen estas líneas de trabajos futuros:

- Mejorar el trazado de la red de caminos para así poder construir los senderos sobre un grafo correctamente diseñado.
- Crear algoritmos que permitan el cálculo de información técnica, como por ejemplo, el nivel dependiente de un sendero, el tiempo que se tarda en recorrer un camino teniendo en cuenta su pendiente, entre otros.
- Crear más parámetros de configuración para que el usuario administrador pueda continuar diseñando la página a su gusto.
- Mejorar el diseño de la interfaz de usuario.
- Construir un sistema de filtros para organizar las búsquedas de información de los puntos de interés y/o servicios por categorías, tipos, temas, subtemas según el contenido.
- Mejorar el formulario que permite *enrutar* dos puntos conocidos, de tal manera que se puedan filtrar por varios municipios los puntos de la red.

9 Fuentes de información

9.1 Referencias

[1] I. Otero Pastor. "Conceptos básicos", en *Sistemas de Información Geográfica: Teoría y práctica*. Madrid: Dextra, 2016, 13-33.

[2] B. Jackson. "WordPress vs Drupal – ¿Cuál es Mejor? (Pros y Contras)". Kinstra. https://kinsta.com/es/blog/wordpress-vs-drupal/ (acceso: 28 de octubre de 2018).

[3] Anónimo. "Comparativa Drupal, Joomla y WordPress". isyourweb. https://isyourweb.com/comparativa-drupal-joomla-y-wordpress/ (acceso: 28 de octubre de 2018).

[4] J.L. García Grandes. "Cómo ver servicios WMS en Leaflet y acceder a sus datos" MappingGIS. https://mappinggis.com/2018/01/accediendo-a-los-datos-de-un-servicio-wms-con-leaflet/ (acceso: 14 de abril de 2019).

[5] *Requirement,* IEE Std. 610-1990.

[6] V. Gómez. "Arquitectura en Tres Capas". Instinto Binario. https://instintobinario.com/arquitectura-en-tres-capas/ (acceso: 12 de abril de 2019).

[7] V. Ayala. "Bases de datos", en *Sistemas de Información Geográfica. Tomo I [Recurso electrónico]*. S.l : s.n, 2012, 181-202.

[8] F. Martín Asín. "Astronomía", en *Astronomía*. Madrid: Editorial Paraninfo, 1982, 15-21.

[9] Mª F. Andrés de Araujo, "Modelización de la flexión litosférica generada por la Isla de Tenerife sobre la Isla de Gran Canaria en los últimos 4 millones de años", Tesis doctoral, Dpto. de Física, Universidad de las Palmas de Gran Canaria, Las Palmas de Gran Canaria, España, 2015. [En línea]. Disponible en: https://accedacris.ulpgc.es/handle/10553/19429

[10] Clases de Apoyo. *Cómo calcular la proyección de un punto sobre una recta*. (13 de septiembre de 2015). Acceso: 13 de febrero de 2018. [Vídeo en línea]. Disponible en: https://www.youtube.com/watch?v=lmtvRE_oCDc

[11] Anónimo. "Velocidad estándar al caminar". CaminarMas.com. http://www.caminarmas.com/velocidad-estandar-al-caminar_189.html (acceso: 4 de abril de 2019).

[12] Manz. "Responsive Web Design". Lenguaje CSS. https://lenguajecss.com/p/css/propiedades/responsive-design (acceso: 20 de marzo de 2019).

[13] J. Saints. "Howto install Postgresql and PostGIS on Ubuntu Trusty 14.04 with secure tunnel for connections". Jon Saints. http://www.saintsjd.com/2014/08/13/howto-install-postgis-on-ubuntu-trusty.html (acceso: 1 de marzo de 2019).

9.2 Bibliografía

- PHP
  - https://www.php.net/manual/es/index.php
- JavaScript, AJAX, jQuery
  - https://www.w3schools.com/
- Leaflet
  - https://leafletjs.com/reference-1.4.0.html
- PostgreSQL
  - https://www.postgresql.org/docs/
  - http://www.postgresqltutorial.com/
- PostGIS
  - https://postgis.net/documentation/
- pgRouting
  - https://docs.pgrouting.org/
- Wikipedia
  - https://es.wikipedia.org/wiki/Wikipedia:Portada

ANEXO I.- Manual de usuario

Introducción

La aplicación que a continuación se mostrará consiste en una web que permite construir senderos a partir de una red de caminos. Asimismo, da la posibilidad al usuario de consultar la información que crea oportuna sobre estos caminos. Además, el administrador de la página cuenta con una página de configuración para modificar aquellos aspectos del sitio web que considere oportuno. El acceso a la página es el siguiente:

http://193.145.147.50/nestor/ConoceYDescubreGranCanaria/

Creación de senderos punto a punto

Para la creación de senderos punto a punto es necesario acceder a la pestaña *La red -> Sendero punto a punto*. Una vez en la página, se podrá ver la siguiente vista, que nos permitirá conocer la información de la red con solo pasar el ratón sobre la misma:

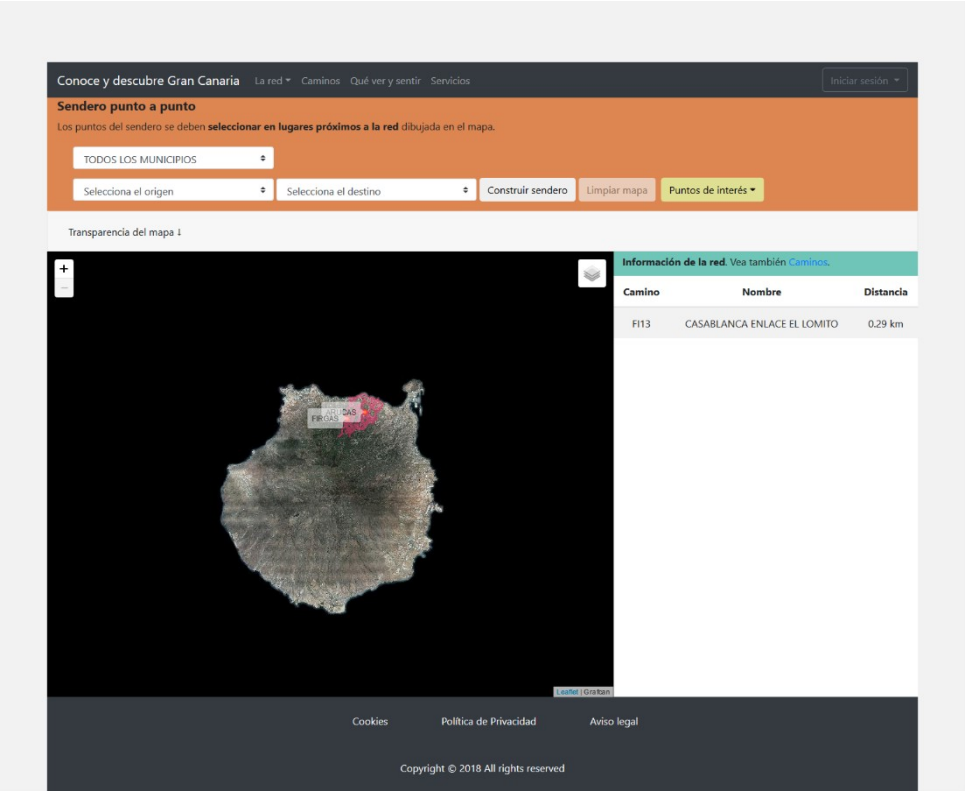


Ilustración 101.- Página Sendero punto a punto

Ahora, tenemos tres opciones de crear senderos:

1. Para crear un sendero entre dos puntos interactuando con la red simplemente es necesario hacer clic sobre cualquier punto de la red.

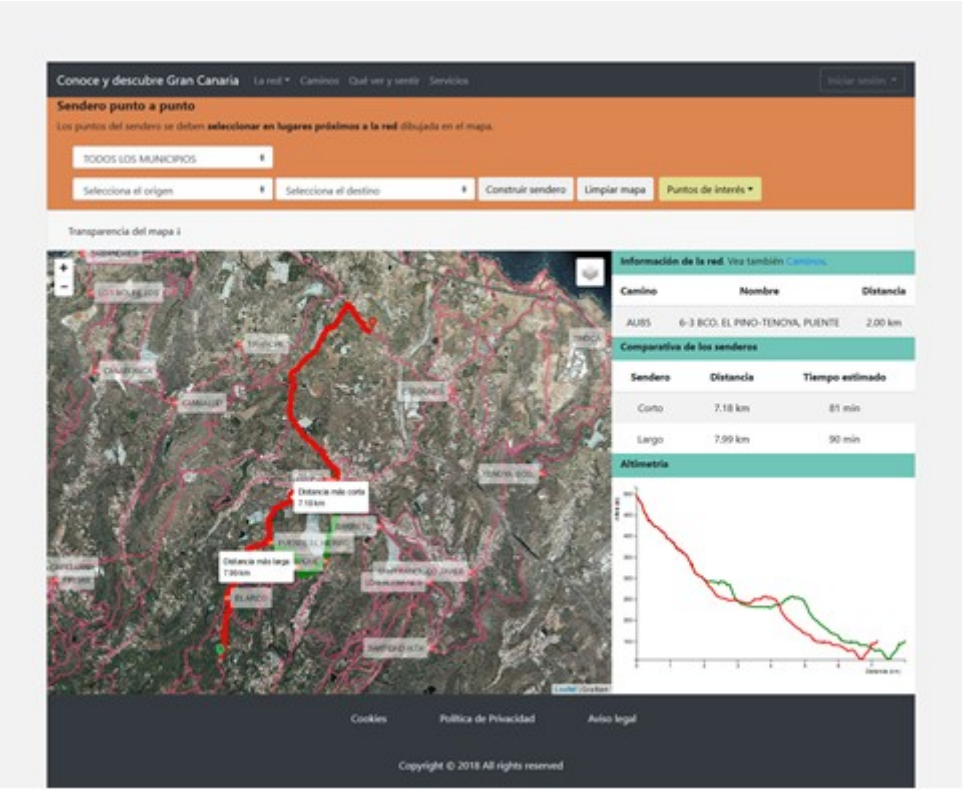


Ilustración 102.- Creación de sendero punto a punto interactuando con el mapa

2. Para crear un sendero entre dos puntos interactuando con el formulario simplemente hay que elegir los puntos en el mismo.

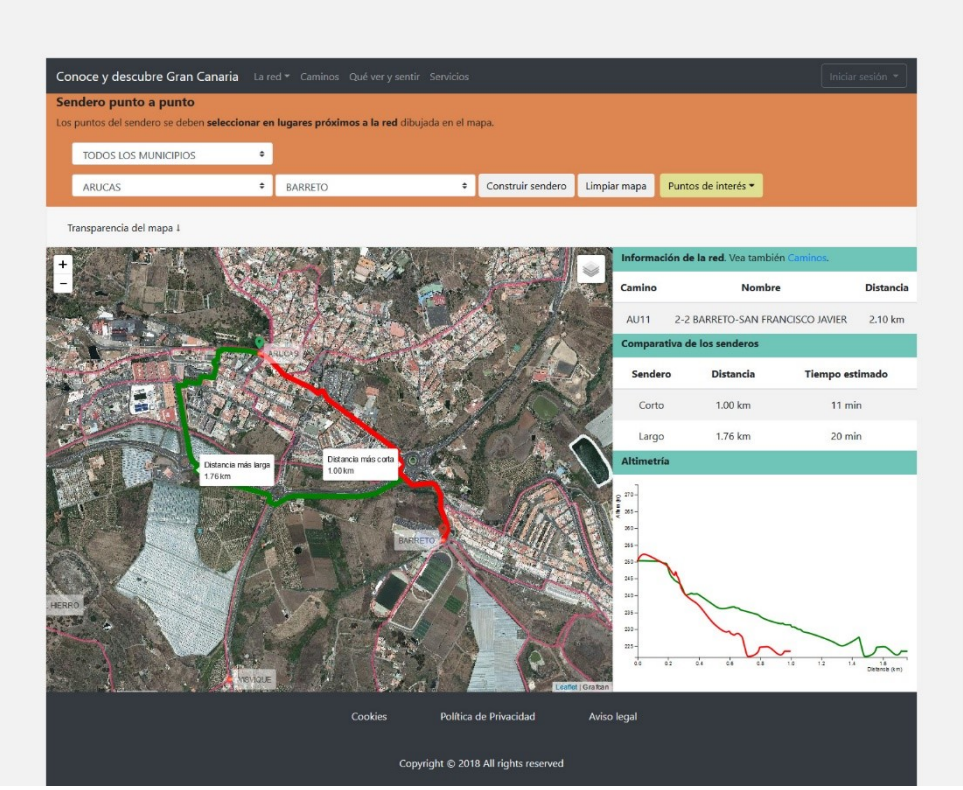


Ilustración 103.- Creación de sendero punto a punto interactuando con el formulario

3. Para crear un sendero interactuando con los puntos de la red, solo basta con clicar en el punto y pulsar en el botón que aparece en el *popup*.

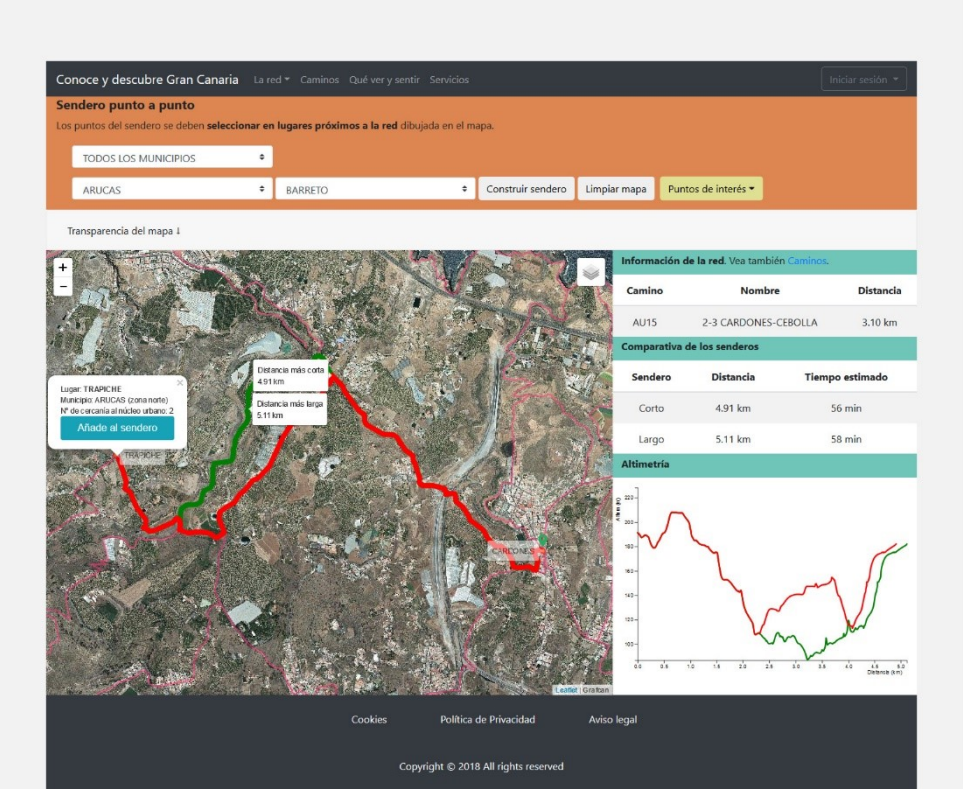


Ilustración 104.- Creación de sendero punto a punto interactuando con los puntos de la red

Si se quisiera limpiar el mapa se debe pulsar el botón *Limpiar mapa.*

Creación de senderos multipunto

En este caso, para acceder a la página que se encarga de crear los senderos multipunto, hay que acceder a la pestaña *La red -> Sendero entre muchos puntos*. Una vez dentro, nos encontraremos con una vista similar a la página *Sendero punto a punto*.

Para crear el sendero solo es necesario activar el botón *Construir Sendero* e ir seleccionando los puntos en la red que se deseen.

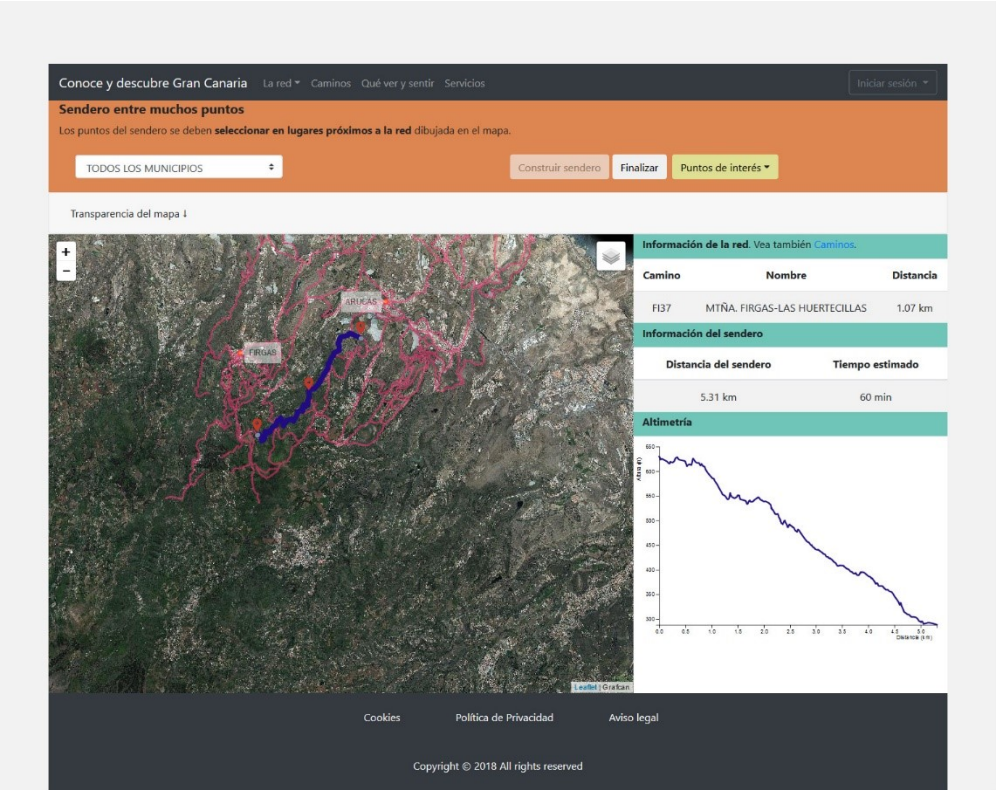


Ilustración 105.- Creación de senderos multipunto

Si se quisiera ver los caminos de un solo municipio, con cambiar el municipio que se encuentra en el selector de los municipios es suficiente.

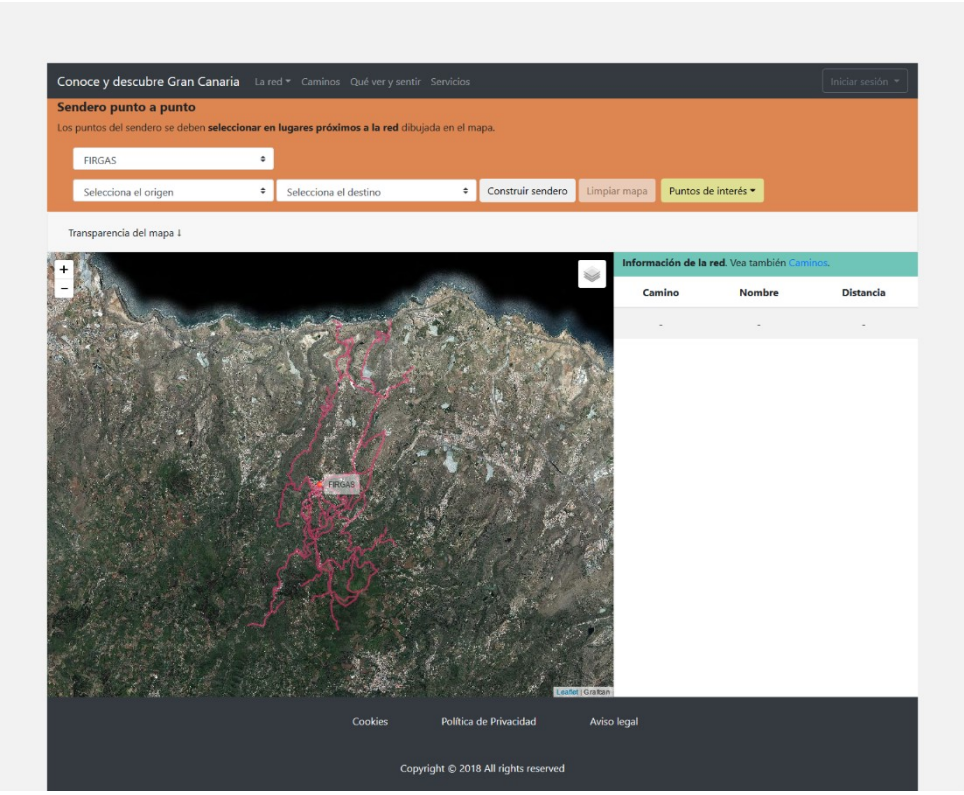


Ilustración 106.- Visualización de los caminos de un municipio

Visualización de la información

La página web dispone de tres pestañas que permiten la visualización de la información de forma ordenada. Estas son: *Caminos*, *Qué Ver y sentir*, *Servicios*. Con hacer clic en la pestaña ya se tiene acceso a la página. Una vez dentro, se mostrará, en los tres casos, una página similar a la que se mostrará a continuación. Se podrá apreciar en la parte superior un sistema de filtrado de información para que el usuario pueda ir organizando el contenido según su interés.

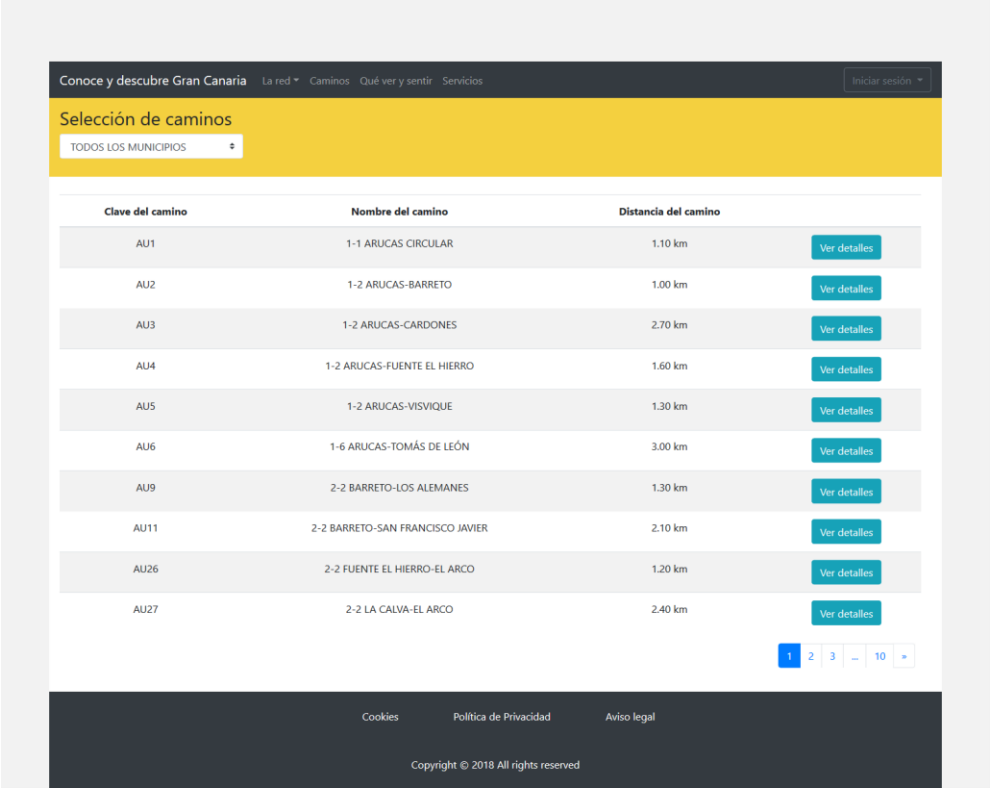


Ilustración 107.- Visualización de caminos

Como se ve en la imagen, el usuario podrá acceder al detalle de la información que está visualizando.

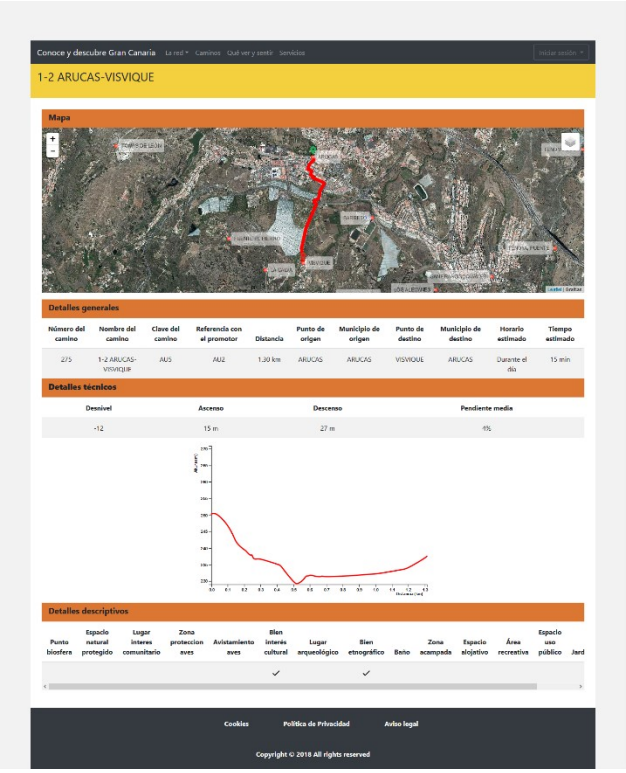


Ilustración 108.- Visualización de los detalles de un camino

Visualización de puntos de interés y modificación de las capas en el mapa

Si se desean mostrar los puntos de interés, en cualquiera de las páginas que construye senderos, solo basta con hacer clic en el botón *Puntos de interés* y seleccionar de la lista los puntos que se desean.

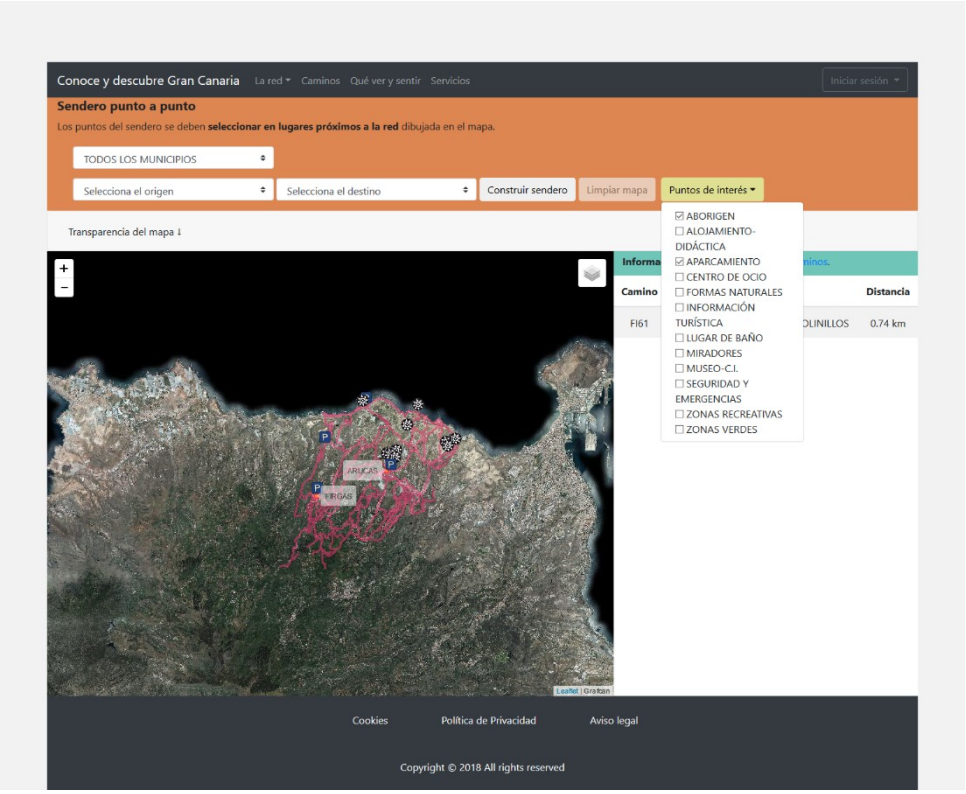


Ilustración 109.-Visualización de puntos de interés

Otra de las funcionalidades disponibles es la combinación de capas en el mapa, así como su nivel de transparencia. Sobre el mapa se puede ver un icono, en la parte superior derecha  , que permite cambiar y/o combinar las diferentes capas disponibles. Además, para modificar la opacidad de las capas transparentes hay que desplegar el menú *Transparencia del mapa*.

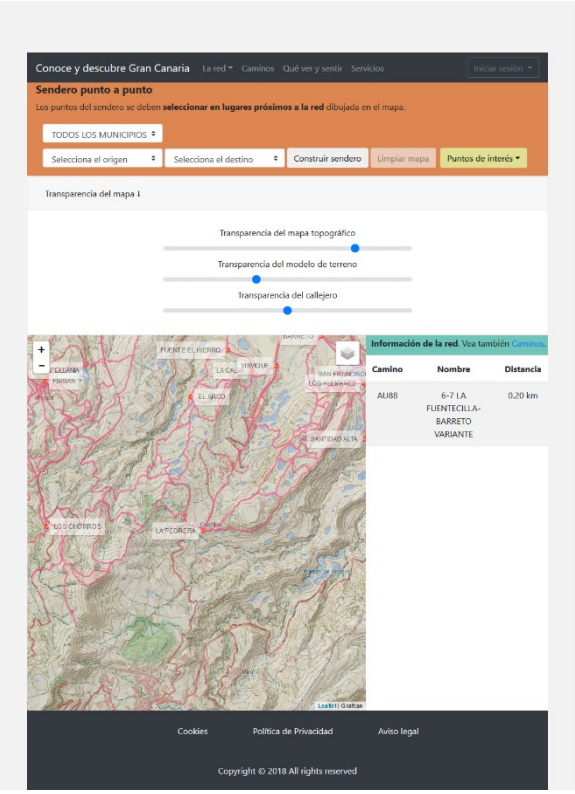
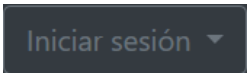


Ilustración 110.- Modificación de las capas del mapa

Inicio de sesión y página de configuración

El usuario administrador tiene la posibilidad de iniciar sesión y de acceder a la página de configuración. Solo es necesario entrar clicar en el botón *Inicio de sesión* situado en la parte superior izquierda de la barra de menú 

y completar el formulario de inicio de sesión con las credenciales.

Una vez iniciada la sesión, se puede acceder al botón de usuario, que da acceso a la página de configuración.

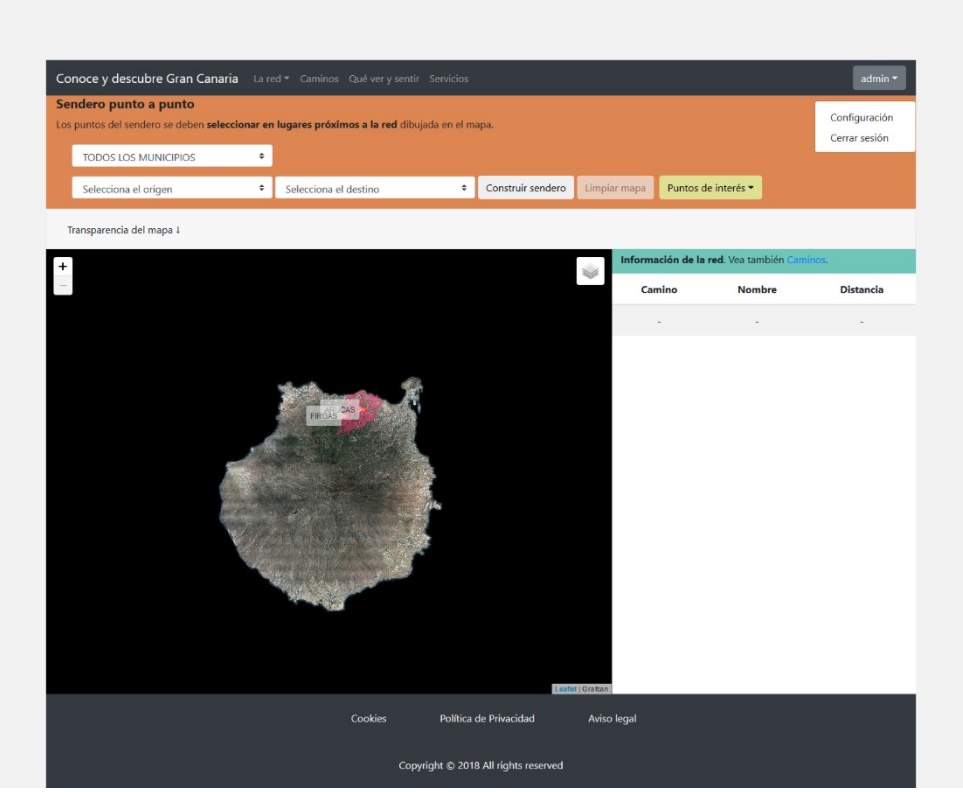


Ilustración 111.- Inicio de sesión

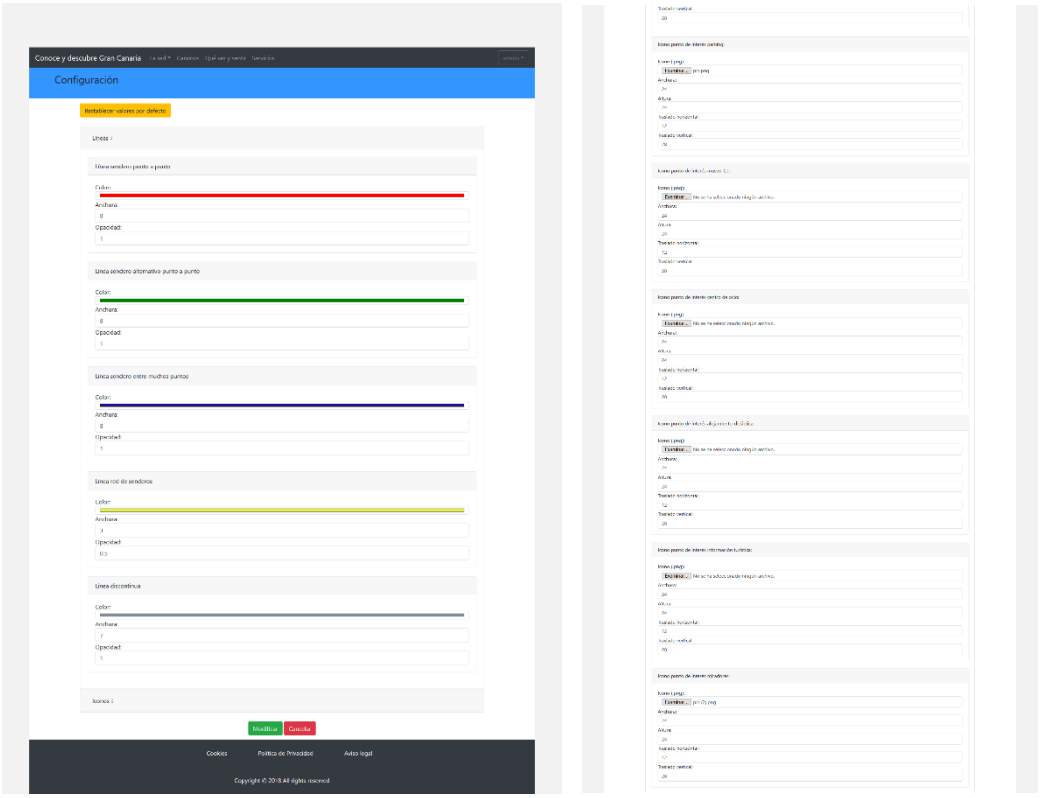


Ilustración 112.- Página de configuración

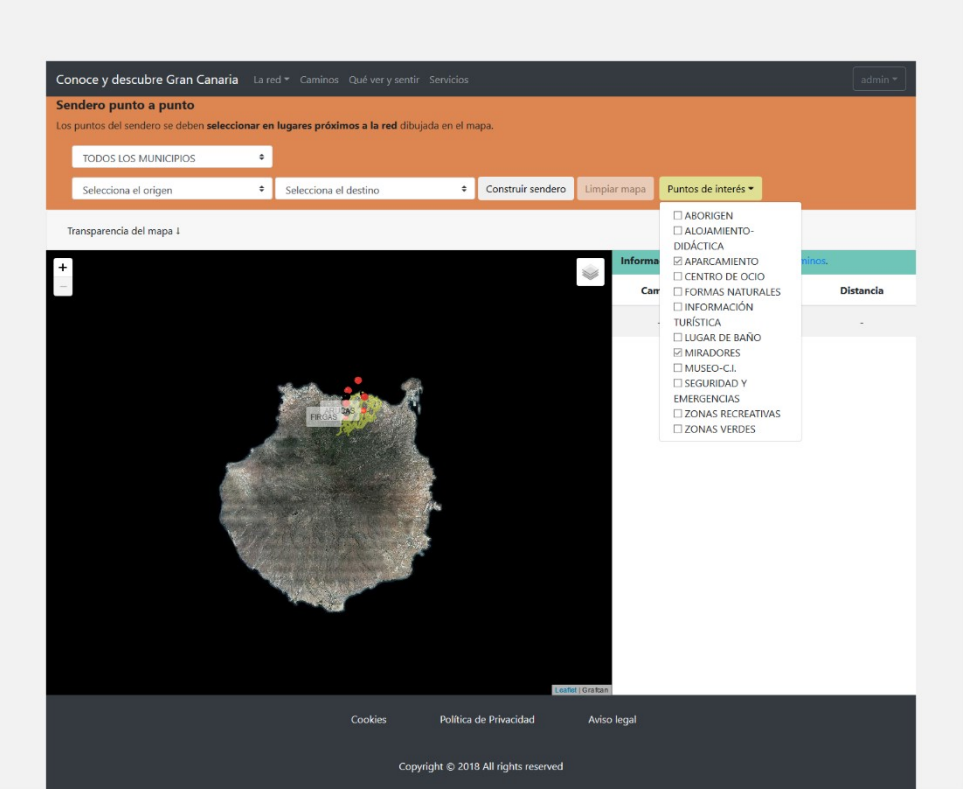


Ilustración 113.- Visualización de los cambios realizados

Enlace a demo

A continuación se proporciona un enlace a un vídeo donde se podrá visualizar una demo del trabajo realizado:

https://www.youtube.com/watch?v=b6ly-cPwFK4