Universidad Politécnica de Cartagena

IMPLEMENTACIÓN DE UN SISTEMA DE INFORMACIÓN GEOGRÁFICA CON INTERFAZ WEB EN EL CONTEXTO DE SMART CITY APLICADO A LA OPTIMIZACIÓN DE SERVICIOS MUNICIPALES

AUTOR: Alfonso Vivancos Cayuela

Tutor: José María Molina García-Pardo

2017

Índice

1 Introducción

Es evidente que la población crece cada vez más, resaltando en este crecimiento la población urbana. A raíz de tal crecimiento se están ocasionando problemas de organización social, gestión del territorio y, también, medioambientales. Dados estos problemas, es importante que las ciudades sepan actuar correctamente frente a ellos, anticipándose y cambiando el modelo actual de ciudad para que dichos problemas no sean más graves de lo que pueden llegar a ser. Por este motivo surge el nuevo modelo de ciudad inteligente, más conocido como smart city, con el objetivo de lograr una gestión eficiente en todos los sectores de la ciudad (urbanismo, transporte, servicios, energía, etc.) teniendo en cuenta los principios de Desarrollo Sostenible expuestos en el Programa 21 promovido por Naciones Unidas [1]

La concentración de cada vez más individuos en la ciudad da origen a una sobrepoblación sin precedentes, pero también se originan nuevos retos en la gestión urbana como son:

- Los recursos hídricos
- La lucha contra el gas de efecto invernadero y la contaminación del aire.
- Cuestionamientos de ciertos modos de transporte debido a la escasez de combustibles fósiles
- Los problemas sociales que con llevan a la creación de guetos
- La inseguridad
- Generación de residuos de forma excesiva
- El aumento del consumo de energía, lo que refleja que puede conducir a una degradación ambiental
- La pobreza y la exclusión.

En este contexto, por lo que las ciudades inteligentes y sostenibles está tratando de reducir el impacto ambiental, sino también para repensar los modelos de acceso a los recursos, el transporte, la gestión de residuos, aire acondicionado de edificios y sobre todo la gestión de la energía (producción, transporte, etc.).

De hecho, mientras que las ciudades ocupan hoy el 2 % de la superficie de la tierra, que son el hogar de 50 % de la población mundial, consumen el 75 % de la

energía producida y son responsables del 80 % de las emisiones CO2. Tanto la energía como la principal fuente de emisiones de CO2, la ciudad y su gente son los primeros afectados por los peligros del calentamiento global. El éxito de la transición hacia una sociedad baja en carbono se basa, por tanto, en gran medida de lo que las ciudades van a decidir. Su rápida participación es esencial para mejorar el rendimiento medioambiental de las zonas urbanas. Es por estas razones que las ciudades se consideran aspectos más destacados de la batalla contra el cambio climático. [2]

1.1 Planteamiento

El presente TFG está enmarcado dentro de la red de Cátedras que la Universidad Politécnica de Cartagena mantiene con el Ayuntamiento de Cartagena. Más concretamente, en la cátedra denominada *TICs aplicadas a la optimización de servicios municipales,* dirigida por Dr. José María Molina García-Pardo.

La creación de la Cátedra surge tras el Acuerdo Marco firmado entre ambas instituciones, y la apuesta decidida del equipo de gobierno actual del Ayuntamiento de Cartagena por colaborar con la Universidad en temas de investigación aplicada que redunden en mejoras para la ciudadanía. [3]

1.2 Objetivo

El objeto de este proyecto es el estudio de la eficiencia en la gestión de los diferentes servicios municipales y del ahorro en consumo de combustibles por diferentes causas. La mejora en la gestión de estos servicios solo se puede conseguirse con el empleo de nuevas tecnologías en los diferentes ámbitos de actuación. En resumen, convertir a Cartagena en una ciudad más **participativa**, más **sotenible**, más **eficiente** y con mejor **calidad de vida**.

Esta eficiencia en los servicios municipales repercutirá de manera directa sobre la calidad de vida urbana dentro de nuestro término municipal y su ahorro económico. Para ello se ha hehco un inventario de la infraestructura de la que dispone la ciudad de Cartagena.

Una vez se tiene una visión general de dicho inventario, la propuesto es unificar en un centro de mando del Ayuntamiento los diferentes servicios:

- Alumbrado público:

    - Cuadros de mando .
    - Farolas.
- Sistemas de riego.
- Transporte público: Posicionamiento en tiempo real.
- Cámaras de tráfico y señales luminosas (semáforos).
- Expendedores de la ORA.
- Contenedores:
    - RSU.
    - Papel y cartón.
    - Envases plásticos.

1.3 Plan de trabajo

El trabajo durante los doce meses de duración de la cátedra se puede dividir en las siguientes fases, siendo el inicio de la misma en el mes de abril:

1. Estudio y visita de los medios y sistemas de comunicación existentes en Cartagena.
2. Petición y recepción de algunos datos de posicionamiento con objeto de integración en el GIS.
3. Adaptación del lugar de trabajo a las necesidades requeridas (red, equipos, espacio de trabajo).
4. Reuniones y acuerdos con varios técnicos de los diferentes servicios requeridos para implantar.
5. Estudio e investigación de posibles soluciones hardware y software para los distintos frentes abarcables en el proyecto.
6. Diseño de solución hardware para la capa de luminarias/farolas del GIS.
7. Pruebas y estudio de diferentes GIS: ArcGIS y QGIS.
8. Reunión para compra de ArcGIS. Se estudió el presupuesto y el resutlado fue no viable para la compra de ArcGIS.
9. Decisión de elegir QGIS como plataforma desarrolladora para implantar nuestro proyecto.
10. Petición de diferentes materiales para implementación hw necesarios para el desarrollo del control de las luminarias de Cartagena.

11. Periodo de aprendizaje del software QGIS además del lenguaje de programación Python.
12. Adaptación de algunos datos de posicionamiento para su correcta visualización en mapa.
13. Implementación de distintos datos de posicionamiento: Luminarias, Cuadros de Mando, Contenedores, expendedores de la ORA, etc.
14. Desarrollo de un módulo software integrable dentro del GIS para el tracking de autobuses.
15. Desarrollo de una demo que muestra la simulación ficticia de rutas de autobuses recorriendo el municipio de Cartagena.
16. Pruebas con servidores web para implementar el proyecto en la web.

2 *Smart City*

Primeramente, cabe aclarar que Smart City es un anglicismo que en castellano significa *Ciudad Inteligente*.

Entrando más en detalle se puede definir como aquella ciudad que aplica las tecnologías de la información y de la comunicación (TIC) con el objetivo de proveerla de una infraestructura que garantice:

- Un desarrollo sostenible.
- Un incremento de la calidad de vida de los ciudadanos.
- Una mayor eficacia de los recursos disponibles.
- Una participación ciudadana activa.

Por lo tanto, son ciudades que son sostenibles económica, social y medioambientalmente. La Smart City nace de la necesidad de mantener una armonía entre estos aspectos.

2.1 Características de una *Smart City*

Para entender mejor el concepto de Smart city se detallarán las características que tienen en común este tipo de ciudades. Estas características se clasifican dentro de 5 ámbitos que son: el gobierno, la movilidad, la sostenibilidad, la población y la economía. La aplicación íntegra de todas estas características es lo que le hace a una ciudad convertirse en ciudad inteligente.

2.1.1 Gobierno

El Gobierno ocupa un papel fundamental dentro de una smart city. Éste debe ser transparente, es decir, se encarga de que todos los datos sean **accesibles** y estén **abiertos** a los ciudadanos para que la población sea participativa y esté interconectada. Para ello utiliza movimientos como el *Open Data* y *Open Government*[1] los cuales favorecen el intercambio y la aportación de datos a los ciudadanos a través de la web, estos datos son totalmente públicos con el fin de que los agentes sociales también los puedan utilizar. Es aquí donde cobran importancia las TICs las cuales ayudan a llevar a cabo una buena gestión y crecimiento de la

[1] El Gobierno Abierto u Open Government tiene como objetivo que la ciudadanía colabore en la creación y mejora de servicios públicos y en el robustecimiento de la transparencia y la rendición de cuentas.

smart city permitiendo que los agentes públicos, los agentes privados y los ciudadanos estén interrelacionados.

2.1.2 Movilidad

La movilidad de una smart city está relacionada con la sostenibilidad, la seguridad y la eficiencia de los sistemas de transporte e infraestructuras, pero también está relacionada con la **accesibilidad local**, **nacional** e **internacional**. Dentro del ámbito de la movilidad de una smart city cobran mucha importancia los PMUS[2]. Un PMUS es una agrupación de acciones que tienen la finalidad de implantar maneras de desplazamiento más sostenibles en una ciudad como caminar, utilizar la bicicleta o el transporte público. Estas acciones ayudan tanto al crecimiento económico como al medio ambiente para conseguir una mejor calidad de vida para los ciudadanos.

2-1 Rótulo genérico de PMUS

2.1.3 Sostenibilidad

Otra de las características que posee una smart city es la de ser una ciudad atractiva gracias a sus cualidades naturales y medioambientales. Para conseguir esto, una smart city debe tomar medidas correctas de gestión y protección del medio y, con ello, desarrollar una estrategia basada en las características intrínsecas del territorio potenciando los **atractivos medioambientales** y reduciendo las debilidades.

Existe un instrumento común en todos los ámbitos de una smart city llamado Programa 21 [4] que tiene un peso importante dentro de la protección y planificación medioambiental. Este proyecto se basa en la creación de una herramienta para controlar la **gestión medioambiental** de una localidad, pero su buen funcionamiento dependerá de lo sólida que sea su aplicación y del cumplimiento por parte de todos los agentes.

[2] Planes de Movilidad Urbana Sostenible

2.1.4 Población

En una smart city la población es uno de los aspectos más importantes ya que es decisivo para que ésta tenga éxito. En primer lugar, la población debe ser **participativa**, es decir, debe formar parte dentro de los procesos que establece el gobierno de la ciudad como por ejemplo en el desarrollo de la legislación o en el desarrollo de proyectos.

Y, en segundo lugar, el gobierno de una smart city debe saber que es necesario contar con la **intervención** de los ciudadanos porque estos, a nivel individual, tienen más fuerza que el propio gobierno y por ello es imprescindible que colaboren conscientemente y estén bien informados.

2.1.5 Economía

Por último, la última característica de una smart city está relacionada con la economía. Es importante para este modelo de ciudad desarrollar una economía **sostenible** que sea competitiva y mejore la calidad de vida pues, cuanto menor sea el gasto en gasolina mejor será la gestión del tráfico de la ciudad o, cuanto mejor sea la calidad del aire menor será el gasto en sanidad.

Además, la buena organización urbana de una smart city **atrae inversiones** y crea empleo sostenible como por ejemplo en el sector energético o en el sector de las TICs. Cada vez son más las empresas que buscan invertir en smart cities y que las convierten en más competentes y con mayores posibilidades.

2.2 Cartagena en el contexto de las *Smart Cities*

Cartagena cuenta con la información y las infraestructuras suficientes para ser una *smart city* o ciudad inteligente. En estos momentos lo que se necesita es la unificación de esas infraestructuras y el análisis inteligente de la información que dichas infraestructuras ya proporcionan, además del desarrollo de aplicaciones para que esa información llegue al ciudadano a través de sus smartphones, tablets y demás dispositivos conectados a la red.

2-2 Logotipo de la RECI

Además, en el mes de julio de 2016 se aprobó el *Plan Director de Smart City,* lo cual podría impulsar a Cartagena en su adhesión a la RECI[3] [5].

2.3 ¿Por qué surgen las *Smart Cities*?

Dotar de mayor rapidez a la hora de actuar frente a algún imprevisto en cualquiera de las infraestructuras públicas.

Además de la implementación en el software libre QGIS, el cual es multiplataforma, se ha previsto la misma en una plataforma web para asegurar la *escalabilidad* y la *disponiblidad* de los datos.

[3] Red Española de Ciudades Inteligentes

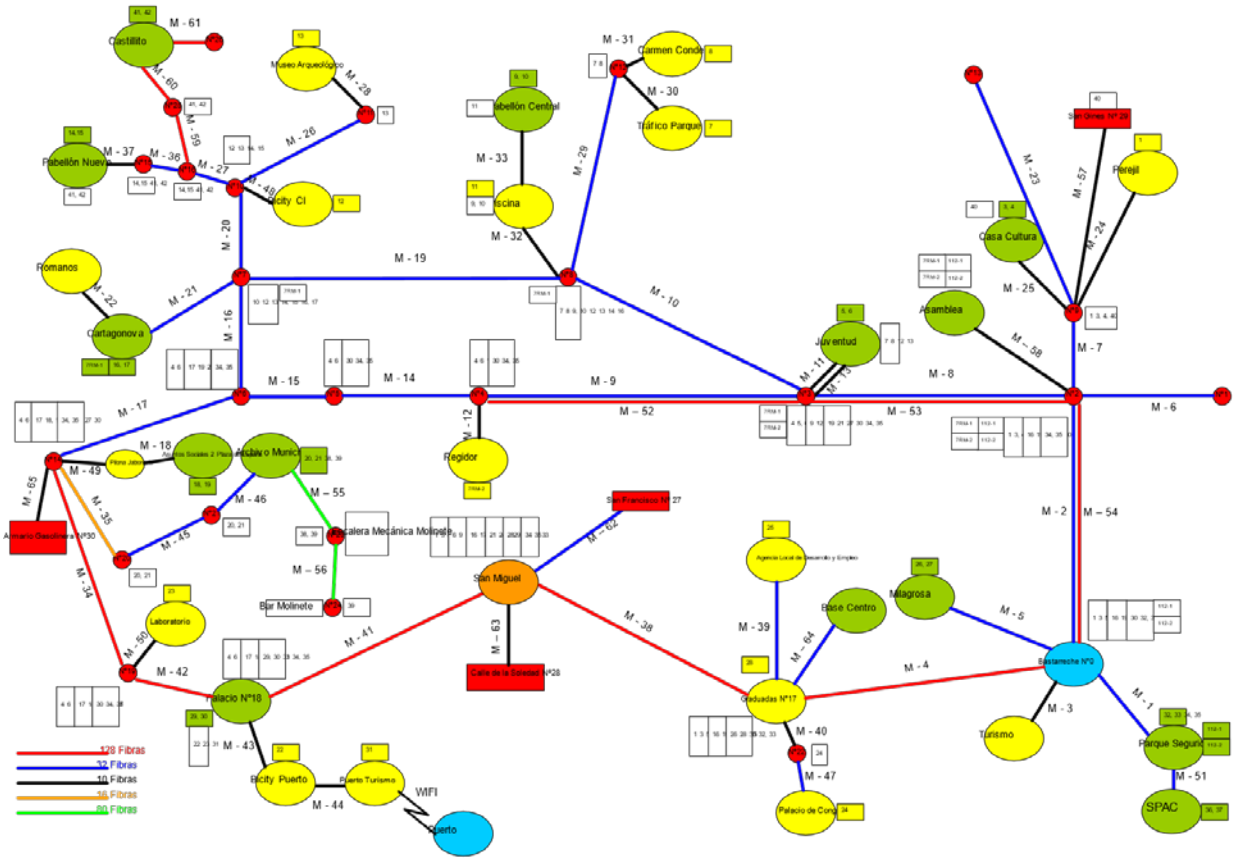
3 Infraestructuras de Cartagena

Cartagena cuenta con una amplia infraestructura tecnológica que, si bien no se puede llamar *Smart City*, si se pusieran a trabajar todas ellas en conjunto se podría dar forma de manera que pasara a ser una pequeña *Smart City*.

3.1 Red de fibra óptica

La red de fibra óptica de las infraestructuras públicas de la ciudad fue creada en dos fases gracias al plan E de 2009 [6] y al de 2010.

Esta fibra óptica conecta todos los edificios administrativos de la ciudad, creando así una intranet tipo *MAN*[4]



3-1 Esquema de la red de fibra óptica de Cartagena

En el dibujo se aprecian los colores de las fibras según el número de microfibras que hay en cada manguera. Cada acometida de fibra va numerada, y en cada edificio público se indica qué fibra parte de ese edificio y qué fibra llega.

[4] Metropolitan Area Network o red de área metropolitana. Actualmente en desuso, se podría nomenclar como *LAN*

3.2 Líneas de bus

Cartagena cuenta con 14 líneas de bus. Cada bus está geolocalizado en todo momento. Como hardware de seguimiento se usan sistemas embebidos como el de la foto 3-2.

3-2 Ordenador embebido a bordo de los autobuses

El principal problema de este sistema es que en la empresa de autobuses no saben cómo funciona y los datos pasan por un CPD en Murcia antes de acabar en las oficinas de la empresa donde sí tienen implantado el seguimiento en tiempo real.

Cada sistema embebido como el de la foto va conectado a una antena GPS que es la que pasa la señal satelital al ordenador para poder realizar la triangulación y el posicionamiento en sí.

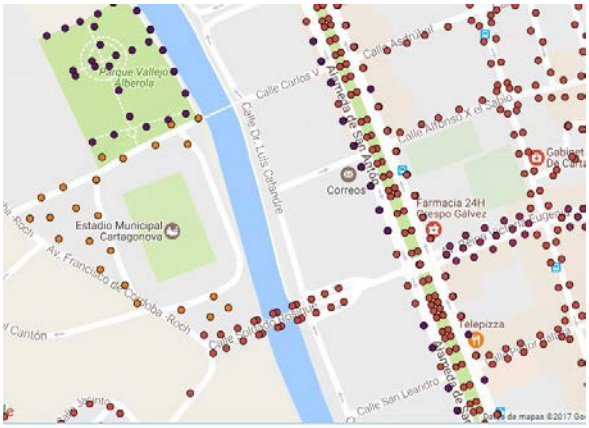
3-3 Módulo GSM/GPRS

También llevan incorporados un módem *cinterion mc35i* el cual permite la comunicación del ordenador de seguimiento vía GSM/GPRS (800/1900 Mhz). El

motivo de la incorporación de este tipo de comunicaciones es la cobertura que ofrece frente a otros sistemas.

3.3 Luminarias

La ciudad de Cartagena cuenta con una extensa variedad de luminarias, tanto de diferentes potencias como de diferentes tipos de iluminación.



3-4 Ubicación de las luminarias

Una parte de la cátedra consistía en dotar a las luminarias de un sistema de encendido/apagado remoto basado en condiciones, como por ejemplo la intensidad de la luz del día. Otra característica implementada ha sido un sistema de avisos mediante correo electrónico si el consumo de un ramal[5] bajaba de un margen establecido. La idea era implementar en el gis, ya sea versión web o versión de escritorio, avisos luminosos cuando se diera esta circunstancia. Aun así, se ha implementado la posición real de las farolas y de los cuadros de mando tanto en la versión de escritorio como la versión web. En ambas interfaces se ha coloreado cada farola según la potencia de la bombilla.

Para esto se ha usado como prototipo un PLC de la marca Siemens modelo Logo 8, una pinza amperimétrica y un circuito customizado.

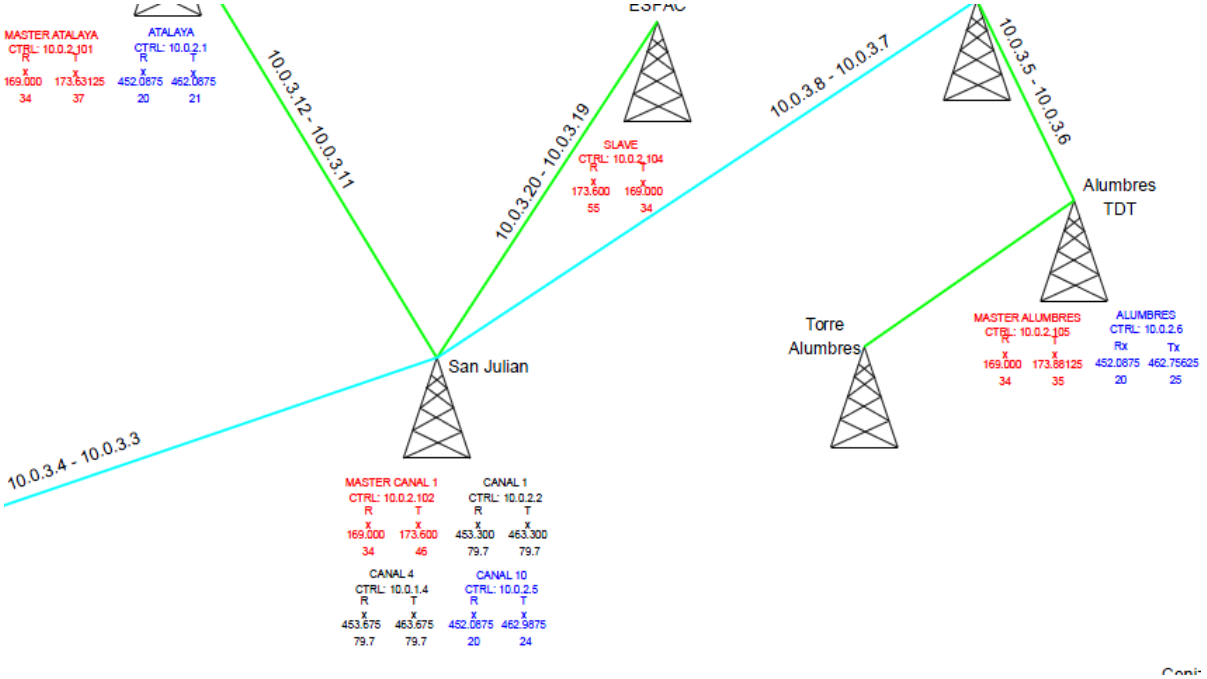
3.4 Radioenlaces

La ciudad cuenta con una red de radioenlaces IP que se extiende gracias a bases y/o antenas ubicadas en las partes más altas de la misma.

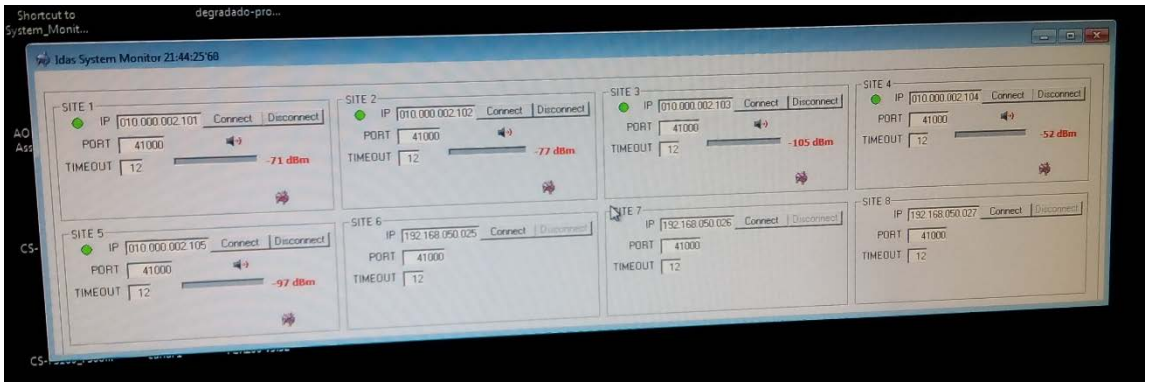
[5] Un ramal nace en un cuadro de mando y se extiende por una o varias acometidas de farolas, como puede ser una calle o avenida entera.

Los radioenlaces los usan los cuerpos de seguridad, el cuerpo de bomberos y protección civil para comunicaciones de voz.

El sistema está monitorizado por dos servidores situados en el parque de seguridad. Para ello se creó un programa en C# que comprueba que las conexiones estén activas y las activa en caso de que haya caído alguna.



3-5 Esquema de las torres de radioenlace de la ciudad, así como los canales que usa y la frecuencia de dichos canales



3-6 Interfaz gráfica del programa de monitorización y control

3.5 Auditorio El Batel y Palacio consistorial

Aunque ambos edificios no forman parte de la infraestructura *Smart* de la ciudad, los he metido en este apartado por ser dos *Smart Building*.

El Palacio consistorial situado en la Plaza del Ayuntamiento cuenta con infraestructura domótica que, si bien es bastante antigua, dota al edificio de la

entidad *Smart*. La domótica es poco eficiente energéticamente hablando ya que los actuadores no son electrónicos; son relés de baja tensión.

Los controladores son de la marca TAC Xenta, de los modelos 401C y 501 entre otros. Esta domótica se controla desde un pc con Windows XP, con Internet Explorer 6 y con Java 1.1, todo ello versiones obsoletas repletas de agujeros de seguridad.

Esta domótica está orientada a controlar iluminaciones interiores según los sensores de movimiento y las horas del día. La climatización también forma parte de la domótica del edificio pero es una instalación independiente, más moderna y mucho más fácil de ampliar y/o actualizar.

3-7 En la fila de arriba los controladores, en la de abajo los actuadores del Palacio Consistorial

El **auditorio y palacio de congresos El Batel** cuenta con una complejísima y moderna infraestructura domótica, que abarca desde el control de iluminaciones hasta la climatización por zonas, pasando por las telecomunicaciones, si bien se podría mejorar la domótica. Una de las actuaciones deseadas por el ayuntamiento es la incorporación de unas pantallas táctiles para el control de iluminarias y climatización, ya que la única manera de realizar cambios manuales a ambas es

mediante acceso por un pc situado en una zona recóndita del auditorio o vía remota a ese pc, con los riesgos de seguridad que ello conlleva.

3-8 Cabecera de la instalación domótica del Batel

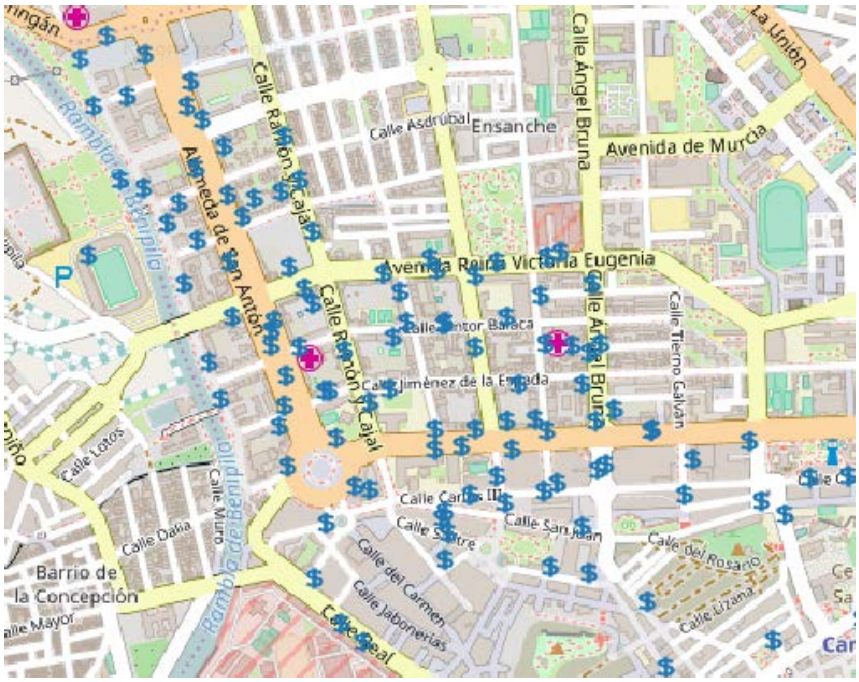
3-9 Actuadores con sus cabeceras de una zona

Los controladores usados son de la marca TAC Xenta modelos 401,411 y 491. El Gateway que conecta todos los controladores con el sistema de control (pc) es un TAC Xenta 913.

3.6 Servicios O.R.A.

Disponemos de las ubicaciones de los expendedores de la O.R.A. y de acceso al servidor esclavo situado en una máquina virtual del CPD del ayuntamiento.

Este servidor nos informa en tiempo real del estado del expendedor; si está en servicio, no comunica, en mantenimiento, si necesita papel, etc.



3-10 Vista general de la ubicación de los expendedores

Las ubicaciones fueron tomadas hace tiempo, por lo que el margen de error en la ubicación de cada expendedor puede ser de hasta 10 metros. Por consiguiente y con ayuda de Google Maps, los corregimos todos a mano y los insertamos en una capa del software.

Habría sido de gran interés añadir esta información en tiempo real a las capas del GIS, tanto la versión web como la versión de escritorio. Pero nos topamos con el problema de que desde fuera no se puede acceder al servidor de datos de la O.R.A. y tampoco nos daban acceso por motivos de seguridad.

3.7 Centro de control del ayuntamiento

En la primera planta del ayuntamiento se instalaron 32 pantallas múltiples a modo de centro de control. Estas pantallas se unen para hacer dos pantallas gigantes de 4x4.

Para cada pantalla gigante hay un PC con división de 3 pantallas; dos monitores pequeños y el gigante de la pared.

3-11 Pantallas del centro de control. Al fondo se observa una de las pantallas gigantes con la interfaz web del GIS

4 Sistemas de Información Geográfica (SIG)

4.1 ¿Qué es un SIG?

Según la Wikipedia, *es cualquier sistema de información capaz de integrar, almacenar, editar, analizar, compartir y mostrar la información geográficamente referenciada*.

Según el Instituto Geográfico Nacional, el término SIG[6] tiene tres vertientes:

- SIG como disciplina.
- SIG como proyecto.
- SIG como software.

La acepción principal es la de SIG como proyecto, la cual se puede enunciar como: *Conjunto integrado de medios y métodos informáticos, capaz de recoger, verificar, almacenar, gestionar, actualizar, manipular, recuperar, transformar, analizar, mostrar y transferir datos espacialmente referidos a la Tierra*.

Por lo tanto, desde una interface de usuario, un SIG nos permitirá visualizar, modificar, analizar e interpretar la información geográfica de la que dispongamos.

4.2 Tipos de datos

Los SIG manipulan dos tipos de datos básicos:

- Información espacial.
- Información descriptiva.

4.2.1 Información espacial

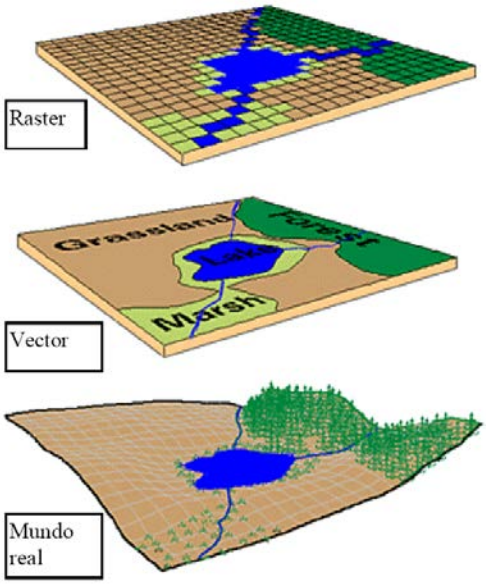
También llamada información geográfica. Esta describe la localización y la forma de las características geográficas, tales como la orografía[7] del suelo, entre otras. Tales datos se almacenan en dos formatos: [7]

- Formato **ráster**[8]**.** Aquí el espacio se divide en pequeños cuadrados (celdas) del mismo tamaño. Con un ráster de tamaño de celda más pequeño se pueden representar más entidades, entidades más pequeñas o más detalle en la extensión de entidades [8].

[6] Sistema de Información Geográfica
[7] Se refiere tanto a las elevaciones que puedan existir como a la forma de las mismas.
[8] Existe cierta confusión en la manera de escribir esta palabra. Siguiendo el razonamiento de [19] y según se ha usado en [8], se ha optado por usar tilde.

- Formato de **datos vectorial.** Expresa las características geográficas como vectores, representando fielmente las relaciones topológicas entre los elementos de la capa ráster. Se usan tres elementos en el sistema vectorial:
    - *Puntos*. Se utilizan para marcar ubicaciones o lugares concretos, tales como una farola o un contenedor.
    - *Líneas*. Lista de coordenadas *x,y* ordenadas que representan la forma de entidades geográficas. Se podría decir que es un conjunto de dos o más puntos unidos. En este proyecto, podría ser, por ejemplo, una tubería.
    - *Polígonos*. Es un área acotada definida por un conjunto ordenado de coordenadas *x,y* donde la primera coordenada es también la última. Podríamos equipararlo con un conjunto de líneas con la peculiaridad de que donde acaba la última es donde empieza la primera. Los polígonos se suelen utilizar para representar ubicaciones bidimensionales, tales como edificios.

En la tabla 1 se pueden apreciar las diferentes ventajas e inconvenientes de cada formato.



4-1 Cómo se modela la información en un SIG a partir del mundo real

4.2.2 Información descriptiva

Este tipo de dato suministra información geográfica. Es decir, describe la información geográfica, tales como tipo de vía, tipo de río, etc…

Se puede decir que son los atributos de cada celda de la capa ráster. Esta información se relaciona con la información geográfica mediante las tablas de atributos.

Modelo Raster	
Ventajas	*Inconvenientes*
Utiliza una estructura de datos muy simple.	La estructura de datos es menos compacta.
Las superposiciones de las diferentes coberturas se implementan de forma rápida y eficiente.	Algunas relaciones topológicas son difíciles de representar
Permite una forma más eficiente de representación cuando la variación espacial es muy alta. El modelo raster es muy apropiado para el tratamiento de imágenes de satélite.	La información original se generaliza una vez que se traspasa al sistema, tanto cuanto más grande sea la dimensión de las celdas.
Modelo vectorial	
Ventajas	*Inconvenientes*
Posee una estructura de datos muy compacta.	La estructura de datos es más compleja
La salida en papel presenta muy buenos resultados.	Si la variación espacial es baja, resulta poco eficiente la aplicación.
	El procesamiento de imágenes digitales no puede ser realizado eficientemente en este tipo de formato.

Tabla 1 Ventajas e inconvenientes de los formatos raster y vectorial

4.3 Georreferenciación

Georreferenciar es posicionar una información en un lugar único y bien definido en un sistema de coordenadas y *datum*[9] específicos.

Se distinguen dos modalidades de georreferenciación: la directa y la indirecta o discreta.

4.3.1 Georreferenciación directa

En esta modalidad se usa un sistema de coordenadas establecido junto a un sistema de proyección. Este sistema de proyección no es más que una función

[9] Conjunto de puntos de referencia y modelo asociado de la forma de la tierra en la superficie con los cuales las medidas de la posición son tomadas para definir el sistema de coordenadas geográfico.

matemática (ecuación 1) que convierte las coordenadas entre un plano tridimensional y otro bidimensional.

$$x = a\lambda \quad (1)$$

$$y = a\ \ln[\tan\left(\frac{\pi}{4} + \frac{\phi}{2}\right)] \quad (2)$$

$$h = k = \frac{\sqrt{(1 - e^2 \sin^2 \phi)}}{\cos \phi} \quad (3)$$

Ecuación 1. Fórmula correspondiente a la latitud (1) y a la longitud (2) en la proyección Web Mercator

En este proyecto se va a usar la proyección ***Web Mercator***, también conocida como *Google Web Mercator, Spherical Mercator*, *WGS 84 Web Mercator* o *WGS 84/Pseudo-Mercator* [9]. El identificador estándar para dicha proyección es **EPSG:3875**[10] o WGS84 Pseudo-Mercator.

En la *Ecuación 1* tenemos:

- λ es la longitud del elipsoide en radianes.
- Φ es la latitud del elipsoide en radianes.
- *a* es el semieje mayor del elipsoide. En la proyección que nos ocupa tiene un valor de 6378137 metros.
- *e es* la excentricidad del elipsoide. En el EPSG:3875 tiene un valor de $8.18191908426 \times 10^{-2}$.
- *h* es el factor de escala del meridiano.
- *k* es el factor de escala del paralelo.

4.3.2 Georreferenciación indirecta

En este tipo de georreferenciación las coordenadas se obtienen, mediante procesado, de puntos de cuales disponemos información postal, dirección o similares y no sus coordenadas.

Normalmente se suelen usar bases de datos noSQL[11] para este tipo de georreferenciación debido a su mayor eficiencia frente a bases de datos tradicionales a la hora de manejar muchos datos en tiempo real.

[10] EPSG son las singlas del *European Petroleum Survey Group* [20]
[11] Sistemas de gestión de bases de datos que no usan SQL como lenguaje principal de consultas.

4.4 Servicios WMS

El servicio *Web Map Service* es un estándar que proporciona una respuesta HTTP definiendo un mapa de datos referenciados espacialmente (véase apartado *7.3.2*). Obviamente, para que haya una respuesta ha de haber una petición.

4.4.1 GetCapabilities

La primera petición que se realiza es **GetCapabilities** y tendría la siguiente forma:

```
http://pccastejon.upct.es:10080/geoserver/wms?request=GetCapabilities
```

El servidor responderá con un documento *XML* tal que así:

```
▼<WMS_Capabilities xmlns="http://www.opengis.net/wms" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001
 updateSequence="1031" xsi:schemaLocation="http://www.opengis.net/wms http://pccastejon.upct.es:10080/geoserver/schemas/wms/1.3.0/
  ▶<Service>...</Service>
  ▼<Capability>
    ▼<Request>
      ▼<GetCapabilities>
         <Format>text/xml</Format>
        ▼<DCPType>
          ▼<HTTP>
            ▼<Get>
               <OnlineResource xlink:type="simple" xlink:href="http://pccastejon.upct.es:10080/geoserver/ows?SERVICE=WMS&"/>
             </Get>
            ▼<Post>
               <OnlineResource xlink:type="simple" xlink:href="http://pccastejon.upct.es:10080/geoserver/ows?SERVICE=WMS&"/>
             </Post>
           </HTTP>
         </DCPType>
       </GetCapabilities>
      ▼<GetMap>
         <Format>image/png</Format>
         <Format>application/atom+xml</Format>
         <Format>application/json;type=utfgrid</Format>
         <Format>application/pdf</Format>
```

4-2 Respuesta a una petición GetCapabilities

Con esta petición, nuestro software sabrá qué capas tiene, en qué formatos se pueden pedir dichas capas y las demás opciones si las tuviera.

4.4.2 GetMap

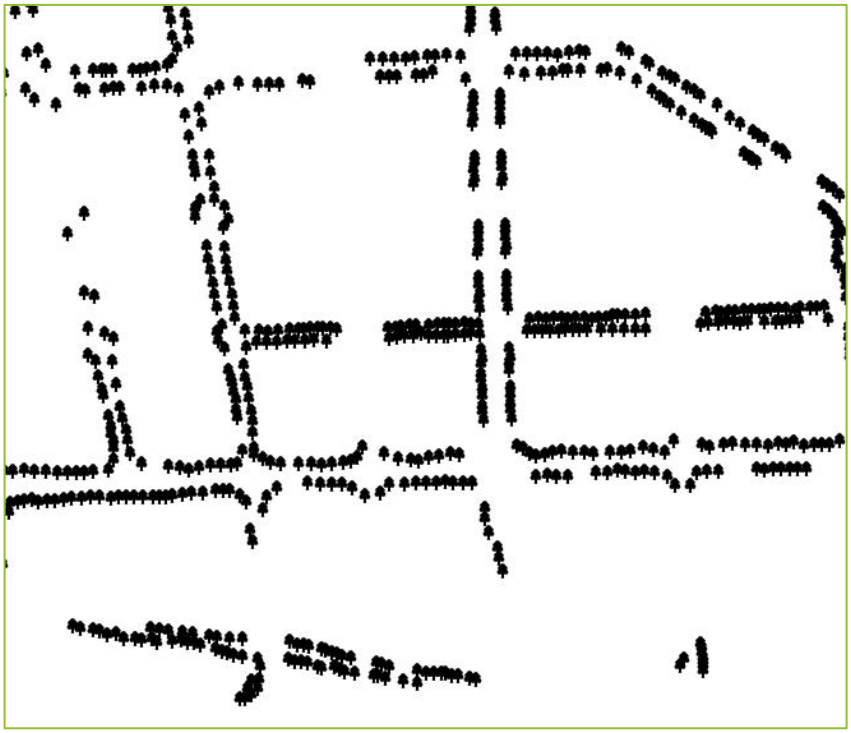
La siguiente petición lógica sería pedir uno de los mapas (capa) disponibles en el servidor. La petición para dicha tarea es **GetMap** y tiene esta estupenda pinta:

```
http://pccastejon.upct.es:10080/geoserver/cartagena/wms?service=WMS
&version=1.1.0&request=GetMap&layers=cartagena:arbolado
&styles=&bbox=-110995.96810252989,4522659.676822481,-
108518.71111587081,4525347.48507118
&width=707&height=768&srs=EPSG:3857&format=image/jpeg
```

Los parámetros usados en la petición se explican a continuación:

- **Service:** Como se ha comentado anteriormente, el servicio que estamos demandando. En este caso es WMS.
- **Version:** La versión del servicio WMS. En el ejemplo se hace uso de la versión 1.1.0. Las versiones existentes en la actualidad son:
    - **1.0.0 (abril 2000)**
    - **1.1.0 (junio 2001)**
    - **1.1.1 (enero 2002)**
    - **1.3.0 (enero 2004)**
- **Request:** La petición a ejecutar, es decir, le pedimos un mapa al servidor.
- **Layers:** Qué capas del mapa le estamos pidiendo. En el ejemplo, la capa del arbolado de Cartagena realizada por los compañeros de otra cátedra. Se pueden pedir varias capas de una sola vez separándolas con el punto y coma.
- **Styles:** Qué estilos se usarán para renderizar el mapa. En este caso, como usamos el que hay por defecto, lo dejamos vacío.
- **Bbox:** Es el *bounding box*, es decir, el área del mapa. Los parámetros que se le pasan son *longitud mínima, latitud mínima, longitud máxima, latitud máxima* en ese orden. Los separadores decimales son el punto.
    - La latitud es un número decimal comprendido entre -90.0 y 90.0
    - La longitud también es un número decimal, pero comprendido entre -180.0 y 180.0
- **Width** y **Height:** Anchura y altura, respectivamente, del mapa de salida en pixeles.
- **SRS:** *Spatial Reference System* o sistema espacial de referencia. Aquí es el EPSG:3857, que es el que se usará a lo largo de todo el proyecto.
- **Format:** El formato en el que se pide el mapa. En esta petición, una parcela en formato imagen JPG. Los posibles formatos son:
    - *PNG.* Este es el formato por defecto.
    - *PNG8*
    - *JPEG*
    - *JPEG-PNG.* Aquí el formato devuelto se elige automáticamente entre JPEG y PNG

- *GIF*
- *TIFF*
- *TIFF8*
- *GeoTIFF*
- *GeoTIFF8*
- *SVG*
- *PDF*
- *GeoRSS*
- *KML*
- *KMZ*
- *OpenLayers.* Genera una aplicación HTML
- *UTFGrid*



4-3 Capa que se devuelve con la petición

4.4.3 GetFeatureInfo

Cuando alguna de las capas que tenemos en nuestro mapa tiene activado el atributo de *queryable* ("preguntable"), se pueden pedir datos de determinadas zonas haciendo click con el ratón en dichas zonas. Se enviará entonces una petición del tipo **GetFeatureInfo** a nuestro servidor de mapas.

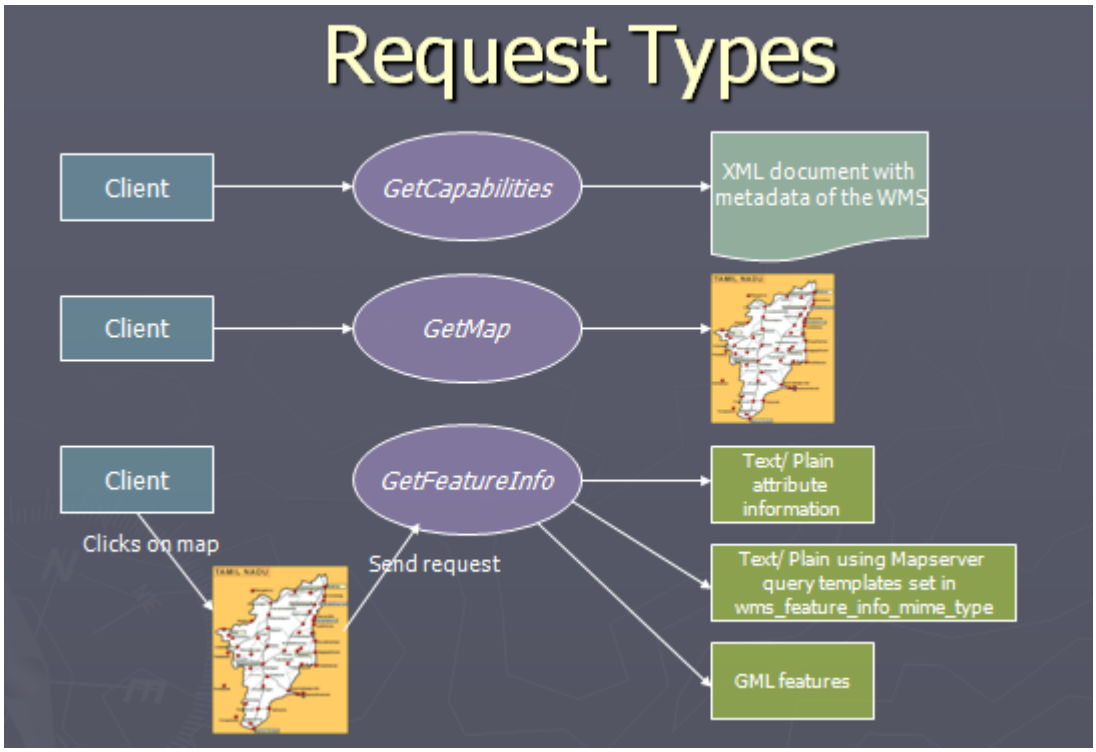
```
http://pccastejon.upct.es:10080/geoserver/cartagena/wms?SERVICE=WMS
&VERSION=1.1.1&REQUEST=GetFeatureInfo&FORMAT=image%2Fpng
&TRANSPARENT=true
&QUERY_LAYERS=cartagena%3AHidrante
&STYLES&LAYERS=cartagena%3AHidrante
&INFO_FORMAT=text%2Fhtml&FEATURE_COUNT=50&X=50&Y=50
&SRS=EPSG%3A3857&WIDTH=101&HEIGHT=101&BBOX=-
120439.6576339703%2C4523504.337729007%2C-
119475.7204398635%2C4524468.274923114
```

Muchos de los parámetros son los mismos que los que se pasan con la función **GetMap**, a excepción de:

- **Transparent:** Indica si la capa a la que estamos preguntando la información es transparente o no.
- **Query_layers:** Capas separadas por comas a las que se pregunta por la información.
- **Feature_count:** Número máximo de respuestas a devolver a nuestra petición.
- **X e y:** Coordenadas en pixeles del elemento solicitado.
- **Info_format:** El formato en el que queremos obtener la respuesta. Los formatos válidos son:
    - **Text (texto):** Información en texto simple. Si el parámetro está vacío se usará este formato por defecto.
    - **GML 2 o GML 3:** *Geographic Markup Languaje* o Lenguaje de marcado geográfico. Sigue un esquema XML. La versión 3 soporta características complejas.

- **HTML:** Se pueden especificar plantillas en el lado del servidor para representar la información devuelta. Es la usada en este ejemplo.
- **JSON:** Representación simple de la información serializada.
- **JSONP:** Devuelve la información también en JSON, pero personalizada.

En la siguiente imagen tenemos un esquema de cómo sería una petición a un servidor WMS:



4-4 Diferentes peticiones a un servidor WMS

5 Metodología

En este apartado se describirá el software empleado para la realización del mapa web. A saber:

- Sistemas operativos *Lubuntu Xenial 16.04* y *Windows 10 x64 Home*.
- Programa de código libre *QGIS.*
- Servidor de mapas *GeoServer.*
- *Heron Mapping Client*, el cual a su vez está compuesto por:
  - *ExtJS v3*
  - *OpenLayers 2.12*
  - *GeoExt 1.1*
- Servidor Web *Apache*
- Base de datos *PostGIS*, el cual es una ampliación de la base de datos *PostgreSQL*.

5.1 Sistemas operativos

El sistema operativo Lubuntu se ha usado para alojar el servidor *Geoserver,* que es el que sirve las capas vía WMS. Esta versión es una modificación del conocido Ubuntu. Dicha modificación consiste en el cambio del escritorio *KDE* por *LXDE*, siendo menos pesado en cuanto a consumo de recursos.

Por otra parte, Windows 10 se ha utilizado como entorno de producción donde hemos creado las capas y los mapas con el software *QGIS*, del que también hablaremos en capítulos siguientes.

5.2 QGIS

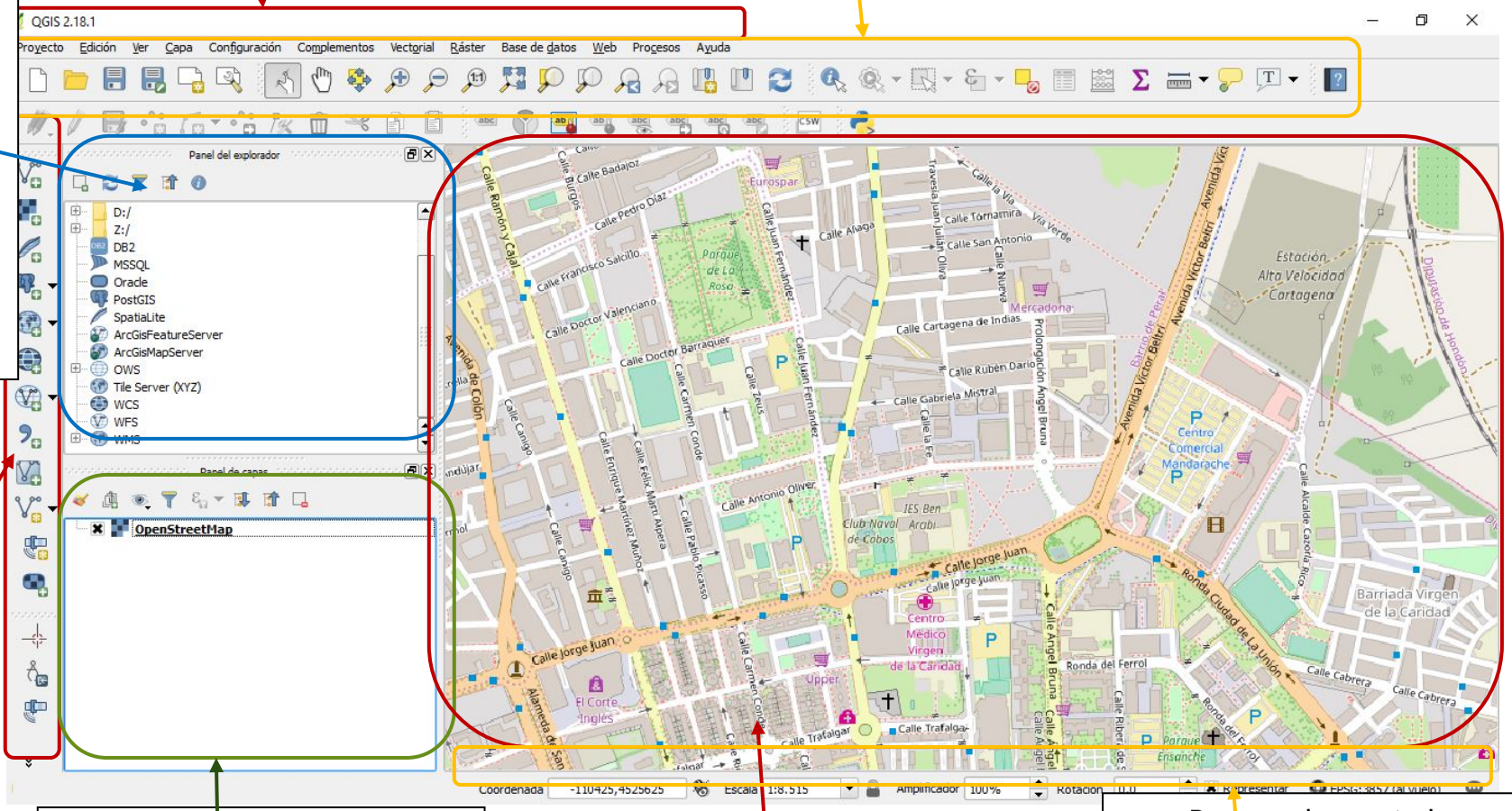
Se han usado diferentes versiones, pues a lo largo de la cátedra han ido saliendo actualizaciones del mismo. Hemos aplicado estas actualizaciones ya que no comprometían la compatibilidad del trabajo con las versiones posteriores.

Hemos optado por QGIS por ser el más actual y el más económico (gratuito). Se consideró la alternativa española GVSIG, pero la comunidad de desarrolladores no es tan activa y las actualizaciones menores que corrigen bugs suelen tardar más en aparecer. Por otra parte, ARCGis, de sobra conocido en el ámbito de la UPCT, tiene un precio exorbitado para el uso que le íbamos a dar y el ayuntamiento no contaba con presupuesto para adquirir las licencias pertinentes.

Navegador. Podremos navegar tanto por nuestro PC como por los servidores WMS que tengamos añadidos en busca de capas.

Menú de la aplicación

Accesos directos a las funciones más usadas. Se puede personalizar. Estas funciones también están en el menú



Barra para añadir capas

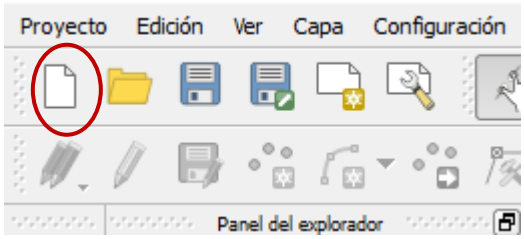
Panel de capas añadidas. Aparecerán tanto las activas como las desactivadas.

Vista principal del mapa. Aquí se visualizarán las capas que tengamos activas en el panel de capas.

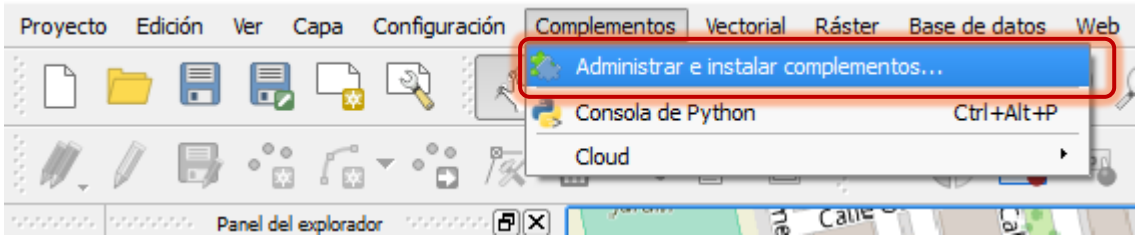
Barra de estado con información útil como posición del puntero en el mapa, formato de proyección, zoom, etc.

5.2.1 Primeros pasos

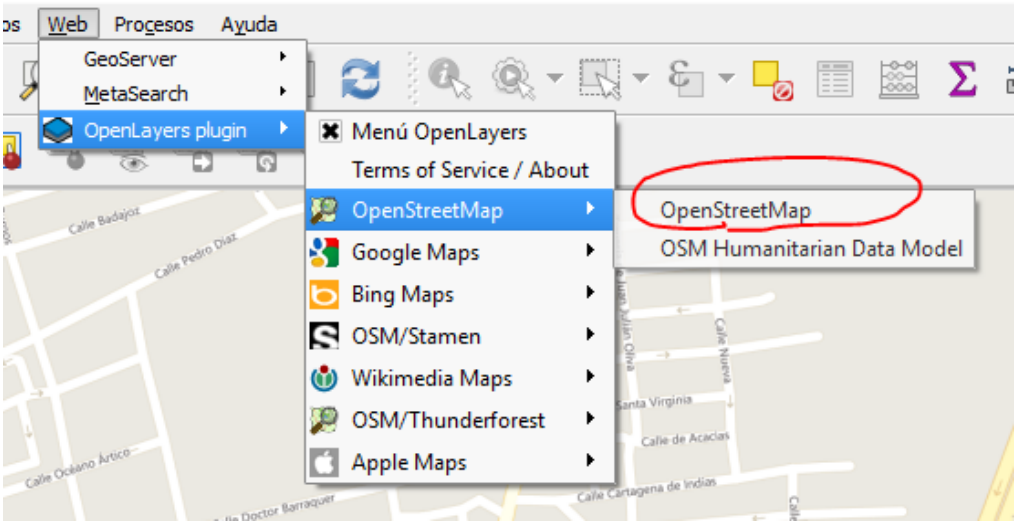
Tras instalar el programa desde su sitio oficial[12] lo abrimos desde el menú de inicio, cerramos las sugerencias y creamos un proyecto nuevo.



Tras crear el proyecto y ponerle nombre, vamos a instalar un complemento que nos permita poner OSM [13] como mapa base.



5-1 Hacemos click en administrar complementos. En la ventana que se nos abre buscamos OpenLayers plugin, lo seleccionamos y lo instalamos



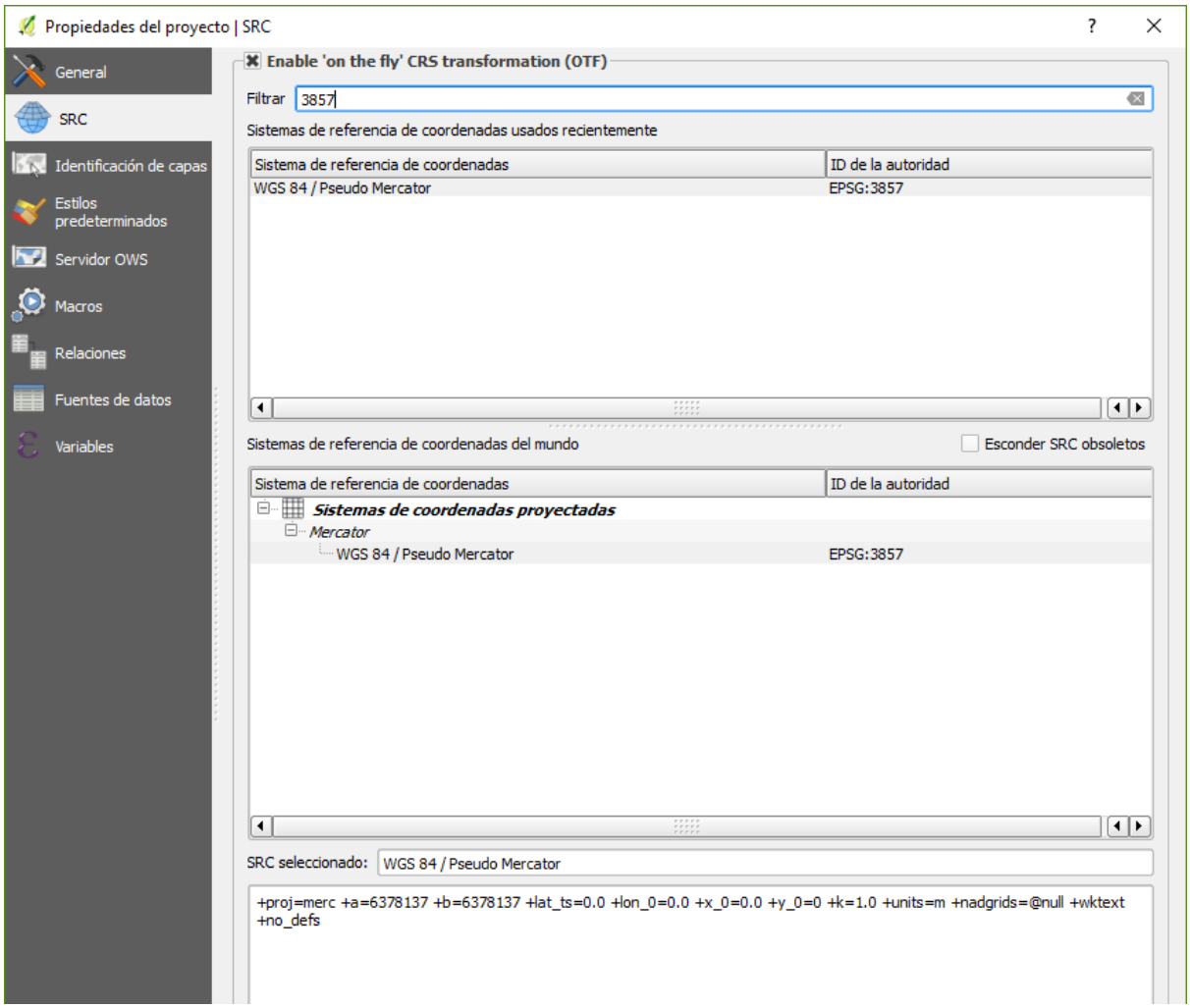
5-2 Una vez instalado, nos vamos a Web ▫ OpenLayers Plugin ▫ OpenStreetMap ▫ OpenStreetMap. Se puede elegir cualquier otro, pero el de Google da fallo

Tras esto, en el panel lateral de la izquierda nos aparecerá la capa de OpenStreetMaps.

El siguiente paso es configurar en qué coordenadas vamos a trabajar. Como ya hemos comentado anteriormente, por motivos de compatibilidad con las capas base

[12] http://qgis.org/en/site/forusers/download.html
[13] OpenStreetMap

de Google Maps y de OpenStreetMap la proyección elegida es **EPSG:3857.** Para ello pulsamos *CTRL+P*[14] y, en el cuadro de búsqueda donde nos dice *filtrar* escribimos 3857, seleccionamos el que nos aparece en el cuadro de abajo y pulsamos *aceptar*.



5-3 Seleccionamos el sistema de referencia de nuestro proyecto

Una vez hechos estos pasos ya tenemos nuestro mapa base listo y el sistema de referencia para las proyecciones seleccionado.

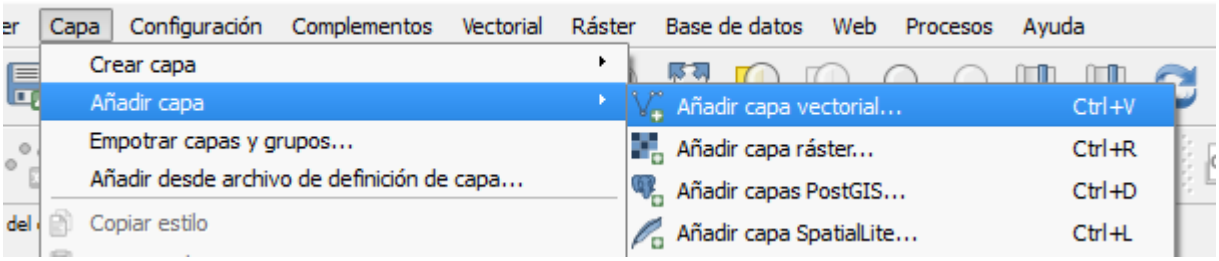
5.2.2 Adición de las capas restantes

El resto de capas las podemos añadir de dos formas diferentes: mediante los *shapefiles*[15] que tenemos en el disco duro o mediante servicios online como *WMS*. Como estamos empezando desde cero, añadiremos las capas que previamente nos hemos descargado o conseguido por algún medio.

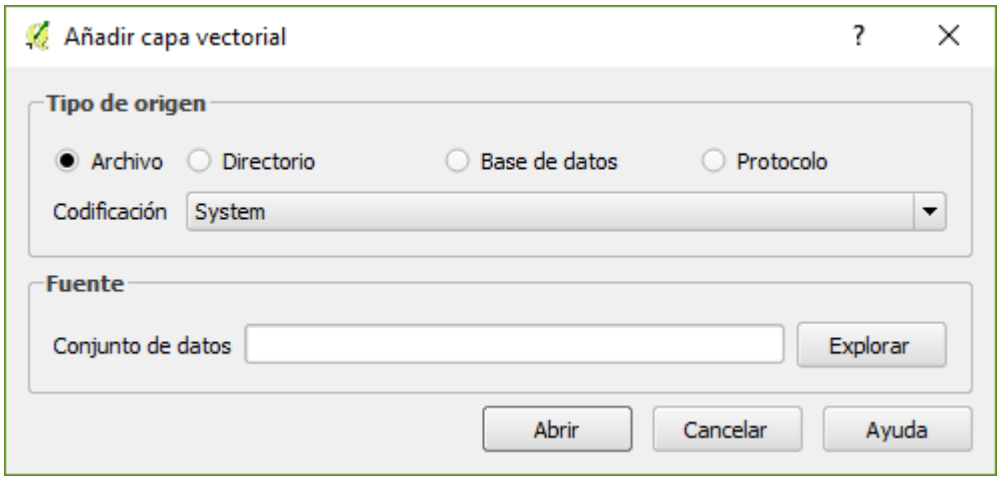
[14] También nos podemos ir al menú *Proyecto → Propiedades del proyecto…*

[15] Formato de archivo que se creó para la utilización de ArcView GIS, pero actualmente se ha convertido en formato estándar de facto para el intercambio de información geográfica entre SIG por estar muy bien documentado.

Para ello, nos vamos al menú *capa → añadir capa → capa vectorial…*

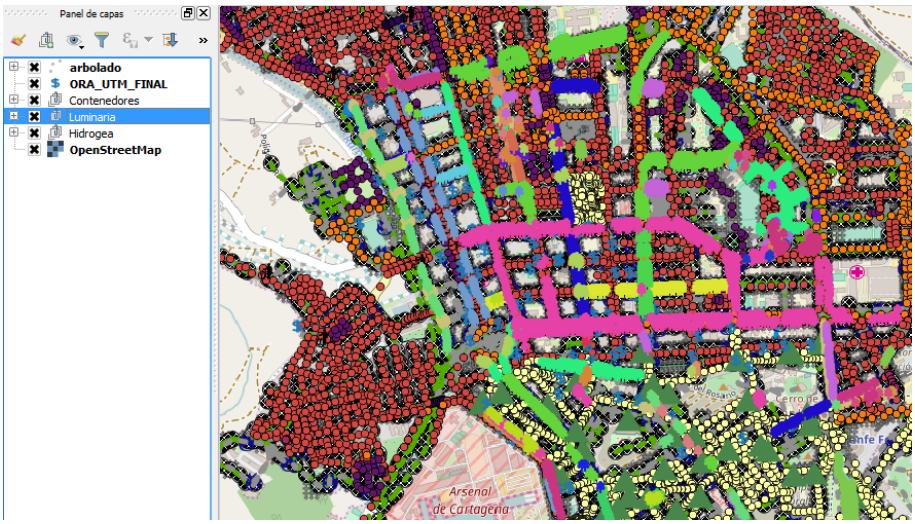


O bien elegimos el icono  de la barra lateral izquierda.



5-4 Seleccionamos, o bien el fichero que queremos añadir, o el directorio donde se encuentran los ficheros. Pulsamos aceptar cuando lo hayamos hecho.

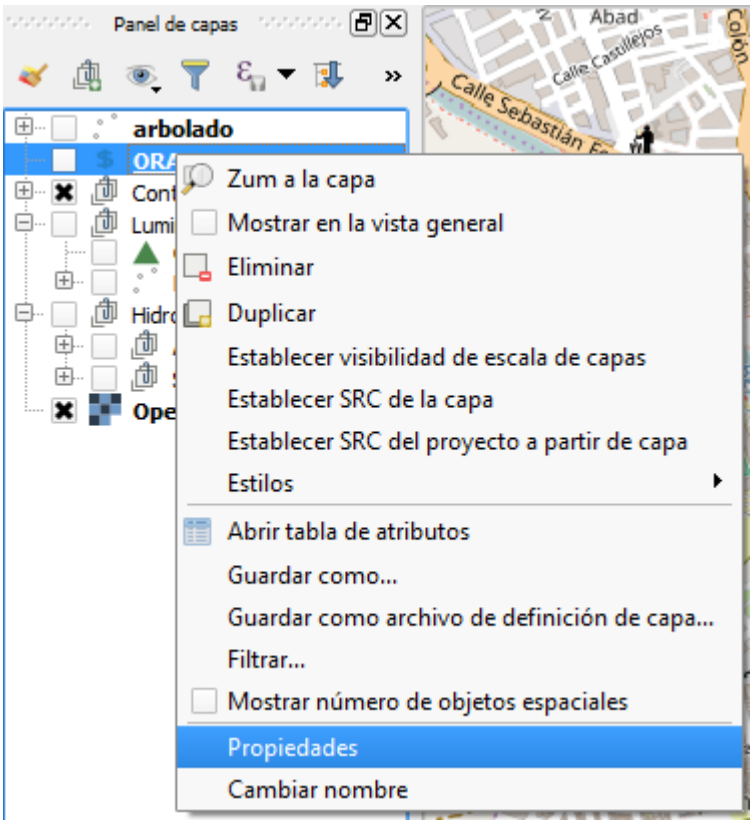
Una vez añadidas las capas, la vista principal del mapa nos quedarán tal que así:



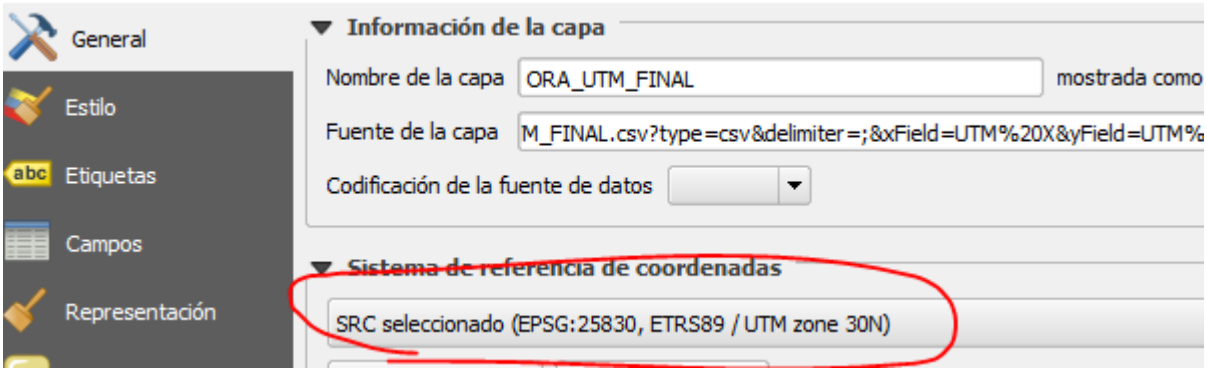
5-5 Aspecto de cómo se queda el programa con sus capas

Nota: Es recomendable no tener activadas todas las capas de golpe a no ser que contemos con una máquina muy potente, ya que el flujo no será el esperado y será casi imposible trabajar bien con el programa.

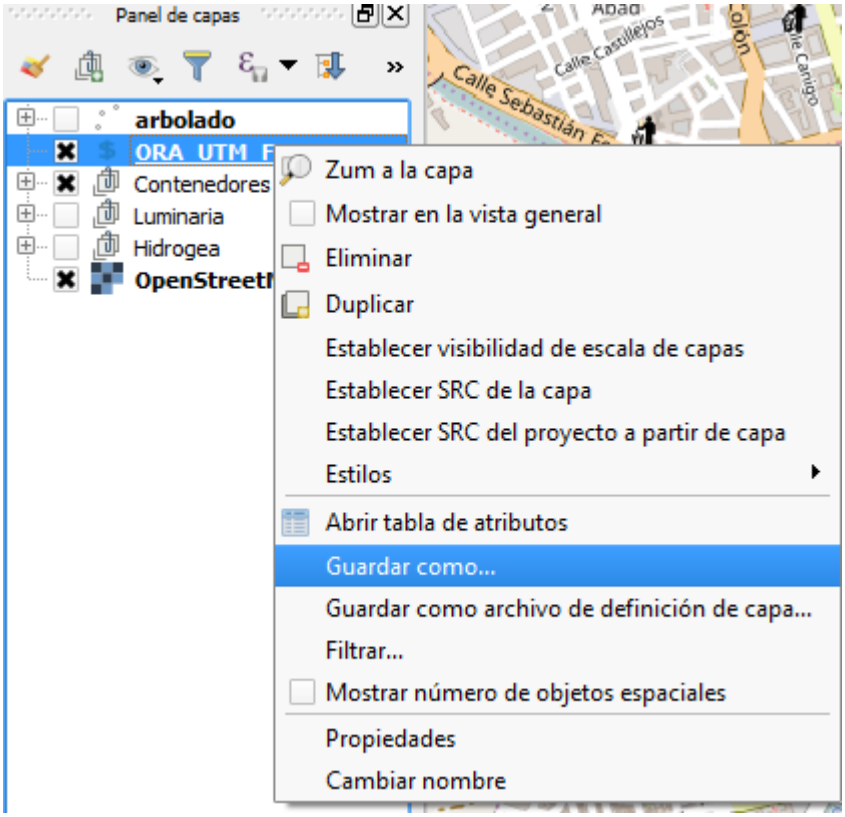
Una vez añadidas las capas pasamos a comprobar que están en la proyección deseada (EPSG:3857). Las que no lo estén, las guardaremos en dicha proyección. Aunque para trabajar en qgis no es necesario reproyectar las capas, en el servidor que utilizaremos sí que es necesario, ya que una ambigüedad en el formato no permite al mismo reconocer en qué proyección está cada capa, por lo que es imposible que sean reproyectadas por el propio servidor.



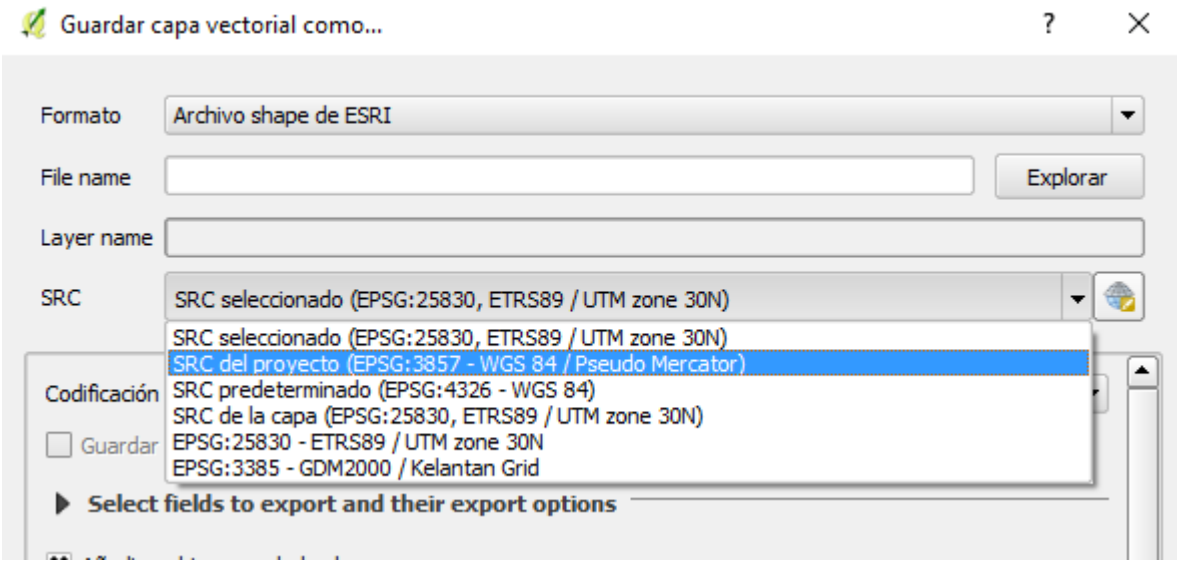
5-6 Clicamos con el botón derecho encima de cada capa y pulsamos sobre propiedades



5-7 En este caso el sistema de coordenadas nativo no coincide con el que vamos a usar



5-8 Guardamos la capa como…

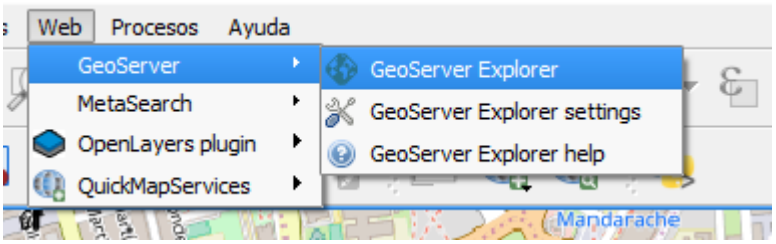


5-9 Elegimos la ruta del archivo y seleccionamos el SRC que nos interesa, en este caso EPSG:3857

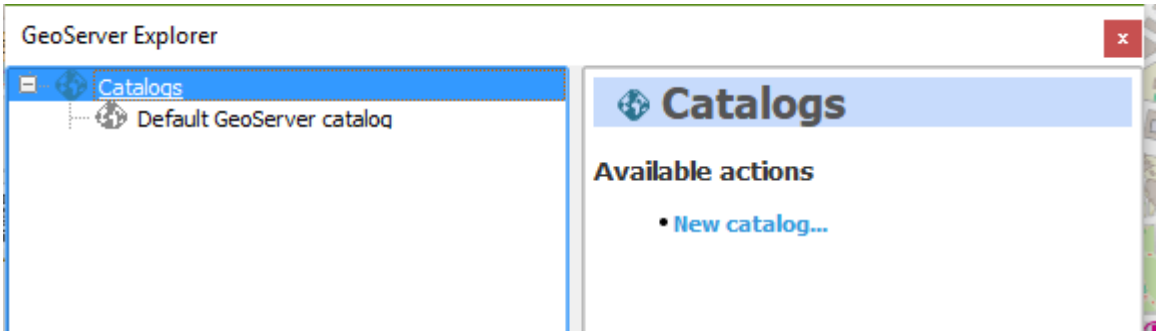
Realizamos estos pasos con todas las capas que no estén en la proyección con la que vamos a trabajar. Cuando estén todas las capas guardadas en nuestro sistema de referencia, las volvemos a añadir quitando las que ya teníamos.

5.2.3 Subiendo las capas a GeoServer

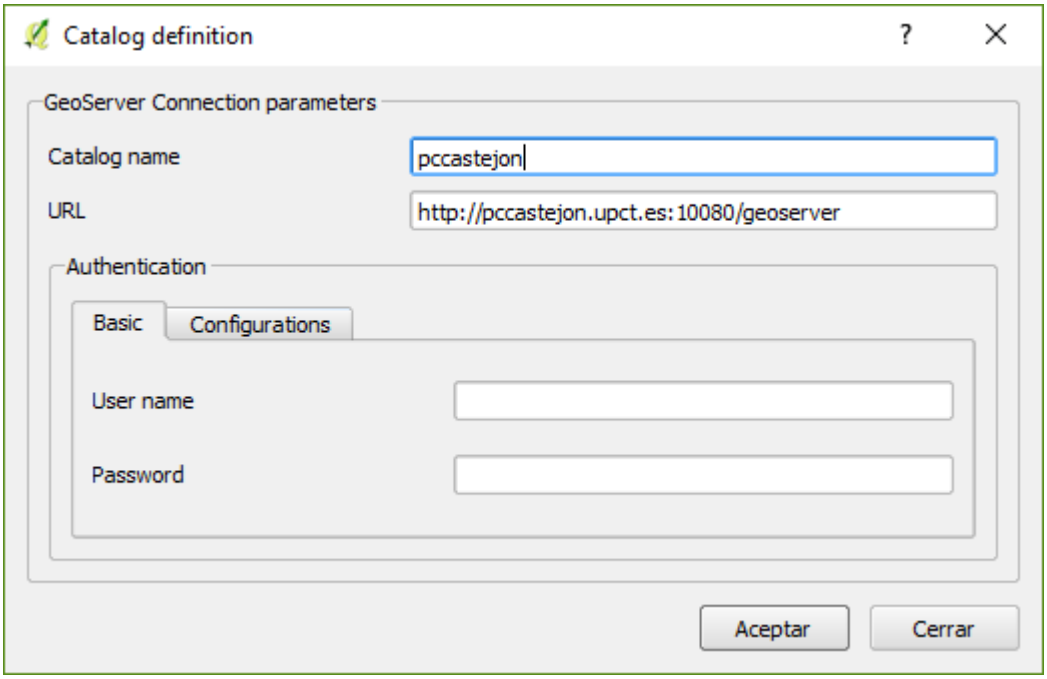
Ya estamos listos para añadir el plugin que nos comunicará con nuestro servidor de capas *GeoServer*. Abrimos el administrador de complementos tal y como se muestra en la ilustración 6-1, buscamos *GeoServer ExpIrorer* y pulsamos sobre instalar.



5-10 Abrimos el plugin desde el menú de la aplicación

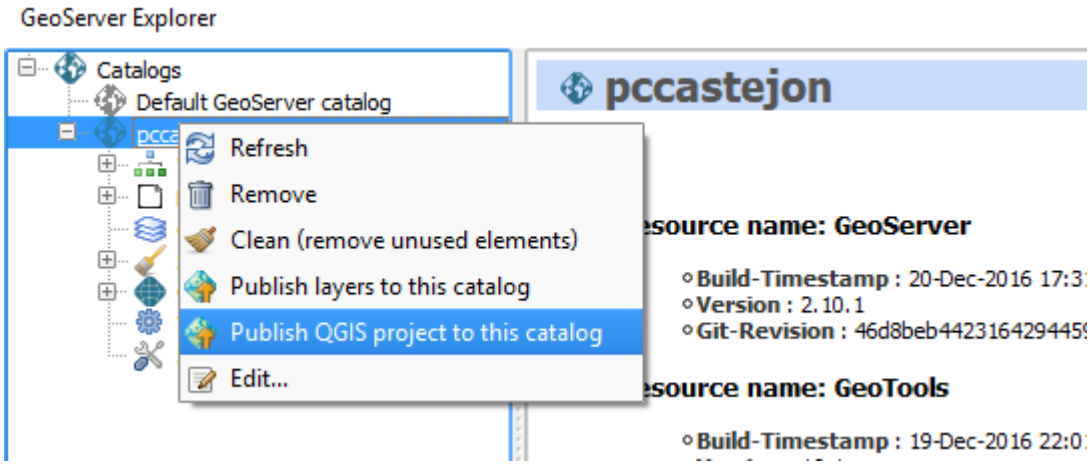


5-11 En la ventana que nos aparece pulsamos sobre New Catalog



5-12 Elegimos el nombre que le pondremos al servidor, la URL y credenciales de acceso si fuera necesario

En la ventana que nos aparecía en 6-11 nos aparecerá ahora nuestro servidor con toda la información que éste nos proporciona.

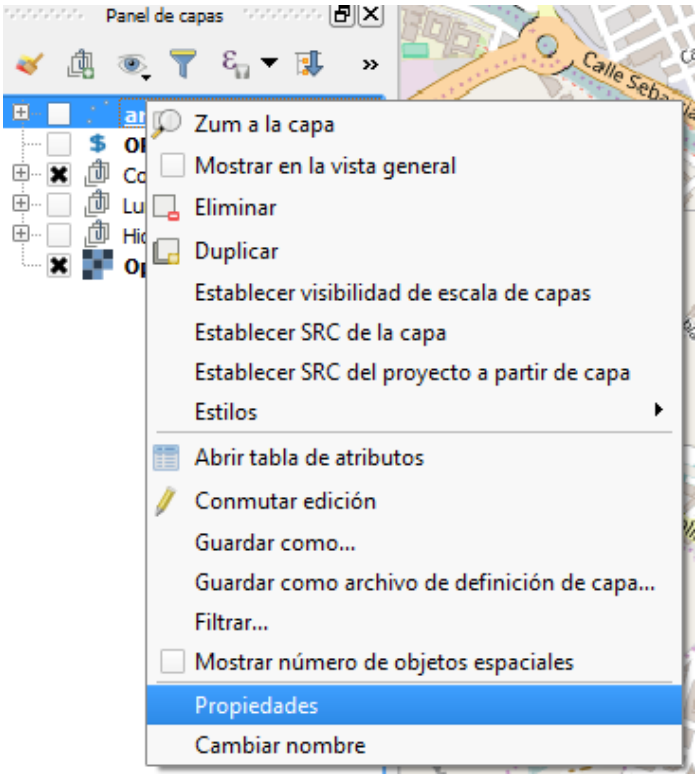


5-13 En esta ventana, pulsamos con el botón derecho sobre el nombre del servidor y pulsamos sobre la opción que está iluminada en azul en la imagen

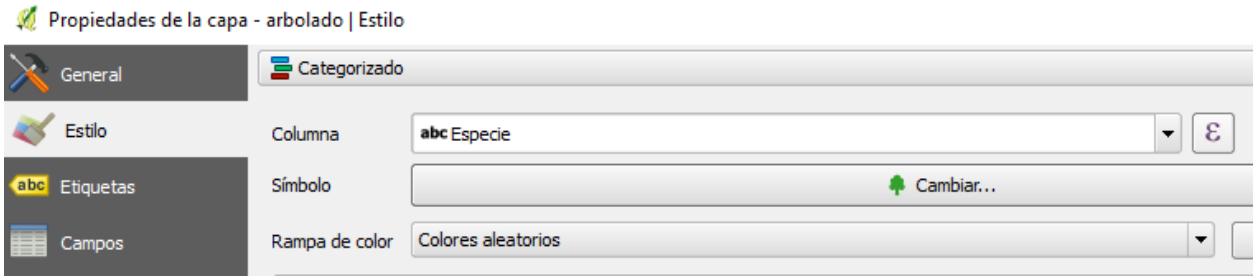
Con esta acción tendremos nuestro proyecto en el servidor WMS junto a las capas y a las proyecciones deseadas. El trabajo con el servidor lo dejaremos para más adelante. Ahora vamos a dar color al mapa.

5.2.4 Coloreando capas

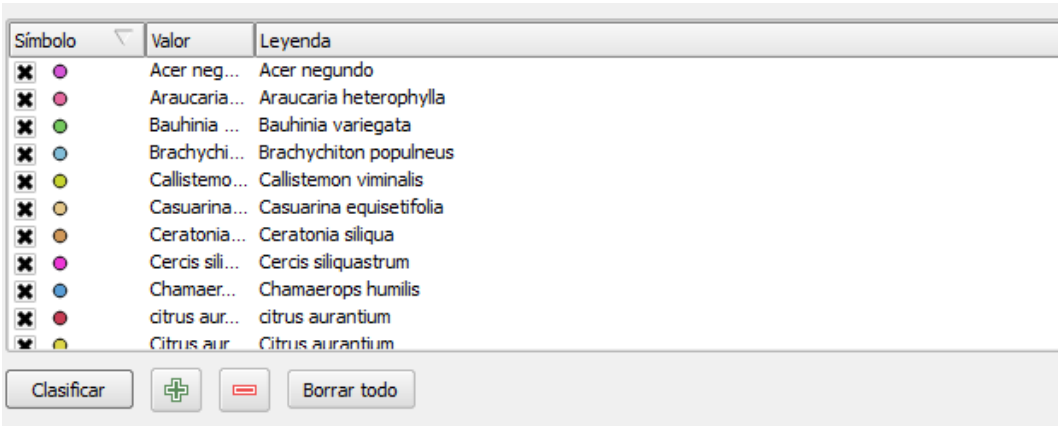
Empezamos por colorear e iconizar por categorías la capa del arbolado.



5-14 Vemos las propiedades de la capa del arbolado

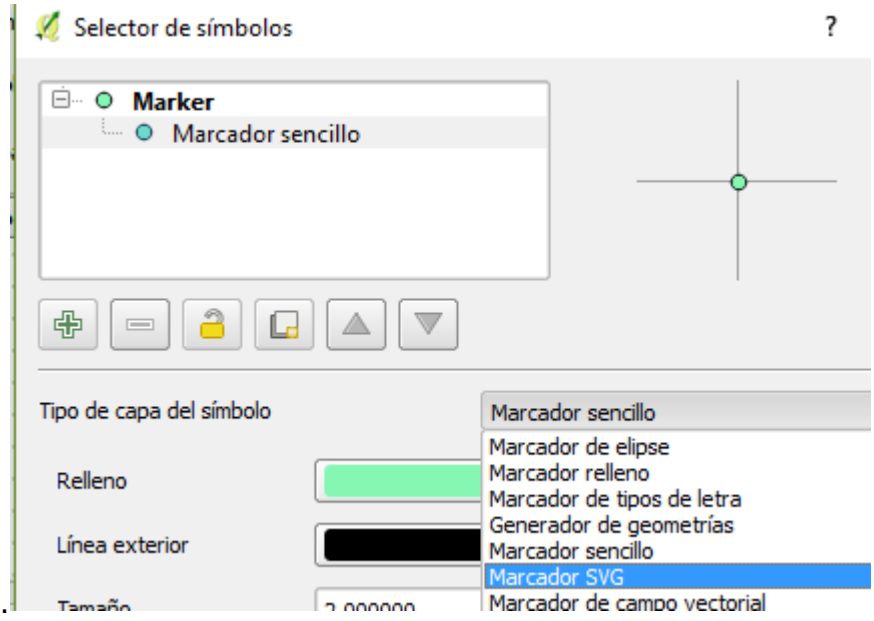


5-15 En la nueva ventana nos vamos a la pestaña *estilo*, elegimos *categorizado*. Como vamos a categorizar por especies, en *columna* elegimos especie. Los colores a gusto personal

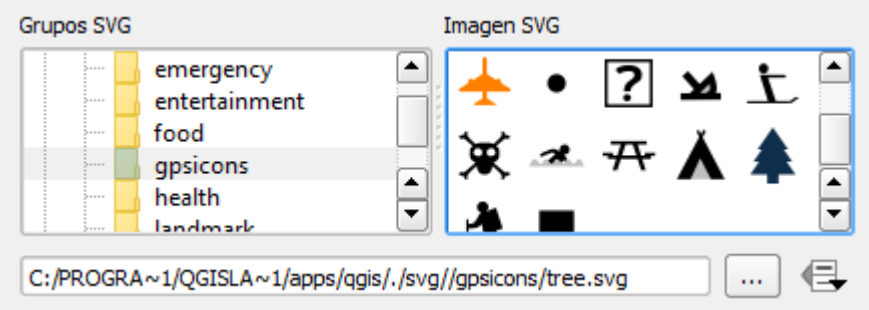


5-16 Al pulsar sobre clasificar nos quedará tal que así

Ahora vamos a ponerle un icono de árbol, que es lo suyo, ¿no?. En 6-15 pulsamos sobre *cambiar*.

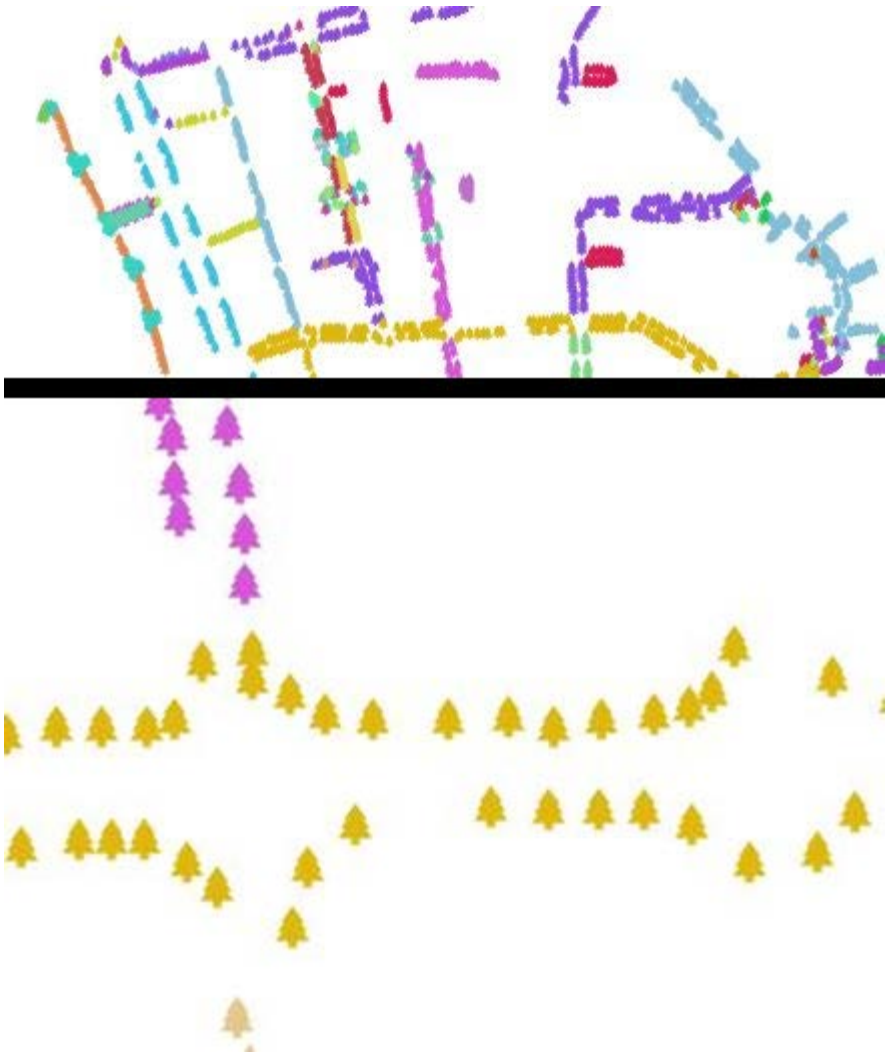


5-17 Seleccionamos *Marcador sencillo* y en *tipo de capa del símbolo* elegimos *marcador SVG*



5-18 Desplazamos la ventana hacia abajo y elegimos un icono que creamos apropiado para nuestra capa

Pulsamos aceptar en todas las ventanas que teníamos abiertas para que se vayan guardando los cambios.



5-19 Arriba una vista general de cómo queda y abajo una vista más detallada

Vamos realizando esto con todas las capas que tenemos, simplemente varía que no todas está categorizadas, como por ejemplo la de los expendedores de la ORA.

5.3 Servidor de mapas *GeoServer*

GeoServer es un software de código abierto escrito en java que permite a los usuarios ver, editar y compartir datos geospaciales. Estos datos usan estándares abiertos desarrollados por el *OGC*[16]

Fue creado en 2001 por *The Open Planning Project*, una startup sin ánimo de lucro establecida en Nueva york. Esta empresa quería crear una serire de herramientas para impulsar la transparencia democrática y para ayudar a los gobiernos a ser más transparentes. GeoServer fue la primera de esa serie de herramientas.

Al mismo tiempo que se creó GeoServer, la OGC (el consorcio OpenGIS por aquel entonces) estaba trabajando en un protocolo que hiciera que los datos geospaciales estuvieran disponibles directamente via web usando *GML* o lenguaje de marcado geográfico. También se creó un servicio web de mapas, un protocolo para crear y mostrar mapas creados a partir de datos espaciales. [10]

5.3.1 Instalación

La instalación del servidor la realizaremos en Linux, más concretamente en lubuntu, tal como se ha especificado en el apartado 6.1.

El primer paso es instalar la versión 8 de la máquina virtual Java. Se puede instalar la versión 9, pero recomiendan la versión 8 de Oracle. Hemos optado por la versión 8 de openjdk y funciona sin ningún tipo de problema.

```
alfonso@dc7700p:~$ sudo apt-get install openjdk-8-jdk
```

Una vez esté instalado, nos cambiamos al directorio donde queremos instalar el servidor (lo hemos instalado en el recomendado por la documentación) y clonamos el repositorio de github en nuestra máquina:

```
alfonso@dc7700p:~$ cd /usr/share/
alfonso@dc7700p:/usr/share$ git clone https://github.com/geoserver/geoserver.git
```

Tardará un rato en descargarse por completo. Tras esto, añadimos la variable *GEOSERVER_HOME* a nuestro perfil de usuario. Para ello podemos editar el archivo */home/usuario/.profile* o introducir lo siguiente:

[16] Open Geospatial Consortium

```
alfonso@dc7700p:~$ echo "export GEOSERVER_HOME=/usr/share/geoserver" >> ~/.profile
```

Para aplicar el cambio, cerramos la sesión y la volvemos a abrir. También se pueden aplicar los cambios introduciendo:

```
alfonso@dc7700p:~$ . ~/.profile
```

Ya tenemos la variable asignada. Ahora tenemos que modificar el propietario del directorio para que el usuario actual (y los de un determinado grupo) pueda modificar archivos y subdirectorios.

```
alfonso@dc7700p:~$ sudo chown -R alfonso:sicomo /usr/share/geoserver/
```

El parámetro *-R* indica que el cambio de propietario también se aplique a los subdirectorios.

Ya lo tenemos todo. Nos colocamos en el directorio donde se encuentra el ejecutable del servidor (si no lo estábamos ya) y lo ejecutamos con un script que incluye el propio repositorio.

```
alfonso@dc7700p:~$ cd /usr/share/geoserver/bin/
alfonso@dc7700p:/usr/share/geoserver/bin$ ./startup.sh
```

Saldrán un montón de letras y, si no da error, podremos acceder a la interfaz web del servidor mediante la dirección http://127.0.0.1:8080/geoserver

También se ha añadido una tarea *cron* para comprobar cada hora que se está ejecutando el servidor. En caso de no estar ejecutándose, lo arranca de nuevo.

```
alfonso@dc7700p:/usr/share/geoserver/bin$ crontab -e
```

5-20 Abrimos el editor de tareas

```
 GNU nano 2.5.3                  Archivo: /tmp/crontab.v6d5dJ/crontab

GEOSERVER_HOME = /usr/share/geoserver
@hourly pidof java || (echo 'Executing..' && sh /usr/share/geoserver/bin/startup.sh) #Geoserver runs
```

Indicamos la variable que hemos añadido anteriormente al archivo *.profile*. Esto es necesario porque, al no iniciar sesión con el usuario (el servidor será un servicio de fondo) no se establece la variable, por lo que indicamos a cron que la establezca.

La segunda línea pide el *pid*[17] de java (solamente tenemos una instancia de java funcionando en el servidor. En caso de tener más servicios de fondo que funcionasen con java, deberíamos mejorar el script). Como hay un *OR*, si la orden *pidof java* devuelve algún valor, no se ejecutaría lo de la derecha del *or*, es decir, lo que está entre paréntesis. En caso de no devolver ningún valor, se mostraría por pantalla (en este caso en el *syslog*, ya que no se ejecuta de forma interactiva) lo que está entre comillas simples y, seguidamente, se ejecutaría el servidor.

Esta tarea se hizo porque los errores del servidor provocaban que éste se cerrara. Los errores podían ser lanzados por diversas maneras: subir capas corruptas, usar símbolos no aceptados, agotar la memoria disponible para el proceso, etc…. También provoca que el servidor se ejecute si el PC es reiniciado tras un corte de corriente.

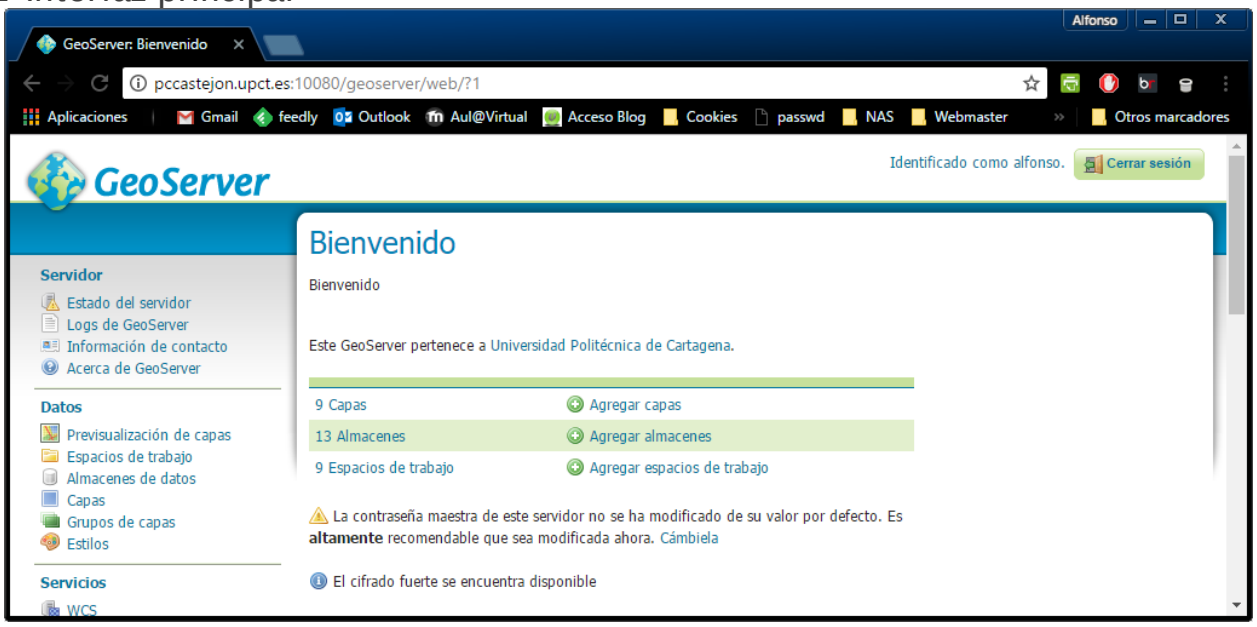
El puerto ha sido cambiado al 10080 ya que el que venía por defecto tenía restringido la conexión desde el exterior por el firewall de la universidad. Este parámetro se cambia editando el archivo */usr/share/geoserver/start.ini*

```
# HTTP port to listen on
jetty.port=10080
```

5-21 Cambiamos el puerto de escuhca por defecto

[17] Abreviatura de process ID, o sea, ID del proceso. Es un número entero usado por el kernel de algunos sistemas operativos para identificar un proceso de forma unívoca.

5.3.2 Interfaz principal



5-22 Página principal de GeoServer

Introducimos nuestro nombre y contraseña. La primera vez que accedamos deberá ser con las credenciales por defecto (usuario admin y contraseña *geoserver*). Es muy recomendable cambiarlas la primera vez que accedamos para evitar accesos no deseados.

Si hemos subido el proyecto como se ha indicado en el apartado 6.2.3 la ventana será similar a la de la imagen 6-22. En caso contrario la instalación incluirá unas capas de ejemplo.

A la izquierda aparece el menú principal desde el que se realizan todas las operaciones del servidor.

- **Estado del servidor.** Muestra el estado y los parámetros del servidor. También permite liberar memoria y borrar cache.
- **Logs de GeoServer**. Muestra los log del servidor. Útil si queremos determinar fallos del servidor.
- **Información de contacto**. Aquí establecemos la información pública de contacto del servidor. Estos datos serán accesibles a través de las peticiones WMS.
- **Acerca de GeoServer**. Información de las versiones de software y de los módulos así como enlaces a la documentación oficial.

- **Previsualización de las capas**. Desde aquí podemos obtener una vista previa de la capa que deseemos en diferentes formatos. Útil para visualizar y explorar capas antes de añadirlas al servicio web. La configuración por defecto permite que no haya que estar logueado para previsualizar las capas. Esta opción se puede cambiar.
- **Espacios de trabajo**. Lista de espacios de trabajo disponibles en el servidor. También permite añadir, editar y/o borrar los ya disponibles. Un espacio de trabajo tendrá varios almacenes de datos.
- **Almacén de datos**. Aquí tenemos los almacenes de datos de cada espacio de trabajo. Podemos añadir o borrar un almacén. Cada almacén de datos está compuesto de una o más capas.
- **Capas**. Lista de capas que hemos añadido. Desde aquí podemos añadir, editar o borrarlas.
- **Grupos de capas**. Podemos agrupar las capas de manera que de una sola petición obtengamos varias capas superpuestas. Esto es útil si queremos añadir dos capas como capa base por ejemplo.
- **Estilos**. Estilos que se asociarán a cada capa. Por ejemplo, si tenemos una capa de carreteras, podríamos tener un estilo con el tipo de línea, su grosor y su color. Aquí tenemos los estilos que tenemos en QGIS, ya que estos han sido subidos al usar el plugin..
- **Servicios**. Aquí podemos configurar los parámetros de cada uno de los servicios que ofrece nuestro servidor.
- **Global**. Para configurar los mensajes, el conjunto de caracteres utilizados e incluso el uso de un proxy.
- **JAI**. Abreviatura de Java Advanced Imagin. Desde esta sección podemos ajustar los diferentes parámetros para la generación de teselas para los servicios WMS y WCS.
- **Coverage access**. En este apartado se configura el pool de ejecuciones y el cacheado de ImageIO.
- **Capas en cache**. Desde aquí podemos obtener una vista previa de las imágenes de las capas que ya han sido generadas, es decir, que se mantienen en la cache.

- **Valores por defecto de cacheado**. La sección anterior se ajusta desde aquí. Desde qué formato se ha de generar para la cache hasta las dimensiones de las teselas.

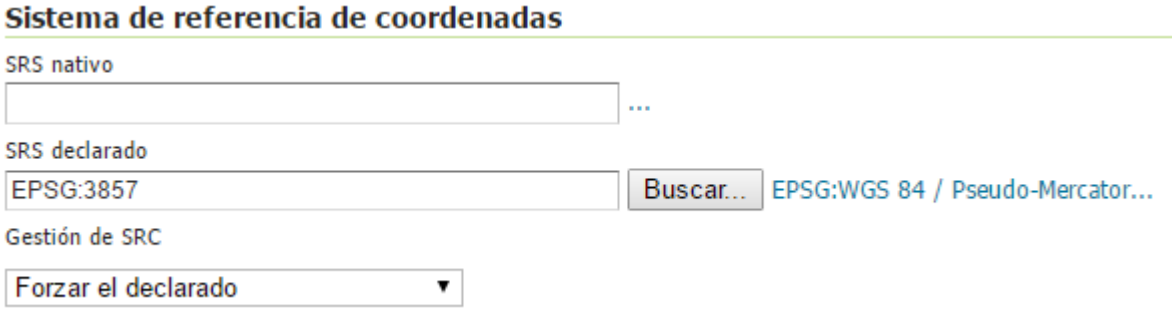
5-23 Menú principal situado en la izquierda de la interfaz

5.3.3 Configurando las capas que hemos añadido

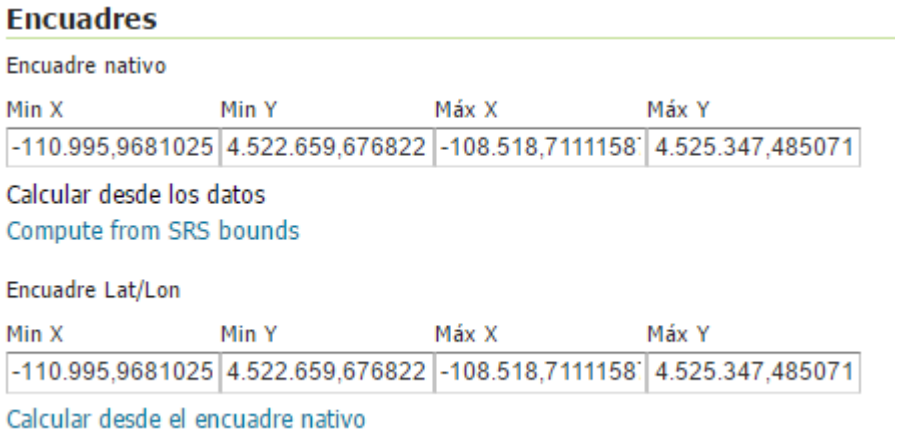
Tipo	Title	Nombre de la capa	Almacén	Habilitada?	SRS nativo
	Hidrante	cartagena:Hidrante	Hidrante	✓	EPSG:3857
	CuadrosMando	cartagena:CuadrosMando	CuadrosMando	✓	EPSG:3857
	Imbornal	cartagena:Imbornal	Imbornal	✓	EPSG:3857
	Luces	cartagena:Luces	Luces	✓	EPSG:3857
	Red abastecimiento	cartagena:redabto	Redabto	✓	EPSG:3857
	Red saneamiento	cartagena:Redsane	Redsane	✓	EPSG:3857
	Contenedores	cartagena:Contenedores	todos	✓	EPSG:3857
	arbolado	cartagena:arbolado	arbolado	✓	EPSG:404000
	PozoReg	cartagena:PozoReg	PozoReg	✓	EPSG:3857

5-24 Capas que tenemos una vez hemos borrado las de demostración y hemos subido el proyecto con el QGIS

En la imagen observamos que el SRS nativo de una de las capas no coincide con la proyección que necesitamos. Esta es la razón de haberlas convertido con el QGIS. Vamos a modificarla pulsando sobre el nombre de la capa.



5-25 Nos desplazamos hacia abajo y en el SRS declarado escribimos EPSG:3857 y forzamos el declarado, ya que es al que hemos convertido la capa con el QGIS



5-26 En los encuadres pulsamos *calcular desde los datos* y *calcular desde el encuadre nativo*.

Cuando lo tengamos, nos desplazamos hacia abajo y pulsamos el botón de guardar. Ahora deberíamos tener las capas tal que así:

Tipo	Title	Nombre de la capa	Almacén	Habilitada?	SRS nativo
	Hidrante	cartagena:Hidrante	Hidrante	✓	EPSG:3857
	CuadrosMando	cartagena:CuadrosMando	CuadrosMando	✓	EPSG:3857
	Imbornal	cartagena:Imbornal	Imbornal	✓	EPSG:3857
	Luces	cartagena:Luces	Luces	✓	EPSG:3857
	Red abastecimiento	cartagena:redabto	Redabto	✓	EPSG:3857
	Red saneamiento	cartagena:Redsane	Redsane	✓	EPSG:3857
	Contenedores	cartagena:Contenedores	todos	✓	EPSG:3857
	arbolado	cartagena:arbolado	arbolado	✓	EPSG:3857
	PozoReg	cartagena:PozoReg	PozoReg	✓	EPSG:3857

5-27 Todas las capas ya tienen la proyección que queremos.

Las opciones de los demás apartados las vamos a dejar por defecto, ya que no necesitamos tocar nada para este proyecto. Para proyectos más grandes o con mayor afluencia de tráfico deberíamos modificar, por ejemplo, el cacheado de teselas.

Esta es la principal razón de convertir la proyección de las capas con QGIS ya que, como puede observarse en la imagen 6-24, GeoServer no reconoce la proyección nativa de la capa. Esto es debido a un bug a la hora de escribir la proyección. Si se dispusiera de la proyección nativa, no habría que convertirlas con QGIS, ya que GeoServer tiene la capacidad de convertirlas al vuelo y cachearlas para mostrarlas.

5.4 Servidor Apache y PostgreSQL

En las primeras etapas de implementación de la interfaz web se utilizaron ambos productos para montar una interfaz web ya *"prefabricada"*. Esta interfaz web tiene el nombre de *MapBender 3*.

Esta web requiere para funcionar correctamente:

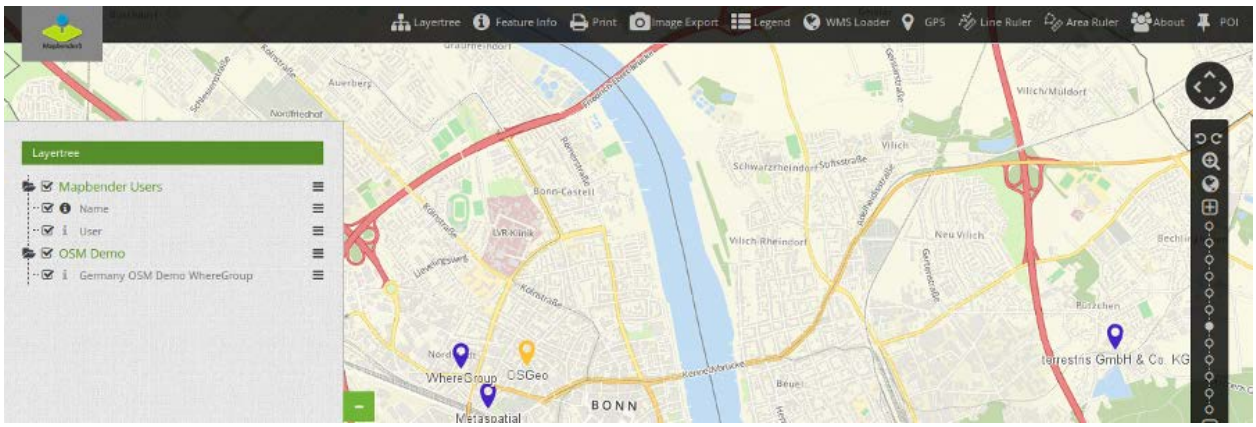
- PHP 5.5.4 como mínimo (php5)
- PHP CLI (php5-cli)
- Extensión PHP SQLite (php5-sqlite). Debido al bajo rendimiento como grandes bases de datos, se puede usar la extensión de PHP pgsql para PostgreSQL, de ahí que se haya usado.
- Extensión PHP cURL (php5-curl)
- PHP Internationalization (php5-intl). Para el soporte multi idioma.

- PHP GD para imprimir(php5-gd)
- PHP Multibyte String (php5-mbstring)
- PHP FileInfo.
- APACHE mod_rewrite.
- OpenSSL (opcional)

La gran ventaja de usar PostgreSQL es la existencia de una extensión llamada *PostGIS* que convierte dicho sistema de base de datos en una base de datos espacial. De hecho, en el blog especializado *mappingGIS* recomiendan su uso. [11].

La principal razón de la no elección de este sistema es la flexibilidad y escalabilidad que ofrece frente a la solución empleada. Si bien para lo que se lleva hecho del proyecto habría sido suficiente, no así a corto plazo donde se tenía intención de implementar rutas en tiempo real.

Como esta solución no ha sido finalmente implementada no se comentará en profundidad en este proyecto.



5-28 Interfaz principal de mapbender 3

6 Implementación

Para la implementación del servicio web se ha usado la librería javascript *Heron-MC*. Heron facilita la creación de aplicaciones de mapas web gracias a otra librería llamada *GeoExt*.

GeoExt es una librería bastante potente que combina las librerías *OpenLayers* para los mapas y *Ext JS* para la interfaz gráfica basada en la web.

Aunque existe escasa documentación en la web sobre este potente framework, siguiendo los ejemplos de los que dispone en la web original se puede sacar bastante partido, no sin antes realizar un aprendizaje sobre las funciones y sobre Javascript en general.

6.1.1 Página principal

Como toda página web, cualquier aplicación ha de tener un archivo php o html como archivo principal. En este caso se trata del archivo *index.html* y se puede ver su contenido:

```
<meta http-equiv="Content-Type" content="text/html;charset=utf-8"/>
```

Código 1 Indica el contenido del archivo (texto html) y la codificación del mismo (utf-8)

```
<title>Cartagena GIS :)</title>
```

Código 2 El título de la web que aparecerá en la barra del navegador

```
<link rel="stylesheet" type="text/css"
href="http://cdnjs.cloudflare.com/ajax/libs/openlayers/2.12/theme/de
fault/style.css"/>
```

Código 3 La hoja de estilos que vamos a usar. En este caso usamos una externa alojada en una CDN

```
<script type="text/javascript"
src="http://maps.googleapis.com/maps/api/js?key=80a507C5oPpHf9Y"></s
cript>
```

Código 4 Enlace a la librería de mapas de Google Maps junto con la key de la API. Usaremos Google Maps como uno de los mapas base. La API aquí expuesta es ficticia.

```
<link rel="stylesheet" type="text/css" href="js/ext-all.css"/>

<script type="text/javascript" src="js/ext-base.js"></script>

<script type="text/javascript" src="js/ext-all.js"></script>
```

Código 5 Librerías de Ext JS. En este caso usamos la versión que tenemos descargada, aunque se podrían usar las de la CDN.

```
<link rel="stylesheet" type="text/css" href="js/style.css"/>

<script type="text/javascript" src="js/OpenLayers.js"></script>
```

Código 6 Adición de las librerías OpenLayers. Usamos también una versión que hemos descargado previamente. Igualmente se pueden usar las disponibles en la CDN

```
<script type="text/javascript" src="js/GeoExt.js"></script>
```

Código 7 Aquí se indica que vamos a usar la librería GeoExt. Misma situación que las dos anteriores

```
<script type="text/javascript" src="js/Heron.js"></script>
```

Código 8 Llamada a la librería Heron. También podemos acudir a la CDN

```
<script type="text/javascript" src="01layers.js"></script>

<script type="text/javascript" src="02layertree.js"></script>

<script type="text/javascript" src="03layout.js"></script>
```

Código 9 Aquí va nuestro programa. Han de ir en ese orden o no funcionará. También se puede juntar todo en un mismo archivo, pero a la hora de editar es más versátil editar las partes por separado.

```
<script type="text/javascript" src="js/es_ES.js"></script>
```

Código 10 Indicamos el uso del lenguaje español.

6.1.2 Layers

En este archivo se definen las capas que vamos a utilizar en la aplicación web, así como las opciones del mapa y los botones que tendrá la barra superior. Además, se incluyen las secciones de la barra lateral izquierda.

Empezaremos dividiendo el código en *namespaces*, una manera eficiente y organizada de crear objetos con sus variables. Ideal para proyectos muy grandes donde queremos que las variables de esos objetos no sean globales. Esta función la aporta la librería Ext JS de *Sencha*.

Antes de empezar con el código vamos a ilustrar un ejemplo del uso de *namespaces*. [12]

En JavaScript las variables declaradas al principio son globales, es decir, pueden ser llamadas desde cualquier parte del código. Pongamos como ejemplo la función *trim*

```
function trim(str) {
  return str.replace(/^\s+|\s+$/g, "");
     }
```

Código 11 función *trim*

Esta función es accesible globalmente usando el nombre de la propia función. En una página que incluya funciones de diferentes sitios (como es el caso de este proyecto) se podría dar el caso de que existieran dos funciones con el mismo nombre pero con diferente fin. Usando los *namespaces* reducimos la posibilidad.

En el código tenemos un *namespace* tal que así:

```
Ext.namespace("Heron.options.map");
```

Código 12 Definición de un *namespace*

El equivalente sería:

```
if (!Heron) var Heron = {};
if (!Heron.options) var Heron.options = {};
if (!Heron.options.map) var Heron.options.map = {};
```

Código 13 Equivalencia de un *namespace*

Esto significa, si no existe la variable global *Heron,* créala. Si la variable Heron no tiene la propiedad *options*, créalo. Si esa propiedad no tiene la propiedad *map*,

créala. Des esta manera nos aseguramos de que si la variable existe, no crear una igual con diferente estructura.

Una vez explicado el funcionamiento de los *namespaces* pasamos a ver el código fuente del archivo. Esta vez lo mostraremos por partes solamente para no explayarnos demasiado.

```
Ext.namespace("Heron.options");
Ext.namespace("Heron.scratch");
```

Código 14 Definimos dos de los namespaces que vamos a utilizar.

De ahora en adelante los namespaces se incluirán pero no se comentarán, ya que todos tienen la misma finalidad.

```
OpenLayers.Util.onImageLoadErrorColor = "transparent";
OpenLayers.ProxyHost = "/cgi-bin/proxy.cgi?url=";
OpenLayers.DOTS_PER_INCH = 25.4 / 0.28;
```

Código 15 Opciones de Openlayers; cuando no sea posible cargar una capa, que la capa error sea transparente. El proxy es debido a una restricción de seguridad, ya que javascript no permite hacer peticiones XMLHttpRequest a servidores remotos. Dots per inch es usado para calcular la resolución del mapa.

6.1.2.1 Heron.options.map.settings

```
Heron.options.map.settings = {...};
```

Código 16 Vamos a comentar el texto que hay dentro de esta sección

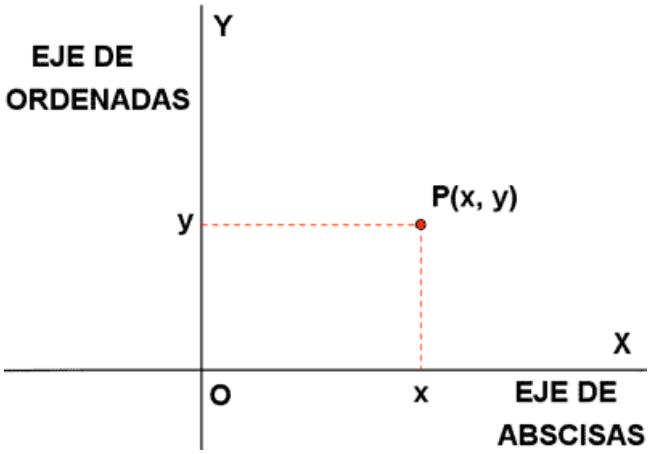
```
projection: 'EPSG:3857', //google maps
units: 'm',
```

Código 17 Usamos la proyección que usa Google, como se comentó en anteriores capítulos. Las distancias se medirán en metros.

```
maxResolution: 'auto',
maxExtent:        "-136295.86944710786,4517104.980097919,-76847.70956450791,
4540034.727132753",
center: "-109597.211,4527649.899",
```

Código 18 La resolución máxima dejamos que sea elegida automáticamente. maxExtent indica la porción máxima del mapa que será mostrada (área reducida al campo de Cartagena). Centramos el mapa con el atributo *center*

Las coordenadas del atributo *maxExtent* se proporcionan:



6-1 El formato de representación de coordenadas es [min_X,min_Y,max_X,max_Y]

En la ilustración, el punto P sería en donde se centra el mapa. Las coordenadas pueden ser negativas, pues el planeta entero sería:

maxExtent:"-20026376.39,-20048966.10, 20026376.39,20048966.10"

```
zoom: 13,
```

Código 19 El zoom de partida, pues solamente nos interesa el campo de cartagena

6.1.2.2 Heron.scratch.urls

```
Heron.scratch.urls = {...};
```

Los objetos aquí definidos son direcciones de servicios WMS. Se ha hecho así para, a la hora de pedir capas, no haya que escribir la dirección entera.

```
IGN: 'http://www.ign.es/wmts/ign-base?service=WMTS',
GEOSERVER_TMS: 'http://gis.vivancos.eu:8078/geoserver/gwc/service/tms',
GEOSERVER:'http://pccastejon.upct.es:10080/geoserver',
CALLEJERO:"http://callejero.murcia.es/wms?",
CALLEJERO_WFS:"/wfs",
CARTOMUR:"http://cartomur.imida.es/SgdWms3D/SgdWms_cartomur.dll/WMS?",
CATASTRO:"http://ovc.catastro.meh.es/Cartografia/WMS/ServidorWMS.aspx?",
CARTOGRAFIA: "http://geo.cartagena.es/wmts_carto/wmtservice.aspx"
```

Código 20 Lista de direcciones que proporcionan servicos WMS (y algunos más). Nuestro GeoServer se encuentra en la tercera línea :)

Algunas de estas direcciones han sido usadas en pruebas o con intención de integrarlas más adelante. Se ha optado por dejarlas para futuras referencias.

```
Heron.PDOK.urls = {
      CARTAGENA: Heron.scratch.urls.GEOSERVER + '/cartagena/wms?'
};
```

Código 21 Aquí abreviamos al igual que en el código anterior, la URL de nuestro servidor WMS + el workspace (cartagena)+ servicio (WMS)

6.1.2.3 Heron.scratch.layermap

```
Heron.scratch.layermap = {…};
```

Aquí vamos cargando las capas una a una. Como la forma de cargarlas es igual en casi todas, pondremos el código de algunas de ellas. En el apartado 6.4.2.4 tenemos la lista de capas que hay en nuestra aplicación web.

```
openstreetmap: ["OpenLayers.Layer.OSM", "OSM"],
gmaps: ["OpenLayers.Layer.Google", "Google Streets",
            {
                type: google.maps.MapTypeId.TERRAIN,
                isBaseLayer: true,
                buffer: 0,
                visibility: false,
                singleTile: false,
                sphericalMercator: true
            }    ],
```

Código 22 Definición de las capas de OSM y de Google

Como primera palabra tenemos la propiedad que comentábamos en 6.4.2 sobre los *namespaces*. Estos nombres son `openstreetmap` y `gmaps`. El primer valor del array es el tipo de capa a cargar. `OpenLayers.Layer.OSM` es el valor predefinido para cargar una capa base de OpenStreetMap. `OpenLayers.Layer.Google` es para las capas de Google. Esta capa sí tiene opciones.ç

La segunda propiedad es el nombre que le vamos a poner a la capa, al que haremos referencia. En este caso `OSM` y `Google Streets`

La tercera propiedad se aplica solamente a la capa de Google. Esta es un array de más propiedades. Las propiedades usadas son:

- **Type:** El tipo de capa que le vamos a pedir a Google
  - google.maps.MapTypeId.SATELLITE: Vista de satélite
  - google.maps.MapTypeId.ROADMAP: Mapa de carreteras.
  - google.maps.MapTypeId.TERRAIN: Mezcla de los dos anteriores.
- **isBaseLayer:** Indicamos que es una capa base. De esta manera obligamos a que, al menos una capa base, quede marcada.
- **Buffer:** Indicamos cuántas teselas que queden fuera de la vista cargaremos. Lo dejamos a 0 o la carga será más lenta.
- **Visibility:** Indica si la capa es visible. Como hemos indicado que es una capa base, este atributo no afecta a la visibilidad de la misma.

- **singleTile:** Indica si carga el área visible como una única tesela o como varias teselas.
- **sphericalMercator:** El valor es *true* ya que vamos a usar dicha proyección.

```
arbolado: ["OpenLayers.Layer.WMS", "arbolado",
            Heron.PDOK.urls.CARTAGENA,{
            layers: "cartagena:arbolado",
            format: "image/png",
            transparent: true,
            isBaseLayer: false,
            singleTile: false,
            visibility: true,
            alpha: true
            }]
```

Código 23 Carga de la capa de árboles

Lo nuevo en esta definición de capa es la tercera propiedad (`Heron.PDOK.urls.CARTAGENA`) . Esta propiedad hace referencia a la URL donde se va a realizar la petición WMS de la capa que indicamos en `layers` (`cartagena:arbolado`).

Las nuevas propiedades aquí son:

- **format:** El formato que queremos que nos sea devuelta la capa. Hemos elegido png pero podríamos haber pedido jpg, ya que GeoServer permite ambos formatos, además de los vistos en el apartado 5.2.
- **transparent:** Queremos que la capa sea transparente, de lo contrario se taparán unas a otras y no se vería ni siquiera la capa base.
- **Alpha:** Una de las características que posee el formato png es el de la transparencia. Indicando que esta capa tiene canal alfa indicamos que puede ser transparente.

Al pedir el formato png al servidor se puede omitir la propiedad de transparencia, ya que se da por hecho que ese formato de capas es transparente [13].

6.1.2.4 Heron.options.map.layers

En este corto apartado se indican qué capas se van a añadir al árbol final. Como siempre, esta propiedad es otro array.

```
Heron.options.map.layers = [
      Heron.scratch.layermap.redabto,
      Heron.scratch.layermap.hidrante,
      Heron.scratch.layermap.redsane,
      Heron.scratch.layermap.gmaps,
      Heron.scratch.layermap.luces,
      Heron.scratch.layermap.openstreetmap,
      Heron.scratch.layermap.cmando,
      Heron.scratch.layermap.imbornal,
      Heron.scratch.layermap.pozoreg,
      Heron.scratch.layermap.contenedores,
      Heron.scratch.layermap.arbolado
];
```

Código 24 Lista de capas que se van a añadir a la aplicación final. Aunque esté definidas en el apartado anterior, si no aparecen aquí, no pueden ser añadidas.

6.1.2.5 Heron.options.map.toolbar

Aquí se define la barra de botones de la parte superior del mapa.

6-2 Botones superiores

```
{type: "featureinfo", options: {…},
```

6-3 El primer botón empezando por la izquierda de la ilustración 10-28 se define aquí. Las opciones del mismo también se definen en este mismo sitio.

```
{type: "-"} ,
{type: "pan"},
{type: "zoomin"},
{type: "zoomout"},
{type: "zoomvisible"},
{type: "coordinatesearch", options: {onSearchCompleteZoom: 8}},
{type: "-"} ,
{type: "zoomprevious"},
{type: "zoomnext"},
{type: "-"},
{type: "measurelength", options: {geodesic: false}},
{type: "measurearea", options: {geodesic: false}},
{type: "-"},
{type: "addbookmark"},
{type: "help", options: {tooltip: 'Help and info for this example',
contentUrl: 'help.html'}}
```

6-4 Los botones, su comportamiento y las posibles opciones de cada botón están definidos aquí. El resto de botones se puede encontrar en el archivo ToolbarBuilder.js

6.1.2.6 Info y bookmarks

Dejamos para el final dos *namespaces* que harán la función de paneles informativos o enlaces de interés. Se pueden customizar.

```
Ext.namespace("Heron.options.info");
Heron.options.info.html ='<div class="hr-html-panel-body”><div>';
```

6-5 Agregamos un namespace que más adelante se indicará que su posición es el lateral izquierdo. Este panel será el panel de info

```
Ext.namespace("Heron.options.bookmarks");
Heron.options.bookmarks =
        [
            {
                id: 'test1',
                name: 'Test One',
                desc: 'Enlace de prueba uno',
                layers: ['OpenBasisKaart OSM', 'TNO Boorgaten'],
                x: 133993,
                y: 473167,
                zoom: 10
            },
            {
                id: 'test2',
                name: 'Test 2',
                desc: 'Enlace de prueba dos',
                layers: ['Luchtfoto (PDOK)'],
                x: 194194,
                y: 465873,
                zoom: 13
            }
        ];
```

6-6 El panel bookmarks o enlaces favoritos. Puede ser desde una web externa, hasta una capa concreta con un zoom concreto.

6.1.3 Layertree

En este fichero se configura la estructura *en cascada* o *árbol* de las capas que se mostrarán en el panel de capas de la izquierda de la página.

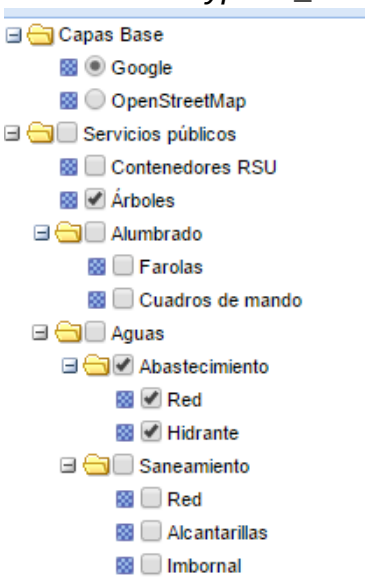
```
var treeTheme = [
 {text: 'Capas Base', expanded: true, children: [
  {nodeType: "gx_layer", layer: "Google Streets", text: 'Google' },
  {nodeType: "gx_layer", layer: "OSM", text: 'OpenStreetMap' }
  ]
 },
 {text: 'Servicios públicos', expanded: true, nodeType: 'hr_cascader',
children: [
  {nodeType: "gx_layer", layer: "contenedores", text: "Contenedores" },
  {nodeType: "gx_layer", layer: "arbolado", text: 'Árboles' },
 {text: 'Alumbrado', nodeType: 'hr_cascader', children: [
  {nodeType: "gx_layer", layer: "Luces", text: "Farolas" },
  {nodeType: "gx_layer", layer: "CMando", text: 'Cuadros de mando' }
 ]
 },
 {text: 'Aguas', nodeType: 'hr_cascader', children: [
  {text: 'Abastecimiento', nodeType: 'hr_cascader', children:[
   {nodeType: "gx_layer", layer: "redabto", text: 'Red' },
   {nodeType: "gx_layer", layer: "hidrantes", text: 'Hidrante' }
  ]
 },
 {text: 'Saneamiento' , nodeType: 'hr_cascader', children:[
  {nodeType: "gx_layer", layer: "redsane", text: 'Red' },
  {nodeType: "gx_layer", layer: "alcantarillas", text: 'Alcantarillas' },
  {nodeType: "gx_layer", layer: "imbornal", text: 'Imbornal' }
 ]}
 ]}
 ]}
];
Ext.namespace("Heron.options.layertree");
Heron.options.layertree.tree = treeTheme;
```

6-7 Ordenación de capas por nodos y categorías.

Aunque hay muchas opciones de ordenar las capas, se ha elegido la opción de sub capas desplegables por comodidad.

Cada grupo tiene unas subcapas (*children*) definidas por el *nodeType gx_layer*, lo cual posibilita la inclusión de un *checkbox* para activar o desactivar. Esto no es posible en las capas base. [14]

El *nodetype hr_cascader* en los grupos de capas nos permite activar/desactivar las capas en grupos en lugar de tener que ir una a una.



6-8 Orden de las capas según el código.

La penúltima línea define el *namespace* sobre el que se almacena el árbol de capas. Finalmente, en la última línea se indica que se aplique la organización que hemos definido (se puede ver en la primera línea) como *treeTheme*.

6.1.4 Layout

El último fichero de nuestra aplicación define la disposición de los elementos del mapa. Cuando hablamos de los elementos nos referimos a los cuatro paneles que componen la web.

```
Ext.namespace("Heron");
Ext.namespace("Heron.options");
Ext.namespace("Heron.options.layertree");
```

6-9 Las tres primeras líneas definen los namespaces. Se puede prescindir de ellas ya que están definidos en los archivos ya explicados.

Inicialmente definimos un único panel que será el contenedor de los otros cuatro paneles.

```
xtype: 'panel',
id: 'hr-container-main',
layout: 'border',
border: false,
items: […]
```

6-10 Definición del panel principal

Los cuatro paneles secundarios se engloban dentro del array *ítems*. Indicamos que es el panel principal, además de por la jerarquía, con el *xtype* panel.

Dentro del principal tenemos los otros cuatro a modo de ítems pertenecientes al principal. Dichos ítems son `hr-container-north`, `hr-menu-left-container`, `hr-map-and-info-container` y `hr-menu-right-container`.

El panel `hr-container-north` es el panel que contiene la imagen de cabecera

6-11 Panel de cabecera

El siguiente panel es el de la izquierda, `hr-menu-left-container`.Dicho panel es el contenedor de el árbol de capas y de los apartados *info* y *favoritos*. Estos últimos están definidos dentro del panel como *ítems*. Cabe destacar que en el apartado donde definimos el árbol de capas también se programan las acciones que tendrá el pulsar el botón derecho del ratón sobre una capa con la sintaxis `contextMenu`. ¿Y cómo indicamos que las capas van en este apartado? Con la opción `hropts` indicamos que el namespace `Heron.options.layertree` va aquí. Recordemos del apartado 6.4.3 la estructura de dicho árbol. También se podría poner todas las capas con su jerarquía aquí, pero romperíamos la modularidad del programa y si necesitáramos modificarlo no sería tan cómodo.

6-12 Ubicación de cada panel en la interfaz web

Además, hay también dos paneles que forman el *acordeón* al completo. Uno es el de Información y el otro es el de Favoritos.

El panel central, `hr-map-and-info-container`, contiene a su vez al mapa en sí con el identificador `hr-map`. En este, las `hropts` son `Heron.options.map` y todas las

sub propiedades y sub-objetos, como `Heron.options.map.settings` o `Heron.options.map.layers` (6.4.2.4).

El último panel, `hr-menu-right-container`, es el de la derecha. En este hay dos pestañas, a destacar la que muestra la leyenda de las capas que tengamos activas. Esto se consigue con `hr-layerlegend-panel`, que muestra la leyenda mediante *GetFeautreInfo* como se vio en el apartado 5.3

7 Conclusiones

El objetivo principal de este proyecto era el de dotar a la ciudad de Cartagena con un nexo entre las diferentes infraestructuras de las que dispone la ciudad en la actualidad. Si bien no se ha conseguido la totalidad del objetivo, en parte por trabas burocráticas, sí que se ha establecido la piedra angular del que podría ser un proyecto mucho más ambicioso. Eso ya dependerá del ayuntamiento y de las futuras inversiones.

Nos consta, tal y como se comentó en el apartado (), que en julio de 2016 se aprobó el Plan Director[18]. De esto se desprende la conclusión de que al ayuntamiento tiene la intención, si bien a medio-largo plazo, de continuar con la implantación y mejora de las infraestructuras orientadas a las *SmartCities*.

Como conclusiones podemos destacar:

- La utilización de las nuevas tecnologías en el día a día del funcionamiento de una ciudad ofrece unas posibilidades enormes para la gestión y optimización de los recursos y procesos. La aplicación web desarrollada en el presente TFG permite la recopilación de información sobre las infraestructuras principales de forma sencilla y rápida. Como línea futura se tiene la intención de dotar a la plataforma de la capacidad de recopilar dicha información incluso in situ, con la ayuda de un dispositivo móvil desde el que el propio técnico, e incluso un ciudadano, puede ir añadiendo información o informando de averías. En algunos casos, estas ventajas tecnológicas, pueden llegar a suponer un considerable ahorro económico y material.
- La información georreferenciada obtenida por la aplicación puede ser de una utilidad mayúscula, pudiéndose utilizarse en numerosos supuestos, como, por ejemplo:
    - Enviar alertas a los técnicos del ayuntamiento, que se encuentren en un radio de acción determinado en el momento de la aparición de una avería, reduciendo el tiempo de respuesta a la hora de solucionar dicha avería.

[18] www.cartagena.es/gestion/documentos/10860.pdf

- Obtener información sobre eficiencia en las zonas iluminadas de la ciudad que puede ser estudiada y analizada de forma local con el objetivo de aumentar la eficiencia del alumbrado de la ciudad y reducir el coste energético, que en el año 2016 ascendía a 6 millones de euros [15].
- La información georreferenciada almacenada en la base de datos, puede ser cruzada con la información de otras bases de datos, o incluso información en tiempo real, para obtener información aún más completa y compleja. Por ejemplo, se pueden cruzar los datos almacenados de la densidad del tráfico por tramos horarios con las rutas de los autobuses para optimizar las rutas.

8 Futuras líneas

La aplicación ofrece grandes posibilidades de desarrollo futuro gracias a su estructuración modular, pensada para ser ampliada y mejorada. Estas mejoras pueden ser:

- Automatizar el proceso de geolocalización de paradas de autobuses utilizando un dispositivo móvil. El usuario de la aplicación simplemente activaría ésta desde su dispositivo móvil, emitiendo el aviso de que nos encontramos en la parada para que obtenga la información de geolocalización (gracias al sistema GPS del dispositivo móvil), de forma que le aparezca toda la información relativa a esa parada (líneas de bus que paran, tiempo estimado de llegada, etc), sin tener que realizar la operación de búsqueda manual con los mapas de Google Maps.
- Ampliar las opciones de la aplicación añadiendo soporte para:
  - Gestión Integrada, que sea casi igual de versátil y efectivo el consultar/crear datos tanto de un dispositivo móvil como de un ordenador de sobremesa.
  - Cálculos de rutas, tanto para autobues como para los ciudadanos que circulen por la ciudad con su automóvil.
  - Gestión económica de los expendedores de la zona azul (O.R.A.)
    - Gestión técnica de las infraestructuras en general..
    - Recursos humanos y prevención de riesgos laborales.

        - Realización de cálculos estadísticos y análisis avanzados de la información, añadiendo módulos adicionales para la gestión estadística.
        - Añadir soporte, para otros programas o aplicaciones; posibilidad de exportar los datos y/o los resultados estadísticos a otros formatos para posibilitar la lectura desde otras aplicaciones.
- Posibilidad de guardar los datos históricos para colaborar con la *Iniciativa de datos abiertos del Gobierno de España*[19]. Otra forma de colaboración es permitir el acceso de sólo lectura a datos no sensibles a cualquier persona o institución que lo desee.

[19] http://datos.gob.es/

9 Referencias bibliográficas

[1] Naciones Unidas, «División de Desarrollo Sostenible de las Naciones Unidas,» Junio 1992. [En línea]. Available: http://www.un.org/spanish/esa/sustdev/agenda21/.

[2] B. S. T. Cárdenas, «Smart City: El avance del futuro,» 22 10 2015. [En línea]. Available: http://www.monografias.com/trabajos-pdf5/smart-city-avance-del-futuro/smart-city-avance-del-futuro.shtml.

[3] Universidad Politécnica de Cartagena, «Notas de prensa,» 15 Febrero 2016. [En línea]. Available: http://www.upct.es/saladeprensa/notas.php?id=4719.

[4] Wikimedia Foundation, «Programa 21,» 17 Febrero 2017. [En línea]. Available: https://es.wikipedia.org/w/index.php?title=Programa_21&oldid=97038881.

[5] E. Sahún, «La planificación de una ciudad inteligente: el caso de Cartagena,» 29 Noviembre 2016. [En línea]. Available: http://nae.es/la-planificacion-de-una-ciudad-inteligente-el-caso-de-cartagena/.

[6] «Fondo Estatal de Inversión Local,» 2009. [En línea]. Available: http://www.seat.mpr.gob.es/fondosinversionlocal/de-200001-a-500000/cartagena.html.

[7] C. San Emeterio Villalaín, «Desarrollo de una Aplicación Para La Realización De Medidas Del Canal MIMO-UWB En Interiores Basada En SIG,» 2011.

[8] ESRI, «Tamaño de celda de datos ráster,» 2012. [En línea]. Available: http://help.arcgis.com/es/arcgisdesktop/10.0/help/index.html#//009t00000004000000.

[9] Wikimedia Foundation, INC, «Web Mercator,» 27 Febrero 2017. [En línea]. Available: https://en.wikipedia.org/wiki/Web_Mercator.

[10] Open Source Geospatial Foundation, «History,» 31 Mayo 2017. [En línea]. Available: http://docs.geoserver.org/latest/en/user/introduction/history.html.

[11] A. MORALES, «7 motivos para utilizar PostGIS,» 25 Septiembre 2012. [En línea]. Available: https://mappinggis.com/2012/09/por-que-utilizar-postgis/.

[12] K. Krukow, «Designing client/server web-applications,» 28 Febrero 2008. [En línea]. Available: http://higher-order.blogspot.com.es/2008/02/designing-clientserver-web-applications.html.

[13] D. Tengshe, «Why 'Transparent' Parameter in OpenLayers is a Server Side Property?,» 17 Febrero 2013. [En línea]. Available: https://gis.stackexchange.com/questions/52714/why-transparent-parameter-in-openlayers-is-a-server-side-property.

[14] GeoExt, «API Reference,» 2010. [En línea]. Available: http://geoext.org/v1/lib/GeoExt/widgets/tree/LayerNode.html.

[15] Ayuntamiento de Cartagena, «PARTIDAS DE GASTOS,» 11 Enero 2016. [En línea]. Available: http://hacienda.cartagena.es/gestion/documentos/5174.pdf.

[16] The OWASP Foundation, «Testing for AJAX Vulnerabilities (OWASP-AJ-001),» 12 Mayo 2013. [En línea]. Available: https://www.owasp.org/index.php/Testing_for_AJAX_Vulnerabilities_(OWASP-AJ-001)#Attacks_and_Vulnerabilities.

[17] MICAELA, «Arrays en Javascript: la diferencia entre [ ] y { },» 5 Julio 2008. [En línea]. Available: http://www.elwebmaster.com/articulos/vectores-en-javascript-la-diferencia-entre-y.

[18] Instituto Geográfico Nacional, «Sistemas de Información Geográfica,» Enero 2016. [En línea]. Available: https://www.ign.es/ign/layoutIn/actividadesSistemaInfoGeografica.do.

[19] lazarus1907, «raster en español, ¿con o sin acento?,» 5 Febrero 2007. [En línea]. Available: https://forum.wordreference.com/threads/raster-en-espa%C3%B1ol-con-o-sin-acento.377765/#post-2226337.

[20] Wikimedia Foundation, «European Petroleum Survey Group,» 27 Noviembre 2014. [En línea]. Available: https://es.wikipedia.org/wiki/European_Petroleum_Survey_Group.

10 ANEXOS

10.1 El archivo PROXY.CGI

Este archivo, visto en el Código 5 tiene la función de permitir al código javascript realizar peticiones a páginas web que se encuentren en un servidor diferente al que se encuentra dicho código.

Esta restricción no es de javascript, sino de los navegadores. Se implementó para evitar ataques de XMLHttpRequest[20] tales como *SQL Injection*, *Cross Site Scripting* (XSS), etc. [16].

```
#!/usr/bin/env python

import urllib2
import cgi
import sys
import os

# Dominios de confianza
allowedDomains = [
    'ign.es',
    'cartomur.imida.es',
    'ovc.catastro.meh.es',
    'geo.cartagena.es',
      'gis.vivancos.eu:8078',
      'gis.vivancos.eu',
      'pccastejon.upct.es',
      'upct.es'
]

# Add to have specific hosts, mostly IP-addresses
allowedHosts = [
    '144.76.93.238'
]

method = os.environ["REQUEST_METHOD"]

# Fetch the url query param
if method == "POST":
    qs = os.environ["QUERY_STRING"]
    d = cgi.parse_qs(qs)
    if d.has_key("url"):
        url = d["url"][0]
    else:
        url = "http://www.openlayers.org"
else:
    fs = cgi.FieldStorage()
    url = fs.getvalue('url', "http://www.openlayers.org")

try:
    # Gives: www.openlayers.org
    host = url.split("/")[2].split(':')[0]
```

[20] Interfaz empleada para realizar peticiones HTTP y HTTPS a servidores Web

```
    # Gives: openlayers.org
    domain = '.'.join(host.split(".")[-2:])

    if (allowedDomains and domain not in allowedDomains) and (allowedHosts
and not host in allowedHosts):
        print "Status: 502 Bad Gateway"
        print "Content-Type: text/plain"
        print
        print "Blocked access to host/domain: %s/%s" % (host, domain)
        print
        print os.environ

    elif url.startswith("http://") or url.startswith("https://"):
        headers = {}

        if os.environ.get("HTTP_ACCEPT"):
            headers['Accept'] = os.environ["HTTP_ACCEPT"]
        if os.environ.get("HTTP_FIWARE_SERVICE"):
            headers['FIWARE-Service'] = os.environ["HTTP_FIWARE_SERVICE"]
            if os.environ.get("HTTP_FIWARE_SERVICEPATH"):
                headers['FIWARE-Servicepath']                              =
os.environ["HTTP_FIWARE_SERVICEPATH"]
            if os.environ.get("HTTP_X_AUTH_TOKEN"):
                headers['X-FI-WARE-OAuth-Header-Name'] = 'X-Auth-Token'
                headers['X-FI-WARE-OAuth-Token'] = 'true'
                headers['X-Auth-Token']                                    =
os.environ.get("HTTP_X_AUTH_TOKEN")

        if method == "POST":
            length = int(os.environ["CONTENT_LENGTH"])
            headers['Content-Type'] = os.environ["CONTENT_TYPE"]

            body = sys.stdin.read(length)
            r = urllib2.Request(url, body, headers)
            y = urllib2.urlopen(r)
        else:
            r = urllib2.Request(url, headers=headers)
            y = urllib2.urlopen(r)

        # print content type header
        i = y.info()
        if i.has_key("Content-Type"):
            print "Content-Type: %s" % (i["Content-Type"])
        else:
            print "Content-Type: text/plain"

        if i.has_key("Content-Disposition"):
            print "Content-Disposition: %s" % (i["Content-Disposition"])

        print
        # print str(os.environ)
        print y.read()

        y.close()
    else:
        print "Content-Type: text/plain"
        print
        print "proxy.cgi: illegal proxy request."

except Exception, E:
    print "Status: 500 Unexpected Error"
```

```
    print "Content-Type: text/plain"
    print
    print "proxy.cgi: some unexpected error occurred. Error text was:", E
```

10.2 Arrays en JavaScript

Una duda muy frecuente en el uso de **Javascript** es cuándo es necesario usar **llaves o corchetes** al trabajar con **arrays**.

Los corchetes se usan para series que poseen valores simples, mientras que las llaves son utilizadas para las series en que hay distintos objetos y propiedades con diferentes valores. Este último caso es bastante similar al sistema de PHP.

De todas formas, debemos tener en claro que en JS por convención, las llaves definen objetos y no arrays. Veamos un par de ejemplos:

Ejemplo del uso de corchetes “[ ]”:

```
var answers = ['yes','no','maybe'];
var names = ['David','Kristina','Charlie','Angela'];
```

Ejemplo del uso de llaves “{ }”:

```
var programmer = { 'name':'David Walsh', 'url':'http://davidwalsh.name',
'girl':'Kristina'}
```

[17]