Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

**UNIVERSIDAD DE JAÉN**
*Escuela Politécnica Superior*

Trabajo Fin de Grado

# INTEGRACIÓN EN APLICACIONES DE MAPAS DE MAPAS DE VISIBILIDAD

**Alumno: José Morón Rodríguez.**

Tutores: Rafael Jesús Segura Sánchez
José María Noguera Rozúa
Dpto.: Informática.

**Septiembre, 2015**


Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Rafael Segura Sánchez, tutor del Trabajo fin de Grado titulado: Integración en aplicaciones de mapas de mapas de visibilidad, que presenta José Morón Rodríguez, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2015

El alumno:

MORON RODRIGUEZ, JOSE (FIRMA)
Firmado digitalmente por MORON RODRIGUEZ, JOSE (FIRMA)
Nombre de reconocimiento (DN): c=ES, serialNumber=77349558Y, sn=MORON, givenName=JOSE, cn=MORON RODRIGUEZ, JOSE (FIRMA)
Fecha: 2015.09.02 10:10:44 +02'00'

José Morón Rodríguez

Los tutores:

NOMBRE SEGURA SANCHEZ RAFAEL JESUS - NIF 26003853F
Firmado digitalmente por NOMBRE SEGURA SANCHEZ RAFAEL JESUS - NIF 26003853F
Nombre de reconocimiento (DN): c=ES, o=FNMT, ou=FNMT Clase 2 CA, ou=701000772, cn=NOMBRE SEGURA SANCHEZ RAFAEL JESUS - NIF 26003853F
Fecha: 2015.09.01 12:55:54 +02'00'

Rafael Segura Sánchez

NOMBRE NOGUERA ROZUA JOSE MARIA - NIF 77343781W
Firmado digitalmente por NOMBRE NOGUERA ROZUA JOSE MARIA - NIF 77343781W
Nombre de reconocimiento (DN): c=es, o=FNMT, ou=fnmt clase 2 ca, ou=703001932, cn=NOMBRE NOGUERA ROZUA JOSE MARIA - NIF 77343781W
Fecha: 2015.09.01 22:23:47 +02'00'

José María Noguera Rozúa

# Índice

# 1. Introducción al problema

En estos meses el alumno José Morón Rodríguez ha analizado, diseñado y desarrollado el TFG, Integración en aplicaciones de mapas de mapas de visibilidad, el cual esta supervisado por Rafael Segura Sánchez y como segundo tutor José María Noguera Rozúa.

## 1.1. Descripción del problema.

El trabajo fin de grado, de ahora en adelante TFG, consiste en integrar los mapas de visibilidad en aplicaciones de mapas. El estudio de la visibilidad en el ámbito de la arqueología es uno de los principales usos de los SIG (Sistemas de Información Geográfica). Las aplicaciones de dichos estudios han ido desde estudios de monumentos, yacimientos o en este caso castillos, fortalezas, torres, torreones etc… Estos estudios le han permitido a los investigadores descubrir zonas defensivas aun no documentadas, son más importantes cuando se trata de periodos históricos, como en el Medievo, que era necesario crear redes de fortificaciones que cubrieran una gran área empleando los mínimos recursos. Uno de los problemas de estos estudios es que normalmente se obvian factores de la época, puesto que no se computan algunos aspectos de la época para el estudio de la visibilidad. Otro problema es que estos estudios solo se pueden realizar en aplicaciones de escritorio por lo tanto es complicado llevarlo al terreno de estudio a no ser que te imprimas el mapa dado por el SIG. Este trabajo tratará de dar solución a este último problema, traeremos de llevar los mapas generados por los SIG a las aplicaciones de mapas preferiblemente móviles para así poder ver estos mapas en cualquier parte

## 1.2. Motivación

Este trabajo está dentro de la mención de sistemas gráficos la cual está totalmente justificada ya que hay que conocer las distintas estructuras de datos que conforman los datos geográficos dentro de las aplicaciones de mapas. Dentro de la rama de Tecnologías de la Información este trabajo se puede integrar ya que es necesario conocer las tecnologías para procesar las imágenes y también las herramientas necesarias para servir los mapas a través de internet.

Otro motivo por el cual escogí este trabaja es mi gran afición por los castillos de la provincia de Jaén, la cual es una de las zonas con más castillos de España. También está mi fascinación por los mapas, y este trabajo me ha permitido estudiar más sobre este tema.

## 1.3. Objetivos

El objetivo principal de este proyecto es llevar los mapas de la visibilidad desde las aplicaciones de escritorio a aplicaciones de móviles. Para ello hay que realizar un análisis del problema, un diseño de la solución e implementar dicha solución.

Para llevar a cabo este objetivo se necesitara:

- Realizar un estudio de los mapas de visibilidad. Que son los mapas de visibilidad, que se necesita para sacar estos mapas, y con qué aplicación se obtienen.
- Buscar una forma de poner los mapas de visibilidad en los dispositivos móviles, ya sea por medio de una aplicación móvil o una aplicación web.
- Estudio la georreferenciación de imágenes, que es necesario para georreferenciar una imagen dentro de un SIG y que es necesario para llevarlo a aplicaciones móviles.
- Analizar las aplicaciones móviles nativas y las aplicaciones web. Analizar cuál es mejor para nuestro problema.
- Diseño e implementación de la aplicación.

## 1.4. Estructura del documento.

Éste documento está estructurado en distintos capítulos con subapartados:

En primer lugar analizaremos el problema a resolver, recopilaremos los objetivos del trabajo se analizaran distintas soluciones y se escogerá la que más se ajuste a ella.

A continuación abordaremos el diseño del sistema realizando los diagramas y esquemas que sean necesarios para que el equipo de desarrollo pueda consultar y que les sirva de guía en la programación. También se planificaran las tareas y se dará un coste del trabajo a realizar.

En el siguiente capítulo trataremos los detalles de la implementación de acuerdo con los detalles de diseño estudiados anteriormente.

Por último, tras la finalización del proyecto tendremos una reflexión sobre las conclusiones a las que hayamos llegado y complementaremos con posibles mejoras del proyecto.

# 2. Análisis

En este capítulo analizaremos en los distintos problemas del TFG e intentaremos dar una solución óptima para cada uno de ellos. Al final del capítulo se describirá una solución inicial.

## 2.1. Mapas de visibilidad

Como hemos dicho antes los mapas de visibilidad son una de las aplicaciones típicas de los SIG dentro del ámbito de la arqueología. Estos estudios representan el campo visible desde un punto dentro de un mapa. Pero qué necesitamos para realizar estos estudios, muchas son las aplicaciones que realizan estos estudios pero todas necesitan los siguientes elementos:

- **Un MDE**. Para poder realizar el estudio es obligatorio disponer de un Modelo Digital de Elevaciones de la zona en la que vamos a realizar el estudio.
- **Un punto de referencio u observación**. Será el punto desde el cual se realice los cálculos de visibilidad. En nuestro caso serán los castillos, torreones o fortalezas. También puede ser varios puntos pero en este caso se procederá a realizar los cálculos desde cada punto para obtener la cuenca visual completa.
- **Una altura**. La altura de punto de referencia u observación. No nos saldrá el mismo resultado si la altura que consideremos es la del torreo más alto o de una persona.
- **Un radio**. Esta distancia será el alcance que tendrá el mapa de visibilidad. Es muy importante tener una distancia óptima ya que a mayor radio la obtención del mapa puede que se demore demasiado.

Una vez visto los elementos necesarios para obtener el estudio de visibilidad de un punto o varios vamos a ver cuántos tipos de estudios podemos obtener. Aunque son varios los tipos que podemos obtener vamos a enumerar los principales:

- **Un sólo campo visual**. Éste algoritmo es el más sencillo de todos, simplemente calcula si una celda del MDE es visible desde otra. Como resultado da una matriz binaria en el que un valor “1” la celda es visible y “0” si no lo es.

- **Mapa de visibilidad múltiple**. Éste tipo consiste en lo mismo de lo anterior pero en vez de tener en cuenta un solo punto de observación, se tienen en cuentan varios puntos de observación. El resultado obtenido sigue siendo una matriz binaria en la cual el valor “1” es que la celda es visible desde algún punto y el valor “0” no es visible desde ningún punto.
- **Visibilidad acumulativa**. Éste algoritmo también tiene en cuenta varios puntos de observación. Consiste en sumar las distintas matrices, obtenidas por uno de los métodos anteriores, en una sola y así obtener una matriz la cual ya no es binaria si no que en cada celda tendrá un valor entre 0 y N, siendo N el número de puntos de observación. Si una celda tiene valor N significa que es visible desde todos los puntos de observación.
- **Visibilidad Total**. Éste tipo de estudio es quizá el que más cueste de procesar, puesto que el cálculo que realiza es el siguiente, calcula todas las celdas visibles desde todas las celdas existentes en el mapa. Si nuestra área es muy grande y el MDE que usamos es muy preciso, el resultado será un tiempo de procesamiento muy considerable.

### 2.1.1. Sistemas de información Geográfica (SIG)

Un SIG es un sistema hardware y software que permite capturar, gestionar, editar, analizar y visualizar datos espaciales.

Estos sistemas representan más que un simple mapa ya que son capaces de representar información de modo visual de forma automática, con diversos formatos y adaptándose a la escala, necesidades y distintos formatos de salida.

Suelen manejar dos tipos de datos, vectoriales o raster. Los datos vectoriales son aquellos que se representa mediante puntos, líneas y/o polígonos. Los datos raster son aquellos que son una representación discreta de una superficie real, esta representación es una matriz de NxN en la cual cada celda representa un valor asociado a una porción cuadrada del mundo real.

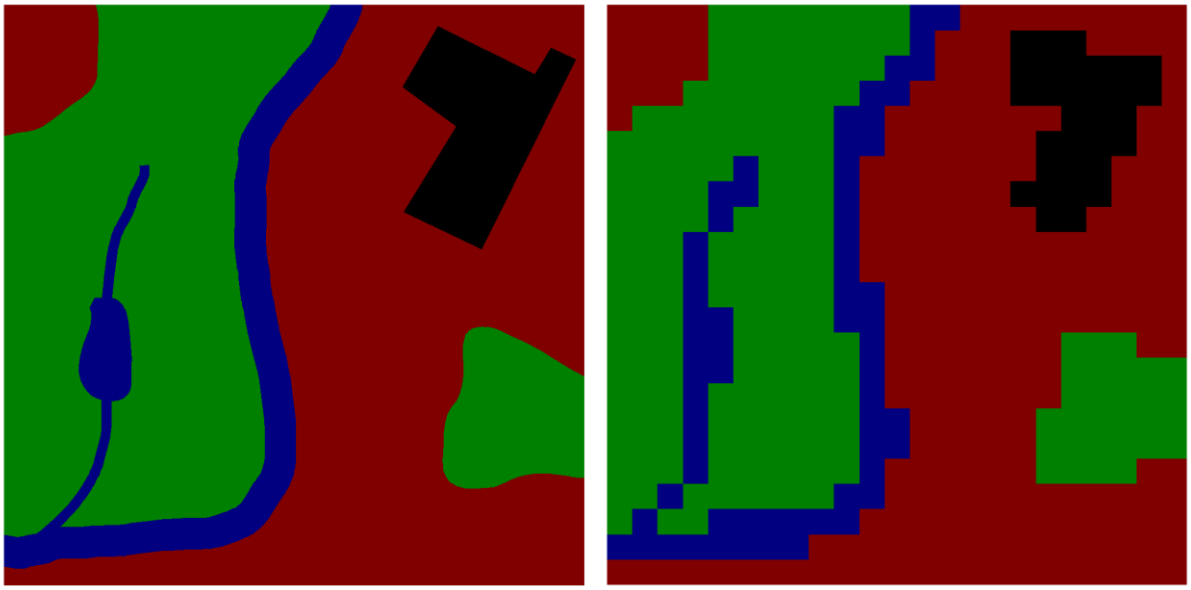


Ilustración 1 Interpretación cartográfica vectorial (izquierda) y raster (derecha) de elementos geográficos.

Dentro de los elementos necesarios para obtener un mapa de visibilidad uno de ellos es un MDE (Modelo Digital de elevaciones) este modelo es un dato raster en el cual cada celda representa la altura media de esa región. Para obtener los mapas de visibilidad se ha utilizado un MDE de la provincia de Jaén en cual cada celda es de $50m^2$.

Otra de las características de los SIG es que son capaces de interpretar cualquier proyección cartográfica; una proyección cartográfica es un sistema de representación grafico que establece una relación entre los puntos de la curvatura de la tierra con los puntos de la superficie plana del mapa. Otro de los elementos es un punto de observación o varios, este punto tiene que estar en la misma proyección en la que este el MDE. Cabe destacar que según donde te descargar el MDE así será el sistema de coordenadas. Para este TFG se ha creado un MDE a partir de datos obtenidos por la junta de Andalucía, estos datos se encontraban en la proyección ED50 UTM y su código ESPG es 23030.

Un SIG para mostrar la información visual ya sea raster o vectorial utiliza un sistema de capas o una pila de Datos. Los primeros datos en llegar serán los primeros que se dibujen en pantalla.

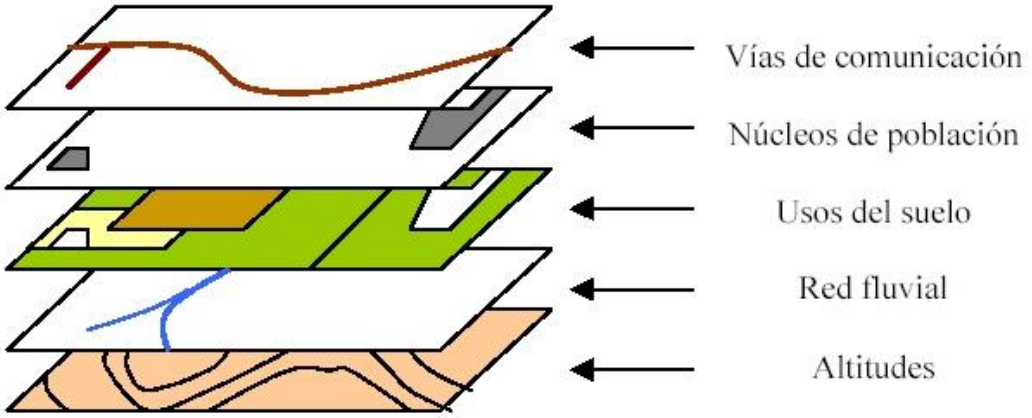



Ilustración 2. Ejemplo de capas de un SIG

En el mercado hay varios SIG que te dan los mapas de visibilidad pero al ser de pago han sido descartados ya que la obtención de los mapas de visibilidad es algo que te da cualquier SIG libre.

### 2.1.2. Conclusiones

Como se puede observar no hemos analizado el algoritmo para obtener la cueca visual de una región a partir de un punto ya que es no es uno de los objetivos del trabajo. Hemos analizado los distintos elementos y las herramientas necesarias para obtener los mapas de visibilidad.

Después de haber analizado distintos SIG y ver los distintos algoritmos se ha llegado a la conclusión que se debe elegir un SIG gratuito ya que hay muchas opciones, las cuales obtienen resultados bastante notables. Un SIG gratuito y con una gran comunidad hispanohablante es el gvSIG, cuenta con licencia GNU GPL v3. Fue desarrollado por la comunidad Valenciana en el año 2004, pero actualmente está impulsado por la asociación gvSIG. Esta desarrollado en Java funcionando con cualquier sistema operativo. También cuenta con las herramientas necesarias para obtener los cuatro tipos de mapas de visibilidad antes expuestos, por lo tanto se ha escogido este software para la obtención de mapas de visibilidad.

## 2.2. Aplicaciones móviles

Como se ha expuesto inicialmente el cálculo y la visualización de mapas de visibilidad deben realizarse en aplicaciones de escritorio imposibilitando llevar estos

mapas a aplicaciones móviles. En este capítulo vamos a estudiar las distintas aplicaciones móviles y a elegir las más adecuada para nuestro problema.

Dentro de las aplicaciones móviles que podemos realizar existen tres tipos

- **Aplicación móvil nativa**: este tipo son que se desarrollan para un tipo de dispositivos móvil. Tienen la ventaja que pueden acceder completamente al dispositivo (Sensores, Notificaciones, Cámara, etc…). Pero en su contra tienen que desarrollarse una para cada tipo de dispositivo (iPhone, iPad, Android, Windows Phone, etc…)

- **Aplicación web:** este tipo se desarrollan con tecnologías Web (HTML, PHP, JSP, JavaScript, CSS, etc...) y pueden ser vistas desde cualquier dispositivo. En su contra esta que tienes que tener conexión a internet, no puedes hacer uso de todos los sensores del dispositivo, y tampoco tienes acceso al dispositivo.
- **Aplicación Hibrida**. Este tipo es una mezcla de los dos anteriores anqué su experiencia de usuario es como la de una aplicación web puede ser desarrollada a la vez para cualquier plataforma, ya que está construida con tecnólogas web

Para este trabajo se va a realizar una aplicación web ya que al tratarse de una aplicación de mapas se necesita que el usuario esté conectado continuamente para poder descargar los mapas y siendo esto un requisito más que un inconveniente y teniendo en cuenta los distintos dispositivos portátiles (móviles, Tablet, notebook, etc...) que hay hoy en día en el mercado nos tenemos que decidir por desarrollar una aplicación web. Además de esta forma se puede llegar a más usuarios con solo un proyecto.

### 2.2.1. Aplicaciones WEB

Una aplicación web es una tipo de aplicación Cliente/Servidor donde el cliente normalmente es un navegador y el protocolo que se utiliza es HTTP. En la actualidad hay varios lenguajes en los cuales poder programar estas aplicaciones los dos más destacados son PHP y JAVA con sus Java Server Pages (JSP).

Para desarrollar un proyecto en JSP se necesita un servidor compatible con contenedores Servlet, normalmente estos servidores encarecen los servicios del hosting ya que necesitan de una arquitectura especial.

Otro de los lenguajes es PHP, las características que han hecho decantarse por este lenguaje son:

***Simple***: Permite generar código de una forma rápida, facilita la reutilización y tiene una curva de aprendizaje realmente rápida. Del mismo modo, el mantenimiento se hace más sencillo.

***Rápido***: No necesita demasiados recursos del sistema y tiene unos tiempos de respuesta muy aceptables.

***Seguro***: Bien utilizado, ofrece protección contra ataques.

***Estable***: Posee un sistema de administración de recursos y manejo de variables bastante fuerte y estable ante caídas.

Otro de los aspectos relevante a decidir en una aplicación web es el uso de Framework. Un FrameWork define unas normas y unos estándares de trabajo que hacen que el desarrollo de la aplicación sea más rápida y más fácil. Para aplicaciones web suelen estar basados en el patrón de diseño MVC (Modelo Vista Controlador)

- **Modelo:** El modelo es la parte que maneja las llamadas a las BBDD, deben de tener las operaciones CRUD básicas. Normalmente por cada entidad fuerte en la BBDD hay un modelo.
- **Vista:** Es la que interactúa con el usuario, mostrándole la información debidamente formateada. Se trata de la interfaz de usuario la cual está separada de los datos y de la lógica de negocio.
- **Controlador:** Aquí nos encontramos con la capa de negocio, básicamente procesa los datos llama a la vista para mostrarlos y al modelo para obtener o editar los datos de la BBDD.

Uno de los framework en php que cumplen estos requisitos es CodeIgnitier, Los puntos a favor de este framework sobre el resto de framework que se consideraron fueron los siguientes:

- Adaptado para PHP5. El uso correcto de la Programación Orientada a Objetos con PHP5 ofrece una aplicación más estable y rápida.
- Pequeña curva de aprendizaje. En muy poco tiempo puede aprenderse los conceptos básicos de manejo de controladores, uso de modelos y llamada a vistas; así como el uso de bibliotecas para ampliar funcionalidades del sistema.
- Patrón MVC. El framework es compatible con el patrón arquitectónico Modelo-Vista-Controlador.
- Facilidad de instalación. Basta con tener acceso FTP para subir Codeigniter en el servidor y su configuración se realiza en ficheros fácilmente identificables: configuración general, configuración de base de datos, paginación, sesiones... Es relativamente sencillo programar una pequeña aplicación que pregunte al usuario éstos parámetros y los inserte automáticamente en los ficheros adecuados.

Otro aspecto importante dentro de nuestra aplicación web es si se utiliza FrameWork para el front-end o para la interfaz de usuario, la cual sea más fácil de diseñar y de implementar. Para programar una aplicación web que sea adaptable para móviles necesitamos de este tipo de apoyo ya que sin él la duración del proyecto se elevaría considerablemente. El framework seleccionado para este motivo es BootStrap, te permite de forma sencilla y rápida adaptar el diseño para varios tipos de pantallas y obteniendo así un diseño responsivo.

## 2.3. Aplicaciones de mapas.

La cartografía web o web mapping es el proceso de diseñar, aplicar, generar y visualizar datos geoespaciales a través de Word Wide Web. Para nuestro proyecto lo que necesitamos que cumpla nuestra aplicación es visualizar mapas de alguna fuente cartográfica, por ejemplo Google Maps, Bing Maps o OpenStreetMaps, también es necesario que sea capaz de superponer imágenes a dichos mapas, que serán los mapas de visibilidad.

Para nuestro caso buscamos una solución la cual nos permita poner una capa base de cualquier cartográfica y poder añadirle más capas dinámicamente y de forma sencilla.

Cualquier servidor de mapas que tenga una API pude ofrecerte eso pero en nuestro caso está el problema de la proyección, como se ha comentado antes un mapa siempre tiene una proyección y todos los procesos que realices sobre dicho conjunto de datos tiene que ser sobre dicha proyección, y el proceso devuelve algún tipo de salida la salida estará sobre esa proyección.

Primero se pensó en Google maps pero resultaba muy complicado cambiar de proyección ya que la que utiliza Google maps no es la misma que la de nuestros mapas de visibilidad.

Así se investigó más sobre el tema y nos encontramos con una biblioteca JavaScript que funciona de forma muy parecida a un SIG, OpenLayers es una biblioteca de JavaScript de código abierto que puede cargar cualquier mapa de cualquier servidor además se puede crear capas con imágenes de forma dinámica y de manera muy sencilla. Gracias a que trabaja muy fácilmente con Pro4js, otra librería JavaScript para el cambio de proyecciones, no hay problemas de proyección ya que puedes transformar la proyección de manera muy sencilla y rápida.

## 2.4. Propuesta de solución.

Despues de haber analizado el problema y descompuesto el problema hemos visto soluciones para cada uno de estos subproblemas.

Primero el tipo de aplicación, desde un principio se pensó hacer una aplicación móvil nativa, pero después de estudiar los tipos de aplicaciones y ver las dificultades que conyevan las aplicaciones de mapas se optó por una aplicación web pero adaptada a móviles.

Luego está la parte de los mapas, al pensar que se iba a hacer una aplicación móvil nativa se pensó en google maps ya que en la actualidad hay mas usuarios Android que de otros sistemas. Pero al introducir los mapas de visibilidad dentro de un mapa de google maps y ver que hay que transformar las coordenadas de una

proyección a otra y que los márgenes de error eran bastantes altos (minimo 200m por pixel) se investigó un poco más sobre el tema y nos encontramos con la librería OpenLayers la cual daba solución a todos nuestros problemas.

En conclusión una solución aceptable para nuestro problema es montar una aplicación de mapas web la cual tenga un diseño adaptable tanto a PC como a dispositivos móviles. Para el desarrollo de la misma se va a montar un servidor XAMPP el cual trae todo lo necesario para implementar una aplicación WEB en PHP con la base de datos MySQL y para poder crear los mapas dentro de nuestra aplicaicion utilizaremos la librería OpenLayers. Como framework de trabajo en PHP utilizaremos CodeIgniter y para el diseño adaptable de la aplicación utilizaremos el framework CSS+JavaScript Bootstrap.

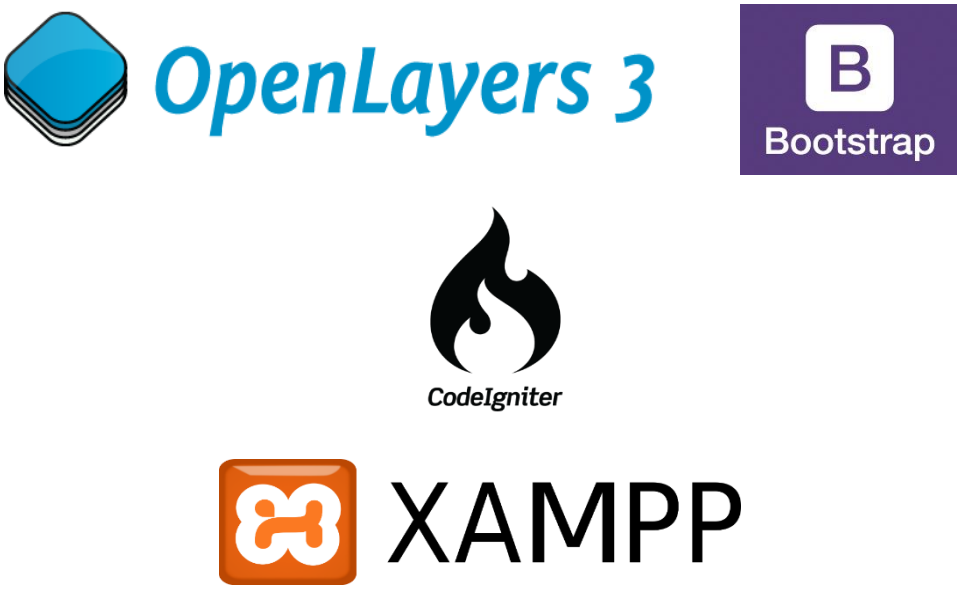



Ilustración 3. Conjunto de herramientas de la aplicación

# 3. Diseño y planificación

## 3.1. Requerimientos funcionales / no funcionales.

En este apartado vamos a ver los requerimientos tantos funcionales como no funcionales de la aplicación. Los requerimientos son aquellos que definen las funciones del sistema.

### 3.1.1. Requerimientos Funcionales

Vamos a describir pero a alto nivel los requerimientos funcionales que debería tener la aplicación web

- El usuario normal puede mostrar/ocultar los mapas de visibilidad.
- Un usuario normal puede ver los mapas de visibilidad según un periodo histórico.
- Un usuario normal puede elegir la configuración de la aplicación.
- Herramientas CRUD (Create Read Update Delete) POI’s.
- Herramientas CRUD para Culturas.
- Herramientas CRUD para Tipo de POI’s.
- El administrador puede cambiar la contraseña.

### 3.1.2. Requerimientos no Funcionales

Los requerimientos no funcionales son aquellos que definen los aspectos que debe tener nuestra aplicación pero sin llegar a ser funciones del sistema.

- En la medida de lo posible se evitaran recargar la página para actualizar los datos
- La aplicación tendrá un diseño responsive.
- La parte de administración estará protegida por contraseña

## 3.2. Planificación y costes

En el diagrama de Gantt que se adjunta más adelante, podemos ver las tareas las cuales van a componer el TFG. Observamos que se divide en cuatro grandes grupos:

- **Estudio bibliográfico** en cual se llevara a cabo un estudio de los distintos SIG del mercado, y las herramientas necesarias para obtener los mapas de visibilidad, tales como los MDE, coordenadas dentro del mapa, cambio de sistemas de proyección e interpretación de los datos obtenidos
- **Estudio de aplicaciones de mapas**: en este apartado se llevara a cabo un estudio de las distintas herramientas para integrar los mapas de visibilidad obtenidos por los SIG en aplicaciones WEB de mapas. Esto conlleva la transformación de imágenes TIFF georreferenciadas a imágenes PNG georreferenciadas, ya que estas últimas son más livianas y pueden soportar transparencia. También llevara a cabo como se puede utilizar la georreferenciación de la imágenes PNG en las aplicaciones de mapas
- **Diseño de aplicación**: en este apartado se diseñara el esquema Entidad Relación de la BBDD necesario para guardar todos los datos referentes a los POI's y el diseño mediante StoryBoard de las distintas vistas de la aplicación. La duración de este apartado está justificada debido a que la aplicación no necita una gran BBDD ni tampoco demasiadas vistas para obtener sus objetivos.
- **Implementación y corrección de fallos**: en el último apartado se implementaran los Modelos, Controladores y Vistas necesarias según el diseño. Habrá un modelo por cada entidad de la BBDD, un Controlador para ese modelo el cual será el intermediario entre las vistas. También hay un tiempo asignado para buscar y corregir los errores que surjan.

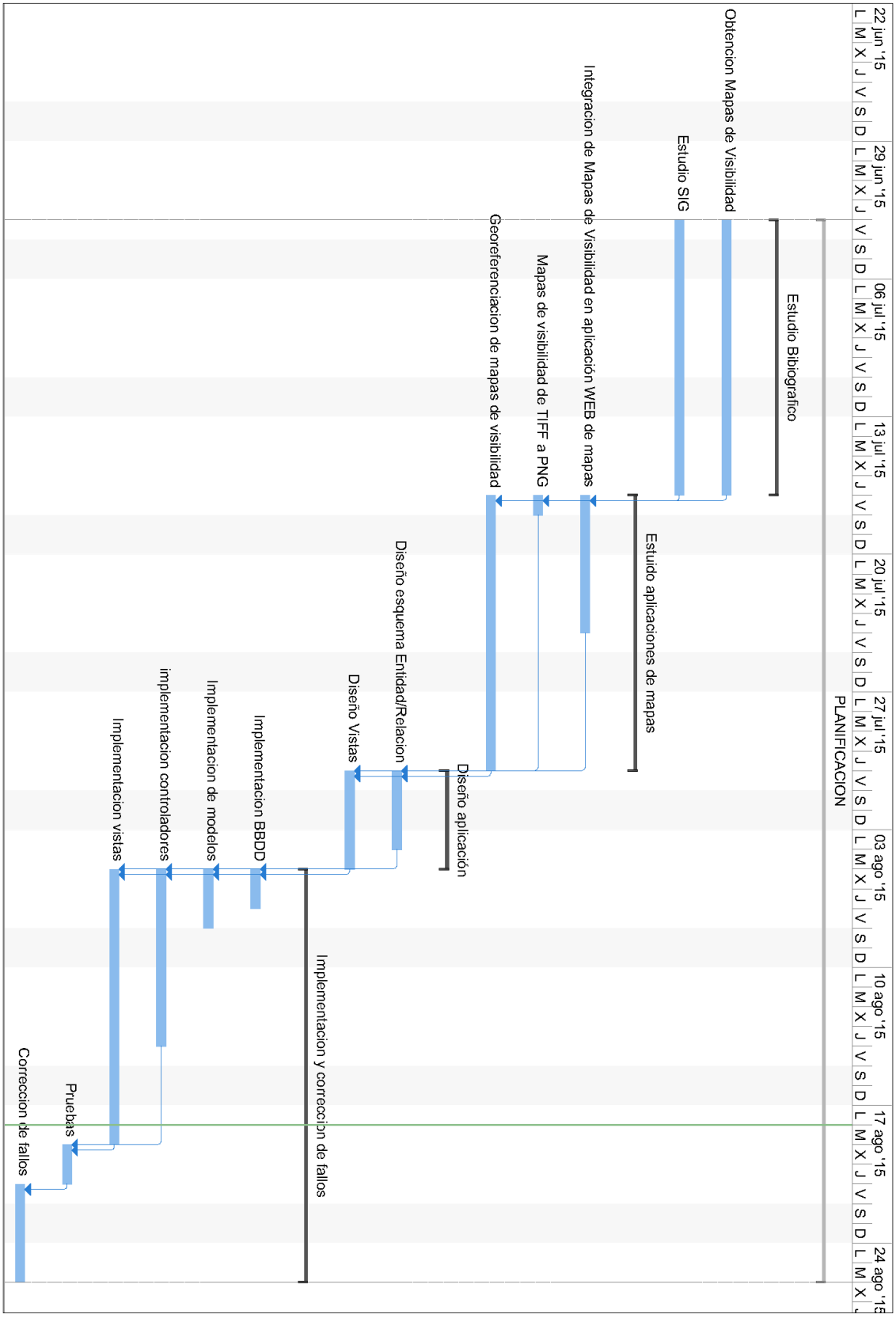



Ilustración 4. Planificación

Como podemos ver la duración del TFG es de 38 días, vamos a desglosar los gastos que suponen este proyecto:

De recursos materiales necesitamos un ordenador el que se ha utilizado para desarrollar este proyecto es un ASUS X5000 y su precio es de 2000€ amortizable en 5 años según las normas contables de la empresa.

El software necesario es

| Descripción | Unidades | Precio/Ud. |
|---|---|---|
| Notepad ++ | **1** | **0€** |
| XAMPP para Windows | **1** | **0€** |
| Codeigniter 2.2.0 | **1** | **0€** |
| OpenLayers 3.6 | **1** | **0€** |
| GvSIG | **1** | **0€** |
| GlobalMapper | **1** | **451€** |
| Dominio .es | **1** | **8 €** |
| Hosting con Apache, PHP MySQL | **1** | **30 €** |

**Tabla 1. Software necesario y precio**

En la parte de recursos humanos necesitamos a un analista de sistemas y un Analista-Programador que se hará cargo del resto de las funciones, incluyendo el diseño web. Según el convenio colectivo en vigor para el sector de las TIC [1]este tipo de categorías al año es de 23.505,74€ y 22.993,80€ respectivamente, en días esto se traduce en:

$$\frac{23.505.74\ €/año}{365\ dias/año} = 64{,}40\ €/dias \qquad \frac{22.993{,}80\ €/año}{365\ dias/año} = 63.00€/dias$$

Con todo esto el coste total del proyecto es de:

| Descripción | Unidades | Precio/Ud. | Total imputable |
|---|---|---|---|
| Ordenador ASUS x5000 | 1 | 2000 € | 66.67 € |
| Licencia GlobalMapper | 1 | 451 € | 451 € |
| Analista de Sistemas | 12 | 64.40€ | 772.80€ |
| programador/a Diseñador/a de página Web | 38 | 63.00€ | 2.394.00€ |
| Dominio .es | 1 | 8 € | 8 € |
| Hosting con Apache, PHP, MySQL | 1 | 30 € | 30 € |
| **Total** | | | 3.722,47€ |

**Tabla 2. Presupuesto.**

[1] Resolución del 18 de marzo de 2009(BOE del 4 abril de 2009) de la Dirección General de Trabajo por el que se registra y se publica XVI convenio colectivo estatal de empresas de consultoría y estudio de mercado y de opinión pública.

El coste total de la aplicación para su desarrollo, implantación y funcionamiento durante el primer año es de 3.722,47€.

### 3.2.1. Ciclo de vida

La producción de cualquier producto software se puede dividir en cuatro ciclos:

1. **Planificación**: En ella se recogen los datos necesarios para identificar los requisitos que conlleva el proyecto.
2. **Análisis**: Identificación de los problemas que puede conllevar el proyecto y si se considera oportuno, abandonarlo antes de comenzar la fase de desarrollo
3. **Desarrollo**: Una vez que se ha decidido el aceptar el proyecto, se empieza con las tareas de diseño, desarrollo y pruebas
4. **Evaluación**: Reunión con el cliente y el equipo para evaluar el rumbo del proyecto.

Aunque actualmente para desarrollos de aplicaciones web se suelen seguir las metodologías agiles tales como SCRUM, Programación Extrema, etc… para este caso se ha seguido una metodología más “tradicional” que es la del ciclo de vida en espiral. Se ha escogido esta metodología de trabajo frente a las agiles debido a que el tamaño de la aplicación es pequeño y el ciclo de vida en espiral te da los mismos resultados que las tecnologías agiles.

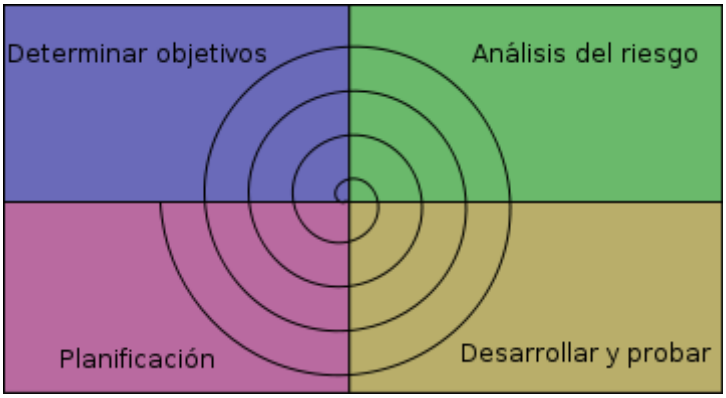



Ilustración 5. Ciclo de vida en espiral

## 3.3. Diseño de aplicación.

Primero vamos a diseñar los casos de uso a partir de los requerimientos funcionales, y a continuación vamos ver los diagramas de secuencia del programa

### 3.3.1. Casos de uso

Los casos de uso de este TFG son dos, el del usuario normal y el del administrador. Un esquema de casos de uso es en el que se especifican que requerimientos funcionales va a tener la aplicación y quien va tener acceso a ese requerimiento.

#### 3.3.1.1. Caso de uso Usuario

Primero vamos a ver los casos de uso para el usuario normal, el que solo puede acceder a ver los mapas de visibilidad.

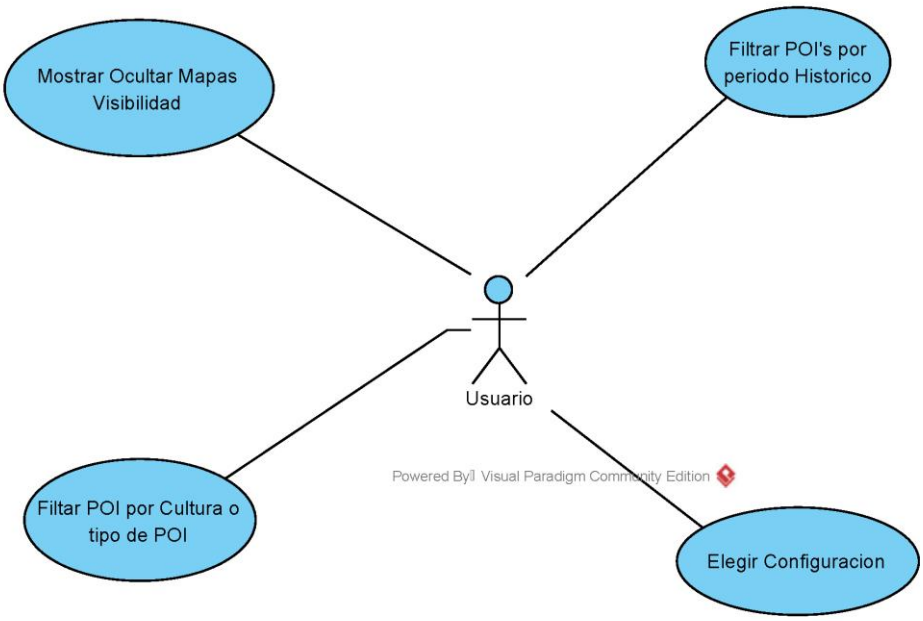



Ilustración 6. Caso de uso usuario

Un usuario normal podrá elegir la configuración que quiera así poder ver los mapas de visibilidad asignados a esa configuración

El usuario podrá filtra los POI por periodo histórico y también por cultura o el tipo de POI y así poder elegir que mapas ve y cuales no ve

También está la posibilidad de que el usuario quiera mostrar/ocultar algún mapa de visibilidad en concreto.

#### 3.3.1.2. Caso de uso Administrador

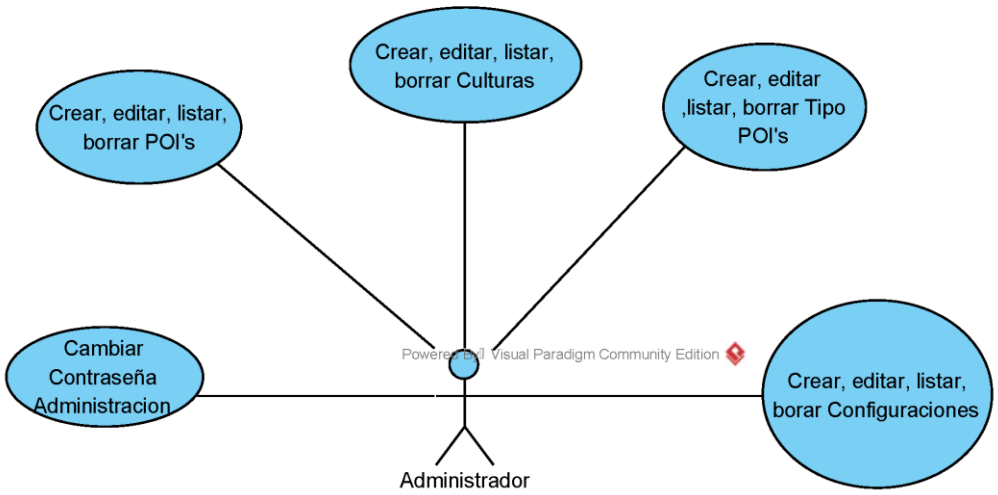



Ilustración 7. Caso de Uso Administración

El administrador del sistema tendrá permisos para poder crear, editar, listar, borrar tanto los POI's así como sus tipos, las distintas culturas y las configuraciones. A estas funciones se le denomina CRUD, lo cual quiere decir que el administrador tiene permisos para Crear Editar Listar Borrar estos elementos de la BBDD

El administrador también podrá cambiar su nombre y su contraseña de la BBDD

### 3.3.2. Diagramas de secuencia

Los diagramas de secuencia muestran como los pasos a seguir según qué acciones. En este caso como vamos a desarrollar una aplicación WEB la cual cuenta con el patrón de diseño MVC (Modelo-Vista-Controlador), dividiremos los diagramas de secuencia a dos, el del usuario normal y el del administrador, los demás elementos del diagrama serán el Modelo, la vista y el controlador, dejando claro quien llama a quien. En este caso se ha obviado la BBDD ya que dentro del patrón de diseño se especifica que el modelo es quien llama a la BBDD y obtiene los datos.

#### 3.3.2.1. Diagrama de secuencia pantalla Principal

Primero vamos a ver el diagrama de secuencia de la pantalla principal. Como podemos ver el actor principal es un usuario normal. En este diagrama de secuencia se puede ver la interacción que tiene el usuario normal con el sistema.

En este diagrama podemos ver como el usuario pide la página principal al controlador de la misma y este devuelve la vista correspondiente. También podemos observar como el usuario elige la configuración que le ofrece la página principal y esta le envía al controlador el id y este a su vez consulta al modelo los País con dicha configuración, cuando el controlador obtiene los datos devuelve la información de los País así como los mapas de visibilidad y es la vista quien se los muestra al usuario.

También se puede observar el proceso de filtrar ya sea por un periodo Histórico, culturas o tipos de País. El usuario selecciona el filtro en la vista, esta pasa todos los datos necesarios para el filtro. El controlador le pide al modelo los País filtrados por esos datos y a continuación le devuelve a la vista los datos ya filtrados. Finalmente la vista se actualiza con los datos recibidos del controlador y se actualiza.

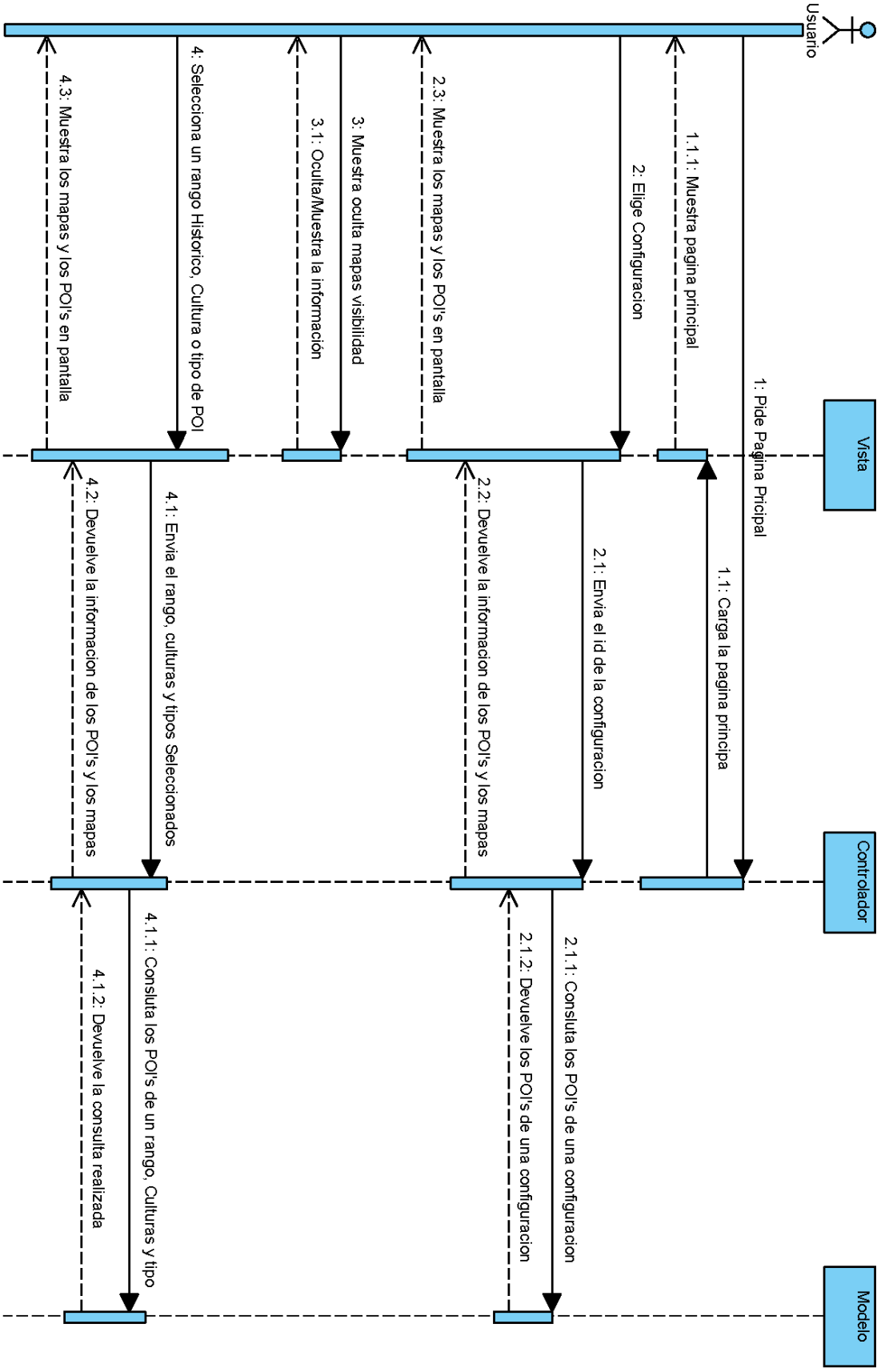



**Ilustración 8. Diagrama de secuencia del usuario**

#### 3.3.2.2. Diagrama de secuencia Administración

En este diagrama podemos ver la iteración de las operaciones CRUD de los distintos elementos del sistema. Como se ve ya no es un usuario cualquier quien realiza estas acciones sino un usuario administrador. Como antes el administrador pide la página de configuración al controlador y es el controlador quien carga la vista necesaria para dicha configuración y la vista quien muestra la información.

Para insertar un elemento la vista envía al controlador todos los datos necesarios para crear un elemento nuevo, este llama al modelo para que cree en la BBDD un elemento nuevo con esos datos. A continuación consulta los elementos y le pasa a la vista todos los elementos incluido el creado en formato JSON para que la vista pueda actualizar los datos.

Para editar un elemento la vista envía al controlador los datos necesarios para editar un elemento. El controlador llama al modelo para que edite los elementos en la BBDD. Mientras la vista actualiza los datos en pantalla.

Finalmente para borrar la vista envía el id del elemento al controlador y es el controlador quien llama al modelo para eliminar el elemento con ese id. Mientas la vista actualizara los datos en pantalla.

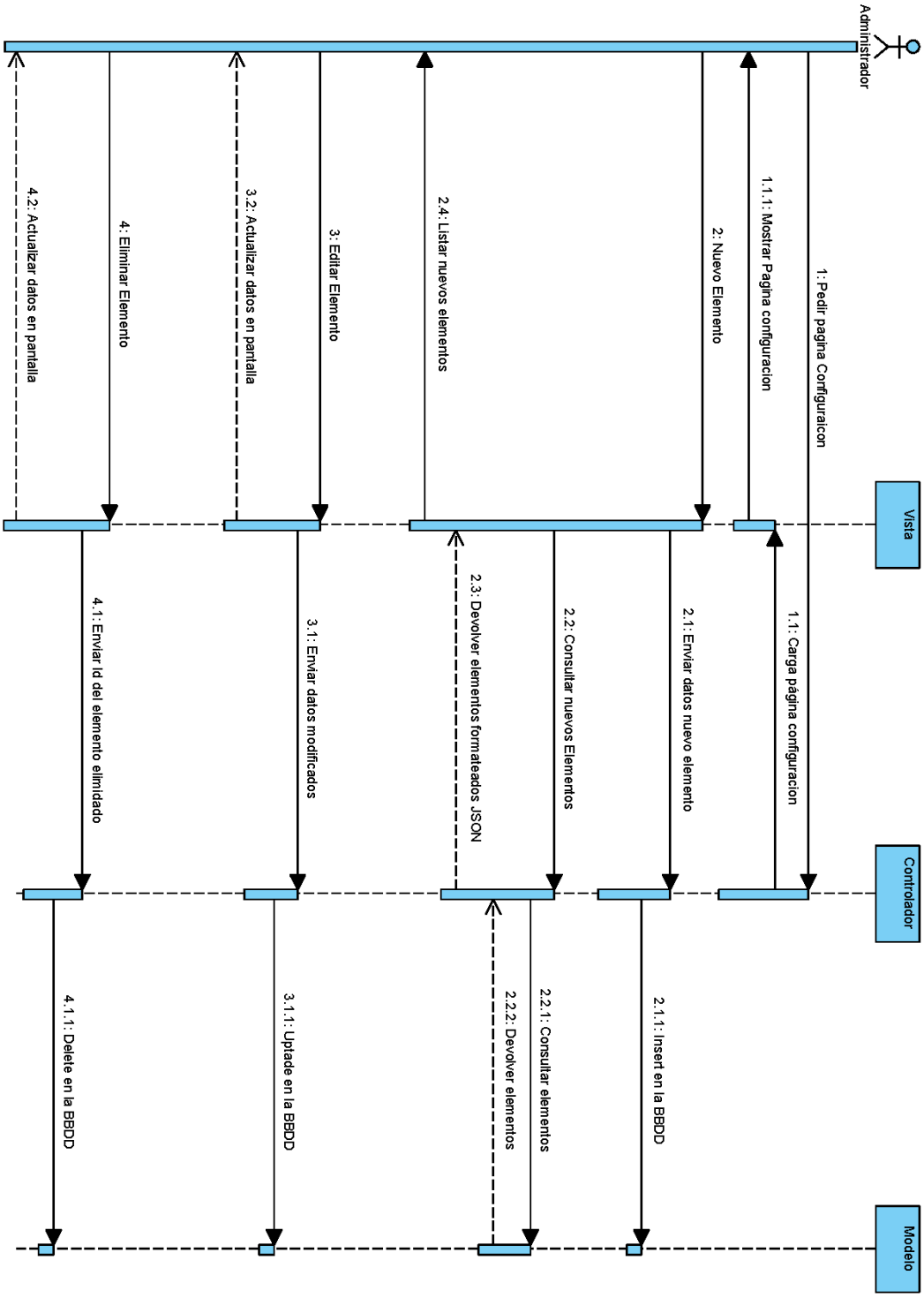



**Ilustración 9. Diagrama de secuencia del administrador**

### 3.3.3. Esquema entidad relación

Después de ver los requisitos, casos de uso y diagramas de secuencia vamos a diseñar el esquema de entidad relación de la aplicación

Como podemos observar en el esquema tenemos 5 tablas las cuales se describeb a continuación:

- **Configuración**: representa una configuración de la aplicación, uno de los requerimientos es poder filtrar los país con rango de edades o periodos históricos los cuales están ligados a las configuraciones
- **Bando/ Cultura**[2]: representa la cultura que pertenecía un POI. Para poder identificar mejor los mapas según las distintas culturas se guarda el color el cual se le asignara a los mapas de visibilidad.
- **Tipo_poi**: representa el tipo de fortificación (castillo, torres, fortalezas, torreones, puestos de vigilancia, etc…) desde los cuales se ha calculado la cuenca visual
- **POI**: representa un punto en el mapa, puede ser el punto desde donde se ha calculado el mapa o cuenca visual o uno más representativo, dado que hay varios tipos de cálculo. Un POI puede tener una sola configuración, un solo bando, y es solo de un tipo. También tiene un periodo en cual estaba activo
- **Usuarios**: Representa los usuarios administrador, los cuales tendrá un aserenarme y una contraseña.

[2] Inicialmente se le llamo bandos pero a mitad del desarrollo de la aplicación se decidió cambiar el nombre a culturas, como el tiempo es ajustado y cambiar un nombre de una tabla conlleva un enorme riesgo se decidió dejar la parte de programación como bando pero la parte visual como Culturas, en los diseños se ha incluido como cultura.

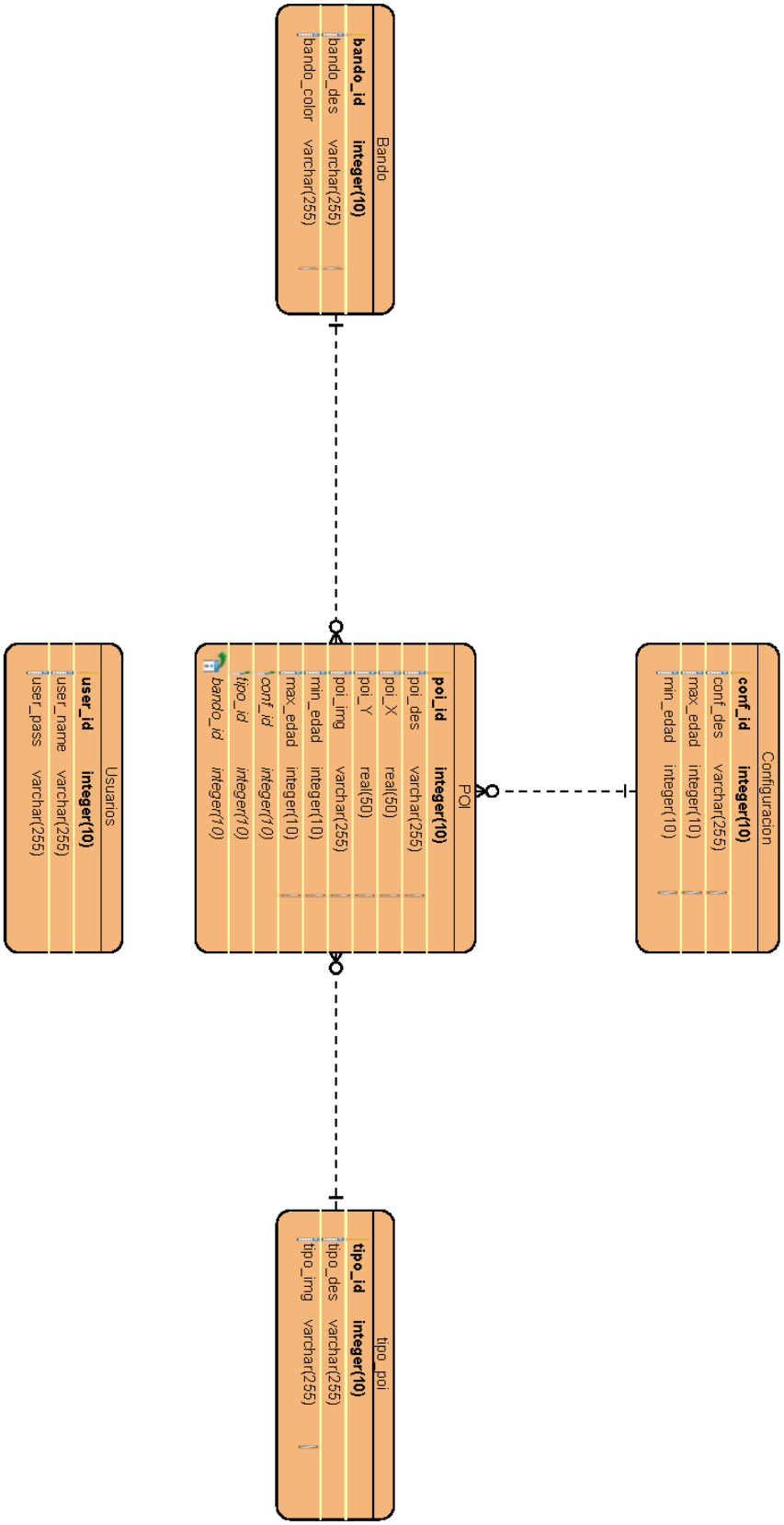



**Ilustración 10 Esquema entidad Relación**

### 3.3.4. StoryBoards.

Los storyBoards dentro del diseño web son dibujos de pantallas los cuales muestran de forma muy sencilla el diseño de la página, los botones, y la posición de los mismos. En cierto modo son una guía para el programador web para que pueda ver lo que tiene que hacer de manera rápida y sencilla.

Los storyBoard están dibujados en Paint ya que el objetivo principal de este componente es diseñar y colocar de manera visual los distintos elementos de la aplicación.

#### 3.3.4.1. Pantalla Principal

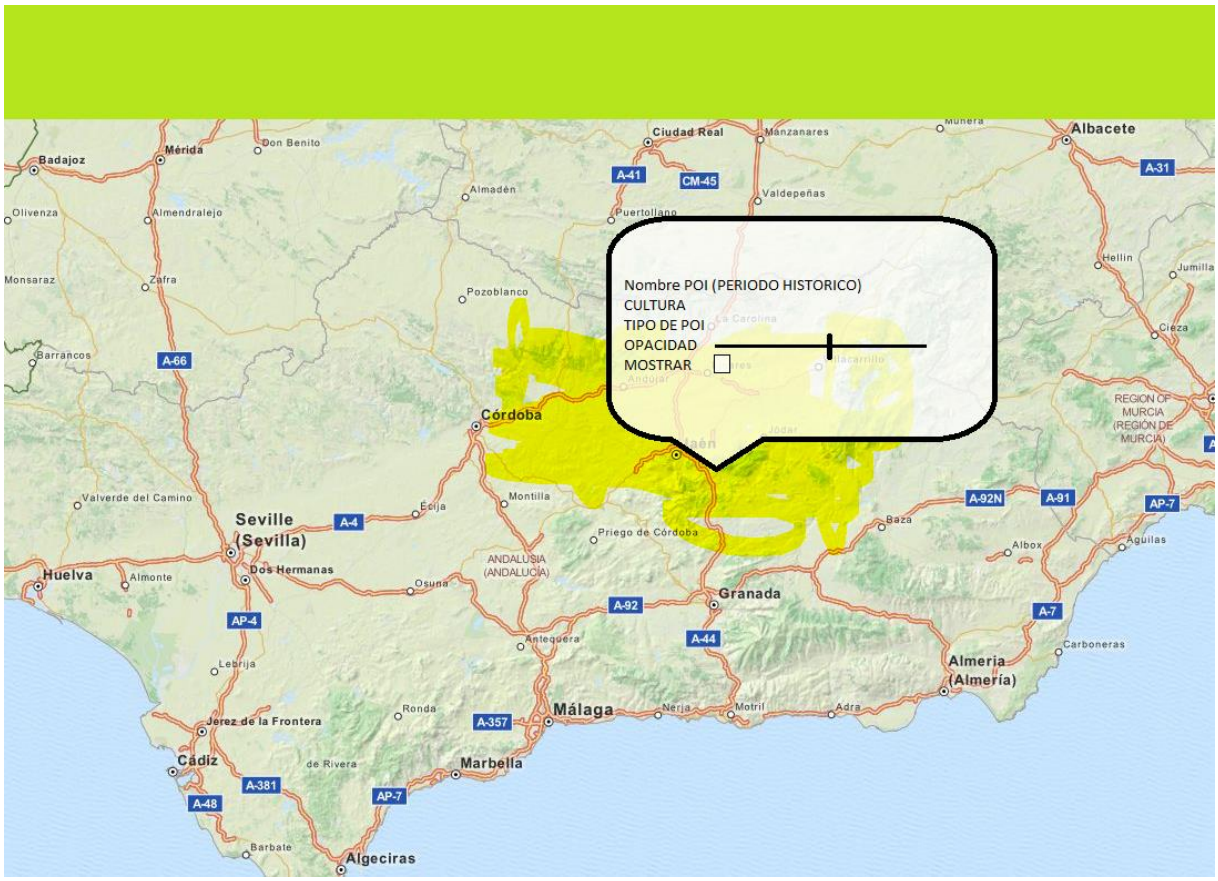



Ilustración 11. StoryBoard pantalla principal

Como se ve en la imagen la pantalla principal es muy básica simplemente muestra el mapa y los mapas de visibilidad. Al hacer click sobre ellos o sobre un markert puesto de la posición del POI se mostrara un popup con la información y los controles necesarios para ocultar/mostrar la capa así como su opacidad.

#### 3.3.4.2. Pantalla de administración.

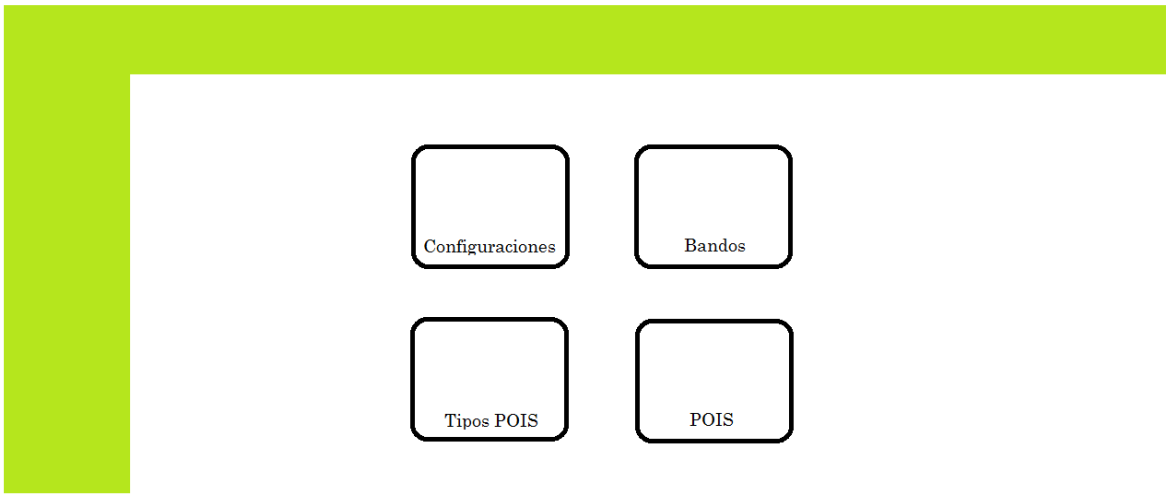



Ilustración 12. StoryBoard Pantalla administración

Esta pantalla es la principal de la administración, simplemente se mostrara como un menú en el cual podremos ir a los distintos formularios de administración

#### 3.3.4.3. Pantalla administración POI's

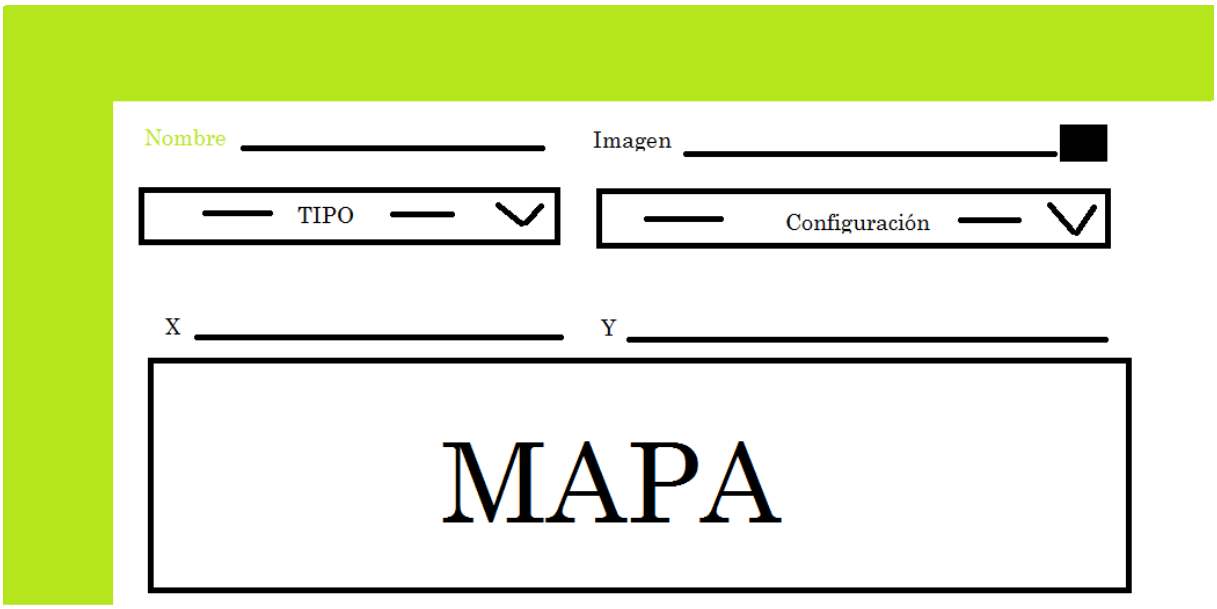



Ilustración 13. StoryBoard administración POI's

Para editar los POI's se contara con un Mapa en el cual se mostraran y al hacer click sobre ellos se rellenara el formulario para poder editarlos. Este formulario también sirve para poder insertar un POI para insertar las coordenadas X e Y se habrá que hacer click sobre una zona libre del mapa.

#### 3.3.4.4. Pantalla administración Culturas

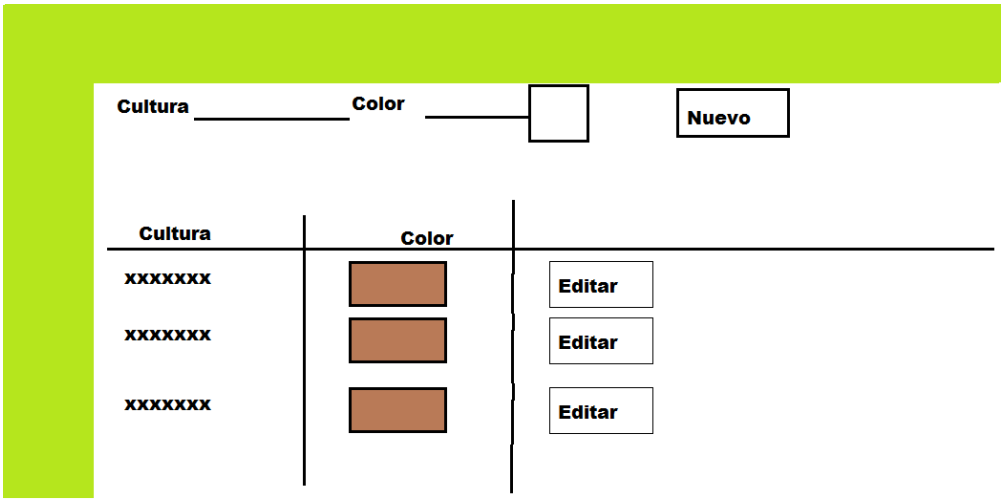



Ilustración 14. StoryBoard Culturas

En esta pantalla se podrá ver las distintas culturas así como sus colores asignados en una tabla. También contará con un formulario en la parte superior para insertar uno nuevo. Para editar una cultura habrá un botón al final de la tabla el cual cambiara la fila y la dejara con campos de formulario y también se añadirá un botón de borrar.

#### 3.3.4.5. Pantalla de administración tipos de POI's

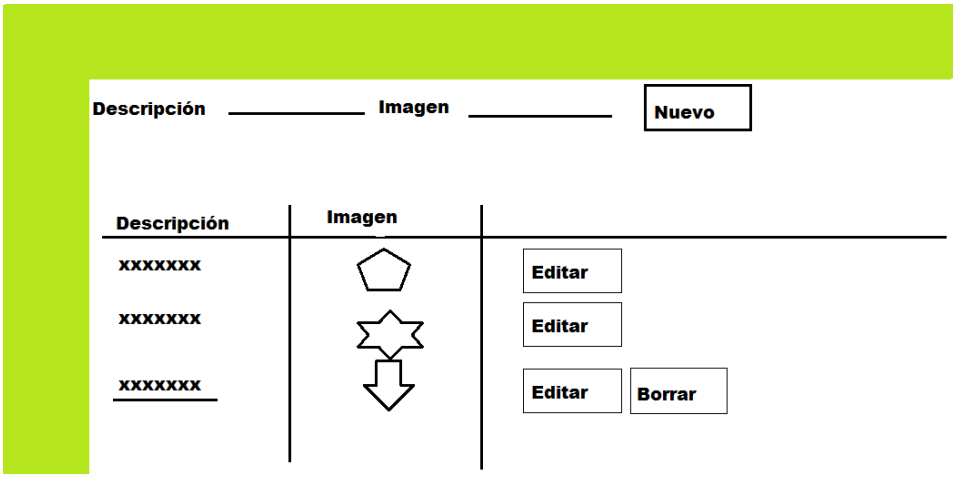



Ilustración 15. StoryBoard Tipos POI's

Esta pantalla cuenta con las mismas funcionalidades que la pantalla de Culturas.

#### 3.3.4.6. Pantalla de administración de configuración.


Ilustración 16. StoryBoard Configuración.

En esta pantalla se incluyen también las mismas funcionalidades que en las dos anteriores.

# 4. Implementación

Este capítulo se va a dividir en 3 subcapítulos los cuales trataran de los distintos aspectos de la implementación, el servidor utilizado, el framework PHP y la biblioteca JavaScript OpenLayers

## 4.1. XAMPP


Ilustración 17. Logo XAMPP

Como servidor se ha utilizado XAMPP en su versión 3.2.1 XAMPP es un servidor independiente de plataforma, software libre, que consiste principalmente en el sistema de gestión de bases de datos MySQL, el servidor web Apache y los intérpretes para lenguajes de script: PHP y Perl. Los paquetes incluidos en esta versión son:

- Apache 2.4.16.
- MySQL 5.6.25.
- PHP 5.6.11.
- phpMyAdmin 4.4.12.
- OpenSSL 1.0.1.
- XAMPP Control Panel 3.2.1.
- Webalizer 2.23-04.
- Mercury Mail Transport System 4.63.
- FileZilla FTP Server 0.9.41.
- Tomcat 7.0.56 (with mod_proxy_ajp as connector).
- Strawberry Perl 7.0.56 Portable

Como podemos ver en esta distribución se incluye phpmyadmin, que se trata de una herramienta web para manejar de manera más sencilla las bases de datos dentro de MySQL.

Para crear una base de datos nueva simplemente haremos click sobre icono de nueva dentro del menú de la izquierda:

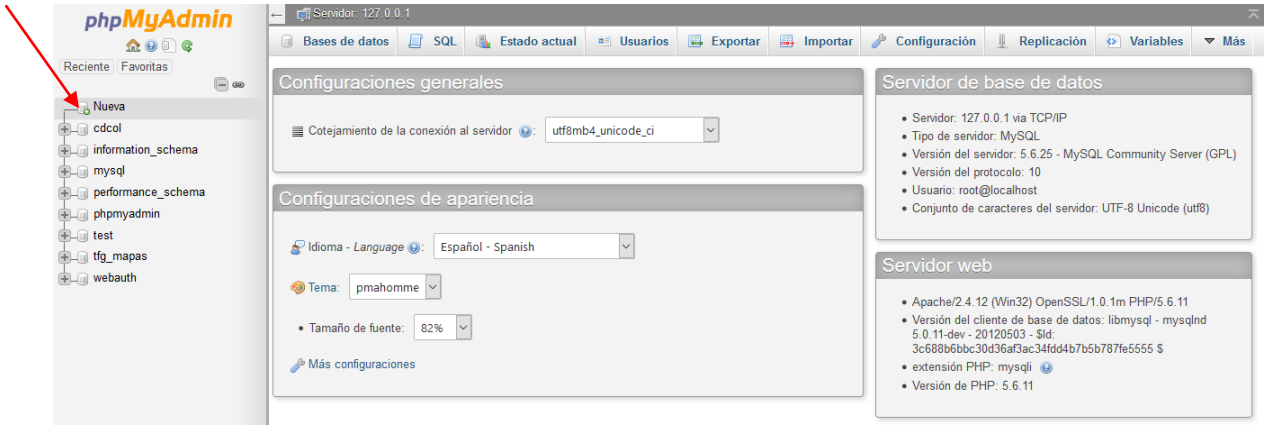



Ilustración 18. Pantalla principal phpMyAdmin

## 4.2. CodeIgniter


Ilustración 19. Logo CodeIgniter

CodeIgniter es un framework para desarrollo de aplicaciones web usando PHP. Permite desarrollar proyectos mucho más rápido que lo que podría hacer si escribiera el código desde cero, proveyéndonos de un rico conjunto de bibliotecas para las tareas más comunes, así como y una interfaz sencilla y una estructura lógica para acceder a esas bibliotecas. CodeIgniter permite enfocarnos creativamente en nuestro proyecto al minimizar la cantidad de código necesaria para una tarea dada.

Los recursos mínimos que necita codeigniter para poder ser ejecutado son los siguientes:

- PHP versión 5.1.6 o más reciente.

- Las bases de datos que se soportan actualmente son MySQL (4.1+), MySQLi, MS SQL, Postgres, Oracle, SQLite, y ODBC.

El siguiente gráfico ilustra como los datos fluyen a través del sistema de codeigniter.

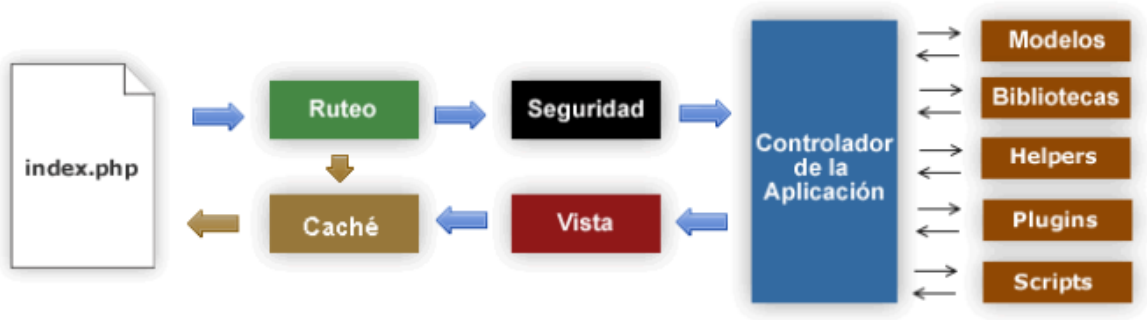



Ilustración 20. Flujo de trabajo CodeIgniter

1. El index.php sirve como el controlador frontal, inicializando los recursos básicos que necesita CodeIgniter para ejecutar.
2. El Ruteador examina la solicitud HTTP para determinar que debería hacer con ella.
3. Si existe el archivo de caché, se lo envía directamente al navegador, sin pasar por la ejecución normal del sistema.
4. Seguridad. Antes que se cargue el controlador de la aplicación, por razones de seguridad se filtran la solicitud HTTP y cualquier otro dato enviado por el usuarios.
5. El controlador carga el modelo, las bibliotecas del núcleo, helpers, y cualquier otro recurso requerido para procesar una solicitud específica.
6. La Vista terminada se procesa y se envía al navegador para que se pueda ver. Si el caché está habilitado, la vista se cachea primero para que las siguientes solicitudes que la necesiten puedan ser servidas.

CodeIgniter está basado en el patrón de desarrollo Modelo-Vista-Controlador. MVC es un enfoque de software que separa la lógica de la aplicación de la presentación.

- El Modelo representa estructuras de datos. Tipicamente las clases del modelo contendrán funciones que ayudaran a devolver, insertar y actualizar información de la base de datos.
- La Vista es la información que se presenta al usuario. Una vista será normalmente una página web, pero en CodeIgniter, una vista también puede ser un fragmento de página como el encabezado o pie de página.
- El Controlador sirve como un intermediario entre el Modelo, la Vista y cualquier otro recurso necesario para procesar la solicitud HTTP y generar una página web.

Otro aspecto importante de codeingniter son las URL, por defecto, las URLs en CodeIgniter están diseñadas para ser amigables con los motores de búsqueda y las personas. En lugar de usar el enfoque estándar de las "query string" para las URLs que es sinónimo de sistemas dinámicos, CodeIgniter usa el enfoque basado en segmentos. Siguiendo el enfoque Modelo-Vis*ta-Controlador, los segmentos en la URL normalmente representan:

```
ejemplo.com/index.php/clase/función/ID
```

1. El primer segmento representa la **clase** del controlador que se debería invocar.
2. El segundo segmento representa la **función** de la **clase**, o método que se debería llamar.
3. El tercer y cualquier otro segmentos adicionales, representa el **ID** y cualquier variable que se pasará al controlador.

La Clase URI y el Helper de URL contienen funciones que hacen fácil trabajar con datos de URI. Además para mayor flexibilidad, sus URLs se pueden remapear usando la funcionalidad de Ruteo de URI.

Por defecto, el archivo index.php esta incluido en las URLs se puede quitar fácilmente este archivo usando un archivo .htaccess con algunas reglas simples. Aquí hay un ejemplo de tal archivo, usando el método "negativo" donde todo se redirecciona excepto los ítems especificados:

```
RewriteEngine on
RewriteCond $1 !^(index\.php|images|robots\.txt)
RewriteRule ^(.*)$ /index.php/$1 [L]
```

Una vez visto los aspectos más básicos del framework CodeIgniter vamos a ver por partes los Modelos los controladores y las vistas implementadas para este TFG.

### 4.2.1. Modelos.

CodeIgniter tiene un archivo de configuración que le permite almacenar los valores de conexión de la base de datos (usuario, contraseña, nombre de la base de datos, etc.). El archivo de configuración está ubicado en application/config/database.php.

```
$db['default']['hostname'] = 'localhost'; //Aquí el nombre del host
$db['default']['username'] = 'root'; //Aquí el nombre de usuario
$db['default']['password'] = ''; //Aquí la contraseña
$db['default']['database'] = 'tfg_mapas'; //Aquí el nombre de la base de datos
$db['default']['dbdriver'] = 'mysql'; //Aquí el controlador de la base de datos.
$db['default']['dbprefix'] = '';
$db['default']['pconnect'] = TRUE;
$db['default']['db_debug'] = TRUE;
$db['default']['cache_on'] = FALSE;
$db['default']['cachedir'] = '';
$db['default']['char_set'] = 'utf8';
$db['default']['dbcollat'] = 'utf8_general_ci';
$db['default']['swap_pre'] = '';
$db['default']['autoinit'] = TRUE;
$db['default']['stricton'] = FALSE;
```

Los parámetros de configuración se almacenan en un array multidimensional para el uso local quedo de la siguiente forma.

Los modelos son clases de PHP que se diseñan para trabajar con información de la base de datos. CodeIgniter viene con una clase de base de datos abstracta muy rápida y completa que soporta tanto las estructuras tradicionales como los patrones Active Record. Las funciones de base de datos ofrecen una sintaxis clara y sencilla.

El prototipo básico para la clase de un modelo es este:

```
class Nombre_modelo extends CI_Model {
        function __construct()  {
                parent::__construct();
        }
}
```

Donde Nombre_modelo es el nombre de su clase, esta clase tiene que heredar de la clase base CI_Model.

Los modelos desarrollados para este trabajo han sido los siguientes:

- configuracion_model
- tipoPOI_model
- usuario
- POI_model
- bando_model

Cada uno de estos modelos representa una tabla de la base de datos, por lo tanto tendrán los métodos necesarios para listar todos los elementos de la tabla, insertar nuevos elementos, editar un elemento ya existente y eliminar un elemento. Como hemos dicho antes codeigniter hace uso de una clase para trabajar con la base de datos vamos a ver algunos ejemplos de la implementación.

#### 4.2.1.1. Listar todos los elementos de una tabla:

```
$query = $this->db->get('configuracion');
return $query->result();
```

Como tenemos cargado automáticamente la clase database no hace falta la línea para cagarla. Luego la sentencia `$this->db->get('configuracion');` nos da un SELECT * FROM configuración, el resultado de la consulta se optiene con `$query->result();` esto devuelve un array multidimensional con los elementos de la consulta. Para poder acceder a ellos simplemente hace falta acceder al nº de fila y luego al nombre de la columna en la base de datos.

#### 4.2.1.2. Obtener un elemento por medio de su ID

```
$this->db->select('*');
$this->db->from('configuracion');
$this->db->where('conf_id', $id);
$this->db->limit(1);
$query = $this->db-> get();

if($query -> num_rows() == 1){
        $rows = $query->result();
        $this->id = $rows[0]->conf_id;
        $this->description = $rows[0]->conf_des;
        $this->max_edad = $rows[0]->max_edad;
        $this->min_edad = $rows[0]->min_edad;
        return $this;
}
return FALSE;
```

En este caso hace uso de los métodos para construir una consulta, con select le pasamos la cadena que ve dentro de la sentencia select, con from elegimos la taba, con where añadimos sentencias WHERE a la consulta el primer parámetro es una cadena con el nombre de la columna y el segundo el valor al que debe ser igual. Después comprobamos que solo ha salido un resultado y guardamos en las propiedades del modelo los datos recibidos.

#### 4.2.1.3. Insertar un nuevo elemento:

```
$this->db->insert('configuracion', $data);
```

Con esta sentencia se crea el INSERT INTO correspondiente, hay que tener en cuenta que el array de datos que se le pase sus indicies deben coincidir con los nombres de las columnas en la base de datos.

#### 4.2.1.4. Editar un elemento.

```
$this->db->where('conf_id', $data['conf_id']);
return $this->db->update('configuracion', $data);
```

Con el método where añadimos a la sentencia que se vaya a ejecutar la cláusula SQL WHERE el primer parámetro es una cadena con el nombre de la columna y el segundo el valor al que debe ser igual. Luego se llama al método update el cual genera una sentencia SQL UPDATE INTO, aquí como con el método insert el array que se le pasa los indicies deben coincidir con las columnas en la base de datos.

#### 4.2.1.5. Eliminar un elemento de la base de datos

```
$this->db->delete('configuracion',array('conf_id'=>$id));
```

En este caso se ha utilizado el método **delete** el cual genera una sentencia SQL DELETE FROM, el primer parámetro es el nombre de la tabla y el segundo es un array para generar la cláusula where de la sentencia, el índice del array es el campo de la tabla y el contendido el valor al que tiene que ser igual.

#### 4.2.1.6. Obtener el ID insertado

```
$this->db->select_max('tipo_id');
$query = $this->db->get('tipo_poi');
if($query -> num_rows() == 1){
        $rows = $query->result();
        if (is_null($rows[0]->tipo_id)){
                return 1;
        }
        return $rows[0]->tipo_id;
}
return -1;
```

En esta ocasión utilizamos el selector SELECT_MAX el cual nos selecciona el máximo de una columna. Esta función hay que llamarla justo después de haber insertado un registro en la base de datos para que nos dé el máximo o en este caso el id que se acaba de insertar en la tabla.

#### 4.2.1.7. Obtener POI's filtrados.

```
$this->db->select('*');
$this->db->from('poi');
$this->db->where('conf_id', $data['id_conf']);
if (isset($data['rango'])){
        $array = array('poi_ini >=' => $data['rango'][0], 'poi_fin <=' => $data['rango'][1]);
        $this->db->where($array);
}
$this->db->where_in('bando_id', $data['bandos']);
$this->db->where_in('tipo_id', $data['tipoPOI']);
$query = $this->db-> get();
return $query->result();
```

En este caso hay que hacer una sentencia un poco mas complicada, pero gracias a Active Record de codeingniter es fácil de estructurar. Primero con el método select le pasamos lo que queremos seleccionar que en este caso es todo, luego con from le decimos la tabla, que es la de POI, y a continuación vamos añadiendo las sentencias WHERE, primero la configuración, luego si se nos ha pasado el rango o periodo histórico creamos la setencia WHERE con el rango, por cada vez que se llama al método where se añade un AND a la sentencia SQL final, a continuación le añadimos la sentecias WHERE IN de la cultura y del tipo de POI, la sentencia SQL diseñada es la siguente:

```
SELECT *
FROM `poi`
WHERE conf_id=id
AND `poi_ini`>=min_edad
AND `poi_fin`<=max_edad
and bando_id IN(array_id_bandos)
and tipo_id in (array_id_tipos)
```

### 4.2.2. Controladores

Los controladores son el corazón de nuestra aplicación, ya que determinan como se manejan las solicitudes HTTP.

Un Controlador es simplemente un archivo de clase que se nombra de una forma en la que se puede asociar con una URI, por ejemplo:

tfgmapas.jmoron.es/index.php/admin/login

CodeIgniter intentaría encontrar un controlador llamado admin.php y el segundo segmento de la URI determina qué función del controlador se llama. Como vemos en el ejemplo se intentará llamar al controlador admin.php y dentro de este se llamara a la función login (). Si hubiera más segmentos estos serían los parámetros que se le pasa a la función. Todos los controladores deben heredar de CI_Controller.

Para nuestra aplicación se han desarrollado los siguientes controladores:

- **inicio.php**, controlador simple solo carga las vista de la página principal
- **admin.php**, se encarga del login/logout del administrador y también de cargar el menú de administración
- **adminConfiguracion.php**, se encarga de cargar la página de administración de las configuraciones, asi como de llamar al modelo para insertar, editar y eliminar una configuración. También tiene funciones para devolver las configuración/es en formato json.
- **adminbandos.php**, se encarga de cargar la página de administración de las culturas, asi como de llamar al modelo para insertar, editar y eliminar una cultura. También tiene funciones para devolver las cultura/s en formato json.
- **adminPOI.php** se encarga de cargar la página de administración de las POI's, asi como de llamar al modelo para insertar, editar y eliminar un POI y de la carga de los mapas en el servidor. También tiene funciones para devolver los POI's en formato json.
- **adminTipoPOI.php** se encarga de cargar la página de administración de los tipos de POI's, asi como de llamar al modelo para insertar, editar

y eliminar un tipo de POI y de la carga del icono del tipo. También tiene funciones para devolver los tipos de POI en formato json.

A continuación vamos a ver las funciones más relevantes dentro los controladores o en algunos casos partes concretas.

#### 4.2.2.1. Codificación de datos en formato JSON

Estos métodos devuelven alguna propiedad formateada en formato JSON[3], es un formato ligero para el intercambio de datos. La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente en AJAX. El siguiente código muestra como a partir de un array obtenemos los datos en formato JSON.

```
public function get_configurations_json(){
        //CARGAMOS EL MODELO DE CONFIGURACION
        $this->load->model('configuracion_model');
        //LEEMOS LA BD DE LAS DISTINTAS CONFIGURACIONS
        $template_data['configuration']=$this->configuracion_model->get_configurations();
        //LISTAMOS LA CONFIGURACIONES
        echo json_encode($template_data['configuration']);
}
```

Y el resultado es el siguiente:

```
[
        {"conf_id":"1",
        "conf_des":"Jaen",
        "max_edad":"300",
        "min_edad":"-200"},
        {"conf_id":"4",
        "conf_des":"Granada",
        "max_edad":"1200",
        "min_edad":"-300"},
        {"conf_id":"6",
        "conf_des":"Cordoba",
        "max_edad":"200",
        "min_edad":"-100"}
]
```

Estas funciones al cumplir una función determinada se le ha añadido _json al final del nombre de la función para que se pueda ver a simple vista.

[3] JavaScript Object Notation

#### 4.2.2.2. Subida de archivos al servidor.

Para nuestra aplicación es necesario que el usuario sea capaz de subir archivos al servidor. Para ello vamos hacer uso de la librería de codeigniter Upload, la cual nos facilita bastante la tarea de subir cualquier archivo al servidor.

```
// Cargamos la libreria Upload
$this->load->library('upload');

// Configuración para el Archivo
$config['upload_path'] = 'assets/visibilityMaps/';
$config['allowed_types'] = 'gif|jpg|png';
$config['file_name'] = 'map_'.$newPoi.'.png';
// Cargamos la configuración del Archivo
$this->upload->initialize($config);
// Subimos archivo
if ($this->upload->do_upload('Imagen')){
        $data = $this->upload->data();
}else{
        echo $this->upload->display_errors();
}
```

El array $config contiene la configuración básica para la carga de archivos, en el índice upload_path, introducimos la ruta donde va a ir el archivo hay que tener en cuenta codeigniter toma referencia de sus carpetas a partir de donde está el archivo index.php, no donde se encuentra el controlador. El segundo índice es allowed_types el cual pondremos separados por “|” los tipos de archivos permitidos en la carga. El tercer índice es el nombre del nuevo archivo, este es opcional pero a fin de ordenar los mapas de visibilidad y los iconos de tipos de POI’s los archivos se han renombrado de la siguiente forma:

- Mapas de visibilidad: van a la ruta ‘assets/visibilityMaps/' y se nombra de la siguiente forma 'map_idPOI.png'.
- Archivos de georreferenciación: van a la ruta ‘assets/visibilityMaps/worldInfo/’ y son renombrados de la siguente forma, ‘wordInfo_ idPOI.pgw’.
- Iconos de los tipos de POI’s: van a la ruta ‘assets/img/IconsPOIs/’ y son renombrados de la siguiente forma ‘tipoPOI_idTipoPOI.png’.

#### 4.2.2.3. Cambio de color de los mapas de visibilidad.

El cambio de color de los mapas de visibilidad se hace en dos sitios en el controlador del POI cuando se inserta o se edita la cultura a la que pertenece el poi o cuando se cambia de color la cultura que en este caso se cambia todos los mapas asociados a dicha cultura.

```
//Creamos el objeto imagen, como son png uilizamos esta funcion
$im = imagecreatefrompng("./assets/visibilityMaps/map_".$item->poi_id.".png");
//Se ponen estas dos lineas para conservar las transparecinas de la imagen.
imagealphablending($im, true);
imagesavealpha($im, true);
//Normalmente el primer pixel es de color transparente
$colorBlanco=imagecolorat($im, 0,0);
$ancho=imagesx($im); //devuelve el ancho de la imagen
$alto=imagesy($im); //devuelve el alto de la imagen

$hex = str_replace("#", "", $hex);
$r = hexdec(substr($hex,0,2));
$g = hexdec(substr($hex,2,2));
$b = hexdec(substr($hex,4,2));
 $rgb = array($r, $g, $b);//pasamos de HEX a RGB
//Obtenemos el color del pixel para cambiar
$colorPixel = imagecolorallocate($im,$rgb[0],$rgb[1],$rgb[2]);
for ($i=0; $i<$ancho; $i++){
        for ($j=0; $j<$alto; $j++){
                if (imagecolorat($im, $i,$j)!=$colorBlanco){//Si es distinto el pixel cambiamos el color
                        imagesetpixel ( $im ,$i , $j , $colorPixel );
                }
        }
}
//header('Content-Type: image/png');
imagepng($im, "./assets/visibilityMaps/map_".$item->poi_id.".png");
```

Para poder cambiar de color los pixeles de una imagen utilizamos la librería de PHP GD 2.0 la cual se utiliza para el tratamiento de imágenes en PHP. Lo primero que hay que hacer es crear una variable con la imagen para ello utilizamos la función imagecreatefrompng , como en nuestro caso son png se utiliza esta. Para poder conservar la transparencia en la imagen a continuación añadimos las funciones imagealphablending($im, true); imagesavealpha($im, true). Luego recorremos la imagen y comparamos pixel por pixel, si el pixel es distinto que uno de muestra que hemos cogido antes le cambiamos el color. Finalmente volemos a guardar la imagen con el mismo nombre.

#### 4.2.2.4. Lectura de archivos World.

Los archivos World o en ingles World File son archivos de texto plano los cuales utilizan los GIS para georreferenciar los datos raster. Estos archivos tienen un formato como el siguiente:

- Línea 1: tamaño del pixel en el eje X en el mapa units/pixel
- Línea 2: rotación respecto al eje Y
- Línea 3: rotación respecto al eje X
- Línea 4: tamaño del pixel en el eje Y, este valor siempre es negativo
- Línea 5: coordenada x del pixel de arriba a la izquierda
- Línea 6: coordenada y del pixel de arriba a la izquierda

La proyección de estas coordenadas están en un archivo .prj, pero como en nuestro caso la proyección siempre es la misma (ED 50 uso 30) se ha obviado este archivo.

Para leer este archivo desde php y asi sacar la caja envolvente de la imagen se ha utilizado el siguiente código.

```
//Abrimos el archivo World Info del POI correspondiente
$fp = fopen("assets/visibilityMaps/worldInfo/wordInfo_".$poi->poi_id.".pgw", "r");
$lineas;
$j=0;
while(!feof($fp)) {
        $lineas[$j] = fgets($fp); // Guardamos en un array las lineas
        $j++;
}
fclose($fp);
$prueba[$i]['minX']=(float)$lineas[4]; // la línea 5 es la coordenada X
$prueba[$i]['minY']=(float)$lineas[5]; // la línea 6 es la coordenada Y
//Obtenemos el alto y el ancho de la imagen para calcular las otras coordenadas que faltan
$nombre_fichero= asset_url()."visibilityMaps/map_".$poi->poi_id.".png";
list($ancho, $alto, $tipo, $atributos) = getimagesize("./assets/visibilityMaps/map_".$poi->poi_id.".png");
//el ancho-alto de la imagen por el tamaño del pixel nos da el desplazamiento de los pixeles
$prueba[$i]['maxX']=((float)$lineas[4]+ ($ancho*(float)$lineas[0]));
$prueba[$i]['maxY']=((float)$lineas[5] +($alto*(float)$lineas[3]));
```

### 4.2.3. Vistas

Una vista es simplemente una página web o un fragmento de página, tal como un encabezado, pie de página, una barra lateral, etc. De hecho, las vistas se pueden embeber flexiblemente dentro de otras vistas (dentro de otras vistas, etc., etc.) si necesita este tipo de jerarquía.

No podemos llamar directamente a las vistas, las tiene que cargar un controlador. Vamos a ver como los controladores cargan las vitas:

```
$this->load->view('templates/header_admin');
$this->load->view('admin/admin_POI');
$this->load->view('templates/footer_admin');
```

Las llamadas `$this->load->view` cargan los datos desde el directorio de Views y dentro de este directorio busca los subdirectorios y carga los archivos php que contienen las vistas, si hay más de un tipo de llamadas codeingniter las cargara todas juntas y a la vez, permitiéndonos separar los distintos elementos de nuestra aplicación (cabecera, cuerpo, etc…) en varios archivos que podemos reutilizar en otros casos.

Los datos se pasan del controlador a la vista por medio de un array o un objeto en el segundo parámetro en la función de carga de la vista. De esta manera podemos pasar los datos necesarios para crear una tabla dinámicamente por ejemplo.

Las vistas están separadas si son de administración o son normales, las cabeceras y los pies de página también.

Como se explicó antes para la interfaz de usuario se eligió el framework de trabajo Bootstrap, y para hacerlo más amigable con los móviles y darle un estilo al más puro Material Desing de Android.

#### 4.2.3.1. Cabeceras

Estos archivos son los que incluyen los ficheros javascript y css para el correcto funcionamiento de la aplicación.

Como librerías JavaScript contamos con:

- **jQuery 2.1.3**, nos perímete de manera sencilla interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX
- **Bootstrap**: Los componentes de JavaScript para Bootstrap proveen elementos adicionales de interfaz de usuario como diálogos, tooltips y carruseles. Estos componentes están basados en jQuery.
- **jQuery UI** es una biblioteca de componentes para jQuery que le añaden un conjunto de plug-ins, widgets y efectos visuales.
- **jQuery UI Touch Punch** Activa los eventos de toque para los distintos elementos de jQuery UI
- **ripples:** Librería para el tema de Bootstrap de Material Desing
- **material**: Librería para el tema de Bootstrap de Material Desing
- **Bootstrap Colorpicker**: Plug-in colorpicker para Bootstrap.
- **PROJ4JS**: Librería JavaScript para transformar coordenadas de un sistema a otro, incluyendo transformaciones de datum.
- **Openlayers**: Librería JavaScript para la carga de mapas.
- **noUiSlider**: Librería JavaScript para crear un slider.

Como archivos CSS contamos con los siguientes:

- **Bootstrap**: CSS para los componentes de BootStrap
- simple-sidebar
- jquery-ui
- roboto.min.css
- material.min.
- ripples.min.css
- estilos.css
- nouislider.min.css

- bootstrap-colorpicker.css

#### 4.2.3.2. Elementos BootStrap.

Como se ha comentado antes BootStrap es un framework de trabajo para darle a las aplicaciones un diseño *responsive* el cual nos valdra para ver la aplicación en una pantalla normal de ordenador como en la pantalla de un dispositivo móvil. Para BootStrap los dispositivos móviles son lo mas importante.

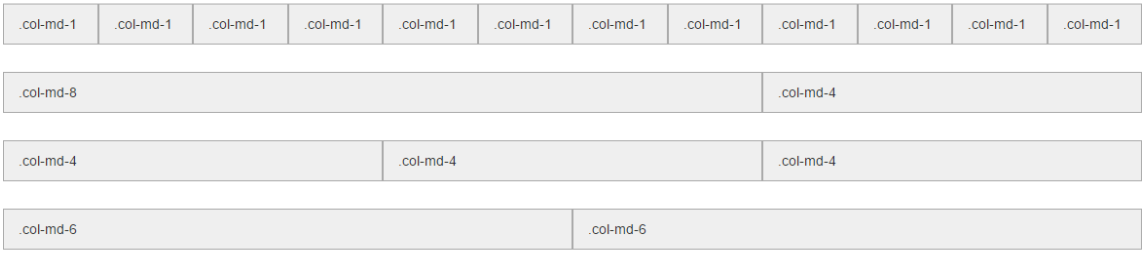
Antes de comenzar debemos preparar nuestra aplicación para que las páginas se muestren correctamente y el zoom funcione bien en los dispositivos móviles, es importante añadir la siguiente etiqueta dentro de la cabecera `<head>` de las páginas:

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

Bootstrap incluye una rejilla o retícula fluida pensada para móviles y que cumple con el diseño web *responsive*. Esta retícula crece hasta 12 columnas a medida que crece el tamaño de la pantalla del dispositivo. Bootstrap incluye clases CSS para utilizar la rejilla directamente en tus diseños y también define mixins de LESS para poder crear diseños más semánticos.

```
<div class="row">
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
</div>
```

```
<div class="row">
  <div class="col-md-8">.col-md-8</div>
  <div class="col-md-4">.col-md-4</div>
</div>
<div class="row">
  <div class="col-md-4">.col-md-4</div>
  <div class="col-md-4">.col-md-4</div>
  <div class="col-md-4">.col-md-4</div>
</div>
<div class="row">
  <div class="col-md-6">.col-md-6</div>
  <div class="col-md-6">.col-md-6</div>
</div>
```



La rejilla se ha utilizado en todas las páginas de administración aunque la parte de administración está restringida solo para ordenadores, esta se puede adaptar a todo tipo de pantallas.

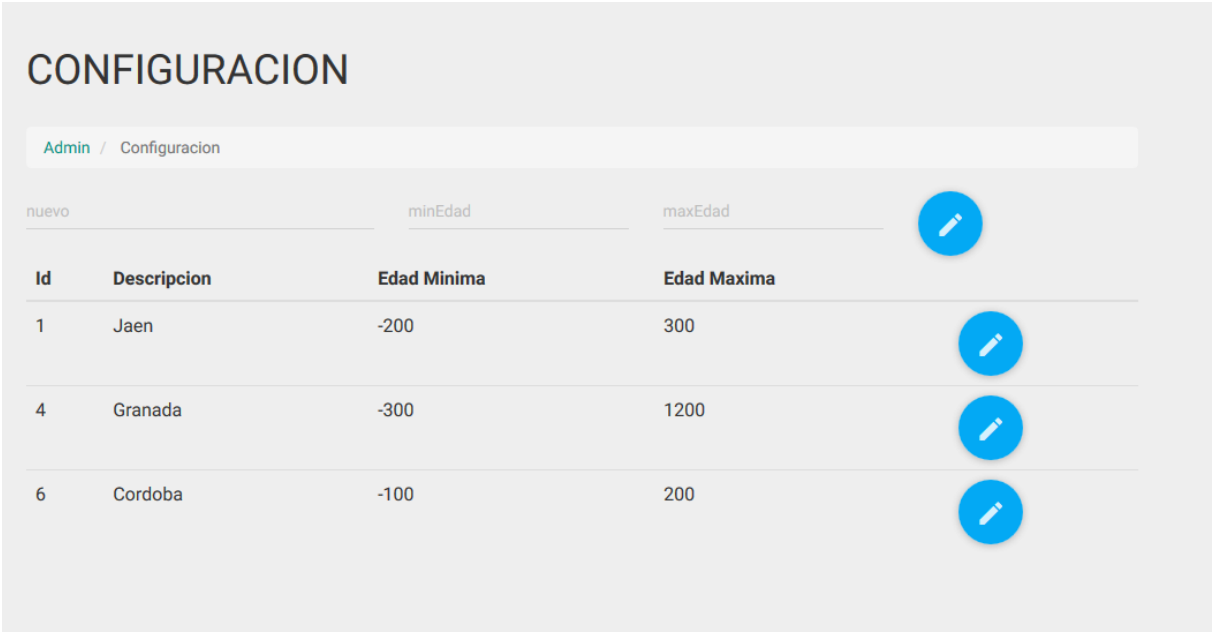



Ilustración 21. Pantalla ordenador.

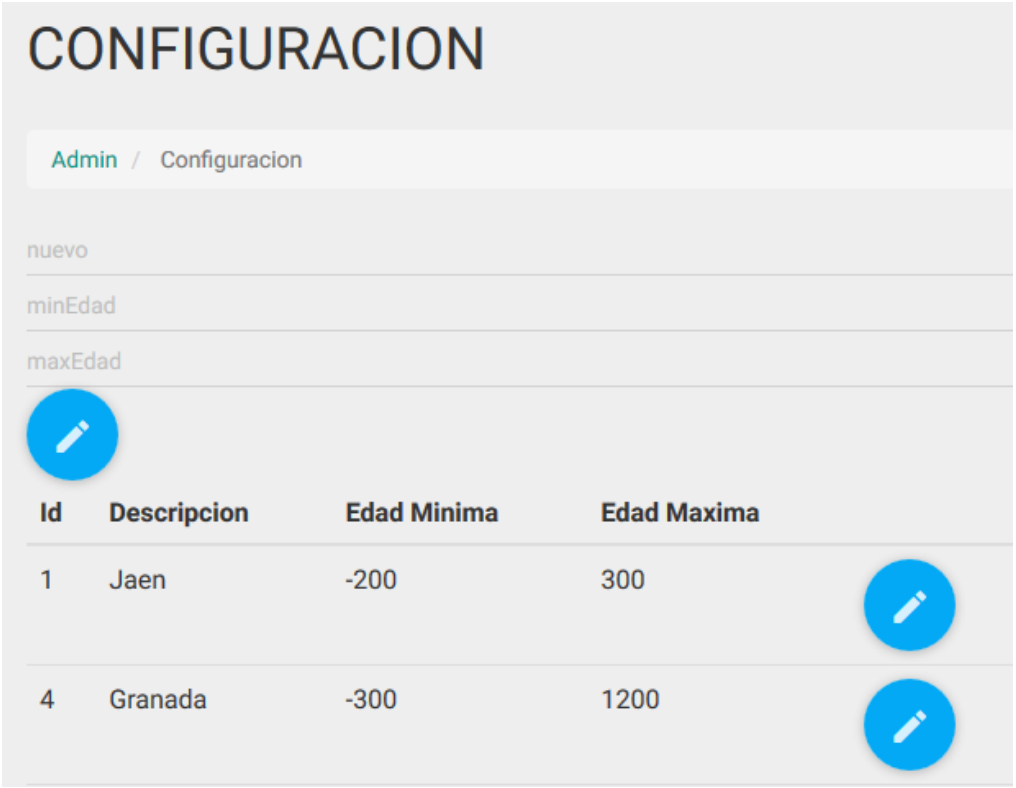



Ilustración 22. Pantalla dispositivo móvil

Estas dos imágenes son un ejemplo de como actua el sistema de rejilla en la parte de administración. Para los botones se ha utilizado el tema de Material Desing para BootStrap el cual puede dar ese estilo al botón.

Otro de los componentes más utilizados es modal.js, este componente JavaScript provee de los elementos necesarios para realizar ventanas emergentes que se sobrepongan por encima de todo.

El uso de este componente es muy sencillo simplemente creas el elemento HTML y lo dejas con el estilo display:none, y cuando desees lo muestras mediante javascript.

```
<div class="modal fade" id="confirm" tabindex="-1" role="dialog" aria-labelledby="myModalLabel">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
            <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                <span aria-hidden="true">&times;</span>
            </button>
            <h4 class="modal-title" id="titleConfirm"> Titulo</h4>
            </div>
            <div class="modal-body" id='bodyConfirm'>
                Cuerpo del Modal
            </div>
            <div class="modal-footer">
                Pie del modal
            </div>
        </div>
    </div>
</div>
```

Esto es para crear el elemento, al titulo, cuerpo y pie de la ventana Modal podremos acceder mediante los selectores de jQuery asi poder editar el contendido. Para mostrarlo/ocultarlo por pantalla seria de la siguiente forma.

```
$('#confirm').modal('toggle');
```

Y visualmente se vería de la siguiente forma.

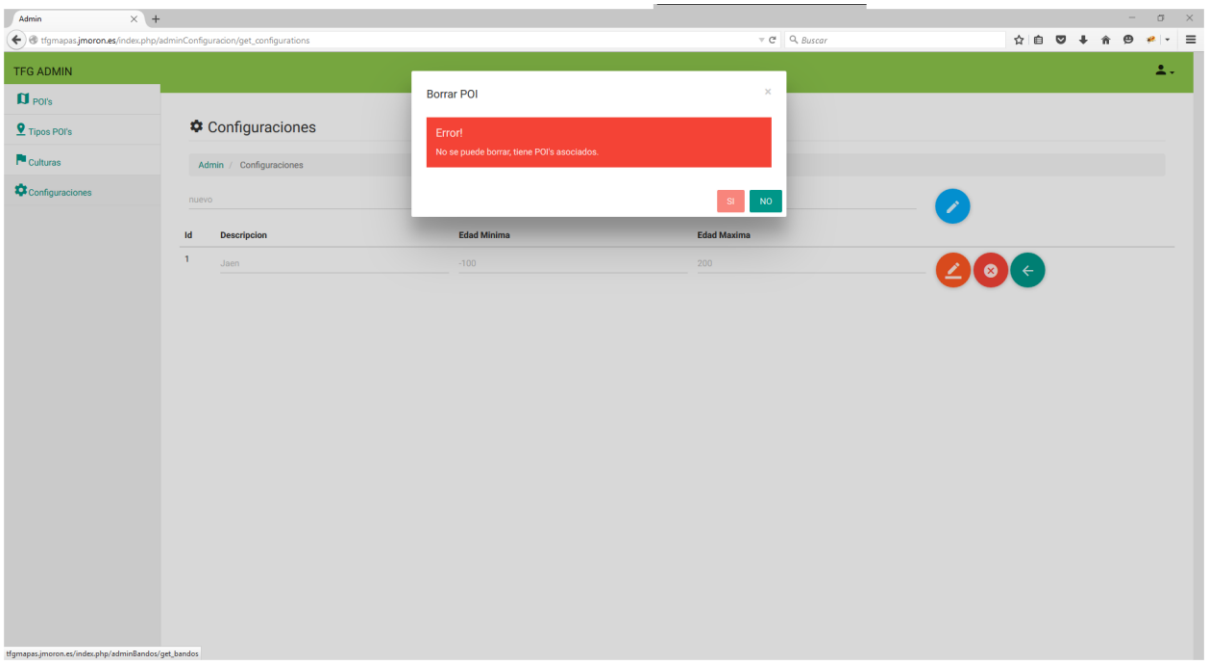



Ilustración 23. Ejemplo de Modal

Otro elemento utilizado son los Popovers, estos elemento son pequeñas cajas que se superponen con contendio. Para configurarlo utilizamos el siguiente código:

```
$(element).popover({
        'placement': 'top',
        'animation': true,
        'html': true,
        'content': html
    });
$(element).popover('show');
```

El primer parámetro es la posición del Popover, el siguiente si se activa la animación de tipo fade o no, el siguiente nos permite introducir código HTML, y finalmente el código HTML del Popover. Con la última línea mostramos el Popover.


Ilustración 24. Ejemplo de popover

#### 4.2.3.3. Usos de jQuery.

jQuery es uno de los complementos más esenciales para el desarrollo web, usado en millones de sitios en toda la web, ya que nos facilita mucho el desarrollo de aplicaciones enriquecidas del lado del cliente, en Javascript, compatibles con todos los navegadores.

Los usos mas básicos de jQuery son, cambiar la clase CSS de un elemento, editar el html del elemento, manejador de eventos, leer/editar el valor de campos de formularios.

```
$('#leyendaRango').css('modal-dialog modal-lg');
```

```
$('#btnCapa').html('Ocultar');
```

```
$('#delete').click(function(){
        var id=$('#data_id').val();
        var borrarEditar=$('#borrar_editar').val();
        if (borrarEditar==1)
                editConfirm(id);
        else
                borrarConfirm(id);
});
```

Pero quizás el más usado de todos sean las llamadas AJAX. AJAX, acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

Estas llamadas se utilizan en la aplicación para llamar las funciones de los controladores que anteriormente vimos y denominamos como JSON. También se utilizan para poder editar y borrar elementos dinámicamente dentro de las tablas. Vamos a ver varios ejemplos:

```
url__ = './adminConfiguracion/delete/'+id;
$.ajax({
            url: url__,
}).done(function() {
  $( this ).addClass( "done" );
});
```

Este caso es el más sencillo de todos ya que simplemente se llama al método del controlador correspondiente y como parámetro se le pasa el id.

```
url__ = 'adminConfiguracion/edit/'+descrip;
$.post(url__, {des: descrip, minEdad: minEdad, maxEdad: maxEdad, id: id}, function(respuesta) {
        nuevos();
});
```

Aquí vemos un ejemplo en el cual se ha sustituido la llamada $.ajax por $.post, esto significa que a la URL le vamos a pasar parámetro por el método POST. Como se ve el primero parámetro es la URL, luego los parámetros con ese formato y por último la función que se ejecutara cuando se complete la llamada AJAX. En este caso es utilizada para editar un elemento de la BBDD.

```
var URL = "./adminConfiguracion/get_configurations_json";
$.getJSON( URL)
.done(function( data ) {
        $.each( data, function( i, item ) {

                /**CODIGO PARA CREAR UNA nuevaFila, un String con código HTML*/
                $(nuevaFila).appendTo('#table-content');

        });
});
```

Este ejemplo ilustra las llamadas las cuales devuelven un JSON. Con .done le pasamos la función la cual llamara cuando termine de procesar la llamada AJAX. En este caso se hace uso de la función de jQuery $.each la cual representa un buble for

each, y para cada objeto del array data lo pasamos a una variable local llamada ítem para poder manejar más fácilmente el objeto JSON.

```
$.ajax({
        type: "GET",
        url: URL,
        async: false,//Ponemos que sea síncrono
        dataType: "json",
        success: function(data) {
                tam=data.tam;
        }
});
```

Como se ha dicho antes las llamadas AJAX por definición son asíncronas es decir que se produce la llamada y el código javascript no se queda parado hasta que esta termine, si no que se puede seguir ejecutando otro código. Pero qué pasa si queremos consultar un dato el cual vamos a necesitar en nuestro script más adelante, pues este ejemplo resuelve ese problema ya que convierte las llamadas AJAX en síncronas, las cual tienen que ser esperadas a que terminen. Esto es posible si ponemos la propiedad async a false.

#### 4.2.3.4. Rangos con noUiSlider

noUiSlider es una librería JavaScript que nos ayuda a la creación de slider de forma muy sencilla.

```
var softSlider = document.getElementById('PopupSlider');
noUiSlider.create(softSlider, {
        start: 50,
        range: {
                min: 0,
                max: 100
        }
});
```

Como se puede observar noUiSlider no hace uso de jquery si no de las funciones nativas de javascript. De esta forma estamos diciento que en el elemento del DOM con id PopupSlider cree un slider que este a la mitad y tenga de rango entre 0 y 100.

```
noUiSlider.create(slider, {
        start: [json.min_edad, json.max_edad], // Handle start position
        step: 10,
        connect: true, // Display a colored bar between the handles
        direction: direccion,
        orientation: orientacion,
        behaviour: 'tap-drag', // Move handle on tap, bar is draggable
        range: {// Slider can select '0' to '100'
                'min': Number(json.min_edad),
                'max': Number(json.max_edad)
        }
});
```

Este es un caso más completo de creación del slider. Vamos a ver cada uno de los parámetros de creación:

- Start: indicamos el valor Handle o en este caso el de los dos Handle. Step: indicamos el paso que tiene el Handle cuando se mueve, en este caso el valor se incrementra o disminuirá de 10 en 10.
- Connect: si se colorea el rango entre los Handle.
- Direction: la dirección del slider por defecto es de izquierda a derecha (ltr) es decir el minimo estará a la izquierda y el máximo a la derecha, también puede ser de derecha a izquierda (rtl). En el caso de que la orientación sea vertical ltr significa de arriba abajo y rtl de abajo arriba.
- Orientation: la orentacion del slider por defecto horizontal pero se puede configurar para ponerla vertical con el valor *vertical.*
- Behaviour: la manera en la que selecciona el Handle en este caso es el modo de pulsar y arrastrar.
- Range: el rango del slider con un valor minimo y un máximo.

```
slider.noUiSlider.set(arrayEdad);
```

También cuenta con los métodos para poder editar el valor del slider asi como la posición del Handle. Si tiene dos Handle se le pasara un array con dos datos, en el caso de que solo se quiera actualizar uno, en el array uno de los datos será nulo. Si solo tiene un Handle se pasa el valor.

```
slider.noUiSlider.on('change', function (values, handle) {
        $('#MinEdad').val(values[0]);
        $('#MaxEdad').val(values[1]);
        arrayEdad = values;
        actualizarCapas();
});
```

Otra cosa es que se le pueden poner eventos como change que llamara a la función que se describe cada vez que cambie el valor del slider.

```
slider.noUiSlider.destroy();
```

Para cambiar un slider primero hay que destruirlo y después crearlo de nuevo. Para destruirlo simplemente llamamos a esta función.

Visualmente el slider quedaría de las siguientes formas.

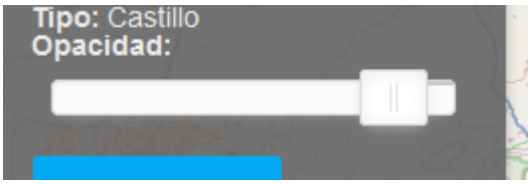



Ilustración 25. Ejemplo de noUiSlider con un Handle

Un slider simple para manejar la opacidad de los mapas de visibilidad dentro del mapa.

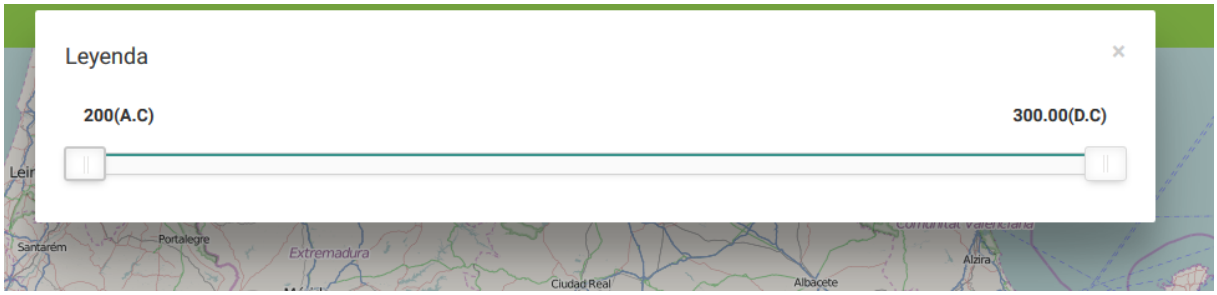



Ilustración 26. Ejemplo de noUiSlider con dos Handles

Un slider con dos Handles para poder seleccionar el rango de edades por el cual filtrar los mapas de visibilidad.

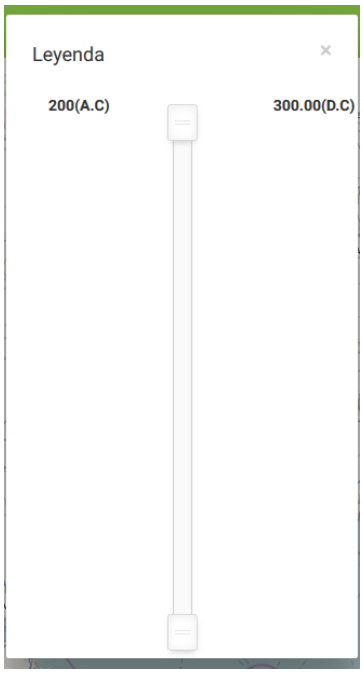



Ilustración 27. Ejemplo noUiSlider vertical.

Y un slider vertical para el diseño móvil y así aprovechar el máximo de pantalla. Este slider cumple la misma función que el anterior.

## 4.3. Openlayers.


Ilustración 28. Logo OpenLayers 3

OpenLayers es una biblioteca de JavaScript de código abierto bajo una derivación de la licencia BSD para mostrar mapas interactivos en los navegadores web. OpenLayers ofrece un API para acceder a diferentes fuentes de información cartográfica en la red: Web Map Services, Mapas comerciales (tipo Google Maps, Bing, Yahoo), Web Features Services, distintos formatos vectoriales, mapas de OpenStreetMap, etc.

Inicialmente fue desarrollado por MetaCarta en Junio del 2006. Desde el noviembre del 2007 este proyecto forma parte de los proyectos de Open Source

Geospatial Foundation. Actualmente el desarrollo y el soporte corren a cargo de la comunidad de colaboradores.

Crear un mapa con Openlayer es fácil, primero tenemos que incluir la librería.

```
<script src="http://openlayers.org/en/v3.8.2/build/ol.js" type="text/javascript"></script>
```

Luego introducimos el div donde va a ir el mapa y le asociamos estilos CSS para configurar el alto y el ancho.

```
<div id="map" class="map"></div>
<style>
    .map {
      height: 400px;
      width: 100%;
    }
</style>
```

Y finalmente con JavaScript llamamos al constructor ol.Map para crear el mapa.

```
var map = new ol.Map({
    target: 'map',
    layers: [
      new ol.layer.Tile({
        source: new ol.source.MapQuest({layer: 'sat'})
      })
    ],
    view: new ol.View({
      center: ol.proj.transform([37.41, 8.82], 'EPSG:4326', 'EPSG:3857'),
      zoom: 4
    })
  });
```

Como podemos ver en este ejemplo el constructor del mapa recibe tres parámetros los cuales son los básicos para poder crear un sencillo mapa. El primero de todos es el id en el cual se va a dibujar/renderizar el mapa. El segundo son las capas que va a tener el mapa. Y por último tenemos la vista aquí se configura el centro de la vista y el zoom inicial del mapa.

A continuación vamos a ver distintos elementos de openlayers que se han utilizado en esta aplicación.

#### 4.3.1. Capa base o capa principal.

Como hemos dicho antes esta capa es la que primero se dibuja, por lo tanto esta de fondo de nuestro mapa y por donde vamos a empezar a construir toda nuestra

aplicación. Openlayer puede trabajar con cualquier sevidor de mapas, por lo tanto se puede elgir cualquier capa base. Para esta aplicación se ha elegido los mapas de Open Street Maps.

Ilustración 29. Logo OpenStreetMap

OpenStreetMap (también referido como OSM) es un proyecto colaborativo para crear un mapa libre y actualizable de todo el mundo; por medio de una comunidad de usuarios, es decir personas con un objetivo común, que ceden su tiempo desinteresadamente y sin fines de lucro para tener la posibilidad de ver, copiar, modificar, y usar información geográfica (como esta) de cualquier parte del mundo sin restricciones de ningún tipo. OpenStreetMap pretende crear un mapamundi al igual que la Wikipedia pretende crear una enciclopedia.

La elección de este tipo de cartografía ha sido por la sencillez que se integra en OpenLayers, por la gran comunidad que tiene tras de sí aportando datos al mapa.

```
var capaBase = new ol.layer.Tile({
   source: new ol.source.OSM()
});
```

Así de sencillo es crear una capa de OSM en openlayers, el tipo de capa es Tile o por rejilla.

### 4.3.2. Capas mapas de visibilidad

La siguiente capa es la capa de los mapas de visibilidad. Al ser varios mapas los que se muestren a la vez necesitamos un conjunto de datos que nos permita visualizar varias capas a la vez. OpenLayers ofrece un objeto llamado LayerGroup

que consiste en un grupo de capas para ello hay que asignarle otro objeto llamado Collection que funciona lo mismo que un vector.

Primero creamos de forma global el objeto Collection.

```
var arrayCapasMapas = new ol.Collection();
```

Luego vamos asignando capas al collection. Las capas que se van a asignar son del tipo image. Estas capas necesitan una fuente para coger las imágenes, dicha fuente necesita una URL, una caja envolvente 2D, y una proyección.

```
var extent = [item.minX, item.minY, item.maxX, item.maxY];
var newLayer = new ol.layer.Image({
        source: new ol.source.ImageStatic({
                url: './assets/visibilityMaps/map_' + item.poi_id + '.png',
                imageExtent: ol.proj.transformExtent(extent, 'EPSG:23030', 'EPSG:3857'),
                //Transformamos la BB de ED50 a WGS84 Web Mercator
                projection: projPrin
        })
});
```

Los datos necesarios para crear una capa los cojemos atraves de AJAX y por medio de JSON podemos tener objetos con los cuales leer los datos necesarios para crear la caja envolvente y consultar el id. Las coordenadas de la caja envolvente deben ser transformadas de su procyeccion a la del mapa. En la aplicación como todos lo mapas de visibilidad están en la proyección ED50 uso 30N se van a pasar de esa proyección a WGS84 Web Mercator que es la proyección principal. Para ello primero tenemos que definir la proyección ED50:

```
proj4.defs("EPSG:23030", "+proj=utm +zone=30 +ellps=intl" +
    " +towgs84=-131,-100.3,-163.4,-1.244,-0.020,-1.144,9.39 " +
    " +units=m +no_defs");
```

Cuando tengamos la capa creada hay que añadirla al objeto Collection creado antes y también le añadimos el id y editamos la opacidad al 85%

```
newLayer.set('id', item.poi_id);//AÑADIMOS EL ID PARA PODER IDENTIFICARLO
newLayer.setOpacity(0.85);
arrayCapasMapas.push(newLayer);
```

Finalmente añadimos el objeto collection al objeto layer.Group y esta capa se la podemos añadir al mapa.

```
//**CREAMOS LA CAPA CON EL CONJUNTO DE CAPAS***/
var mapasVisibildad = new ol.layer.Group({
   layers: arrayCapasMapas
});

var map = new ol.Map({
   layers: [
      capaBase,
      mapasVisibildad
   ],...
```

Para mostrar/ocultar el mapa cambiamos la propiedad visible del mapa

```
function ocultarCapa(idFeat){
   for (i = 0; i < arrayCapasMapas.getLength(); i++) {
      var capa = arrayCapasMapas.item(i);
      var id = capa.get('id');
      if (idFeat === id) {
         capa.setVisible(!capa.getVisible());
         if (capa.getVisible()){
            $('#btnCapa').html('Ocultar');
         }else{
            $('#btnCapa').html('Mostrar');
         }
      }
   }
}
```

Para ver que capa debemos buscar dentro del vector collection cual capa queremos, para ello comparamos con el id de la capa cambiamos el valor de la propiedad Visible. Para la opacidad de cada capa utilizamos el mismo método.

```
for (i = 0; i < arrayCapasMapas.getLength(); i++) {
            var capa = arrayCapasMapas.item(i);
            var id = capa.get('id');
            if (featu.get('id') === id) {
                        softSlider.noUiSlider.set(capa.getOpacity()*100);
            }
}
```

### 4.3.3. Capa de markers.

Los marker son los iconos los cuales señalan un punto dentro del mapa. Para esta capa se ha utilizado el mismo método que en las capas de visibilidad. Para guardar los marker en vez de utilizar una Collection utilizamos un source.Vector. Esta será la capa de los marker

```
var vectorSource = new ol.source.Vector({//VECTOR PARA LOS MARKET
   //create empty vector
});
```

En este caso se añaden objetos del tipo Feature

```
var coordinates = [item.poi_X, item.poi_Y];
var iconFeature = new ol.Feature({
            geometry: new ol.geom.Point(coordinates),
            name: item.poi_des,
            population: 4000,
            rainfall: 500
});
```

Para estos objetos se necesita una geometría del tipo punto, a la cual le pasamos las coordenadas del POI. Con esto no muestra el icono del tipo de POI para ello necesitamos un style que mostramos a continuación.

```
var iconStyle2 = new ol.style.Style({
        image: new ol.style.Icon(/** @type {olx.style.IconOptions} */ ({
                anchor: [1, 1],
                anchorXUnits: 'pixels',
                anchorYUnits: 'pixels',
                opacity: 1,
                src: './assets/img/IconsPOIs/tipoPOI_' + item.tipo_id + '.png'
        }))
});
iconFeature.set('id', item.poi_id);//AÑADIMOS EL ID PARA PODER IDENTIFICARLO
iconFeature.set('descrip', item.poi_des);
iconFeature.set('tipoId', item.tipo_id);
iconFeature.set('bandoId', item.bando_id);
iconFeature.set('edadMax', item.poi_max);
iconFeature.set('edadMin', item.poi_min);
iconFeature.setStyle(iconStyle2);//Le añadimos el estilo segun el tipo de POI
```

Con esto creamos el estilo con el icono del tipo de POI y le añadimos las distintas propiedades que necesitamos.

### 4.3.4. Eventos del mapa.

Para poder hacer que se muestre los popover de los POI tenemos que añadirle un evento onclick al mapa. Openlayers nos ofrece un manejador de envetos para nuestro mapa y asi de manera sencilla podemos crear dicho evento e identificar si el lugar donde se ha pulsado es un icono o no.

Para crear dicho evento utilizamos la función del objeto map, le pasamos el evento en este caso click y la función la cual tiene que reailzar.

```
//Evento Clck Mapa
map.on('click', function (evt) {
   var element = popup.getElement();
   $(element).popover('destroy');
   var feature = map.forEachFeatureAtPixel(evt.pixel,
         function (feature, layer) {
            return feature;
         });
   if (feature) {
      crearPopOver(feature);
   }
});
```

Como vemos lo primero que hacemos es destruir cualquier popover que hubiera en el mapa. A continuación comprobamos si donde se ha pulsado es un objeto Feature, si se el objeto no es nulo llamamos a la función en la cual se crea los popover.

Para crear un popover dentro del mapa primero necesitamos colocar el popover dentro de una superposición en el mapa esto es posible gracias al método addOverlay del mapa.

```
//Añadimos el POPUP Al mapa
var popup = new ol.Overlay({
   element: document.getElementById('popup')
});
map.addOverlay(popup);
```

Luego en la función crearPopOver simplemente recuperamos el elemento de la superposición y creamos el PopOver como se ha visto en el apartado anterior.

### 4.3.5. Controles del mapa.

Openlayers por defecto en el mapa solo trae los controloles de zoom, pero si queremos incluir controles para mostrar una leyenda o para otra cosa debemos incluirlos dentro del mapa. Para crear más controles al mapa tenemos que utilizar el objeto Control. Para poder asignar un objeto control al mapa hay que seguir varios pasos. Primero debemos crear un namespace.

```
/**
 * Define a namespace for the application.
 */
window.app = {};
var app = window.app;
```

A continuación definimos la clase JavaScript la cual tendrá el código necesario para crear los controles del mapa.

```
/**
 * @constructor
 * @extends {ol.control.Control}
 * @param {Object=} opt_options Control options.
 */
app.LeyendaControl = function (opt_options) {

   var options = opt_options || {};

....Continua
```

Una vez creado este objeto, dentro del constructor vamos creando los elementos necesarios para crear los botones y darle funcionalidad. Para ello primero creamos la función que se llamara en los eventos de los botones.

```
var onClickTipos = function (e) {
            consultarTipos();
            $('#confirm').modal('toggle');
};
```

Como vemos esta función llama a otra la cual configurara el modal de la leyenda y después visualizara el modal. Después de esto debemos crear el botón el cual le añadiremos los eventos para que llame a esta función

```
var buttonTipos = document.createElement('button');
buttonTipos.innerHTML = 'T';
buttonTipos.title = 'Tipos';
buttonTipos.className = 'TipoButton';
buttonTipos.addEventListener('click', onClickTipos, false);
buttonTipos.addEventListener('touchend ', onClickTipos, false);
```

Con esto creamos un elemento HTML del tipo button, al cual le añadimos el código HTML, el título, la clase CSS los eventos. Para nuestro control de leyenda añadimos dos eventos el click el cual se realiza con el raton del PC y el touchend el cual representa el final de un toque en una pantalla táctil. A continuacion debemos crear un elemento div que contenga todos los botones creados.

```
var element = document.createElement('div');
element.className = 'leyendaControl ol-unselectable ol-control';
element.title = 'Tipos';
element.appendChild(buttonConfiguracion);
element.appendChild(buttonTipos);
element.appendChild(buttonBandos);
element.appendChild(buttonRango);
```

Creamos el elemento HTML div y le añadimos la clase CSS y los botones los cuales serán los que tengan el control de la leyenda. Despues tenemos que llamar al constructor de la clase Control al cual le pasamos el div que acabaos de crear y un objeto target que será uno de las propiedades del objeto pasado por parámetro.

```
ol.control.Control.call(this, {
        element: element,
        target: options.target
});
```

Por ultimo debemos hacer que la clase creada herede de la clase ol.control.Control y con esto ya podemos añadir nuestros propios controles a nuestro mapa.

```
ol.inherits(app.LeyendaControl, ol.control.Control);

var map = new ol.Map({
  controls: ol.control.defaults({
    attributionOptions: /** @type {olx.control.AttributionOptions} */ ({
      collapsible: false
    })
  }).extend([
    new app.LeyendaControl()
  ]), ... Continua
```

Con la primera línea hacemos que la clase que hemos definido antes herede de la clase ol.control.Control la cual nos permite que OpenLayers sea capaz de identificar elementos de HTML y sus eventos asociados y colocarlos sobre el canvas del mapa. En las últimas líneas vemos como le añadimos los controles personalizados creando una nueva instancia de la clase antes definidas.

# 5. Conclusiones y posibles mejoras.

En las últimas décadas los SIG como se conocen, esto es, en programas de escritorio, están evolucionando gracias al desarrollo tecnológico de la informática e internet desarrollando aplicaciones que permiten desplegar los mapas en la web. Este trabajo ha tratado de crear una aplicación de mapas en la cual el usuario sea capaz de ver los mapas de visibilidad en dicha aplicación. También ha sido capaz de llevar esta aplicación a los dispositivos móviles.

En los últimos años las aplicaciones web dirigidas a los dispositivos móviles ha crecido gracias a los diseños responsivos y a los frameworks que ayudan al diseño de estas aplicaciones haciéndolo más fácil. Como cabía de esperar hoy en día una aplicación web sin un framework como codeigniter o como bootstrap, para el diseño adaptativo de dispositivos móviles, es una locura ya que estos framework al ser totalmente gratuitos y de rápido aprendizaje hacen que el desarrollo de aplicaciones web sea fáciles, rápidas y poco costosas.

Aunque el trabajo cumple con los requisitos aún queda algunas mejoras que se pueden hacer.

Hay periodos históricos en los que los no hubo ningún cambio. Por lo tanto cabe de esperar que los rangos de edad sean dinámicos no estáticos como se ha diseñado desde un principio. Esta mejora, es posible gracias al slider noUiSlider, cuando se empezó a desarrollar la aplicación esta biblioteca no podía hacer rangos dinámicos, pero en su última versión lo permite.

Otra mejora seria que los mapas de visibilidad no dependieran de la proyección ED50 si no que se pudiera subir cualquier mapa en cualquiera proyección y la aplicación sea capaz de transformar dicha proyección. Esta mejora es más complicada que la anterior ya que implica subir un archivo .prj, los SIG lo utilizan para determinar la proyección de la imagen raster, para cada uno de los mapas y al ser leídos hay que cambiar dicha proyección a la del mapa base.

La última mejora es pasar del 2D al 3D. Los mapas son en 2D pero en los últimos años WebGL ha crecido como opción para crear contendido 3D dentro de la web. Recientemente Google Maps en su versión web se puede ver una representación del mundo en 3D. Para openlayers ya existe un proyecto en el cual se puede ver una superposición de una imagen sobre el cañón del colorado en 3D.

## Tabla de ilustraciones.

## Bibliografía

Benkler, Y. (2012). *El Pingüino y el Leviatán: Por qué la cooperación es nuestra arma más valiosa para mejorar el bienestar de la sociedad.*

*Bootstrap*. (s.f.). Obtenido de http://startbootstrap.com/

*BootStrap ColorPicker*. (s.f.). Obtenido de http://mjolnic.com/bootstrap-colorpicker/

*bootstrap material design*. (s.f.). Obtenido de https://fezvrasta.github.io/bootstrap-material-design/bootstrap-elements.html

Carrillo, R. C. (s.f.). Practica 8. *Trabajar con un MDE*.

*JQuery API*. (s.f.). Obtenido de https://api.jquery.com/

LanceTalent. (s.f.). *LanceTalent*. Obtenido de http://www.lancetalent.com/blog/tipos-de-aplicaciones-moviles-ventajas-inconvenientes/

*Manual de usuario Codeigniter*. (s.f.). Obtenido de https://ellislab.com/codeigniter/user-guide/index.html

*Manual PHP para imagenes*. (s.f.). Obtenido de http://php.net/manual/es/ref.image.php

*noUiSlider*. (s.f.). Obtenido de http://refreshless.com/nouislider/

*Openlayers API*. (s.f.). Obtenido de http://openlayers.org/en/v3.7.0/apidoc/index.html

PÉREZ, L. M. (s.f.). *Estudios de visibilidad en Arqueologia*. Obtenido de IniSIG : http://inisig.com/estudios-de-visibilidad-en-arqueologia/

*stackoverflow*. (s.f.). Obtenido de http://stackoverflow.com/

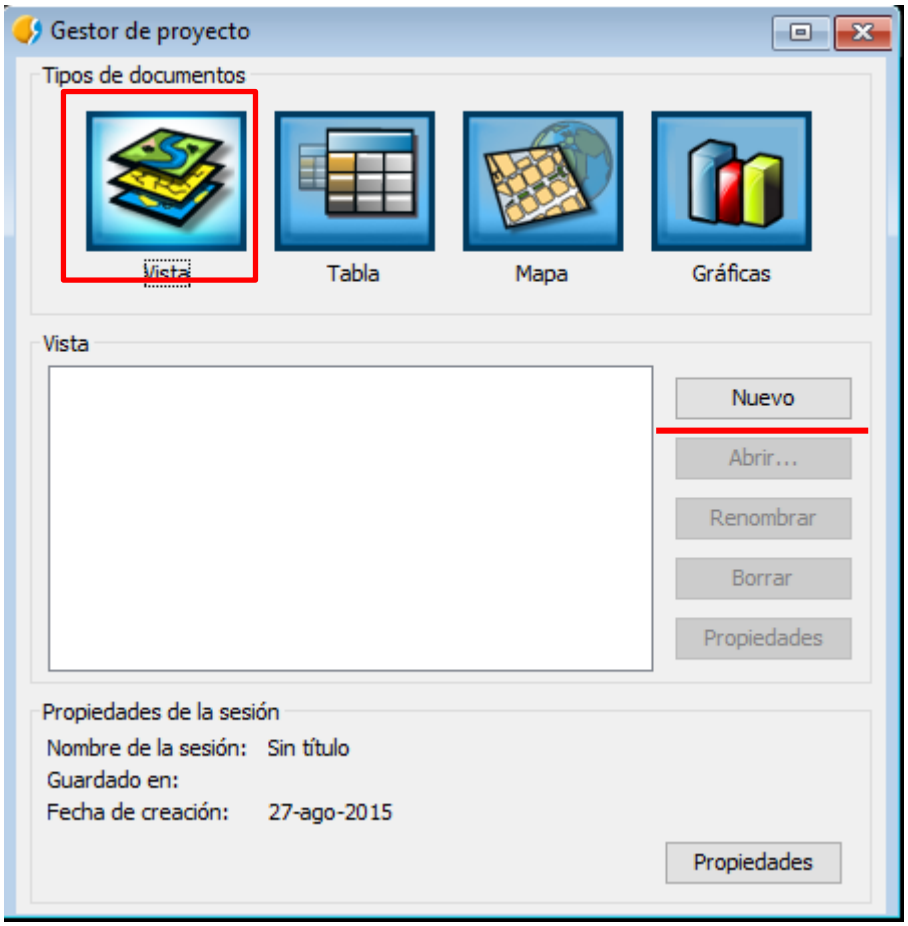
Velasco, J. G. (2009). *Energías renovables.* Editorial Reverte.

Wikipedia. (s.f.). *Wikipedia*. Obtenido de World File: https://en.wikipedia.org/wiki/World_file

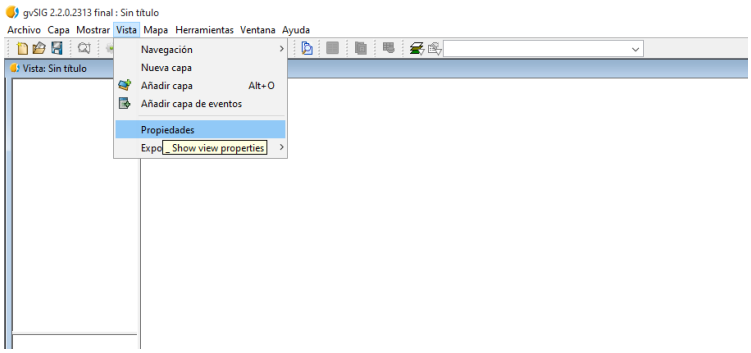
## Anexo I. Como crear Mapas de visibilidad y pasarlos a PNG georreferenciados.

En este anexo vamos a ver cómo crear los mapas de visibilidad con gvSIG para después transformalos con global mapper a png georreferenciados.

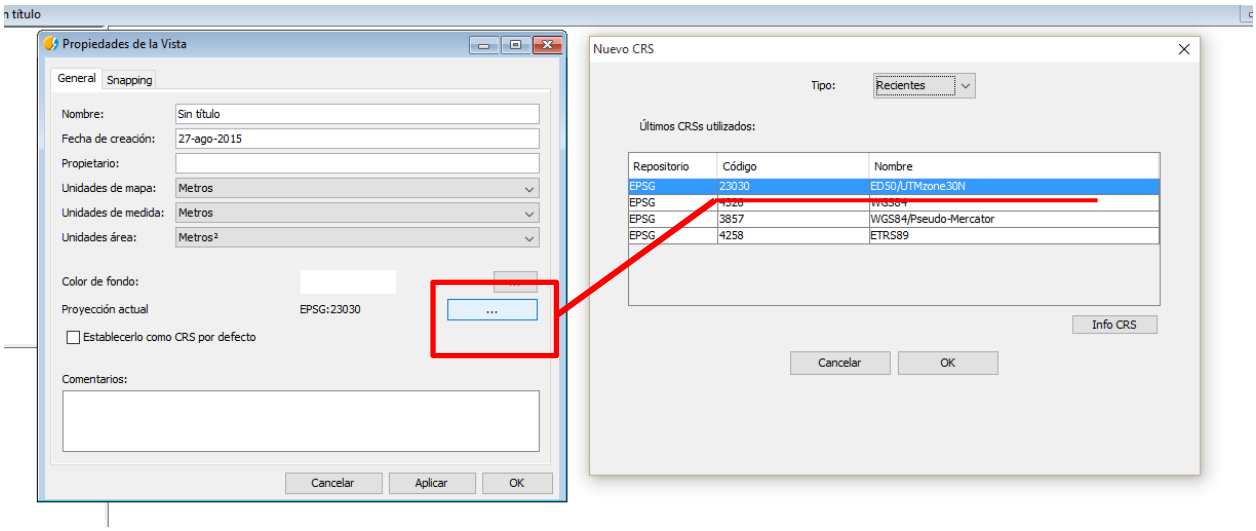
1. Abrimos el gvSIG.
2. Seleccionamos nueva vista.



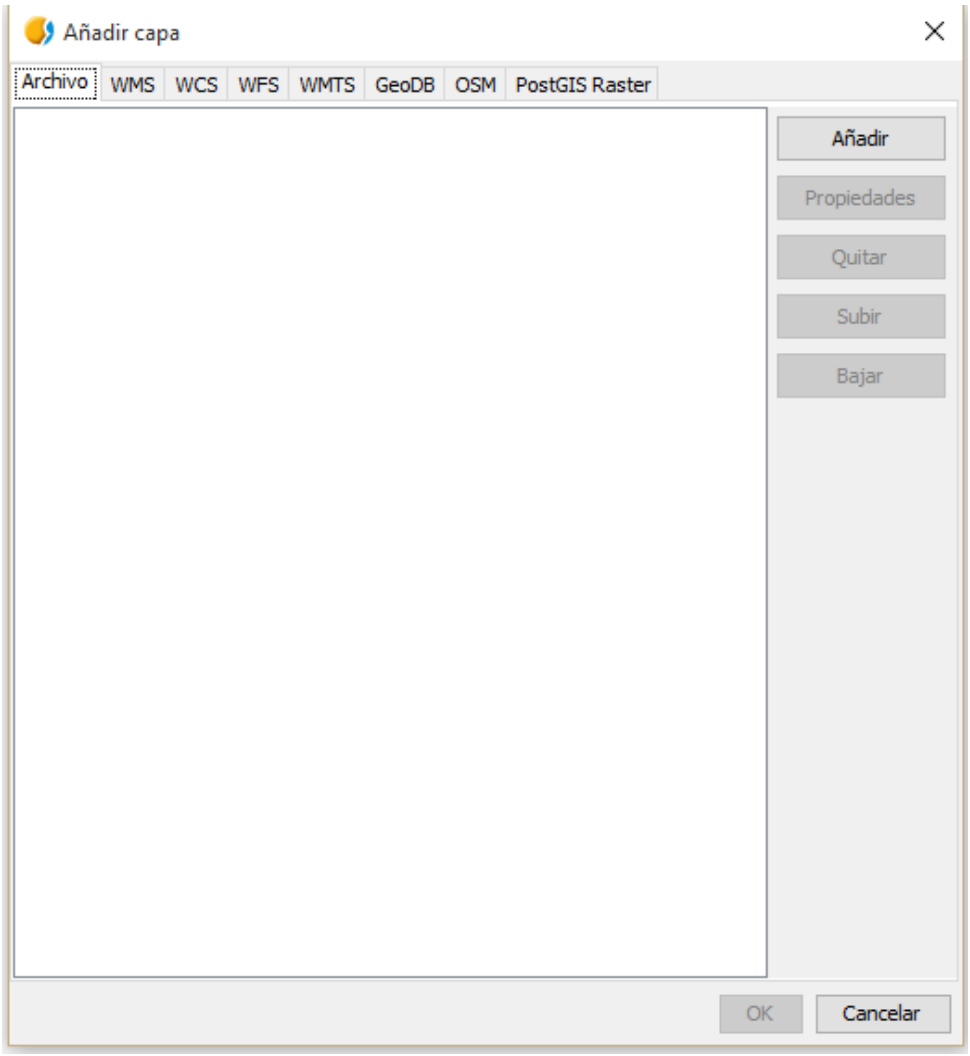

3. Ahora pinchamos sobre vista>propiedades:



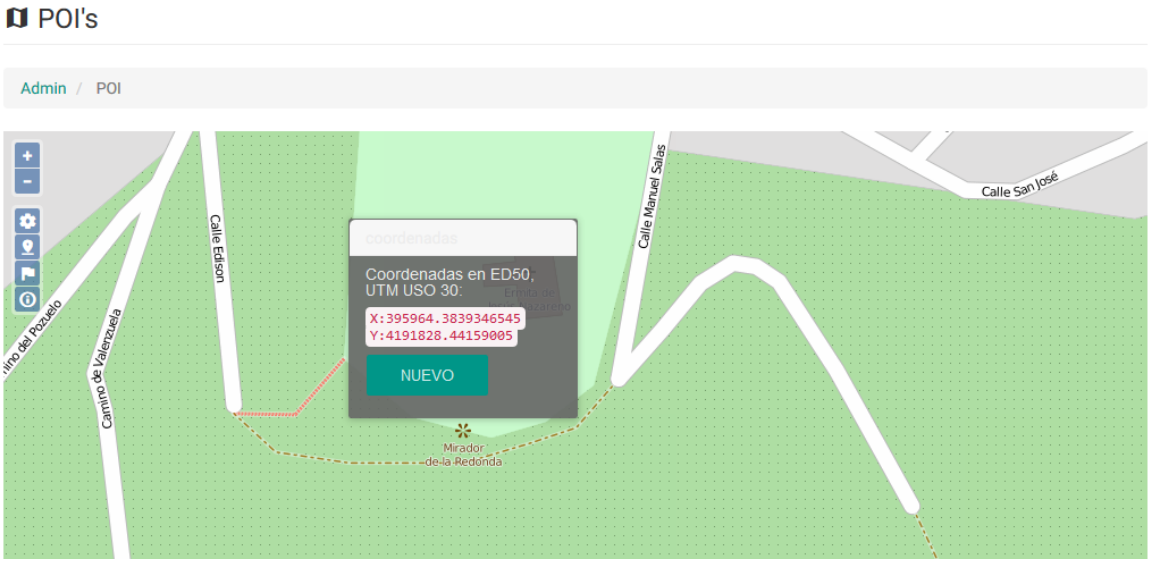

4. Y seleccionamos la proyección de nuestro mapa que en nuestro caso es la ED50 uso 30 cuyo código EPSG es 23030



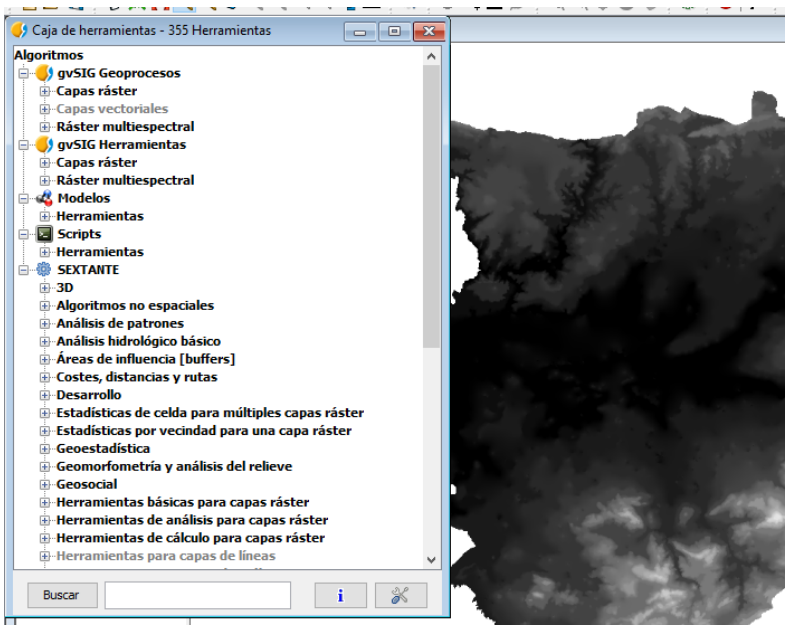

5. A continuación pichamos sobre vista>nueva capa



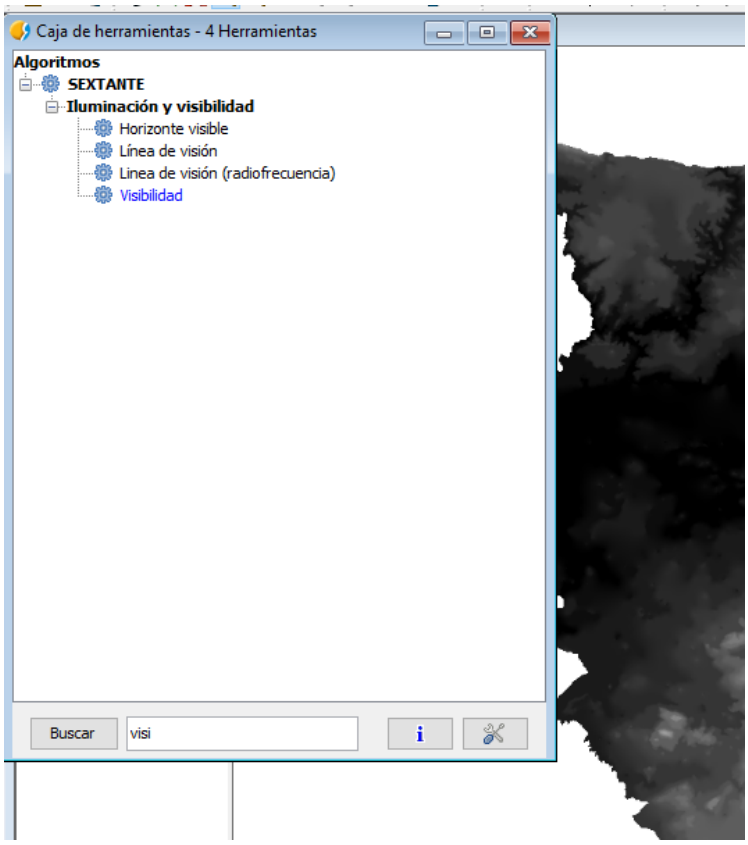

6. Seleccionamos la imagen MDE jaen_50m_recortado_rellenado.tiff, y le damos a teseladas.
7. Para saber las coordenadas en ED50 uso 30 de algún punto nos podemos ir a nuestra aplicación, Administracion>POI's y en el mapa cuando pinchemos sobre una zona nos saldrán las coordenadas X e Y en la proyección ED50 uso 30



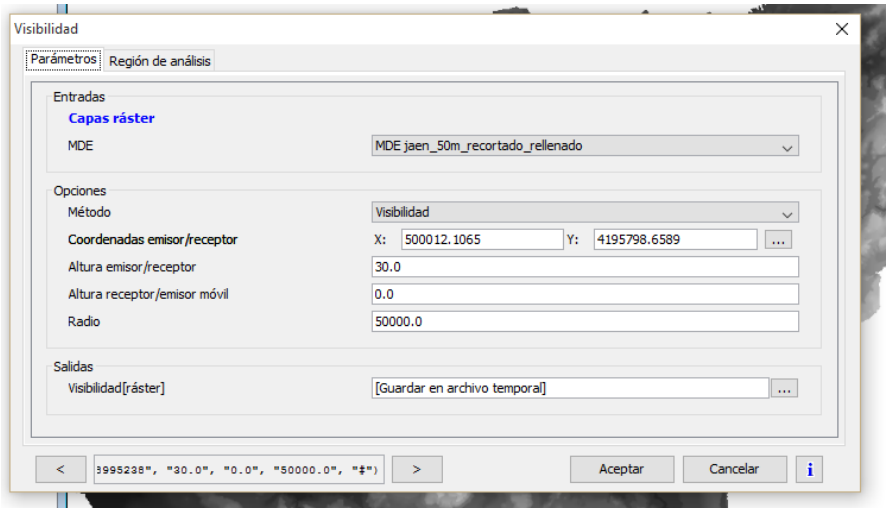

8. Una vez obtenidas las coordenadas desde donde vamos a realizar el estudio de visibilidad. Nos vamos al GvSIG, en el menú Herramientas>Geoprocesamiento>Caja de Herramientas.




9. Buscamos la herramienta de visibilidad y la abrimos.

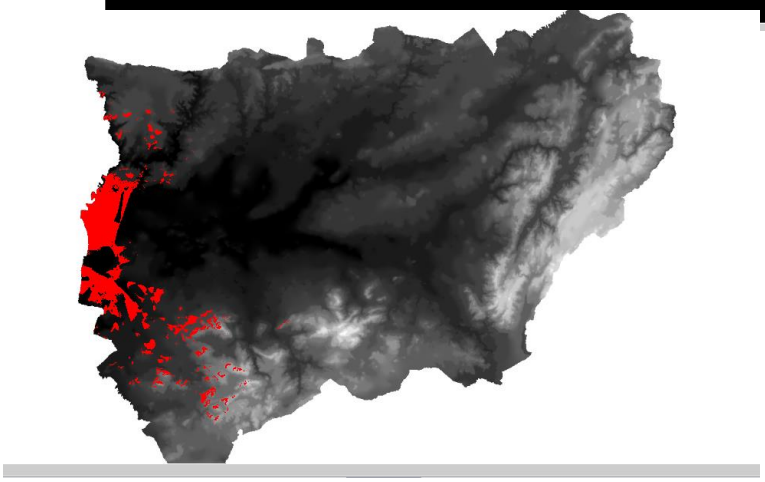



10. Ahora en los campos X, Y introducimos las coordenadas buscadas con nuestra aplicación. Introducimos la altura del POI, y el radio. Si el radio es demasiado grande tardara más en procesar los datos.



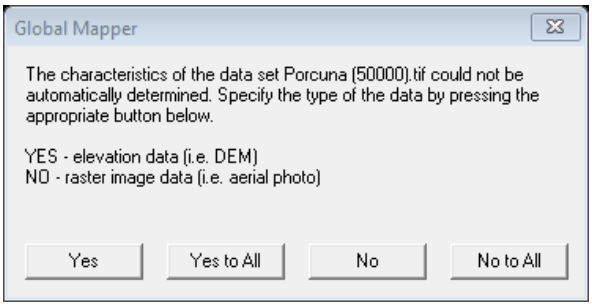

11. Una vez terminado el proceso el resultado que vemos es algo como esto. Si queremos podemos activar las tablas de color de la capa para ver el resultado mejor.

Sin Tablas de Color

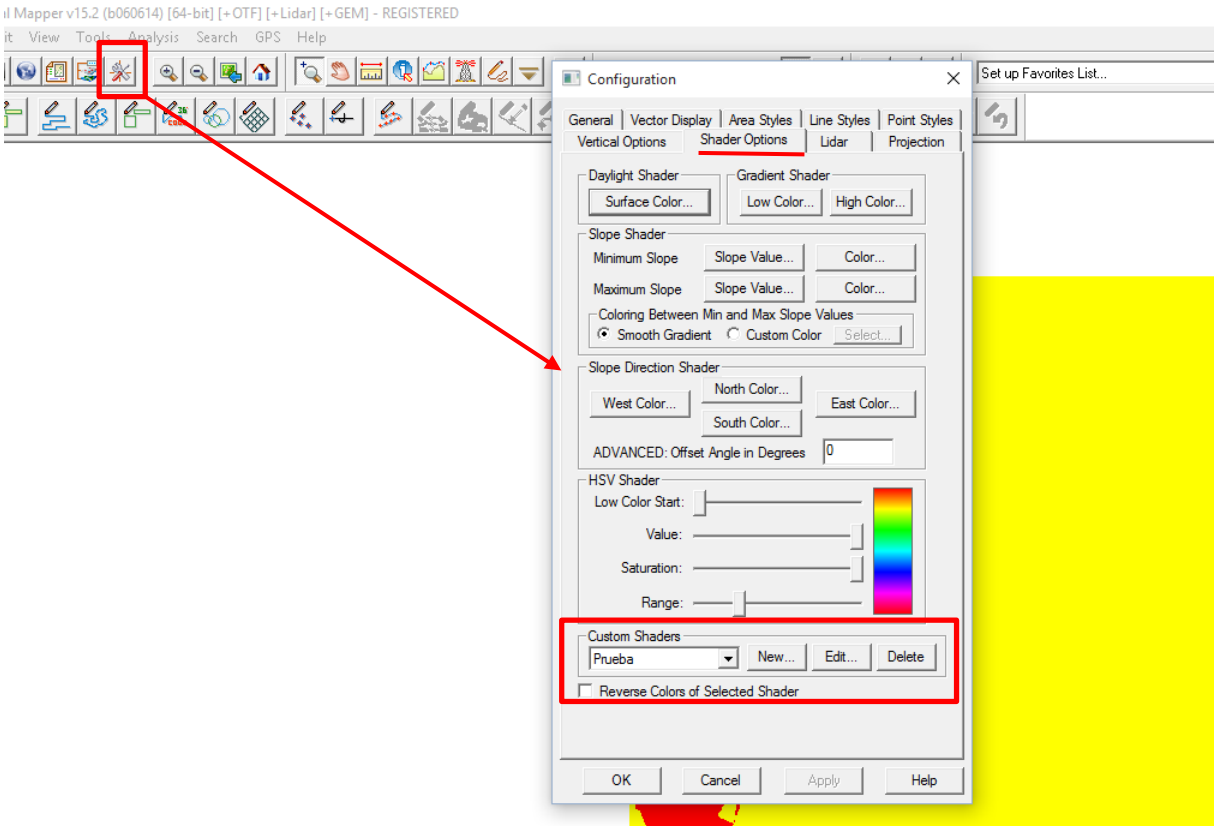


Con Tablas de Color

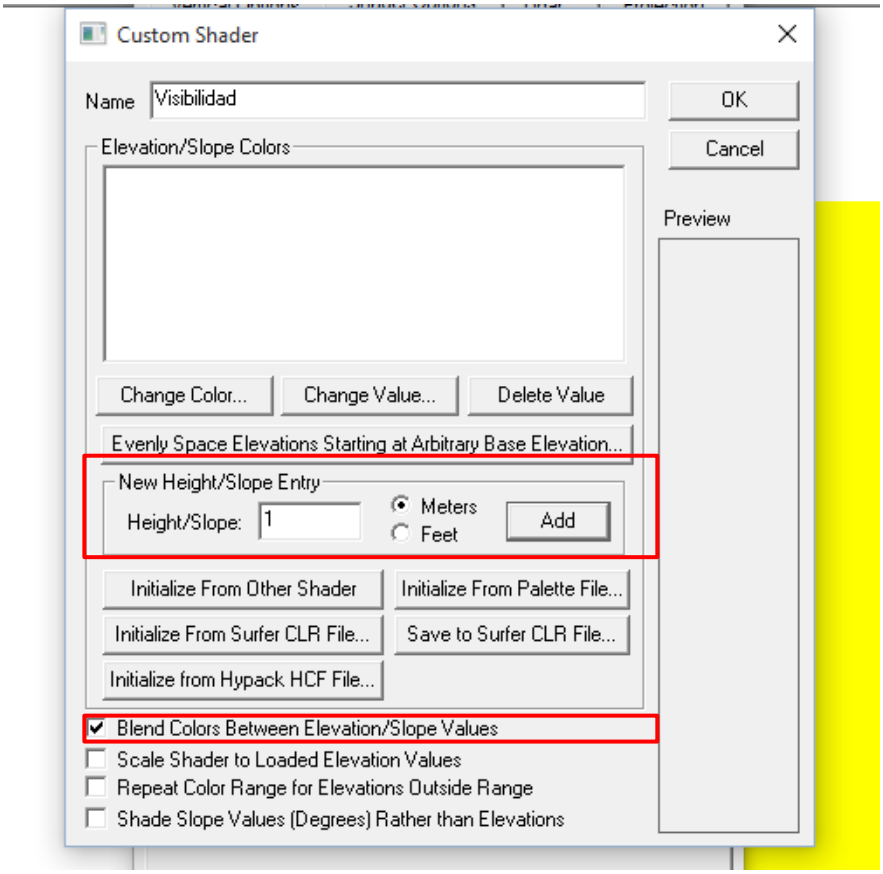
12. Una vez obtenidos los mapas de visibilidad podemos guardar la capa haciendo click en el menú capa>Exportar>guardar Como.
13. Una vez guardado el mapa de visibilidad abrimos el global mapper y cargarnos la capa creada. Y le damos que si es un DEM.



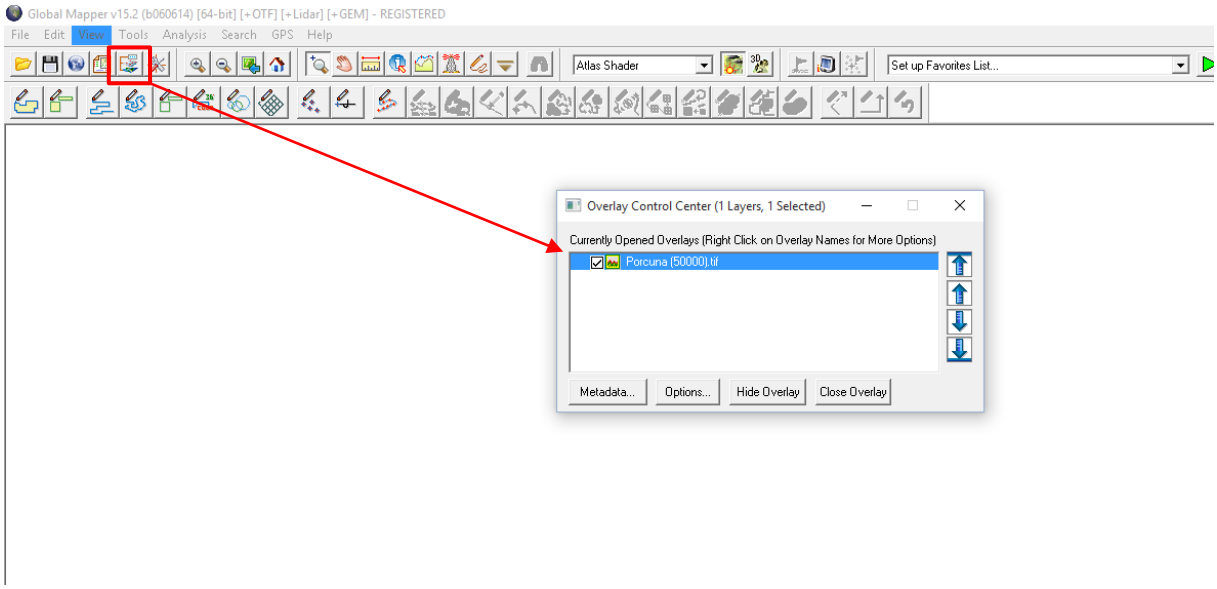

14. Introducimos la proyección del mapa, (EPGS 23030).
15. Nos creamos un “Blend Mode” para ello nos vamos a configuración>shader options. Y en la ventana hacemos cick en Custom Shaders en new.


Configuration
General
Vector Display
Area Styles
Line Styles
Point Styles
Vertical Options
Shader Options
Lidar
Projection
Daylight Shader
Surface Color...
Gradient Shader
Low Color...
High Color...
Slope Shader
Minimum Slope
Slope Value...
Color...
Maximum Slope
Slope Value...
Color...
Coloring Between Min and Max Slope Values
Smooth Gradient
Custom Color
Select...
Slope Direction Shader
West Color...
North Color...
East Color...
South Color...
ADVANCED: Offset Angle in Degrees
0
HSV Shader
Low Color Start:
Value:
Saturation:
Range:
Custom Shaders
Prueba
New...
Edit...
Delete
Reverse Colors of Selected Shader
OK
Cancel
Apply
Help
Set up Favorites List...

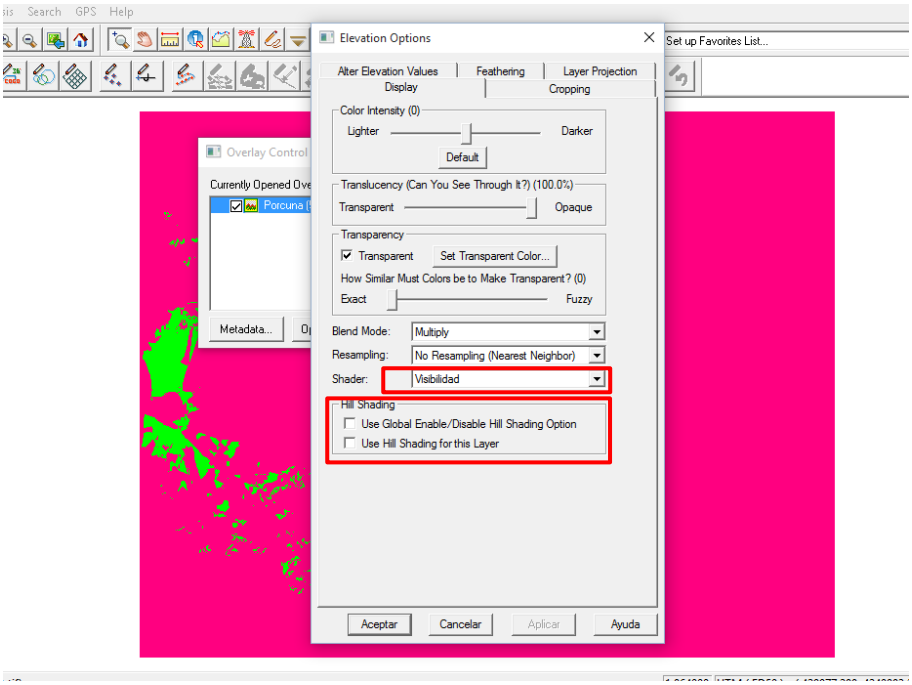
16. Añadimos las alturas de nuestro mapa (en este caso solo 0 y 1). Recordemos que 0 es la parte que no es visible y 1 la que si. En la parte de abajo deseleccionamos el checkBox de blend colors Between Elevation/Slope values.



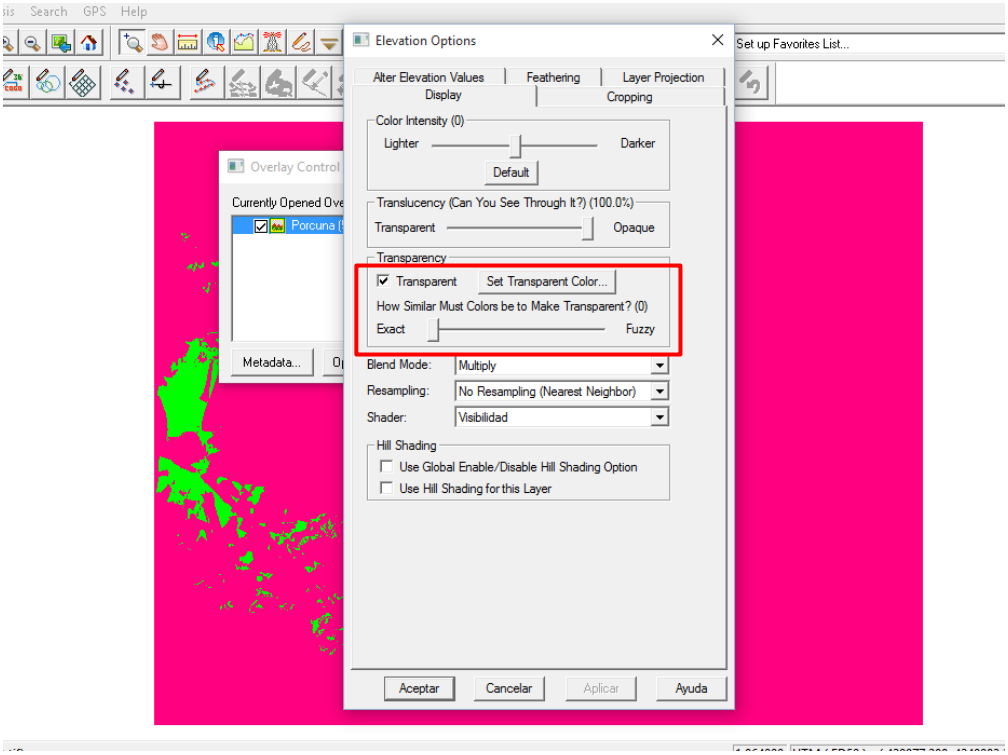

17. Ahora abrimos la configuración de la capa para ello pinchamos abrimos el Control centrer con este icono. 



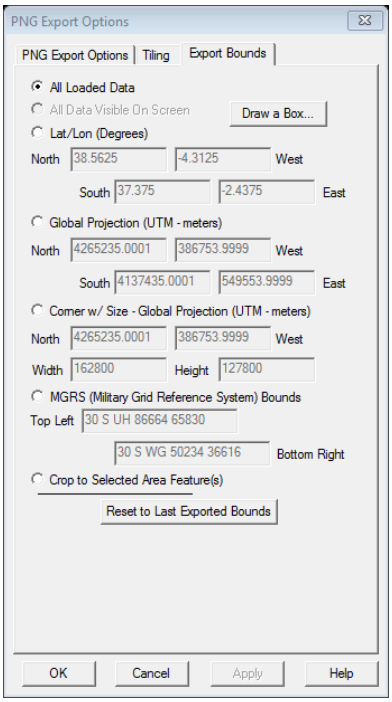
18. Hacemos doble click sobre la capa y abrimos el menú de opciones de la misma. Dentro de la pestaña de display seleccionamos shader que acabamos de crear y deseleccionamos las opciones de Hill Shading.

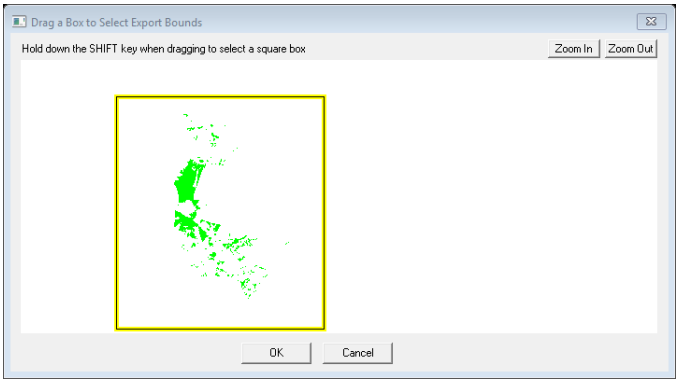



19. Una vez hecho esto vamos a seleccionar un el color asignado a la altura 0 como transparente. Para ello en la misma venta de opciones de capa seleccionamos set Transparent Color y colocamos el slider en Exact. Seleccionado el color le damos aceptar.



20. Finalmente exportamos la imagen a PNG. Para ello hacemos click sobre file>Export>Expor Raster/Image Format… Seleccionamos el tipo PNG
21. Y como solo queremos un trozo de la imagen seleccionamos la pestaña Export Bounds y le damos al botón Draw a Box. Ajustamos la caja lo mejor que se pueda a nuestro mapa. Y cuando lo tengamos le damos aceptar.



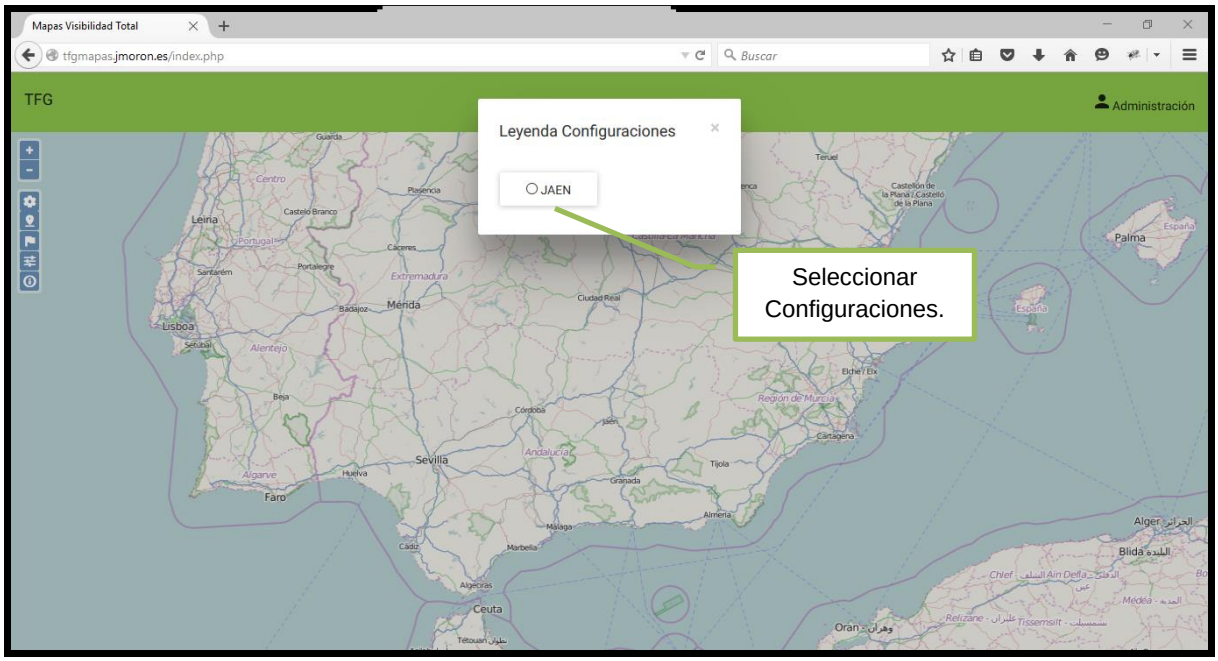




22. Y Finalmente le damos a OK y guardamos nuestro Mapa donde queramos.

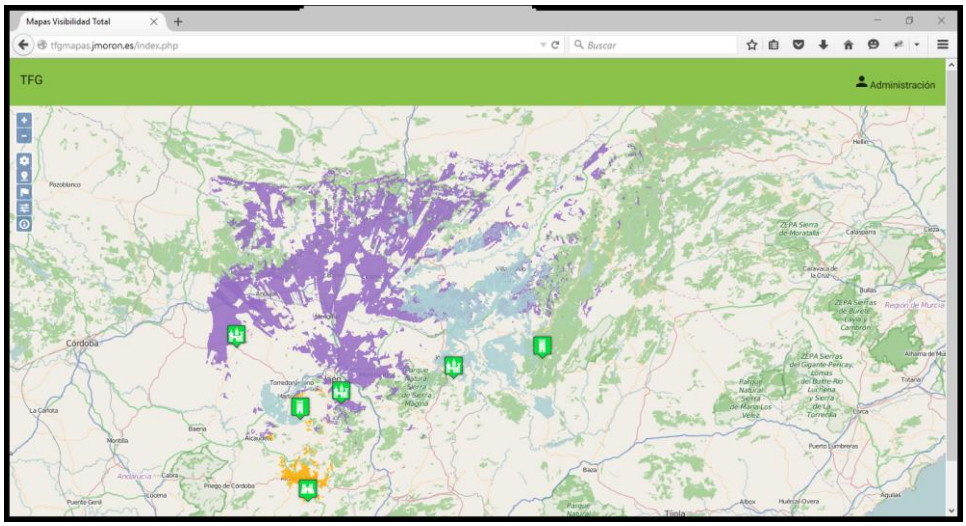
# Anexo II. Manual de usuario.

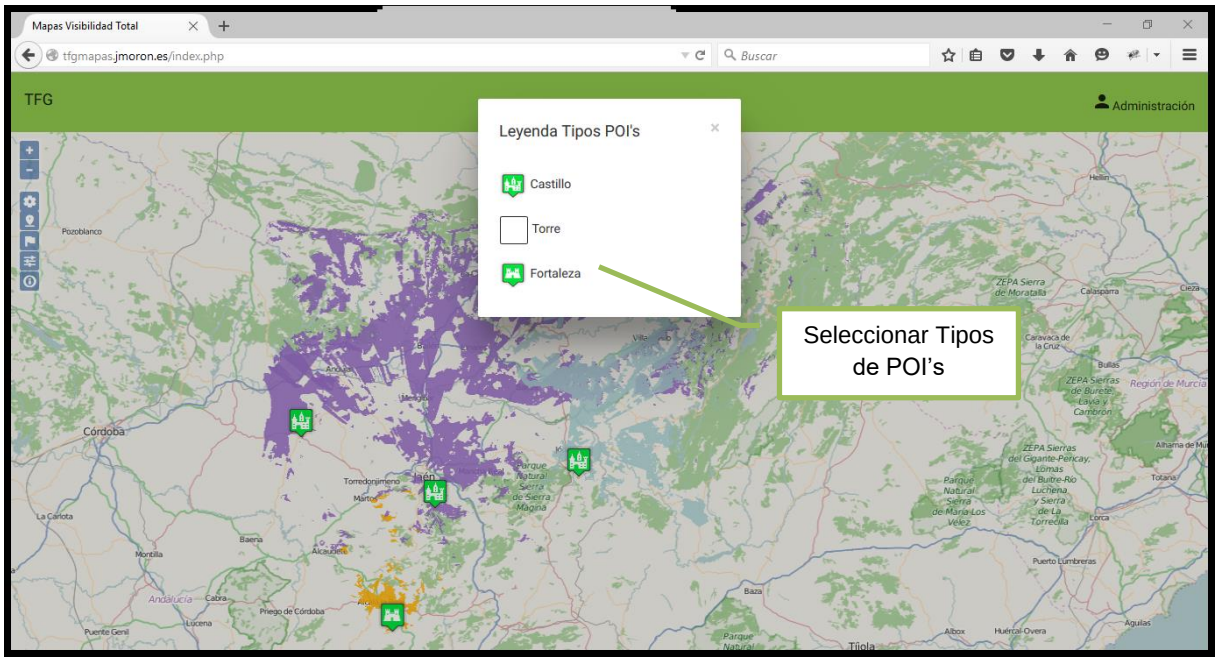
## Usuario Normal

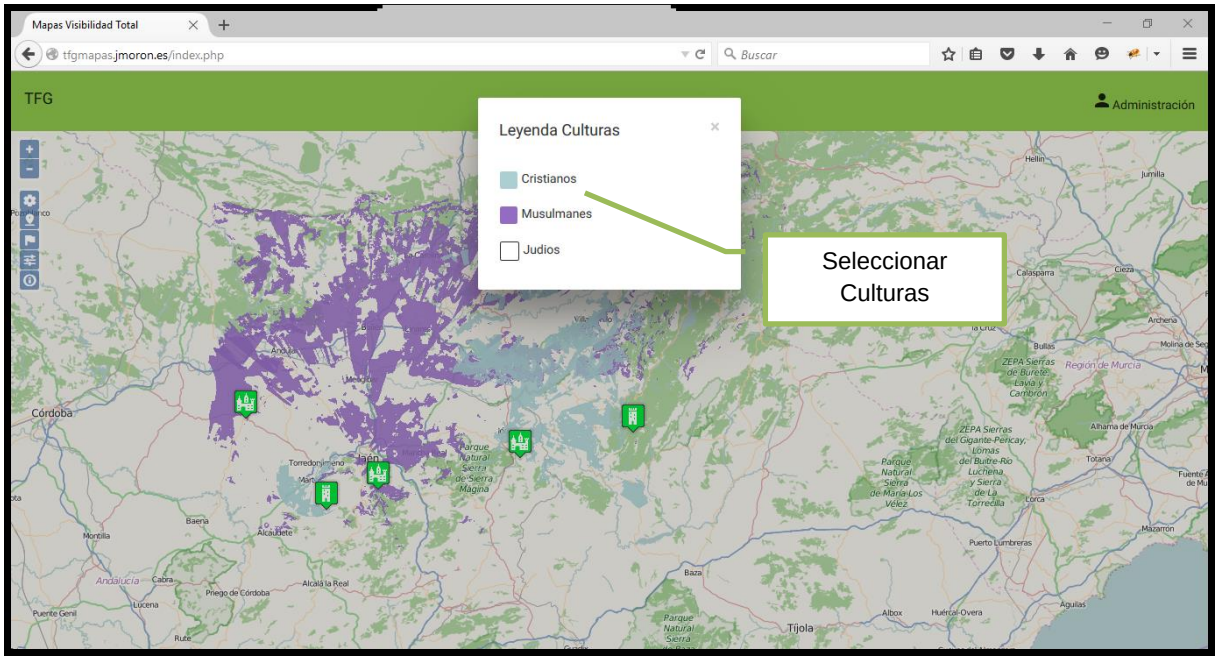
Un usuario normal solo puede ver los mapas de visibivildad subidos por el administrador. Para ver los mapas de visibilidad deberá elegir una configuración primero, para ello pulsamos el botón  del mapa. Aparecera una ventana para seleccionar las configuraciones posibles.

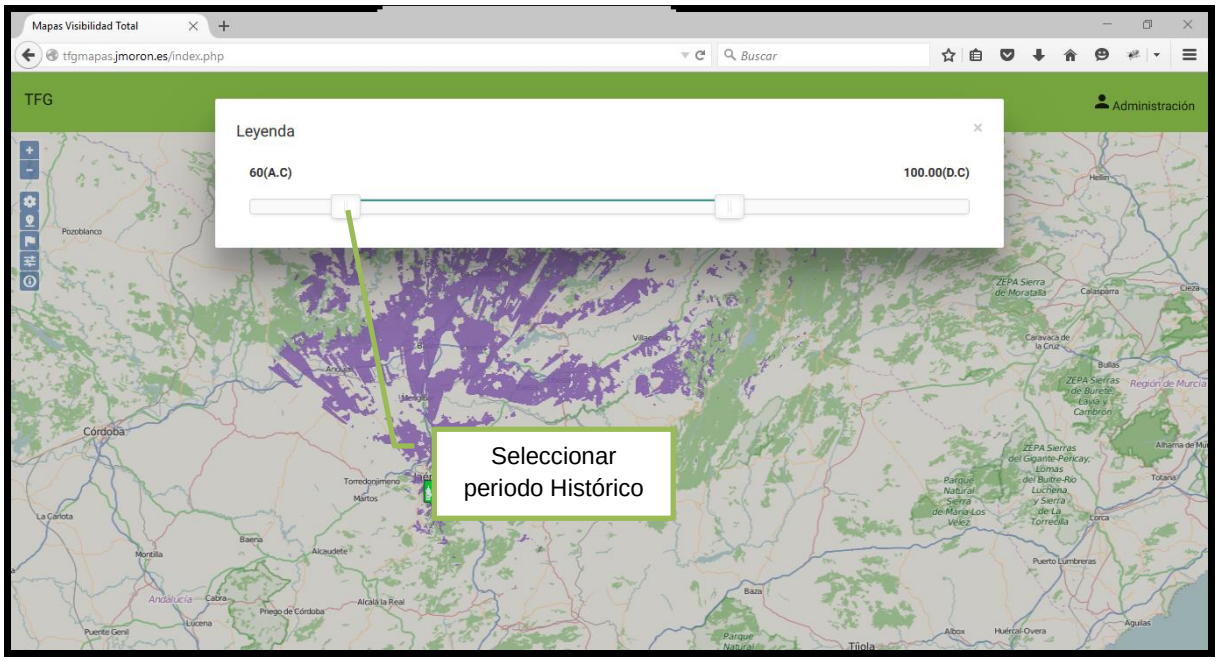



Una vez seleccinado la configuración deseada aparecerán todos los mapas asignados a dicha configuración. El usuario podrá filtrar dichos mapas por Culturas, Tipo de POI's y un periodo histórico.

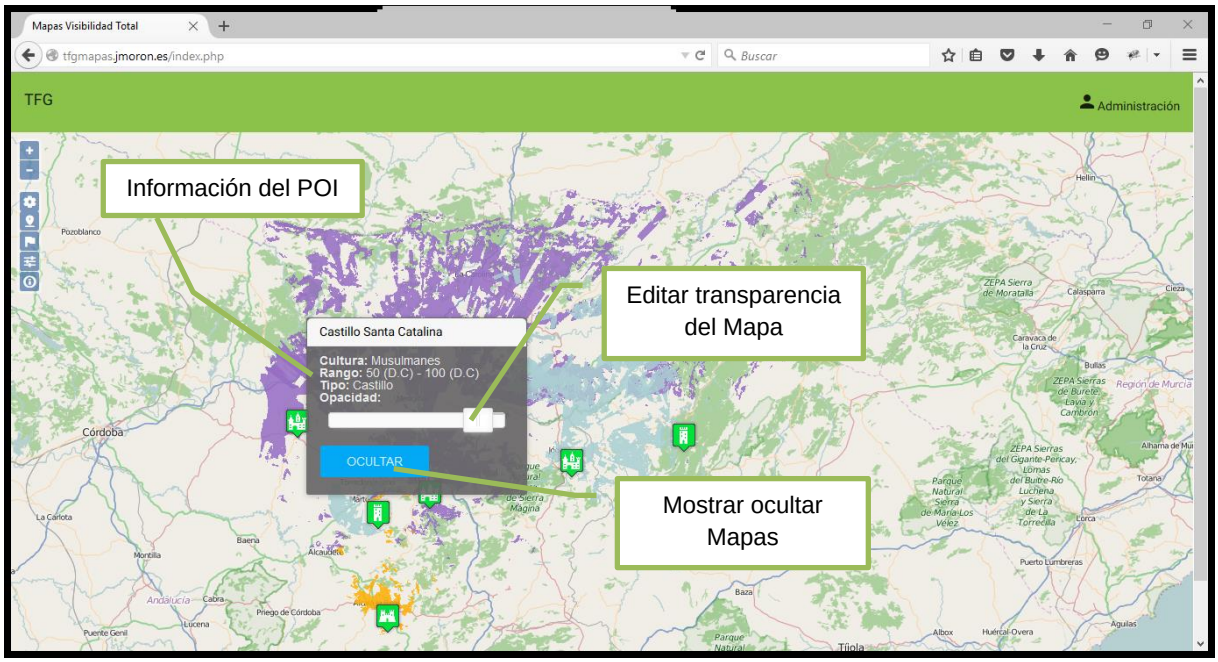



Mapas Visibilidad Total
tfgmapas.jmoron.es/index.php
Buscar
TFG
Administración
Leyenda Tipos POI's
Castillo
Torre
Fortaleza
Seleccionar Tipos de POI's
Córdoba
Jaén


Mapas Visibilidad Total
tfgmapas.jmoron.es/index.php
Buscar
TFG
Administración
Leyenda Culturas
Cristianos
Musulmanes
Judios
Seleccionar Culturas
Córdoba
Jaén



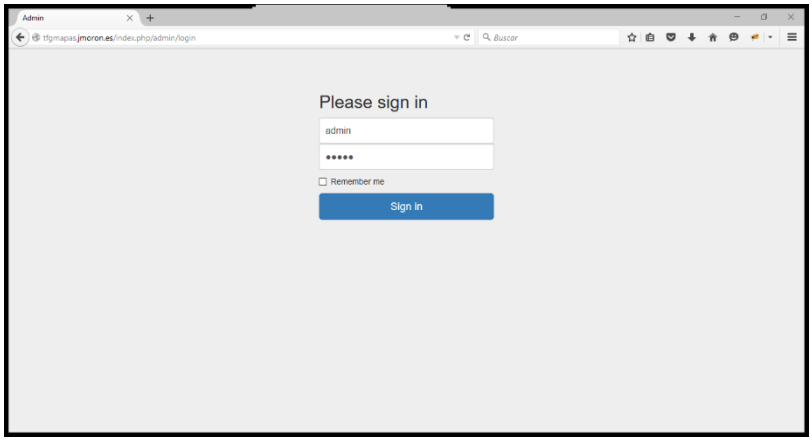

El usuario también pude mostrar ocultar los mapas a gusto suyo, como también se puede modificar la transparencia del mapa.



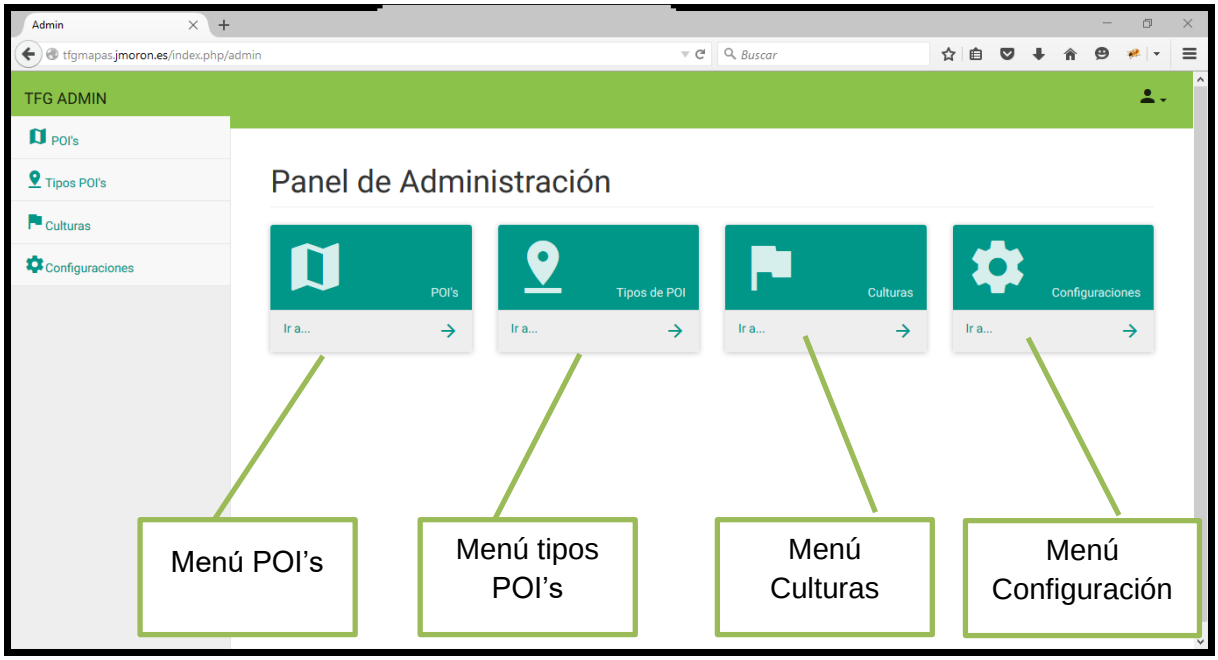
## Usuario Administrador.

El adminisrador es el responsable de crear las configuraciones, las distintas culturas, los tipos de POI's y de los POI.

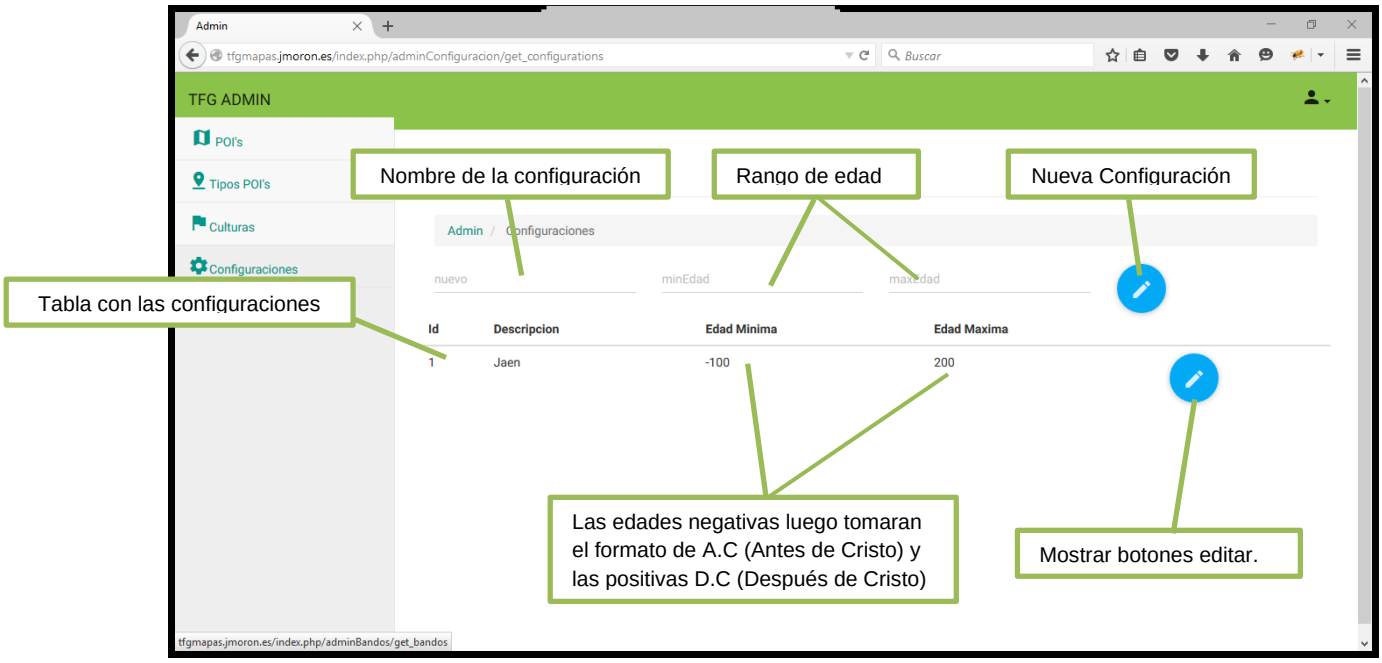
Primero hay que hacer login para ello el usuario y contraseña por defecto que tiene la aplicación es admin admin, se aconseja cambiarla una vez accedido la primera vez.

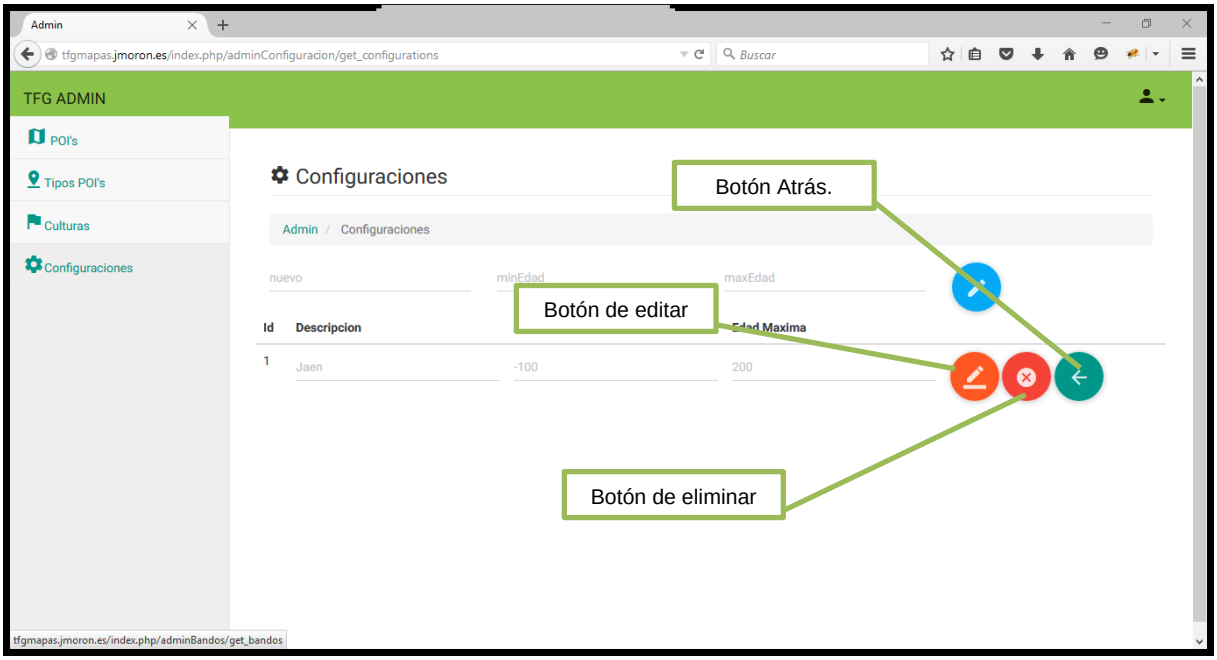



Una vez logeados nos saldrá la pantalla principal de administración en la cual podemos ver todos los menús.

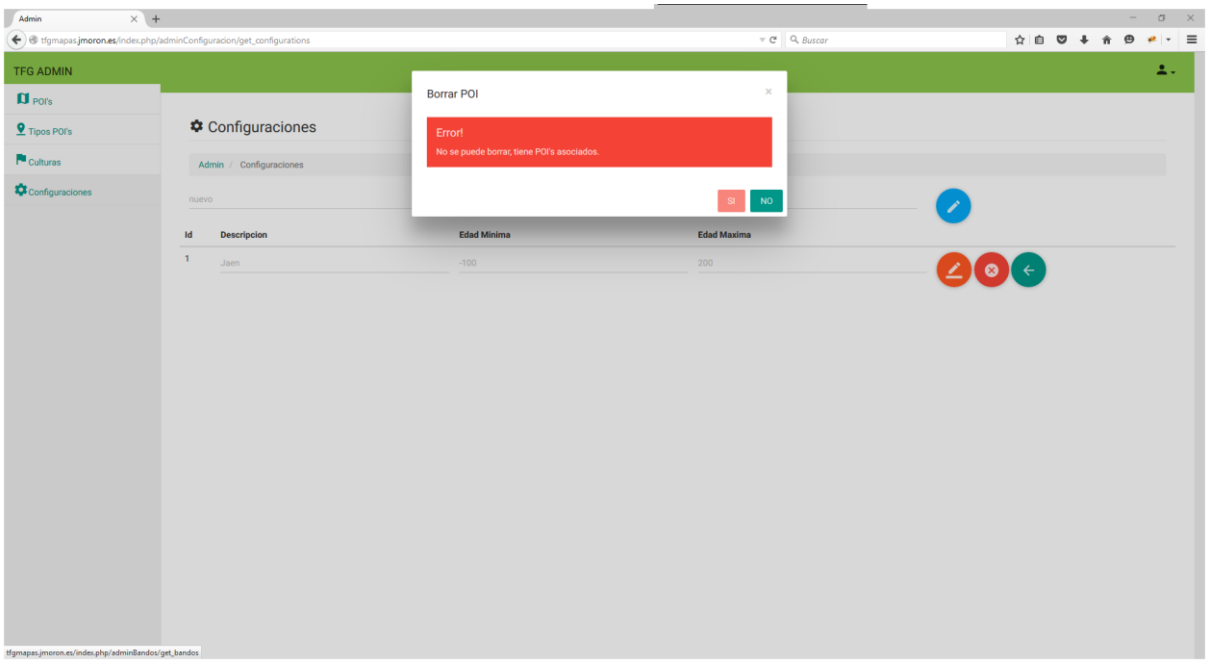


Primero vamos a ver el menú de Configuración

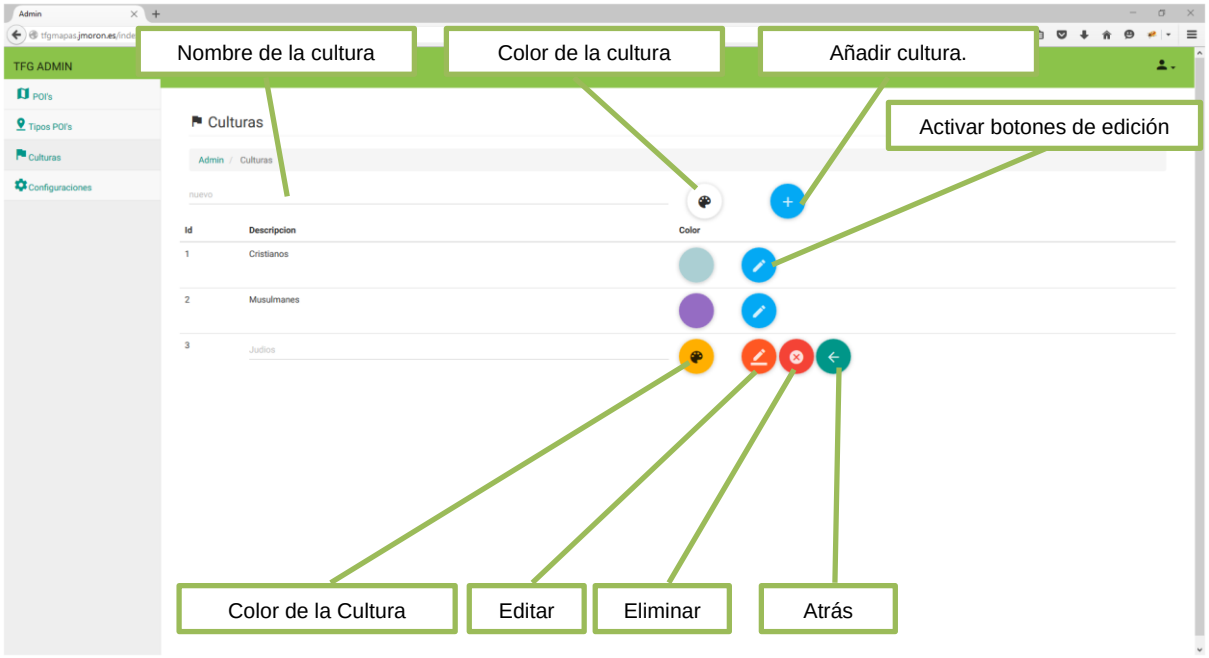


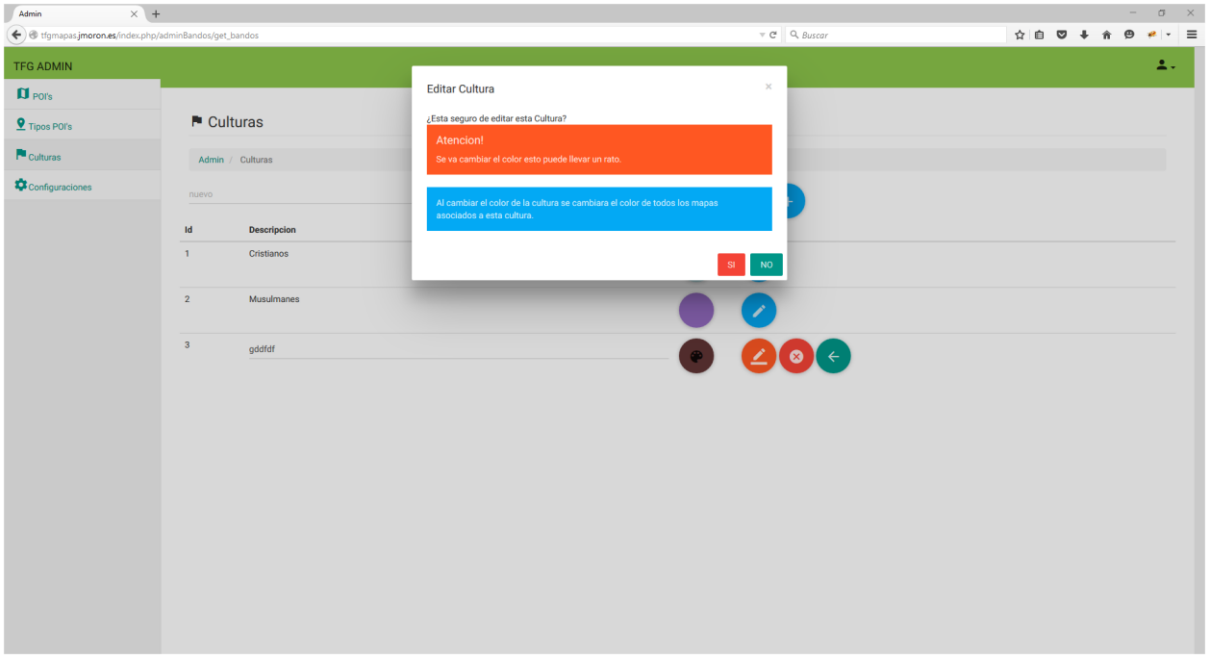

No se puede borrar una configuración, Cultura, Tipo de POI si tiene algún POI asociado.




Ahora vamos a ver el menú de Culturas. Es muy parecido al de configuraciones pero solo que los campos son distintos

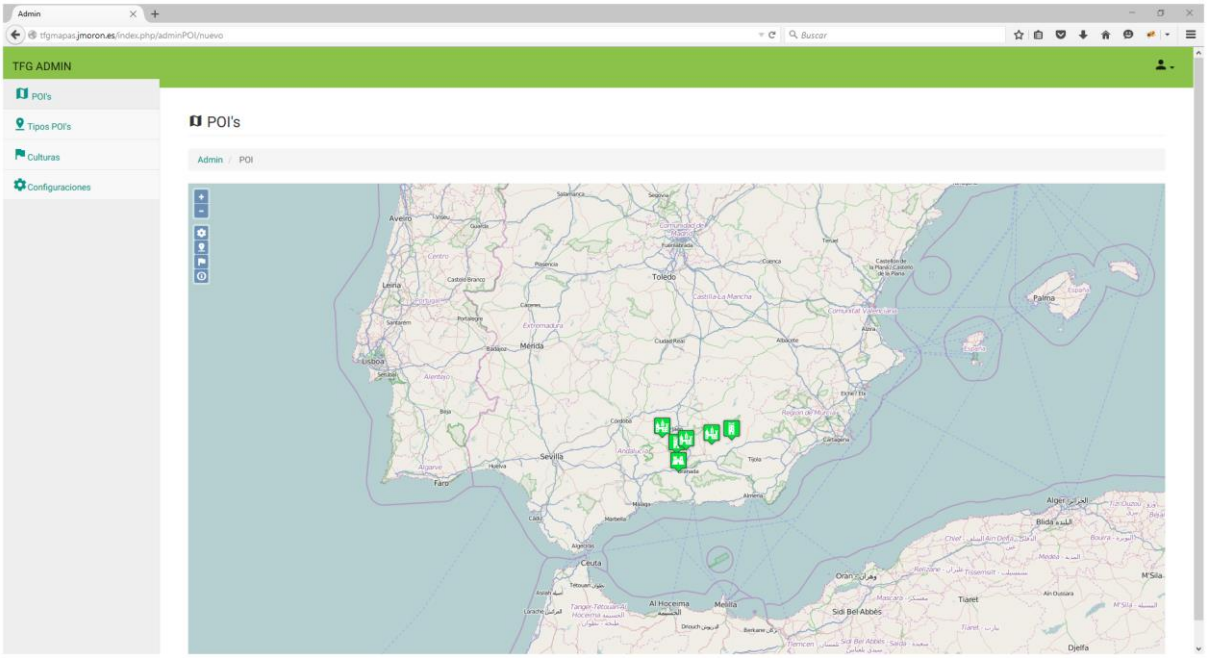



Al cambiar el color de la cultura los mapas asociados también cambiaran de color. Debido a que esto es un proceso el cual puede tardar el botón de Mostrar botones de edición se desactivara hasta que termine el proceso en el servidor.

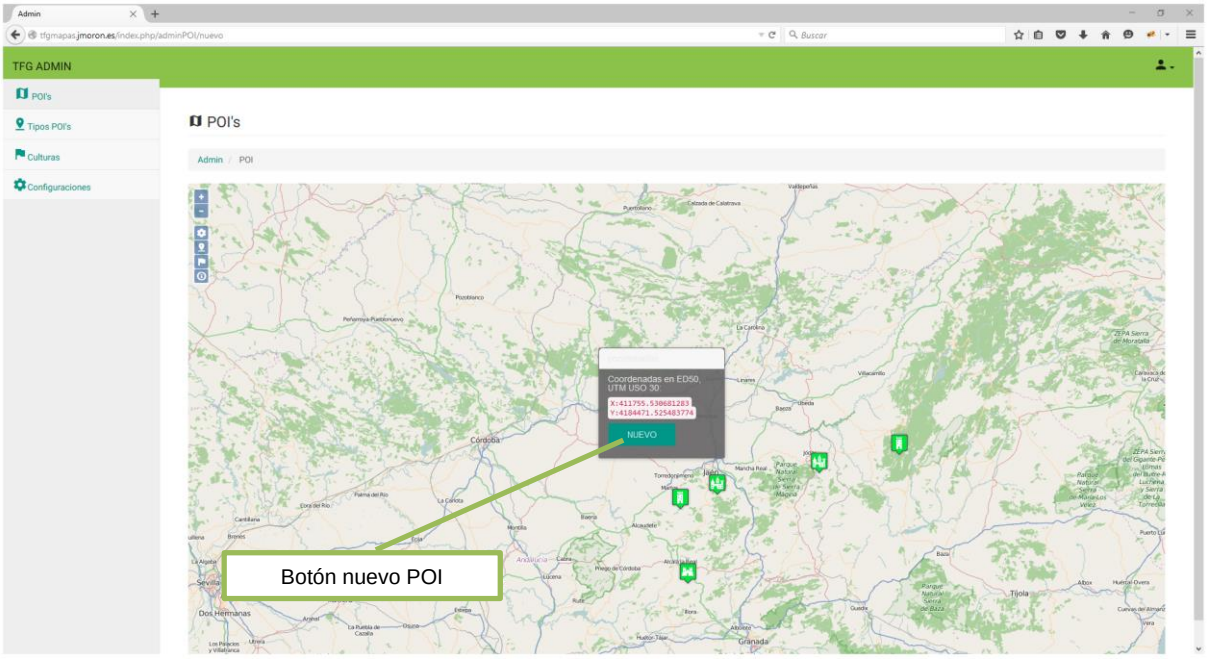


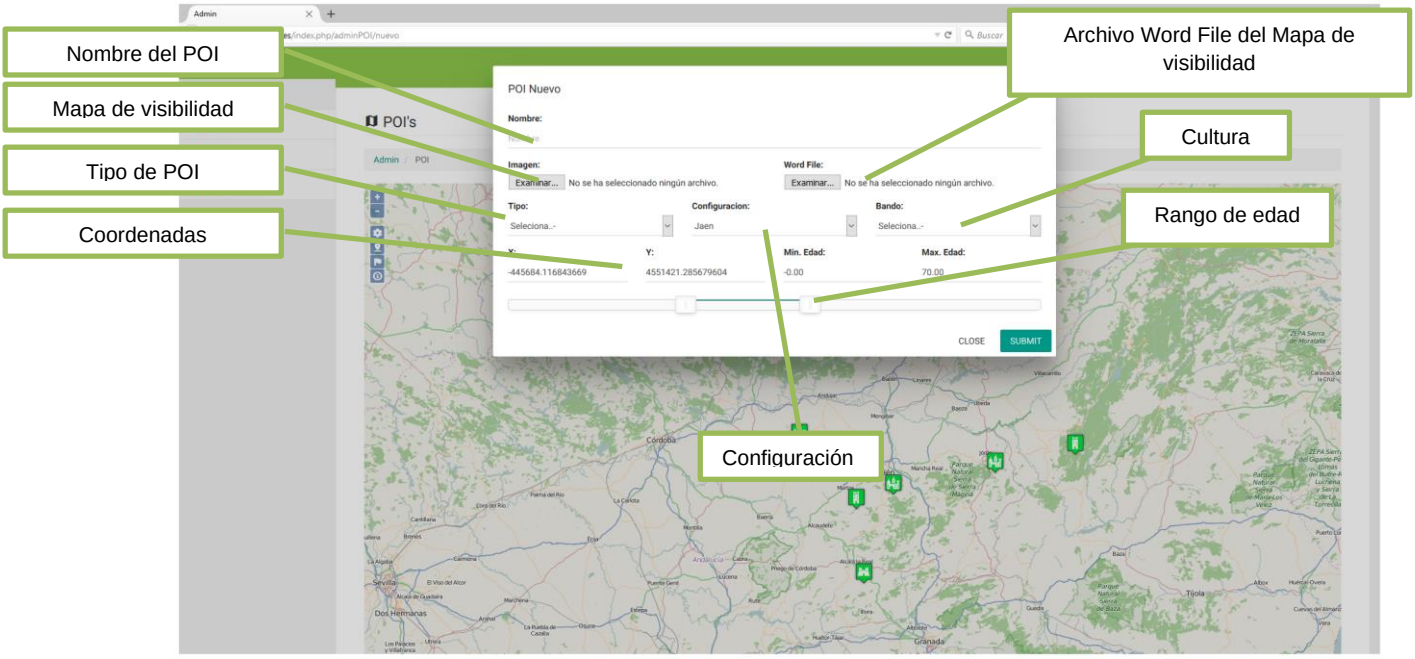

Otro menú es el de tipo de POI's, este menú se puede configurar los distintos tipos, darle un nombre y asignarle un icono. El modo de utilizarlo sigue siendo el mismo que hemos visto en los anteriores menús.

El último de los menús es del POI's este menú es totalmente distinto a los demás. En este menú se puede ver un mapa y dentro del mapa se pueden ver los POI's insertados en una configuración determinada.

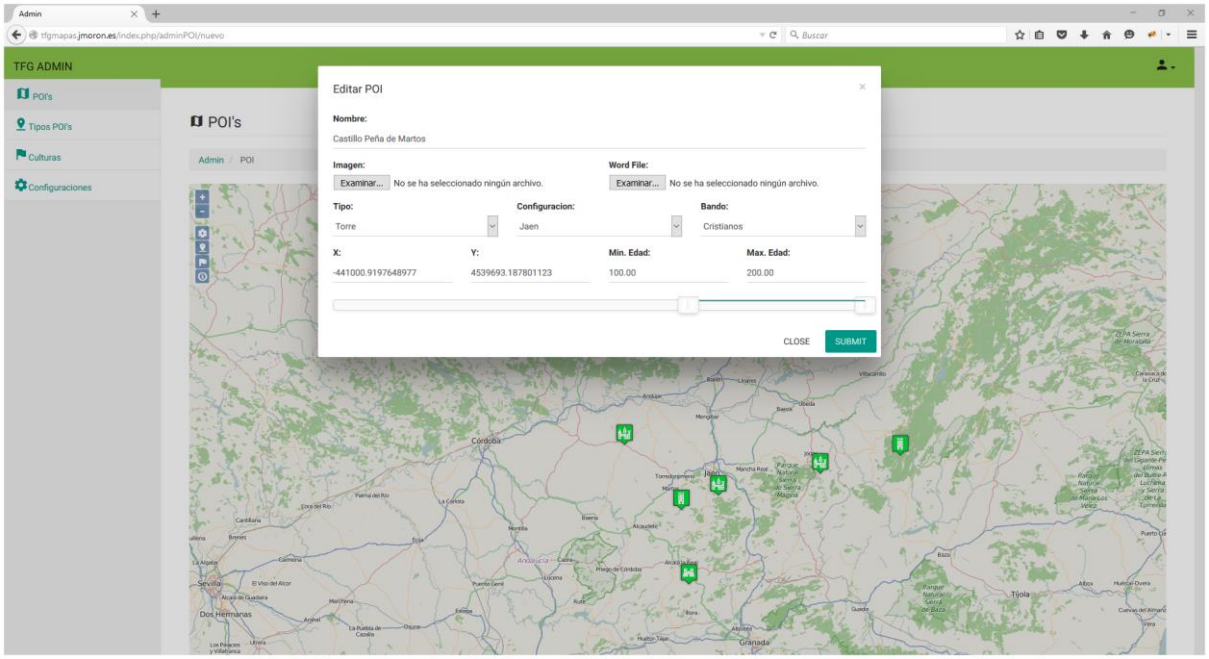



Para insertar un nuevo POI hay que hacer click sobre un lugar vacio en el mapa y darle a nuevo. Y rellenamos el formulario correctamente.

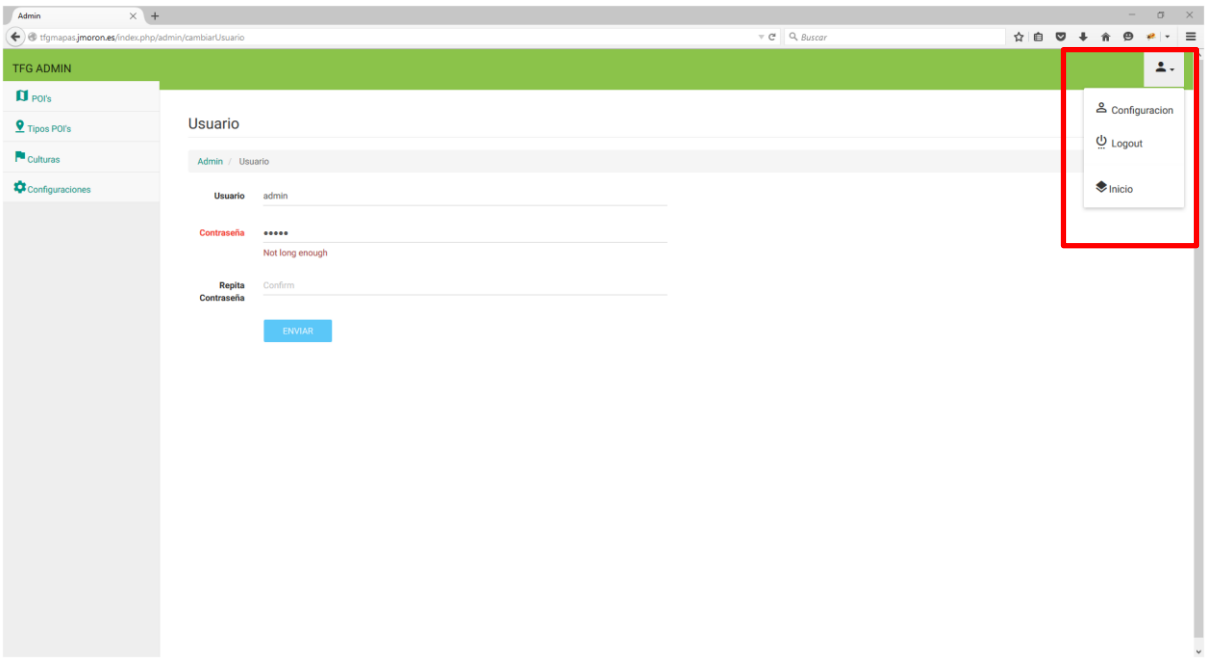





Si hacemos click sobre el icono de un poi nos saldrá un popover dándonos la opción de editar el poi o eliminarlo. Si escogemos editar nos saldrá el formulario con la información del POI a editar.



Para cambiar la contraseña del usuario o el nombre hacemos click en el icono de la parte superior derecha y seleccionamos Configuración. En ese menú aparecen también las opciones de salir de la administración cerrando la sesión o de volver a la pantalla de inicio.



# Anexo III. Instalación de la aplicación.

Veremos a continuación algunos detalles sobre el proceso de instalación y configuración. Afortunadamente para las personas con menos experiencia, los pasos son bien simples.

Requisitos de servidor

La versión 2 de CodeIgniter es únicamente compatible con PHP 5. En concreto necesitarás PHP 5.1.6 o superior.

Por lo que respecta a las bases de datos, CodeIgniter es compatible con unas cuantas, las más habituales en el desarrollo de webs: MySQL (4.1 o posterior), MySQLi, MS SQL, Postgres, Oracle, SQLite.

Instalación

1.- Sube la aplicación a tu servidor

Ahora tienes que subir la carpeta CI_TFG a tu servidor web.

2.-Importar base de datos

En este paso hay que importar el archivo SQL que está en TFG_CI\assets\BBDD a la base de datos para crear todas las tablas.

3.- Configurar la base de datos

En este último paso tendrás que indicar los datos de acceso a la base de datos que piensas utilizar. Para ello tenemos que editar el archivo system/TFG_MAPAS/config/database.php e indicar los parámetros de conexión al servidor de base de datos, como el nombre del servidor y nombre de la base de datos, el usuario y la contraseña.

Con esto ya tenemos todo listo para usar nuestra aplicación.