
Trabajo Final de Máster

# Open Atlas del Sistema de Sanidad Pública de Cataluña

Ingeniería Informática

Curso 2014-2015, 2º Semestre

*Alumno:*
Víctor Soler

*Dirigido por:*
Anna Muñoz Bollas

11 de junio de 2015

# FICHA DEL TRABAJO FINAL

| | |
|---|---|
| **Título del trabajo:** | Open Atlas del Sistema de Sanidad Pública de Cataluña |
| **Nombre del trabajo:** | Víctor Soler Viñuelas |
| **Nombre del consultor:** | Anna Muñoz Bollas |
| **Fecha de Entrega (mm/aaaa):** | 06/2015 |
| **Área del Trabajo Final:** | Sistemas de Información Geográfica |
| **Titulación:** | Máster en Ingeniería Informática |

**Resumen del Trabajo (máximo 250 palabras):**

El uso de los Sistemas de Información Geográfica han cobrado gran importancia en los últimos años en el ámbito de ánalisis de datos, puesto que son capaces de mejorar la interpretación de los datos, aportando una componente geográfica que en algunos casos podría explicar las causas de los resultados.
Con esta base y teniendo en cuenta la importancia que los SIG desempeñan en el ámbito sanitario pues son capaces de identificar zonas de riesgo o justificar la inversión del gasto sanitario, este proyecto pretende aplicar los SIG a indicadores sanitarios para crear un producto basado en aplicaciones *open source* que sitúe sobre el mapa los indicadores sanitarios que la *Generalitat de Catalunya* publica de manera abierta.

**Abstract (in English, 250 words or less):**

The use of Geographic Information Systems have become very important in recent years in the scope of data analysis since they are capable of enhancing the interpretation of data by providing a geographic component that in some cases could explain the causes of the results.
With this foundation and taking into account the importance that GIS play in the health sector because they are able to identify areas of risk or investment justify health spending, this project aims to apply GIS to health indicators to create a product based on open source applications which places on the map the health indicators openly published by *Generalitat de Catalunya*.

**Palabras clave (entre 4 y 8):**

*Webmapping*, *choroplet*, *Sistemas de Información Geográfica (GIS)*, *Servidor de mapas*, *GeoSpatial*, *OpenSource*, *Javascript*

# Índice general

# Índice de figuras

# Capítulo 1

# Introducción

## 1.1. Contexto y justificación del trabajo

Periódicamente la Generalitat de Catalunya publica una serie datos del SISCAT (*Sistema Sanitari Integral dUtilització Pública de Catalunya*) que proporcionan información de problemas de salud en la población y su evolución en el tiempo con la finalidad de poder valorar su tendencia y su distribución geográfica: son los indicadores de salud. Con ello, se pueden evidenciar y generar tendencias de la situación sanitaria de un conjunto poblacional lo que permite por una parte, hacer una vigilancia de la situación sanitaria y por otra parte identificar posibles desigualdades de salud en el conjunto de la población y subconjuntos de población.

Así pues, este proyecto consiste en la implementación de un servidor de mapas y un visor web de mapas para mostrar a través de un portal los indicadores de salud publicados por la Generalitat de Catalunya con el objetivo de mostrarlos en un contexto geográfico de manera que el usuario pueda interactuar de manera sencilla tanto con los mapas como con los indicadores de salud.

## 1.2. Objetivos del trabajo

Los principales objetivos del presente documento es la identificación de las diferentes tareas que abarca el proyecto y su estimación de tiempo mediante la elaboración de un cronograma de actividades.

### 1.2.1. Objetivos generales del trabajo

Los objetivos generales que se espera cumplir con la finalización del proyecto son:

- Diseño e implementación de un servidor de mapa.
- Diseño e implementación de un visor GIS web.
- Integración de datos ráster y vectoriales procedentes de diferentes fuentes.

### 1.2.2. Objetivos específicos del trabajo

Los objetivos específicos del trabajo son aquellos en los que se profundiza en mayor manera el esfuerzo del trabajo:

- Conocer a fondo alguno de los servidores de mapas open source.
- Analizar las diferentes librerías JavaScript que permiten desarrollar un visor web con componente geográfica.

- Visualizar de manera dinámica información estadística de los datos.
- Seleccionar de manera interactiva las diferentes capas de salud que se quieran visualizar.

## 1.3. Planificación del Trabajo

El proyecto se ha dividido en varios grupos de actividades que finalmente han sido agrupadas por PECs para conseguir una planificación más práctica marcada por las fechas clave del semestre (las propias de las PEC):

1. Fase de planificación (PEC1) (*e*quivalente a la PEC 1)
2. PEC2: Fase de investigación y Preparación del entorno de trabajo
3. PEC3: Fase de desarrollo
4. Fase final (PEC4)
5. Fase de seguimiento y control

### 1.3.1. Fase de planificación (PEC1)

En este caso, la fase se corresponde con la primera PEC. El objetivo de esta fase (y por tanto también de la PEC) es identificar las actividades necesarias y su planificación temporal para finalizar el proyecto de forma exitosa. La finalización de esta fase concluye con la entrega de la PEC. Aunque la PEC será entregada, se entiende que la planificación inicial puede sufrir variaciones debido a posibles contratiempos. El número de páginas previstas para la PEC es de 17 páginas.

### 1.3.2. PEC2: Fase de investigación y Preparación del entorno de trabajo

Esta fase comprende las diferentes tareas de análisis y de recopilación de información de las tecnologías a utilizar y la información a tratar. En esta fase se determina el alcance y las especificaciones técnicas del proyecto. Además, recoge la fase de preparación del entorno de trabajo en la que se instalará y configurará la estación de trabajo para realizar correctamente el TFM.

### 1.3.3. PEC3: Fase de desarrollo

Comprende las actividades de diseño y desarrollo del software. Se ha incluido también en esta fase el tratamiento e inserción de datos cartográficos y alfanuméricos.

### 1.3.4. Fase final (PEC4)

Contempla las actividades relativas a la documentación y cierre de proyecto que incluye como elemento principal la finalización de la memoria del proyecto y la elaboración de la presentación virtual. Como la memoria se desarrolla durante todo el semestre, esta fase se inicia con el nacimiento propio del TFM. Además, incluye como hitos las entregas previas del avance de la memoria que se van a ir realizando durante el avance del semestre.

### 1.3.5. Fase de seguimiento y control

Esta actividad se realizará a lo largo de todo el proyecto e implica, como principales tareas, controlar y verificar que se sigue el plan inicial y, en caso de no poder cumplirlo, tomar las medidas correctoras oportunas modificando el plan.

# Capítulo 2

# Hitos y entregables

Los principales hitos y entregables a tener en cuenta son (ordenados por fecha):

| Hitos o Entregables | Fecha |
|---|---|
| Inicio de proyecto | 03/03/2015 |
| PEC 1: Entrega parcial | 08/03/2015 |
| PEC 1: Plan de trabajo | 10/03/2015 |
| PEC 2: Entrega | 07/04/2015 |
| Memoria: Entrega parcial | 04/05/2015 |
| PEC 3: Entrega | 12/05/2015 |
| Producto operativo 100 % | 12/05/2015 |
| Memoria: Entrega parcial | 01/06/2015 |
| Presentación virtual: Entrega parcial | 05/06/2015 |
| PEC 4 | 08/06/2015 |
| Presentación virtual | 08/06/2015 |
| Memoria | 08/06/2015 |
| Código fuente | |

# Capítulo 3

# Análisis de riesgos

Con el análisis de riesgos se pretende identificar aquellas vulnerabilidades que implican cualquier tipo de impacto negativo en el resultado final o parcial (relativo a cada entrega o PEC) del proyecto.

Una falta de análisis de riesgos o una incorrecta identificación puede provocar que el proyecto no culmine de forma satisfactoria.

A continuación se detallan los principales riesgos que, a priori, se identifican para este proyecto, su probabilidad materializarse, su nivel de impacto sobre el proyecto y las acciones.

## 3.1. Evaluación de riesgos

| Código | Riesgo | Descripción | Probabilidad | Impacto |
|---|---|---|---|---|
| **R01** | Disponibilidad de tiempo | Por motivos laborales en los que no se puede absorber picos de trabajo o hay que dar solución a algún cliente, no dispondré del tiempo libre necesario y planificado para el trabajo. | Muy Alta | Muy Alto |
| **R02** | Dificultades técnicas con servidor de mapas o librería Javascript | Hasta ahora he trabajado con el API JavaScript de *Google Maps* y *Google Maps Engine* trabajando de servidor de mapas. También también he cargado WMSs, archivos GeoJSON, KMLs o respuestas XML de algún servlet propio accediendo a bases de datos *Postgis* sobre *Google Maps*. Pero montar un servidor de mapas es algo totalmente nuevo y como tal puede presentar dificultades. | Baja | Baja |

| Código | Riesgo | Descripción | Probabilidad | Impacto |
|---|---|---|---|---|
| **R03** | Problemas de rendimiento con el ordenador | El ordenador tiene unas prestaciones físicas algo escasas. Puede repercutir en lentitud o imposibilidad de trabajo. | Medio | Baja |
| **R04** | Selección de tecnologías | Puede darse el caso de seleccionar una tecnología que no sea la más adecuada para acometer el proyecto o que sea de difícil implementación (por ejemplo, por falta de experiencia). | Medio | Media |
| **R05** | Baja por enfermedad | Siempre es una posibilidad sufrir alguna enfermedad en el desarrollo del proyecto que impida continuar el desarrollo del trabajo de manera temporal. | Medio | Media |

## 3.2. Plan de contingencia

Detalle de las acciones que corrigen los riesgos en caso de producirse:

| Código | Riesgo asociado | Descripción | Riesgo residual |
|---|---|---|---|
| **MR01** | **R01** y **R05** | Al no disponer de más tiempo, se reducirán las horas de descanso lo que podría provocar de manera indirecta un descenso en una leve disminución en la calidad del TFM. | Medio |
| **MR02** | **R01** y **R05** | Llegado el caso, si no se dispone de más tiempo habrá que considerar disminuir la calidad de los entregables finales (PECs, memoria, video o aplicación) o reducir las funcionalidades a implementar en la aplicación. | Alto |
| **MR03** | **R01** | En caso de no disponer de más tiempo libre debido a motivos laborales y cuando **MR01** no sea posible, habrá que replantear objetivos del TFM lo que implicará de manera directa una disminución en la calidad del TFM. | Alto |
| **MR04** | **R02** | En caso de que no se pueda obtener éxito con el servidor de mapas, se contemplará la posibilidad de instalar otro servidor de mapas y, en caso extremo, utilizar otro sistema de visualización de mapas. | Bajo |
| **MR05** | **R03** | Se puede tener disponibilidad temporal de un ordenador con mejores características de Hardware. | Muy bajo |
| **MR06** | **R04** | Si la tecnología adecuada no es la más adecuada o presenta grandes dificultades, se volverá a planificar el proyecto considerando el cambio de tecnologías y si existe la posibilidad de acabar el proyecto de manera satisfactoria sin tener que sacrificar ningún objetivo del proyecto ni fechas de entrega. | Alto |

# Capítulo 4

# Requisitos de material

## 4.1. Requisitos de Software

Aunque todas las aplicaciones y librerías necesarias en el desarrollo del proyecto se sabrá con exactitud tras la fase de investigación de tecnologías (principalmente porque la selección de tecnologías determinará parte de librerías a utilizar), en un primer momento el listado de software necesario está formado por las siguientes aplicaciones:

- **QGIS**: Aplicación de software libre para abrir documentos y fuentes de datos cartográficos.
  Se utilizará como mínimo la versión 1.8.
- **Postgres** y **Postgis**: Sistema gestor de base de datos de open source que permite trabajar con datos espaciales. Su uso para cartografía está muy consolidado lo que hace que sea muy estable en productos tanto en desarrollo como en producción.
- **Máquina Virtual de Java (JDK y SDK)**: Necesaria para la ejecución de contenedores de aplicaciones como **Tomcat** y para ejecutar código **Java**. Pendiente de confirmar el servidor de mapas y las tecnologías a utilizar. La versión utilizada será Java 7.
- **GIMP** Editor de imágenes *open source*.
- **DbVisualizer** Herramienta que permite realizar funciones adicionales sobre una amplia gama de sistemas de bases de datos. Su uso principal será el de desarrollar diagramas de bases de datos.
- **Google Chrome y Firefox** Navegadores web que permiten visualizar páginas web y que resultan muy útiles para la depuración de JavaScript y contenido HTML y CSS.
- **Gantter for Google Drive** Extensión de *Google Chrome* para realizar diagramas de Gant de proyectos.
- **Cacoo** Extensión de Google Chrome para realizar diagramas UML.
- **Microsoft Office ó OpenOffice (o LibreOffice)** La hoja de cálculo puede ser necesaria para visualizer datos (por ejemplo, los indicadores de sanidad) o elaborar gráficas de informes (aunque probablemente no sea necesario en el proyecto).
  El procesador de textos será necesario para la elaboración de documentos (como la memoria final).
- **SVN Server ó GIT** Servidor utilizado como repositorio de control de versiones de proyectos de software. Aunque no es necesario, es muy útil para distribuir copias del código de fuente, sincronización en equipos de trabajo o como almacén de código fuente donde registrar todos los cambios efectuados en el código fuente.

- **LaTex** Sistema para la creación de documentos de calidad tipográfica. Se utilizará para componer la memoria final.
- **Sistema operativo**: se utilizará *Windows 7* y *Ubuntu 12.4* indistintamente.

Las librerías Javascript a utilizar aún no están determinadas.

## 4.2. Requisitos de Hardware

Las máquinas en las que se desarrollará el proyecto son 2 y, aunque el proyecto carece de requisitos de hardware determinados, están compuestas por el siguiente hardware:

- CPU Intel i5 2430 2.40 GHz
- 8 GB de RAM
- 500 GB de disco duro

La otra máquina dispone del siguiente hardware:

- Intel Core 2Duo 8700 2.53 GHz
- 4 GB de RAM
- 150GB de disco duro

Ambas máquinas disponen de conexión a internet mediante una ADSL de 6 MB y 2 discos duros externos (500GB y 200 GB) que se utilizarán como almacenes de copias de seguridad.

# Capítulo 5

# Estado del arte

En este capítulo se pretende detallar el problema a resolver en el presente Trabajo Final de Máster (TFM) así como abordar algunos trabajos similares y describir brevemente la tecnología actual necesaria para llevar a cabo de forma exitosa este TFM.

## 5.1. Descripción del problema

El problema a solucionar en el presente TFM consiste en la implementación de un visor de mapas capaz de mostrar los indicadores de salud publicados por la *Generalitat de Catalunya* de manera abierta en su portal web.

Uno de los requisitos principales del TFM consiste en la implementación del producto final haciendo uso de librerías *open source* y posteriormente detallar en un estudio sus ventajas y sus limitaciones así como los probelmas encontrados durante la fase de desarrollo.

## 5.2. Precedentes y trabajos similares

A continuación se muestran varios ejemplos de aplicaciones que abordan problemas similares al que se presentan en el presente TFM

### 5.2.1. Atlas electrónico de Salud de Catalunya

*Nexus Geogràfics*[1] es una empresa con sede central en Gerona y que cuenta con más de 15 años de experiencia desarrollando su actividad profesional en la creación e implantación de software y proyectos de información geográfica como es el caso de la herramienta *Atlas electrónico de Salud de Catalunya*.

Dicha herramienta es una aplicación desarrollada en el año 2006 a petición del *Departament de Salut* de la *Generalitat de Catalunya* con la intención de que los profesionales de la salud puedan realizar por ellos mismos consultas cartográficas basadas criterios sanitarios a la par de que profesionales en cartografía puedan actualizar los datos facilmente. Además, el *Atlas* utiliza en parte cartografía disponible en el **Hipermapa** del *Departament de Territori i Sostenibilitat* de la *Generalitat de Catalunya* [2]

Al tratarse de un proyecto destinado a los profesionales sanitarios, no se puede probar para ver en qué consiste exactamente (orígenes de datos, calidad de las imágenes, etc.) pero existe un documento proporcionado por la *Generalitat* [7] del que se pueden extraer algunas conclusiones sobre la aplicación.

Tal y como se muestra en la figura 5.1 La estructura planteada en la aplicación es modular para que puedan intervenir diferentes perfiles de usuarios de manera que existen

[1] http://www.nexusgeografics.com

[2] http://hipermapa.ptop.gencat.cat/hipermapa/client/151208/base_high_cat.html

publicadores cartográficos, publicadores de indicadores, usuarios a través del visor web y usuarios de visor móvil (llama la atención que el dispositivo móvil al que hacen referencia sean PDAs pero hay que tener en cuenta que en aquellos años aún no existían dispositivos *smartphones* ni *tablets* como los que conocemos hoy en día).

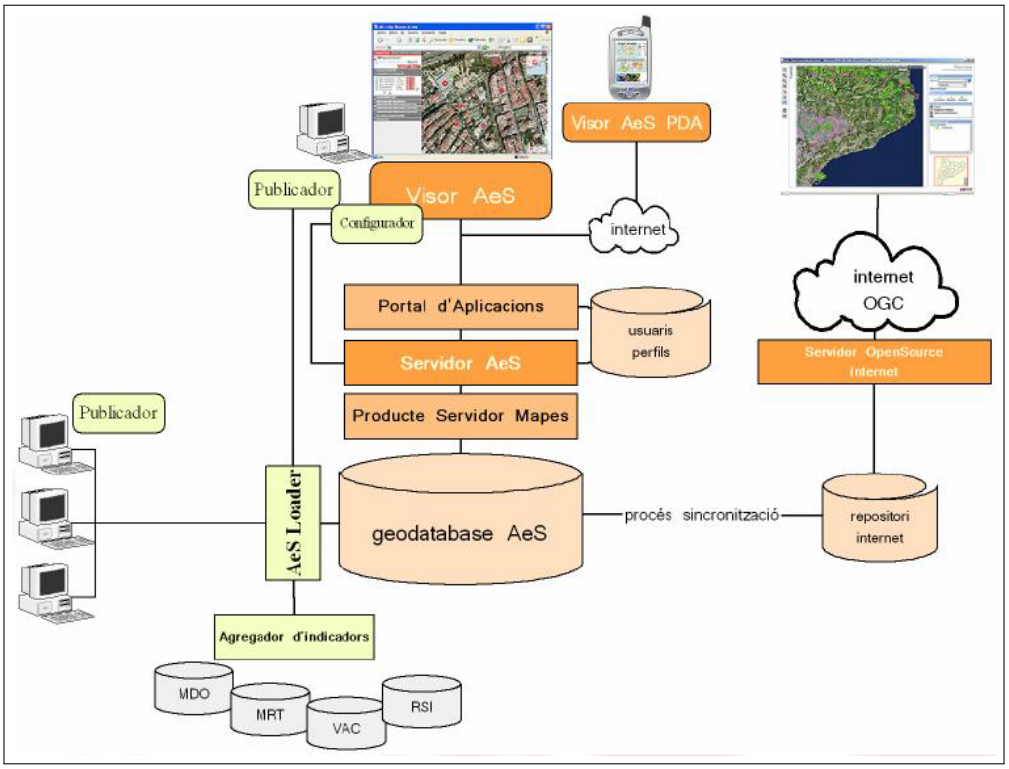


**Figura 5.1**: Estructura del Atlas Electrónico de Salud de Cataluña
Fuente: Sistema de Información Geográfica del Departamento de Salud de la Generalitat de Cataluña

### 5.2.2. New Zealand Health Quality & Safety Comission Atlas of Healthcare Variation

Es un proyecto del gobierno de Nueva Zelanda que visualmente ofrece una interfaz sencilla y atractiva.

En el apartado técnico, se basa en una aplicación desarrollada en *Adobe Flash* llamada *Instant Maps* [3]. Aunque Flash permite desarrollar interfaces gráficas muy visuales, tiene la desventaja que no se puede ejecutar en la mayoría de dispositivos móviles actuales.

Existe una versión móvil migrada a Javascript y HTML que no se basa en Flash y que tiene un aspecto visual menos llamativo [4].

En la figura **??** se puede ver los diferente smódulos que tiene el Atlas, dónde el protagonismo de la aplicación cae en primer momento en el propio mapa y después en las gráficas. El mapa temático, como tal, no ofrece información adicional de la cartografía, no da opción de visualizar imágenes satelitales ni interactuar con ninguna otra base cartográfica.

[3] http://www.instantatlas.com
[4] http://www.hqsc.govt.nz/assets/Health-Quality-Evaluation/Atlas/DeprivationHTML/atlas.html

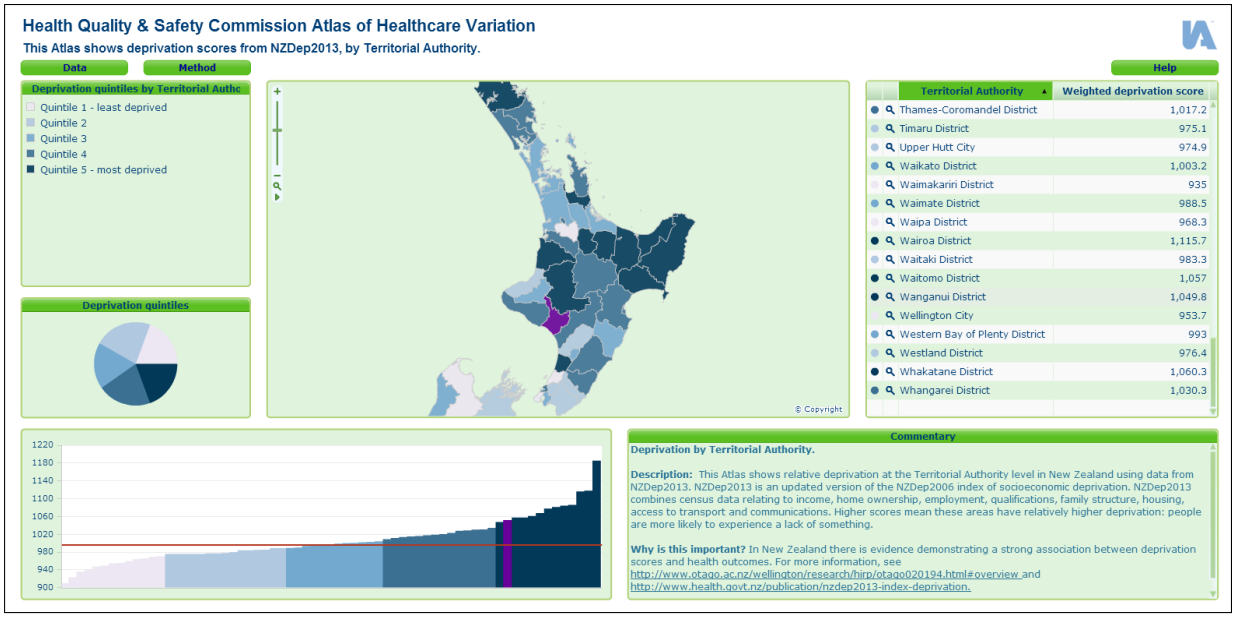


**Figura 5.2**: Página principal del Atlas of Healthcare Variation
Fuente: Captura de pantalla del Atlas

## 5.3. Tecnologías implicadas

A continuación se pretende describir brevemente las características de las principales librerías y productos de software que caracterizan este TFM sin pretender que sea un estudio exhaustivo o un análisis comparativo profundo. Para ello se han dividido las librerías/software en diferentes categorías:

- **Librerías Javascript para Cartografía**: librerías utilizadas como visor SIG de mapas y datos cartográficos
- **Librerías Javascript para Visualización y Análisis de datos**: librerías que sirven para el análisis de datos, la representación y renderizado de gráficos.
- **Servidores de mapas**: diferentes soluciones de software que actúan como servidores de cartografía.

### 5.3.1. Librerías Javascript para Cartografía

Esta sección trata de describir brevemente las librerías Javascript open source más utilizadas hoy en día para la representación cartográfica a través de la web. No pretende ser un estudio de mercado de las posibles librerías a utilizar, sino un breve acercamiento.

#### OpenLayers

Es un framework desarrollado y mantenido por la *Open Source Geospatial Foundation (OSGeo)* aunque originalmente fue desarrollado por la empresa MetaCarta en 2005 con el objetivo de ser la alternativa libre a otras librerías propietarias como Google Maps principalmente. En la actualidad **OpenLayers** es un proyecto estable y maduro formado por una comunidad que mantiene y desarrolla constantemente el proyecto. A nivel técnico, **OpenLayers** está desarrollado en Javascript orientado a objetos siendo el resultado un

conjunto de librerías con una sólida estructura. Además, soporta una amplia variedad de estándares abiertos (WMS, WFS, JSON, GML,...) y propietarios (*Google*, *Yahoo*, ...) lo que permite a **OpenLayers** la posibilidad de visualizar datos de diferentes proveedores . Además, soporta varios sistemas de referencia, funciones de transformación de coordenadas. Cabe destacar que es un framework muy completo y con muchas características idóneo para aplicaciones avanzadas pero precisamente por la potencia que ofrece presenta también una curva de aprendizaje mayor que otras librerías y que además está en función de la versión de **OpenLayers** a utilizar.

**Año de primer lanzamiento**: 2006

**URL**:http://openlayers.org

**Última versión estable**: 3.4.0 (marzo 2015)

### LeafLet

**LeafLet** es una librería escrita 100 % en Javascript cuyo primer lanzamiento fue en 2011. Las principales ventajas que ofrece respecto a *OpenLayers* son el tamaño de código que es significativamente menor y, al ser más nueva, saca partido de las mejoras que han sufrido los estándares y los navegadores web como son el uso de HTML5 y CSS3 sin dejar por ello de ser compatibles con otros navegadores anteriores (por ejemplo, Internet Explorer 7). **LeafLet** se presenta como una librería moderna diseñada con simplicidad y usabilidad que no implementa todas las funciones necesarias en un SIG sino que implementa de forma sencilla las funciones que la mayoría de desarrolladores necesitan. Estas premisas son las que hacen que **LeafLet** tenga una curva de aprendizaje baja favoreciendo la iniciación a su uso. Al igual que OpenLayers, da soporte a WMS, GeoJSON y KML entre otros. Pero a diferencia de *OpenLayers*, no da soporte a GML ni WFS aunque para este último existen plugins.

**Año de primer lanzamiento**: 2011

**URL**: http://leafletjs.com

**Última versión estable**: 0.7 (noviembre 2013)

### ModestMaps

**ModestMaps** es una librería que inicialmente se desarrolló en *AS3* y que posteriormente se portó a otros lenguajes, entre ellos Javascript resultando en una librería muy ligera. Comparte algunas características con *Leaflet*: peso muy reducido, API simple, compatibilidad cross-browser. Pero a diferencia de *Leaflet*, no incluye entres sus características: implementación de controles mínimos (p.e. ausencia de botones de zoom), uso de GeoJSON ni KML. Para enmendar estas desventajas, se suele combinar **ModestMaps** con el uso de librerías que extienden sus funciones (como la librería *Wax*) y con el uso de *TileMill* para evitar el uso de GeoJSON.

**Año de primer lanzamiento**: 2010

**URL**: http://modestmaps.com

**Última versión estable**: 3.3.6 (octubre 2014)

#### Polymaps

Es una librería Javascript que pretende ser rápida y mejorar la carga de grandes conjuntos de datos. Para ello basa su origen de datos en GeoJSON mientras que delega su renderizado en SVG lo cual hace que no sea compatible con navegadores antiguos. Como principal inconveniente de Polymaps destaca su falta documentación y de información con ejemplos que no funcionan al 100 %.

**Año de primer lanzamiento**: 2010?

**URL**: http://polymaps.org

**Última versión estable**: 2.5.1 (abril 2011)

#### Otras Librerías

Existen otras librerías que podrían ser consideradas para realizar un proyecto SIG, pero que quedan fuera del alcance de este proyecto. Las librerías a tener en cuenta, bien sea por su popularidad o bien por su potencial o interés que despierta son:

- **Geomajas**: implementación con Java (mediante GWT). Incluye la parte del servidor y facilita en gran medida la parte de la interfaz gráfica. Aunque Tiene El uso de GWT puede ser su principal hándicap al considerar su curva de aprendizaje y que el resultado puede ser menos ligero que otras aplicaciones.
- **GeoExt**: framework para desarrollar aplicaciones RIA. Está integrado por OpenLayer y ExtJS.
- **Mapfish**: framework disponible para Python (mediante Pylons), PHP y Ruby que utiliza OpenLayers y GeoExt.
- **Geomoose**: escrito en HTML y Javasacript, se basa en OpenLayers y MapServer. También tiene algunas librerías escritas en PHP. A primera vista parece que carece de la sencillez de otras librerías Javascript puesto que su despliegue en el servidor web consiste en varios directorios y múltiples archivos.

### 5.3.2. Librerías Javascript para Visualización y Análisis de datos

Uno de los requisitos del proyecto consiste en implementar funcionalidades de analísis estdístico y crear zonas de influencia mediante diagramas de Voronoi. Para ello, se utilizarán librerías open source de análisis y representación de datos en la parte de cliente, librerías en Javascript.

#### D3

**Data-Drive Documents** (**D3**) funciona de manera similar a otros frameworks Javascript (por ejemplo, **jQuery** o **Node.js**) en cuanto a selectores se refiere pero brinda la posibilidad de generar animaciones así como interactuar con los gráficos generados. Una de sus características principales es que permite cargar conuntos de datos grandes y en diferentes formatos (GeoJSON entre ellos) lo que le hace altamente integrable en aplicacion SIG.

El hecho de utilizar estándares y emplear HTML, CSS y SVG hace de **D3** una librería que obtiene el máximo beneficio de los nuevos navegadores web siendo además *cross-browser*.

Su buena integración y su alta modularidad y eficiencia junto con la amplia comunidad que la utiliza y le da soporte de forma activa consiguen que sus principales inconvenientes (curva de aprendizaje y poca compatibilidad con navegadores antiguos) puedan desconsiderarse.

**Año de primer lanzamiento**: 2011

**URL**:http://d3js.org

**Última versión estable**: 3.5.5 (febrero 2015)

### Cytoscape.js

**Cytoscape.js** es un port de la librería *Cytoscape*. Aunque es una librería enmarcada dentro de la teoría de grafos y redes, se puede utilizar para el análisis y visualización de datos. Al igual que *D3*, soporta la entrada de datos mediante JSON.

Uno de sus puntos fuertes es que está pensada para que el usuario pueda interactuar con los gráficos generados.

**Año de primer lanzamiento**: 2011

**URL**:http://js.cytoscape.org

**Última versión estable**: 2.3.12 (abril 2015)

### Turf

Es una librería Javascript creada por *Mapbox* para el análisis espacial que incluye operaciones de análisis espacial y operaciones estadísticas integradas modularmente. Además, permite crear datos en formato GeoJSON que es el tipo de dato espacial con el que trabaja.

Trabaja con coordenadas en WGS84 y una de sus principales ventajas es que está orientado puramente al análisis SIG pero de forma sencilla (por ejemplo, las distancias las devuelve en metros o en millas). Además, es altamente integrable con librerías de mapas (como *Leaflet*) y con *D3*.

**Año de primer lanzamiento**: 2013

**URL**:http://turfjs.org

**Última versión estable**: 1.4.8 (febrero 2015)

### Processing.js

**Processing.js** es un port del lenguaje *Processing* a Javascript. Basicamente consiste en escribir código en *Processing* e incrustarlo dentro del código HTML como si fuese código Javascript. Utiliza HTML5 para procesar animaciones, visualización de datos y animaciones interactivas principalmente aunque es soportado por algunos navegadores más antiguos (por ejemplo, *Internet Explorer 9*).

Su principal inconveniente es que fundamentalmente se trata de otro lenguaje de programación.

**Año de primer lanzamiento**: 2008

**URL**:http://processingjs.org

**Última versión estable**: 1.4.8 (marzo 2014)

#### Otras Librerías

Existen muchas otras librerías en Javascript para representación de datos, análisis, presenteaciones, etc. Entre también muy populares se encuentran:

- **Raphael.js**: librería para facilitar el uso de gráficos vectoriales en la web mediante SVG y VML. Da soporte a navegadores antiguos (por ejemplo, *Internet Explorer 9* y *Firefox 3.0*).
- **dc.js**: enfocada principalmente al uso de gráficos de tipo *charts*.

### 5.3.3. Servidores de mapas

Una pieza fundamental para realizar con éxito el producto resultante de este TFM (el *Atlas*) es el servidor de mapas. Un servidor de mapas es una aplicación residente en un servidor que publica la información geográfica del proyecto (municipios, provincias, ...) a través de Internet. Es necesario destacar que el servidor de mapas no tiene por qué estar instalado en la misma máquina en la que está instalado la base de datos ni el servidor web donde se aloja la aplicación.

Sin pretender ser una comparativa a fondo, sino simplemente una pequeña búsqueda para encontrar el producto que mejor se adapte a este proyecto, en esta sección se analizarán 3 aplicaciones que son de alguna manera un referente entre los servidores de mapas: GeoServer, MapServer y Mapnik.

#### Criterios del servidor de mapas

Para escoger un servidor de mapas adecuado se ha tenido en cuenta los siguientes 4 factores:

- Cumplimiento de estándares OGC[5]: si el servidor de mapas sigue estándares OGC implica que los datos serán accesibles desde un mayor número de clientes y que no siguen un estándar propietario que obliga a acceder desde un determinado tipo de clientes.
- Formatos soportados: a qué tipo bases de datos se pueden conectar (Postgis, Oracle, MySQL), qué tipos de archivos vectoriales son capaces de cargar, conexión con rásters.
- Rendimiento: aunque para este caso los requisitos de rendimiento no son muy exigentes debido principalmente a que los datos cartográficos que se disponen no son muy elevados, se debe considerar que el servidor proporcione un rendimiento óptimo.
- Facilidad de uso: es muy valorable la facilidad/dificultad del servidor para poder añadir/modificar capas, almacenes de datos y, en general poder configurar cualquier aspecto del servidor.

[5] http://www.opengeospatial.org/standards

### GeoServer

**GeoServer** es una referencia entre los servidores de mapas, aunque que no es de los más antiguos su facilidad de administración y edición hacen que sea un producto muy sencillo de utilizar, hecho que sumado al alto cumplimiento de estándares OGC (WMS, WCS, WFS y WFS-T) ha resultado en un incremento de su uso en sus últimas versiones.

**GeoServer** está escrito y compilado en Java por lo que para poder utilizarlo es necesario utilizar un contenedor de aplicaciones Java como pueda ser *Apache Tomcat*[6]. El hecho de que sea una aplicación Java implica una desventaja competitiva respecto a otros servidores que puedan estar escritos en C/C++ puesto que se asume que los servidores escritos en C/C++ serán más rápidos y harán mejor gestión de la memoria y de los recursos del sistema.

Haciendo un breve resumen, estos son los puntos fuertes y débiles de **GeoServer**:

| Ventajas | Inconvenientes |
|---|---|
| Aplicación muy portable debido a Java | Gestión de memoria y de recursos limitada por Java |
| Correcta implementación de WMS, WCS, WFS y WFS-T | Necesita tener Java instalado y un contenedor de aplicaciones como Tomcat |
| Interfaz GUI a través de páginas web para administración y configuración (también mediante archivos XML) | |
| Soporta Postgres/Postgis, shapefiles y rásters principalmente | |
| Estilo de capas basado en formato SLD | |
| Curva de aprendizaje baja | |

**Año de primer lanzamiento**: 2001

**URL**:http://geoserver.org

**Última versión estable**: 2.1.4 (junio 2012)

### MapServer

**MapServer** es probablemente el proyecto más veterano en cuanto a publicación de mapas se refiere siendo desarrollada su primera versión en 1994 para cubrir las necesidad existentes que otras aplicaciones comerciales de publicación de mapas existentes no cubrían.

**MapServer** está escrito y compilado en C y a día de hoy está mantenido por **OSGeo**[7] (OSGeo mantiene un buen número de proyectos de software geoespacial).

Haciendo un breve resumen, estos son los puntos fuertes y débiles de **GeoServer**:

[6] http://tomcat.apache.org
[7] http://www.osgeo.org

| Ventajas | Inconvenientes |
|---|---|
| Aplicación compilada para la mayoría de sistemas operativos | Configuración a través de archivos *map* que siguen un estándard propio. |
| Eficiente gestión de recursos y memoria | No dispone de interfaz gráfica para su gestión/administración |
| Correcta implementación de WMS, WCS, WFS y WMC | Curva de aprendizaje alta |
| Interfaz GUI a través de páginas web para administración y configuración (también mediante archivos XML) | |
| Soporta Postgres/Postgis, shapefiles y rásters principalmente | |
| Estilo de capas basado en formato SLD y GML principalmente | |

**Año de primer lanzamiento**: 1994

**URL**: http://mapserver.org

**Última versión estable**: 6.4.1 (enero 2014)

#### Conclusión

Para el desarrollo del proyecto se ha pretendido seleccionar un servidor de mapas que cumpla con la mayoría de factores definidos en 5.3.3. Para ello, se ha primado la facilidad de uso siempre y cuando los otros factores no sean gravemente castigados.

## 5.4. Tecnologías seleccionadas

En lo relativo a las librerías de cartografía en Javascript, y teniendo en cuenta Comparativa de librerías cartográficas en Javascript, la librería que se va a utilizar es LeafLet. AL ser librerías nuevas para mi, el principal motivo por el que me he decantado por LeafLet es su sencillez. OpenLayers se ve una librería muy potente pero muy estricta y que no ofrece atajos para cosas sencillas como sí tiene LeafLet. LeafLet parece más productiva y con el poco tiempo del que se dispone para finalizar el TFM parece la mejor solución: sencilla, ágil, curva de aprendizaje pequeña y con muchas funcionalidades implementadas.

Respecto a las librerías de visualización y análisis de datos, las librerías a utilizar son D3 y Turf. Los motivos que me llevan a seleccionar estas librerías son:

- son dos librerías que aportan algunas funcionalidades diferentes pero que se pueden complementar entre ellas
- ambas soportan *cross-browsing* y son librerías bastante estables
- cuentan con gran cantidad de ejemplos y de documentación
- se integran bien con LeafLet

Por otra parte, en la referente a la interfaz gráfica existen varias librerías que son todas muy interesantes, sobre todo: *Bootstrap*, *jQuery* (con sus variantes *jQuery UI* y *jQuery*

*Mobile*), *YUI* y *ExtJs* (aunque el tipo de licencia en el que se distribuye me ofrecen dudas). La seleccionada para hacer la interfaz es **jQuery** debido a que tiene una gran comunidad que le da soporte y dispone de una gran variedad de plugins.

En la parte del servidor de mapas, el servidor a utilizar es **GeoServer**: se distribuye como servlets Java (con o sin servidor de aplicaciones) y cumple con los principales estándares OGC (cumple enteramente con WMS, WCS y WFS). Parte de su core es la librería *GeoTools* que está en constante desarrollo y cubre con creces la mayoría de necesidades presentes en un proyecto SIG.

Además, se utilizará el motor de base de datos *Postgres* puesto que, a parte de ser un motor open source, su extensión espacial *Postgis* es a día de hoy uno de los mejores motores espaciales de bases de datos open source. Postgis implementa una gran variedad de funciones espaciales así como reproyección de coordenadas y manejo de rásters.

# Capítulo 6

# Origen de la información

En este capítulo se hace una pequeña introducción a las fuentes de datos a utilizar en el TFM.

## 6.1. Indicadores de salud

Un indicador de salud es una variable que mide de manera objetiva datos o sucesos relacionados con la salud. El análisis y estudio de los indicadores de salud sirven para medir en el tiempo sucesos que pueden estar relacionados con factores biodemográficos y permiten tomar decisiones politico-sanitarias.

A nivel estadístico se puede una gran variedad de conclusiones relacionadas con grupos sociales, pero si se considera la geografía asociada a los indicadores se pueden llegar a conclusiones relacionadas con, por ejemplo, la cantidad y calidad de infraestructuras sanitarias.

### 6.1.1. Fuente de los indicadores de salud

En el caso de los indicadores de salud necesarios para desarrollar el TFM, la *Generalitat de Catalunya* publica varios conjuntos de datos en su portal [1] guardando la privacidad de la personas que representan dichos indicadores.

Los indicadores se pueden descargar en la mayoría de los casos en archivos de datos tabulados de tipo *Excel* o *CSV*. Los datos que se han descargado son datos que contienen de alguna manera algún atributo de tipo espacial: la asociación a un municipio, una comarca o una provincia. Si los datos no contienen esa componente geográfica no es válido para representarlo en el Atlas. En la figura 6.1 se muestra la página de descarga de indicadores de salud de la Generalitat de Catalunya.

[1] http://observatorisalut.gencat.cat

| | Tots | Tots | Tots |
|---|---|---|---|
| **Nombre del indicador** | **Ámbito de la información** | **Subámbito de la información** | **Origen principal de los datos** |
| Población asegurada por el CatSalut | 1.Demográfico y social | | SIIS (Sistema Integrado de Información de Salud) |
| Porcentaje de sobreenvejecimiento | 1.Demográfico y social | | SIIS (Sistema Integrado de Información de Salud) |
| Tasa de interrupciones voluntarias del embarazo | 1.Demográfico y social | | Registro de Interrupciones voluntarias del embarazo |
| Población con doble cobertura sanitaria | 1.Demográfico y social | | Encuesta de Salud de Cataluña |
| Prevalencia de sedentarismo | 2.Estilos de vida | | Encuesta de Salud de Cataluña |
| Prevalencia de consumo de tabaco | 2.Estilos de vida | | Encuesta de Salud de Cataluña |
| Prevalencia de consumo de riesgo de alcohol | 2.Estilos de vida | | Encuesta de Salud de Cataluña |
| Población con percepción de buena salud | 3.Estado de salud | Percepción de salud | Encuesta de Salud de Cataluña |
| Prevalencia de exceso de peso declarado en adultos | 3.Estado de salud | Problemas de salud | Encuesta de Salud de Cataluña |
| Prevalencia declarada de diabetes | 3.Estado de salud | Problemas de salud | Encuesta de Salud de Cataluña |
| Prevalencia de enfermedad o problema de salud crónico | 3.Estado de salud | Problemas de salud | Encuesta de Salud de Cataluña |
| Prevalencia de riesgo de padecer trastorno mental en adultos | 3.Estado de salud | Problemas de salud | Encuesta de Salud de Cataluña |

**Figura 6.1**: Ejemplo de indicadores de salud publicados por la Generalitat de Catalunya
Fuente: Observatorio del Sistema de Salud de Cataluña

## 6.2. Datos cartográficos

### 6.2.1. Municipios

Los datos cartográficos que van a hacer falta para este TFM son básicamente los municipios de Cataluña. Teniendo los municipios, podemos agruparlos posteriormente para generar otros órdenes de agrupamiento: se pueden ordenar por comarcas, por provincias o por Región Sanitaria que no tiene por qué coincidir con las provincias administrativas.

Para descargar los municipios existen 2 fuentes de información oficiales:

- **Centro Nacional de Información Geográfica: CNIG**[2]. Es un portal del Gobierno de España a través del cual se puede descargar información cartográfica de las provincias de España. Para hacer uso de este servicio es necesario registrarse de manera gratuita como usuario.

  Para descargar la información se debe acceder a la página principal del CNIG, desde el menú *Enlaces* se accede a la web *Centro de Descargas* donde a través del enlace *Descarga gratuita u obtención de información geográfica digital para uso no comercial, con aceptación de licencia de uso*. En el buscador de productos, se debe seleccionar el producto *CartoCiudad* y a continuación las 4 provincias de Cataluña (ver figura 6.2). Al realizar la descarga de cada provincia, se obtiene un archivo


**Figura 6.2**: Selección de descarga de datos cartográficos
Fuente: CNIG

  comprimido que incluye los siguientes archivos shapefiles:

  - Códigos postales: polígonos que identifican los códigos postales de la provincia.

[2] http://www.cnig.es

- Líneas auxiliares: líneas auxiliares extraídas de Catastro y que forman parte de la cartografía de fondo urbano como son líneas de acera y cursos fluviales.
- Manzana: polígonos que también forman parte de la cartografía de fondo urbano. Representan las manzanas de los edificios de ciudades extraídos de Catastro.
- **Municipio**: son los polígonos que demarcan las áreas de los municipios de la provincia. Es sin duda, una de la capa que más aporta para este proyecto.
- Portal_PK: portales y puntos kilométricos.
- Topónimos: es un shapefile de puntos que hace referencia a topónimos urbanos.
- Tramo_Vial: shapefile de líneas que relaciona tramos con viales.

Se puede consultar de manera más detallada la información de cada shapefile o capa en la siguiente URL: http://ftpcdd.cnig.es/PUBLICACION_CARTOCIUDAD/cartociudad_callejero_metadatos_serie.xml

- **Institut Cartogràfic i Geològic de Catalunya (ICGC)**[3]. De manera similar al CNIG, el ICGC también pone a disposición de los usuarios registrados en su servicio (igualmente de manera gratuita) una amplia variedad de datos cartográficos de Cataluña a través de su portal dedicado a la infraestructura cartográfica, el antiguo *Institut Cartogràfic de Catalunya*: http://www.icc.cat.

Para descargar datos cartográficos desde este portal se debe acceder primero a su visualizador que tiene el aspecto que se muestra en la figura 6.3. En las opciones

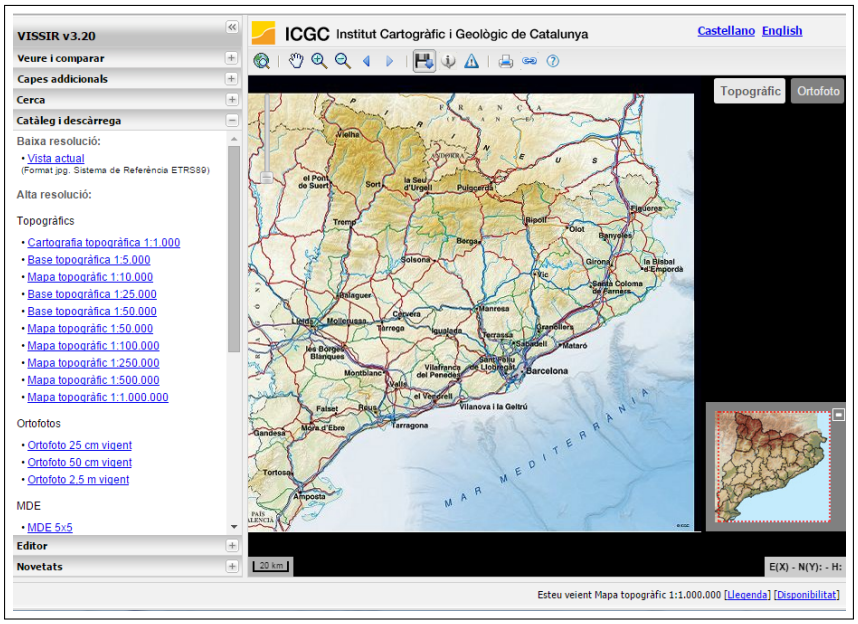


**Figura 6.3**: Vista general del visualizador
Fuente: ICGC-ICC

situadas en la parte izquierda del mapa se ubica la opción *Catálogo y descarga* desde donde se puede descargar los shapefiles necesarios para este TFM bajo la opción **Base Municipal 1:50.000**.

Este archivo comprimido contiene en diferentes shapefiles la siguiente información:

[3] http://www.icgc.cat

- Comarcas (áreas): delimitación de cada comarca.
- Municipios (áreas): polígonos que representan todos y cada uno de los municipios de Cataluña.
- Municipios (líneas): líneas delimitadoras de municipios. Cada línea representa la división entre dos municipios.
- Municipios (puntos): puntos con el centro urbano de cada municipio.
- Provincias (áreas): polígonos de las 4 provincias.

Además, el archivo comprimido también tiene dos shapefiles adicionales con los topónimos a nivel comarcal y poblacional.

Aunque con la información del *CNIG* se podría trabajar, la información del *ICGC* facilita enormemente el trabajo puesto que ya tiene agrupada la información en órdenes superiores a los municipios, es decir, comarcas y provincias. Si se eligieran los datos del *CNIG* sería necesario agrupar los municipios en provincias (agrupando a través de su código INE de provincia) y en comarcas (a través de listados adicionales que relacionasen municipios y comarcas). Todo este trabajo ya viene resuelto con los datos del *ICGC*.

### 6.2.2. Centros de salud

El principal problema que he encontrado con los centros de salud es que, a pesar de haber una buena página de la *Generalitat* [4] (ver figra 6.4) con un buscador de centros con múltiples opciones, no he encontrado un listado que se pueda tratar directamente con los datos de los centros.

A pesar que se pueden mostrar los centros atacando la capa WMS de la *Generalitat de Catalunya*, resulta interesante tener la información de los centros para poder ubicarlos directamente en el mapa sin necesidad del WMS y para poder generar los diagramas de Voronoi. Además, resulta especialmente interesante los datos de los centros para saber qué centros pertenecen qué Región Sanitaria y para analizar el tiempo de desplazamiento mínimo.

Para poder obtener estos datos de los centros, he desarrollado un pequeño script (ver Script para obtener los centros de salud) en Javascript y jQuery que saca por la consola del navegador los datos listos para insertar en un archivo CSV.

Para poder geolocalizar las direcciones obtenidas con el script de Script para obtener los centros de salud, he desarrollado una aplicación Java que realiza el proceso de geocodificación mediante el servicio *Geocoder* de *Google* [5].

[4] http://catsalut.gencat.cat/ca/ciutadania/centres-sanitaris/cercador
[5] https://developers.google.com/maps/documentation/geocoding/?hl=es

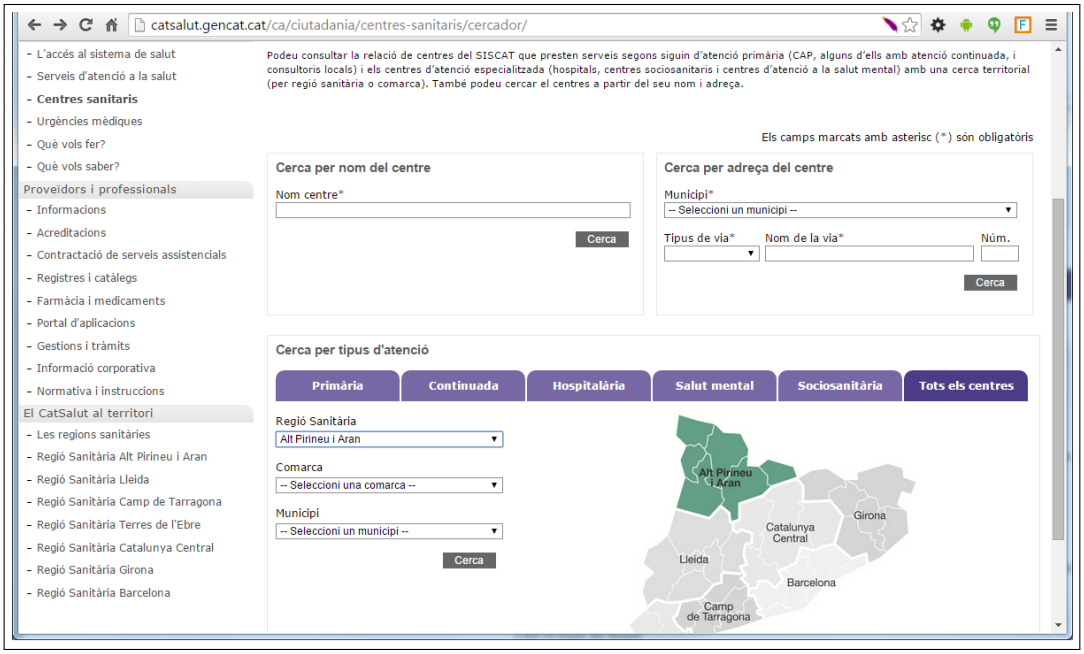


**Figura 6.4**: Buscador de centros de salud la Generalitat de Catalunya
Fuente: CatSalut. Servei Català de la Salut

# Capítulo 7

# Etapa de diseño

Aunque el diseño y el desarrollo de este proyecto puede llevarse a cabo de múltiples formas, se ha decidido llevar a cabo un diseño en 3 capas y 1 nivel:

- **Capa de datos**: capa donde se almacenan los datos y se gestiona el acceso a ellos.
- **Capa de negocio**: capa encargada de la gestión de la lógica de negocio y de recibir las peticiones del cliente,
- **Capa de presentación**: es la capa que ve el cliente, en este caso una página web.

Además, se dice que está implementada en 1 nivel puesto que se utiliza un sola máquina para implementar las diferentes capas.

En la siguiente figura se sintetiza a grandes rasgos la esencia de la arquitectura en 3 capas adoptada:

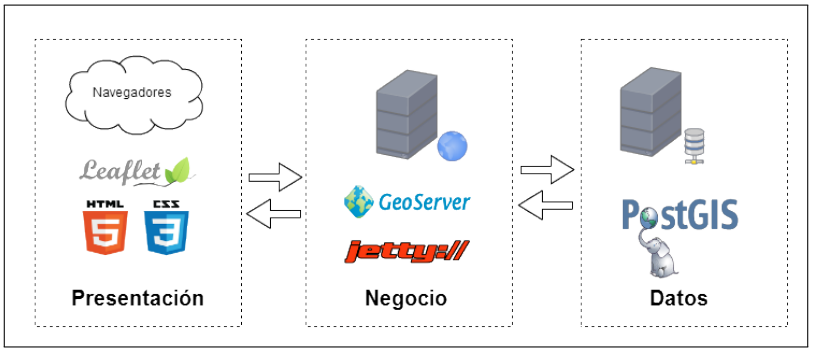


**Figura 7.1**: Diseño en tres capas

## 7.1. Capa de datos

En esta capa residen los datos tanto alfanuméricos como cartográficos almacenados en una base de datos relacional con soporte espacial como es **Postgres 9.4** junto con su extensión **Postgis 2.1**.

Para importar los datos se ha utilizado:

- El propio plugin para importar *shapefiles* de **Postgis**.
- Scripts SQL para importar datos tanto cartográficos como alfanuméricos obtenidos de páginas web. Algunos de estos scripts han sido creados con la hoja de cálculo **Excel** puesto que el origen eran archivos CSV y su manipulación en **Excel** es muy sencilla.

### 7.1.1. Datos almacenados

A continuación se amplia y matiza la información y conceptos concernientes a los datos utilizados en el proyectos y que se han descrito anteriormente en el capítulo Origen de la información.

#### Datos cartográficos

Los datos cartográficos que se han utilizado en el proyecto se dividen en 2 bloques:

- Datos que representan entidades administrativas: provincias, comarcas, municipios y capitales de municipios y provincias. Son datos cuyo origen es el **Institut Cartogràfic i Geològic de Catalunya (ICGC)** y representan dichas entidades y las relaciones entre ellas (por ejemplo, qué municipios pertenecen a qué comarcas).

- Regiones Sanitarias: aunque el **Siscat** no ofrece directamente datos cartográficos, si se puede obtener a través de varias publicaciones del *DOGC*[1] y de la propia página web del **CatSalut**[2] las comarcas que conforman las diferentes regiones sanitarias, por lo que obtener la geometría de cada región sanitaria simplemente consiste en realizar un agrupamiento por comarcas (por ejemplo, en SQL).

- Centros Sanitarios: el proceso para obtener la geometría de los centros sanitarios queda detallado en Centros de salud y consiste en un proceso de consulta en una página del **Gencat** y un posterior proceso de geolocalización (*geocoding*, en inglés). El proceso no ha resultado todo lo efectivo que me hubiese gustado por lo que he tenido que hacer alguna correccón en *QGIS*.

El primer bloque de datos consiste en 4 archivos *shapefile* con información geográfica proyectada en ETRS89 (concretamente en SRID 25831) que posteriormente ha sido reproyectada a WGS84 (en SRID 4326).

Además, toda la información geográfica que implique polígonos (provincias, comarcas, municipios y regiones sanitarias) ha sido sometida a un proceso de simplificación geométrica con una herramienta on-line (`http://mapshaper.org`) que permite aplicar facilmente diferentes sistemas de simplificación geométrica como es el caso del *algoritmo de Douglas-Peucker*.
Este proceso de simplificación a pesar de implicar una pérdida de la calidad de la información vectorial contribuye en minimizar el tamaño de la información que la aplicación cargará cuando el cliente entre en la página web, pero como la finalidad del proyecto es mostrar los indicadores sanitarios sobre un mapa se puede asumir la pérdida de calidad y detalle de la cartografía empleada en beneficio de la mejora de respuesta de la aplicación. En la siguiente tabla se muestra el tamaño de cada conjunto de datos original y después de aplicarle la simplificación:

#### Datos del *SisCat*

A través del **SisCat**[3] podemos obtener información relativa a la organización del propio **SisCat**: las entidades básicas necesarias para el proyecto y sus relaciones. Así pues, obtenemos las diferentes comarcas que forman cada una de las 7 regiones sanitarias

[1] *Diari Oficial de Catalunya*
[2] `http://catsalut.gencat.cat/ca/catsalut-territori/regions-sanitaries`
[3] *Sistema Sanitari Integral d'Utilització Pública de Catalunya*

| Datos | Tam. Original | Tam. Final |
|---|---|---|
| Shapefile de Municipios | 7.32 MB | 1.01 MB |
| Shapefile de Comarcas | 2.83 MB | 51 KB |
| Shapefile de Provincias | 1.78 MB | 33 KB |

**Cuadro 7.1**: Tamaño de los conjuntos de datos vectoriales utilizados

de Cataluña (descrito anteriormente en Datos cartográficos) y también el listado de áreas básicas de salud.

### Indicadores de salud

Los datos relacionados con los indicadores de salud de la Generalitat de Catalunya se presentan generalmente en archivos Excel o archivo tabulados CSV que permiten manipular facilmente sus datos.

El primer problema relacionado con los indicadores de salud radica en busca y seleccionar indicadores válidos: no todos los indicadores que ofrece la Generalitat de Catalunya en su página web[4] son válidos para este trabajo puesto que no todos se pueden relacionar con cartografía. De todos los indicadores disponibles, sólo un número reducido contemplan una componente geográfica que pueda ser usada en este trabajo. Las componentes geográficas que se han utilizado han sido municipios, comarcas, provincias y región sanitaria. Cualquier indicador que tenga sus datos por cualquiera de estas 4 componentes ha sido escogido para ser mostrado en el proyecto puesto que se disponen de la geometría de sus componentes geográficas.

En la siguiente tabla se muestra los indicadores sanitarios que se han utilizado:

| Descripción del indicador | Entidad geográfica |
|---|---|
| Donación de sangre (2013) | Municipio |
| Donación de sangre (2013) | Comarca |
| Esperanza de vida (2009, 2010 y 2011) | Comarca |
| Donación de sangre (2013) | Provincia |
| Donación de sangre (2013) | Región Sanitaria |
| Población asegurada por el CatSalut (2015) | Región Sanitaria |
| Porcentaje de sobreenvejecimiento (2014) | Región Sanitaria |
| Tasa de indicendencias de tuberculosis (2011) | Región Sanitaria |
| Tasa de interrupción de embarazo voluntario (2012) | Región Sanitaria |
| Tasa de mortalidad por accidentes de tráfico (2012) | Región Sanitaria |
| Tasa de mortalidad por suicidio o autolesión (2012) | Región Sanitaria |

**Cuadro 7.2**: Indicadores de salud empleados en el proyecto

[4] http://observatorisalut.gencat.cat/es/central_de_resultats/informes_cdr/dades_obertes

#### Datos del INE

Como última fuente de datos, cabe destacar el INE[5] que permite descargar el censo municipal a fecha 1 de enero de 2014 de todos los municipios de España agrupados por provincias a través de su página web: http://www.ine.es/dynt3/inebase/es/index.html?padre=517&dh=1.

Estos datos censales servirán para saber cuantos habitantes tiene cada municipio y poder hacer una estimación de la cobertura de cada centro sanitario con *diagramas de Voronoi*.

## 7.2. Capa de negocio

La capa de negocio contiene y desarrolla la lógica de la aplicación, define los objetos y servicios de negocio y las relaciones entre ellos y sus dependencias. Esta capa es una capa intermediaria entre la capa de datos y la de presentación encargada de procesar las peticiones realizadas por el usuario y devolver los resultados.

Generalmente, en esta capa intervienen servidores de aplicaciones (como *Tomcat* y *Jetty* si la aplicación está desarrollada en Java) y se suelen implementar con *frameworks* que facilitan su desarrollo y mantenimiento al desarrollador (como es el caso de *Spring* en Java o *Django* en Python).

Para este proyecto, la capa de negocio se ha delegado completamente en **Geoserver**. Una opción era implementar una aplicación sencilla en algún lenguaje de programación que gestionase las peticiones del usuario y le devolviése los datos solicitados implementando para ello las conexiones necesarias con la base de datos y los mecanismo necesarios para recibir peticiones *AJAX* y encapsular los datos en *XML* o *JSON*.

Pero **Geoserver** ya hace todo esto sin que se tenga que escribir ni una línea más de código en la lógica de negocio. **Geoserver** puede ser ejecutado con su propio contenedor de aplicaciones que además hace de servidor web: **Jetty**[6]. Esto quiere decir que con una instalación básica de **Geoserver** no es necesario instalar un servidor web adicional y que las páginas web de la aplicación pueden ubicarse directamente en su servidor web.

## 7.3. Capa de presentación

La capa de presentación es la capa encargada de mostrar la información y la aplicación al cliente, actúa como interfaz entre el usuario final y el resto de la aplicación.

La implementación de esta capa difiere en función del tipo de aplicación que se va a desarrollar. En el caso de este proyecto teniendo en cuenta que se trata de una aplicación web para la que se ha delegado la capa de negocio totalmente en **Geoserver**, no se seguirá un patrón *Modelo-Vista-Controladro (MVC)* puro.

El desarrollo para la vista se hará con HTML (siguiendo el estándard HTML5) junto con CSS (para definir los estilos visuales de la página web), Javasript (para la implementación de las acciones y comportamientos de la aplicación) y la librería **Leaflet** para la visualización de mapas. Además para el intercambio de datos desde el servidor a la vista se utilizará *JSON*[7] por su facilidad de integración en código Javascript y su menor

[5] Instituto Nacional de Estadística
[6] http://www.eclipse.org/jetty
[7] http://json.org

sobrecarga respecto a otros protocolos de intercambio de datos.

### 7.3.1. Funciones

Las funciones que se deben implementar en la interfaz para que el usuario pueda interactuar con la aplicación, son:

- Mostrar capa WMS de Gencat de los centros sanitarios.
- Mostrar capas WMS de Geoserver de las divisiones administrativas (provincias, comarcas y municipios).
- Seleccionar y visualizar indicadores de salud: ver gráficas y datos sobre mapa.
- Interactuar con el mapa (*zooming*, *panning*, ...)
- Seleccionar el tipo de mapa a visualizar (satélite u otra cartografía base)
- Mostrar las zonas de influencia de los centros sanitarios mediante diagramas de Voronoi

El caso de uso de la figura 7.3.1 muestra la funcionalidad de la aplicación:

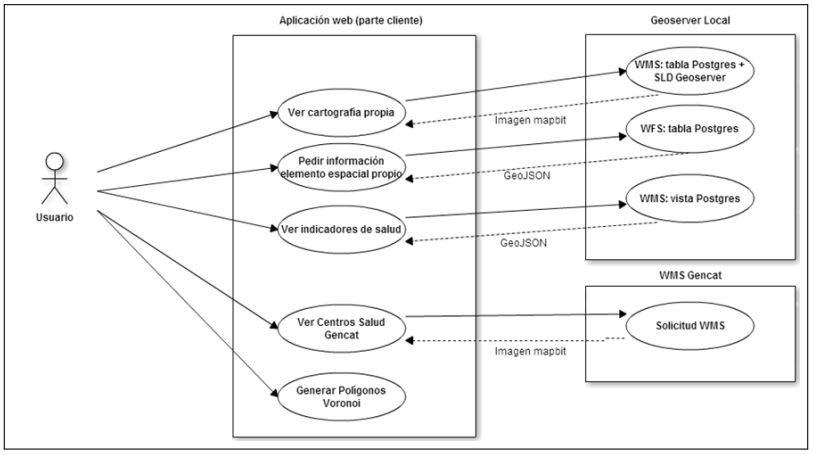


**Figura 7.2**: Diagrama de uso

### 7.3.2. Diseño

Se pretende que la interfaz que vea el usuario sea una interfaz sencilla e intutiva donde el mapa quede restringido al área geográfica de Cataluña y sea el mapa el protagonista principal de la aplicación. Es por esta última razón por la que se ha diseñado una aplicación en la que el mapa ocupa casi la totalidad de la pantalla con elementos que permiten ser ocultados y accesibles facilmente que dan acceso a la gestión de capas y visualización de los indicadores de salud.

El *wireframe* de la figura 7.3.2 muestra el diseño de la capa de presentación de la aplicación:

En el *wireframe* se muestran una barra lateral ubicada en la parte derecha de la pantalla que da acceso a la selección y visualización de los indicadores de salud. Esta barra se puede esconder para mostrar más contenido del mapa. Cuando se seleccione y

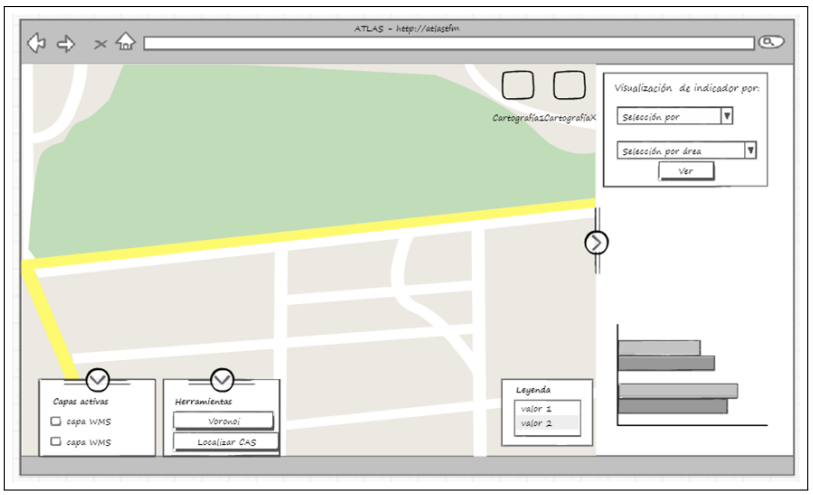


**Figura 7.3**: Diseño de la vista de la aplicación

se visualice un indicador de sanidad, además de mostrarse la gráfica del indicador en el mismo panel derecho, se mostrará una leyenda con los datos que se representan en el mapa. Dicha leyenda quedará ubicada en la parte inferior derecha del mapa.

Por otra parte, existen 2 paneles ubicados en la parte inferior izquierda de la pantalla: el primero da acceso a las capas WMS disponibles en la aplicación mientras que el segundo da acceso a herramientas como la cobertura geográfica de los centros sanitarios mediante diagramas de Voronoi.

Finalmente, en la parte superior derecha del mapa se ubica un control que permite cambiar la cartografía base sobre la que visualizar los datos. Esta cartografía consiste en un mapa con imagen satélite, un mapa de color gris, otro mapa con la cartografía de *OpenStreetMaps* y otro de *MapQuest*. La finalidad de este control es que el usuario pueda seleccionar diferentes cartografías base.

# Capítulo 8

# Etapa de desarrollo

Esta etapa comprende desde la instalación del sofrware necesario en el equipo de desarrollo hasta la implementación de los diseños descritos en Etapa de diseño.

Para ello se describirá brevemente la instalación de las aplicaciones necesarias en cada capa en el equipo de desarrollo para poder desarrollar satisfactoriamente el proyecto.

El equipo de desarrollo del que se dispone cuenta con un sistema operativo *Microsoft Windows 7* de 64 bits.

## 8.1. Capa de datos

### 8.1.1. Instalación

El primer paso para preparar la instalación de la capa de datos e indespensable para para poder instalar las aplicaciones necesarias en el resto de capas es la instalación del motor de base de datos.

La base de datos que se va a instalar es **Postgres 9.4** y para dar soporte a los datos espaciales se instalará su complemento **Postgis 2.1**.

La instalación de Postgres en Windows es una instalación muy sencilla que se lleva a cabo con un instalador de aplicaciones Windows en el que los pasos más importantes pasan por elegir las carpetas de instalación y datos a utilizar y el puerto de escucha de Postgres. En este caso se ha definido el puerto por defecto de Postgres, el puerto 5432. La instalación de Postgis se ha llevado a cabo mediante el instalador de paquetes y plugins de Postgres: **StackBuilder**.

### 8.1.2. Implementación

Una vez instalado el motor de la base de datos, el siguiente paso es crear la base de datos, las tablas necesarias para almacenar los datos e importar los datos. La mayor parte de estas acciones se pueden realizar en Postgres tanto por consola como por su interfaz gráfica (*pgAdmin*) junto con el plugin de importación de shapefiles.

La base de datos que se ha creado se ha llamado `atlas_tfm` y se han generado 4 esquemas para almacenar la información:

- `cartografia`. Guarda las tablas con la cartografía principal obtenida del ICGC.
- `indicadores`. Contiene las tablas necesarias para almacenar los datos de los indicadores de salud.
- `siscat`. Tiene las tablas que representan los objetos de dominio básicos del Sistema de Salud necesarios para el proyecto.
- `temp`. Es un esquema donde insertar la información en *crudo* para procesarla y guardarla convenientemente en las tablas correspondientes.

Y en la siguiente figura se muestra la estructura de la base de datos a nivel lógico:

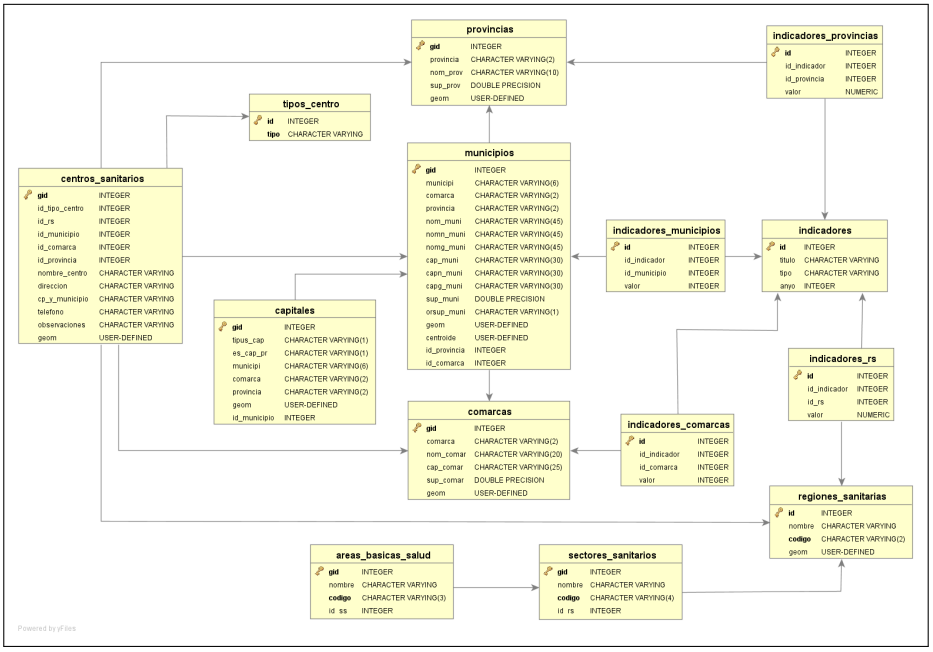


**Figura 8.1**: Detalle de la base de datos

Finalmente, para favorecer el rendimiento de la base de datos se han generado índices en aquellos campos que pueden resultar útiles para realizar búsquedas o cruces con otras tablas y en los campos de geometría.

### 8.1.3. Esquema indicadores

Si alguno de los esquemas y sus tablas merece la pena ser comentado en detalle es el esquema `indicadores`. Este esquema contiene la tabla principal `indicadores.indicadores` que guarda un registro de todos los indicadores disponibles en la aplicación. Por cada registro se guarda el tipo de división administrativa por el que se filtra (región sanitaria, comarca, municipio o provincia), el año del indicador, las unidades en las que se expresa el indicador y finalmente un campo de descripción que se utiliza para tener más información del indicador así como de qué fuente de información procede.

Por otra parte, este esquema tiene también 4 tablas más que son las que guardan los datos en sí de los indicadores: una tabla para los indicadores de municipios, otra para comarcas, otra tabla para provincias y otra para las regiones sanitarias. De esta forma, si existe un indicador de tasa de *Esperanza de vida por comarcas* de 3 aõs, esta tabla tendrá tantos registros como comarcas y aõs de registro existan (3 x 41 comarcas).

Finalmente, merece la pena destacar que este esquema presenta 5 vistas SQL que son las que son mostradas a través de **Geoserver**. Mediante estas vistas se obtienen los valores finales a mostrar y que son procedentes de diferentes tablas. De estas 5 vistas sólo una no tiene datos espaciales: `indicadores.v_indicadores_disponibles`. Ello se debe a que esta vista se utilizará para que la aplicación conozca qué indicadores existen.

En el anexo Vistas SQL se adjuntan las vistas creadas.

## 8.2. Capa de negocio

La implementación de la capa de negocio consiste en la instalación y la configuración de **Geoserver**.

En primer lugar, se ha instalado la última versión estable de **Geoserver** que actualmente es la versión 2.6.2. Aunque Geoserver puede ser descargado como un archivo *WAR* para ser desplegado en un contenedor de aplicaicones Java, también permite ser descargado como una versión de instalador de Windows que al ser instalado instala también un contenedor de aplicaciones (**Jetty**) y da la posibilidad de elegir en que puerto escucha Geoserver a través de Jetty. En el caso del equipo de desarrollo se ha elegido el puerto 8080 como puerto de escucha.

Trás una sencilla instalación, Geoserver queda instalado. Para comenzar a configurar y desarrollar en el entorno de trabajo de Geoserver sólo es necesario entrar desde un navegador web a la URL `http://localhost:8080/geoserver`.

### 8.2.1. Aspectos básicos de Geoserver

En este punto, antes de continuar con la descripción de la implementación de la capa de negocio es recomendable describir a grandes rasgos el funcionamiento de Geoserver.

El funcionamiento de Geoserver se basa en *espacios de trabajo* que no son otra cosa sino contenedores de capas y datos que posteriormente serán publicados. Cada espacio de trabajo puede contener diferentes *almacenes de datos*. Los *almacenes de datos* son conexiones a bases de datos (también pueden ser conexiones a servidores WMS, acceso directo a shapefiles en el sistema o archivos raster).

Finalmente, cada *almacén de datos* permite crear nuevas *capas*. Si tenemos un almacén de datos cuyo origen es una conexión Postgres/Postgis, la capa que creemos será la conexión directa a una tabla o vista de un esquema de esa base de datos.

Entonces, para publicar datos de una tabla o vista de Postgres, lo primero es definir un espacio de trabajo y sus almacenes de datos. Entonces creamos las capas y finalmente se le dan estilo. La vía por la que se le da estilo a una capa en Geoserver es principalmente mediante archivos *SLD*. Los archivos SLD son archivos XML que definen las reglas necesarias para mostrar de una determinada manera los datos espaciales a mostrar en el mapa.

### 8.2.2. Configuración de Geoserver

Seguún lo explicado en la sección anterior, para configurar Geoserver se ha creado un *espacio de trabajo* que se ha llamado `TFM_ATLAS`. A este espacio de trabajo se le han activado los servicios *WCS*, *WFS* y *WMS*.

Posteriormente, se han definido 3 *almacenes de datos* según se muestra en la figura 8.2: Finalmente se han creado y publicado diferentes capas, de las cuales las más importantes son:

En el anexo Estilos SLD de Geoserver se adjuntan los estilos SLD más significativos que se han utilizado y la salida generada.

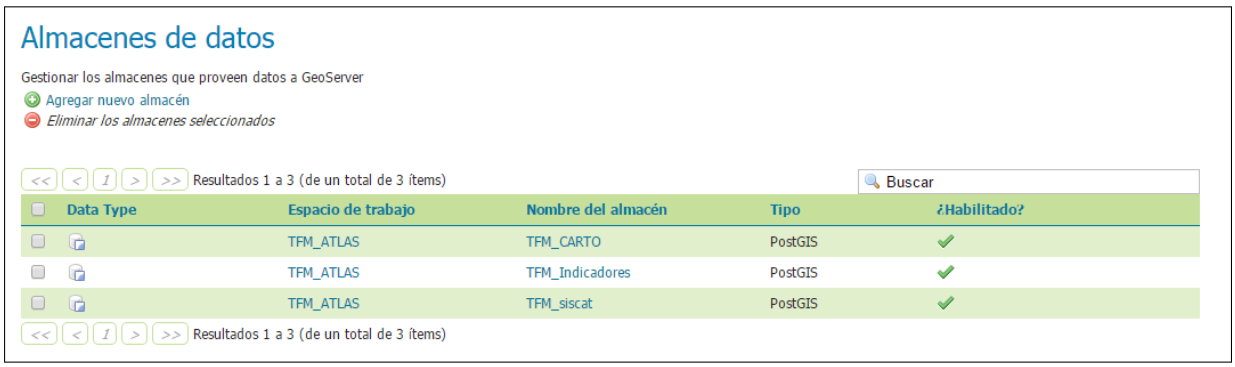


**Figura 8.2**: Almacenes de datos definidos en Geoserver

| Nombre de capa | Vista/Tabla Postgres | Descripción |
|---|---|---|
| TFM_ATLAS: capitales_vista | cartografia.v_capitales | Muestra como puntos los municipios (se activan a un nivel de zoom alto[1]). Las capitales de provincias son mostradas con un icono diferente. Muestra también el nombre del municipio. Posteriormente se utilizará como conexión WMS. |
| TFM_ATLAS: municipios | cartografia.municipios | Muestra los polígonos que conforman los municipios. Posteriormente se utilizará como conexión WMS. |
| TFM_ATLAS: comarcas | cartografia.comarcas | Muestra los polígonos que conforman las comarcas. Posteriormente se utilizará como conexión WMS. |
| TFM_ATLAS: comarcas_textos | cartografia.comarcas | Muestra los texto de forma independiente a sus polígonos. Posteriormente se utilizará como conexión WMS. |
| TFM_ATLAS: provincias | cartografia.provincias | Muestra los polígonos que conforman las provincias. Posteriormente se utilizará como conexión WMS. |
| TFM_ATLAS: provincias_textos | cartografia. v_textos_provincias | Muestra los texto de forma independiente a sus polígonos. Posteriormente se utilizará como conexión WMS. |
| TFM_ATLAS: v_indicadores_rs | indicadores. v_indicadores_rs | Muestra todos los indicadores de las regiones sanitarias como polígonos. Se utilizará en una llamada *WFS* filtrada por indicador de salud a mostrar. |
| TFM_ATLAS: v_indicadores_ provincias | indicadores. v_indicadores_provincias | Muestra todos los indicadores de salud de las provincias como polígonos. Se utilizará en una llamada *WFS* filtrada por indicador de salud a mostrar. |
| TFM_ATLAS: v_indicadores_ comarcas | indicadores. v_indicadores_comarcas | Muestra todos los indicadores de salud de las comarcas como polígonoss. Se utilizará en una llamada *WFS* filtrada por indicador de salud a mostrar. |
| TFM_ATLAS: v_indicadores_ municipios | indicadores. v_indicadores_municipios | Muestra todos los indicadores de salud de los municipios como polígonoss. Se utilizará en una llamada *WFS* filtrada por indicador de salud a mostrar. |
| TFM_ATLAS: v_indicadores_ disponibles | indicadores. v_indicadores_disponibles | Es la única capa que no muestra información geográfica. Se utiliza para saber qué indicadores de salud hay disponibles en la aplicación y es solicitado en formato JSON. |

**Cuadro 8.1**: Tamaño de los conjuntos de datos vectoriales utilizados

## 8.3. Capa de presentación

La interfaz de usuario se ha desarrollado con tecnología web: HTML (siguiendo el estándar 5), CSS (siguiendo la versión 3) y Javascript.

Para garantizar el correcto uso de los estándares de HTML y CSS se ha verificado satisfactoriamente el código con los validadores correspondientes del *W3C*: `https://validator.w3.org` y `https://jigsaw.w3.org/css-validator`.

En lo referente a Javascript, las librerías utilizadas han sido:

- **jQuery** (1.11.3) y **jQuery UI** (1.11.4), como extensión de Javascript, para el manejo del DOM, creación de funciones manejadoras de eventos.
- **jQuery DataTables** extiende las tablas HTML permitiendo añadir funciones y comportamientos como el ordenamiento de datos sin recargar la página.
- **Leaflet** como librería de visualización de mapas. Muy sencillo de utilizar y con curva de aprendizaje relativamente pequeña.
- **D3** para la visualización de datos. Permite manipular los datos (por ejemplo, desde JSON directamente) y representarlos con una gran variedad de opciones.
- **Turf** para realizar operaciones espaciales y visualizarlas en mapas.

### 8.3.1. Código Javascript

Se han escrito 3 archivos de Javascript:

- `ui.js`. Es la librería encargada de gestionar y definir el comportamiento de la interfaz web. Al cargar la página web establece las funciones manejadoras de eventos (*listeners*) de los componentes principales de la interfaz: botones, tablas, diálogos. Se describe las partes más relevantes de la librería en el anexo **??**.
- `layerFactory.js`. Es una librería que ofrece 2 clases para que la aplicación pueda hacer llamadas a servicios WMS y WFS además de definir los estilos de mapas y colores. Se puede ver el código en el anexo **??**.
- `libreria.js`. Es un librería con bastante peso en la aplicación: es la encargada de inicializar el mapa y de realizar las llamadas para obtener los datos de los indicadores de salud y WMS. En el anexo **??** se describen sus funciones más importantes.

#### Diagrama de Voronoi

La representación de los diagramas de Voronoi consiste en hacer una estimación de la cobertura de población que cada *Centro de Atención Primaria (CAP)* tiene sobre los municipios. Como se disponen de la población por municipios, dato que proporciona el INE (ver Datos del INE), el proceso de estimación se lleva a cabo siguiendo el siguiente procedimiento:
para cada diagrama de Voronoi, la población que le corresponde se obtiene calculando la intersección de Voronoi con el área del municipio (lo que nos dará el porcentaje de

incidencia del polígono de Voronoi en el municipio) y se multiplica por la población del municipio. La siguiente ecuación sintetiza el proceso:

$$Cobertura(voronoi_i, munipio_j) = \frac{Area(municipio_i, voronoi_j))}{S_{municipio_i}} \times Poblacion_i \quad (8.1)$$

Por otra parte, para generar los diagramas o polígonos de Voronoi, se pretendía hacer de manera dinámica en el cliente mediante Javascript:
se leen los CAP de Geoserver y se generan los diagramas de Voronoi con la librería *Turf.js* (1.192 polígonos). A continuación se leen los municipios también de Geoserver (947) y se comprueba qué diagramas de Voronoi intersecan con qué municipios y así determinar el área de la intersección. Pero este proceso en la parte del cliente no es efectivo, genera tiempos de respuesta no admisibles en la capa del cliente al intentar *cruzar* los 1.192 polígonos de Voronoi con los 947 polígonos de los municipios.

Una aproximación para solucionar este problema sería calcular los diagramas de Voronoi al cargar la página y cuando se mostrasen no rellenarlos de ningún colos significativo, de manera que el usuario tendría más información al hacer click en el polígono de Voronoi que sería cuando se haría el cálculo de ese polígono respecto a los municipios.

La solución por la que he optado consiste en procesar los diagramas de Voronoi y calcular sus coberturas en el servidor mediante el uso de SQL en Postgres. Para ello he utilizado el código Javascript que genera los polígonos de Voronoi y se han insertado en la base de datos tal y como se detalla en el anexo *Script para generar los diagramas de Voronoi*.

## 8.4. Puesta en producción

Una vez desarrollado el producto, el siguiente paso es ponerlo en producción. Para ello se ha empleado una instancia gratuita de una máquina virtual de **Amazon** mediante su servicio **EC2**[2].

Las características de esta máquina son:

- Procesador Intel Xeon E5 2670
- Memoria RAM de 1 GB
- Disco duro de 30 GB
- Sistema operativo *Windows 2008 R2 Server* de 32 bits

Estas características del servidor, aunque son limitadas, son suficientes para alojar la aplicación y probarla por internet, fuera del entorno `localhost` de desarrollo.

El software que se ha instalado en la máquina es:

- Geoserver 2.6.2 (instalador para Windows) con Jetty
- Postgres 9.4
- Postgis 2.1
- Java 8

[2] *Amazon Elastic Compute Cloud (EC2)* es un servicio de computación en la nube que permite escalar rapidamente

# Capítulo 9

# Resultados

El producto que se ha obtenido es una aplicación web que consiste en una única página web que utiliza la tecnología AJAX y Javasript para cargar datos de forma dinámica.

Tal y como se ha descrito en la sección Puesta en producción, la aplicación se ha desplegado en una instancia de una máquina *Amazon EC2* cuyo principal atractivo es disponer de una dirección IP pública para poder comprobar de forma real la aplicación aunque sea con unas características de hardware reducidas. La dirección IP dónde está desplegada la aplicación es `http://52.16.49.164:8080/atlastfm`.

## 9.1. Vista principal

La primera imagen que el usuario ve al entrar en la aplicación consiste en una mapa sin muchos detalles ni colores (la página en general se ha diseñado con colores muy sobrios) centrado en Cataluña y cuyo ámbito se restringe a la misma: aunque el usuario mueva el mapa hacia otras localizaciones (acción de *panning*) el mapa no muestra más allá de Cataluã con la idea de no distraer al usuario de la finalidad del proyecto (el Atlas de Cataluña) y de evitar efectos no deseados por parte de las acciones del usuario 8). En la figura 9.1 se observa la imagen inicial que ve el usuario.

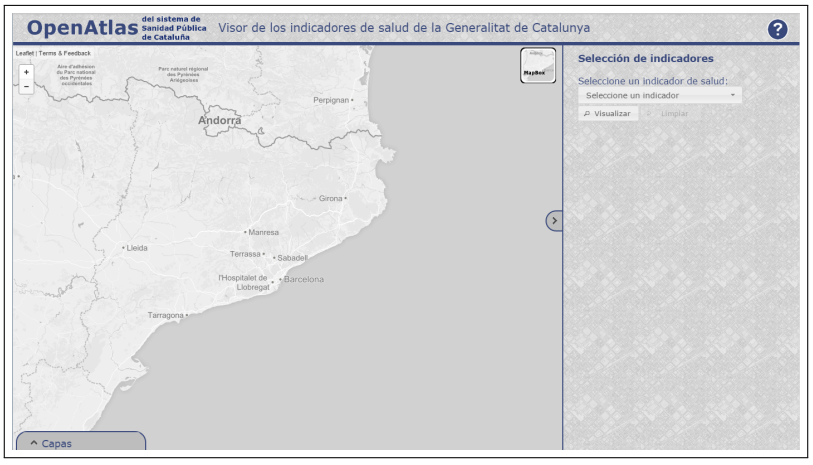


**Figura 9.1**: Vista inicial de la aplicación

Existen varios elementos diferenciados en la vista inicial (ver figura 9.1):

- El mapa, que dado su papel es el elemento que más espacio ocupa.
- Título en la barra superior y que da acceso a un botón de ayuda ubicado en la parte superior derecha.
- Botón de capas del mapa. Es un botón que esconde o muestra un panel con las capas disponibles en el mapa que son en definitiva conexiones WMS a Geoserver. Por defecto está escondido.

- Panel de indicadores. Ubicado en la parte derecha del mapa y ocupando casi todo el alto del mapa, este panel se puede ocultar y permite seleccionar los indicadores de salud para visualizarlos en el mapa.
- Controles del mapa. Los controles del mapa están dispersos en el propio mapa. Los controles principales son:
  - Acciones de zoom: está ubicado en la esquina superior izquierda del mapa, es un elemento por defecto.
  - Cartografía base: como se muestra en la figura 9.3 permite seleccionar diferentes fondos cartográficos (entre ellos una imagen satélite).
  - **Diagramas de Voronoi**: bajo el nombre de **Ver cobertura** y ubicado debajo del control del *zoom*, este control puede tardar en aparecer entre 2 y 4 segundos desde que se carga la página web. Esto se debe a que cuando empieza a cargar la aplicación comienza la carga de los diagramas de Voronoi y de los municipios y hasta que no finaliza el proceso de carga no se da la opción al usuario para evitar que el usuario pueda activar la opción sin que esté realmente disponible ver (la figura 9.2).

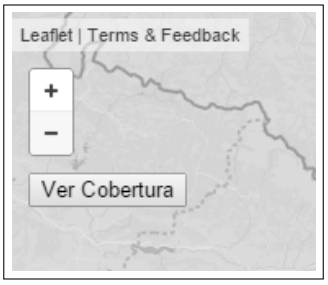


**Figura 9.2**: Control para mostrar los diagramas de Voronoi

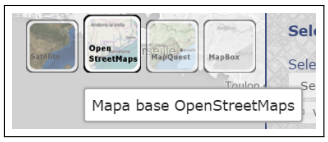


**Figura 9.3**: Mapas disponibles

## 9.2. Activación de capas

Para profudizar en el proceso de publicación de mapas de Geoserver sirviendo capas WMS, se han habilitado varias capas que el usuario puede activas y desactivas en el panel de capas.

En el panel de capas disponibles (ver figura 9.4) y junto cada capa, existe un icono de información que da libertad al usuario a obtener más información la capa en cuestión y acceder a la fuente oficial de esta información.

De todas las capas, sólo la capas *Centros de Salud (WMS propio)* es interactiva con el usuario: el usuario puede hacer click sobre sus elementos y obtener información de dicho elemento como muestra la figura 9.5.


**Figura 9.4**: Capas disponibles

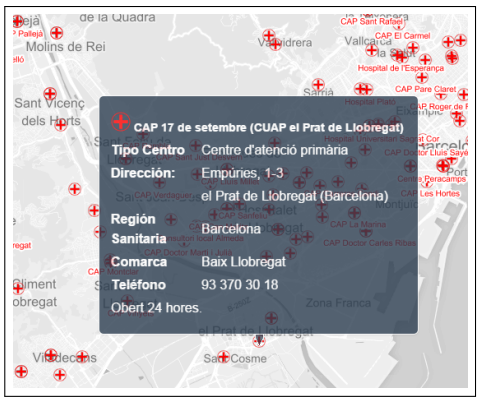


**Figura 9.5**: Ejemplo de información de un elemento de la capa

## 9.3. Visualización de indicadores

La visualización de indicadores puede hacer a través del panel ubicado en la parte derecha de la ventana. En dicho panel el usuario puede seleccionar qué indicador ver de los diferentes indicadores descritos en la sección Indicadores de salud (ver figura 9.7).

Una vez seleccionado el indicador, se mostrará un mapa *coroplético* donde cada polígono tendrá un color en función del indicador (ver figura **??**). Por otra parte, en el panel de *selección de indicador* se mostrará una tabla con los valores de cada entidad geográfica representada en el mapa (región sanitaria, comarca, ...) y su valor asociado del indicador mostrado. Junto con esta tabla aparece también una gráfica que muestra el estado global de ese indicador en la geografía catalana y que ha sido generada con *D3.js*.

Finalmente, para completar la visualización del indicador (ver figura 9.8), se muestra un cuadro de información que explica el indicador y contiene un acceso a la fuente oficial de información y una leyenda en la parte inferior derecha para mostrar los valores representados.

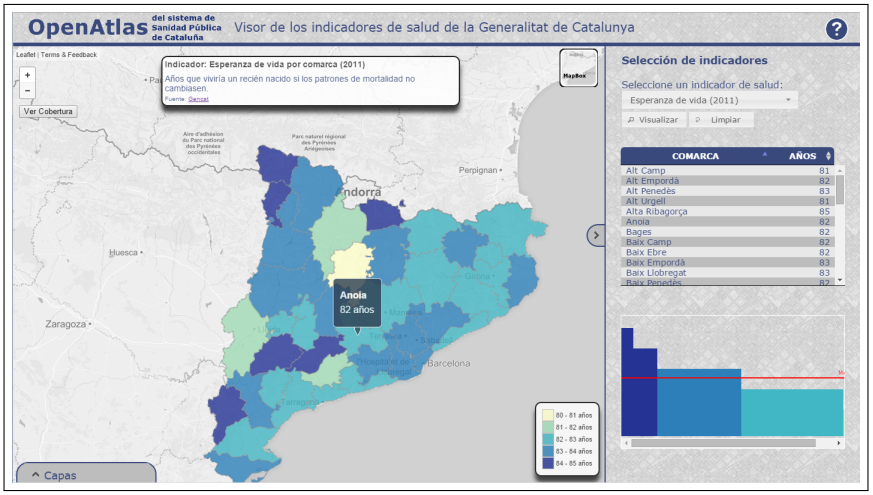


**Figura 9.6**: Vista general de un indicador seleccionado

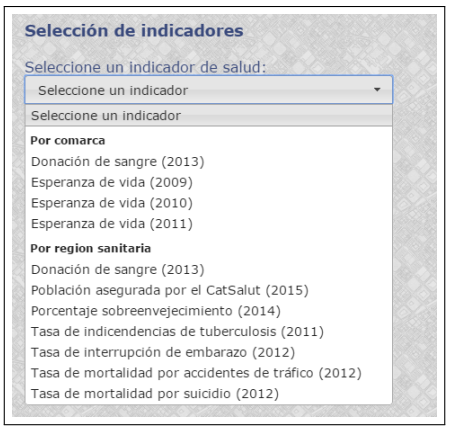


**Figura 9.7**: Selección de indicadores

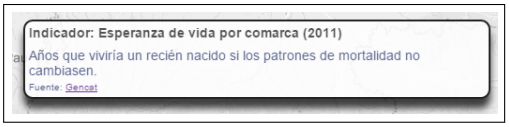


**Figura 9.8**: Cuadro informativo del indicador

## 9.4. Visualización de coberturas

Para acabar de explicar las funcionalidades de la aplicación, se mostrará la visualización de diagramas de Voronoi que sirven para estimar la cobertura de población cada centro sanitario.

Como se ha explicado anteriormente en la sección Vista principal de este mismo capítulo y recordando las observaciones descritas en la sección *Diagrama de Voronoi* del capítulo *Etapa de desarrollo*, los diagramas de Voronoi se generan una única vez en un proceso *off-line* en el servidor de manera que cuando un usuario entra en la aplicación, la aplicación descarga los diagramas de Voronoi generados anteriormente sin perder tiempo en generarlos. Una vez descargados los datos necesarios para generar los diagramas de Voronoi es cuando se activa la opción para que el usuario pueda visualizarlos mediante el botón *Ver cobertura* (ver figura 8).

Al mostrarse los diagramas de Voronoi en el mapa (ver figura 9.9), de manera análoga a la visualización de indicadores, se muestra una leyenda de colores (en la esquina inferior derecha del mapa) y un cuadro de información adicional (ubicado en el centro superior de la ventana).

Al pulsar en el área de cobertura de Voronoi se muestra un *popup* con los datos de cobertura sobre los municipios con los que interseca el polígono. Se puede ver un ejemplo en la figura 9.9.

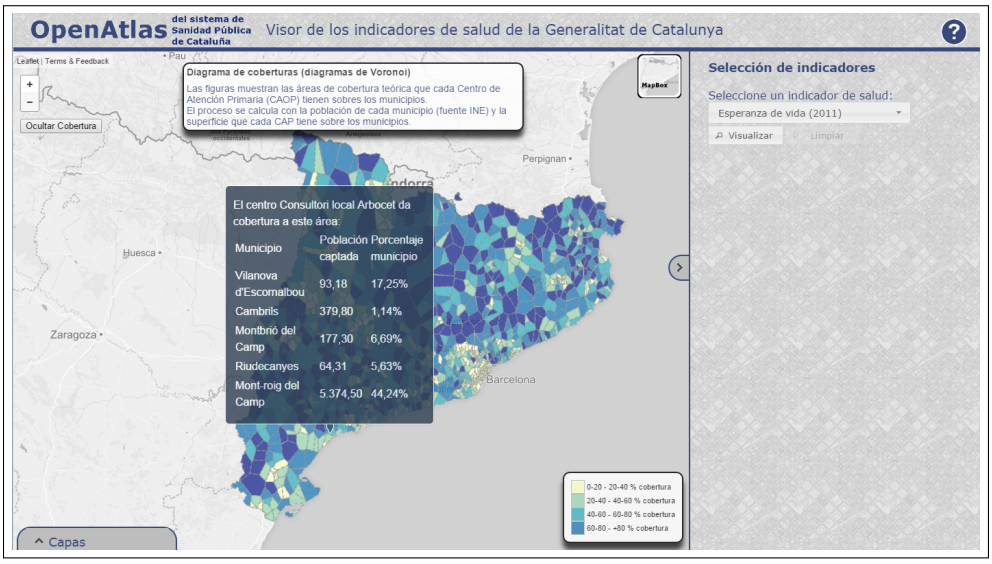


**Figura 9.9**: Captura de los diagramas de Voronoi

# Capítulo 10

# Conclusiones

Para finalizar la memoria del proyecto, en este capítulo se describirán las lecciones aprendidas durante el desarrollo del proyecto además de analizar los posibles problemas encontrados y proponer futuras líneas de trabajo.

## 10.1. Resultados

Para poder afirmar que el proyecto ha cumplido con los objetivos propuestos en su inicio basta con comprobar las funcionalidades del producto final:

- El producto se basa 100 % en tecnología *open source*: Geoserver, Leaflet, D3.js
- Se ha implementado un visor de mapas que sirve datos vectoriales cartográficos y alfanuméricos de diferentes fuentes de información.
- El producto implementa un visor GIS web que muestra datos estadísticos de manera dinámica.
- El visor permite interactuar con diferentes capas e indicadores de salud.

Con estas funcionalidades implementadas en el producto final, se puede afirmar que el proyecto ha alcanzado con éxito los objetivos planteados al inicio del proyecto.

## 10.2. Lecciones aprendidas

Actualmente la publicación de datos espaciales a través de mapas en la web se ha convertido en una herramienta casi imprescindible para los desarrolladores de aplicaciones y contenidos web debido a la mejora que han experimentado en los últimos años los Sistemas de Información Geográfica en gran parte al abaratamiento de costos en los SIG y gracias a la mejora y al acercamiento de mapas on-line al público en general como es el caso de *Google Maps*. Este acercamiento al gran público ha ocasionado también que se haya despertado interés desde hace unos años en los SIG por parte del mundo de *open source*. Este hecho ha provocado que el mundo de los SIG viva en una transformación en la que grandes empresas mantenían el dominio, en cuanto a sistemas SIG se refiere, estando los desarrolladores a merced de los cambios estratégicos que las empresas adoptasen sobre sus productos. Como ejemplo reciente: Google cerrará su servicio *Google Maps Engine*[1].

La explicación del anterior párrafo se debe a que una de las principales lecciones aprendidas en este proyecto ha sido **Geoserver**: esta aplicación ha despertado gran interés, sobre todo porque como usuario/desarrollador afectado por el cierre de un servicio de

[1] https://developers.google.com/maps-engine/

mapas, Geoserver se convierte en una opción en la que la comunidad *open source* mantiene un producto que no atiende a razones estratégicas de una empresa, sino a querer desarrollar y mejorar un buen producto.

Durante el trabajo con Geoserver la comparación con productos comerciales ha sido una constante: sencillez para realizar ciertas tareas (devolver un archivo GeoJSON o conectar con una base de datos) y complejidad para realizar otras tareas (la creación de estilos).

Otra lección muy gratificante ha sido conocer la riqueza de aplicaciones (en forma de programas o de librerías Javascript) que existe relacionadas con el mundo SIG. Conocer la existencia de los diferentes servidores de mapas *open source* y su funcionamiento, conocer nuevas librerías Javascript para análisis estadístico y espacial supone ampliar las posibilidades de mejora en los desarrollos que se hacen en el día a día. El haber instalado cada uno de los servidores de mapas y haber hecho un estudio de las librerías de visualización de mapas en Javasript en la etapa inicial del proyecto me ha permitido abordar de mejor el proyecto, teniendo claro qué medios utilizar.

Pero a lo apuntado en el párrafo anterior, hay que ñadir como conocimiento adquirido que es necesario conocer las limitaciones y casos de uso de dichas aplicaciones: las perspectivas que tenía pensado con *turf.js* eran bastante más altas de lo que finalmente conseguí desarrollar en los diagramas de Voronoi.

Y finalmente, quisiera destacar el hecho de simplificar y pre-procesar la información antes de servirla al cliente para conseguir mayor fluidez en el cliente y evitar malas experiencias por parte del usuario.

## 10.3. Problemas encontrados

Durante el desarrollo del proyecto se han encontrado varios problemas, algunos de los cuales han repercutido significativamente en el resultado final del proyecto.

Los principales problemas encontrados han sido:

- **Riesgos materializados**. En el capítulo *Análisis de riesgos* se describieron varios riesgos de los cuales 2 se han materializado a lo largo del proyecto teniendo consecuencias negativas en el resultado. Para minimizar los riesgos en el producto final opté por bajar la calidad de algún entregable al quitarle tiempo en su preparación.

- **Volumen de datos**. Aunque ha sido un problema que no ha repercutido de ningún modo en el resultado del producto ni del proyecto en general, creo que es interesante comentarlo: al cargar la cartografía de Cataluña no tuve la precaución de simplificar inicialmente la geometría, de manera que las pruebas en *localhost* no mostraban la realidad de un problema que apareció al publicar la aplicación en un servidor de internet y el tiempo que tardaba el cliente en cargar los datos. La solución pasó por simplificar las geometrías (ver capítulo *Etapa de diseño* para más detalles).

- **Diagramas de Voronoi**. Aunque la generación de diagramas de Voronoi con *D3.js* no es compleja, si ejecutaba el proceso en el cliente, como tenía pensado, llegaba tiempos de espera más que inaceptables debido a los cÍculos de intersección con municipios. Aunque la solución por la que que he apostado ha sido hacer los procesos en off-line en el servidor, otra buena solución habría sido hallar la intersección de cada centro sanitario sólo con su municipio.

## 10.4. Trabajo futuro

Aunque el proyecto ha conseguido cumplir con los objetivos propuestos inicialmente, consideraría trabajo futuro las siguientes líneas de trabajo:

- Creación de una versión móvil
- Creación de un módulo de gestión para poder añadir o eliminar indicadores de salud.
- Creación de funciones para calcular el centro de salud más cercano y su ruta.
- Creación de funciones para poder imprimir o exportar el mapa visto en pantalla

# Anexos

# Comparativa de librerías cartográficas en Javascript

## Código

Este ejercicio consiste en la creación de una página web con un mapa y la inserción de un punto (*marker*) que al hacerle click aparezca un cuadro de diálogo (*infowindow* o *popup*). La motivación de este ejercicio es obtener un primer acercamiento a las diferentes librerías aunque sin entrar en detalle y realizar una comparación con otra librería de referencia que conozco bastante bien: la API de Javascript de **Google Maps** (versión 3).

## Google Maps

El código de **Google Maps**, es un código sencillo de entender y bastante corto (aproximadamente 20 líneas de código *Javascript*) y que sólo ha llevado unos 5 minutos desarrollarlo. La salida del código se muestra en la figura 1.

**Listing 1**: Código en Google Maps

```
<!DOCTYPE html>
<html><head><title>Mapa con Google Maps</title>
<script type="text/javascript" src="http://maps.googleapis.com/maps/api/js?
    sensor=false"></script>
<style type="text/css">
html, body{ margin: 0px;    padding: 0px; }
html, body, #map_canvas { width: 100%;  height: 100%; }
</style>
<script type="text/javascript">
var infowindow = new google.maps.InfoWindow();
var position = new google.maps.LatLng(39.474016, -0.377382);
function init() {
  var map = new google.maps.Map(document.getElementById("map_canvas"), {
    zoom: 12,
    mapTypeId: google.maps.MapTypeId.ROADMAP,
    center: position
  });
  var marker = new google.maps.Marker({
    position: posicion,
    map: map,
    clickable: true,
    draggable: true,
    title: "un marker de ejemplo"
  });
  google.maps.event.addListener(marker, "click", function(event){
    infowindow.setOptions({
      content: "Coordenadas: " + this.getPosition().lng() + "," + this.getPosition().
          lat()
    });
    infowindow.open(map, marker);
  });
}
</script>
</head>
<body onload="init();">
```

```
  <div id="map_canvas"></div>
</body>
</html>
```

# OpenLayers

Respecto a **OpenLayers**, he comparado su última versión y la anterior estable (versión 2). En ambos casos el resultado ha sido un código denso duplicando el número de líneas (aproximadamente 50 líneas de *Javascript* para el caso de la versión 2 y 120 para la versión 3). Se puede ver la salida para la versión 2 en la figura 2 mientras que la figura 3 muestra el resultado de la versión 3.

**Listing 2**: Código en OpenLayers v2

```
<!DOCTYPE html>
<html><head><title>Mapa con OpenLayers</title>
<script src="http://www.openlayers.org/api/OpenLayers.js"></script>
<link rel="stylesheet" href="http://openlayers.org/en/v3.4.0/css/ol.css" type="text/
    css">
<style type="text/css">
html, body{ margin: 0px; padding: 0px; }
html, body, #map_canvas{ width: 100%; height: 100%; }
</style>
<script type="text/javascript">
var posicion = [-0.377382, 39.474016];
var evt;
function init(){
  var puntosLayer = new OpenLayers.Layer.Vector("Puntos");
  var pointGeometry = new OpenLayers.Geometry.Point(posicion[0], posicion[1]).
      transform("EPSG:4326", "EPSG:3857");
  var marker = new OpenLayers.Feature.Vector(pointGeometry, { name:"un marker de
      ejemplo"}, {
    pointRadius: 16,
    fillOpacity: 1,
    externalGraphic:"js/openlayers-v3.4.0-dist/images/marker-icon.png"
  });
  puntosLayer.addFeatures([marker]);

  var map = new OpenLayers.Map("map_canvas", {
    layers: [new OpenLayers.Layer.OSM()],
    projection: new OpenLayers.Projection("EPSG:3857"),
    center: [marker.geometry.x, marker.geometry.y],
    zoom: 12,
    eventListeners: {
      featureclick: function(e) {
        //alert("Marker: " + e.feature.attributes.name + "\r\nCoordenadas: " +
            e.feature.geometry.x + "," + e.feature.geometry.y);
        infowindow = new OpenLayers.Popup.FramedCloud("chicken",
           e.feature.geometry,
           new OpenLayers.Size(200, 200),
           "example popup",
           null, true);
           map.addPopup(infowindow);
      }
    }
  });
  map.addLayer(puntosLayer);

  var infowindow = new OpenLayers.Popup.FramedCloud(
      i, //id
      lonLat, // lonlat
      new OpenLayers.Size(200, 200), // size
      "Point:<---", // content
      null, // anchor
      true  // close
  );
  map.addPopup(infowindow);

```

```
  puntosLayer.events.register('click', infowindow, function (event){
    evt = event;
    console.log(event);
    this.toggle();
  });

  var dragFeature = new OpenLayers.Control.DragFeature(puntosLayer);
  map.addControl(dragFeature);
  dragFeature.activate();
}
</script>
</head>
<body onload="init();">
  <div id="map_canvas"></div>
</body>
</html>
```

**Listing 3**: Código en OpenLayers v3

```
<!DOCTYPE html>
<html><head><title>Mapa con OpenLayers</title>
<script src="http://openlayers.org/en/v3.4.0/build/ol.js" type="text/javascript">
    </script>
<link rel="stylesheet" href="http://openlayers.org/en/v3.4.0/css/ol.css" type="text/
    css">
<style type="text/css">
html, body{ margin: 0px; padding: 0px; }
html, body, #map_canvas { width: 100%; height: 100%; }
</style>
<script type="text/javascript">
var posicion = [-0.377382, 39.474016];
function init(){
  var marker = new ol.Feature({
    geometry: new ol.geom.Point(ol.proj.transform(posicion, "EPSG:4326", "EPSG:3857")
        ),
    name: "un marker de ejemplo",
  });
  var puntosLayer = new ol.layer.Vector({
    source: new ol.source.Vector({
      features: [marker]
    }),
    style: new ol.style.Style({
      image: new ol.style.Icon({
        src: "js/openlayers-v3.4.0-dist/images/marker-icon.png"
      })
    })
  });
  var layerOSM = new ol.layer.Tile({ source: new ol.source.OSM() });
  var map = new ol.Map({
    interactions: ol.interaction.defaults().extend([new app.Drag()]),
    target: 'map_canvas',
    layers: [layerOSM, puntosLayer],
    view: new ol.View({
      center: ol.proj.transform(posicion, "EPSG:4326", "EPSG:3857"),
      zoom: 12
    })
  });

  var infowindow = new ol.Overlay({
    element: document.getElementById("popup"),
    positioning: "bottom-center",
    stopEvent: false
  });
  map.addOverlay(infowindow);

  map.on("click", function(evt){
    var feature = map.forEachFeatureAtPixel(evt.pixel, function(feature, layer){
      return feature;
    });
    if(feature){
      var geometry = feature.getGeometry();
      infowindow.setPosition(geometry.getCoordinates());
```

```
51        var coords4326 = ol.proj.transform(geometry.getCoordinates(), "EPSG:3857", "
              EPSG:4326");
52      }
53    });
54  }
55
56  // Logica del Drag&Drop
57  window.app = window.app || {};
58  window.app.Drag = function(){
59    ol.interaction.Pointer.call(this,{
60      handleDownEvent: window.app.Drag.prototype.handleDownEvent,
61      handleDragEvent: window.app.Drag.prototype.handleDragEvent,
62      handleMoveEvent: window.app.Drag.prototype.handleMoveEvent,
63      handleUpEvent: window.app.Drag.prototype.handleUpEvent
64    });
65    this.coordinate_ = null;
66    this.cursor_ = 'pointer';
67    this.feature_ = null;
68    this.previousCursor_ = undefined;
69  };
70  ol.inherits(app.Drag, ol.interaction.Pointer);
71  window.app.Drag.prototype.handleDownEvent = function(evt){
72    var map = evt.map;
73    var feature = map.forEachFeatureAtPixel(evt.pixel, function(feature, layer){
74      return feature;
75    });
76    if(feature){
77      this.coordinate_ = evt.coordinate;
78      this.feature_ = feature;
79    }
80    return !!feature;
81  };
82  window.app.Drag.prototype.handleDragEvent = function(evt){
83    var map = evt.map;
84    var feature = map.forEachFeatureAtPixel(evt.pixel, function(feature, layer) {
85      return feature;
86    });
87    var deltaX = evt.coordinate[0] - this.coordinate_[0];
88    var deltaY = evt.coordinate[1] - this.coordinate_[1];
89    var geometry = (this.feature_.getGeometry());
90    geometry.translate(deltaX, deltaY);
91    this.coordinate_[0] = evt.coordinate[0];
92    this.coordinate_[1] = evt.coordinate[1];
93  };
94  window.app.Drag.prototype.handleMoveEvent = function(evt) {
95    if (this.cursor_) {
96      var map = evt.map;
97      var feature = map.forEachFeatureAtPixel(evt.pixel, function(feature, layer){
98        return feature;
99      });
100     var element = evt.map.getTargetElement();
101     if (feature){
102       if (element.style.cursor != this.cursor_){
103         this.previousCursor_ = element.style.cursor;
104         element.style.cursor = this.cursor_;
105       }
106     }
107     else if (this.previousCursor_ !== undefined){
108       element.style.cursor = this.previousCursor_;
109       this.previousCursor_ = undefined;
110     }
111   }
112 };
113 window.app.Drag.prototype.handleUpEvent = function(evt) {
114   this.coordinate_ = null;
115   this.feature_ = null;
116   return false;
117 };
118 </script>
119 </head>
120 <body onload="init();">
121   <div id="map_canvas">
122     <div id="popup"></div>
```

```
123   </div>
124 </body>
125 </html>
```

# LeafLet

El código en **LeafLet** es un código muy similar en tamaño al generado por **Google Maps** pero llegando a ser más simple e intuitivo. La figura 4 muestra una captura con el resultado.

**Listing 4**: Código en LeafLet

```
1  <!DOCTYPE html>
2  <html><head><title>Mapa con LeafLet</title>
3  <script src="js/leaflet-0.7/leaflet.js"></script>
4  <link rel="stylesheet" href="js/leaflet-0.7/leaflet.css" />
5  <style type="text/css">
6  html, body{ margin: 0px; padding: 0px; }
7  html, body, #map_canvas { width: 100%; height: 100%;}
8  </style>
9  <script type="text/javascript">
10 var infowindow = L.popup();
11 var posicion = [39.474016, -0.377382];
12 function init() {
13   var map = L.map("map_canvas").setView(posicion, 12);
14   L.tileLayer("http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png", {
15     attribution: "Map data &copy; <a href=\"http://openstreetmap.org\">OpenStreetMap<
         /a> contributors, <a href=\"http://creativecommons.org/licenses/by-sa/2.0/\">
         CC-BY-SA</a>, Imagery <a href=\"http://cloudmade.com\">CloudMade</a>"
16   }).addTo(map);
17
18   var marker = L.marker(posicion, {
19     draggable: true,
20     title: "un marker de ejemplo"
21   }).addTo(map);
22
23   marker.on("click", function(event){
24     infowindow.setLatLng(event.latlng)
25       .setContent("Marker: " + event.target.options.title + "<br/>Coordenadas: " +
           marker.getLatLng().lng + "," + marker.getLatLng().lat)
26       .openOn(map);
27   });
28 }
29 </script>
30 </head>
31 <body onload="init();">
32   <div id="map_canvas"></div>
33 </body>
34 </html>
```

# ModestMaps

A continuación se muestra el código generado para **ModestMaps** (la figura 5 recoge una captura de pantalla):

**Listing 5**: Código en ModestMaps

```
1  <!DOCTYPE html>
2  <html><head><title>Mapa con ModestMaps</title>
3  <script src="js/modestmaps.min.js"></script>
4  <style type="text/css">
5  html, body{ margin: 0px; padding: 0px; }
6  html, body, #map_canvas{ width: 100%; height: 100%;}
7  img.report{}
8  </style>
```

```
<script type="text/javascript">
var map;
var posicion = [39.474016, -0.377382];
function init(){
  var layer = new MM.TemplatedLayer("http://tile.openstreetmap.org/{Z}/{X}/{Y}.png");
  map = new MM.Map('map_canvas', layer);
  map.setCenterZoom(new MM.Location(posicion[0], posicion[1]), 12);
  var location = new MM.Location(posicion[0], posicion[1]);
  new Marker(map, location, {name: "un marker de ejemplo"})
}
Marker = function(map, location, content){
  this.coord = map.locationCoordinate(location);
  this.dimensions = new com.modestmaps.Point(20, 20);
  this.offset = new com.modestmaps.Point(this.dimensions.x/2, -this.dimensions.y/2);

  var follower = this;
  var callback = function(m, a) { return follower.draw(m); };
  map.addCallback('panned', callback);
  map.addCallback('zoomed', callback);
  map.addCallback('centered', callback);
  map.addCallback('extentset', callback);

  this.div = document.createElement('div');
  this.div.style.position = 'absolute';
  this.div.style.width = this.dimensions.x + 'px';
  this.div.style.height = this.dimensions.y + 'px';

  var img = document.createElement("img");
  img.style.position = "absolute";
  img.src = "js/openlayers-v3.4.0-dist/images/marker-icon.png";
  this.div.appendChild(img);
  map.parent.appendChild(this.div);
  this.draw(map);
}

Marker.prototype = {
  div: null,
  coord: null,
  offset: null,
  dimensions: null,
  margin: null,
  draw: function(map){
    try {
      var point = map.coordinatePoint(this.coord);
    }catch(e){  return; }

    if(point.x + this.dimensions.x + this.offset.x < 0){
      this.div.style.display = 'none';
    }
    else if(point.y + this.dimensions.y + this.offset.y < 0){
      this.div.style.display = 'none';
    }
    else if(point.x + this.offset.x > map.dimensions.x){
      this.div.style.display = 'none';
    }
    else if(point.y + this.offset.y > map.dimensions.y){
      this.div.style.display = 'none';
    }
    else{
      this.div.style.display = 'block';
      this.div.style.left = point.x + this.offset.x + 'px';
      this.div.style.top = point.y + this.offset.y + 'px';
    }
  },
};
</script>
</head>
<body onload="init();">
  <div id="map_canvas"></div>
</body>
</html>
```

# PolyMaps

Finalmente, la figura 6 muestra el resultado generado por el siguiente código para **PolyMaps**:

**Listing 6**: Código en PolyMaps

```
<!DOCTYPE html>
<html><head><title>Mapa con PolyMaps</title>
<script src="js/simplegeo-polymaps-2265203/polymaps.min.js"></script>
<style type="text/css">
html, body{ margin: 0px; padding: 0px;  }
html, body, div#map_canvas, svg.map { width: 100%; height: 100%;  }
svg { display: block; overflow: hidden; }
div#map_canvas{ position: relative; border: solid 4px #ccc; background: #eee; }
.compass .back { fill: #eee;  fill-opacity: .8; }
.compass .fore {  stroke: #999;  stroke-width: 1.5px; }
.compass rect.back.fore { fill: #999; fill-opacity: .3; stroke: #eee; stroke-width: 1
    px;
  shape-rendering: crispEdges;
}
.compass .direction { fill: none;}
.compass .chevron { fill: none; stroke: #999; stroke-width: 5px; }
.compass .zoom .chevron { stroke-width: 4px; }
.compass .active .chevron, .compass .chevron.active {
  stroke: #fff;
}
.compass.active .active .direction { fill: #999; }
</style>
<script type="text/javascript">
var tips={};
var posicion = [39.474016, -0.377382];
var po = org.polymaps;
function init() {
  var map = po.map().container(document.getElementById("map_canvas").appendChild(
      po.svg("svg")))
    .center({lat: posicion[0], lon: posicion[1]})
    .zoom(12)
    .add(po.interact());

  map.add(po.image().url(po.url("http://{S}tile.openstreetmap.org" + "/{Z}/{X}/{Y}.
      png")
          .hosts(["a.","b.","c.",""])));

  map.add(po.compass().pan("none"));
  var marker = {"id":1,
    "properties": {"name": "un marker de ejemplo"},
    "geometry": {"type": "Point", "coordinates": [posicion[1], posicion[0] ]}
  };
  map.add(po.geoJson().on("load", loadLayer).features([marker]));
}
function loadLayer(layer){
  for(var i = 0; i < layer.features.length; i++) {
    var f = layer.features[i];
    var g = f.element = po.svg("image");
    g.setAttributeNS(po.ns.xlink, "href", "js/openlayers-v3.4.0-dist/images/
        marker-icon.png");
    g.setAttribute("width", 32);
    g.setAttribute("height", 32);
    g.setAttribute("x", -16);
    g.setAttribute("y", -16);
    g.setAttribute("transform", f.element.getAttribute("transform"));
    f.element.parentNode.replaceChild(g, f.element);
  }
}
</script>
</head>
<body onload="init();">
  <div id="map_canvas"></div>
</body>
</html>
```

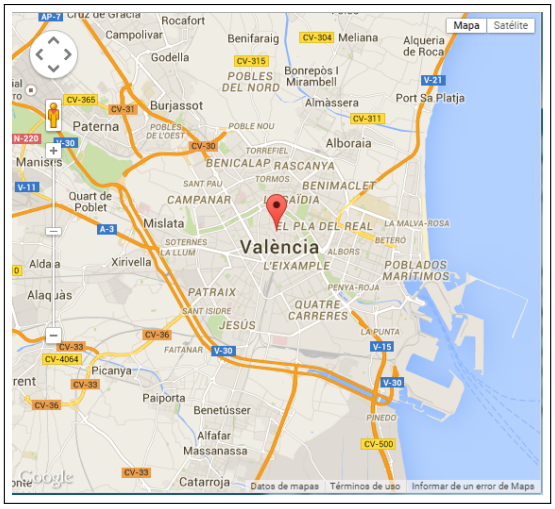


**Figura 1**: Resultado con Google Maps

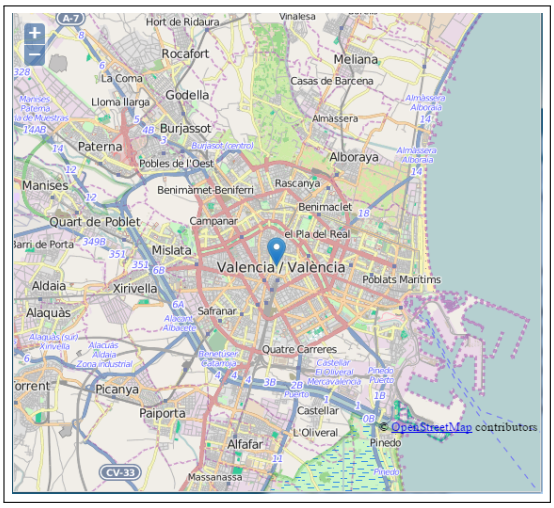


**Figura 2**: Resultado con OpenLayers v2

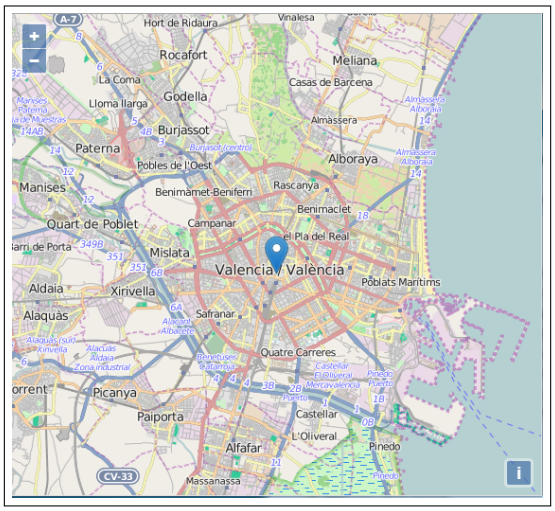


**Figura 3**: Resultado con OpenLayers v3

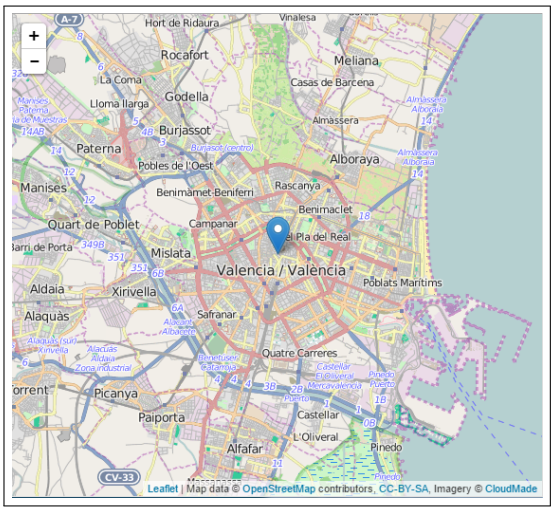


**Figura 4**: Resultado con LeafLet

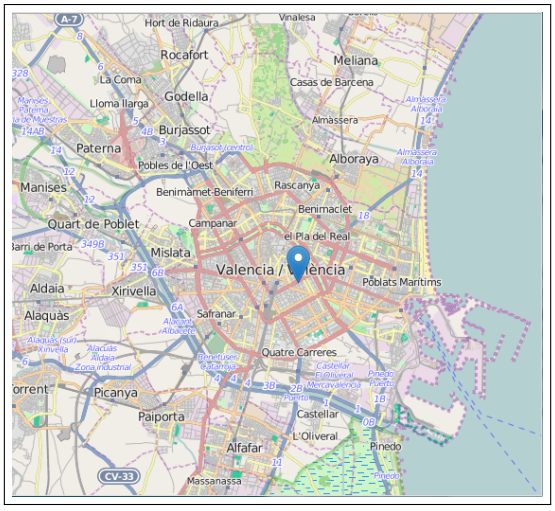


**Figura 5**: Resultado con ModestMaps

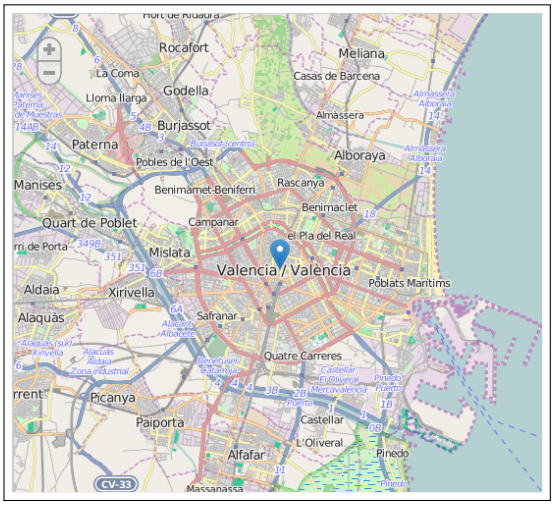


**Figura 6**: Resultado con Polymaps

# Conclusiones

Al realizar este ejercicio, he de reconocer que la librería **LeafLet** me ha causado una muy grata sorpresa: parece muy intuitiva y no he tenido que dedicarle mucho más de 30

minutos para realizar el ejercicio y navegar por su documentación.

Por otra parte, la experiencia de **PolyMaps** y **ModestMaps** ha sido bastante decepcionante por mi parte:

- Entre las dos, han consumido varias horas para pode realizar el ejercicio.
- En ninguno de los 2 casos se ha podido finalizar un *marker* clicable.
- ModestMaps no implementa controles de mapas y además necesita implementar clases adicionales.
- Polymaps requiere además modificar código CSS para poder visualizar correctamente.

Finalmente, **OpenLayers**, a pesar de contar con una amplia comunidad y una buena documentación, llama la atención la cantidad de código que hay que escribir para realizar el mismo resultado. Además, el cambio entre versiones es notorio, llegando a ser más sencillo el uso de la versión 2.

# Anexos

# Script para obtener los centros de salud

El script que se muestra a continuación está escrito en Javascript y tiene como finalidad obtener por la consola del navegador web la información de los centros sanitarios de la Generalitat de Catalunya.

## Código

Se ha omitido la inclusión de las librerías Javascript, pero es necesario utilizar *jQuery*.

**Listing 7**: Código para obtener los centros de salud por región sanitaria

```
/**
 * Script para extraer los centros de salud de la pagina
 * http://catsalut.gencat.cat/ca/ciutadania/centres-sanitaris/cercador/
 * @language: javascript
 */
var tipoCentro, rs;
var DELIMITER = ";";
$("div.mapa_nom_tipuscentre").each(function(){
  tipoCentro = $(this).text().trim();
  var contenidoTipoCentro = $(this).next();
  var nombreComarca, nombreCentro, regionSanitaria;
  var nombrePoblacion, nombreCentro, direccion, localidad, telefono, horario;

  contenidoTipoCentro.find("div").each(function(){
    if($(this).hasClass("mapa_nom_rs"))
      rs = $(this).text().trim();
    else if($(this).hasClass("mapa_nom_comarca"))
      nombreComarca = $(this).text().trim();
    else if($(this).hasClass("mapa_nom_poblacio"))
      nombrePoblacion = $(this).text().trim();
    else if($(this).hasClass("mapa_nom_nomcentre"))
      nombreCentro = $(this).text().trim();
    else if($(this).hasClass("mapa_content_nomcentre")){
      var detalles = $(this).html().trim();
      var tmp = detalles.trim().split("<br>");
      direccion = tmp[0].trim().replace("\r", "").replace("\n", "");
      localidad = tmp[1].trim().replace("\r", "").replace("\n", "");
      try{
        telefono = tmp[2].trim().split("</strong>")[1].trim();
      }catch(err){}
    }
    else if($(this).hasClass("mapa_horari_text")){
      horario = $(this).text().trim();
      console.log(tipoCentro + DELIMITER + rs + DELIMITER + nombreComarca + DELIMITER
           + nombrePoblacion + DELIMITER + nombreCentro + DELIMITER + direccion +
          DELIMITER + localidad + DELIMITER + telefono + DELIMITER + horario);
    }
  });
});
```

# Ejecución del script

Para ejecutar el script es necesario entrar en la página del buscador [2] con un navegador que disponga de una consola de Javascript como es el caso de *Google Chrome* o *Mozilla Firefox*.

A continuación se buscan todos los centros sanitarios de una Región Sanitaria y se abre la consola y se ejecuta el código como muestra la figura 7.

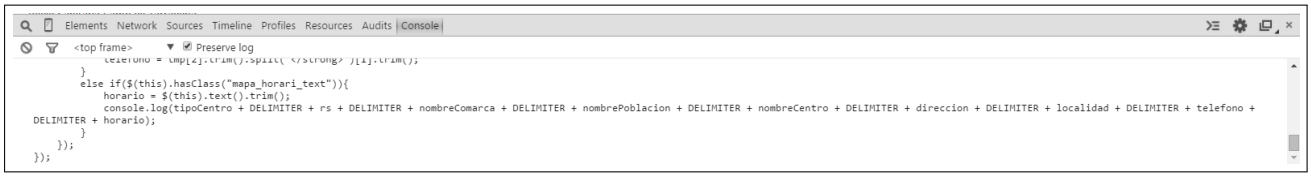


**Figura 7**: Ejemplo del código en la consola de Chrome

Al ejecutar el código, se genera la información de todos los centros sanitarios de la región seleccionada tal y como muestra la figura 8.

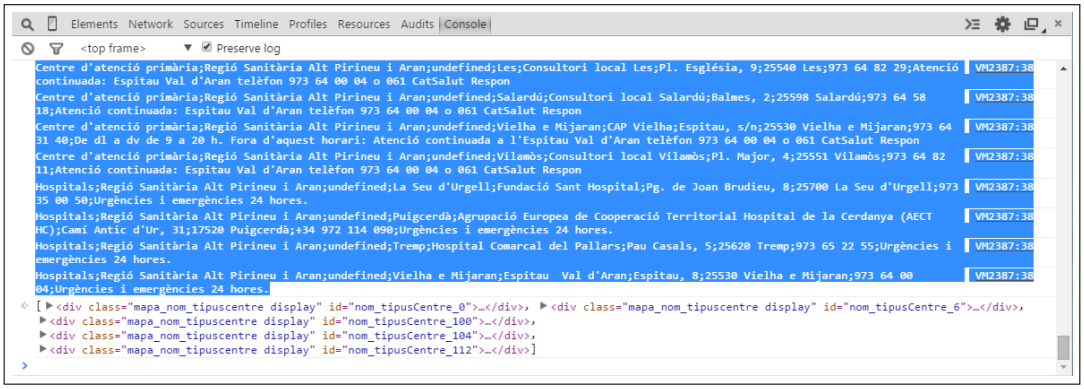


**Figura 8**: Ejemplo del output del script en la consola de Chrome

Para finalizar, sólo hay que copiar el código y pegarlo en un editor de textos.

[2] http://catsalut.gencat.cat/ca/ciutadania/centres-sanitaris/cercador/

# Anexos

# Geocoder en Java

## Código

El siguiente código hace uso de las librerías para Java de *Google* [3] (lo setters y getters se han omitido en favor de la mejor comprensión del código).

**Listing 8**: Código geocoder

```
1  import java.io.BufferedReader;
2  import java.io.BufferedWriter;
3  import java.io.FileInputStream;
4  import java.io.FileOutputStream;
5  import java.io.IOException;
6  import java.io.InputStreamReader;
7  import java.io.OutputStreamWriter;
8  import java.util.ArrayList;
9  import java.util.Collection;
10 import java.util.List;
11 import java.util.concurrent.TimeUnit;
12 import com.google.maps.GeoApiContext;
13 import com.google.maps.GeocodingApi;
14 import com.google.maps.GeocodingApiRequest;
15 import com.google.maps.model.GeocodingResult;
16
17 public class Geocoder extends AuthenticatedDirections {
18   private GeoApiContext context;
19   transient private final static String API_KEY = "XXXXXXXXXXXXXXXXXXXX";
20   transient private final static String USER = "XXXXXXX@gmail.com";
21   private final static String UTF8 = "UTF8";
22   private String region;
23   private String language;
24
25   public static void main(String []args){
26     new Geocoder().localizar("tmp/direcciones_PAC2.csv", ";");
27   }
28
29   public Geocoder(){
30     context = new GeoApiContext();
31     context.setApiKey(API_KEY).setEnterpriseCredentials(USER, API_KEY);
32     context.setQueryRateLimit(3).setConnectTimeout(1, TimeUnit.SECONDS)
33         .setReadTimeout(1, TimeUnit.SECONDS).setWriteTimeout(1, TimeUnit.SECONDS);
34     region = "es";
35     language = "es";
36   }
37
38   public void localizar(final String archivo, final String separator){
39     BufferedReader br = null;
40     try {
41       br = new BufferedReader(new InputStreamReader(new FileInputStream(archivo),
           UTF8));
42     }catch(Exception e) {
43       e.printStackTrace();
44     }
45
46     String linea = null;
47     int numLinea = 0;
48     List<Fila> filas = new ArrayList<Fila>();
49     BufferedWriter bw = null;
50     try {
```

[3] https://developers.google.com/maps/documentation/webservices/client-library

```
51        bw = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(archivo + "
              .OUT.csv"), UTF8));
52      }catch(Exception e1) {
53        e1.printStackTrace();
54      }
55
56      if(bw == null){
57        System.err.println("Ha habido un error accediendo a archivo de destino");
58        System.exit(1);
59      }
60
61      System.out.println("[Leyendo direcciones] Empezando");
62      try {
63        while(br != null && br.ready() && (linea = br.readLine()) != null){
64          filas.add(new Fila(++numLinea, linea, separator));
65        }
66      }catch(Exception e){}
67      try{
68        br.close();
69      }catch(Exception e){}
70      System.out.println("[Leyendo direcciones] Se han leido " + filas.size() + "
            direcciones");
71
72
73      while(!filas.isEmpty()){
74        Fila fila = filas.remove(0);
75        GeocodingApiRequest geoApi = GeocodingApi.newRequest(context)
76            .address(fila.getCalle() + ", " + fila.getMunicipio() + ", " + fila.
                  getProvincia())
77            .language("ES");
78        GeocodingResult[] resultados = null;
79        try {
80          resultados = geoApi.await();
81          if(resultados[0] != null){
82            if(resultados[0].geometry != null && resultados[0].geometry.location !=
                  null){
83              fila.setX(resultados[0].geometry.location.lat).setY(resultados[0].
                      geometry.location.lng)
84                .setAproximacion(resultados[0].geometry.locationType.name());
85            }
86            fila.setDireccionGoogle(resultados[0].formattedAddress);
87          }
88          bw.write(fila.getLineaTexto() + ";" + fila.getDireccionGoogle() + ";" + fila.
                getX() + ";" + fila.getY() + ";" + fila.getAproximacion() + "\r\n");
89        }catch (Exception e) {
90          e.printStackTrace();
91        }
92      }
93
94      try {
95        bw.close();
96      } catch (IOException e) {
97        e.printStackTrace();
98      }
99    }
100 }
101
102 class AuthenticatedDirections {
103   // Esta parte esta extraida de la documentacion y ejemplos de Google
104   public static Collection<Object[]> contexts() {
105     Collection<Object[]> contexts = new ArrayList<Object[]>();
106
107     // Travis can't run authorized tests from pull requests.
108     // http://docs.travis-ci.com/user/pull-requests/#Security-Restrictions-when-
            testing-Pull-Requests
109     if (System.getenv("TRAVIS_PULL_REQUEST") != null && !"false".equals(System.getenv
            ("TRAVIS_PULL_REQUEST"))) {
110       return contexts;
111     }
112     if (System.getenv("API_KEY") != null) {
113       GeoApiContext context = new GeoApiContext().setApiKey(System.getenv("API_KEY"))
              ;
114       contexts.add(new Object[] { context });
```

```
115     }
116     if (!(System.getenv("CLIENT_ID") == null || System.getenv("CLIENT_SECRET") ==
            null)) {
117       GeoApiContext context = new GeoApiContext().setEnterpriseCredentials(System.
              getenv("CLIENT_ID"),System.getenv("CLIENT_SECRET"));
118       contexts.add(new Object[] { context });
119     }
120     if (contexts.size() == 0) {throw new IllegalArgumentException("No credentials
          found! Set the API_KEY or CLIENT_ID and CLIENT_SECRET environment variables
          to run tests requiring authentication.");
121     }
122     return contexts;
123   }
124   protected AuthenticatedDirections() {}
125 }
126
127 class Fila{
128   private int linea;
129   private String id;
130   private String patio;
131   private String provincia;
132   private String municipio;
133   private String calle;
134   private double x;
135   private double y;
136   private String aproximacion;
137   private String direccionGoogle;
138   private String lineaTexto; // la linea tal cual (raw)
139   public Fila(final int linea, final String lineaTexto, final String separator){
140     this.linea = linea;
141     this.setLineaTexto(lineaTexto);
142     if(lineaTexto != null){
143       String campos[] = lineaTexto.split(separator);
144       if(campos != null){
145         try{
146           this.id = campos[0];
147           this.patio = campos[1];
148           this.provincia = "Catalunya"; //campos[3]
149           this.municipio = campos[6];
150           this.calle = campos[5];
151         }catch(Exception e){ System.err.println(e);}
152       }
153     }
154   }
155 }
```

# Anexos

# Vistas SQL

## Vistas de los indicadores de salud

**Listing 9**: Creación de las vistas utilizadas para los indicadores de salud

```sql
-- Vista para saber que indicadores existen en la aplicacion, no tiene campo geometrico
CREATE OR REPLACE VIEW indicadores.v_indicadores_disponibles AS
 SELECT id, titulo, tipo, anyo, unidades, explicacion,
        CASE indicadores.tipo
            WHEN 'municipio'::text THEN 'v_indicadores_municipios'::text
            WHEN 'comarca'::text THEN 'v_indicadores_comarcas'::text
            WHEN 'provincia'::text THEN 'v_indicadores_provincias'::text
            WHEN 'region sanitaria'::text THEN 'v_indicadores_rs'::text
            ELSE NULL::text
        END AS vista
 FROM indicadores.indicadores
 ORDER BY tipo, titulo, anyo;


-- Vista para obtener los indicadores sanitarios de los municipios
CREATE OR REPLACE VIEW indicadores.v_indicadores_municipios AS
 SELECT a.id, a.id_indicador, a.id_municipio, a.valor, b.geom, b.nomn_muni AS nombre,
      c.titulo, c.anyo, c.tipo
 FROM indicadores.indicadores_municipios a, cartografia.municipios b, indicadores.
     indicadores c
 WHERE a.id_municipio = b.gid AND a.id_indicador = c.id;


-- Vista para obtener los indicadores sanitarios de las comarcas
CREATE OR REPLACE VIEW indicadores.v_indicadores_comarcas AS
 SELECT a.id, a.id_indicador, c.titulo, c.anyo, c.tipo, a.id_comarca, a.valor, b.geom
     , b.nom_comar AS nombre
 FROM indicadores.indicadores_comarcas a, cartografia.comarcas b, indicadores.
     indicadores c
 WHERE a.id_comarca = b.gid AND a.id_indicador = c.id;


-- Vista para obtener los indicadores sanitarios de las provincias
CREATE OR REPLACE VIEW indicadores.v_indicadores_provincias AS
 SELECT a.id, a.id_indicador, a.id_provincia, a.valor, b.geom, b.nom_prov AS nombre,
     c.titulo, c.anyo, c.tipo
 FROM indicadores.indicadores_provincias a, cartografia.provincias b, indicadores.
     indicadores c
 WHERE a.id_provincia = b.gid AND a.id_indicador = c.id;


-- Vista para obtener los indicadores sanitarios de las regiones sanitarias
CREATE OR REPLACE VIEW indicadores.v_indicadores_rs AS
 SELECT a.id, a.id_indicador, a.id_rs, a.valor, b.geom, b.nombre, c.titulo, c.anyo, c
     .tipo
 FROM indicadores.indicadores_rs a, siscat.regiones_sanitarias b, indicadores.
     indicadores c
 WHERE a.id_rs = b.id AND a.id_indicador = c.id;
```

## Vistas de los centros sanitarios

**Listing 10**: Creación de los centros sanitarios

```
-- Vista que muestra los centros sanitarios con todos sus detalles
CREATE OR REPLACE VIEW siscat.v_centros_sanitarios AS
 SELECT a.gid, b.nom_comar AS comarca, c.nom_muni AS muni, e.nombre AS rs, d.nom_prov
      AS provincia, a.id_tipo_centro, f.tipo AS tipo_centro, a.nombre_centro AS
     nombre, a.direccion, a.telefono AS telf, a.observaciones, a.cp_y_municipio AS cp
     , a.geom
 FROM siscat.centros_sanitarios a
     LEFT JOIN cartografia.comarcas b ON a.id_comarca = b.gid
     LEFT JOIN cartografia.municipios c ON a.id_municipio = c.gid
     LEFT JOIN cartografia.provincias d ON a.id_provincia = d.gid
     LEFT JOIN siscat.regiones_sanitarias e ON a.id_rs = e.id
     LEFT JOIN siscat.tipos_centro f ON a.id_tipo_centro = f.id;

-- Vista que muestra los centros sanitarios de forma simplificada
CREATE OR REPLACE VIEW siscat.v_centros_sanitarios_simple AS
SELECT gid::smallint AS gid, direccion,  cp_y_municipio, nombre_centro, geom
FROM siscat.centros_sanitarios;
```

# Vistas de para municipios y provincias

**Listing 11**: Creación de los centros sanitarios

```
-- Vistas utilizadas para crear mapas en Geoserver
CREATE OR REPLACE VIEW cartografia.v_capitales AS
 SELECT a.gid::bigint AS gid, a.tipus_cap, a.es_cap_pr, a.municipi, a.comarca, a.
     provincia, a.geom, b.gid::bigint AS id_municipio, b.nom_muni
  FROM cartografia.capitales a, cartografia.municipios b
  WHERE a.id_municipio = b.gid;

-- Vista utilizada para emplazar los textos de las provincias en el centroide en
    Geoserver
CREATE OR REPLACE VIEW cartografia.v_textos_provincias AS
 SELECT gid, nom_prov, st_centroid(geom) AS geom
 FROM cartografia.provincias;
```

# Anexos

# Estilos SLD de Geoserver

## Estilo de capitales

Este estilo se utiliza para representar como puntos los nombres de los municipios. Las 4 capitales de provincias aparecen con un estilo diferente y el resto de municipios aparecen en un zoom mayor.

**Listing 12**: Archivo SLD para capitales

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
 xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
 xmlns="http://www.opengis.net/sld"
 xmlns:ogc="http://www.opengis.net/ogc"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <NamedLayer>
    <Name>Capitales</Name>
    <UserStyle>
      <Title>capitales</Title>
      <Abstract>Capitales de Catalu\~na representadas con estrellas</Abstract>
    <FeatureTypeStyle>
     <Rule>
      <ogc:Filter>
        <ogc:PropertyIsEqualTo>
          <ogc:PropertyName>es_cap_pr</ogc:PropertyName>
          <ogc:Literal>N</ogc:Literal>
        </ogc:PropertyIsEqualTo>
      </ogc:Filter>
      <MaxScaleDenominator>300000</MaxScaleDenominator>
      <PointSymbolizer>
         <Graphic>
          <ExternalGraphic>
             <OnlineResource xlink:type="simple" xlink:href="http://localhost:8080/
                 icons/capital_small.png" />
             <Format>image/png</Format>
          </ExternalGraphic>
          <Size>16</Size>
         </Graphic>
       </PointSymbolizer>
     </Rule>
    </FeatureTypeStyle>
    <FeatureTypeStyle>
     <Rule>
      <ogc:Filter>
        <ogc:PropertyIsEqualTo>
          <ogc:PropertyName>es_cap_pr</ogc:PropertyName>
          <ogc:Literal>N</ogc:Literal>
        </ogc:PropertyIsEqualTo>
      </ogc:Filter>
      <MaxScaleDenominator>150000</MaxScaleDenominator>
       <TextSymbolizer>
         <Label>
           <ogc:PropertyName>nom_muni</ogc:PropertyName>
         </Label>
         <Font>
           <CssParameter name="font-family">Arial</CssParameter>
           <CssParameter name="font-size">10</CssParameter>
           <CssParameter name="font-style">normal</CssParameter>
         </Font>
```

```
        <LabelPlacement>
          <PointPlacement>
          <AnchorPoint>
            <AnchorPointX>0.5</AnchorPointX>
            <AnchorPointY>0.5</AnchorPointY>
          </AnchorPoint>
          <Displacement>
            <DisplacementX>4</DisplacementX>
            <DisplacementY>8</DisplacementY>
          </Displacement>
          </PointPlacement>
        </LabelPlacement>
        <Halo>
                 <Radius>2</Radius>
                   <Fill>
                   <CssParameter name="fill">#FFFFFF</CssParameter>
                 </Fill>
               </Halo>
        <Fill>
          <CssParameter name="fill">#000000</CssParameter>
        </Fill>
      </TextSymbolizer>
    </Rule>
  </FeatureTypeStyle>

<FeatureTypeStyle>
    <Rule>
     <ogc:Filter>
       <ogc:PropertyIsEqualTo>
         <ogc:PropertyName>es_cap_pr</ogc:PropertyName>
         <ogc:Literal>S</ogc:Literal>
       </ogc:PropertyIsEqualTo>
     </ogc:Filter>
     <PointSymbolizer>
        <Graphic>
         <ExternalGraphic>
            <OnlineResource xlink:type="simple" xlink:href="http://localhost:8080/
                icons/capital_big_highlight.png" />
            <Format>image/png</Format>
         </ExternalGraphic>
         <Size>24</Size>
        </Graphic>
      </PointSymbolizer>
      <TextSymbolizer>
        <Label>
          <ogc:PropertyName>nom_muni</ogc:PropertyName>
        </Label>
        <Font>
          <CssParameter name="font-family">Arial</CssParameter>
          <CssParameter name="font-size">12</CssParameter>
          <CssParameter name="font-style">normal</CssParameter>
          <CssParameter name="font-weight">bold</CssParameter>
        </Font>
        <LabelPlacement>
          <PointPlacement>
          <AnchorPoint>
            <AnchorPointX>0.5</AnchorPointX>
            <AnchorPointY>0.5</AnchorPointY>
          </AnchorPoint>
          <Displacement>
            <DisplacementX>4</DisplacementX>
            <DisplacementY>16</DisplacementY>
          </Displacement>
          </PointPlacement>
        </LabelPlacement>
        <Halo>
                 <Radius>2</Radius>
                   <Fill>
                   <CssParameter name="fill">#FFFFFF</CssParameter>
                 </Fill>
               </Halo>
        <Fill>
          <CssParameter name="fill">#000000</CssParameter>
```

```
123           </Fill>
124         </TextSymbolizer>
125       </Rule>
126     </FeatureTypeStyle>
127
128
129     </UserStyle>
130   </NamedLayer>
131 </StyledLayerDescriptor>
```

El resultado que genera es el siguiente:

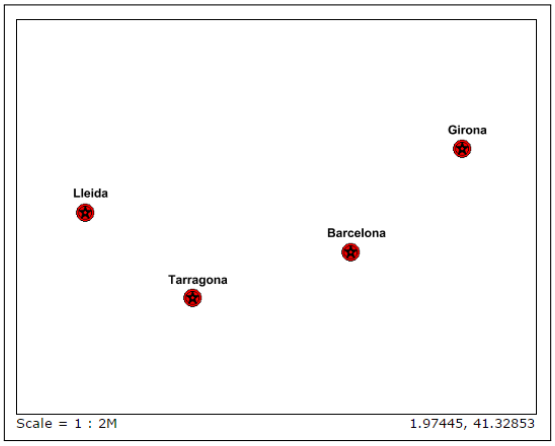


**Figura 9**: Muestra en escala 1:2M

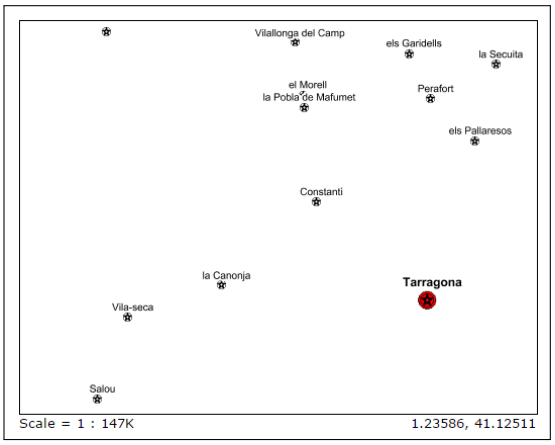


**Figura 10**: Muestra en escala 1:147K

# Anexos

# Script para generar los diagramas de Voronoi

El siguiente código Javascript genera los diagramas de Voronoi de los centros de salud cargados previamente de Geoserver.

El script lee los centros mediante una llamada WFS a Geoserver, genera los políogonos que representan sus diagramas de Voronoi (utilizando la libreria *D3.js*) y finalmente muestra en el cuerpo de la página web el contenido GeoJSON que se puede guardar como un archivo de texto desde el navegador.

## Script Voronoi JS

**Listing 13**: Función *solicitarGeoJson*

```
var centrosSanitarios = [];
var poligonosVoronoi = [];

/* Obtiene los centros sanitarios en GeoJSON y genera markers */
function cargarCentrosSanitarios(){
  $.ajax({
    // Solicito al WFS los centros sanitarios cuyo id_tipo_centro=2 (centros de atenci\'on primaria)
    url: "/geoserver/TFM_ATLAS/ows?service=WFS&version=1.1.0&request=GetFeature&typeName=TFM_atlas%3Av_centros_sanitarios_simple&CQL_FILTER=id_tipo_centro%3D2&outputFormat=json&SrsName=EPSG%3A4326&format_options=callback%3AgetJson",
    dataType: "json",
    jsonpCallback: "getJson",
    success: function(jsonData){
      var icon = L.icon({iconUrl: "img/marker.png"});
      jsonData.features.forEach(function(f){
        var latLng = L.latLng(f.geometry.coordinates[1],f.geometry.coordinates[0])
        centrosSanitarios.push(L.marker(latLng, {icon: icon}));
      });
    }
  });
}

/* Genera los diagramas de Voronoi del array de markers "centros sanitarios" */
function generarVoronoi(){
  var voronoi = d3.geom.voronoi().x(function(d) {return d.getLatLng().lng}).y(function(d) {return d.getLatLng().lat})
  var voronoiPol = voronoi(centrosSanitarios);

  voronoiPol.forEach(function(p){
    var coordinates = [];
    p.forEach(function(d){ coordinates.push(L.latLng(d[1], d[0])); });
    var poligono = L.polygon(coordinates);
    poligonosVoronoi.push(poligono);
  });
}

 /* Carga los diagramas de Voronoi y espera 3 segundos para darle tiempo a procesarlo (evito que la funcion de Voronoi sin tener las geometrias de los centros sanitarios)
 Finalmente muestra en pantalla un documento GeoJSON con los diagramas de Voronoi
 */
```

```
37 cargarCentrosSanitarios();
38 window.setTimeout(function(){
39   generarVoronoi();
40
41   $("body").html('{"type":"FeatureCollection","features":[');
42   for(var i = 0; i < poligonosVoronoi.length; i++){
43     $("body").append(JSON.stringify(poligonosVoronoi[i].toGeoJSON()));
44     if(i < poligonosVoronoi.length - 1)
45         $("body").append(",");
46     $("body").append("</br>");
47   }
48   $("body").append(']},"crs":{"type":"name","properties":{"name":"urn:ogc:def:crs:EPSG
       ::4326"}}}');
49 }, 3000);
```

# Salida en el navegador

El punto de partida es abrir un navegador de internet (preferiblemente *Chrome* o *Firefox*) y abrir la *consola de Javascript*. En la consola se pega el código Función *solicitarGeoJson* y se ejecuta como muestra la figura 11.


**Figura 11**: Ejecución del código en la consola de Chrome

Una vez que tenemos la salida como se muestra en la figura 12, se guarda el archivo como un archivo con extensión *".json"* y se abre en un editor de GIS como *QGIS*, como muestra la figura 13.

El último paso es guardar el archivo como un shapefile e importar a *Postgis*

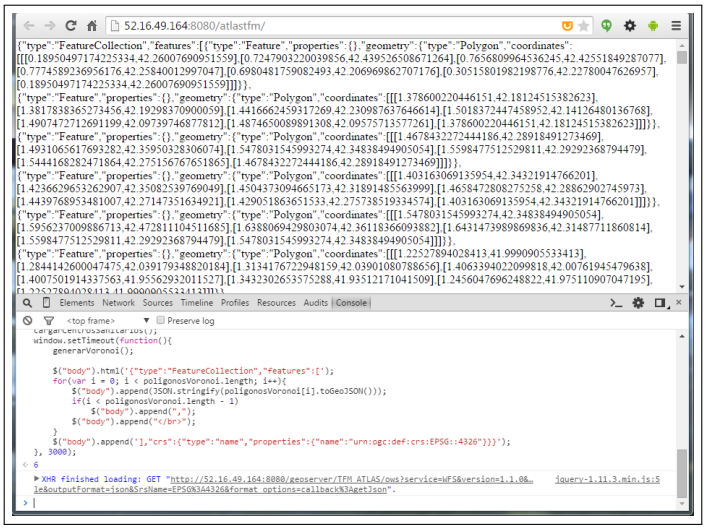


**Figura 12**: Resultado del código

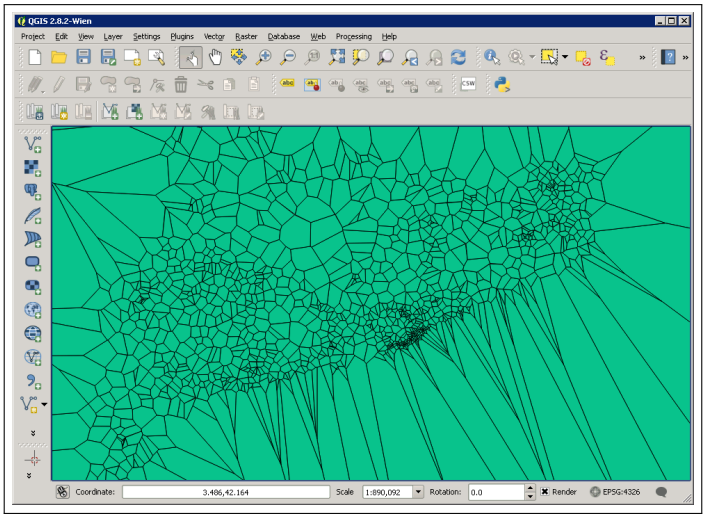


**Figura 13**: Archivo GeoJSON en QGIS

# Anexos

# SQL para generar los diagramas de Voronoi

Con los polígonos de Voronoi insertados en una tabla temporal de la base de datos (`temp.voronoi`) se debe realizar una serie de funciones para poder calcular la cobertura de los centros sanitarios.

## Polígonos de Voronoi

Al cargar los polígonos de Voronoi obtenemos unos polígonos que quedan de manera irregular (figura 14). Para mejorar la forma de esos polígonos se deben recortar aquellos polígonos que exceden la geografía de Cataluña para obtener el resultado que se muestra en la figura 15.

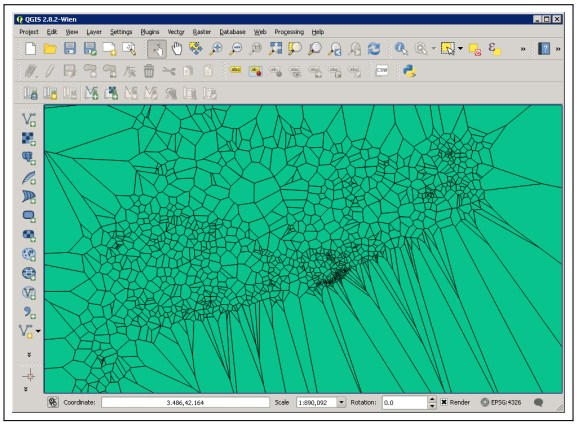

**Figura 14**: Vista en QGIS de los polígonos de Voronoi originales

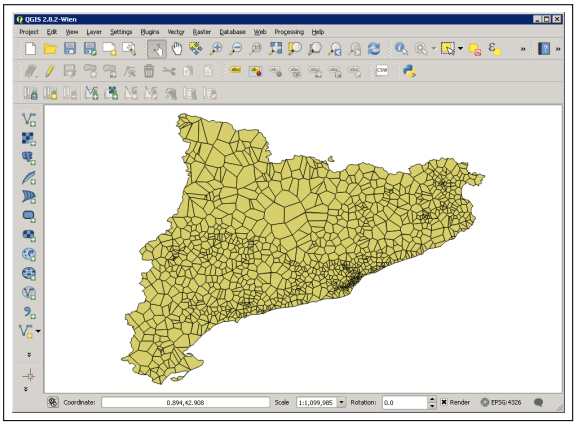

**Figura 15**: Vista en QGIS de los polígonos de Voronoi ajustados

Para lograr el resultado de la figura 15, se han insertado los polígonos utilizando la siguiente consulta SQL en Postgis:

**Listing 14**: Inserción de polígonos de Voronoi

```
1 -- Creo la tabla con los poligons recortados al contorno de Catalunya
2 CREATE TABLE siscat.centros_sanitarios_voronoi AS
3 SELECT gid, st_intersection(b.geom, a.geom) as geom FROM temp.voronoi b, (SELECT
      ST_UNION(geom) as geom FROM cartografia.provincias) a;
4
5 ALTER TABLE siscat.centros_sanitarios_voronoi ADD CONSTRAINT
      siscat_centros_sanitarios_voronoi_pk PRIMARY KEY(gid);
6 CREATE INDEX ssiscat_centros_sanitarios_voronoi_gist ON siscat.
      centros_sanitarios_voronoi USING gist (geom);
```

Con el siguiente código creo una tabla que contendrá para cada polígono de Voronoi las información de con qué municipios intersecan:

**Listing 15**: Tabla de intersecciones

```
-- Creo una tabla con las intersecciones entre Voronoi y los municipios (descarto
    poligonos con interseccion menor al 1 por cien
CREATE TABLE siscat.voronoi_intersecciones AS
select a.gid AS id_voronoi, b.gid AS id_municipio, b.poblacion * st_area(
    st_intersection(b.geom, a.geom))/st_area(b.geom) AS poblacion_municipio, st_area(
    st_intersection(b.geom, a.geom))/st_area(b.geom) * 100 AS porcentaje_area
FROM siscat.centros_sanitarios_voronoi a, cartografia.municipios b
WHERE a.geom && b.geom AND st_intersects(a.geom, b.geom) AND st_area(st_intersection(
    b.geom, a.geom))/st_area(b.geom) * 100 > 0.9;

-- Creacion de claves ajenas y principal
ALTER TABLE siscat.voronoi_intersecciones ADD COLUMN gid serial;
ALTER TABLE siscat.voronoi_intersecciones ADD CONSTRAINT
    siscat_voronoi_intersecciones_pk PRIMARY KEY(gid);
ALTER TABLE siscat.voronoi_intersecciones ADD CONSTRAINT
    siscat_voronoi_intersecciones_fk1 FOREIGN KEY (id_voronoi)  REFERENCES siscat.
    centros_sanitarios_voronoi (gid) MATCH SIMPLE ON UPDATE CASCADE ON DELETE CASCADE
    ;
ALTER TABLE siscat.voronoi_intersecciones ADD CONSTRAINT
    siscat_voronoi_intersecciones_fk2 FOREIGN KEY (id_municipio)  REFERENCES
    cartografia.municipios (gid) MATCH SIMPLE ON UPDATE CASCADE ON DELETE CASCADE;
```

Finalmente, se construyen las 2 vistas que se consultarán en Geoserver: la primera vista contiene los polígonos de Voronoi y el nombre del Centro de Atención Primaria que le corresponde (este valor es un array porque se dan casos que un mismo polígono contiene más de un Centro de Atención Primaria).

**Listing 16**: Tabla de intersecciones

```
-- Creacion de la vista que devolvera los poligonos de Voronoi y los Centros de salud
     que estan dentro
CREATE OR REPLACE VIEW siscat.v_poligonos_voronoi AS
SELECT a.*, array_agg(b.nombre_centro)::text as nombre_centro, array_agg(b.gid)::text
     as id_centro
  FROM siscat.centros_sanitarios_voronoi a, siscat.centros_sanitarios b
  WHERE a.geom ~ b.geom AND st_contains(a.geom, b.geom) AND b.id_tipo_centro = 2
  GROUP BY a.gid;
```

La segunda vista no tiene información geográfica, contiene los valores de cada polígono de Voronoi. Si un polígono de Voronoi interseca con varios municipios, aparecerán tantos registros como municipios intersequen con el polígono de Voronoi:

**Listing 17**: Tabla de intersecciones

```
-- Creacion de la vista que devolvera la informacion adicional de los poligonos (no
    incluye geometria)
CREATE VIEW siscat.v_voronoi_datos_json AS
  SELECT id_voronoi, id_municipio, poblacion_municipio, porcentaje_area, b.municipi,
      b.nom_muni, b.poblacion
  FROM siscat.voronoi_intersecciones a, cartografia.municipios b
  WHERE a.id_municipio = b.gid
  ORDER BY id_voronoi;
```

**AJAX** *Asynchronous Javascrip And XML*, técnica de desarrollo web ejecutada en el navegador para intercambiar datos con el servidor de manera asíncrona sin tener que recargar la página.

**GNIG** Centro Nacional de Información Geográfica.

**ICGC** Institut Cartogràfic i Geodòsic de Catalunya.

**JSON** *JavaScript Object Notation*, estándar abierto y ligero para el intercambio de datos alterntivo a XML.

**KML** *Keyhole Markup Language*, esquema XML utilizado para la representación de datos geográficos.

**OGC** *Open Geospatial Consortium* Consorcio internacional compuesto por empresas públicas y privadas con la finalidad de crear y mantener estandareas abiertos SIG.

**Shapefile** Formato de archivo informático que guarda datos espaciales y que consituye un estándar de intercambio de datos geométricos.

**SLD** *Styled Layer Descriptor*, esquema XML utilizado para definir un estilo visual de las capas en los mapas.

**SQL** *Structured Query Language*, lenguaje declarativo empleado en el acceso a base de datos relacionales.

**W3C** *World Wide Web Consortium*, organización internacional que produce recomendaciones y estándares para la World Wide Web.

**WCS** *Web Coverage Service*, servicio que permite realizar peticiones de consultas geográficas a través de la web.

**WFS** *Web Feature Service*, servicio estándar utilizado para proporcionar la información relativa a las entidades geométricas almacenadas.

**WMS** *Web Map Service*, protocolo estándar utilizado para servir datos georeferenciados. Generalmente genera datos en formato imagen.

Bibliografía

[1] Michael Bostock, Vadim Ogievetsky, and Jeffrey Hee. D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

[2] New Zealand Health Quality & Safety Comission. Atlas of healthcare variation methodology. Disponible en: http://www.hqsc.govt.nz/assets/Health-Quality-Evaluation/Atlas/DeprivationSF/Method_demography.pdf.

[3] De Pietri D, Dietrich P, Mayo P, Carcagno A, and De Titto E. Indicadores de accesibilidad geográfica a los centros de atención primaria para la gestión de inequidades. *Rev Panam Salud Publica*, 34(6):45260, 2013.

[4] Óscar Pastor David Roldán, Pedro J. Valderas. *Aplicaciones Web: un enfoque práctico*. Ra-Ma, 2010.

[5] Prat E., Sánchez J., Pesquer L., Olivet M., Aloy J., Fusté J., and Pons X. Estudio sobre la accesibilidad de los centros sanitarios públicos de cataluña. *XIII Congreso Nacional de Tecnologías de la Información Geográfica.*, 2008.

[6] Emiliano Aránguez Ruiz [et al.]. *Salud y territorio: aplicaciones prácticas de los sistemas de información geográfica para la salud ambiental*. Sociedad Española de Sanidad Ambiental, 2012.

[7] Departament de Salut Generalitat de Catalunya. *Atlas electrónico de salud de Cataluña: Sistema de Información Geográfica del Departamento de Salud*. 2006. Disponible en: http://www.gencat.es:8000/salut/depsalut/pdf/eseh6.pdf.

[8] Eric Hazzard. *OpenLayers 2.10 Begginer's Guide*. Apress, 2011.

[9] Paul S. Heckbert and Michael Garland. Survey of polygonal surface simplification algorithms. *School of Computer Science Carnegie Mellon University*.

[10] Leaflet. Documentación en línea. Disponible en: http://leafletjs.com/examples.html.

[11] José Carlos Marínez Llario. *PostGIS 2: Análisis Avanzado*. Universidad Politécnica de Valencia, 2012.

[12] Malcolm Maclean. *D3 Tips and Tricks*. LeanPub, 2013.

[13] Tom MacWright. Modestmaps + leaflet: new choices for web apis. Disponible en https://www.mapbox.com/blog/modest-maps-and-leaflet-new-choices-web-apis, 2011.

[14] Servicios Sociales e Igualdad de España Ministerio de Sanidad. Mapa de referencia para el sistema de información de atención primaria (siap). *Información y estadísticas sanitarias 2015*, 0, 2014.

[15] L. Monge de la Cruz, J. Torres Herrera, L. López Chico, and C. Navarro Cota. Análisis comparativo de servidores de mapas. 2010.

[16] OpenLayers. Documentación en línea. Disponible en: http://docs.openlayers.org/library.

[17] Albert Botella Plana, Anna Mu noz Bollas, Rosa Olivella González, Joan Carles Olmedillas, and Jesús Rodríguez Lloret. *Introducción a los sismteas de información geográfica y geotelemática*. Editorial UOC, 2011.

[18] Robert Roth, Richard Donohue, Carl Sack, Timothy Wallace, and Tanya Buckingham. A process for keeping pace with evolving web mapping technologies. *Cartographic Perspectives*, 0(78), 2015.

[19] Helen Thomas. 25 useful javascript libraries and tools for creating interactive maps. Disponible en http://www.instantshift.com/2013/08/26/useful-javascript-libraries-for-maps, 2013.