# Aplicación basada en web y en software libre para la gestión municipal de incidencias y actuaciones en el espacio público

Dionisio Martínez Soler

Universitat Oberta de Catalunya

*dmsoler@gmail.com*

6 de junio de 2016

# Índice

# Requisitos de la aplicación

## Requisitos funcionales:

- visualización de mapa del municipio y de sus infraestructuras;
- introducción de registros de incidencias y actuaciones geolocalizadas con un click sobre el mapa;
- visualización total o filtrada de incidencias registradas;
- modificación de incidencias registradas;
- exportación de incidencias a otras aplicaciones.

## Otros requisitos:

- interoperable y usando formatos estándar;
- basada en software libre;
- accesible en red, en intranet o en Internet;
- mapas y datos geográficos abiertos.

# Planteamiento de la solución

## Base de datos espaciales

Base de datos que siga el estándar
*Simple feature access - Part 2: SQL option* [OGC, 2010]:
implementa el tipo de datos `Geometry`.

PostGIS, SpatiaLite...

## Interfaz web

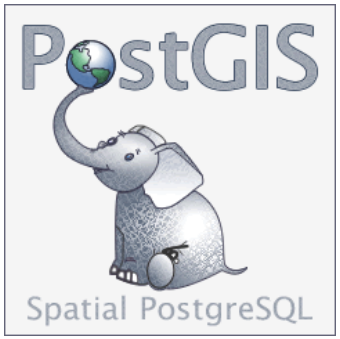
HTML+CSS+JavaScript

Bibliotecas JavaScript: Leaflet, OpenLayers, proj4js, jQuery...

## Datos geográficos abiertos

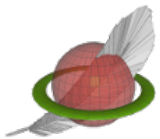
- Mapas: OpenStreetMap
- Ortofotos: PNOA
- Datos vectoriales sobre infraestructuras municipales: IDEPO

# Bases de datos espaciales




1. Estructura cliente-servidor $\Rightarrow$ requiere configurar y mantener un servidor.
2. Versátil, gran cantidad de funciones espaciales.
3. Eficaz en entornos multiusuario con muchos accesos simultáneos.

# Bases de datos espaciales



**SpatiaLite**: extensión espacial para SQLite

1. No utiliza estructura cliente-servidor ⇒ no requiere configuración y mantenimiento.
2. La base de datos está contenida en un fichero ⇒ copia y migración muy simple.
3. El código de SQLite está en el dominio público ⇒ interoperable.
4. Funciones espaciales suficientes para el proyecto.
5. Almacenamiento binario del tipo de datos `Geometry` con funciones para introducirlo o leerlo en formatos WKT, EWKT, KML, GML, y GeoJSON.
6. Gestionable con utilidades gráficas y de consola.
7. Gestionable en Python con un módulo específico.
8. Algunas limitaciones en entornos multiusuario con muchos accesos simultáneos.

# Diseño de una base de datos SpatiaLite

## Código SQL

```
CREATE TABLE Incidencias (
 id INTEGER NOT NULL PRIMARY KEY,
 tipo TEXT(50),
 radio INTEGER,
 inicio TEXT(22),
 fin TEXT(22)
);
```

| Campo | Contenido |
|---|---|
| id | getTime(): momento de creación de incidencia en milisegundos. |
| tipo | 50 caracteres de texto libre. |
| radio | Radio de área afectada en metros. |
| inicio | |
| fin | Fecha y hora en formato YYYY-MM-DDTHH:MM[+-]HH:MM |

# Diseño de una base de datos SpatiaLite

## Creación del campo de tipo Geometry

```
SELECT AddGeometryColumn(
  'Incidencias',
  'geometry',
   4326,
  'POINT',
  'XY',
   1
);
```

| | |
|---|---|
| 4326 | Coordenadas en EPSG:4326 (grados de lon/lat como GPS). |
| POINT | Tipo de objeto geográfico: punto. |
| XY | Coordenadas bidimensionales. |
| 1 | Campo NOT NULL: toda incidencia debe tener una localización. |

# Bibliotecas JavaScript: OpenLayers vs. Leaflet


1. Código sencillo.
2. Más ligera.
3. Menos versátil.
4. Funcionalidades de creación y edición de nuevos objetos geográficos vectoriales mediante *plugins* externos.

OpenLayers 3

1. Código más complejo.
2. Más pesada.
3. Más versátil.
4. Funcionalidades de creación y edición de nuevos objetos geográficos vectoriales incorporadas.

# Diseño de la interfaz web

- `panelmunicipal.html`
- `panelmunicipal.css`
- `panelmunicipal.js`

**Bibliotecas JavaScript:**


`OpenLayers` ⇒ mapas y datos geográficos.

`proj4js` ⇒ para usar en OpenLayers sistemas de referencia diferentes de EPSG:4326 y EPSG:3857.


`jQuery` ⇒ simplifica el código necesario para la obtención y manipulación de datos.

**Conexión a fuentes de datos geográficos (capas del mapa):**

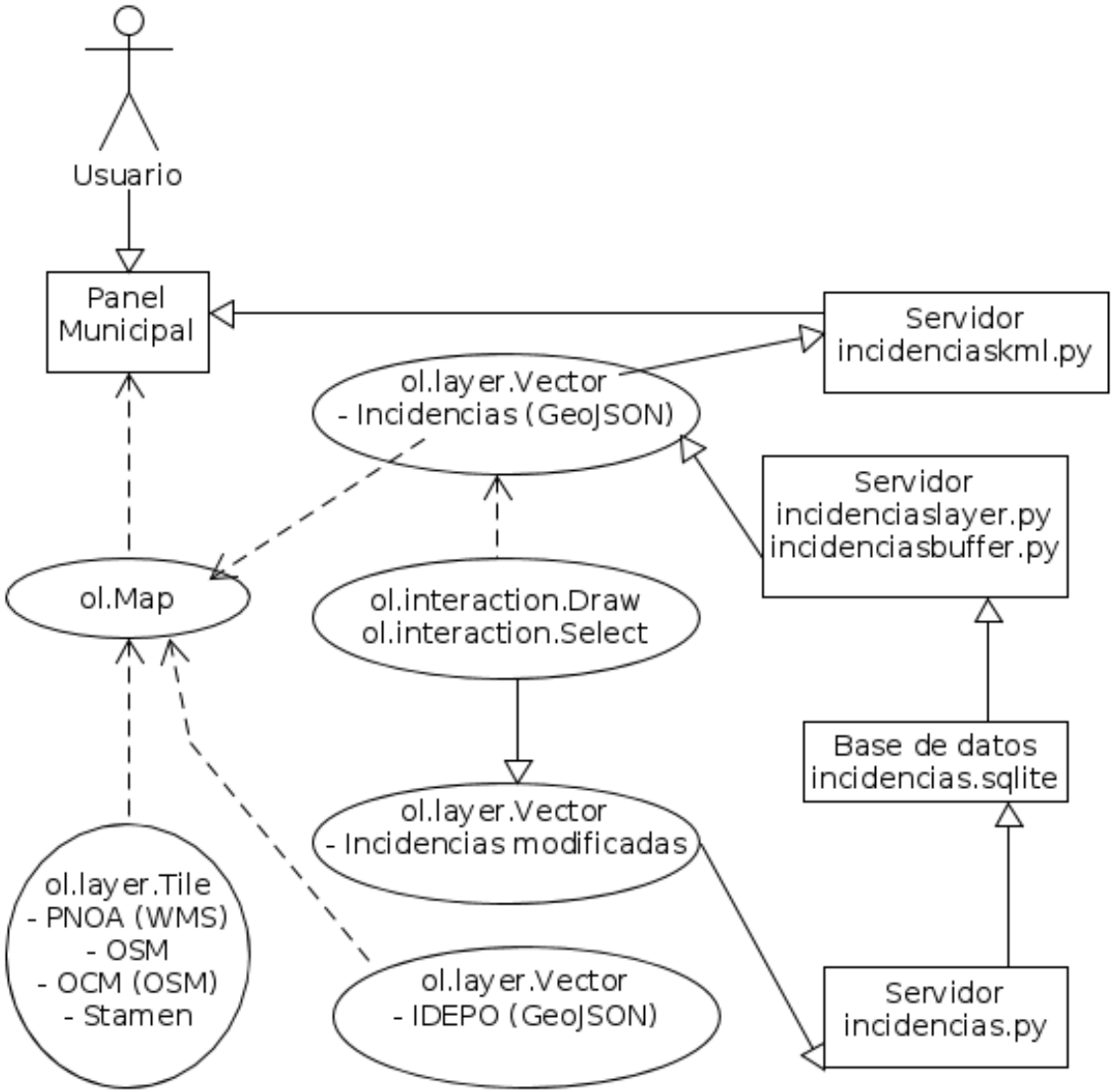

`OpenStreetMap` `http://www.openstreetmap.org`


`OpenCycleMap` `http://www.opencyclemap.org`


`PNOA` `http://pnoa.ign.es`

`IDEPO` `http://ide.depo.es`

# Esquema de funcionamiento de la aplicación



# Conexión de la interfaz web a la base de datos

## Inspirada en...

"A dynamic web application – OpenLayers AJAX with Python and SpatiaLite" [Mearns, 2015]

## *Scripts* CGI en lenguaje de programación `Python`

Usando los módulos:

- `PySpatialite`: lectura y escritura en la base de datos.
- `geojson`: intercambio de datos con la interfaz web.

| | |
|---|---|
| **incidencias.py** | interfaz → GeoJSON ⇒ SpatiaLite → BD |
| **incidenciaslayer.py** | BD → SpatiaLite ⇒ GeoJSON → interfaz |
| **incidenciasbuffer.py** | BD → SpatiaLite (Buffer) ⇒ GeoJSON → interfaz |
| **incidenciaskml.py** | interfaz → KML ⇒ fichero → interfaz. |

**Buffer:** Transform(Buffer(Transform(geometry, 23029), radio), 4326)

# Formatos de intercambio de datos geográficos

1. **GeoJSON**
   - Basado en JSON (*JavaScript Object Notation*, RFC 7159).
   - Actualmente, borrador de RFC [Butler et alii, 2016].
   - Define objeto `Feature`, que contiene `Geometry` + otras propiedades (miembros).
   - `Geometry` contiene un *array* de coordenadas.
   - Define objeto `FeatureCollection` que consiste en un array de `Features`.
   - `Content-type: application/json`
2. **KML**
   - *Keyhole Markup Language*, formato usado por Google Earth.
   - Formato estándar del OGC desde su versión 2.2 (actualmente 2.3), basado en XML.
   - Próximo de GML, siendo un objetivo la convergencia entre ambos.
   - Visualización de datos geográficos en tres dimensiones.
   - Permite asociar a los datos anotaciones e imágenes.
   - Permite describir el recorrido de visualización de los datos y no sólo su localización en el mapa.
   - `Content-type: application/vnd.google-earth.kml+xml`

# Instalación y demo

## Servidor web

- Servidor simple con el módulo `CGIHTTPServer` de Python (mediante el *script* `startserver.sh` incluido).
- Cualquier servidor web que soporte CGI y disponga de un intérprete de Python con los módulos `PySpatialite` y `geojson`.

## Demo

- **En la máquina local (`CGIHTTPServer`):**
  `http://localhost:8000/panelmunicipal.html`
- **Demo disponible en Internet**
  **(hasta el 23 de junio de 2016 inclusive):**
  `https://dmsoler.homenet.org/tfg/panelmunicipal.html`

# Pantalla inicial de la aplicación

# Desarrollo futuro de la aplicación

**Aspectos a desarrollar y mejorar:**

- Formulario intuitivo o calendario para introducir fecha y hora de inicio y fin de las incidencias.
- Formulario intuitivo para filtro de incidencias.
- Nueva funcionalidad: operaciones espaciales entre incidencias y datos IDEPO.
- Carga automática de datos de ficheros GeoJSON (en diferentes sistemas de referencia) existentes en determinado directorio.
- Exportación de datos a formatos diferentes de KML soportados por OpenLayers (KML, GML, GeoJSON o incluso GPX, considerando cada incidencia un *waypoint*).

# Referencias


Butler, H. *et alii* (2016)

*The GeoJSON Format*

IETF Internet Draft,
https://datatracker.ietf.org/doc/draft-ietf-geojson/

Mearns, Ben (2015)

*QGIS Blueprints: Develop analytical location-based web applications with QGIS*

Birmingham, Packt Publishing, 2015.

Open Geospatial Consortium (2010)

*OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option. Version: 1.2.1*

http://www.opengeospatial.org/standards/sfs