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

Dionisio Martínez Soler
Grado en Tecnologías de Telecomunicación,
mención en Ingeniería Telemática
Sistemas de Información Geográfica y Geotelemática

Consultor: Ramon Català Pou
Profesor responsable de la asignatura: Antoni Pérez Navarro

6 de junio de 2016

FICHA DEL TRABAJO FINAL

Título del trabajo:	Aplicación basada en web y en software libre para la gestión municipal de incidencias y actuaciones en el espacio público
Nombre del autor:	Dionisio Martínez Soler
Nombre del consultor:	Ramon Català Pou
Nombre del PRA:	Antoni Pérez Navarro
Fecha de entrega (mm/aaaa):	06/2016
Titulación:	Grado en Tecnologías de Telecomunicación, mención en Ingeniería Telemática
Área del trabajo final:	Sistemas de Información Geográfica y Geotelemática
Idioma del trabajo:	Castellano
Palabras clave:	aplicaciones web, SIG, OpenLayers, GeoJSON, Python, JavaScript

Resumen del trabajo:

El presente trabajo describe el proceso de desarrollo de una aplicación para la gestión de datos georreferenciados sobre incidencias en un municipio, utilizando tecnologías web, software libre y datos abiertos. La aplicación consta de una interfaz web HTML+CSS+JavaScript que utiliza las bibliotecas OpenLayers y jQuery, almacena los datos sobre incidencias en una base de datos SpatiaLite y conecta la interfaz a la base de datos mediante *scripts* CGI escritos en Python, utilizando GeoJSON como formato de intercambio de datos. Para disponer de mapas e imágenes del terreno en la interfaz, se recurre a las ortofotos del Plan Nacional de Ortofotografía Aérea y a los mapas del proyecto OpenStreetMap, y se utilizan datos espaciales públicos disponibles en los servidores estatales españoles, recurriendo también a GeoJSON como formato de carga de datos en la interfaz. Además de la visualización y edición del contenido de la base de datos de incidencias, la aplicación permite también la exportación de dichos datos en formato KML, para su uso en otras aplicaciones.

Abstract:

This paper describes the development of an application for the management of geospatial data about incidents or events in a town by the town council. The application uses web technologies, free software and open data. It consists of a HTML+CSS+JavaScript web interface that makes use of OpenLayers and jQuery libraries, stores its data about incidents and events in a SpatiaLite database and connects the interface with the database by using Python CGI scripts and GeoJSON as geospatial data interchange format. The needed maps are obtained from OpenStreetMap project, orthophotos from Spanish National Orthophotography Plan (PNOA) and further public spatial vectorial data from other Spanish public services, using also GeoJSON as data loading format for the interface. The application allows for the visualization and edition of incidents and events stored in the spatial database, and also for exporting such data in KML format for their use in other applications.

Índice general

Página

Índice de figuras

Página

1. Introducción

1.1. Contexto y justificación del trabajo

El presente documento describe el proceso de realización de un trabajo de fin de grado (en adelante, TFG) del Grado de Tecnologías de Telecomunicación en el área de Sistemas de Información Geográfica y Geotelemática. El proyecto consiste en el desarrollo de una aplicación que permita la gestión de las incidencias y actuaciones en el espacio público en el ámbito de un municipio, de manera que se permita:

- la visualización en pantalla de un mapa del municipio y de los diferentes objetos y recursos situados en él (por ejemplo, redes de saneamiento e iluminación);
- la introducción de registros de incidencias y actuaciones geolocalizadas en una base de datos, mediante un click sobre el mapa en el punto donde se producen y un formulario que permite asociar a ese punto otros campos con información relevante (tipología, radio de afectación, fecha y hora de la incidencia o actuación y duración prevista);
- la visualización sobre el mismo mapa del contenido de la base de datos indicada en el punto anterior, de forma total o filtrada (por ejemplo, por tipo de incidencia o actuación, o por fecha y hora);
- la edición del contenido de los registros de la base de datos mediante su selección en el mapa y la alteración de los campos a través de un formulario;
- la exportación de la base de datos a un formato que permita su visualización en otros sistemas o aplicaciones de información geográfica, de manera que esté disponible para su consulta por el público en general (por ejemplo, KML).

1.2. Objetivos del trabajo

El objetivo del presente trabajo es conocer y aplicar en el desarrollo de un proyecto práctico y utilizable en producción:

- las tecnologías libres basadas en web disponibles para el procesamiento de información geográfica;
- diferentes estándares libres y abiertos de intercambio de datos geolocalizados;
- diferentes repositorios libres y abiertos de información geográfica disponibles públicamente.

La aplicación cumplirá con los siguientes requisitos:

- deberá ser interoperable y usar formatos estándar;
- estará basada en software libre;
- permitirá el acceso para consulta y/o para introducción y edición de datos de incidencias y actuaciones a través de diferentes puestos de trabajo conectados en red, ya sea en la intranet del municipio o en Internet;
- utilizará en la medida de lo posible mapas y datos geográficos abiertos, distribuidos bajo una licencia abierta.

Todo el código generado en el proyecto estará bajo una licencia GPLv3 (Licencia Pública General GNU, versión 3) [2] y la documentación (manual de uso de la aplicación, memoria final del proyecto y presentación multimedia) bajo una licencia FDLv1.3 (Licencia de Documentación Libre GNU, versión 1.3) [1].

1.3. Enfoque y método seguido

Dado el tiempo reducido en el que se desarrolla el trabajo (12 créditos, que corresponden a 120 horas, a lo largo de un período de poco más de 3 meses), la metodología seguida para el desarrollo de la aplicación habrá de ser de *prototipización rápida*, desarrollando modelos básicos de funcionamiento que después puedan ser ajustados y mejorados a medida que van siendo probados.

Tanto el uso de un sistema de control de versiones Git como la utilización de los datos geográficos de un municipio concreto para probar el funcionamiento (el municipio de Tomiño, en la provincia de Pontevedra) facilitarán la tarea de desarrollar la aplicación de forma progresiva. En la sección siguiente se especifica cómo se planifica temporalmente el trabajo, considerando diversos componentes de la aplicación que se desarrollarán de forma independiente como prototipos funcionales que después pueden ser interconectados:

- la base de datos que almacenará la información sobre las incidencias en forma de objetos geográficos;
- la interfaz web que permitirá interactuar de forma visual, sobre un mapa, con los objetos geográficos de la base de datos;
- los datos geográficos que se mostrarán en la interfaz para poder analizar las incidencias en relación a ellos.

Una vez desarrollados u obtenidos cada uno de esos componentes, será necesario interconectarlos para conseguir una aplicación con las funcionalidades pretendidas.

Todo el código necesario será desarrollado con la ayuda de un sistema de control de versiones Git, utilizando la plataforma Gitlab, a través de la siguiente cuenta del autor del proyecto:

https://gitlab.com/u/dmsoler

1.4. Planificación del trabajo

Se describen a continuación las diferentes tareas que deberán ser realizadas a lo largo del proyecto y que figuran en el cuadro-resumen de tareas e hitos principales y en los diagramas de Gantt situados al final de esta sección, en el apartado 1.4.1.

1. **Definición de la base de datos utilizada.**

   Evaluación de las ventajas e inconvenientes de utilizar para el proyecto alguna de de las bases de datos geográficos de código abierto disponibles, especialmente PostGIS [3] y SpatiaLite [4], elección de una de ellas y diseño de un archivo de comandos SQL que permitan la creación de la estructura de la base de datos.

2. **Creación de la interfaz web de la aplicación.**

   Evaluación de las características, ventajas e inconvenientes de las dos bibliotecas JavaScript disponibles: OpenLayers [5] y Leaflet [6], y diseño y desarrollo de una interfaz web con HTML, CSS y JavaScript, utilizando una de las bibliotecas indicadas, de forma que pueda ser utilizada en un navegador.

3. **Conexión de la interfaz web a las fuentes de datos geográficos utilizadas.**

   Obtención de mapas rasterizados basados en OpenStreetMap, ortofotos del Plan Nacional de Ortofotografía Aérea (PNOA) y datos sobre el municipio escogido como prueba (Tomiño, provincia de Pontevedra) procedentes de la Infraestructura de Datos espaciales de la Diputación de Pontevedra.

4. **Acceso de escritura y lectura a la base de datos de almacenamiento de incidencias.**

   Evaluación de distintas opciones (node.js + JavaScript, PHP, Geoserver, etc.) y formatos de intercambio de datos (WKT, GeoJson,etc.) para el acceso de lectura y escritura a la base de datos SpatiaLite de la aplicación, elección de una de ellas y desarrollo del código necesario para el acceso de lectura y escritura a la base de datos a partir de la interfaz web.

5. **Desarrollo de funcionalidades de la interfaz web.**

   Desarrollo del código necesario para que la interfaz web permita al usuario tanto la modificación de la base de datos como la realización de algunas operaciones espaciales sobre los datos disponibles. Asimismo, será necesario desarrollar el código necesario para la funcionalidad de exportación a formato KML.

6. **Pruebas de funcionamiento de la aplicación.**

   Las pruebas de funcionamiento de la aplicación se llevarán a cabo en una estación de trabajo con sistema GNU/Linux, e incluirán:

   - pruebas de acceso local;

- pruebas de acceso a través de una red local desde estaciones de trabajo con diferentes sistemas operativos y navegadores;
- verificación de cada una de las funcionalidades requeridas tanto en acceso local como a través de la red.

7. **Redacción del manual de uso de la aplicación.**

   Una vez concluida y probada la aplicación, se elaborará un pequeño manual de uso que permita su instalación y puesta en marcha.

8. **Redacción de la presente memoria final.**

   Se utilizará como sistema de preparación del documento LaTeX.

9. **Elaboración de la presentación virtual.**

   Se requiere la elaboración de la presentación virtual mediante la plataforma PRESENT@, disponible en el campus UOC.

10. **Preparación del debate virtual**

    Tras la entrega final, se dispone de un período de quince días para la preparación del debate virtual.

1.4.1. Cuadro-resumen de tareas e hitos principales y diagramas de Gantt

En los siguientes cuadros se muestran las tareas e hitos principales del proyecto con la previsión de dedicación temporal en días y en horas, sobre el total de 120 horas (12 créditos) previsto para la elaboración del TFG. Dado que la extensión temporal del proyecto es de 107 días (del 9 de marzo al 23 de junio de 2016), se dedicará al proyecto en media algo más de una hora diaria, aunque esta dedicación podrá ser variable en función de otras tareas y proyectos del autor, del tipo de tareas en cada período del proyecto y de la necesidad de simultanear tareas en algunas fases del mismo. Así, por ejemplo, las dos primeras fases del proyecto tienen una dedicación en horas por día superior a la media, mientras que la última fase, el debate virtual, tiene una dedicación inferior a la media. Asimismo, la tercera fase, antes de la entrega final, requiere la realización simultánea de tareas de redacción de la documentación escrita y de pruebas de la aplicación y resolución de *bugs*.

Tareas e hitos del 9/3/2016 al 12/4/2016 (diagrama de Gantt en la figura 1.1)	**Días**	**Horas**
Definición de la base de datos utilizada	**15**	**20**
Elección de la base de datos	8	10
Memoria (borrador): bases de datos espaciales	16/3/2016	
Diseño de la base de datos	7	10
Fichero de comandos SQL	23/3/2016	
Interfaz web de la aplicación	**20**	**20**
OpenLayers vs. Leaflet	13	10
Memoria (borrador): bibliotecas JavaScript	4/4/2016	
Diseño inicial de la interfaz web	7	10
Interfaz web (primera versión)	12/4/2016	
Prueba de Evaluación Continua (PEC) 2	12/4/2016	

Tareas e hitos del 13/4/2016 al 10/5/2016 (diagrama de Gantt en la figura 1.2)	**Días**	**Horas**
Interfaz web de la aplicación	**28**	**30**
Conexión de la interfaz web a las fuentes de datos	14	15
Memoria (borrador): fuentes de datos espaciales	26/4/2016	
Interfaz web (segunda versión)	26/4/2016	
Desarrollo de funcionalidades de la interfaz web	14	15
Memoria (borrador): formatos de intercambio de datos espaciales	10/5/2016	
Interfaz web (versión beta)	10/5/2016	
PEC 3	10/5/2016	

Tareas e hitos del 11/5/2016 al 6/6/2016 (diagrama de Gantt en la figura 1.3)	**Días**	**Horas**
Pruebas de funcionamiento de la aplicación	**20**	**20**
Pruebas y detección de *bugs*	10	10
Corrección de *bugs*	10	10
Versión final de la aplicación	30/5/2016	
Redacción final	**20**	**20**
Redacción del manual de uso de la aplicación	13	2
Redacción de la versión final de la memoria	13	10
Elaboración de la presentación virtual	13	2
Redacción del informe de competencias transversales	13	2
Versión final de la memoria, presentación e informe	30/5/2016	
Revisión final	7	4
Entrega final	6/6/2016	

Tareas e hitos del 7/6/2016 al 23/6/2016 (diagrama de Gantt en la figura 1.4)	**Días**	**Horas**
Debate virtual	**17**	**10**
Preparación del debate virtual	15	8
Inicio del debate virtual	21/6/2016	
Desarrollo del debate virtual	2	2
Fin del debate virtual	23/6/2016	

Figura 1.1: diagrama de Gantt de 9/3/2016 a 12/4/2016 (Prueba de Evaluación Continua – PEC – 2)
Marzo
Abril
09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 01 02 03 04 05 06 07 08 09 10 11 12
Definición de la base de datos
Elección de la base de datos
Memoria (borrador): bases de datos espaciales
Diseño de la base de datos
Fichero de comandos SQL
Interfaz web de la aplicación
OpenLayers vs. Leaflet
Memoria (borrador): bibliotecas JavaScript
Diseño inicial de interfaz web
Interfaz web (primera versión)
PEC2

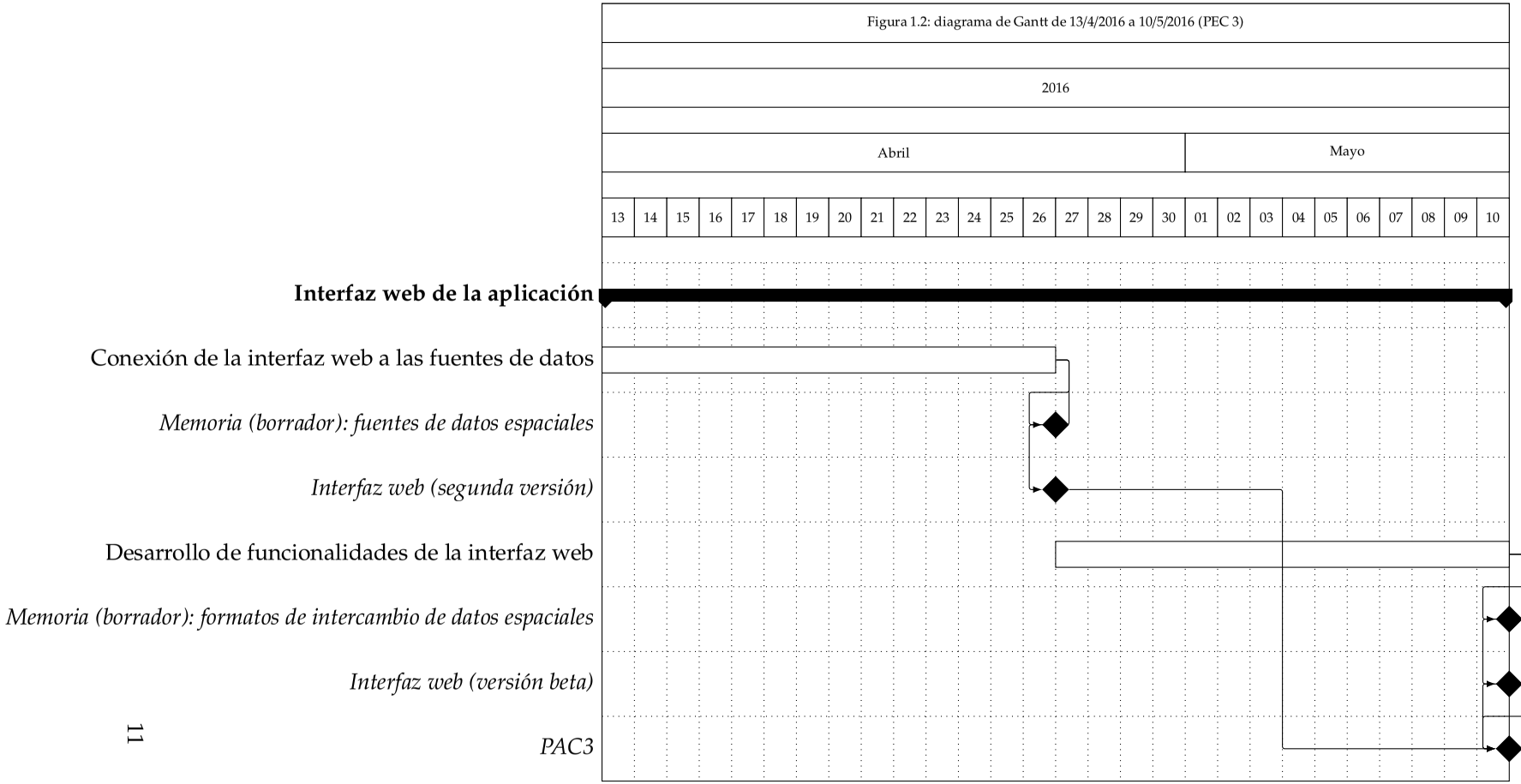
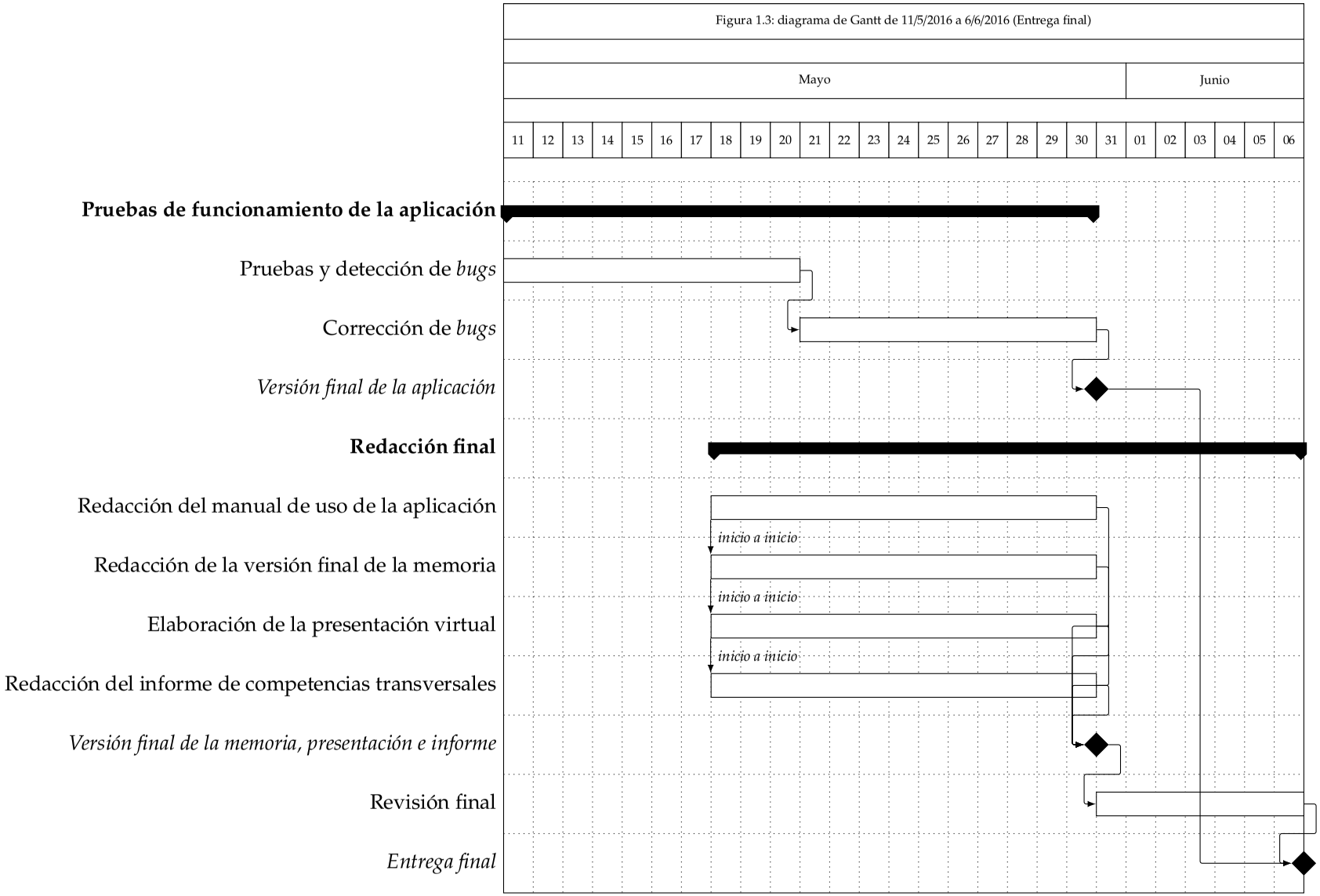


Figura 1.2: diagrama de Gantt de 13/4/2016 a 10/5/2016 (PEC 3)

Figura 1.3: diagrama de Gantt de 11/5/2016 a 6/6/2016 (Entrega final)



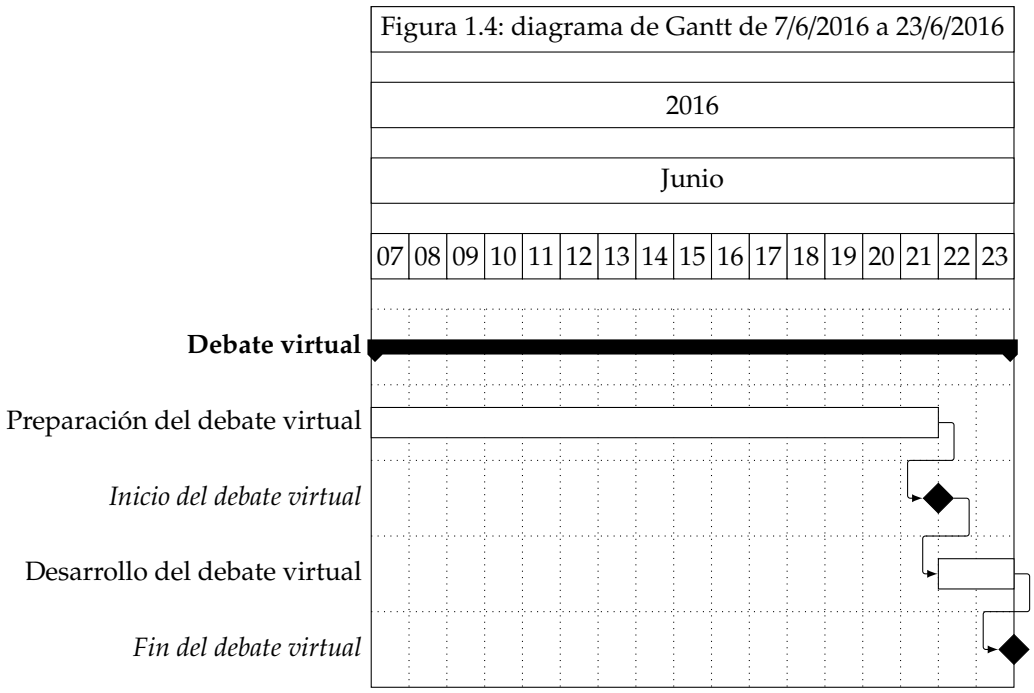


Figura 1.4: diagrama de Gantt de 7/6/2016 a 23/6/2016

1.5. Breve sumario de productos obtenidos

La aplicación desarrollada en el proyecto y que se denominará "Panel Municipal" es una aplicación web con dos componentes: un componente cliente escrito en HTML5+JavaScript+CSS, y un componente servidor escrito en Python y que debe funcionar en cualquier servidor web con soporte CGI y un intérprete Python con los módulos `pyspatialite` y `geojson`. La versión de Python debe ser la 2.7, la última que soporta el módulo `pyspatialite`.

El componente cliente consta de tres ficheros situados en el mismo directorio:

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

Para ejecutar la aplicación basta abrir con cualquier navegador, localmente o a través de un servidor web, el fichero `panelmunicipal.html`. Se suministra también un fichero `idepo.js` que contiene datos geográficos del municipio que se usa como ejemplo (Tomiño, en la provincia de Pontevedra) en formato GeoJSON, obtenidos del Geoportal de la Diputación de Pontevedra.

Todas las funcionalidades relacionadas con la base de datos de incidencias requieren el componente servidor, que consta de los siguientes cuatro *scripts* escritos en el lenguaje de programación `Python`:

- `incidencias.py`: permite crear, modificar o borrar registros en la base de datos de incidencias.

- `incidenciaslayer.py`: permite obtener un volcado completo o parcial de la base de datos de incidencias en formato GeoJSON.
- `incidenciasbuffer.js`: permite obtener en un fichero en formato GeoJSON el conjunto de polígonos resultante de aplicar a los registros de la base de datos de incidencias (ya sea a todos ellos o a una parte seleccionada) la operación espacial `Buffer` de acuerdo con el valor almacenado en el campo `radio`.
- `incidenciaskml.py`: permite la exportación de toda o parte de la base de datos de incidencias a un fichero en formato KML.

La base de datos utilizada está contenida en el fichero `incidencias.sqlite` que se puede crear y verificar mediante los siguientes ficheros:

- `createdatabase.sh`: crea la base de datos en el fichero `incidencias.sqlite` utilizando los comandos SQL contenidos en el fichero siguiente.
- `incidencias.sql`: fichero de comandos SQL para la creación de la estructura inicial de la base de datos.
- `testdatabase.sh`: permite verificar el correcto funcionamiento de la base de datos recién creada, introduciendo en ella un registro de prueba y comprobando que una consulta devuelve su contenido, mediante los comandos SQL contenidos en el siguiente fichero.
- `test.sql`: fichero de comandos SQL para la verificación del correcto funcionamiento de la base de datos recién creada.

Asimismo, se suministra un *script* `startserver.sh` que inicia un servidor web (puerto 8000) que sirve los archivos contenidos en la estructura de directorios y permite utilizar la aplicación de forma completamente funcional, mediante el módulo `HTTPCGIServer` de `Python`.

Todos estos ficheros se suministran en un archivo comprimido, conteniendo la siguiente estructura de directorios:

- `code`
  - `panelmunicipal.html`
  - `panelmunicipal.css`
  - `panelmunicipal.js`
  - `idepo.js`
  - `createdatabase.sh`
  - `incidencias.sql`
  - `testdatabase.sh`
  - `test.sql`
  - `startserver.sh`
  - `cgi-bin`
    - `incidencias.py`

- `incidenciaslayer.py`
- `incidenciasbuffer.py`
- `incidenciaskml.py`

En el apéndice B puede consultarse un manual detallado de instalación y uso del "Panel Municipal". Para comprobar las potencialidades de la aplicación para su funcionamiento en red, se encuentra también disponible una versión de demostración en la siguiente dirección web:

`https://dmsoler.homenet.org/tfg/panelmunicipal.html`

En esta demostración, se han utilizado datos de otro municipio de la provincia de Pontevedra (el municipio de A Guarda), y estará accesible a través de Internet hasta el 23 de junio de 2016, fecha de finalización del debate virtual.

1.6. Breve descripción de los otros capítulos de la memoria

El capítulo 2 describe el proceso de selección y diseño de la base de datos utilizada para el almacenamiento de los datos de incidencias producidas en el municipio. En el capítulo 3 se describe el proceso seguido para el diseño y programación de una interfaz web basada en HTML5+CSS+JavaScript utilizando diversas bibliotecas JavaScript de código abierto disponibles para facilitar el desarrollo, tanto en lo que se refiere a la obtención y visualización de datos espaciales (OpenLayers [5] y Leaflet [6]) como en lo que se refiere al establecimiento de conexiones con diferentes servidores y a la obtención, manipulación e intercambio de datos (JQuery [7]). Asimismo, se describe cómo se han obtenido e incluido en la aplicación los datos geográficos vectoriales y ráster que se visualizan como base para colocar sobre ellos las incidencias que se introducen en la base de datos. Por último, el capítulo 4 describe los mecanismos utilizados para la conexión de la aplicación web a la base de datos de incidencias producidas en el municipio, y los formatos de intercambio de datos utilizados para esa conexión y para la exportación de su contenido.

La aplicación objeto del presente trabajo puede refinarse y mejorarse en diferentes aspectos que se analizan brevemente en el capítulo 5. Finalmente, en el capítulo 6 y último se presentan algunas conclusiones extraídas del proceso de desarrollo del "Panel Municipal".

2. Definición de la base de datos utilizada

Para el almacenamiento de los datos de incidencias producidas en el municipio, necesitaremos una base de datos relacional que pueda almacenar datos geográficos, es decir, que entre los tipos de campos posibles incluya los objetos espaciales como "punto" o "polígono".

Los gestores de bases de datos relacionales capaces de almacenar datos geográficos que están disponibles hoy en día siguen el estándar del OGC *Simple feature access - Part 2: SQL option* [8], que define un tipo de datos denominado `Geometry`, con los subtipos `Point`, `Curve`, `LineString`, `Surface`, `Polygon`, `PolyhedSurface`, `GeomCollection`, `MultiCurve`, `MultiLineString`, `MultiSurface`, `MultiPolygon` y `MultiPoint`. Además de estos nuevos tipos de datos (o, por lo menos, una parte de ellos), las implementaciones conformes con el estándar deben definir funciones para la importación y exportación de datos en los formatos WKT y WKB y funciones espaciales que permiten operar sobre los datos contenidos en los campos de esos nuevos tipos.

Aparte de las implementaciones comerciales, están disponibles dos implementaciones libres y ampliamente utilizadas: PostGIS [3] y SpatiaLite [4].

2.1. PostGIS

PostGIS es una extensión espacial para PostgreSQL [9], uno de los más usados y potentes gestores de bases de datos relacionales de código abierto que están disponibles en este momento. Como la mayoría de estos programas, PostgreSQL utiliza una estructura cliente-servidor: para gestionar bases de datos es necesario instalar y configurar el servidor PostgreSQL e interactuar con él a través de un cliente específico – ya sea un programa completo o una de las bibliotecas de funciones disponibles para diversos lenguajes de programación. PostGIS implementa el estándar OGC-SFS, y dispone de una gran cantidad de funciones espaciales tanto para el procesado de datos vectoriales como de datos rasterizados.

2.2. SpatiaLite

SpatiaLite es una extensión espacial para SQLite [10], un gestor de bases de datos relacionales que no utiliza una estructura cliente-servidor, como es habitual, sino que consiste en una biblioteca de software que permite la manipulación de bases de datos contenidas simplemente en un fichero. Ello permite que no sea necesario ningún tipo de instalación o configuración: simplemente la biblioteca de software implementa las rutinas necesarias para que programas escritos en diferentes lenguajes de programación puedan acceder a la base de datos contenida en el fichero y ejecutar sobre ella comandos SQL. El código de SQLite está en el dominio público.

SpatiaLite extiende las funcionalidades de SQLite introduciendo el tipo de datos `Geometry` de acuerdo con el estándar OGC-SFS. almacena los campos del tipo `Geometry` internamente como un objeto *binario*, y soporta los subtipos `Point`, `LineString`, `Ring`, `GeometryCollection`, `MultiPoint`, `MultiLineString` y `MultiPolygon`. Dado que la representación interna de los objetos espaciales es binaria para permitir mayor rapidez y eficiencia al procesar datos, la interacción a través de comandos SQL con el gestor de base de datos debe hacerse utilizando formatos textuales de intercambio de datos espaciales. SpatiaLite es capaz de escribir y leer datos a partir del contenido de una base de datos utilizando los formatos WKT, EWKT, KML, GML, y GeoJSON, mediante funciones específicas del tipo `As<format>()` y `GeomFrom<format>()`, por ejemplo `AsGeoJSON()` y `GeomFromGeoJSON()`. Asimismo, SpatiaLite crea por defecto en cualquier base de datos el conjunto de tablas especificado en el estándar OGC-SFS, necesario para el correcto funcionamiento de las funciones espaciales que manipulan los objetos geográficos. La lista de funciones espaciales disponible en SpatiaLite puede consultarse en la sección correspondiente de la documentación en línea [11].

2.3. Elección de la base de datos

La elección más lógica para el proyecto es SpatiaLite. PostGIS ofrece sin duda muchas más funcionalidades y PostgreSQL más tipos de datos frente al enfoque minimalista de SpatiaLite y SQLite, pero son funcionalidades que no son necesarias para los requisitos del proyecto, y su uso requeriría un proceso de instalación, configuración y mantenimiento del servidor PostgreSQL excesivamente complejo. Además, disponer de todo el contenido de la base de datos en un fichero accesible mediante multitud de herramientas cuyo código está en el dominio público facilita enormemente cualquier proceso de migración de la aplicación a una nueva máquina o servidor.

Se suele apuntar como inconveniente de SQLite algunas limitaciones en entornos multiusuario (cf. Mearns [12, pág. 138]), que hacen que la mayoría de las páginas web que gestionan datos espaciales prefieran PostGIS. Esas limitaciones están relacionadas con el bloqueo del acceso a la base de datos para escritura por los accesos de lectura y viceversa, que pueden llevar a la denominada "writer starvation": un acceso de escritura a una base de datos que está siendo accedida simultáneamente para lectura por una gran cantidad de usuarios puede quedar a la espera durante mucho tiempo, pues para garantizar transacciones ACID la biblioteca SQLite bloquea el acceso al fichero durante el tiempo que dura cualquier transacción, sea de lectura o de escritura (cf. AA.VV. [13, "File Locking And Concurrency In SQLite Version 3"]). En versiones recientes de SQLite (3.7.0 o posteriores), existe un método de acceso a la base de datos denominado "Write-Ahead Logging" (WAL) que permite el acceso simultáneo de lectura y escritura (cf. AA.VV. [13, "Write-Ahead Logging"]), pero no resultará necesario para la aplicación que estamos desarrollando. Incluso en un escenario con diferentes usuarios de los servicios municipales utilizando simultáneamente la aplicación a través de la red, los accesos necesarios a la base de datos, tanto de lectura como de escritura, son muy puntuales: una vez obtenido el contenido de la base de datos a través de un proceso de lectura, los objetos geográficos (*features*) se almacenan en

la memoria de la aplicación cliente, que se ejecuta en un navegador web, y sólo resulta necesario volver a acceder a la base de datos para introducir una nueva incidencia o modificar alguna de las existentes, o bien para obtener un nuevo volcado (integral o filtrado) del contenido de la base de datos en la visualización de la aplicación.

2.4. Diseño de la base de datos

Para la creación de la base de datos SpatiaLite, se utilizará un fichero de comandos SQL que permita crear la estructura de la base de datos. Atendiendo a los tipos de datos soportados por SQLite, se creará una base de datos con una tabla denominada "Incidencias" que contendrá los siguientes campos:

- `id`: un campo constituido por un número entero, un identificador único que actuará como llave primaria. Para obtener un identificador único, optamos por recurrir al sello de tiempo que se puede obtener mediante la utilización del método `getTime()` sobre un objeto de la clase `Date` en JavaScript: creamos un objeto `Date` cuyo contenido es el momento de la creación de la incidencia (con precisión de milisegundos) y dicho método devuelve el número de milisegundos transcurrido desde las 0 horas del 1 de enero de 1970 hasta el momento de creación de la incidencia. El sello de tiempo como identificador permite también ordenar la incidencias por su fecha de creación. Tal como referíamos en el caso de las limitaciones de SQLite para el acceso concurrente, en un entorno en el que no se prevé una gran cantidad de accesos simultáneos, es altamente improbable (aunque posible) que se intente crear una incidencia en dos accesos simultáneos exactamente en el mismo milisegundo. En el caso muy improbable de que esto ocurra, una de las dos incidencias creadas será sobreescrita por la otra. Para resolver esta dificultad se puede realizar alguna operación matemática con el sello de tiempo y un número generado de forma aleatoria mediante el método `Math.random()`, reduciendo así la posibilidad de coincidencias, pero por el momento parece un defecto de la aplicación asumible en una utilización normal.
- `tipo`: un campo de texto de un máximo de 50 caracteres, que indica el tipo de incidencia.
- `radio`: un número entero que representa el radio del área afectada por la incidencia, expresado en metros.
- `inicio`: um campo de texto de exactamente 22 caracteres, que permite introducir la fecha y hora de inicio de la incidencia en uno de los formatos soportados por SQLite, YYYY-MM-DDTHH:MM[+-]HH:MM, formato en el que el valor situado después de [+-] indica el huso horario.
- `fin`: un campo de texto del mismo tamaño y formato que el campo `inicio`, que permite introducir la fecha y hora de fin de la incidencia.

Nótese que SQLite no soporta tipos de datos específicos para la representación de la fecha y la hora, sino que almacena fecha y hora en un campo de tipo numérico o de texto, utilizando determinados formatos que permiten aplicar

a los valores funciones específicas de conversión de fecha y hora a otros formatos (cf. AA.VV. [13, "SQLite Query Language: Date and Time Functions"])). El uso de uno de los formatos de fecha y hora reconocidos por SQLite permite realizar consultas a la base de datos basándose en comparación de fechas mediante las funciones específicas de tiempo de SQLite, por ejemplo el comando SQL

```
SELECT * FROM Incidencias WHERE inicio < date('now') AND fin > date('now')
```

devolverá todas las incidencias que se encuentren activas en el momento presente.

El código del fichero de comandos SQL necesario para la creación de la base de datos puede verse en el apéndice A.1. Nótese que el campo que contiene los objetos espaciales no puede ser creado como el resto dentro del comando `CREATE TABLE` sino que tiene que ser necesariamente añadido posteriormente mediante la función `AddGeometryColumn()`: sólo así se crea todo lo necesario para el correcto funcionamiento de la base de datos espacial, añadiendo las tablas específicas previstas en el estándar OGC-SFS. Para crear la base de datos con el nombre `incidencias.sqlite`, podemos utilizar la siguiente orden en línea de comandos, si tenemos el código SQL en el fichero `incidencias.sql` y disponemos del ejecutable binario `spatialite` instalado en nuestro sistema:

```
spatialite incidencias.sqlite < incidencias.sql
```

Con ello habremos creado la base de datos. Para testar que ha sido creada correctamente, en el apéndice A.2 se incluye el código que inserta una fila de valores y después la muestra, podemos ejecutarlo mediante:

```
spatialite incidencias.sqlite < test.sql
```

Si todo ha ido bien, la última línea que se nos mostrará en la salida será:

```
1460328181697|Incendio|200|2016-04-10T22:30+02:00|2016-05-10T22:30+02:00|
    {"type":"Point","coordinates":[-8.737299999999999,42]}
```

El ejecutable binario necesario para estas operaciones forma parte del conjunto de utilidades `spatialite-tools` escritas en lenguaje de programación `C` cuyo código fuente se puede obtener en la página web de SpatiaLite [4], y que es necesario compilar para el sistema en el que vayan a ser utilizadas (en el caso de sistemas Microsoft Windows utilizando Visual Studio .NET). Sin embargo, se pueden obtener ya compiladas en diversas distribuciones de sistemas GNU/Linux. En el caso de la distribución Debian, que dispone de un subproyecto dedicado a la inclusión en la distribución de aplicaciones para

sistemas de información geográfica [14] podemos obtener todo lo necesario para estas operaciones simplemente instalando el paquete `spatialite-bin`, y disponemos de todo el software necesario para manipular bases de datos SpatiaLite como la utilizada en este proyecto, incluyendo utilidades gráficas (paquete `spatialite-gui`) y un módulo para el lenguaje de programación `Python` (paquete `python-pyspatialite`).

3. Creación de la interfaz web de la aplicación

Aunque no alcanzan todas las posibilidades de un GIS, las interfaces web basadas en los estándares del W3C (HTML5, CSS y JavaScript) se han difundido mucho, especialmente para la visualización de datos.

Una forma sencilla de comprobar sus posibilidades es utilizar el módulo del sistema de información geográfica libre QGIS [15] denominado `qgis2web` [16], que permite exportar una capa de datos vectoriales definida en QGIS a una página web. Dicho módulo, utilizando una de las dos bibliotecas JavaScript disponibles para el procesado de datos geográficos, OpenLayers [5] o Leaflet [6], muestra los datos vectoriales de una o varias capas de un proyecto QGIS sobre un mapa obtenido de OpenStreetMap (cf. sección 3.3.1), conservando todas las definiciones de estilo de visualización que pudieran estar definidas en las capas de QGIS y permitiendo incluso mostrar de forma interactiva los campos y valores asociados a cada uno de los objetos geográficos de esas capas.

Pero además de las posibilidades que ofrecen para la visualización de datos, esas interfaces permiten también la creación, edición, almacenamiento y exportación a diversos formatos de nuevos datos geográficos creados a partir de ellas, por lo que pueden resultar una opción muy válida e interesante para el desarrollo de pequeñas aplicaciones de procesamiento de datos geográficos como la que es objeto de este trabajo. Su ejecución no requiere más que un navegador estándar, sin necesidad de instalar y configurar complejos sistemas GIS que den soporte a sus funcionalidades.

En Mearns [12, pág. 180 y ss.], bajo el título "A dynamic web application – OpenLayers AJAX with Python and SpatiaLite", se describe una aplicación muy semejante a la que es objeto del presente trabajo, desarrollada a partir de la exportación de un proyecto elaborado en QGIS mediante el módulo `Export to OpenLayers`, módulo que ha dejado de ser desarrollado para incorporarse en el ya citado `qgis2web` [16]. La interfaz resultante es después modificada para permitir la conexión (solamente para lectura) a una base de datos SpatiaLite a la que se accede a través de un servidor minimalista escrito en Python, recurriendo a los módulos `CGIHTTPServer` y `PySpatiaLite`.

Si bien en este ejemplo el acceso a la base de datos es sólo de lectura, para poder mostrar los datos contenidos en ella sobre el mapa, el mismo esquema puede ser utilizado para un acceso de lectura y escritura a la base de datos. La interfaz desarrollada para el presente proyecto utiliza la idea de Mearns [12] ampliándola para que sea posible también crear, editar, almacenar y exportar un conjunto de puntos introducidos por el usuario en el mapa y que representan las incidencias producidas en el municipio, con una serie de datos asociados a cada uno de ellos.

3.1. Bibliotecas JavaScript para el trabajo con datos geográficos en el navegador: OpenLayers y Leaflet

Las dos opciones disponibles para el desarrollo de esta interfaz web, OpenLayers [5] y Leaflet [6], son muy semejantes pero presentan diferencias importantes de las que se derivan ventajas e inconvenientes.

Leaflet es mucho menos versátil y está orientada al desarrollo rápido de aplicaciones que incluyan todas las funcionalidades básicas escribiendo un código sencillo. Es también muy ligera para favorecer su uso en dispositivos móviles. En lo que a este proyecto se refiere, su principal inconveniente es no disponer de funcionalidades para la creación y edición de nuevos objetos geográficos: esas funcionalidades pueden incorporarse mediantes *plugins* externos, pero no están incluidas en la biblioteca Leaflet.

OpenLayers, por el contrario, es una biblioteca más compleja y pesada, que exige escribir más código que Leaflet para obtener una interfaz con las funcionalidades básicas. A cambio, es más flexible y versátil, e incorpora en la propia biblioteca formas de interacción con el mapa que permiten dibujar nuevos objetos sobre él, editarlos y guardarlos en una capa vectorial exportable a diversos formatos. Esta última característica, permitir la creación y edición de nuevos datos sin recurrir a *plugins* externos, la hace preferible a Leaflet para este proyecto.

Debe, sin embargo, tenerse en cuenta que su complejidad la hace también más vulnerable a la aparición de errores inesperados en partes del código que no han sido suficientemente testadas. Recientemente (febrero de 2016) alguno de esos errores afectó a uno de los componentes necesarios para este proyecto (cf. [17]), la interacción `ol.interaction.Select` que permite seleccionar objetos de una capa vectorial.

Globalmente, sin embargo, OpenLayers resulta la mejor opción. Los *bugs* en la biblioteca OpenLayers que puedan eventualmente aparecer durante el desarrollo de la aplicación suelen poder resolverse recurriendo a procedimientos compartidos por sus usuarios en los foros de desarrollo, o bien volviendo a una versión anterior de la biblioteca – pues las sucesivas versiones siguen estando disponibles, dado que hay páginas activas cuyo código no es actualizado y que dependen de esas versiones. Para el desarrollo de la aplicación utilizaremos la última versión de OpenLayers disponible en la fecha actual, la 3.15.1.

3.2. Diseño de la interfaz web de la aplicación con OpenLayers

La interfaz web necesita tres componentes:

- Un fichero HTML que es el que será abierto por el navegador. En él se hará referencia a todos los estilos CSS y scripts utilizados y se incorporarán algunos fragmentos breves de código JavaScript para permitir que la interfaz pueda ser modificada y personalizada de forma básica y simple editando solamente este fichero.

- Un fichero de estilo CSS que controle la apariencia gráfica de la interfaz.

- Un fichero JavaScript que contiene el código necesario para hacer funcionar la interfaz.

Aunque se pueden descargar para su utilización local, la biblioteca OpenLayers y la hoja de estilos CSS asociada se pueden obtener de la web colocando las siguientes líneas en el elemento <head>:

```
<link rel="stylesheet" href="http://openlayers.org/en/v3.15.1/css/ol.css"
type="text/css">
    <script src="http://openlayers.org/en/v3.15.1/build/ol.js"></script>
```

O, en alternativa, se puede utilizar la red de distribución de contenidos para bibliotecas JavaScript de código abierto cdnjs.com:

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/ol3/3.15.1/ol.css"
type="text/css">
    <script src="https://cdnjs.cloudflare.com/ajax/libs/ol3/3.15.1/ol.js"></script>
```

Asimismo, hay que tener que cuenta que OpenLayers sólo incorpora dos sistemas de referencia espacial (SRS) por defecto: EPGS:4326 (WGS84, el utilizado en los sistemas GPS) y EPGS:3857 (Spherical Mercator, el utilizado por los más difundidos servicios de mapas en la web, incluidos Google Maps y OpenStreetMaps), que suelen ser suficientes para la mayoría de las aplicaciones web. Sin embargo, necesitaremos también utilizar EPGS:23029 (ED50, zona UTM 29N), porque es el que utilizan los datos sobre los municipios de la provincia de Pontevedra publicados en el servidor de datos espaciales de la Diputación de Pontevedra [18].

Para incorporar nuevos SRS a las funcionalidades de OpenLayers, hay que recurrir a una biblioteca externa, proj4js [19]. Tal como se puede leer con detalle en el tutorial publicado por Andreas Hocevar [20], para ello basta obtener de la web la biblioteca proj4js y la definición del SRS del servidor http://epsg.io con las siguientes líneas incorporadas en el elemento <head>:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/proj4js/2.3.14/proj4.js">
</script>
<script src="http://epsg.io/23029.js"></script>
```

En alternativa, en vez de obtener la definición del SRS directamente del servidor http://epsg.io, puede incorporarse en el código JavaScript con la siguiente línea:

```
proj4.defs("EPSG:23029","+proj=utm +zone=29 +ellps=intl
       +towgs84=-87,-98,-121,0,0,0,0 +units=m +no_defs");
```

Con cualquiera de las dos opciones, tendremos el SRS EPSG:23029 disponible en la aplicación para poder realizar las conversiones necesarias para la correcta visualización en el SRS del mapa (EPGS:3857) de los datos sobre el municipio de Tomiño que disponibiliza la Diputación de Pontevedra bajo una licencia CC BY 3.0 (Creative Commons Reconocimiento 3.0), como se explica en el apartado 3.3.3.

Por otro lado, colocaremos también en <head> el siguiente código JavaScript para permitir la modificación del comportamiento inicial de la aplicación:

```
<script type="text/javascript">
  var zoomlevel = 13;
  var defaultlat = 42.0000;
  var defaultlon = -8.7373;
  var pnoa = true;
  var osm = true;
  var ocm = true;
  var stoner = true;
  var idepo = true;
  var geojson = false;
</script>
```

Esas variables controlan el nivel de zoom inicial y la localización del centro del mapa (en coordenadas de latitud y longitud EPGS:4326) y determinan las capas base que serán incorporadas al mapa:

pnoa	Capa base consistente en las ortofotografías del Plan Nacional de Ortofotografía Aérea (PNOA) (cf. 3.3.2)
osm	Capa base consistente en los mapas del proyecto OpenStreetMap (cf. 3.3.1)
ocm	Capa base consistente en la renderización de los datos del proyecto OpenStreetMap para la creación de un mapa para ciclistas, disponible en `http://www.opencyclemap.org`. Introduce curvas de nivel (basadas en datos de la NASA que están en el dominio público), por lo que puede resultar útil para visualizar los accidentes del terreno.
stoner	Capa base consistente en la renderización de los datos del proyecto OpenStreetMap para la creación de un mapa en blanco y negro que se pueda imprimir sin gran gasto de tinta de color, disponible en `http://maps.stamen.com`. Puede resultar útil para imprimir un mapa de incidencias. Tiene una licencia Creative Commons BY 3.0
idepo	Capa vectorial que muestra los datos sobre el municipio de Tomiño disponibles en la página de descarga de datos `http://ide.depo.es/datos.html` del Geoportal de la Infraestructura de Datos Espaciales de la Diputación de Pontevedra, bajo una licencia Creative Commons BY 3.0 (cf. 3.3.3).

La última variable, `geojson`, permite incluir en la interfaz un área de texto donde se muestran, en formato GeoJSON, los datos de incidencias nuevas y modificadas que se envían al servidor para su introducción en la base de datos. Este recurso puede resultar útil no sólo durante la fase de desarrollo

de la aplicación, sino también para un uso más especializado de la misma, pues permite copiar su contenido directamente del navegador a cualquier otra aplicación capaz de procesar datos GeoJSON, por ejemplo cualquier otro software de gestión de información geográfica.

En el fichero HTML deberán existir también elementos con los siguientes atributos `id`:

map	El elemento donde será colocado el mapa.
fullscreen	Todo el contenido de este elemento (y nada fuera de él) será visualizado cuando se use el botón de pantalla completa.
srs	Los cambios en este elemento serán observados para seleccionar el SRS utilizado para mostrar las coordenadas de la posición del cursor sobre el mapa.
info	En este elemento se colocará el formulario para la creación o edición de nuevos puntos de la capa de incidencias.
mousepositionform	Es el elemento donde se muestran las coordenadas de la posición del cursor sobre el mapa. Este `id` será usado para aplicarle un estilo CSS.
mouseposition	El elemento donde el control `ol.control.MousePosition` colocará las coordenadas de la posición del cursor sobre el mapa.
layerswitcher	Elemento donde se colocarán los controles para activar o desactivar la visibilidad de las diferentes capas y establecer su nivel de transparencia, de manera que sea posible escoger la capa base o visualizar varias superpuestas por transparencia.
popup	Elemento donde se coloca la información que se muestra al hacer click en el mapa sobre uno de los objetos geográficos de la capa de incidencias o de la capa vectorial con los datos de la Infraestructura de Datos Espaciales de la Diputación de Pontevedra. Debe incluir también elementos con los `id` `popup-content` (donde se colocará la información textual) y `popup-closer` (donde se colocará un elemento clicable para cerrar la información mostrada).

Por último, el fichero HTML debe cargar el *script* con las funcionalidades de la aplicación dentro del elemento `<body>`:

```
<script src="panelmunicipal.js"></script>
```

El *script* `panelmunicipal.js` utiliza la biblioteca OpenLayers (versión 3.x, en este momento 3.15.1) para implementar las funcionalidades de la interfaz. Utilizando los objetos y funciones de la biblioteca, el diseño de la interfaz consiste en un objeto de la clase `ol.Map` al que se añaden:

- Diversos objetos de las subclases `ol.layer.Layer`, que representan las diferentes capas vectoriales o ráster que forman parte del mapa.
- Diversos objetos de la clase `ol.control.Control`. Además de los controles por defecto en el mapa (`ol.control.Attribution`, `ol.control.Rotate` y `ol.control.Zoom`), se le añaden también `ol.control.OverviewMap`, `ol.control.ZoomSlider`, `ol.control.Fullscreen`, `ol.control.ScaleLine` y `ol.control.MousePosition`. Este último muestra las coordenadas de la posición del cursor sobre el mapa, utilizando el SRS que se le indique de entre los tres que utiliza la aplicación.
- Diversos objetos de la clase `ol.interaction.Interaction` que permiten al usuario interactuar con el mapa. Además de las interacciones por defecto en el mapa (`ol.interaction.DragRotate`, `ol.interaction.DoubleClickZoom`, `ol.interaction.DragPan`, `ol.interaction.PinchRotate`, `ol.interaction.PinchZoom`, `ol.interaction.KeyboardPan`, `ol.interaction.KeyboardZoom`, `ol.interaction.MouseWheelZoom` y `ol.interaction.DragZoom`), se le añaden también las interacciones `ol.interaction.Draw` (permite dibujar nuevos objetos geográficos sobre el mapa en una capa vectorial) y `ol.interaction.Select` (permite seleccionar objetos de una capa vectorial para editar sus propiedades o realizar otras operaciones sobre ellos). Se crea también la posibilidad de que el usuario active y desactive estas dos últimas interacciones. La interacción `ol.interaction.Draw` está configurada para dibujar un punto sobre el mapa haciendo click sobre él, y la interacción `ol.interaction.Select` selecciona uno de los puntos previamente dibujados haciendo CTRL+doble click sobre el punto en cuestión.
- Una función que crea controles de visibilidad y opacidad por separado para cada una de las capas del mapa.
- Una función que exporta el contenido de una capa vectorial al formato GeoJSON utilizando el SRS que se le indique. Este formato de intercambio de datos será el que se utilice para el envío de los datos de incidencias nuevas y modificadas al servidor para que se guarden en la base de datos.
- Un formulario para la introducción de los datos de los diferentes campos de cada uno de los puntos dibujados en la capa de incidencias. El *script* observa las modificaciones de los campos de introducción de datos para guardarlos siempre que se produce alguna modificación en su valor, y los envía al servidor para que se guarden en la base de datos. También se permite borrar de la base de datos una incidencia.
- Una función que solicita al servidor todas las incidencias contenidas en la base de datos, o bien una parte de ellas, de acuerdo con un filtro configurable por el usuario.

- Una función que exporta a un fichero KML el conjunto de incidencias obtenidas mediante la función anterior.
- Una función que muestra el contenido de las diferentes propiedades de cualquiera de los objetos vectoriales disponibles en el mapa (tanto en la capa de incidencias como en la capa vectorial de datos de infraestructuras obtenidos de la Diputación de Pontevedra) en un bocadillo de texto abierto sobre el mapa al hacer click sobre un objeto vectorial.

En la figura 3.1 se esquematiza el funcionamiento de la aplicación.

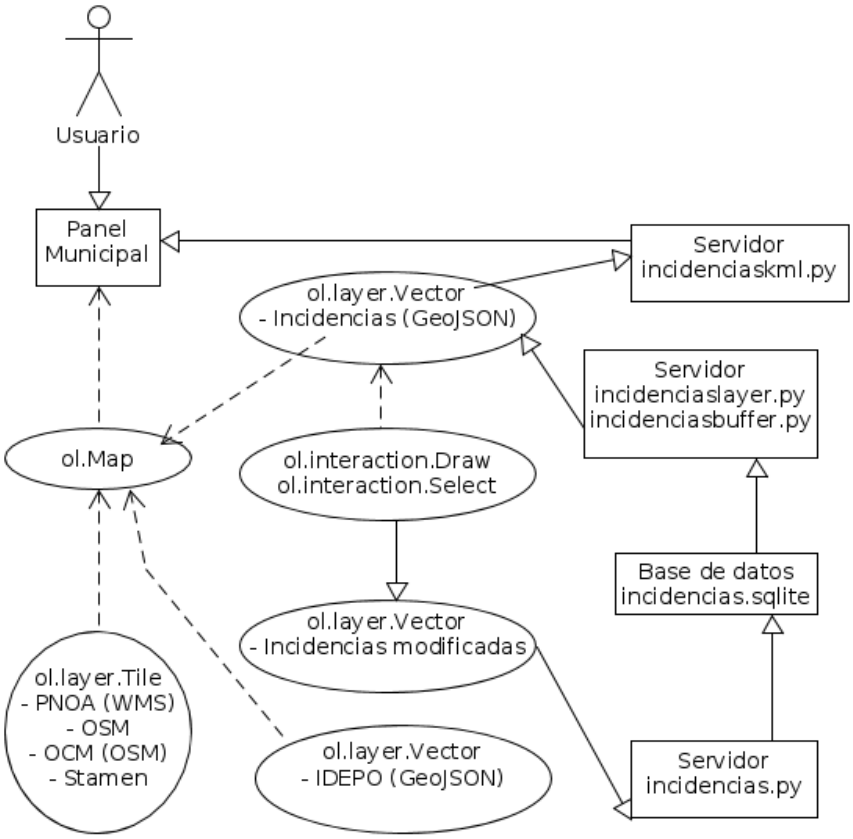


Figura 3.1: Esquema de funcionamiento de la aplicación

3.3. Conexión de la interfaz web a las fuentes de datos geográficos utilizadas

3.3.1. Obtención de mapas rasterizados basados en OpenStreetMap

OpenStreetMap es una base de datos geográficos accesible a través de una API RESTful. Los datos vectoriales que contiene están bajo una licencia ODbL

("*Open Data Commons Open Database License*"), de manera que pueden ser usados libremente para cualquier finalidad, y mejorados con las contribuciones de todos los que lo deseen.

Existen diferentes aplicaciones y servicios web que transforman esos datos vectoriales en mapas rasterizados, por ejemplo la propia interfaz web de OpenStreetMap.org u otros servicios externos, como por ejemplo MTBMap, que producen mapas personalizados basados en esa misma base de datos. Los mapas rasterizados pueden ser usados remotamente: están disponibles en Internet a través de servidores que los disponiblizan en URLs bajo la forma:

```
http://server/Z/X/Y.png
```

donde Z es el nivel de zoom, y X (columna) e Y (fila) son coordenadas que definen la posición de una determinada cuadrícula del mapa en ese nivel de zoom, empezando por la esquina superior izquierda del mapa. Como ejemplo, se puede obtener una imagen de Galicia al nivel 6 de zoom en el siguiente URL:

```
http://tile.openstreetmap.org/6/30/23.png
```

Cada cuadrícula es una imagen de 256x256 píxeles en formato PNG [21]. Su uso debe tener en cuenta la política de uso de OpenStreetMap [22] – o la de otros servidores de mapas, en su caso – debido al importante volumen de tráfico que generan en los servidores.

3.3.2. Obtención de ortofotos del Plan Nacional de Ortofotografía Aérea (PNOA)

La Infraestructura de Datos Espaciales de España [23] ofrece, entre otros servicios, el acceso remoto al PNOA a través de un servicio WMS accesible en:

```
http://www.idee.es/wms/PNOA/PNOA?REQUEST=GetCapabilities
```

Ese servicio, según se indica en la información suministrada por el servidor WMS, puede ser desactivado en cualquier momento, por lo que debe pasar a usarse el siguiente nuevo acceso:

```
http://www.ign.es/wms-inspire/pnoa-ma?REQUEST=GetCapabilities
```

Las ortofotos obtenidas son las más recientes disponibles para la zona, a partir de una escala 1:70000 (para escalas más pequeñas, con menos detalle, se sirven imágenes de satélite). El período de actualización de las imágenes es de 2 o 3 años, y la máxima resolución es de entre 25 y 50 cm. Toda la información sobre este servicio se puede obtener en el portal web del PNOA [24].

3.3.3. Obtención de datos sobre el municipio procedentes de la Infraestructura de Datos espaciales de la Diputación de Pontevedra

Para el desarrollo de la aplicación, consideraremos como ejemplo un municipio de la provincia de Pontevedra, el municipio de Tomiño. La Diputación de Pontevedra disponibiliza a través de su geoportal [18] diversos datos espaciales sobre los municipios de la provincia (infraestructuras, instalaciones, edificios públicos, etc.). Los datos se pueden descargar separados para cada municipio, o bien acceder a ellos mediante un servicio WFS accesible en:

```
http://ide.depo.es:8080/geoserver/idepo/wfs?REQUEST=GetCapabilities
```

Aunque OpenLayers nos permitiría el acceso directo a los datos del servidor WFS, esa estrategia se revela como excesivamente lenta y pesada al realizar pruebas de rendimiento, pues exige una conexión a Internet rápida y una máquina con elevada velocidad de procesamiento. Por ello, se ha optado por la descarga de los datos correspondientes al municipio en cuestión, que después de convertidos a formato GeoJSON utilizando QGIS se pueden usar como fuente para una capa vectorial de OpenLayers.

Además del acceso WFS, la Diputación de Pontevedra disponibiliza la descarga de los datos en formato ESRI Shapefile para cada municipio de la provincia. Una vez descargados, pueden ser abiertos en QGIS y exportados a formato GeoJSON. Los 21 ficheros ESRI Shapefile con datos que disponibiliza el Geoportal se incluyen en un fichero `idepo.js`, constituyendo cada uno de ellos un objeto GeoJSON del tipo `FeatureCollection`, y siendo todos ellos agregados a un array al final del fichero:

```
idepogeojson = [captacion_enc, casa_consistorial, cementerio, cent_cultural,
centro_ensenanza, centro_sanitario, deposito_enc, depuradora_enc, infraestr_viaria,
instal_deportiva, lonja_merc_feria, nucleo_poblacion, parque, potabilizacion_enc,
proteccion_civil, ramal_saneamiento, red_distribucion, tramo_carretera,
tramo_colector, tramo_conduccion, tramo_emisario]
```

A partir de ese array, se pueden leer todos los objetos geográficos ("*features*") contenidos en todas las colecciones para añadirlos a una capa vectorial del mapa en OpenLayers. A diferencia de otras capas, la iniciaremos como capa no visible, pues la gran cantidad de objetos que contiene (el fichero `idepo.js` tiene unas tres mil líneas) hace que el funcionamiento de la aplicación se ralentice en máquinas con menos recursos, pues requieren mucho trabajo del procesador para su renderización gráfica.

4. Conexión de la interfaz web a la base de datos

Una vez que disponemos del mapa con las diferentes capas base en formato ráster (las ortofotos del PNOA, los mapas de OpenStreetMap y de OpenCycleMap, pues estos últimos disponen de curvas de nivel, y los de Stamen, que permiten imprimir con ahorro de tinta) y vectorial (los datos del Geoportal de la Diputación de Pontevedra), sólo resta conectar la interfaz a un servidor donde residirá la base de datos, y que puede estar situado en la misma máquina en la que se ejecuta el programa cliente dentro del navegador, o bien en otra máquina de la red que sirve mediante el protocolo HTTP tanto los ficheros que ejecutan el programa cliente en el navegador (`panelmunicipal.html`, `panelmunicipal.css` y `panelmunicipal.js`) como el acceso a la base de datos a través de *scripts* escritos en el lenguaje de programación Python y que se ejecutan en el servidor HTTP utilizando la *Common Gateway Interface* (CGI).

Python es el lenguaje de programación que resulta más conveniente para esta tarea por disponer de cuatro módulos que la facilitarán:

- El módulo `Pyspatialite` que permite acceder a una base de datos `SpatiaLite` utilizando la API estándar de Python para el acceso a bases de datos [25].

- El módulo `geojson` [26] que implementa todos los tipos de objetos geográficos definidos en la especificación del formato GeoJSON [27] e incluye utilidades para la conversión de una cadena de texto en dicho formato a objetos Python y viceversa.

- El módulo `CGIHTTPServer` que permite ejecutar un servidor HTTP con soporte CGI simple de forma sencilla, para poder realizar pruebas del funcionamiento de la aplicación sin necesidad de configurar un servidor web más complejo.

- El módulo `cgi`, que disponibiliza los datos enviados mediante los métodos `POST` o `GET` de los formularios HTML.

El componente servidor consiste en los siguientes cuatro *scripts* escritos en Python, que aceptan como entrada cadenas de texto enviadas mediante los métodos `POST` o `GET`:

incidencias.py	Acepta datos en formato GeoJSON identificados como `geojsondata` que contengan un objeto del tipo `FeatureCollection`, y para cada objeto del tipo `Feature` contenido en ella realizan una operación `INSERT` en la base de datos (si su propiedad "`id`" no existe en la base de datos) o una operación `UPDATE` (en caso contrario).
incidenciaslayer.py	Cuando no recibe datos, devuelve en la salida el contenido íntegro de la base de datos en formato GeoJSON. Acepta como datos de entrada una cadena de texto identificada como `query`, y en ese caso añade a la operación `SELECT` esa cadena de texto como filtro, precedida de `WHERE`, devolviendo así en la salida solamente los registros de la base de datos que cumplen los criterios especificados en la cadena `query`.
incidenciasbuffer.py	Realiza las mismas funciones que `incidenciaslayer.py` pero substituyendo el punto contenido en el campo `geometry` por un polígono resultante de aplicar la operación `Buffer` sobre el punto utilizando el valor del campo `radio`.
incidenciaskml.py	Acepta datos en formato KML y los devuelve al navegador cliente especificando un nombre y tipo de fichero, de manera que puedan ser descargados en forma de fichero.

La operación espacial realizada por el *script* `incidenciasbuffer.py` sobre el valor del campo `geometry` es la siguiente:

```
Transform(Buffer(Transform(geometry, 23029), radio), 4326)
```

Nótese que, dado que los datos están almacenados en el SRS EPSG:4326 que utiliza grados como unidad y el valor del campo `radio` está expresado en metros, es necesario aplicar una transformación de SRS antes de realizar la operación `Buffer`: se transforma el valor del punto al SRS EPSG:23029, que utiliza los metros como unidad, se realiza la operación `Buffer`, y se transforman de nuevo las coordenadas del polígono resultante al EPSG:4326.

4.1. Formatos de intercambio de datos geográficos

Para el intercambio de objetos geográficos (`features`) entre la parte cliente (OpenLayers) y la parte servidor (la base de datos Spatialite) utilizaremos el

formato GeoJSON, que es el más fácilmente procesable en JavaScript y que está completamente soportado tanto por OpenLayers como por el lenguaje Python con el que se interactúa con la base de datos.

El formato GeoJSON, definido actualmente en un borrador de RFC [27], es un formato de intercambio de datos espaciales que utiliza JSON (*JavaScript Object Notation*, RFC 7159) [28] para la codificación de objetos geográficos. Define un tipo de objeto denominado `Geometry`, que puede ser de varios tipos (`Position`, `Point`, `MultiPoint`, `LineString`, `MultiLineString`, `Polygon` o `MultiPolygon`) y que contiene un miembro denominado "`coordinates`" que indica su posición geográfica mediante un array de valores númericos que corresponden a sus coordenadas. El objeto `Geometry` es uno de los miembros del objeto `Feature`, que es un objeto geográfico que además de `Geometry` contiene un número indeterminado de miembros de cualquier tipo que forman parte de él, y que consiste típicamente en valores de otros campos asociados a él en cualquier base de datos que almacene datos geográficos. Los objetos del tipo `Feature` se pueden agrupar en un array en el interior de otro objeto denominado `FeatureCollection`. Para el intercambio de datos entre el componente cliente y el componente servidor de la aplicación necesitaremos utilizar objetos de los tipos `FeatureCollection`, `Feature`, `Point` y `Polygon`. Los *script* Python que devuelvan datos en este formato deberán precederlos del encabezado `Content-type: application/json`.

Para la exportación de los datos a solicitud del usuario, para su uso en otras aplicaciones de procesamiento de información geográfica o para su publicación en web, se utilizará el formato KML, que es también completamente soportado por OpenLayers.

El formato KML (*Keyhole Markup Language*), desde su versión 2.2 (cf. [29], actualmente la última versión es la 2.3) es uno de los formatos estándar del OGC para el intercambio de datos geográficos. Se trata de un lenguaje de etiquetado basado en XML que está próximo de GML – siendo de hecho una de las intenciones del OGC una mayor aproximación entre ambos formatos – y que se centra en la visualización de datos geográficos en tres dimensiones, incluyendo anotaciones e imágenes asociados a ellos, y describiendo no solamente su presentación sobre el mapa, sino también el recorrido que el usuario debe hacer en su visualización. Denominado "Keyhole" por ser originariamente el formato de almacenamiento de datos geográficos utilizado por un producto de la empresa "Keyhole" (comprada por Google) que fue el precursor inmediato de Google Earth, es actualmente el formato utilizado para la presentación y visualización de datos geográficos en esta aplicación de Google.

Será usado en el Panel Municipal únicamente para la exportación de datos a un formato visualizable por Google Earth o por otras aplicaciones de información geográfica disponibles, o para su publicación en la web. Para este uso, hay que tener en cuenta que el *script* Python que lo procese deberá precederlo cuando lo devuelva al navegador de un encabezado `Content-type` como el siguiente:

```
Content-type: application/vnd.google-earth.kml+xml
```

De esa manera, el fichero enviado será reconocido por el navegador para ejecutar la aplicación o *plugin* que tenga asociados a KML (podrá, por ejemplo, abrir directamente el fichero en Google Earth si esa aplicación está instalada).

5. Desarrollo futuro de la aplicación

Concluido el proyecto, se dispone de una aplicación funcional que cumple con los requisitos enumerados en las secciones 1.1 y 1.2 de la introducción, pero que no obstante podría ser mejorada en diferentes aspectos que se refieren a continuación y que se plantean como posibles desarrollos futuros:

1. La introducción de la fecha y hora de inicio y fin de una incidencia debe hacerse manualmente, en el formato especificado, de forma no intuitiva y que requiere un mínimo aprendizaje. Podría mejorarse la aplicación introduciendo menús o un calendario clicable para el establecimiento de la fecha y hora de inicio y fin de las incidencias.

2. El filtro que se aplica a los datos de las incidencias debe hacerse en lenguaje SQL, de forma no intuitiva y que requiere un mínimo aprendizaje. Podría mejorarse la aplicación introduciendo un formulario más intuitivo para aplicar ese filtro.

3. Una funcionalidad útil, que la aplicación no tiene, es la posibilidad de realizar operaciones espaciales combinando los datos de la base de datos de incidencias y los datos en formato GeoJSON contenidos en el fichero `idepo.js`, para obtener así nuevos datos sobre la interacción entre las incidencias y las infraestructuras del municipio.

4. La inclusión de datos vectoriales diferentes de los suministrados en el fichero `idepo.js` requiere la edición manual de este fichero y conocimientos básicos de programación en JavaScript. La aplicación podría mejorarse permitiendo la carga de forma automática de cualquier fichero en formato GeoJSON que se colocase en determinado directorio.

5. Los datos en formato GeoJSON del fichero `idepo.js` deben utilizar necesariamente el SRS EPSG:23029. La funcionalidad referida en el punto anterior mejoraría aún más si incluyese también la posibilidad de utilizar datos en SRS diferentes.

6. La exportación de datos se realiza únicamente a formato KML. Se podría mejorar esta funcionalidad dando al usuario la opción de escoger para la exportación entre diferentes formatos soportados por la biblioteca OpenLayers (KML, GML, GeoJSON o incluso GPX, considerando cada incidencia un *waypoint*).

6. Conclusiones

El desarrollo de una aplicación como el "Panel Municipal" de incidencias que es objeto del presente trabajo, recurriendo únicamente a tecnologías web estándar (HTML+CSS+JavaScript), software libre, datos abiertos y formatos interoperables, muestra la madurez del software libre de gestión de información geográfica disponible hoy en día. Todo el software necesario para el funcionamiento de una aplicación como esta está disponible en las más extendidas distribuciones de GNU/Linux, gracias a iniciativas como el proyecto Debian GIS [14], o bien puede obtenerse fácilmente a través de Internet.

Es por ello posible disponer de pequeñas aplicaciones de gestión de información geográfica para las necesidades básicas de pequeñas organizaciones, como un ayuntamiento, simples pero potentes, sin necesidad de costosas inversiones en software propietario, y ejecutarlas en cualquier navegador web estándar, disponible hoy en día en cualquier puesto de trabajo de la intranet de la organización o incluso en dispositivos móviles como tabletas.

Para desarrollar una aplicación como esta es necesario, no obstante, escoger con cuidado los recursos utilizados, como hemos venido mostrando a lo largo de esta memoria. Las sucesivas elecciones que se vayan haciendo condicionan el desarrollo posterior de otros componentes. Así, la elección de una base de datos SpatiaLite para el almacenamiento de las incidencias evita la necesidad de configurar un servidor PostgreSQL, pero condiciona la versión de Python que se podrá usar posteriormente en el proyecto (versión 2.7 como máximo); la elección de la biblioteca OpenLayers disponibiliza inmediatamente las funcionalidades de creación y edición de nuevos objetos geográficos, que en Leaflet necesitan *plugins* externos, pero exige un mayor esfuerzo en la elaboración del código JavaScript de la aplicación; y la utilización de una fuente de datos como la Infraestructura de Datos Espaciales de la Diputación de Pontevedra obliga a seleccionarlos previamente y convertirlos a un nuevo formato, en vez de usarlos directamente a través de un servicio WFS, pues su enorme volumen ralentizaría considerable el funcionamiento de la aplicación si se cargasen a través de Internet.

El balance final es positivo: es posible desarrollar una aplicación como la propuesta en un tiempo reducido solamente con tecnologías web y software libre, frente a otras soluciones propietarias más costosas y pesadas y menos interoperables. Y con mayor tiempo de desarrollo se podrían añadir funcionalidades como las indicadas en el capítulo 5 para facilitar el uso de la aplicación y permitir una mayor cantidad de operaciones espaciales entre los datos utilizados.

Glosario

ACID *Atomicity, Consistency, Isolation, Durability*. Acrónimo aplicado a un modelo de acceso a las bases de datos relacionales que busca garantizar que todas las transacciones se ejecutan de forma aislada, completa y consistente y nunca de forma parcial o incompleta, evitando así la pérdida de datos, si bien a costa de ralentizar los accesos, pues en este modelo no es posible realizar diversas transacciones de forma simultánea en un entorno multiusuario.

CGI *Common Gateway Interface*. Se trata de un estándar web, uno de los más antiguos en Internet, que permite que los servidores HTTP ejecuten programas, típicamente escritos en lenguajes interpretados como Perl o Python y que reciben parámetros a partir de los métodos `POST` y `GET` de los formularios HTML, y devuelvan al navegador cliente el resultado de esa ejecución.

ED50 *European Datum 1950*. Sistema de coordenadas y puntos de referencia basado en el elipsoide internacional de 1924 (elipsoide de Hayford) que fue ampliamente usado en toda Europa desde el final de la Segunda Guerra Mundial hasta finales del siglo XX, cuando se comenzó su progresiva sustitución por el ETRS89 – *European Terrestrian Reference System 1989*. Sin embargo, muchos de los datos geográficos disponibles generados por instituciones públicas o empresas privadas en España y otros países de Europa siguen utilizando algún SRS basado en ED50, como es el caso de los datos de la Diputación de Pontevedra utilizados en el presente proyecto.

EPSG *European Petroleum Survey Group*. Organización internacional vinculada a la industria del petróleo que existió hasta el año 2005, cuando fue integrada en la IOGP (International Association of Oil and Gas Producers). Sin embargo, sus siglas EPSG siguen siendo usadas para referirse a la base de datos de SRS creada y mantenida por esta organización, que es la base de todos los procesos de conversión de coordenadas entre sistemas de referencia mediante software y que, bajo la denominación "EPSG Geodetic Parameter Dataset", sigue siendo gestionada por la IOGP, públicamente disponible en Internet en `http://www.epsg-registry.org`. A partir de la información contenida en ella, han surgido otros servicios web que permiten realizar consultas, conversiones y exportar los datos de cada SRS en diferentes formatos utilizados por el software de procesamiento de datos geográficos. El más habitualmente utilizado es `http://epsg.io`.

ESRI Shapefile Formato de almacenamiento de datos geográficos utilizado por el software propietario ArcGIS, de la empresa ESRI. Por ser uno de los primeros formatos difundidos, se ha convertido en

un estándar *de facto* del almacenamiento e intercambio de datos geográficos en los sistemas de información geográfica, y la mayoría de los repositorios disponibles – incluso de datos libres y abiertos – lo utilizan.

EWKT *Extended Well Known Text*. Es una mejora del formato WKT introducida por PostGIS y soportada por diversas herramientas de código abierto, por lo que puede ser considerado de alguna manera un estándar *de facto*, aunque no forme parte de los estándares del OGC.

GeoJSON *Geographical JSON*. Se trata de un formato de intercambio de datos geoespaciales basado en JSON [27]. Define un conjunto de tipos de objetos JSON para la representación de objetos geográficos, incluyendo sus propiedades y su localización en el espacio mediante coordenadas. Recomienda el uso exclusivo del SRS WGS84 para la especificación de coordenadas, aunque permite el uso de otros SRS.

GIS *Geographic Information System*. Un sistema de información geográfica (también conocido por las siglas SIG en español) es un sistema – integrado habitualmente por diferentes componentes de hardware y software – que permite el almacenamiento, manipulación, uso y análisis de datos que contienen referencias espaciales – es decir, objetos geométricos geolocalizados –, la puesta en valor de esos datos y la creación de conocimiento nuevo a partir de ellos. El OGC ofrece en el glosario de su portal web, en `http://www.opengeospatial.org/taxonomy/term/13`, la siguiente definición: "*A computer system for capturing, storing, checking, integrating, manipulating, analyzing and displaying data related to positions on the Earth's surface.*"

GML *Geography Markup Language*. Es el formato estándar del OGC para la representación de información geográfica en formato XML, y es el que usan habitualmente los servidores WMS y WFS. Están en uso dos versiones diferentes y con diferencias importantes, 2.x y 3.x, diferenciadas por ello habitualmente en los GIS y en el software para el procesamiento de datos geográficos.

JSON *JavaScript Object Notation*. Es un formato de intercambio de datos que consiste en cadenas de texto estructurado que representan objetos constituidos por pares de clave y valor denominados "miembros" [28]. Las cadenas de texto estructurado ignoran los espacios entre los delimitadores, de manera que pueden ser mostradas de forma fácilmente comprensible por un ser humano sin dejar de ser al mismo tiempo fácilmente procesables por lenguajes de programación. Está basado en JavaScript y es el formato de elección para el intercambio de datos en la web entre aplicaciones desarrolladas mediante JavaScript.

KML *Keyhole Markup Language*. Es uno de los formatos estándar del OGC para el intercambio de datos geográficos, desde su versión 2.2 (cf. [29]), siendo la última versión disponible la 2.3. Consiste en un lenguaje de etiquetado basado en XML, próximo de GML – siendo de

hecho una de las intenciones del OGC desde que lo aceptó como estándar una mayor aproximación entre ambos formatos – y que se centra en la visualización de datos geográficos en tres dimensiones, incluyendo anotaciones e imágenes asociados a ellos, y describiendo no solamente su presentación sobre el mapa, sino también el recorrido que el usuario debe hacer en su visualización. El nombre "Keyhole" procede de su creación para un producto de la empresa "Keyhole" (comprada por Google) que fue el precursor inmediato de Google Earth. Es actualmente el formato utilizado para la presentación y visualización de datos geográficos en esta aplicación de Google.

OGC *Open Geospatial Consortium*. Es un consorcio, creado en 1994, que agrupa diversas empresas públicas y privadas para la definición de estándares que permitan la interoperabilidad de los datos geográficos. En el desarrollo de la aplicación que es objeto de este trabajo se utilizan para una función u otra los siguientes estándares del OGC (referidos también en el presente glosario): OGC-SFS, WKT, WMS y KML. Para más información, debe consultarse el portal web del OGC en `www.opengeospatial.org`.

OGC-SFS *Open Geospatial Consortium - Simple Feature SQL*. Se trata de un estándar [8] del *Open Geospatial Consortium* que especifica los tipos de objetos geográficos que deben ser implementados por los sistemas de bases de datos relacionales que utilicen SQL.

PNOA *Plan Nacional de Ortofotografía Aérea* del Estado español. Obtiene y disponibiliza públicamente ortofotografías aéreas digitales con una resolución de entre 25 y 50 centímetros de todo el territorio español, actualizadas periódicamente cada dos o tres años, según las zonas. Dispone de un portal en `http://pnoa.ign.es` y de un servicio WMS en `http://www.ign.es/wms-inspire/pnoa-ma?REQUEST=GetCapabilities`.

SQL *Structured Query Language*. Lenguaje utilizado para interactuar con las bases de datos relacionales, tanto para obtener datos de ellas mediante consultas como para realizar modificaciones en su contenido. Es un estándar ISO (ISO/IEC 9075) cuya última actualización data de 2011 (vid. ISO/IEC 9075-1:2011 en `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53681`).

SRID *Spatial Reference Identifier*. Se denomina así a los identificadores, generalmente constituidos por un valor numérico y una autoridad o base de datos de referencia, que sirven para referirse a los SRS en los sistemas de información geográfica. EPSG:4326 o EPSG:23029 son SRID que identifican dos de los SRS utilizados en el presente proyecto mediante el número que los representa en la base de datos de referencia del EPSG.

SRS *Spatial Reference System*. Un Sistema de Referencia Espacial es un sistema que establece la proyección utilizada para dibujar la superficie de la tierra sobre un mapa y las unidades y punto de partida en

el que se expresan las coordenadas utilizadas para la localización de objetos geográficos bidimensionales sobre dicha superficie.

UTM *Universal Transverse Mercator*. Se trata de un SRS que divide la tierra en 60 husos de 6 grados de longitud de anchura cada uno, en cada uno de los cuales se aplica la proyección transversal de Mercator sobre un meridiano, y las coordenadas se miden en metros y no en grados de latitud y longitud como en el caso de WGS84. La Península Ibérica está situada en los husos UTM 29, 30 y 31, quedando el territorio que se usa como ejemplo en la aplicación y del que se obtienen datos geográficos públicos (el sur de Galicia) en el huso UTM 29, cuyo meridiano central sobre el que se aplica la proyección es el -9. El sistema UTM también define bandas de 8 grados de latitud representadas por letras, quedando el territorio referido en la banda T, de manera que la zona UTM correspondiente es la UTM 29T.

W3C *World Wide Web Consortium*. Comunidad internacional dedicada al desarrollo y publicación de estándares web que permitan el desarrollo pleno del potencial de los navegadores web como plataforma de acceso a la información y de ejecución de aplicaciones, garantizando al mismo tiempo la interoperabilidad de las aplicaciones desarrolladas con esta tecnología. Se puede obtener más información sobre el consorcio y sobre sus estándares en su portal web, en `https://www.w3c.org`.

WFS *Web Feature Service*. Se trata de un servicio web estandarizado por el OGC, que consiste en ofrecer a través de un servidor web acceso dinámico a datos que consisten en objetos geográficos ("*features*") vectoriales, que se sirven habitualmente en formato GML o GeoJSON. El servicio puede o no ofrecer la posibilidad de edición y modificación de los datos.

WGS84 *World Geodetic System 84*. Denominado también EPGS:4326, es el SRS utilizado para las coordenadas de latitud y longitud de los sistemas GPS, y es por ello el más conocido y el más legible para los usuarios no especializados.

WKB *Well Known Binary*. Es el formato estándar del OGC para la representación de información geográfica en formato binario.

WKT *Well Known Text*. Es el formato estándar del OGC para la representación textual de información geográfica.

WMS *Web Map Service*. Se trata de un servicio web estandarizado por el OGC, que consiste en la generación, para acceso a través de un servidor web, de mapas georreferenciados que se actualizan de forma dinámica, a medida que van siendo solicitados por el programa cliente, en formato ráster: los mapas servidos son imágenes, en formatos como JPG, PNG, GIF o incluso SVG, que corresponden cada una a una cuadrícula georreferenciada, formando un mosaico.

Referencias

[1] Free Software Foundation. GNU Free Documentation License. [En línea], 2008. URL https://www.gnu.org/licenses/fdl.html. [Consultado el 6 de junio de 2016].

[2] Free Software Foundation. GNU General Public License. [En línea], 2007. URL https://www.gnu.org/licenses/gpl.html. [Consultado el 6 de junio de 2016].

[3] Ramsey, Paul et alii. PostGIS. [En línea], 2016. URL http://postgis.net. [Consultado el 6 de junio de 2016].

[4] Furieri, Alessandro. SpatiaLite. [En línea], 2016. URL https://www.gaia-gis.it/fossil/libspatialite/index. [Consultado el 6 de junio de 2016].

[5] AA.VV. OpenLayers. [En línea], 2016. URL http://openlayers.org. [Consultado el 6 de junio de 2016].

[6] Agafonkin, Vladimir. Leaflet. [En línea], 2015. URL http://leafletjs.com. [Consultado el 6 de junio de 2016].

[7] AA.VV. jQuery. [En línea], 2016. URL https://jquery.com/. [Consultado el 6 de junio de 2016].

[8] Open Geospatial Consortium. *OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option. Version: 1.2.1*. Open Geospatial Consortium, 2010. URL http://www.opengeospatial.org/standards/sfs.

[9] AA.VV. PostgreSQL. [En línea], 2016. URL http://www.postgresql.org/. [Consultado el 6 de junio de 2016].

[10] AA.VV. SQLite. [En línea], 2016. URL https://www.sqlite.org. [Consultado el 6 de junio de 2016].

[11] Furieri, Alessandro. SpatiaLite 4.3.0: SQL functions reference list. [En línea], 2016. URL https://www.gaia-gis.it/gaia-sins/spatialite-sql-4.3.0.html. [Consultado el 6 de junio de 2016].

[12] Ben Mearns. *QGIS Blueprints: Develop analytical location-based web applications with QGIS*. Packt Publishing, 2015. ISBN 978-1-78528-907-1.

[13] AA.VV. SQLite Documentation. [En línea], 2016. URL https://www.sqlite.org/docs.html. [Consultado el 6 de junio de 2016].

[14] AA.VV. Debian GIS Project. [En línea], 2016. URL https://wiki.debian.org/DebianGis. [Consultado el 6 de junio de 2016].

[15] AA.VV. QGIS. [En línea], 2016. URL http://qgis.org. [Consultado el 6 de junio de 2016].

[16] Chadwin, Tom. qgis2web. [En línea], 2016. URL https://github.com/tomchadwin/qgis2web. [Consultado el 6 de junio de 2016].

[17] Hocevar, Andreas. Trilogy of Error (or: Simplicity FTW). [En línea], 2016. URL http://ahocevar.net/openlayers/2016/02/17/trilogy-of-error.html. [Consultado el 6 de junio de 2016].

[18] Deputación de Pontevedra. Geoportal: Infraestructura de Datos Espaciales. [En línea], 2016. URL http://ide.depo.es. [Consultado el 6 de junio de 2016].

[19] Hocevar, Andreas et alii. proj4js. [En línea], 2016. URL http://proj4js.org. [Consultado el 6 de junio de 2016].

[20] Hocevar, Andreas. Proj4js 2.2.x with OpenLayers 3. [En línea], 2014. URL http://ahocevar.net/2014/07/10/proj4js-2-2-x-with-ol3.html. [Consultado el 6 de junio de 2016].

[21] OpenStreetMap. Slippy map tilenames. [Online], 2015. URL https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames. [Consultado el 6 de junio de 2016].

[22] OpenStreetMap. Tile usage policy. [Online], 2015. URL https://wiki.openstreetmap.org/wiki/Tile_usage_policy. [Consultado el 6 de junio de 2016].

[23] Instituto Geográfico Nacional. Infraestructura de Datos Espaciales de España. [En línea], 2016. URL http://www.idee.es. [Consultado el 6 de junio de 2016].

[24] Instituto Geográfico Nacional. Plan Nacional de Ortofotografía Aérea. [En línea], 2016. URL http://pnoa.ign.es. [Consultado el 6 de junio de 2016].

[25] AA.VV. pyspatialite: Python interface to Spatialite. [En línea], 2016. URL https://github.com/lokkju/pyspatialite/. [Consultado el 6 de junio de 2016].

[26] Farwell, Cory et alii. geojson: Python bindings and utilities for GeoJSON. [En línea], 2016. URL https://pypi.python.org/pypi/geojson/. [Consultado el 6 de junio de 2016].

[27] H. Butler, M. Daly, A. Doyle, S. Gillies, T. Schaub, and S. Hagen. *The GeoJSON Format*. IETF Internet Draft, 2016. URL https://datatracker.ietf.org/doc/draft-ietf-geojson/. [Consultado el 6 de junio de 2016].

[28] Bray, Ted (Ed.). *The JavaScript Object Notation (JSON) Data Interchange Format*. IETF RFC7159, 2014. URL https://www.ietf.org/rfc/rfc7159.txt. [Consultado el 6 de junio de 2016].

[29] Open Geospatial Consortium. *KML*. Open Geospatial Consortium, 2010. URL http://www.opengeospatial.org/standards/kml.

Anexos

A. Comandos SQL para la creación de la base de datos

A.1. Comandos SQL para la creación de la base de datos

```
BEGIN;

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

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

COMMIT;
```

code/incidencias.sql

A.2. Comandos SQL para comprobar el funcionamiento de la base de datos

```
BEGIN;

INSERT INTO Incidencias (id, tipo, radio, inicio, fin, geometry) VALUES
     (1460328181697, 'Incendio', 200, '2016-04-10T22:30+02:00', '
    2016-05-10T22:30+02:00', GeomFromText('POINT(-8.7373 42.0000)',
    4326));

SELECT id, tipo, radio, inicio, fin, AsGeoJSON(geometry) FROM
    Incidencias;

COMMIT;
```

code/test.sql

B. Manual de instalación y uso de la aplicación

B.1. Instalación

La aplicación "Panel municipal" requiere un servidor web con soporte CGI, un intérprete de Python (versión 2.7, pues es la última soportada por pyspatialite) y los módulos de Python `pyspatialite` y `geojson`. La mayoría de los servidores web actuales soporta CGI.

B.1.1. Ejecución con el módulo `CGIHTTPServer` de Python

La manera más simple de ejecutar la aplicación es utilizar el módulo de Python `CGIHTTPServer` mediante el script `startserver.sh` incluido, en un ordenador con sistema operativo GNU/Linux. En una distribución Debian, por ejemplo, es suficiente con instalar los siguientes paquetes:

- `python-pyspatialite`
- `python-geojson`

Dichos paquetes instalan como dependencias todo el resto del software necesario: una vez instalados, basta ejecutar el script `startserver.sh` para tener disponible la aplicación en el puerto 8000 de la máquina local y poder abrir la aplicación en cualquier navegador en la siguiente dirección:

```
http://localhost:8000/panelmunicipal.html
```

Para disponer de todas las funcionalidades, es necesario crear previamente la base de datos, tal como se detalla en el apartado B.1.3.

B.1.2. Instalación en otro servidor web

Para instalar la aplicación en otro servidor web, sólo es necesario cumplir con los siguientes requisitos:

- Es necesario activar el soporte CGI, en caso de que no lo esté por defecto, de acuerdo con las características del servidor web en concreto que se utilice.
- Los ficheros `panelmunicipal.html`, `panelmunicipal.css`, `panelmunicipal.js` e `idepo.js` deben encontrarse en el mismo directorio.
- Los ficheros del directorio `cgi-bin` deben estar accesibles en un subdirectorio del directorio anterior que tenga ese mismo nombre. La mayoría de los servidores web redirige ese directorio a otra localización, donde

deberán colocarse los *scripts* en Python que forman parte de la aplicación y se incluyen en ese directorio. Así, por ejemplo, la configuración por defecto del servidor Apache 2 en un sistema GNU/Linux redirige todas las llamadas a un subdirectorio `cgi-bin` al directorio `/usr/lib/cgi-bin`, donde deberán ser colocados los *scripts*.

- El fichero de la base de datos de incidencias, `incidencias.sqlite`, una vez creado, deberá tener permisos de ejecución y escritura por el usuario propietario del proceso bajo el que se ejecute el servidor web. Así, por ejemplo, en la configuración por defecto de un servidor Apache 2 en un sistema GNU/Linux, el fichero deberá pertenecer al grupo `www-data` y tener permisos de escritura y ejecución por los miembros de dicho grupo de usuarios, lo cual puede conseguirse con los comandos:

  ```
  chown root:www-data incidencias.sqlite

  chmod 0770 incidencias.sqlite
  ```

- Si el fichero de la base de datos de incidencias, `incidencias.sqlite`, no puede ser colocado en el mismo directorio que los *scripts* Python (por ejemplo, por ser un directorio sin permisos de escritura para el usuario bajo el que se ejecuta el servidor web), deberá existir en ese directorio un enlace a él. Así, por ejemplo, en el sistema GNU/Linux que nos sirve de ejemplo, y asumiendo que el fichero de la base de datos se encuentra en `/var/www/html`, puede crearse el enlace con el comando:

  ```
  ln -s incidencias.sqlite /var/www/html/incidencias.sqlite
  ```

B.1.3. Creación de la base de datos de incidencias

Para crear la base de datos de incidencias basta ejecutar el *script* `createdatabase.sh` en un sistema GNU/Linux que disponga de las utilidades binarias que se distribuyen bajo el nombre `spatialite-tools` en la página web de SpatiaLite [4]. En una distribución Debian, es suficiente con instalar el paquete `spatialite-bin`.

En otro tipo de sistemas, será necesario instalar esas utilidades precompiladas, o bien compilarlas. En alternativa, se pueden utilizar los comandos SQL del fichero `incidencias.sql` incluido con cualquier herramienta de gestión de bases de datos que disponga de la biblioteca SpatiaLite para interactuar con este tipo de bases de datos.

B.1.4. Personalización del comportamiento inicial de la aplicación

Se puede personalizar de forma básica el comportamiento de la aplicación, editando el fichero `panelmunicipal.html`, que contiene las siguientes variables:

```
<script type="text/javascript">
  var zoomlevel = 13;
  var defaultlat = 42.0000;
  var defaultlon = -8.7373;
  var pnoa = true;
```

```
  var osm = true;
  var ocm = true;
  var stoner = true;
  var idepo = true;
  var geojson = false;
</script>
```

Mediante la asignación de los valores `true` or `false` de las seis últimas variables se puede activar o desactivar respectivamente el uso de diferentes conjuntos de datos geográficos en la aplicación, de acuerdo con la siguiente tabla:

pnoa	Incluye como capa base las ortofotografías del Plan Nacional de Ortofotografía Aérea (PNOA) (cf. `http://pnoa.ign.es`)
osm	Incluye como capa base los mapas del proyecto OpenStreetMap (cf. `http://www.openstreetmap.org`)
ocm	Incluye como capa base los mapas del proyecto OpenCycleMap (cf. `http://www.opencyclemap.org`), que incluyen junto a los datos de OpenStreetMap curvas de nivel basadas en datos de la NASA que están en el dominio público.
stoner	Incluye como capa base un mapa en blanco y negro que se puede imprimir sin gran gasto de tinta de color, creado también a partir de los datos de OpenStreetMap (cf. `http://maps.stamen.com`).
idepo	Incluye una capa vectorial que muestra los datos contenidos en el fichero `idepo.js`. Se incluye un fichero con datos del municipio de Tomiño (Pontevedra), disponibles en la página de descarga de datos `http://ide.depo.es/datos.html` del Geoportal de la Infraestructura de Datos Espaciales de la Diputación de Pontevedra, pero podría contener cualquier conjunto de datos vectoriales en formato GeoJSON (cf. el apartado B.1.5).
geojson	Incluye un área de texto en la cual se visualizan en formato GeoJSON todos los datos de incidencias creados o modificados por el usuario. Es útil solamente para usuarios expertos que quieran cortar y pegar directamente esos datos en otras aplicaciones de gestión de información geográfica, por lo que está desactivada por defecto (cf. el apartado B.2.6).

Las tres primeras variables permiten determinar el lugar en el que estará inicialmente centrado el mapa y el nivel de *zoom* inicial de la visualización:

zoom	Un valor entero entre 1 (menor detalle, visualización de un mapamundi) y 18 (mayor detalle, visualización de un trozo de calle). El valor por defecto (13) es el más razonable para la visualización del área abarcada por un municipio.
defaultlat	Latitud del centro del mapa en grados, en el sistema de referencia EPSG:4326 (el que utilizan los dispositivos GPS).
defaultlon	Longitud del centro del mapa en grados, en el sistema de referencia EPSG:4326 (el que utilizan los dispositivos GPS).

Para determinar los valores pretendidos de latitud y longitud, pueden usarse los valores mostrados en la esquina superior izquierda del mapa en la aplicación al desplazar el ratón sobre el mapa, como se observa en la figura B.1.

Figura B.1: Longitud y latitud de la posición del ratón sobre el mapa

B.1.5. Inclusión de datos en el fichero `idepo.js`

El fichero `idepo.js` contiene un conjunto de datos en formato GeoJSON correspondientes al municipio de Tomiño (Pontevedra) y obtenidos del geoportal de la Infraestructura de Datos Espaciales de la Diputación de Pontevedra (IDEPO), pero puede contener cualquier conjunto de datos en formato GeoJSON que reúnan los siguientes requisitos:

1. Utilizar el SRS EPSG:23029 para especificar las coordenadas.

2. Incluir la definición de uno o más objetos del tipo *FeatureCollection* en formato GeoJSON de la siguiente forma:

   ```
   nombredelobjeto = {
   ...
   }
   ```

   Por ejemplo, el fichero contiene actualmente el objeto:

```
tramo_emisario = {
  "type": "FeatureCollection",
  "crs": {
    "type": "name",
    "properties": {
      "name": "urn:ogc:def:crs:EPSG::23029"
    }
   },
  "features": [
    {
    "type": "Feature",
    "properties": {
       "fase": "2011", "clave": "EM",
        "provincia": "36", "municipio": "054",
        "orden_emis": "001", "tipo_mat": "FC",
        "estado": "B", "long_terre": 511,
        "long_marit": 0, "gid": 1077
     },
     "geometry": {
       "type": "LineString",
       "coordinates": [
[ 520791.88892129494343, 4644275.724412054754794 ],
[ 521241.481215308711398, 4644032.202638223767281 ]
       ]
     }
   }
  ]
}
```

3. Incluir la definición de un array de objetos denominado `idepogeojson` que incluya todos los objetos del punto anterior, de la siguiente forma:

   ```
   idepogeojson = [objeto1, objeto2, ...]
   ```

   Por ejemplo, el fichero contiene actualmente el array:

   ```
   idepogeojson = [captacion_enc, casa_consistorial, cementerio,
   cent_cultural, centro_ensenanza, centro_sanitario,
   deposito_enc, depuradora_enc, infraestr_viaria,
   instal_deportiva, lonja_merc_feria, nucleo_poblacion, parque,
   potabilizacion_enc, proteccion_civil, ramal_saneamiento,
   red_distribucion, tramo_carretera, tramo_colector,
   tramo_conduccion, tramo_emisario]
   ```

B.2. Instrucciones de uso

Una vez completado el proceso de instalación, al abrir en un navegador el URL `http://localhost:8000/panelmunicipal.html` (si se ha utilizado el `CGIHTTPServer` de Python con el *script* `startserver.sh`) o bien un URL de la forma `http://servidor/camino/panelmunicipal.html` correspondiente a

un servidor web en el que se haya instalado la aplicación, visualizaremos lo que se puede ver en la imagen B.2. Nótese que, en todo caso, es imprescindible que el navegador tenga acceso a Internet, incluso cuando la aplicación se ejecuta en la máquina local con el *script* `startserver.sh`, pues la aplicación hace uso de bibliotecas de código y datos externas, a las que accede a través de la Web y sin las cuales no puede funcionar correctamente.

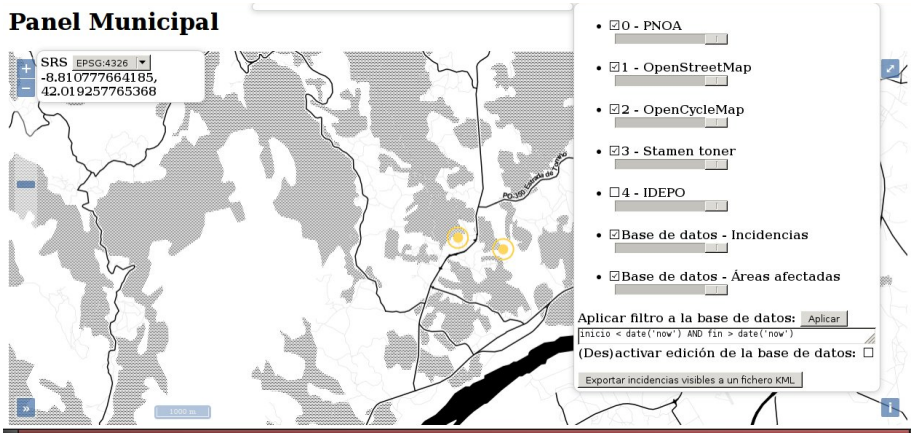


Figura B.2: Pantalla inicial de la aplicación

En esta pantalla inicial, disponemos en las esquinas del mapa de controles que nos permiten mostrar un pequeño mapa de contexto (para situar el municipio que vemos en el mapa en el contexto más amplio de la región en la que se sitúa), mostrar los créditos y la información de *copyright* de las diferentes capas que componen el mapa, cambiar el nivel de *zoom* para visualizar el terreno con mayor o menor detalle, y colocar la aplicación en modo de pantalla completa. La figura B.3 muestra dónde se sitúa cada uno de estos controles.

Figura B.3: Controles situados en las esquinas del mapa

B.2.1. Visualización y transparencia de las diferentes capas superpuestas

En el menú situado en el lado derecho de la pantalla se enumeran las diferentes capas apiladas sobre el mapa, comenzando por la inferior (número 0). Los conjuntos de datos a los que corresponde cada capa están especificados en el apartado B.1.4 de las instrucciones de instalación. Las dos capas no referidas allí corresponden a las incidencias introducidas en la aplicación: la denominada "Base de datos - Incidencias" muestra sobre el mapa los puntos en los que se han producido incidencias en forma de puntos color naranja, mientras que la denominada "Base de datos - Áreas afectadas" muestra sobre el mapa las zonas delimitadas por los círculos que tienen como centro las incidencias y como radio el valor del campo "radio" de cada incidencia almacenada. Estas áreas se muestran en forma de círculos color naranja de borde opaco y superficie translúcida.

Cada una de esas capas se puede activar o desactivar (y en ese caso deja de mostrarse sobre el mapa) mediante una casilla de selección que precede al número que la identifica, y es posible también modificar su opacidad mediante un interruptor deslizante situado bajo su nombre, para permitir visualizar diferentes capas superpuestas, como se aprecia en la imagen B.4, en la que se pueden ver las ortofotografías del PNOA bajo el mapa OpenStreetMap, cuyo nivel de opacidad se ha reducido.

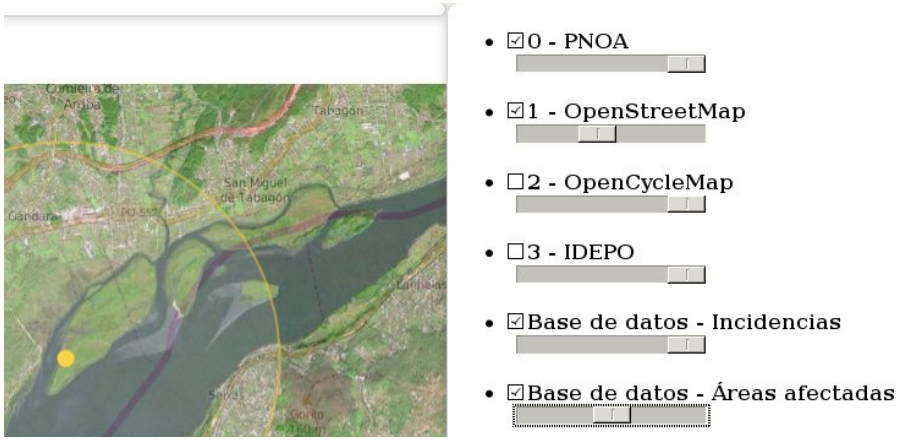


Figura B.4: Menú de capas con diferentes niveles de transparencia

B.2.2. Visualización de datos sobre las incidencias y sobre los objetos de la capa IDEPO

Al hacer click con el botón izquierdo del ratón sobre uno de los puntos color naranja de la capa "Base de datos - Incidencias", se abre un bocadillo sobre el mapa que muestra toda la información disponible sobre ella, como se observa en la figura B.5.

En la esquina superior derecha del bocadillo hay unas aspas que permiten cerrarlo. Si activamos la capa IDEPO, podemos también visualizar por el

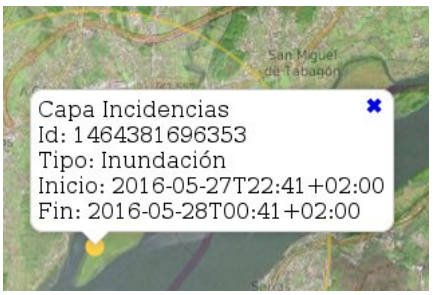


Figura B.5: Visualización de datos de una incidencia

mismo procedimiento toda la información disponible sobre cualquiera de los objetos geográficos que la componen. A modo de ejemplo, en la figura B.6 se muestran los datos disponibles sobre un núcleo de población del municipio pontevedrés de A Guarda.

Figura B.6: Visualización de datos de un objeto de la capa IDEPO

B.2.3. Creación y edición de incidencias

Además de todas las funcionalidades de visualización de datos enumeradas hasta ahora, el principal objetivo de la aplicación es la gestión de una base de datos de incidencias y actuaciones en el espacio público en el ámbito de un municipio. Al iniciarla, se muestran las incidencias que ya forman parte de la base de datos. Para poder crear nuevas incidencias o editar las ya existentes, es necesario marcar la casilla de selección situada en el menú a la derecha de la frase "(Des)activar edición de la base de datos". Al hacerlo, la posición del ratón sobre el mapa deja de mostrarse como una flecha, y pasa a mostrarse como un punto azul. Al hacer click con el botón izquierdo sobre el mapa, en el lugar donde esté situado el punto azul se creará una nueva incidencia en forma de punto color naranja, y se abrirá un menú que nos permite introducir

valores en los campos de datos asociados a esa incidencia (véase la figura B.7).

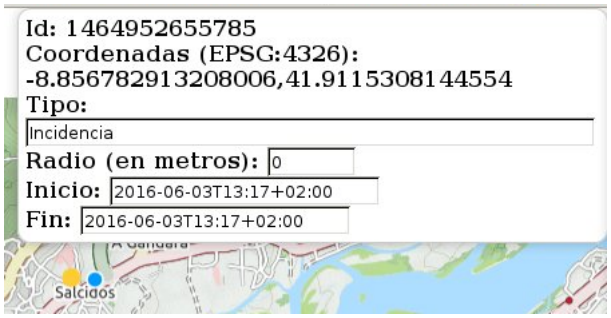


Figura B.7: Creación de una nueva incidencia

Los campos editables son "Tipo" (cualquier texto de un máximo de 50 caracteres), "Radio" (un número entero que expresa, en metros, el radio del área afectada por la incidencia) y la fecha y hora de inicio y fin previsto de la incidencia en los campos "Inicio" y "Fin", respectivamente. En los campos se muestran valores por defecto, que deben ser editados: la incidencia se guarda en la base de datos en el momento en que se altera el valor de cualquiera de los campos, y en ese momento aparecerá el mensaje "Incidencia [número] guardada" (siendo [número] el valor del campo "Id", que contiene un identificador único de la incidencia, que se genera automáticamente al crearla) bajo la frase "(Des)activar edición de la base de datos". A medida que se van alterando los valores de los campos, el mensaje cambia a "Incidencia [número] modificada".

El formato de los campos "Inicio" y "Fin" es YYYY-MM-DDTHH:MM[+-]HH:MM, formato en el que el valor situado después de [+-] indica el huso horario mediante la diferencia en horas en relación al tiempo universal. Esa diferencia horaria con el tiempo universal se genera automáticamente de acuerdo con la zona horaria establecida en el sistema, por lo que no es necesario modificar la que aparece en el valor por defecto del campo, que muestra la fecha y hora actuales en el huso horario del sistema. La letra T mayúscula actúa como separador entre la fecha y la hora. A modo de ejemplo, en la imagen anterior se nos muestra como valor por defecto del campo "Fin" el siguiente valor, que corresponde a las 13:17 del día 3 de junio de 2016 en España (horario de verano, con dos horas de diferencia sobre el tiempo universal):

```
2016-06-03T13:17+02:00
```

Para alterar su valor a las 23:00 del día 6 de junio, modificaremos el texto anterior para que se lea:

```
2016-06-06T23:00+02:00
```

Para editar una incidencia ya existente, debemos seleccionarla haciendo sobre ella doble click con el botón izquierdo del ratón mientras mantenemos pulsada la tecla `Ctrl`. Al hacerlo, se muestran en el formulario de edición los datos de la incidencia, como se puede ver en la figura B.8, y es posible modificar el valor de los campos. Al realizar cualquier modificación, quedará registrada en la base de datos y se visualizará el mensaje "Incidencia [número] modificada", tal como sucedía al crear una incidencia. Se dispone también de un botón

para borrar completamente de la base de datos el registro de la incidencia seleccionada.

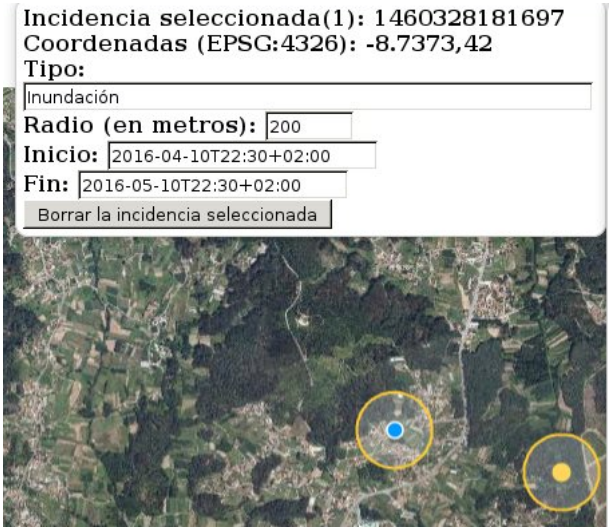


Figura B.8: Edición de una incidencia

B.2.4. Filtrado de las incidencias mostradas

Al iniciarse, la aplicación muestra en las capas correspondientes todas las incidencias contenidas en la base de datos y todas las áreas circulares afectadas, pero es posible aplicar un filtro a las incidencias mostradas. Para ello es necesario introducir en el área de texto "Aplicar filtro a la base de datos" un filtro adecuado y apretar el botón "Aplicar". En el área de texto se muestra por defecto un filtro que puede servir de ejemplo (véase la imagen B.9) y que será uno de los más utilizados, pues muestra únicamente las incidencias activas en el momento actual, es decir, aquellas cuya fecha y hora de inicio es anterior al momento actual y cuya fecha y hora de fin previsto es posterior al momento actual.

Figura B.9: Filtrado de incidencias por fecha y hora

La definición del filtro se realiza utilizando el lenguaje de consulta a bases de datos SQL: el contenido del área de texto es una expresión que se puede colocar en una consulta en lenguaje SQL después del elemento `WHERE`. Se ofrecen a continuación instrucciones básicas para poder crear un filtro de forma sencilla:

- Use cuatro operadores básicos de comparación: ==, `LIKE`, < y >. En el caso del operador LIKE, deberá colocarse un signo `%` como comodín antes y/o después de la cadena de texto que se desea comparar. Algunos ejemplos:
  - `tipo == 'Accidente'`
    Todas las incidencias cuyo campo `tipo` sea exactamente igual a "Accidente".
  - `tipo LIKE '%e'`
    Todas las incidencias cuyo campo `tipo` termine por "a".
  - `tipo LIKE 'a%'`
    Todas las incidencias cuyo campo `tipo` empiece por "a" (la búsqueda no distingue mayúsculas y minúsculas).
  - `tipo LIKE '%a%'`
    Todas las incidencias cuyo campo `tipo` contenga la letra "a".
- Use los conectores lógicos AND y OR. Por ejemplo:

  `tipo == 'Incendio' AND radio >1000`

  devolverá todas las incidencias cuyo campo `tipo` sea exactamente igual a "Incendio" con un área afectada de radio superior a un quilómetro.
- Para realizar filtros por fecha y hora se debe utilizar la función `date()` de SQLite, o bien para obtener la fecha y hora actuales con la expresión `date('now')` o bien para interpretar cualquier cadena del tipo de las utilizadas en los campos `inicio` y `fin` como fecha (por ejemplo `date('2016-06-01')`) o como fecha y hora `date('2016-06-01T23:00+02:00')`). Nótese que en este último caso es necesario indicar la diferencia horaria con el tiempo universal, pues en caso contrario la hora indicada se interpretará en tiempo universal. Algunos ejemplos:
  - `inicio <date('now') AND fin >date('now')` Todas las incidencias activas en el momento actual, es decir, con inicio anterior al momento actual y fin posterior al momento actual.
  - `inicio >date('2016-06-01') AND inicio <date('2016-07-01')` Todas las incidencias con inicio durante el mes de junio de 2016. Nótese que si indicamos sólo la fecha, la hora que se considera para la comparación son las 00:00 horas de la data indicada.

Si queremos volver a disponer del contenido íntegro de la base de datos de incidencias, sin filtrar, basta con borrar completamente el contenido del área de texto, y apretar el botón "Aplicar" para así aplicar un filtro vacío.

B.2.5. Exportación de incidencias visibles en formato KML

Por último, es posible exportar a un fichero en formato KML todas las incidencias que se muestren en el mapa, ya sea el contenido total de la base de datos, o bien el resultado de aplicar algún filtro de acuerdo con lo referido en el apartado anterior. Para ello basta apretar el botón "Exportar incidencias visibles a un fichero KML" para obtener dicho fichero.

B.2.6. Funcionalidades para usuarios expertos: obtención de datos en formato GeoJSON

Los usuarios expertos en el uso de sistemas de información geográfica pueden tener interés en obtener datos en formato GeoJSON para su uso en otras aplicaciones o sistemas. Es posible obtener datos en formato GeoJSON de tres maneras diferentes:

- Se puede obtener el contenido completo de la base de datos de incidencias en formato GeoJSON accediendo al URL `/cgi-bin/incidenciaslayer.py` relativo al directorio raíz del servidor. Así, por ejemplo, si iniciamos la aplicación accediendo a:

  `http://localhost:8000/panelmunicipal.html`

  podemos obtener todo el contenido completo de la base de datos accediendo a:

  `http://localhost:8000/cgi-bin/incidenciaslayer.py`

- Se pueden obtener también por el mismo procedimiento, en formato GeoJSON, los polígonos de forma circular que delimitan las áreas afectadas por las incidencias, resultado de aplicar la operación espacial `Buffer` a todas las incidencias, con el valor de su campo "radio", accediendo al URL `/cgi-bin/incidenciasbuffer.py`. En el ejemplo anterior, accediendo a:

  `http://localhost:8000/cgi-bin/incidenciasbuffer.py`

Figura B.10: Área de texto con datos en formato GeoJSON

- Si se ha configurado en el fichero `panelmunicipal.html` la variable `geojson = true` como se explica en el apartado B.1.4 de las instrucciones de instalación, se dispone de un área de texto en la aplicación, por debajo de la frase "(Des)activar edición de la base de datos", en la que se irán mostrando en formato GeoJSON todas las incidencias que sean creadas o editadas, como se puede observar en la imagen B.10. Se puede copiar el contenido del área de texto para pegarlo en cualquier otra aplicación de gestión de información geográfica capaz de procesar

el formato GeoJSON, o bien en un fichero de texto en el que almacenar los datos.