TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

“Una herramienta para la generación de capas a partir de información no georreferenciada”

Curso 2018/2019

Alumno/a:

Álvaro Ruiz Zamora

Director/es:

Javier Criado Rodríguez

Nicolás Padilla Soriano

AGRADECIMIENTOS

A mis directores, Javier Criado y Nicolás Padilla por el apoyo, ayuda y paciencia durante todo el proyecto, sobre todo las últimas semanas.

A mis padres y hermano por todo el apoyo moral y la confianza que tienen en mí.

A los amigos que han dado su apoyo y conseguido que me centre en acabar este proyecto.

Tabla de contenido

Capítulo 1

Introducción

1. Introducción

El presente trabajo de final de grado se refiere al desarrollo de una aplicación SIG, Sistema de Información Geográfica[1], o GIS por sus siglas en inglés. La información geográfica es un campo relativamente nuevo en informática que comenzó en la década de los 70 y, debido a ello, aún se siguen trabajando y mejorando las aplicaciones SIG existentes, en las que no se contemplan todas las funcionalidades útiles. En el caso de que queramos trabajar con información georreferenciada, debemos utilizar una herramienta adecuada para ello.

En primer lugar, debemos saber de qué se compone un sistema de información geográfica. Para que un sistema se considere un SIG ha de tener:

- Hardware: un equipo utilizado para almacenar, mostrar y procesar datos georreferenciados.
- Software: programas que permitan trabajar con los datos georreferenciados. Un software que forma parte de un SIG es una aplicación SIG.
- Datos: Información geográfica para mostrar y editar utilizando el software y el hardware.

Con una aplicación SIG se han de poder abrir mapas con información geográfica y crear nuevas capas información.

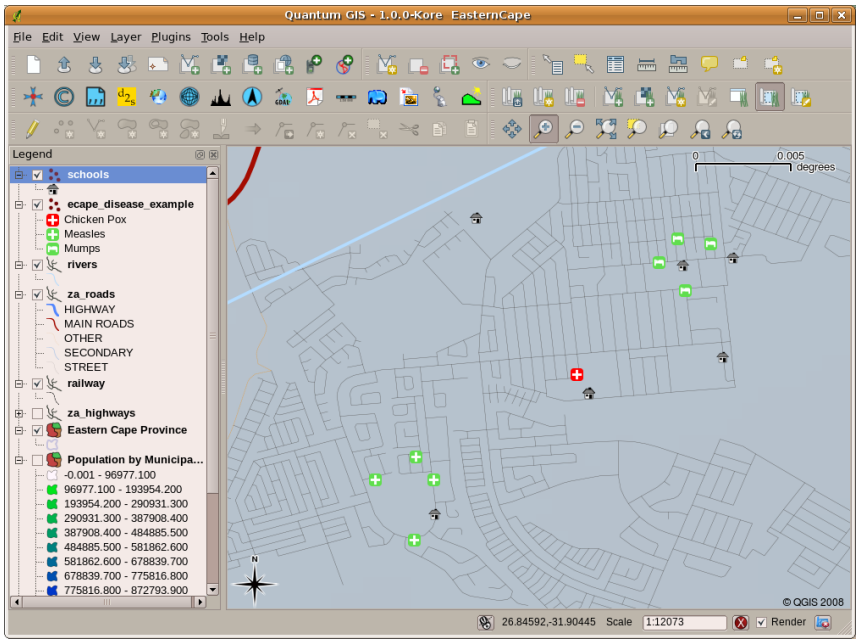


Figura 1: Capa con información geográfica

En la Figura 1 se puede observar un ejemplo de una capa de información sencilla con datos vectoriales. Pero no son los únicos tipos de datos que existen, tal y como se va a explicar en el siguiente punto.

[1] Una Ligera Introducción a GIS. (2019). Disponible en https://docs.qgis.org/2.8/es/docs/gentle_gis_introduction/index.html

1.1. Tipos de datos

En este punto veremos los distintos tipos de datos georreferenciados existentes: datos vectoriales (representaciones de objetos en un mapa) y datos ráster (dividir una capa en una matriz y agregar información a cada celda de esa matriz).

1.1.1. Datos vectoriales

Los datos vectoriales son una manera de representar objetos del mundo real dentro de un ambiente SIG. Los objetos que se pueden referenciar son cualquier elemento localizable, desde un edificio a una planta. Pongámonos en la situación en la que se quiera georreferenciar una zona de Almería[2] (Figura 2). Existen diferentes elementos que podríamos georreferenciar: árboles, farolas, piscinas, casas, carreteras, etc. Los árboles y farolas serían información del tipo punto; las casas y piscinas, polígonos; y las carreteras y senderos, polilíneas.

Figura 2: Imagen aérea de una zona de la ciudad de Almería

Cuando un objeto se puede representar con un solo vértice es un elemento **punto**. Si su representación consta de dos o más vértices, y el primer y último vértice son distintos, se trata

[2] Almería. (2019). Disponible en https://www.google.com/maps/place/Almer%C3%ADa/@36.8274171,-2.4388171,273m/data=!3m1!1e3!4m5!3m4!1s0xd7a9e00ecccf2c1:0x8d9da01f8ebc485e!8m2!3d36.834047!4d-2.4637136

de un elemento **polilínea**; y en el caso que el objeto se represente con tres o más vértices y el primer y último vértice son iguales, estamos ante un elemento **polígono**.

- Elemento punto[3]

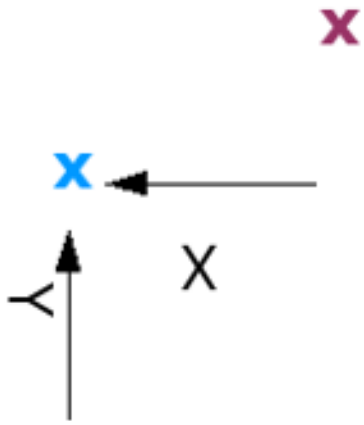


Figura 3: Elemento punto

Un objeto de tipo punto se describe por sus coordenadas X e Y (Figura 3). El punto es descrito mediante atributos, como cuando se representa un árbol o una farola.

- Elemento polilínea

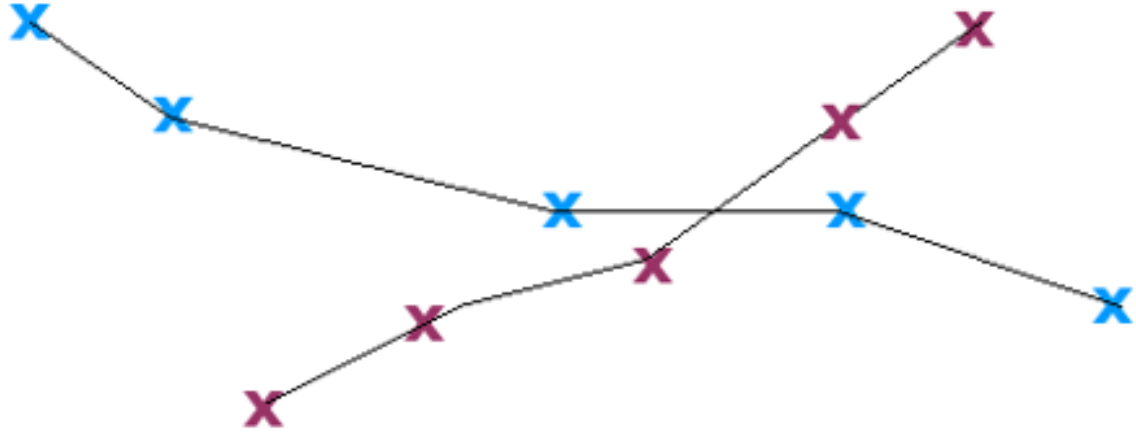


Figura 4: Dos elementos de tipo polilínea

Una polilínea es una secuencia de vértices unidos (Figura 4). Cada vértice posee unas coordenadas X e Y. Los atributos de los vértices describen la polilínea.

[3] Datos Vectoriales. (2019). Disponible en https://docs.qgis.org/2.8/es/docs/gentle_gis_introduction/vector_data.html

- Elemento polígono

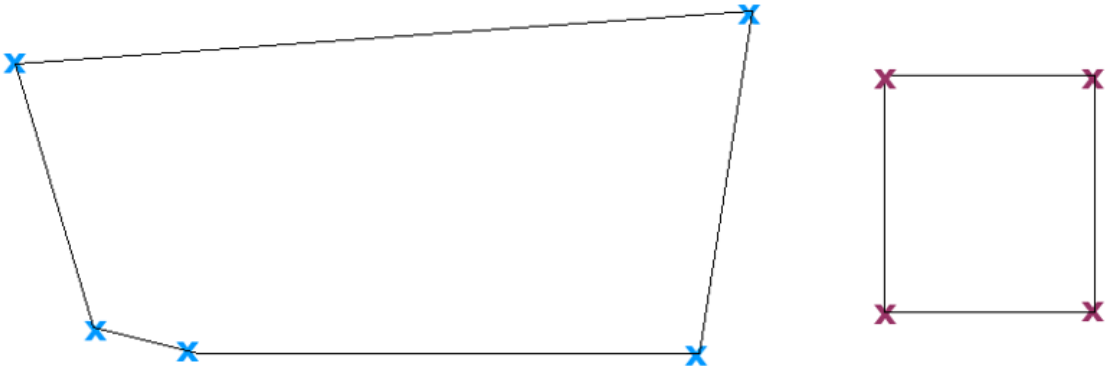


Figura 5: Elemento polígono

Un polígono, al igual que una polilínea, es una secuencia de vértices. Sin embargo, en un polígono el primer y último vértices se localizan siempre en la misma posición (Figura 5).

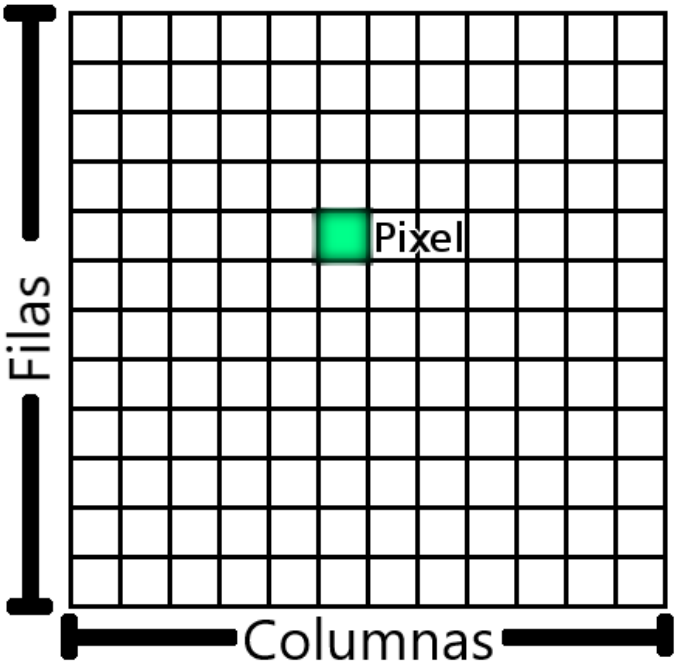
Volviendo a la imagen anterior de la vista aérea de una zona de Almería, podremos identificar diferentes tipos de objetos tal como lo haría un SIG.

Figura 6: Zona de Almería con objetos identificados

En este caso (Figura 6), hemos marcado los elementos de tipo punto, que serían los árboles (azul) y farolas (rojo); los de tipo polilínea: carreteras (verde) y caminos (naranja) y, por último, los elementos de tipo polígono: casas (amarillo) y piscinas (azul).

1.1.2. Datos ráster

En el apartado anterior hemos hablado de los datos vectoriales. Mientras que los datos vectoriales son del tipo punto, línea, polilínea, polígono y los utilizamos para representar objetos que podemos ver, los datos ráster tienen un enfoque distinto. Estos datos se representan con una matriz de píxeles y cada píxel tiene un valor que corresponde a la zona cubierta por ese píxel (Figura 7).



[4]

Figura 7: Conjunto de datos ráster

Un conjunto de datos ráster está compuesto por filas y columnas, y cada par fila-columna representa un píxel (celda). Cada píxel representa una región geográfica, y el valor que toma ese píxel coincide con una característica de la región que ocupa.

Los datos ráster son utilizados para mostrar información que se mantiene en toda la extensión de un área que no es fácilmente divisible en elementos de datos vectoriales.

[4] Datos Raster. (2019). Disponible en https://docs.qgis.org/2.8/es/docs/gentle_gis_introduction/raster_data.html

Como hemos visto antes, con la imagen área de una zona de Almería, hay objetos que diferenciamos rápidamente y les asignamos datos vectoriales de manera rápida; pero si volvemos a la Figura 2, observamos la esquina inferior derecha, el descampado, y quisiésemos diferenciar arbustos dentro de ese descampado, es complicado asignarle información georreferenciada. Podríamos hacer un polígono que englobe todo el descampado, pero podríamos perder información simplificando tanto porque, al crear una entidad geográfica, a toda ella se le asignan los mismos valores. El hecho de crear un único polígono es equivalente a que todo el descampado estuviese cubierto por igual con los mismos arbustos. Por ello existen los datos ráster. Toda la imagen se dividiría en una cuadrícula, cada celda tomaría un valor y de esta manera es mucho más sencillo diferenciar los arbustos que se encuentran dentro de ese descampado.

Los datos ráster se utilizan también como una capa inferior, detrás de capas vectoriales, para tener más información, del mismo modo que se utilizaría una imagen normal para añadir significado a la información, ya que el ser humano entiende mucho mejor si ve algo que reconoce. Si volvemos a la Figura 6, se utiliza una imagen para que sea fácilmente reconocible la información vectorial porque líneas, polígonos y puntos por sí solos es algo más abstracto y costaría más el hecho reconocer su significado.

Las imágenes ráster[5] son útiles porque proporcionan una gran cantidad de detalles e información que no podemos representar con datos vectoriales, como se puede comprobar en la Figura 8, una imagen de temperaturas a nivel mundial.

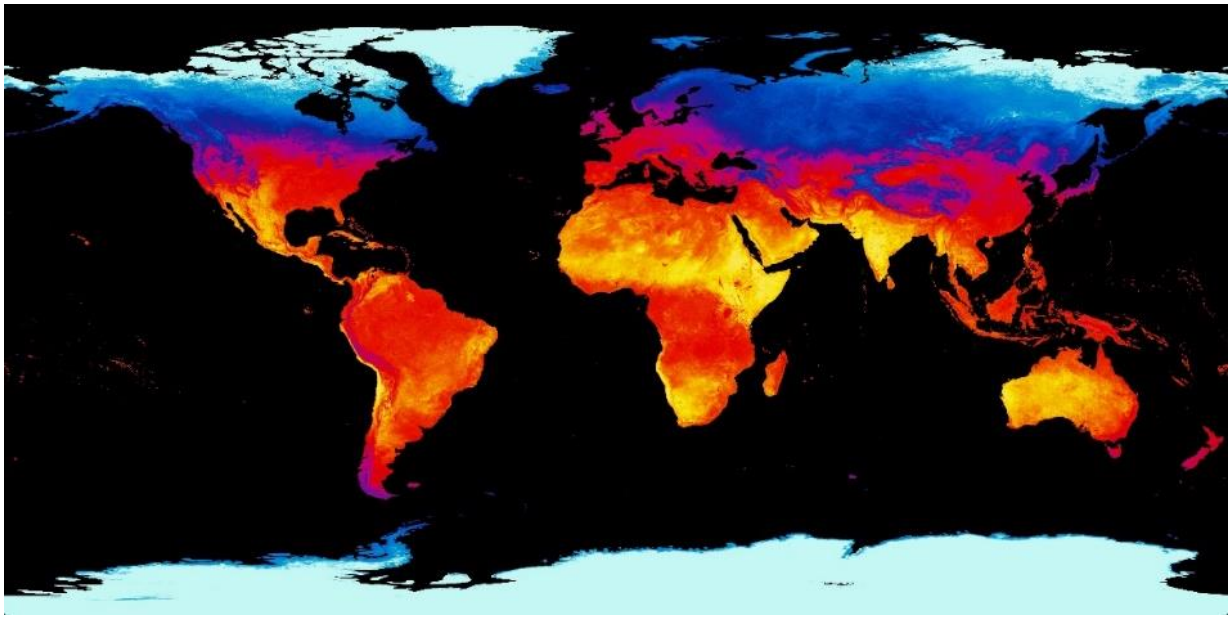


Figura 8: Imagen ráster

[5] Mapa mundial temperatura (2019). Disponible en https://rapidainformacion.files.wordpress.com/2015/03/mapa-mundial-temperatura.jpg

Figura 9: Datos ráster

Los datos ráster también pueden ser no fotográficos, como la capa ráster de la Figura 9[6], que muestra la temperatura mínima promedio en el Cabo Occidental para el mes de marzo de 2009.

1.2. Georreferenciación y resolución espacial

En este subapartado se explicará la georreferenciación y la resolución espacial.

- **Georreferenciación**

La georreferenciación es el proceso de definir el lugar exacto de la superficie terrestre en el que se encuentra un objeto. Esta información se suele almacenar con una foto aérea. Cuando la aplicación GIS abre la fotografía, utiliza la información de posición guardada para poner el lugar exacto de los datos geográficos en la imagen. Normalmente esta información consta de unas coordenadas de los píxeles, X e Y.

- **Resolución Espacial**

Como se ha explicado anteriormente, las capas ráster tienen píxeles. Estos son de un tamaño fijo para toda la capa y determinan la resolución espacial de dicha capa. La resolución espacial la comenzamos a notar al trabajar con una parte pequeña de una imagen. Si nos fijamos en la Figura 8, un mapa mundial de temperaturas, no vemos bordes pixelados y parecen bien definidos, pero si queremos trabajar con una pequeña zona pongamos, por ejemplo, parte de Andalucía y Murcia, la resolución ya pierde calidad, pero sigue siendo buena, por lo que esta imagen en concreto tendría una buena resolución espacial. Los bordes no se pixelan hasta hacerlo prácticamente irreconocible (Figura 10).

[6] Datos Raster. (2019). Disponible en https://docs.qgis.org/2.8/es/docs/gentle_gis_introduction/raster_data.html

Figura 10: Mapa térmico Andalucía y Murcia

Cuanta menos área cubra un píxel, más resolución tendrá esa imagen y más detalles se podrán diferenciar. Por el contrario, si un píxel cubre un área grande se perderán detalles y, aunque con una zona grande parezca que identificamos todo, al quedarnos con una porción de esa imagen no podremos distinguir la mayoría de los detalles.

En este punto ya conocemos los tipos de datos geográficos existentes, pero no cómo podemos almacenarlos, ya que su almacenamiento y gestión es distinta a un campo de texto o un entero. En el siguiente punto se hará referencia a la base de datos y sus particularidades a la hora de guardar información geográfica.

1.3. Introducción a PostGIS[7]

PostGis[8] es una extensión de la base de datos PostgreSQL, que tiene la particularidad de ser capaz de manejar objetos georreferenciados e incluir algunas funciones básicas para su análisis.

Los objetos GIS soportados por PostGIS son los definidos por OpenGIS. Actualmente PostGIS contiene muchas de las características de OpenGIS, aunque no contempla operadores de comparación y convolución que sí contiene Open GIS.

Ejemplos de la representación en modo texto:

POINT(0 0 0)

LINESTRING(0 0, 1 1, 1 2)

POLYGON(0 0 0, 4 0 0, 4 4 0, 0 4 0, 0 0 0)

[7] PostGIS manual. (2019). Disponible en http://www.dcc.fc.up.pt/~michel/TABD/postgis.pdf

[8]Developers, Postgis–Spatial and Geographic Objects PostgreSQL. (2019) Disponible en https://postgis.net/

MULTIPOINT(0 0 0,1 2 1)

MULTILINESTRING((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))

MULTIPOLYGON(((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0)),((

-1 -1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1 0)))

GEOMETRYCOLLECTION(POINT(2 3 9),LINESTRING((2 3 4,3 4 5))

Las entidades que permite crear la extensión PostGIS son entidades con coordenadas en 2 o 3 dimensiones. Además, se puede forzar su conversión a 2 o 3 dimensiones, dependiendo la necesidad del momento, con las funciones force_2d() y force_3d().

En este proyecto únicamente usaremos POINT, LINESTRING y POLYGON para crear capas de puntos, capas de polilíneas y capas de polígonos. No se usarán puntos en 3 dimensiones.

No podemos crear cualquier tipo de dato en cualquier campo. Cada entidad ha de insertarse en un campo que soporte sus características, un polígono sólo podrá ser añadido si en esa tabla existe un campo de polígonos, por lo que hay que añadir columnas geométricas del tipo que se necesite.

Función genérica para crear una columna genérica:

AddGeometryColumn(<nombre_bd>,<nombre_tabla>,<nombre_columna>,<srid>,<tipo>,<dimensión>)

Funciones para crear puntos, polilíneas y polígonos:

AddGeometryColumn(<nombre_bd>,<nombre_tabla>**, 'geom', 4326 , 'POINT', 2);**

AddGeometryColumn(<nombre_bd>,<nombre_tabla>**, 'geom_l', 4326 , 'LINESTRING', 2);**

AddGeometryColumn(<nombre_bd>,<nombre_tabla>**,geom_p, 4326, 'POLYGON', 2);**

En la Figura 11, presentada a continuación, hay un ejemplo de un campo polígono que se ha creado mediante la función vista anteriormente y se ha inicializado con 3 puntos.

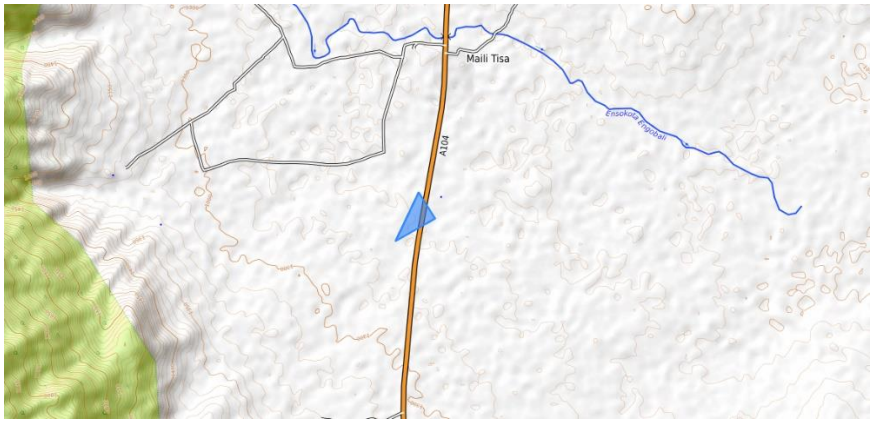


Figura 11 Capa georreferenciada con un polígono

Explicados los tipos de datos geográficos y cómo almacenarlos en una base de datos, es el turno de mostrarlos en un mapa. Para esta aplicación web, se ha utilizado OpenLayers.

1.4. Introducción a OpenLayers

Openlayers[9] es una librería de JavaScript que nos permite añadir un componente de tipo mapa a la página web que estemos diseñando. El visor de mapas de OpenLayers es compatible con los formatos de información geográfica de Google y Microsoft.

En Openlayers se debe tener en cuenta el concepto visto anteriormente de capas georreferenciadas, ya que podemos elegir tener una capa de fondo para dar más información a los datos que queramos mostrar y, además, añadir tantas capas como se necesiten.

Al ser una librería de Javascript, no es necesario instalar nada, únicamente tenemos que indicar las referencias al script de dónde consultar ciertos métodos y tendremos un mapa en nuestra web con infinidad de acciones predefinidas, haciendo de uso ellas en simples scripts. También ha ayudado a su elección el hecho de que sea un mapa dinámico, no tiene una posición fija y se le pueden añadir capas, marcas y controles, todo a la elección y necesidades de quien lo usa.

Para todos los métodos usados, la información la obtiene de un servidor remoto, por lo que es imprescindible tener conexión a internet para utilizarlo.

En la Figura 12 se muestra el mapa por defecto de OpenLayers.


[10]

Figura 12: Ejemplo mapa de OpenLayers.

Llegado este punto, ya se han explicado los tipos de datos geográficos que existen, cómo podemos almacenar esos datos en una base de datos y la manera de mostrarlos en una web.

[9] Openlayers.org. (2019). *OpenLayers - Welcome*. Disponible en https://openlayers.org
[10] Openlayers.org. (2019). Select Features. Disponible en https://openlayers.org/en/latest/examples/select-features.html

Ahora toca hablar sobre el funcionamiento de la aplicación GIS diseñada y la necesidad de esta.

1.5. Introducción a GeoServer[11]

GeoServer es un servidor de código abierto implementado en Java, el cual permite a sus usuarios subir, compartir y editar fatos geográficos. El principal punto fuerte de Geoserver es la interoperabilidad. Permite trabajar con prácticamente todos los formatos y datos generados en software de propietarios.

Geoserver contiene las especificaciones del Open Geospatial Consortium para Web Feature Service (WFS[12]) y el Web Coverage Service (WCS) entre otras y cuenta con un Web Map Service (WMS) integrado.

- **Web Feature Service (WFS):** permite descargar de datos geográficos exportándolos con formato shape o GML.
- **Web Coverage Service (WCS):** permite acceder y descargar la información de datos de tipo ráster.
- **Web Map Service (WMS):** permite ver las capas de datos geográficos en mapas de forma dinámica y obtener información básica sobre ellos.
- **Web Processing Service (WPS):** permite publicar procesos geográficos en la Web. Esto incluye algoritmos, cálculos o modelos que operen sobre datos georreferenciados. tanto en formato ráster como vectorial.

Geoserver permite trabajar tanto con conexión a internet como sin ella. Acciones como publicar una capa, es necesario tener conexión a internet, pero para añadir una capa al workspace no es necesario. Dispone de una interfaz web, Figura 13.

[11] GeoServer. (2019). Disponible en http://geoserver.org/

[12]López, B. (2019). Los GeoServicios y cómo administrarlos en GeoServer. Disponible en https://www.cursosgis.com/que-son-los-geoservicios-y-como-administrarlos-en-geoserver-opengeo-suite/

Figura 13: Pantalla inicial de GeoServer

Esta interfaz web permite trabajar y configurar fácilmente Geoserver desde el menú que dispone a su izquierda.

- En el menú estado del servidor y acerca de se encuentran los detalles técnicos sobre GeoServer.
- En el menú de datos encontramos la previsualización de capas, los almacenes de datos y espacios de trabajo (Figura 14), en este menú se configuran las fuentes de datos y el estilo.
- En el menú servicios se edita la configuración a nivel de servicios.
- En el menú settings se edita la configuración a nivel de servidor.
- En el menú cacheado de Teselas se edita la configuración de la memoria caché.
- En el menú de seguridad se edita la configuración de los controles de acceso (autenticación y autorización).
- En el menú demos encontramos ejemplos de GeoServer.
- En el menú herramientas se permite el acceso a herramientas administrativas.

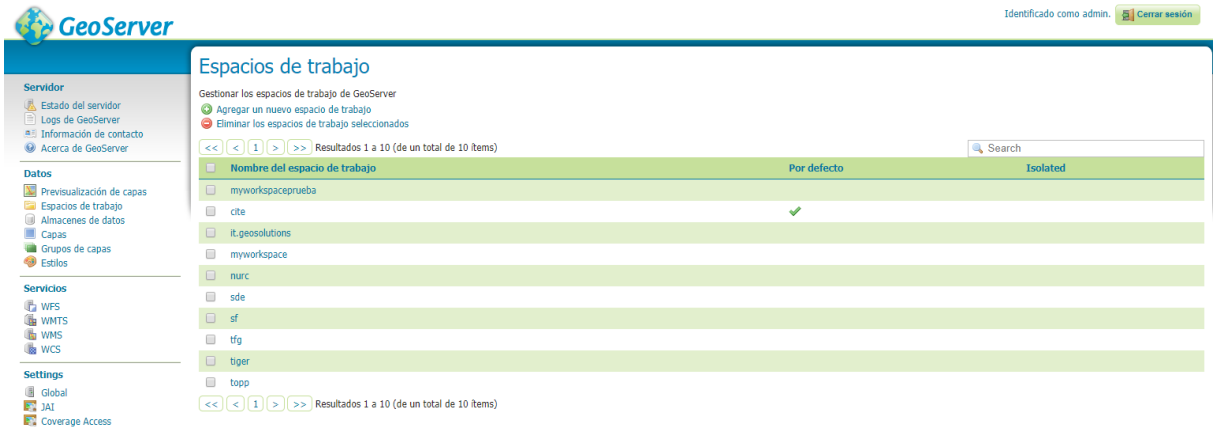


Figura 14: Espacios de trabajo en Geoserver

Llegado este punto, ya se han explicado los tipos de datos geográficos que existen, cómo podemos almacenar esos datos en una base de datos, la manera de mostrarlos en una web y las utilidades de subir una capa a GeoServer. Ahora toca hablar sobre el funcionamiento de la aplicación GIS diseñada y la necesidad de esta.

La característica principal de esta aplicación es la posibilidad de automatizar la creación de nuevas capas de información georreferenciada en una base de datos, así como todas sus funciones para que la información de la nueva capa coincida con los datos de la tabla que no tenía información geográfica, ya sea a partir de unos campos existentes y transformarlos en campos geográficos o introduciendo unas coordenadas para inicializar esa capa.

El desarrollo de sistema de información georreferenciada se debe al interés de facilitar y agilizar la creación de campos georreferenciados en bases de datos antiguas que no disponen de ese campo para, así, asociar información con la que se contaba anteriormente con la geográfica y de esta manera su localización en un mapa sea inmediata.

Para este proyecto se han utilizado los datos georreferenciados, En este caso, esos datos han sido del tipo punto, vector y polígono. Para ello se ha practicado y probado con capas ya publicadas y utilizando el programa QGIS, en el que se han editado y creado capas de los tres tipos utilizados.

1.6. Necesidad de una aplicación SIG que añada capas

La razón principal que motiva el desarrollo de esta aplicación es que, actualmente, una parte importante de la información que se maneja está georreferenciada, con información asociada sobre su posicionamiento geográfico. Este tipo de información mejora la gestión e identificación de recursos.

Partiendo de la base de que cada vez más información se desea georreferenciar, aparece la tesitura en la que, teniendo una base de datos sin ninguna información georreferenciada, se le quiera añadir dicha información.

El proceso para llevar a cabo la incorporación de esta información no es sencillo ni está automatizado y deben escribirse diversos scripts, como añadir la extensión PostGIS a la base de datos, además de crear el tipo de columna, añadir los datos, etc.

En la actualidad, la única manera que tenemos de lograr este proceso es hacerlo paso a paso, ejecutando scripts desde un administrador de bases de datos, y no todo el mundo está familiarizado con estas operaciones de administración y edición, aún con más impacto en el caso concreto de utilizar información georreferenciada. Este proyecto intentará agilizar el proceso para que todo el que lo dese pueda añadir información georreferenciada a su base de datos, ya sea a partir de datos que ya existiesen o inicializando la nueva capa.

Así, con este trabajo de fin de grado se pretende automatizar la construcción de capas de un GIS a partir de bases de datos con información no georreferenciada.

La característica principal de esta aplicación es la posibilidad de automatizar la creación de nuevas capas de información georreferenciada en una base de datos, así como todas sus funciones para que la información de la nueva capa coincida con los datos de la tabla que no tenía información geográfica.

El desarrollo de este sistema de información georreferenciada se debe al interés de facilitar y agilizar la creación de campos georreferenciados en bases de datos antiguas que no disponen de ese campo para, así, asociar información con la que se contaba anteriormente con la nueva información geográfica y, de esta manera, su localización en un mapa sea más sencilla.

Con la intención de poner en uso los conocimientos vistos anteriormente y poder explicar la funcionalidad que podría tener esta aplicación, veamos un ejemplo.

Supongamos que tenemos una aplicación en la que están todos los puntos de carga para vehículos eléctricos en Almería y obtenemos una información de una base de datos que tiene la información guardada tal y como se muestra en la Tabla 1, con un campo para la latitud, otro para la longitud, otro para el lugar y un último para la ciudad.

Latitud	Longitud	Lugar	Ciudad
36.831321	-2.402425	Universidad	Almería
36.838406	-2.323746	Hotel	Almería
36.837086	-2.461902	Aparcamiento	Almería
36.841658	-2.444333	Supermercado	Almería

Latitud	Longitud	Lugar	Ciudad
36.852901	-2.447631	Centro comercial	Almería
36.861695	-2.435328	Centro comercial	Almería
36.861564	-2.435121	Centro comercial	Almería
36.861687	-2.435017	Centro comercial	Almería

Tabla 1: Ejemplo información en base de datos.

Fijándonos en la Tabla 1, tenemos los dos campos de las coordenadas, que no se mostrarían de manera automática en un mapa. Habría que transformar esos dos campos en un punto georreferenciado para, de esta manera, poder visualizarlo en un mapa como una capa con información geográfica.

Sabiendo los tipos de datos que existen, y teniendo únicamente la latitud y la longitud, se debe crear un dato vectorial de tipo punto, como el que se puede ver a través de los puntos mostrados en la Figura 15 (los 8 puntos azules, varios de ellos superpuestos en la zona superior izquierda de la imagen).

Figura 15: Mapa de Almería con los puntos marcados

Con el simple hecho de usar esta aplicación y partiendo de la Tabla 1 que anteriormente no tenía un campo geográfico, hemos creado otra tabla con un campo georreferenciado de tipo punto y ya podemos ver esos datos en un mapa (Figura 15). De esta forma, es posible localizar de un vistazo y de manera sencilla dónde cargar un vehículo eléctrico en la provincia de Almería.

Capítulo 2

Análisis

2. Análisis

A continuación, se procederá a revisar los distintos aspectos del proyecto a realizar mediante un análisis DAFO, se plantearán las distintas fases del desarrollo y se estimarán los plazos en los que se realizarán cada una de las fases.

2.1. Análisis DAFO

Antes de comenzar con el desarrollo de la solución, conviene analizar en qué punto se encuentra nuestra organización. Saber cuáles son nuestras fortalezas para potenciarlas, nuestras debilidades para corregirlas, aprovechar las oportunidades y controlar las amenazas. Esta será la primera parte de la planificación estratégica.

- FORTALEZAS
  - Software con posibilidad de ampliación.
  - Automatización de trabajo tedioso.

- DEBILIDADES
  - No interactúa con cualquier tipo de base de datos.
  - Únicamente disponible con acceso a internet.

- OPORTUNIDADES
  - No hay software que tenga esta funcionalidad.

- AMENAZAS
  - Empresas privadas que realicen esta automatización antes que nosotros.

2.2. Fases de desarrollo

Se puede dividir el trabajo realizado en 4 fases de desarrollo, las cuales son:

1. Análisis de requerimientos.

En esta fase se hace un estudio de la situación actual, detectando los errores e inconvenientes del problema y cuáles son los objetivos que se desean alcanzar. Las etapas de esta fase son:

- **Descripción de la situación actual.**
  Se analiza el modo de añadir capas georreferenciadas a una base de datos ya existente, las distintas extensiones necesarias y las bases de datos que soportan dichas extensiones.
  Recordemos que todo el trabajo es de manera manual, creando tablas, *triggers* y campos georreferenciados mediante comandos.
- **Estudio de mercado.**
  Analizar todo software capaz de crear una capa con información georreferenciada.

- **Definición del problema a resolver.**
  Una vez analizada la situación, hay que detectar los principales problemas de la aplicación a realizar, como la automatización de todos los scripts necesarios y las conexiones a distintas bases de datos.
- **Definición de los objetivos.**
  Los objetivos están orientados principalmente a la resolución de los problemas detectados en el análisis de la situación actual.
- **Identificación de los componentes principales que integra la solución.**
  - Hardware
    - Equipo de sobremesa.
    - Servidor web
  - Software
    - PostgreSQL
    - GeoServer
    - OpenLayers
    - Apache Tomcat
    - Java
  - Tipo de aplicación
    - Aplicación web.

2. Especificación.

Esta fase consiste en describir el comportamiento esperado del software y su interacción con el usuario. El objetivo principal es implementar un sistema informático/software en el que un usuario que tenga una base de datos sin campos georreferenciados pueda añadirle a la tabla deseada de su base de datos un campo georreferenciado, ya sea de puntos, vectores o polígonos. Una vez realizados todos los pasos propuestos en la aplicación web, el usuario podrá ver en un mapa su capa georreferenciada y consultar qué entrada de la tabla corresponde a qué información del mapa. Para realizar todo este proceso de manera satisfactoria, es necesario que el sistema permita las siguientes funcionalidades:

- Conectarse a una BD remota o local.
- Elegir una tabla de la BD con la que hay una conexión.
- Consultar los campos y datos de la tabla elegida.
- Crear tablas nuevas.
- Crear *triggers* para mantener en consistencia entre la nueva tabla y la vieja.
- Crear campos de información geográfica.
- Transformar datos que no son geográficos en datos geográficos.
- Mostrar los datos en un mapa.
- Identificar los datos creados con la información del mapa.
- Subir una capa con información georreferenciada a GeoServer.

3. Diseño y arquitectura.

Una vez se saben las funcionalidades de la aplicación a desarrollar, se debe realizar el diseño de esta, tanto de la interfaz de usuario como el diseño del funcionamiento que proporcione respuestas a las funcionalidades descritas en la segunda parte de la fase.

1. Elección de lenguajes.

2. Elección de base de datos predeterminada para guardar información georreferenciada.

3. Interacciones usuario-sistema.

4. Definición estilo interfaz.

5. Elección de la metodología de desarrollo.

4. Desarrollo.

En esta fase, se desarrollará la aplicación según los criterios de diseño y de arquitectura vistos en el apartado anterior.

Al ser una aplicación web en la que no es necesario mantener datos de una ejecución a otra, no tendremos que diseñar una base de datos para la gestión de la propia información de la aplicación, pero si se desarrollarán los siguientes puntos:

- Elección de tecnologías y lenguajes.
- Desarrollo de *frontend*.
- Desarrollo de *backend*.
- Diseño de la interfaz de usuario.
- Desarrollo de la interfaz de usuario.

Es necesario elegir las tecnologías adecuadas que nos permitan cumplir todos los requisitos de nuestro proyecto.

2.3. Estimación de plazos

Para este proyecto, se ha llevado a cabo una metodología ágil, dividiendo el trabajo en *sprints* de un mes de duración para, de esta manera, poder identificar lo más rápido posible los errores de comunicación y del proyecto y, en consecuencia, los tiempos de modificación sean los menores posibles.

En la Tabla 2 se presenta el cronograma seguido para el desarrollo del proyecto.

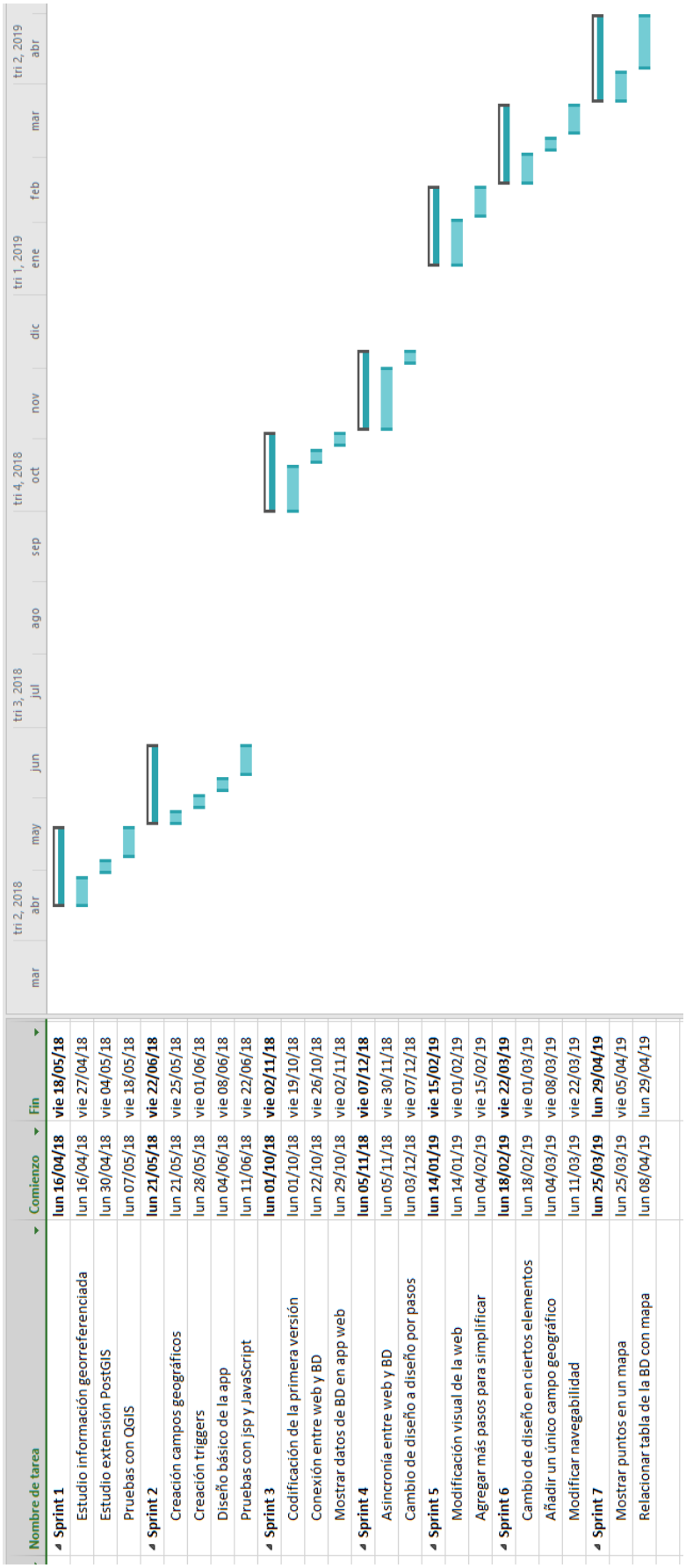
	Duración	**Inicio**	**Fin**
Sprint 1	**1 mes**	**16/04/2018**	**18/05/2018**
• Estudio información georreferenciada	2 semanas	16/04/18	27/04/18

• Estudio y pruebas extensión PostGIS • Pruebas con QGIS	1 semana 2 semanas	30/04/18 07/05/18	04/05/18 18/05/18
Sprint 2	**1 mes**	**21/05/2018**	**22/06/2018**
• Diseño básico de la aplicación • Pruebas con jsp y JavaScript • Creación campos geográficos • Creación triggers	1 semana 2 semana 1 semana 1 semanas	21/05/18 28/05/18 04/06/18 11/06/18	25/05/18 01/06/18 08/06/18 22/06/18
Sprint 3	**1 mes**	**1/10/2018**	**2/11/2018**
• Codificación primera versión • Conexión entre web y BD • Mostrar datos BD en aplicación	3 semanas 1 semanas 1 semanas	1/10/18 22/10/18 29/10/18	19/10/18 26/10/18 02/11/18
Sprint 4	**1 mes**	**5/11/2019**	**7/12/2018**
• Asincronía entre web y BD • Cambio a diseño por pasos	4 semanas 1 semana	5/11/18 3/12/18	30/11/18 07/12/18
Sprint 5	**1 mes**	**14/01/2019**	**15/02/2019**
• Modificación visual elementos web • Agregar pasos para simplificar	3semanas 2 semanas	14/01/19 4/2/19	1/2/19 15/2/19
Sprint 6	**1 mes**	**18/02/2019**	**22/03/2019**
• Cambio de diseño en elementos • Añadir un único campo geográfico • Modificar navegabilidad	2 semana 1 semana 2 semanas	18/2/19 4/3/19 11/3/19	1/3/19 8/3/19 22/3/19
Sprint 7	**1 mes**	**25/03/2019**	**29/04/2019**
• Mostrar puntos en un mapa • Relacionar tabla con mapa	2 semanas 3 semanas	25/3/19 8/4/19	5/4/19 29/4/19

Tabla 2: Cronograma con los diferentes sprints del proyecto

En total, el proyecto se ha desarrollado a lo largo de 7 meses trabajados en los que, de media, el trabajo se ha realizado durante 2 horas diarias. Por lo tanto, se han dedicado las 300 horas que se deben emplear en este proyecto.

La información sobre el cronograma está disponible también en un diagrama de Gantt, presentado en la Figura 16.



Nombre de tarea	Comienzo	Fin
Sprint 1	**lun 16/04/18**	**vie 18/05/18**
Estudio información georreferenciada	lun 16/04/18	vie 27/04/18
Estudio extensión PostGIS	lun 30/04/18	vie 04/05/18
Pruebas con QGIS	lun 07/05/18	vie 18/05/18
Sprint 2	**lun 21/05/18**	**vie 22/06/18**
Creación campos geográficos	lun 21/05/18	vie 25/05/18
Creación triggers	lun 28/05/18	vie 01/06/18
Diseño básico de la app	lun 04/06/18	vie 08/06/18
Pruebas con jsp y JavaScript	lun 11/06/18	vie 22/06/18
Sprint 3	**lun 01/10/18**	**vie 02/11/18**
Codificación de la primera versión	lun 01/10/18	vie 19/10/18
Conexión entre web y BD	lun 22/10/18	vie 26/10/18
Mostrar datos de BD en app web	lun 29/10/18	vie 02/11/18
Sprint 4	**lun 05/11/18**	**vie 07/12/18**
Asincronía entre web y BD	lun 05/11/18	vie 30/11/18
Cambio de diseño a diseño por pasos	lun 03/12/18	vie 07/12/18
Sprint 5	**lun 14/01/19**	**vie 15/02/19**
Modificación visual de la web	lun 14/01/19	vie 01/02/19
Agregar más pasos para simplificar	lun 04/02/19	vie 15/02/19
Sprint 6	**lun 18/02/19**	**vie 22/03/19**
Cambio de diseño en ciertos elementos	lun 18/02/19	vie 01/03/19
Añadir un único campo geográfico	lun 04/03/19	vie 08/03/19
Modificar navegabilidad	lun 11/03/19	vie 22/03/19
Sprint 7	**lun 25/03/19**	**lun 29/04/19**
Mostrar puntos en un mapa	lun 25/03/19	vie 05/04/19
Relacionar tabla de la BD con mapa	lun 08/04/19	lun 29/04/19

Figura 16: Diagrama de Gantt del proyecto

Capítulo 3

Especificación y diseño de la herramienta para la generación de capas a partir de información no georreferenciada

3. Especificación y diseño del sistema

En esta sección se describirán las especificaciones, tanto técnicas como funcionales del sistema a implementar, así como su diseño visual y a nivel de navegabilidad.

3.1. Especificación

Como se indicó en el punto 1.4, no existe un proceso automatizado para crear una nueva tabla en un base de datos que cree una capa con datos geográficos de manera automática. Es necesario ir paso a paso, añadiendo todos los scripts necesarios. Por ello, se propone la implementación de esta aplicación web que automatice en lo posible:

- Creación de tablas con campo geográfico.
- Transformación de datos no georreferenciados a datos que sí lo son.
- Mantener los datos de la nueva tabla en consonancia en cuanto a modificaciones de la tabla original.
- Mostrar la capa de información geográfica creada en un mapa.

3.1.1. Especificaciones técnicas

Para este proyecto, las especificaciones técnicas no son algo prioritario, Este proyecto va a tratar sobre una aplicación web que no trabaja con grandes cantidades de datos, por lo que la aplicación en sí puede estar alojada en un PC o en una máquina virtual, siempre con conexión a Internet. Esto último es una condición imprescindible, ya que las dependencias de OpenLayers no están establecidas de manera local, sino remota.

Para este proyecto serán necesarios los siguientes elementos:

- PC.
- Máquina virtual.
- Dispositivo para acceder a la web.
- Conexión a Internet.
- Base de datos PostgreSQL.
- Servidor para mostrar la aplicación

3.1.2. Especificaciones funcionales

Para que el sistema desarrollado consiga alcanzar el objetivo principal de este proyecto, es decir, implementar un sistema informático o software que automatice el proceso de añadir una capa de información geográfica a una base de datos ya existente, ha de cumplir lo siguiente:

- Conectarse a una base de datos.
- Poder extraer la tabla especificada y sus datos.

- Crear, a elección del usuario, un tipo de capa con información geográfica.
- Crear automáticamente funciones para mantener ambas tablas con la misma información.

#	Nombre	Descripción
1	Elegir la base de datos	El sistema permitirá la conexión a la base de datos elegida por el usuario
2	Usuario y contraseña	El sistema ha de permitir que el usuario escriba las credenciales para conectarse a su base de datos.
3	Conectarse a una base de datos	El sistema permitirá conectarse a una base de datos.
4	Seleccionar una tabla de la BD	El sistema permitirá conectarse a la tabla elegida.
5	Crear una nueva tabla	El sistema permitirá la creación de una nueva tabla.
6	Seleccionar tipo de capa	El sistema permitirá elegir qué tipo de capa se creará
7	Seleccionar campos de la tabla original para añadirlos a la nueva	El sistema permitirá seleccionar campos de la tabla origen para añadir a la tabla creada.
8	Crear capa geográfica	El sistema permitirá crear una capa geográfica a partir de los campos elegidos por el usuario.
9	Inicializar capa	El sistema permitirá inicializar una capa geográfica a partir de datos introducidos por el usuario.
10	Convertir datos a datos geográficos	El sistema podrá transformar datos no georreferenciados en datos que sí lo son
11	Visualizar capa	El sistema podrá mostrar una capa con información geográfica.
12	Mantener tablas en consonancia	El sistema mantendrá actualizada la nueva tabla con respecto a la tabla original.
13	Navegabilidad	El sistema ha de poder volver al resto de pantallas.

14	Consistencia	El sistema ha de poder modificar los datos introducidos por el usuario.

Tabla 3: Especificaciones funcionales y descripción

3.2. Diseño

En este subapartado se hablará sobre el diseño de la aplicación web, su navegabilidad, el almacenamiento de datos temporal y el rol del usuario en dicha aplicación.

3.2.1. Navegabilidad

En la *Tabla 4* se describe la navegabilidad de la aplicación web, el nombre de cada pantalla, las acciones disponibles y hasta qué pantalla nos llevan esas acciones, diferenciando el resultado dependiendo del camino seguido hasta llegar a dicha pantalla.

#	Nombre	Acciones	Ir a	Si Vienes de
1	Pantalla inicial	Crear capa Visualizar capa Fusionar tablas	**2.** seleccionar capas **2.** seleccionar capas	
2	Seleccionar capas	Capa de puntos Capa de vectores Capa de polígonos Atrás Menú	**3.** Seleccionar tipo BD **3.** Seleccionar tipo BD **3.** Seleccionar tipo BD **1.** Pantalla inicial **1.** Pantalla inicial	
3	Seleccionar tipo BD	Siguiente Atrás Menú	**4.** Usuario y contraseña **2.** Crear capas **1.** Pantalla inicial	
4	Usuario y contraseña	Siguiente Atrás Menú	**5.** Nombre tabla trabajo **3.** Seleccionar tipo BD **1.** Pantalla inicial	

5	Nombre tabla de trabajo	Siguiente Atrás Menú	**6.** Seleccionar puntos **7.** Introducir vector **8.** Introducir polígono **4.** Usuario y contraseña **1.** Pantalla inicial	Capa puntos Capa vectores Capa polígono
6	Seleccionar puntos	Siguiente Saltar paso Atrás Menú	**9.** Campos nueva tabla **12.** Introducir puntos **5.** Nombre tabla trabajo **1.** Pantalla inicial	
7	Introducir vector	Siguiente Atrás Menú	**9.** Campos nueva tabla **5.** Nombre tabla trabajo **1.** Pantalla inicial	
8	Introducir polígono	Siguiente Atrás Menú	**9.** Campos nueva tabla **5.** Nombre tabla trabajo **1.** Pantalla inicial	
9	Campos nueva tabla	Siguiente Menú	**10.** Tabla creada **1.** Pantalla inicial	
10	Tabla creada	Siguiente Atrás Menú	**11.** Ver capa **9.** Campos nueva tabla **1.** Pantalla inicial	
11	Ver capa	Menú	**1.** Pantalla inicial	
12	Introducir puntos	Siguiente Atrás Menú	**9.** Campos nueva tabla **6.** Seleccionar puntos **1.** Pantalla inicial	

Tabla 4: Navegabilidad aplicación web

3.2.2. Rol usuario

En esta aplicación web no existe diferenciación entre usuarios, tampoco incluye registro de sesión. Existe un único usuario, que será el encargado de introducir los datos de la base de datos sobre la que quiere trabajar, indicar si quiere crear una nueva capa de información georreferenciada, seleccionar qué tipo de capa será, etc.; dicho usuario es el que ejecuta todas las acciones.

Tareas que realiza el usuario expuestas en la Tabla 3, vista en el punto 3.1.2:

- **#1 Elegir base de datos.**
  El usuario ha de introducir la dirección de su base de datos y el sistema ha de aceptarla para conectarse a ella, ya sea local o remota.
- **#2 Usuario y contraseña.**
  El usuario ha de introducir sus credenciales y el sistema ha de utilizarlas para probar la conexión a la base de datos ya elegida y obtener los datos pertinentes.
- **#4 Seleccionar tabla de la base de datos elegida.**
  El usuario introducirá el nombre de la tabla sobre la que quiere trabajar y el sistema ha de poder recuperar todos los datos de esa tabla.
- **#6 Seleccionar tipo de capa.**
  El usuario ha de poder elegir el tipo de capa que desea crear o visualizar y el sistema ha de poder recordarlo para futuras operaciones.
- **#7 Seleccionar campos de la tabla origen.**
  El usuario ha de poder elegir los campos que quiere que se añadan a la nueva tabla.
- **#9 Inicializar capa.**
  El usuario ha de poder inicializar la capa geográfica creada, ya sea a partir de campos que existen en la base de datos, o bien de datos introducidos manualmente por el usuario.
- **#11 Visualizar capa.**
  El usuario ha de poder seleccionar un campo georreferenciado para mostrarlo en un mapa de la aplicación.

3.2.3. Diseño de almacenamiento de datos temporales

Dado que esta aplicación web no va a almacenar datos permanentemente, no es necesario crear y diseñar una base de datos para la gestión de la información propia de la aplicación. Pero para trabajar con bases de datos ya creadas se han de ir guardando, de manera temporal, parte de los datos que introduce el usuario al usar esta aplicación web. Esos datos serán eliminados al iniciar otra vez la aplicación porque al volver a usar la aplicación el mismo usuario u otro distinto, no debe quedar guardada ninguna información.

Para ello se hace uso de varios arrays temporales, de manera que todo lo que introduce el usuario es guardado para su posterior utilización, como el nombre de la base de datos y su dirección. Al hacer varias conexiones con la base de datos durante el uso de esta aplicación como, por ejemplo, para comprobar que existe esa BD, obtener los datos de la tabla elegida, crear una nueva tabla u obtener datos geográficos para mostrarlos en un mapa, sería un

problema que la aplicación pidiese nombre de la tabla y credenciales en cada una de esas conexiones.

3.2.4. Ciclo de generación y visualización de una capa

El diagrama presentado en la Figura 17 describe un ejemplo de ciclo de ejecución de la aplicación web, en concreto, del proceso de la creación de una capa con datos de tipo punto y su posterior visualización en un mapa.

El proceso de creación de una capa de tipo polilínea o polígono es similar, cambiando la selección del tipo de capa al comienzo del proceso.

1. Comienza con la selección de la creación de una capa de puntos y, a continuación, pide ingresar la dirección de la base de datos y el nombre de esta; se pide usuario y contraseña, tras lo cual comprueba si es correcto o no. Si no es correcto muestra un aviso para permitir la modificación de los datos.

2. Si la información es correcta permitirá la introducción del nombre de la tabla sobre la que se trabajará, si no es correcto lo notificará para modificar el nombre de la tabla.

3. El nombre de la tabla es correcto, por lo que se pedirá el nombre de la tabla que se va a crear. Tras introducirlo, dará la opción, para, si existen campos latitud y longitud, seleccionarlos y transformarlos en datos vectoriales de tipo punto, y en caso de que no existan esos campos, se puedan introducir unas coordenadas para inicializar la capa.

4. Una vez tenemos las coordenadas, o bien por campos de la tabla o introducidas por el usuario, se da la opción de elegir campos de la tabla origen para añadirlos a la nueva tabla. Tras esto se crea la tabla y una vez creada se podrá ver la capa creada en un mapa.

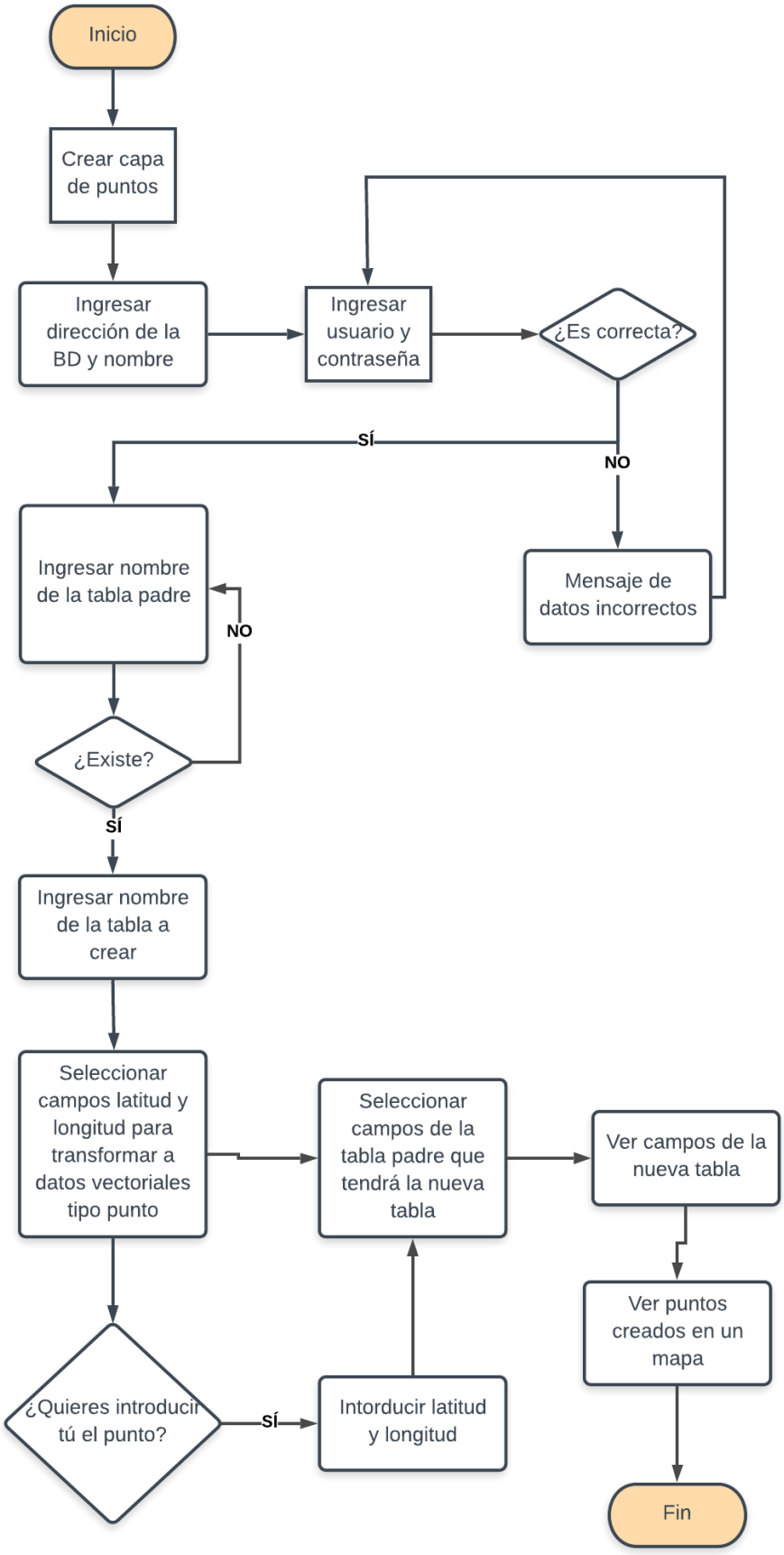


Figura 17: Comportamiento de la acción de crear y mostrar capa geográfica

3.2.4 Ciclo de visualización de una capa

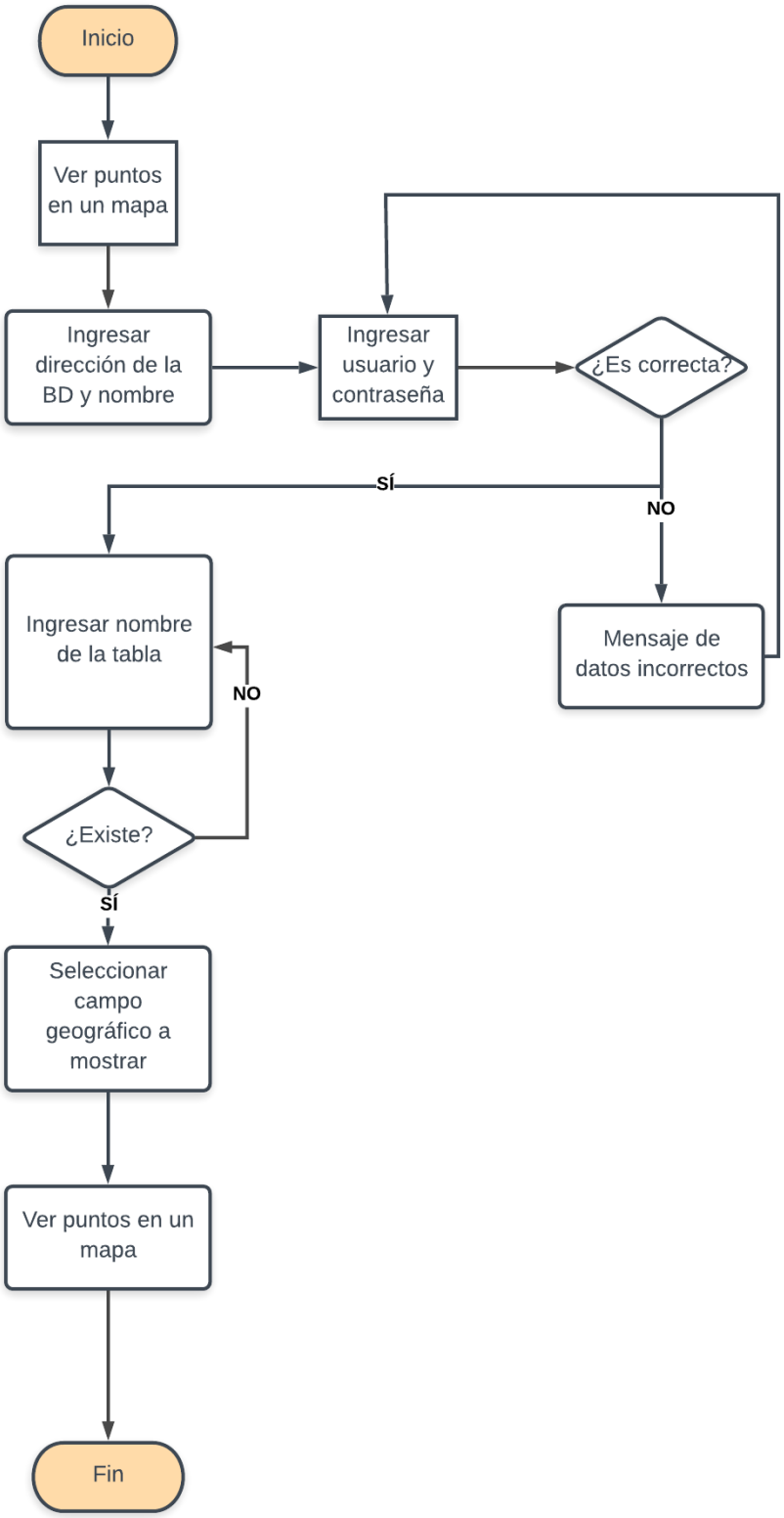


Figura 18: Comportamiento de la acción ver capa geográfica

En el diagrama presentado en la Figura 18, se describe el ciclo de ejecución de la aplicación web para mostrar una capa con información geográfica ya existente en una basa de datos.

1. Se selecciona la opción de visualizar una capa y de qué tipo de capa se trata.

2. Una vez seleccionado el tipo de capa a mostrar se introduce la dirección en la que está ubicada la base de datos y el nombre de esta. Ya introducidos estos datos, el usuario introduce las credenciales, si los datos son incorrectos se mostrará un mensaje indicándolo.

3. Teniendo los datos de la base de datos correctos, el sistema pide introducir el nombre de la tabla en la que está ubicado el campo georreferenciado que se quiere mostrar. Si el nombre de la tabla es incorrecto, se muestra un mensaje de advertencia para poder modificarlo.

4. El nombre de la tabla es correcto, por lo que el sistema mostrará todos los campos de esa tabla para que el usuario seleccione la capa georreferenciada que se quiere mostrar. Una vez seleccionada, el sistema la mostrará en un mapa, ya sea de puntos, polilíneas o polígonos.

Capítulo 4

Implementación de la solución para la generación de capas a partir de información no georreferenciada

4. Implementación de la solución para la generación de capas a partir de información no georreferenciada

Una vez aclaradas las especificaciones y el diseño de este proyecto, comienza el desarrollo de la implementación. El lenguaje de programación será Java, la herramienta de desarrollo será Eclipse[13], la base de datos con la que se trabajará será PostgreSQL[14] junto con su extensión para datos georreferenciados PostGIS[15], y para desplegar la página web, se utilizará el servidor Apache Tomcat[16].

Para la parte de la conexión con la base de datos se ha optado por hacerlo con el lenguaje de programación Java. En cambio para la funcionalidad y visualización de la aplicación web se ha optado por utilizar JavaScript y JavaServerPage. Visualmente se ha modificado el estilo con css.

Para llevar a cabo la implementación es necesario:

- Conexión a Internet
- Equipo con al menos core i5, 4GB de memoria RAM
- Eclipse como herramienta de desarrollo
- PostgreSQL como base de datos con la extensión PostGIS
- Apache Tomcat

Al ser un proyecto hecho con Java, el sistema operativo elegido para su desarrollo es indiferente, en este caso se hará sobre Windows 10.

4.1. Instalación de las herramientas necesarias

Antes de comenzar a implementar, es necesaria una correcta instalación de las herramientas que se vayan a utilizar, por lo que se verá paso a paso su instalación en este apartado. Se verá la instalación de la base de datos con la que se trabajará, PostgreSQL, la extensión que permitirá trabajar con datos georreferenciados, PostGIS, y el entorno de desarrollo, Eclipse.

Todas las instalaciones que veremos serán sobre Windows 10 con programas gratuitos. No ha sido necesario utilizar programas de pago en el proceso de implementación de este trabajo de fin de grado.

4.1.1. PgAdmin 4

En primer lugar, instalaremos la base de datos donde se desarrollarán todas las pruebas.

[13] Foundation, E. (2019). Eclipse Downloads | The Eclipse Foundation. Disponible en https://www.eclipse.org/downloads/
[14] PostgreSQL: Windows installers. (2019). Disponible en https://www.postgresql.org/download/windows/
[15] Developers, P. (2019). PostGIS — Windows Downloads. Disponible en https://postgis.net/windows_downloads/
[16] Apache Tomcat® - Apache Tomcat 8 Software Downloads. (2019). Disponible en https://tomcat.apache.org/download-80.cgi

Ejecutamos el instalador y se nos abrirá un *wizard* de instalación. Le damos al botón *next* y aceptamos las condiciones, tal y como se muestra en la Figura 19 y Figura 20.

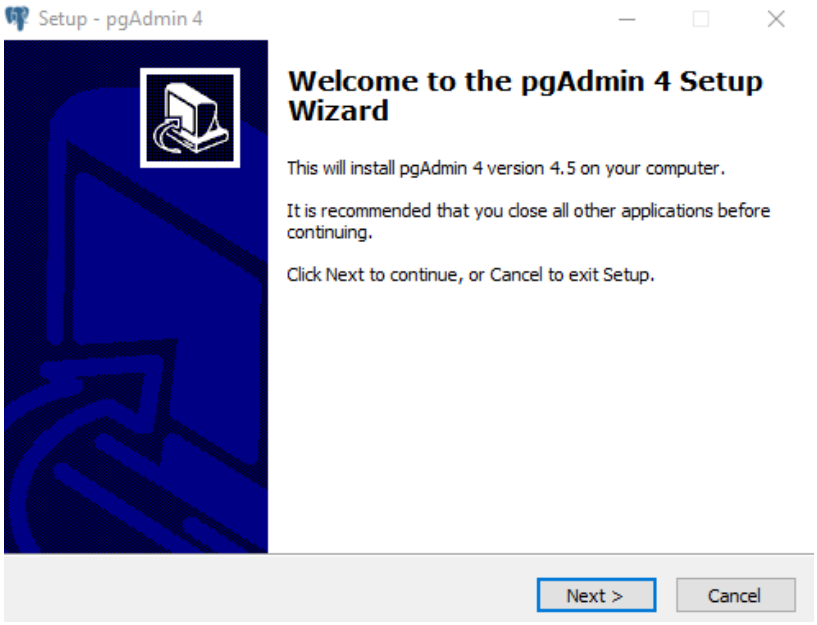


Figura 19: Paso 1 instalación PgAdmin4.

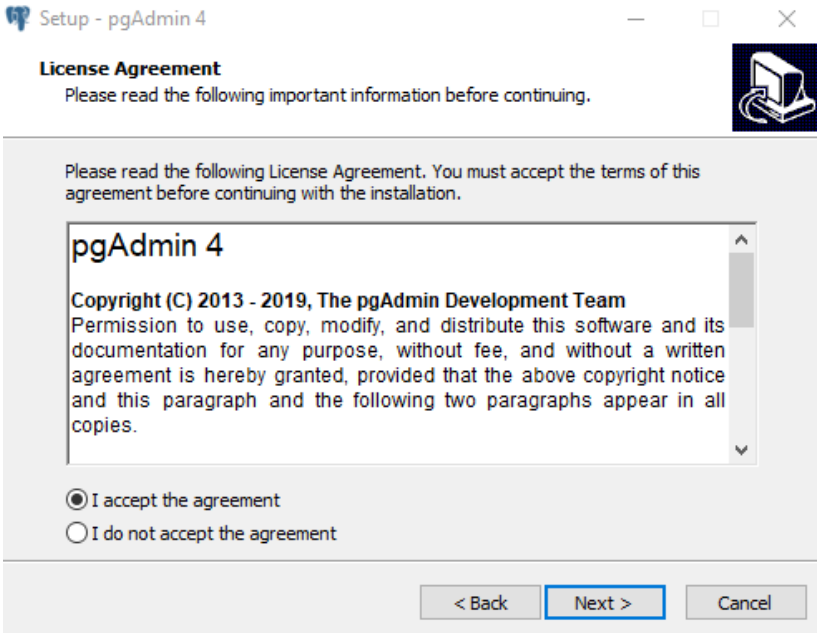


Figura 20: Paso 2 instalación PgAdmin4.

Una vez aceptados los términos y condiciones seleccionamos el directorio donde deseamos instalar el gestor de bases de datos PgAdmin 4, Figura 21.

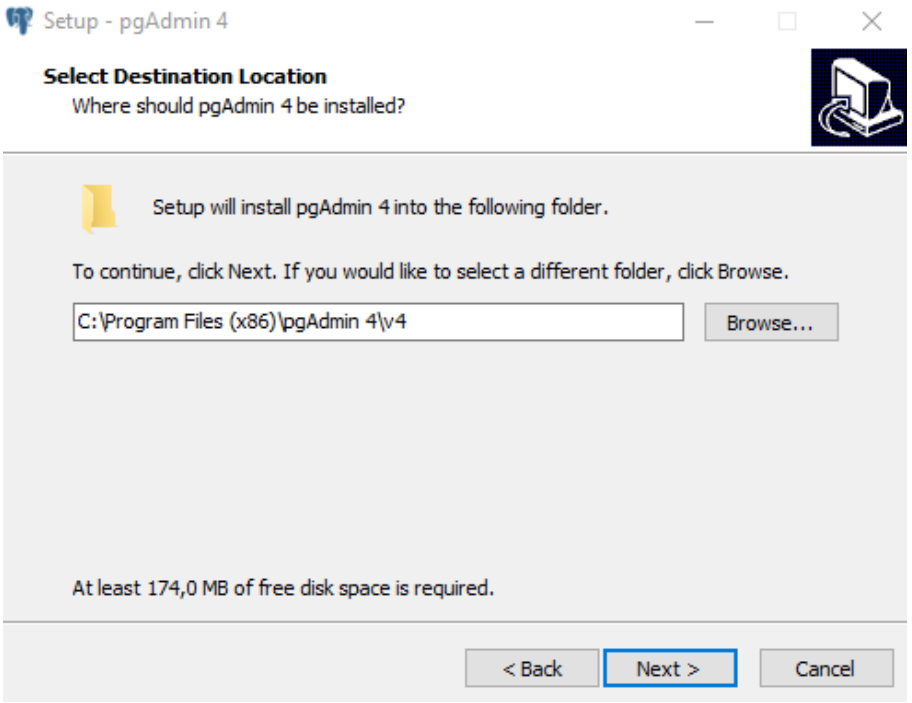


Figura 21: Paso 3 instalación PgAdmin4.

Seleccionamos, o no, crear icono en el escritorio. En este caso de ha decidido no crearlo, ya que el check no se ha marcado, Figura 22.

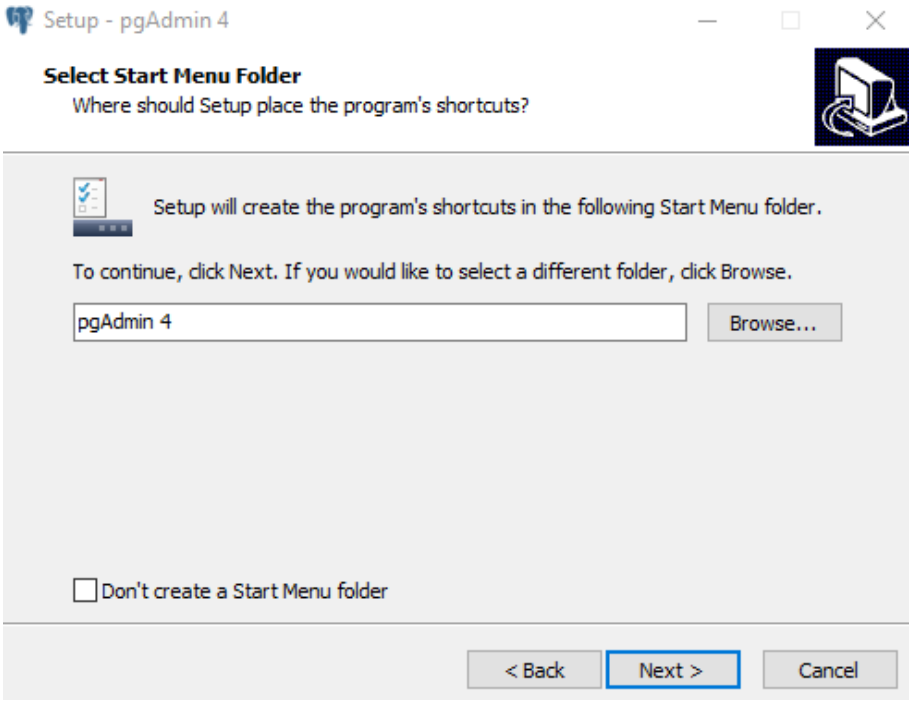


Figura 22: Paso 4 instalación PgAdmin4.

Una vez seleccionado todo correctamente y especificado el directorio le damos al botón de Install para que comience la instalación (Figura 23).

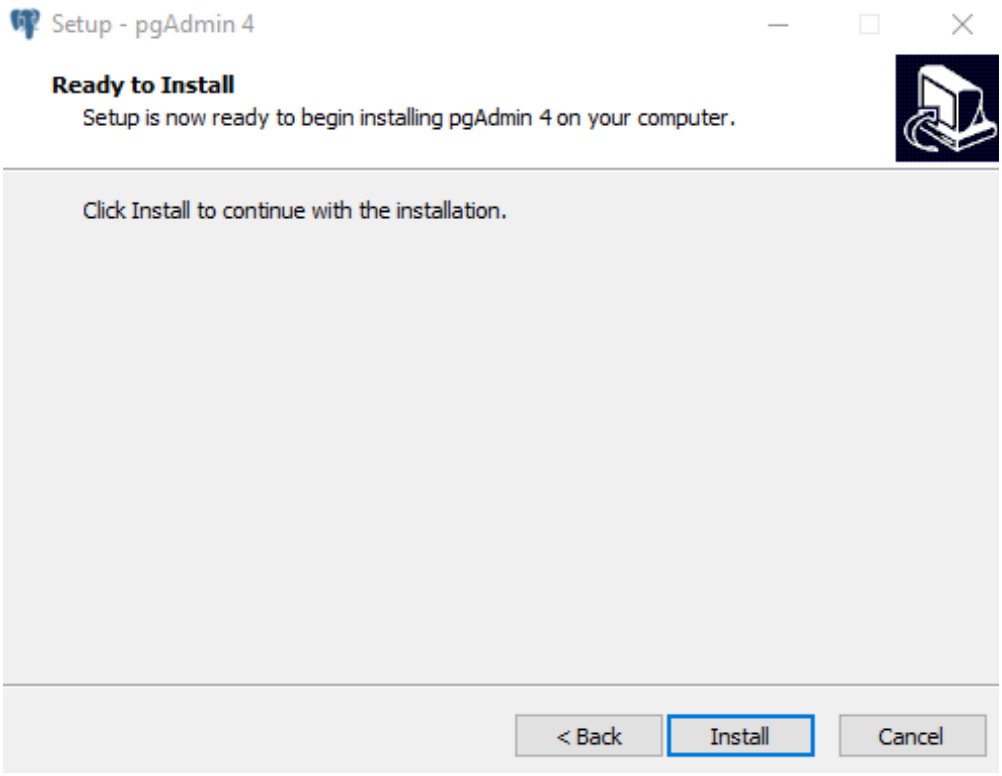


Figura 23: Paso 5 instalación PgAdmin4.

Terminada la instalación nos da la posibilidad de iniciar directamente el programa, como se muestra en la Figura 24, por lo que aceptamos para ver el resultado.

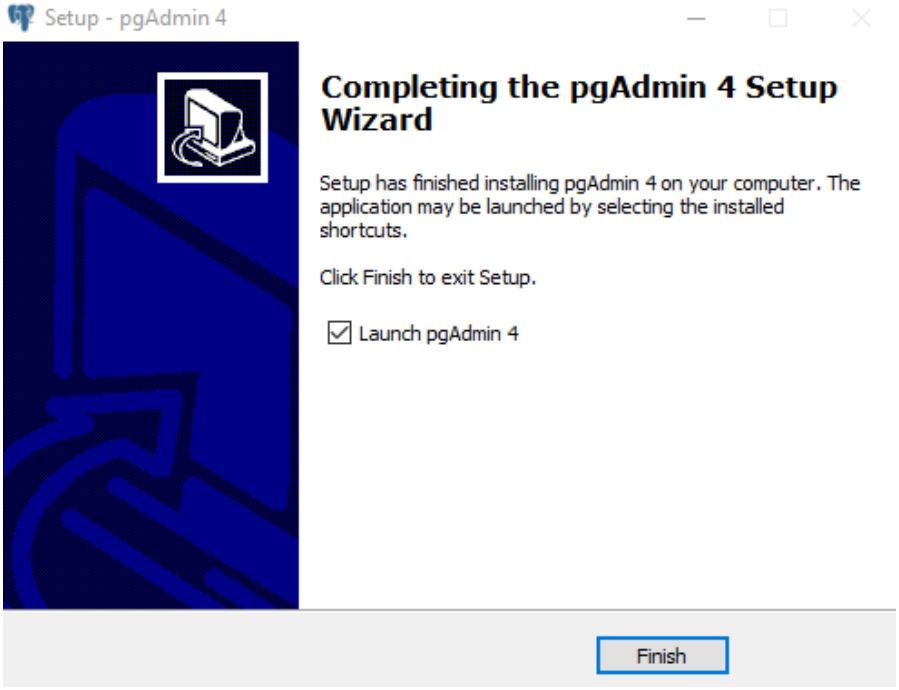


Figura 24: Paso 6 instalación PgAdmin4.

Como podemos comprobar en la Figura 25, simplemente está instalado el gestor de la base de datos PgAdmin, no tenemos ninguna base de datos ni servidores.

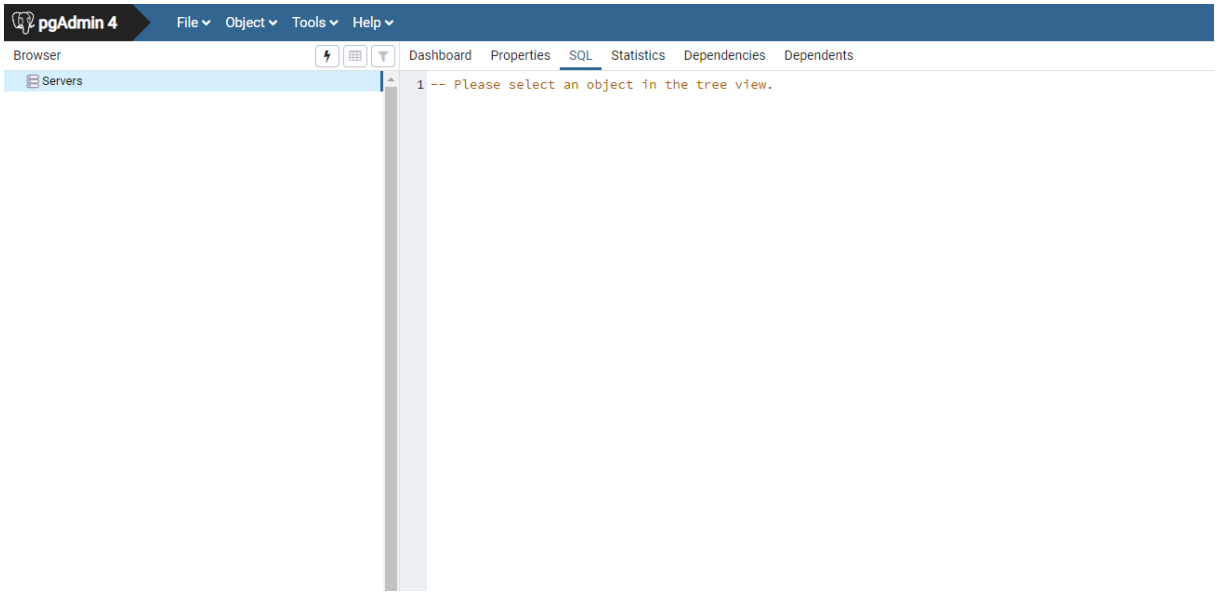


Figura 25: Sección SQL en el gestor de base de datos PgAdmin 4.

Podemos observar que es un gestor bastante completo, en el que podemos ejecutar scripts SQL en la sección marcada como SQL, Figura 25, sección que se utilizará para realizar todas las pruebas de creación de *triggers* y campos georreferenciados, la sección más importante y útil de PgAdmin4 durante el desarrollo de este trabajo de fin de grado.

Para poder añadirle una base de datos, ahora instalaremos PostgreSQL y su extensión PostGIS.

4.1.2. PostgreSQL y PostGIS

Ejecutamos el instalador de PostgreSQL y, tras pulsar en el botón siguiente, Figura 26, elegimos el directorio en el que instalarlo, Figura 27.

Figura 26: Paso 1 instalación PostgreSQL.

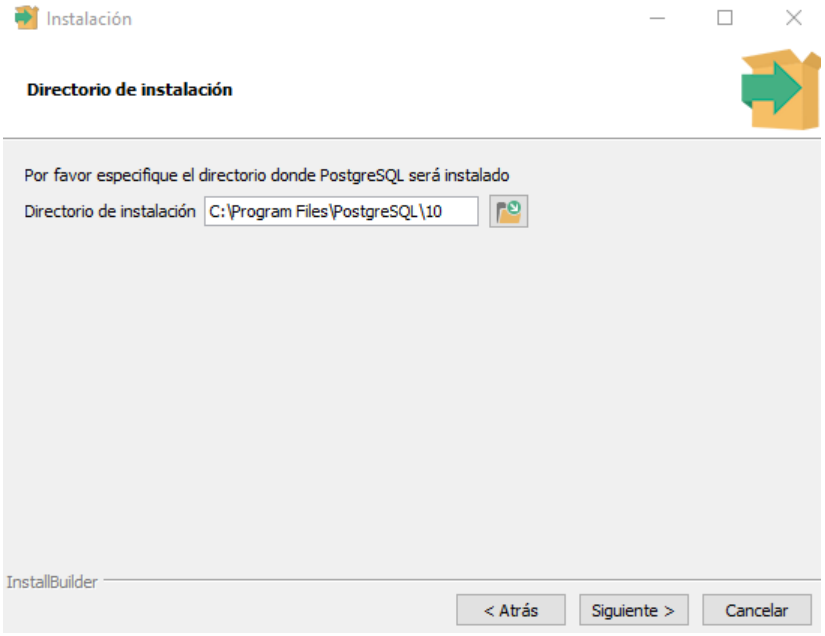


Figura 27: Paso 2 instalación PostgreSQL.

Tras elegir directorio nos salen varias opciones. Dejamos todo marcado: el servidor PostgreSQL, Stack Builder pgAdmin y Command Line Tools, Figura 28.

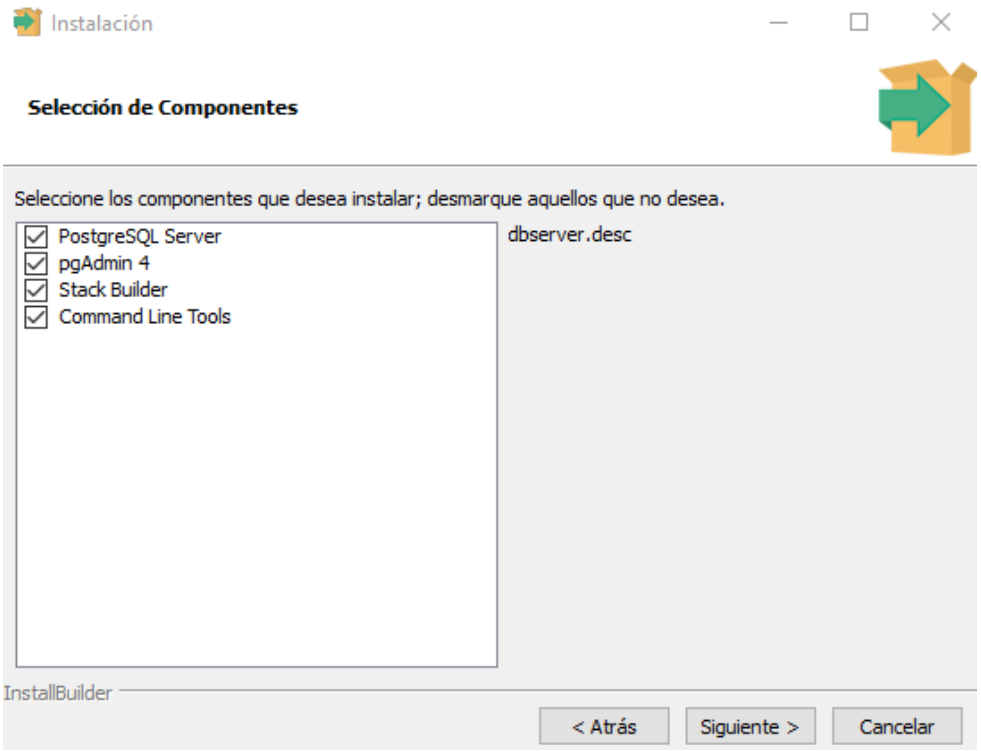


Figura 28: Paso 3 instalación PostgreSQL.

Una vez seleccionado todo, el instalador nos presenta un informe de lo que va a ser instalado, Figura 29.

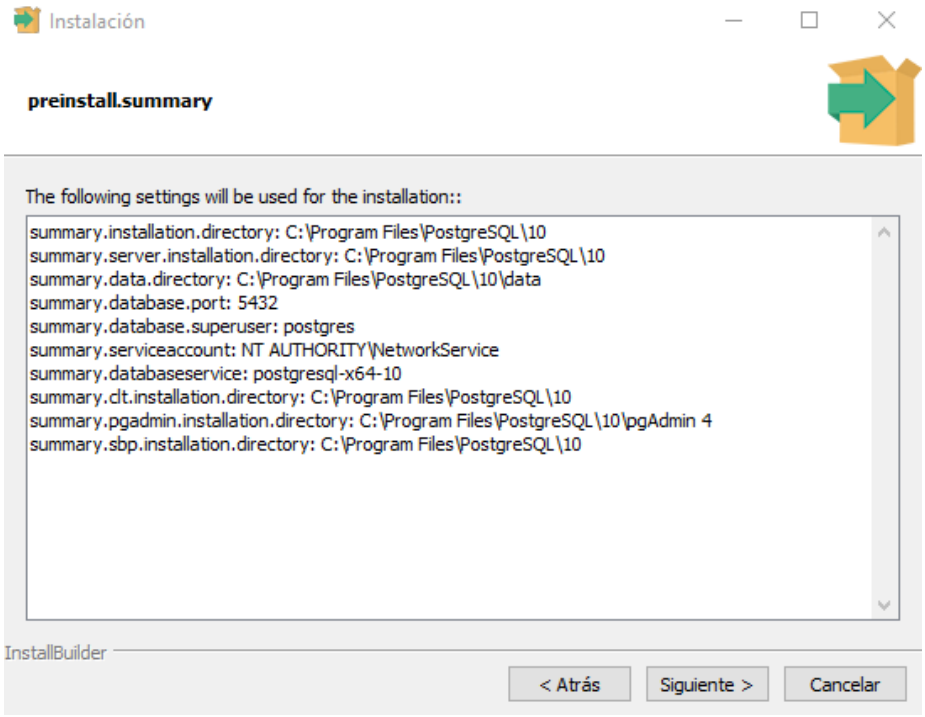


Figura 29: Paso 4 instalación PostgreSQL.

Se acepta la instalación, se espera a que termine y una vez terminada, nos dará la opción de iniciar StackBuilder (Figura 30), un gestor de complementos que proporciona PostgreSQL, y que iniciaremos, ya que será el encargado de proporcionarnos la extensión PostGIS que necesitamos para trabajar con datos georreferenciados en una base de datos.

Figura 30: Paso final instalación PostgreSQL.

Iniciada la herramienta StackBuilder, lo primero que se debe hacer es seleccionar la base de datos que nos interesa (Figura 31), en este caso PostgreSQL 10, para que nos muestre una serie de opciones, que son todas las extensiones habilitadas para PostgreSQL10.

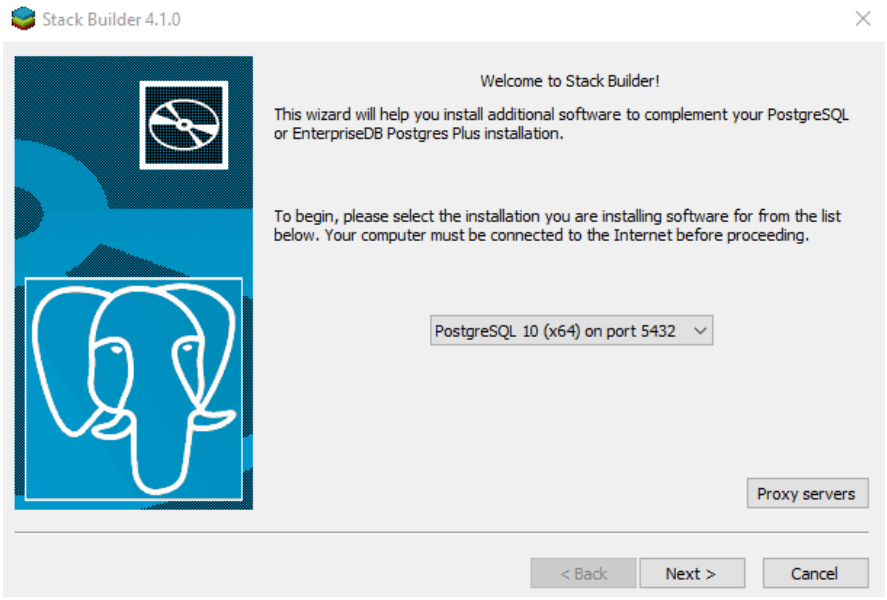


Figura 31: Paso 1 instalación PostGIS.

Para trabajar con datos geográficos, únicamente debemos marcar para instalar la extensión PostGis (Figura 32). En este caso se ha seleccionado la versión 2.4.4.

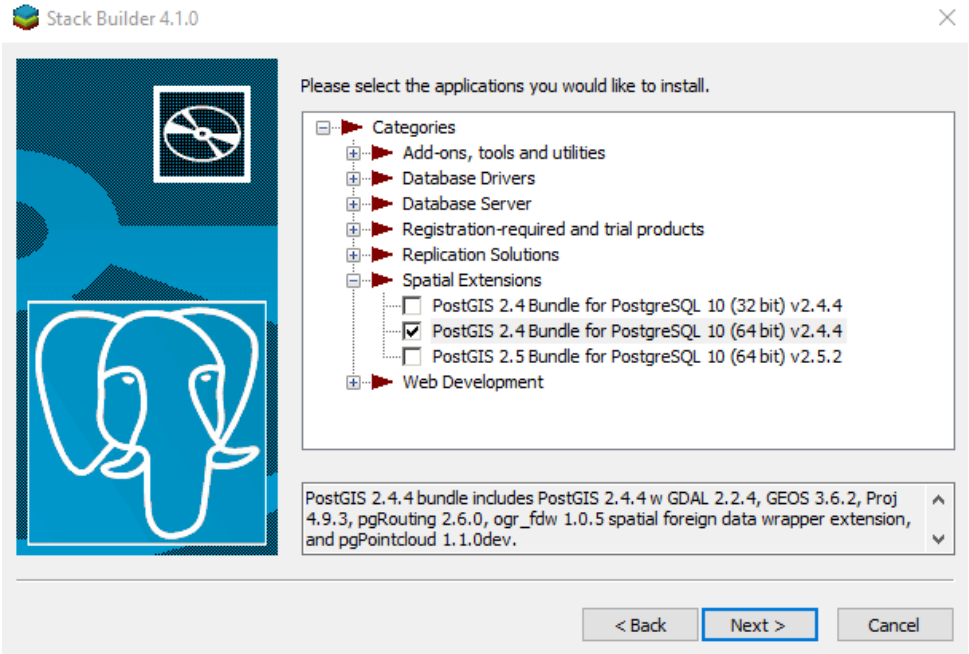


Figura 32: Paso 2 instalación PostGIS.

Tras seleccionar las utilidades que se quieren instalar, seleccionamos el directorio en el que instalaremos la extensión o extensiones marcadas (Figura 33).

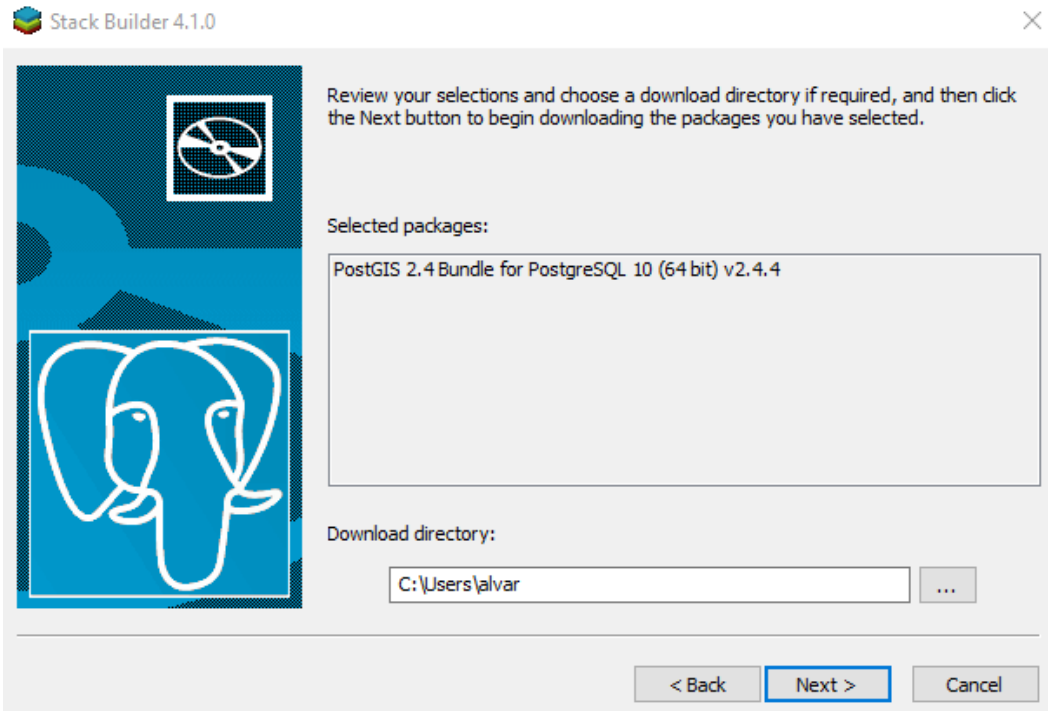


Figura 33: Paso 3 instalación PostGIS.

En la Figura 34 encontramos el inicio del proceso de instalación presionando el botón Next.

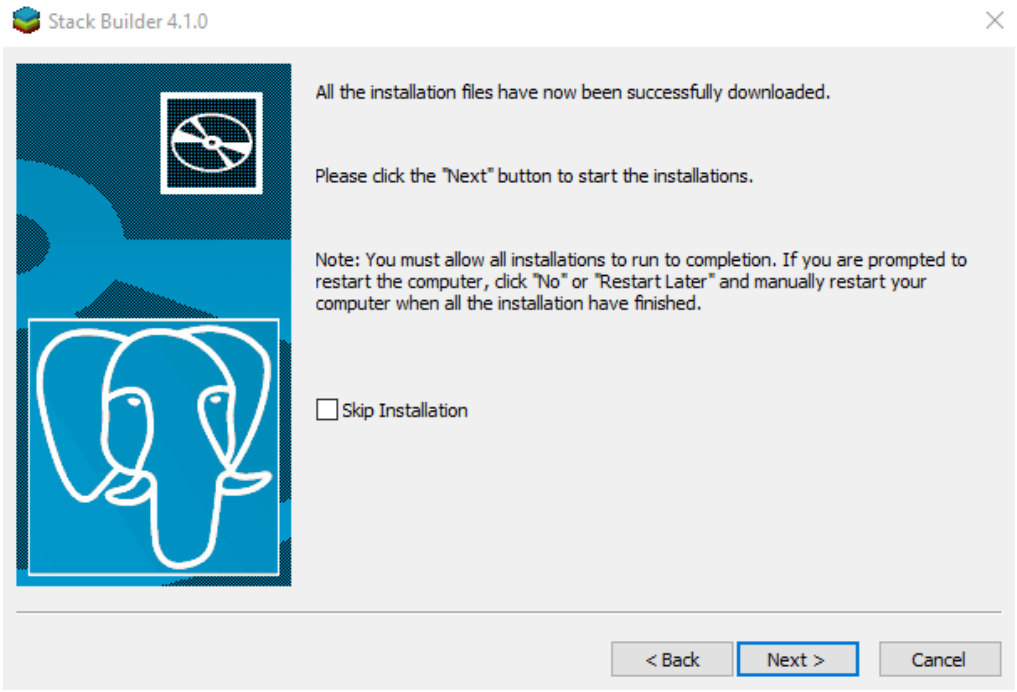


Figura 34: Paso 4 instalación PostGIS.

Debemos aceptar los términos y condiciones de PostGis (Figura 35).

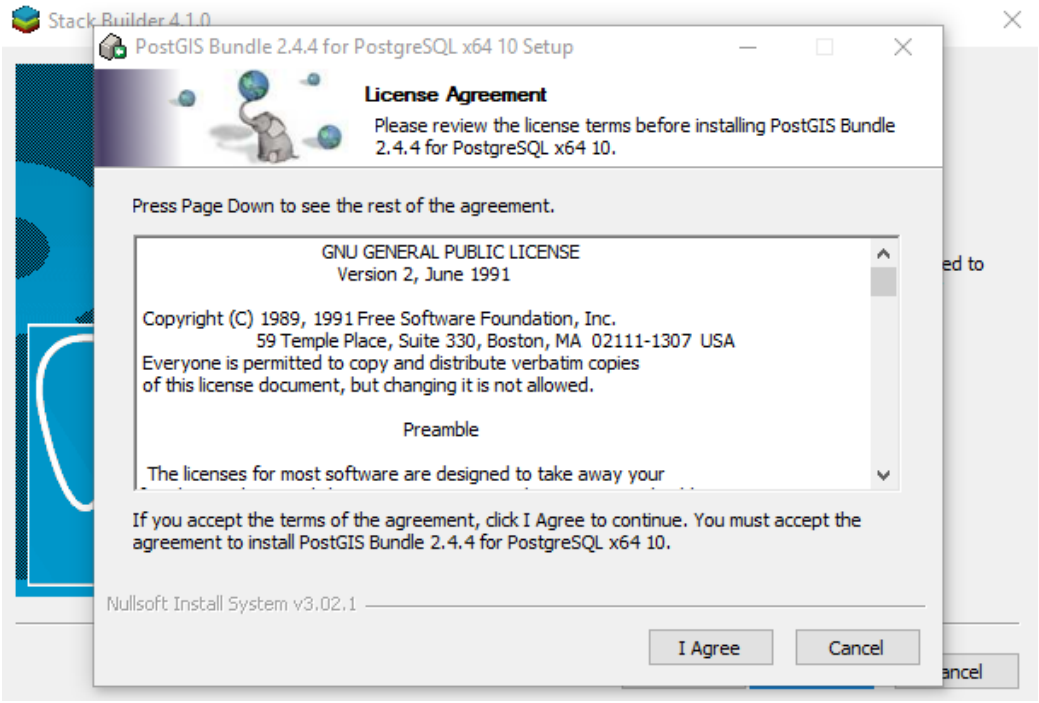


Figura 35: Paso 5 instalación PostGIS.

Una vez aceptados, nos pedirá que seleccionemos si únicamente queremos instalar la extensión PostGIS en PostgreSQL 10, o si también se quiere que cree una base de datos que acepte campos de datos georreferenciados (Figura 36).

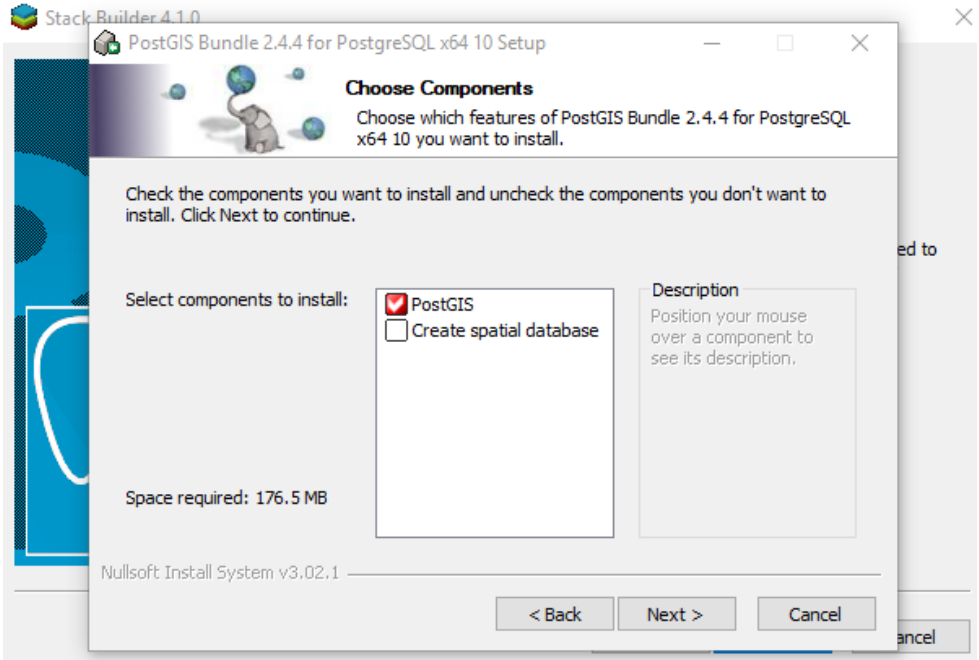


Figura 36: Paso 6 instalación PostGIS.

Cuando se ha instalado la extensión PostGIS, nos da la opción de usar un entorno para información ráster; pero, en caso de tener configuraciones anteriores, las sobrescribiría. Por lo tanto, aceptamos (Figura 37).

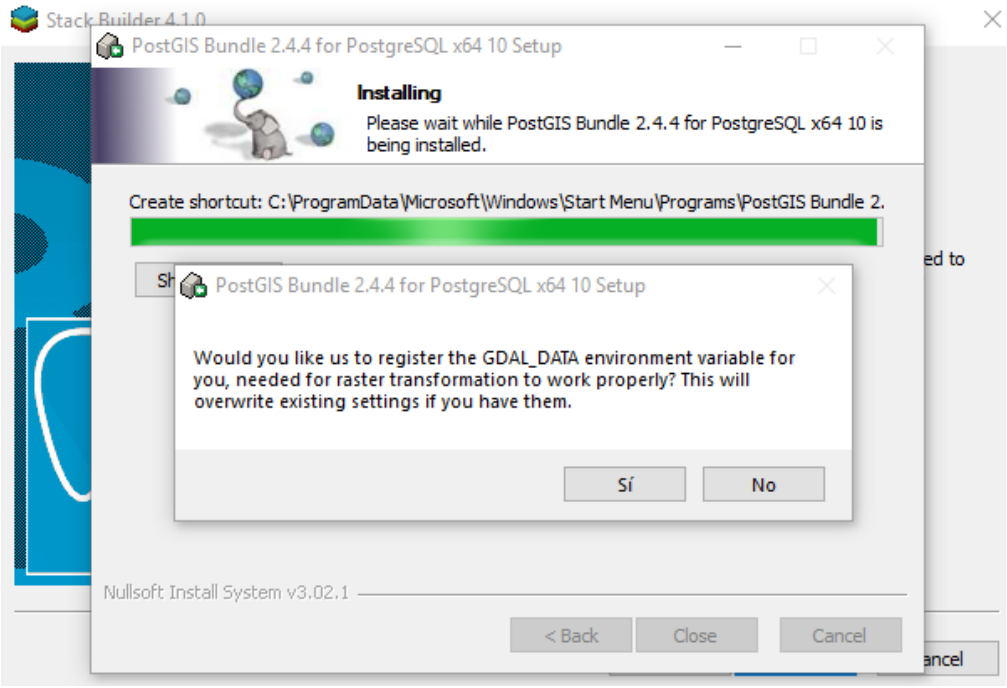


Figura 37: Paso 7 instalación PostGIS.

Una vez instalada la extensión y aceptado el entorno de información ráster, ya se puede crear una base de datos en la que guardar datos georreferenciados.

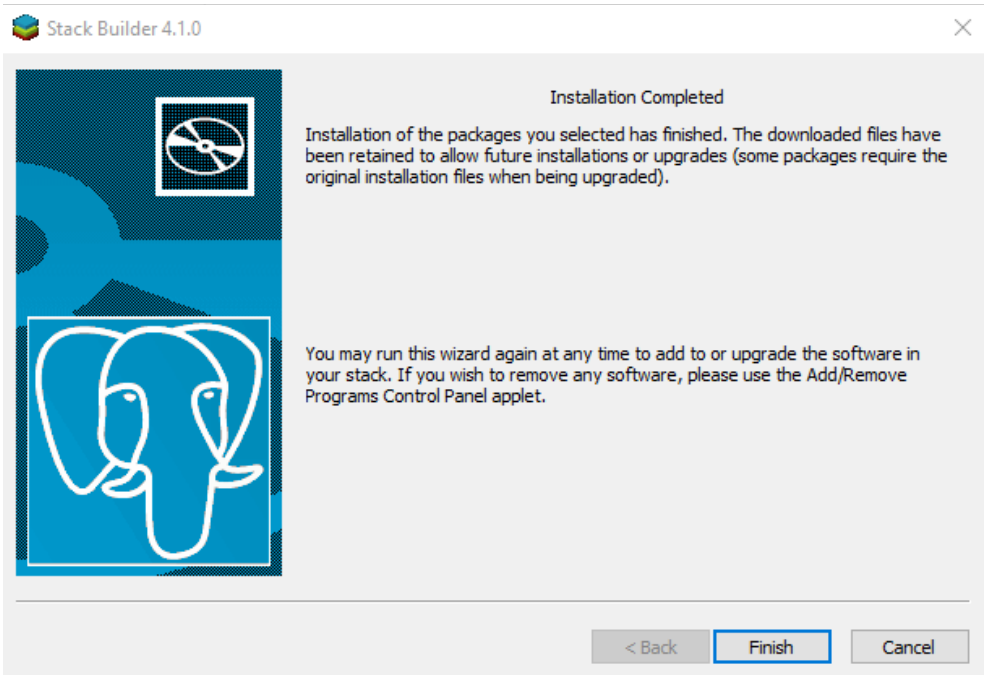


Figura 38: Paso final instalación PostGIS.

4.1.3. Eclipse, Tomcat y driver PostgreSQL

Ahora es el turno del entorno de desarrollo en el que, como se ha comentado anteriormente, Eclipse ha sido el seleccionado.

Descargamos e instalamos la versión IDEE.

También instalamos Tomcat para poder lanzar nuestra aplicación desde un servidor. La opción más cómoda para obtener apache Tomcat es, simplemente descargarse los archivos comprimidos. Se descomprimen y se guardan en una carpeta. En Eclipse se configura el servidor a la carpeta descomprimida y ya se puede utilizar Tomcat.

- **Configuración de Tomcat en Eclipse**

En primer lugar, seleccionamos el archivo que será la pantalla principal de la aplicación, en este caso index.jsp y, con clic derecho sobre ese archivo, *se clica Run As > Run on Server*, tal como se muestra en la Figura 39.

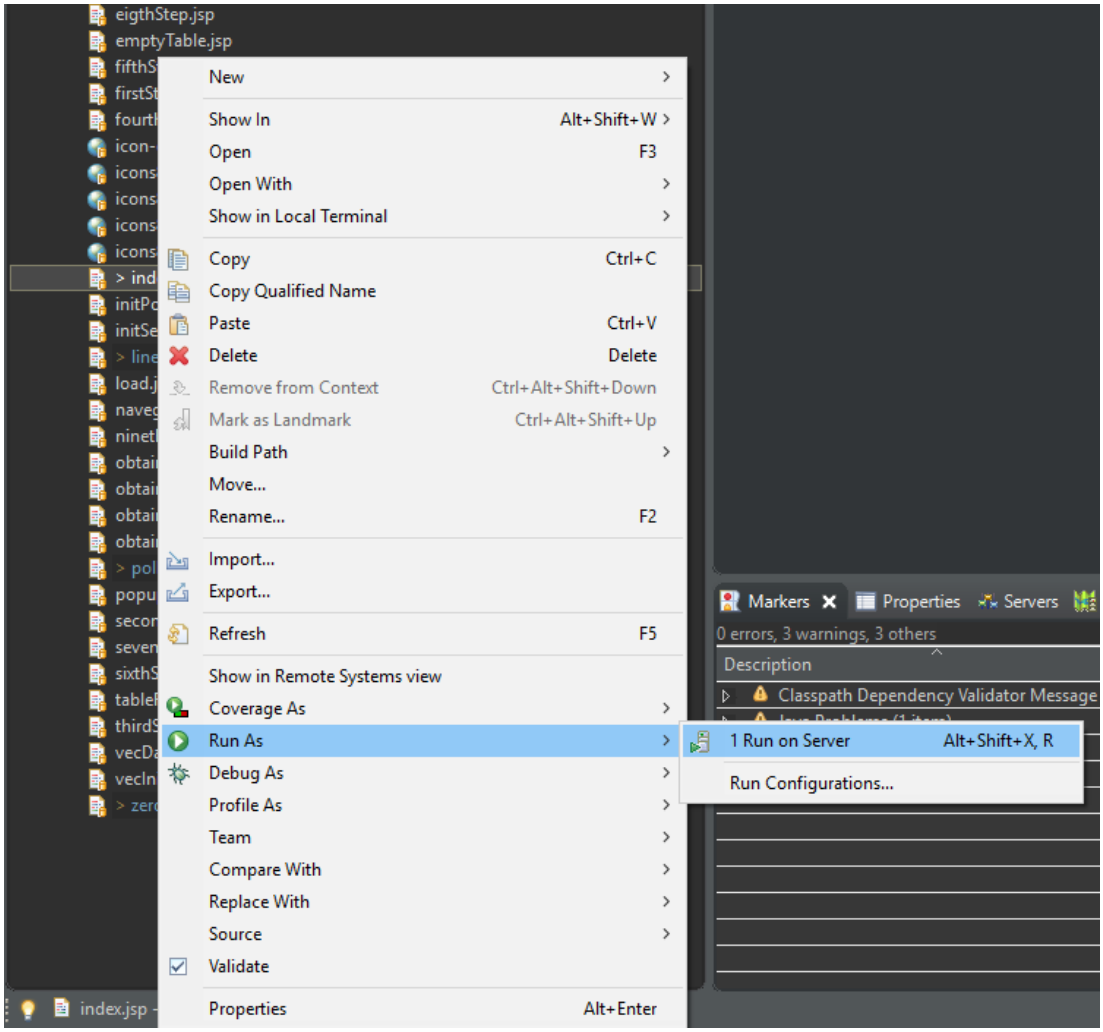


Figura 39: Configuración Tomcat en Eclipse – paso 1.

En la ventana que se abre, deberemos seleccionar la carpeta Apache, y en ella debemos elegir justo la versión de la que disponemos (Figura 40).

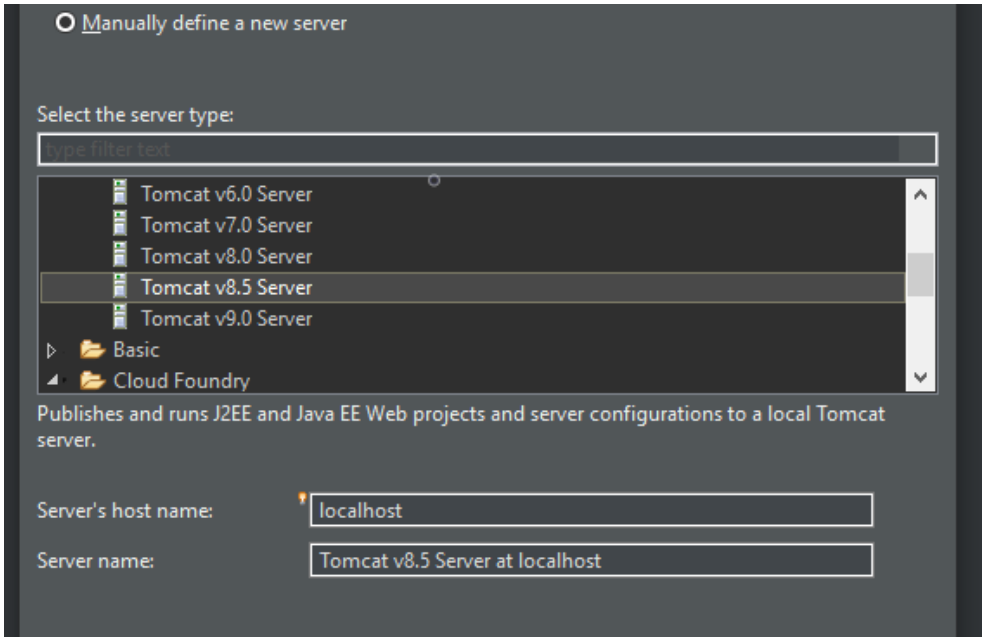


Figura 40: Configuración Tomcat en Eclipse – paso 2.

El siguiente paso consiste en seleccionar el directorio en el que hemos guardado la carpeta descomprimida de apache, donde tiene toda la configuración de la versión elegida en el paso anterior, que necesita Apache Tomcat, tal como se muestra en la Figura 41.

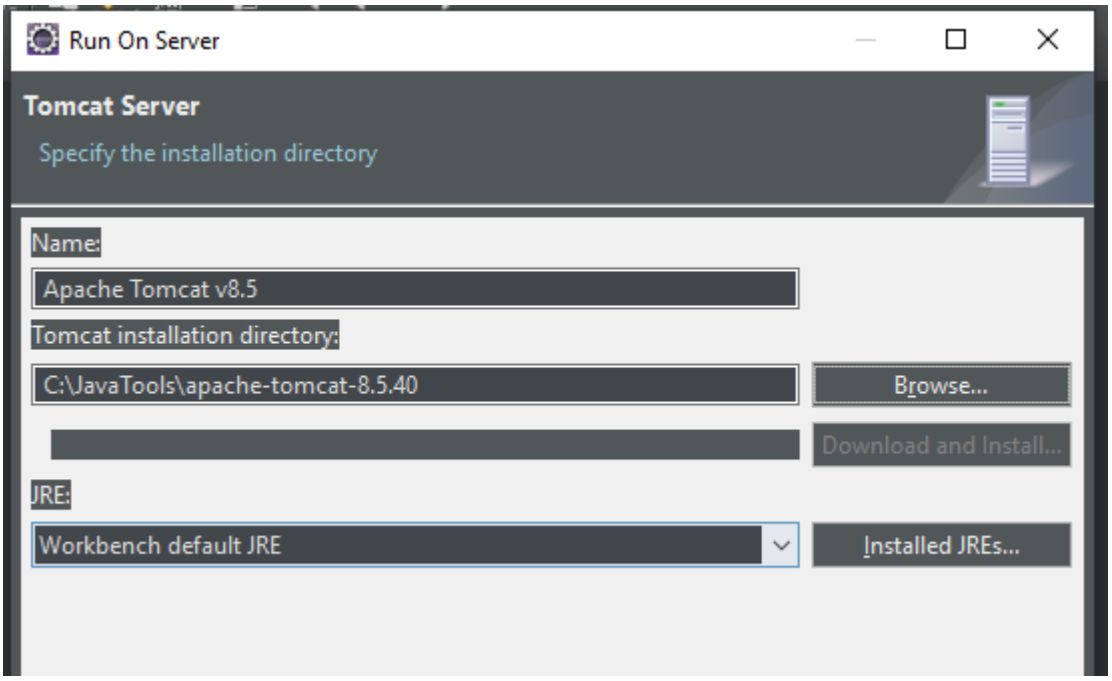


Figura 41: Configuración Tomcat en Eclipse – paso 3.

Tras esto, se pulsa en el botón de *Finish* y se ejecutará el servidor. Por defecto, se desplegará en el puerto 8080, pero este puerto se puede modificar desde el archivo server.xml.

- **Driver PostgreSQL**

Una vez tenemos en un servidor nuestra web, esta debe ser capaz de comunicarse con la base de datos. Para ello es necesario el controlador para la versión que estemos utilizando de la base de datos, en este caso el driver de la versión 10 de PostgreSQL.

En primer lugar, debemos configurar el Build Path del proyecto, al que se accede tal como muestra la Figura 42.

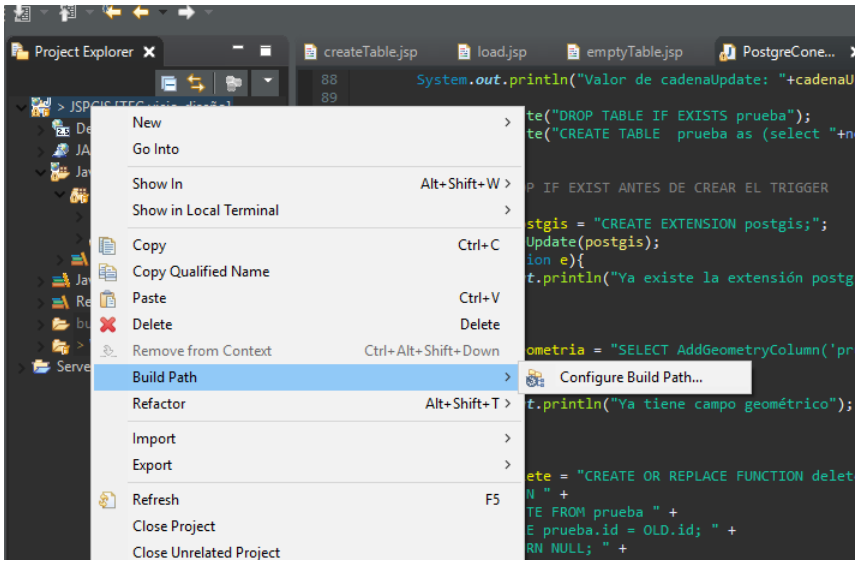


Figura 42: Configuración PostgreSQL en Eclipse.

Dentro del Build Path debemos darle a añadir JARs externos, buscamos la ubicación en la que está el controlador que necesitamos y lo añadimos, (Figura 43). Completado este proceso ya puede establecerse una conexión entre la web y la base de datos.

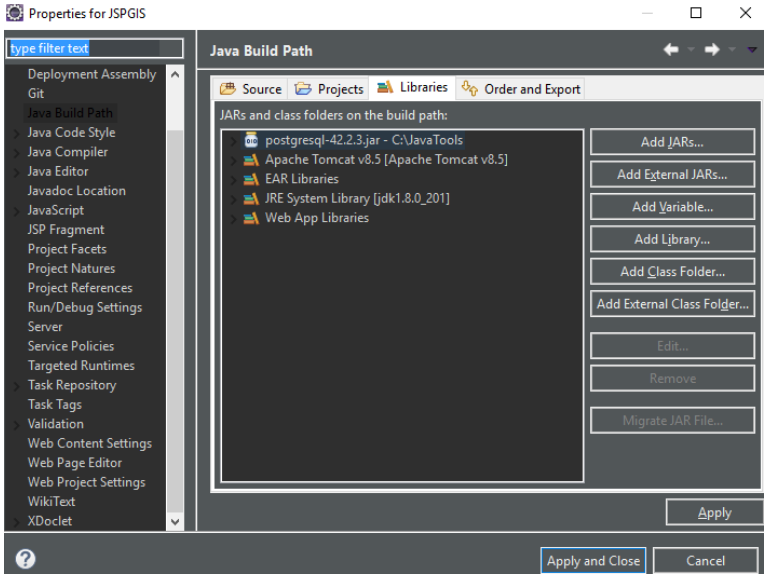


Figura 43: Configuración PostgreSQL en Eclipse.

4.1.3. Despliegue de GeoServer

Para poder utilizar GeoServer en nuestra aplicación, se le proporciona al alumno una máquina virtual con IP 192.168.67.5 en la que se desplegará[17] Geoserver usando Tomcat.

1. En primer lugar accedemos a la máquina virtual autenticándonos mediante ssh mediante el siguiente comando.
   ssh ubuntu@192.168.67.5 -i ~/.ssh/id_rsa

2. Una vez dentro de la máquina virtual instalamos Tomcat 8.
sudo apt-get install default-jre tomcat8 tomcat8-admin

Una vez se ha instalado Tomcat se deben modificar dos archivos, tomcat-user.xml y web.xml.

3. Para modificar el archivo tomcat-user.xml escribimos el siguiente comando:
   sudo nano /etc/tomcat8/tomcat-users.xml
   Agregamos un usuario en Tomcat

```
<role rolename="manager-gui"/>
        <user username="admin" password="admin"
              roles="manager-gui,admin-
gui,manager,admin,manager-script,admin-script"/>
```

4. Para modificar el archivo web.xml es necesario el siguiente comando
   sudo nano /usr/share/tomcat8-admin/manager/WEB-INF/web.xml
   Aumentamos el tamaño de archivo de despliegue

```
<multipart-config>
<!-- 50MB max -> Increase this value below: -->
<max-file-size>92428800</max-file-size>
<max-request-size>92428800</max-request-size>
```

5. Con los archivos modificamos, movemos el archivo .war al directorio de Tomcat desde el cual, cuando arranque utilizará el archivo .war para el despliegue de la aplicación en el puerto por defecto de Tomcat, 8080, ya que no ha sido modificado
   sudo mv geoserver.war /var/lib/tomcat8/webapps/

6. Por último paramos Tomcat y lo volvemos a iniciar para que GeoServer ya esté disponible en el puerto 8080 de la máquina virtual, tal como se muestra en la Figura 44.
   sudo service tomcat8 stop
   sudo service tomcat8 start

[17] GeoServer Installation On Ubuntu Server 18.04 LTS. (2019). Disponible en https://gist.github.com/falu/1c077f67f009ac487305983c78c767b7

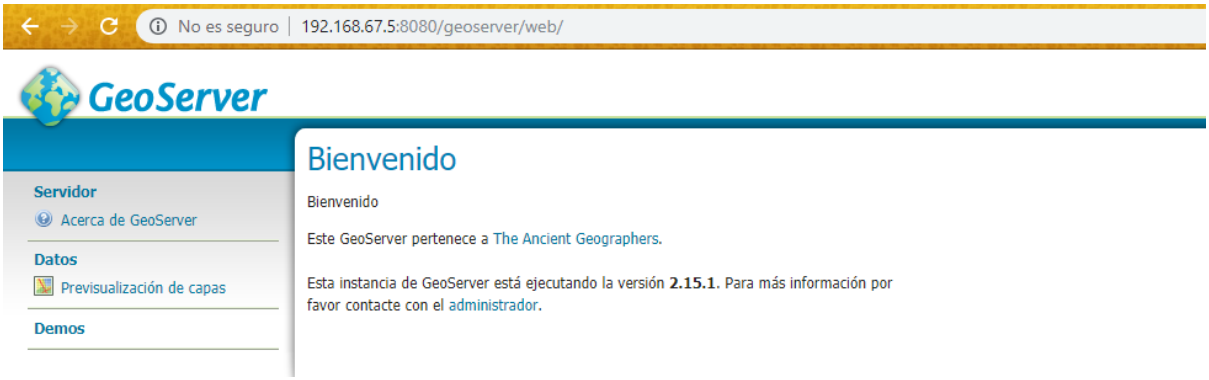


Figura 44: GeoServer desplegado en la máquina virtual proporcionada.

4.2. Implementación

Como se ha comentado en el punto 2.3, en la fase de desarrollo se ha seguido una metodología ágil, en la que el alumno ha sido el desarrollador y los profesores los clientes, con reuniones mensuales en las que se enseñaban las nuevas versiones y funcionalidades de lo desarrollado hasta el momento y así detectar fallos de manera más temprana y que el coste de modificarlos fuese mucho menor.

4.2.1. Primer Sprint

Para desarrollar todo este proyecto, una parte importante ha sido la familiarización con y el estudio de las herramientas necesarias, por lo que se empleó parte de este primer sprint en el estudio de los tipos de datos georreferenciados y las particularidades de la extensión PostGIS, así como con trabajar con capas ya creadas con la aplicación QGIS, en la que se crearon capas georreferenciadas de varios tipos.

Durante el primer Sprint se trabaja, en parte, con la base de datos PostgreSQL y su extensión PostGIS, en la que se prueba a añadir manualmente la extensión, ya que cuando se utilice la aplicación final, la base de datos introducida no tiene por qué tener instalada la extensión.

Se pone en práctica añadir la extensión por si es necesario instalar algún complemento. Para ello, creamos varias bases de datos vacías, nada preinstalado, siguiendo el siguiente proceso.

Con el gestor PgAdmin4, la creación de base de datos es muy sencilla, como vemos en la Figura 45. Debemos darle clic derecho en Databases y seleccionar *Create* > *Database.* Aunque otra opción es desde el menú desplegable Object, en la parte superior del gestor de bases de datos, seleccionar *Create* y clicar en *Database*.

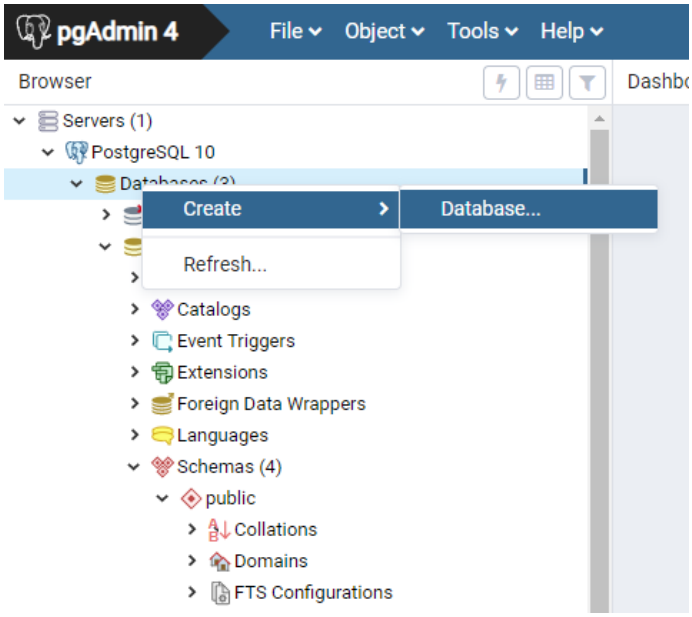


Figura 45: Creación base de datos PgAdmin4.

Para crear la base de datos, aparecerá una ventana nueva, que se superpondrá a PgAdmin4, Figura 46, en la que podremos darle un nombre a la base de datos, cambiar el propietario de esta y modificar seguridad y parámetros. Para las pruebas, con indicar un nombre para la base de datos y dejar al propietario por defecto será suficiente.

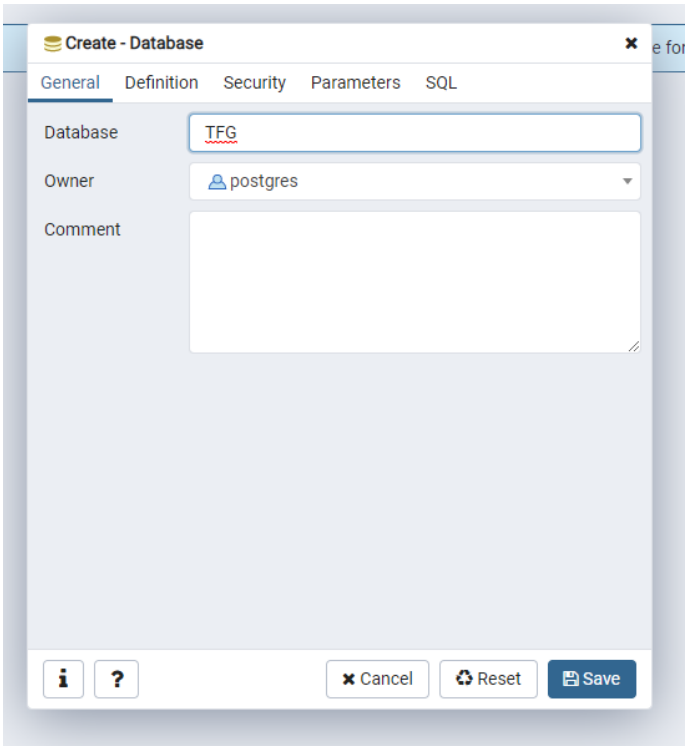


Figura 46: Creación base de datos PgAdmin4.

Comprobamos que no hay ningún *trigger* creado en la nueva base de datos, Figura 47, ya que es la manera más sencilla de comprobar si tiene añadida la extensión PostGIS, porque al

crearla, lo primero que le añade a la base de datos son dos *triggers* para trabajar con información georreferenciada.

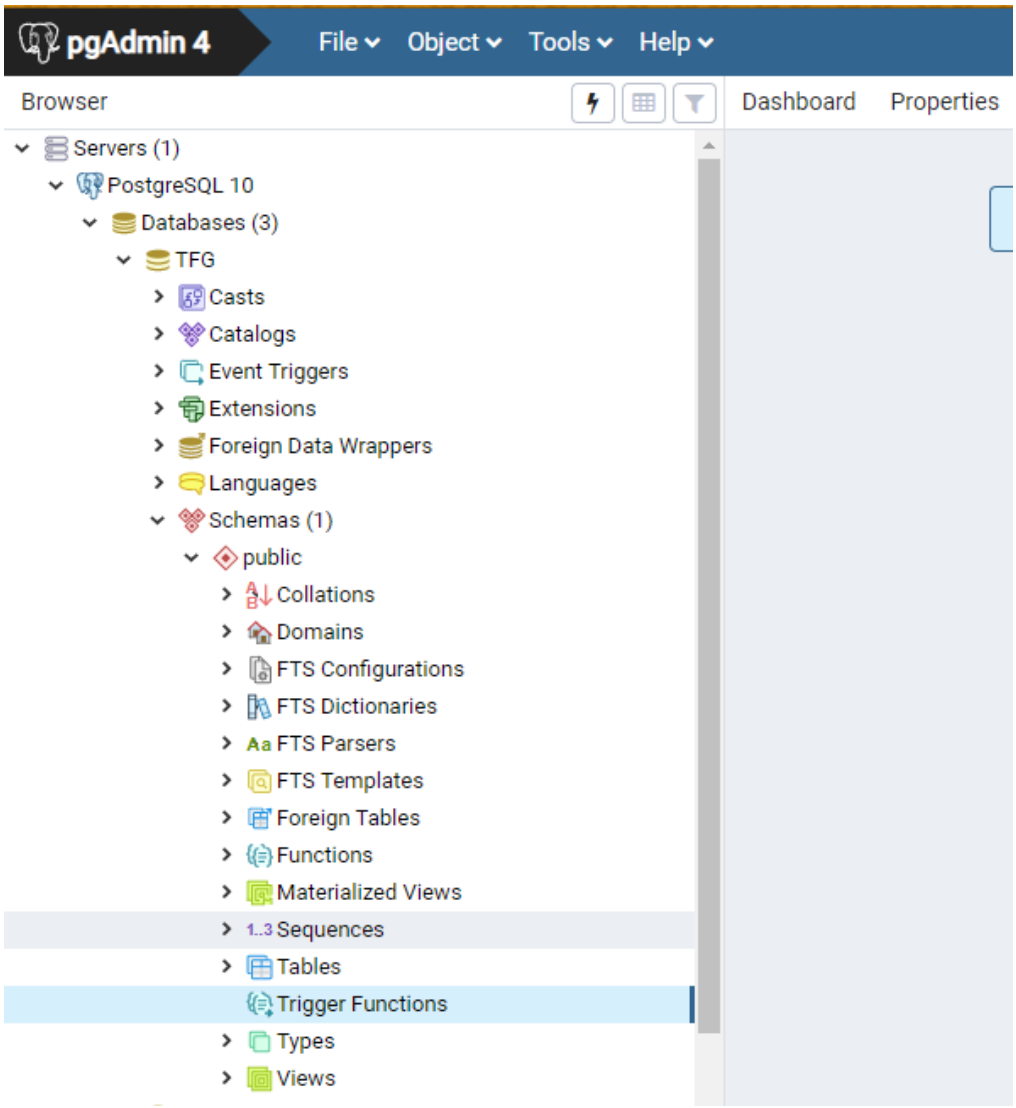


Figura 47: Base de datos creada.

Creamos un script en nuestra base de datos haciendo clic derecho sobre esta y pulsando CREATE Script. Por defecto nos aparecerá el que se muestra en la Figura 48. Lo borramos y escribimos el script que queremos ejecutar.

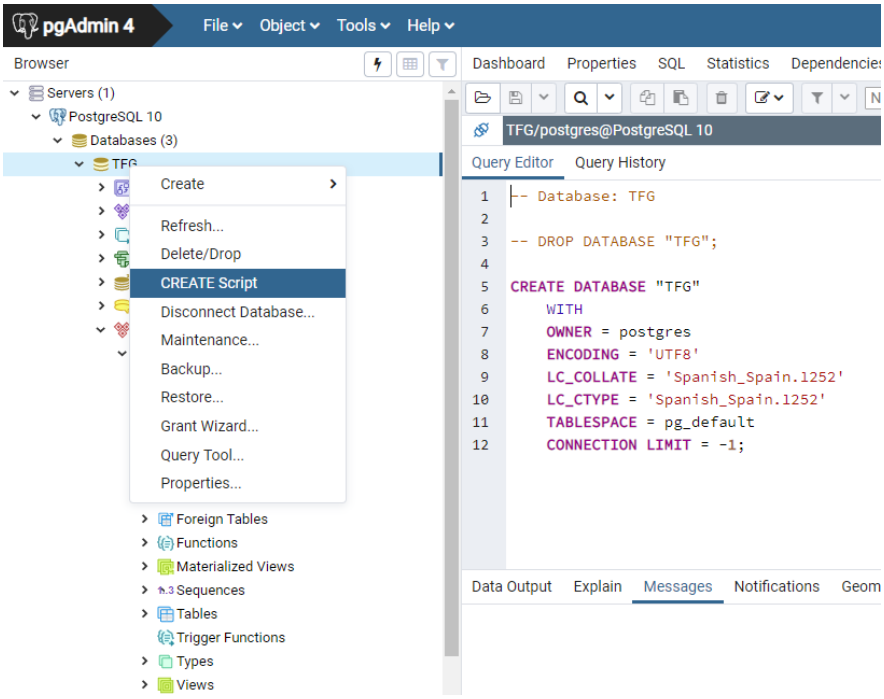


Figura 48: Creación de Script SQL en PgAdmin4.

El script SQL que se ha de poner para crear la extensión PostGIS en una base de datos es el siguiente:

```
CREATE EXTENSION postgis;
```

En la Figura 49 podemos ver como al ejecutar este script, no da ningún error y aparece el mensaje para confirmar que la extensión se crea correctamente

.

Figura 49: Creación de Script SQL en PgAdmin4.

También se puede comprobar que se ha instalado correctamente verificando que los *triggers* asociados a la extensión PostGIS están creados en nuestra base de datos, tal y como se muestra en la Figura 50.

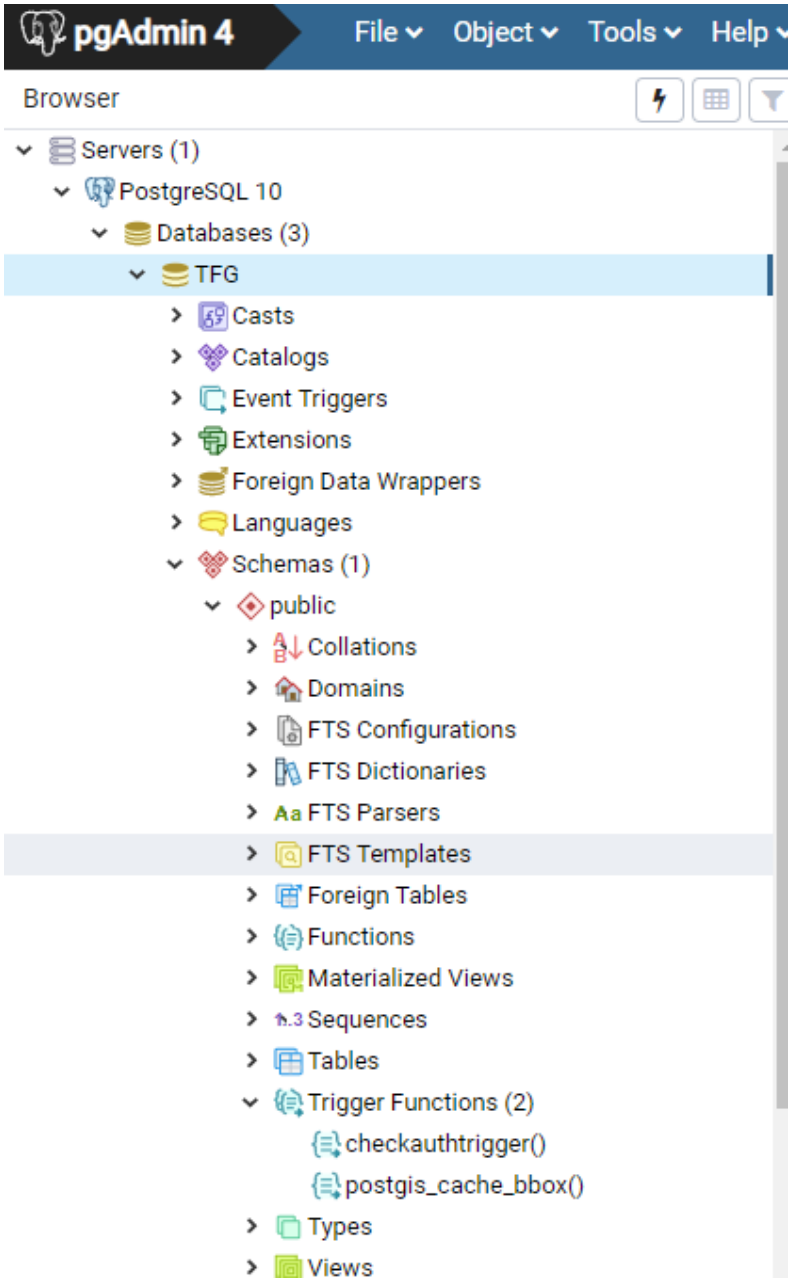


Figura 50: Confirmación del añadido de PostGIS a la base de datos.

Con la introducción de ese script desde la sección SQL de PgAdmin 4, ya tenemos una base de datos en la que podemos crear una tabla que contenga un campo georreferenciado para guardar la información necesaria en él. Con estas pruebas más la previa documentación sobre la extensión y los datos georreferenciados, se llega al final del primer sprint.

4.2.2. Segundo Sprint

Una vez terminada la reunión con los tutores, tras comprobar un avance satisfactorio en el proyecto y comprobado que funciona la extensión, se procede a crear varios campos en distintas tablas que puedan guardar datos georreferenciados. Los datos guardados son tanto de tipo punto como *linestring* y polígonos, que son con los que trabajaremos durante toda la aplicación. Además de hacer pruebas, se crean *triggers* y se comprueba su funcionamiento para que dos tablas siempre tengan sus datos en consonancia. También se comienza con el diseño de la web según lo hablado durante la reunión anterior al comienzo del segundo sprint.

En la Figura 51 se muestra el mapa integrado en PgAdmin4, en el cual podemos ver los datos georreferenciados que estén en una base de datos de PostgreSQL, además de los scripts necesarios para crear el campo que puede almacenar los datos de tipo polígono y la inicialización de un polígono.

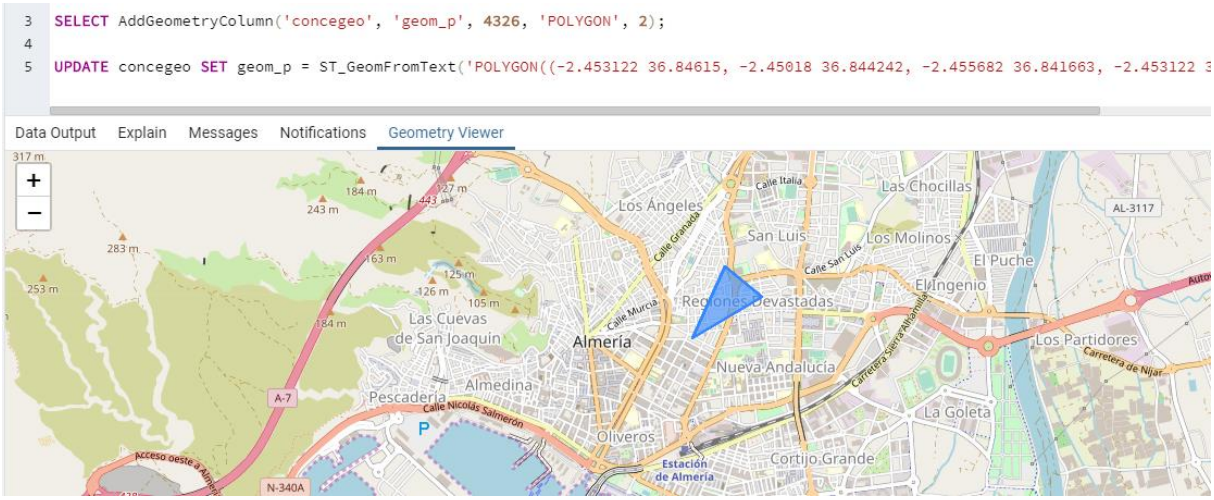


Figura 51: Datos vectoriales de tipo polígono mostrados en PgAdmin4.

A continuación, se mostrarán los scripts necesarios para añadir e inicializar datos de tipo punto y de tipo linestring en la base de datos.

- **Campo de tipo punto**

```
SELECT AddGeometryColumn('concesionarios', 'geom', 4326, 'POINT',
2);
UPDATE concesionarios SET geom = ST_SetSRID(ST_MakePoint('-2.45815',
'36.81885'), 4326);
```

- **Campo de tipo linestring**

```
SELECT AddGeometryColumn ('public','concegeo', 'geom_l', 4326,
'LINESTRING',2);
UPDATE concegeo SET geom_l = ST_GeomFromText('LINESTRING(-122.3 47,
-115 42, -115 43)',4326);
```

Se hacen pruebas añadiendo triggers[18], tanto de inserción, actualización y eliminación, se debe saber cómo se crean los triggers para poder automatizarlos en un futuro.

- **Trigger de inserción:**

```
CREATE OR REPLACE FUNCTION insert_triggerconcegeo12()
RETURNS TRIGGER AS $$ BEGIN INSERT INTO concegeo12(nombre, ciudad)
VALUES (new.nombre, new.ciudad);
RETURN NULL;
END; $$ LANGUAGE 'plpgsql';
```

- **Trigger de actualización:**

```
CREATE OR REPLACE FUNCTION update_triggerconcegeo12()
RETURNS TRIGGER AS $$ BEGIN UPDATE concegeo12
SET nombre = new.nombre AND ciudad = new.ciudad WHERE
concegeo12.nombre = old.nombre AND concegeo12.ciudad;
RETURN NULL; END; $$ LANGUAGE 'plpgsql';
```

- **Trigger de eliminación:**

```
CREATE OR REPLACE FUNCTION delete_triggerconcegeo12()
RETURNS TRIGGER AS $$ BEGIN DELETE FROM concegeo12
WHERE concegeo12.nombre = old.nombre AND concegeo12.ciudad =
old.ciudad;
RETURN NULL;
END; $$ LANGUAGE 'plpgsql';
```

Se ejecutan los scripts mostrados anteriormente y se comprueba que funcionan. Para ello, debemos consultar los datos que hay en ambas tablas recordando que los scripts están hechos con la modificación en una dirección, es decir, se modificará la tabla concegeo12, nombre debido a ser la decimosegunda prueba que se hacía, después de algún cambio en la tabla concesionarios. En la Figura 52 y Figura 53 se mostrarán las tablas con los que comienzan ambas tablas.

[18] PostgreSQL: Documentation: 10: CREATE TRIGGER. (2019). Disponible en https://www.postgresql.org/docs/10/sql-createtrigger.html

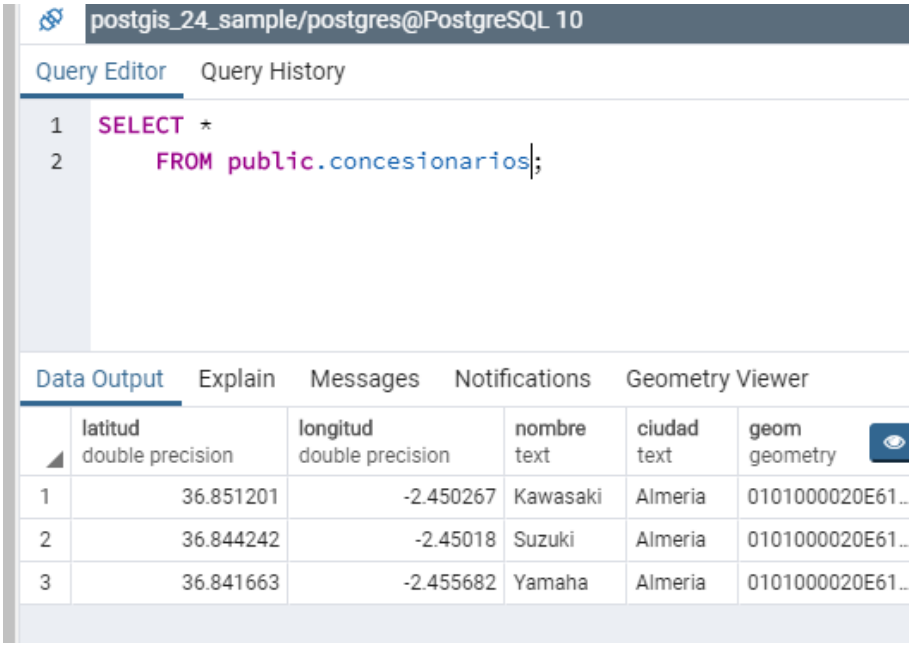


Figura 52: Datos de la tabla concesionarios

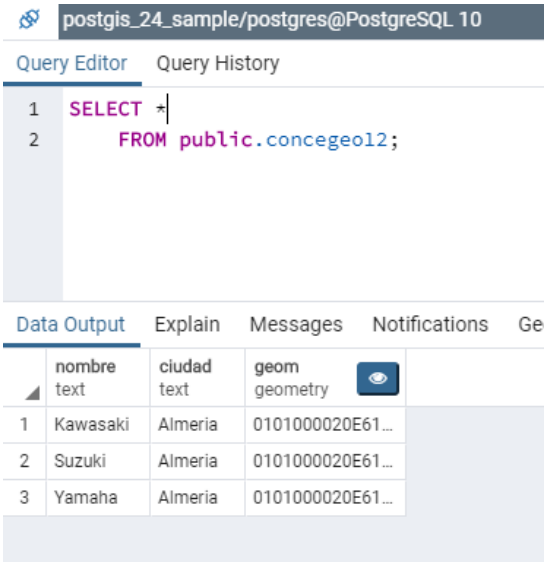


Figura 53: Datos de la tabla concegeo12.

Ahora que sabemos los datos que existen en la tabla, procedemos a borrarlos ejecutando el siguiente script.

```
DELETE FROM public.concesionarios
       WHERE nombre='Kawasaki';
```

Una vez ejecutado volvemos a consultar el contenido de ambas tablas para comprobar que se ha eliminado de las dos, no ha fallado nada y la información que en el campo nombre tenía el valor Kawasaki ha sido eliminada en ambas tablas, como se puede ver en la Figura 54 y Figura 55.

Figura 54: Datos de la tabla concesionarios tras la eliminación.

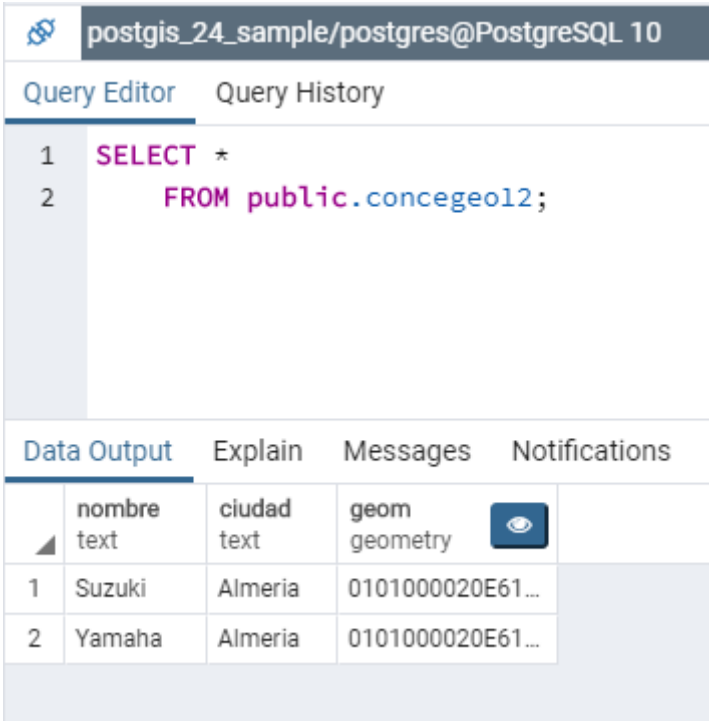


Figura 55: Comprobación de funcionamiento del trigger de eliminación.

Se comienza a codificar lo diseñado y hablado durante la reunión anterior al sprint y surgen los primeros problemas al intentar guardar en un array los datos que se van obteniendo de la

aplicación web para posteriormente pasarlos a la BD, ya que JavaScript[19] por sí solo no tiene conexión con la base de datos. Se puede utilizar una extensión como Node.js, pero en este caso se ha optado por utilizar Java[20]. A este problema se le da solución declarando un ArrayList en Java y asociándolo con un array en Javascript, tal como se muestra a continuación[21].

```
<%!public ArrayList<String> nuevaTablaJ;
nuevaTablaJ = new ArrayList<String>();%>
var nuevaTabla = <%=nuevaTablaJ%>;
```

De esta manera, ya podemos guardar los datos que en un futuro introducirá el usuario en la aplicación web, recogerá JavaScript y pasará a Java para hacer la conexión con la base de datos partiendo de la información introducida por el usuario. Por el momento, para las primeras versiones no hará falta la introducción de datos, todo será automático con una base de datos y tabla predefinidas, pero se prepara la estructura para cuando se necesite.

Se comienza con el diseño y una versión básica de lo que podría ser la aplicación, con los datos mostrados en una tabla situada a la izquierda, los cuales podrían elegirse y pasarse a una tabla en la que se situarían relaciones de 1-1 o relaciones de 1-muchos.

El estilo se ha hecho con CSS[22], incluido en el archivo index.jsp, ya que por el momento en esta página están todos los elementos mostrados en esta primera versión. Al fondo se le ha añadido color, una tonalidad de azul claro, para que no resulte un fondo pesado, tal y como se muestra a continuación y color a las letras del título.

Ejemplo de estilo en CSS:

```
body {
        background: #e4fffd;
}
```

[19] JavaScript Tutorial. (2019). Disponible en https://www.w3schools.com/js

[20] Java Platform Standard Edition 8 Doc, *Docs.oracle.com.* (2019). Disponible en https://docs.oracle.com/javase/8/docs/

[21] Jsp, Mendoza, L. (2019). Accessing java variable from javascript jsp. Disponible en https://stackoverflow.com/questions/18990700/accessing-java-variable-from-javascript-on-same-jsp

[22] CSS Reference, W3schools.com. (2019). Disponible en https://www.w3schools.com/cssref/

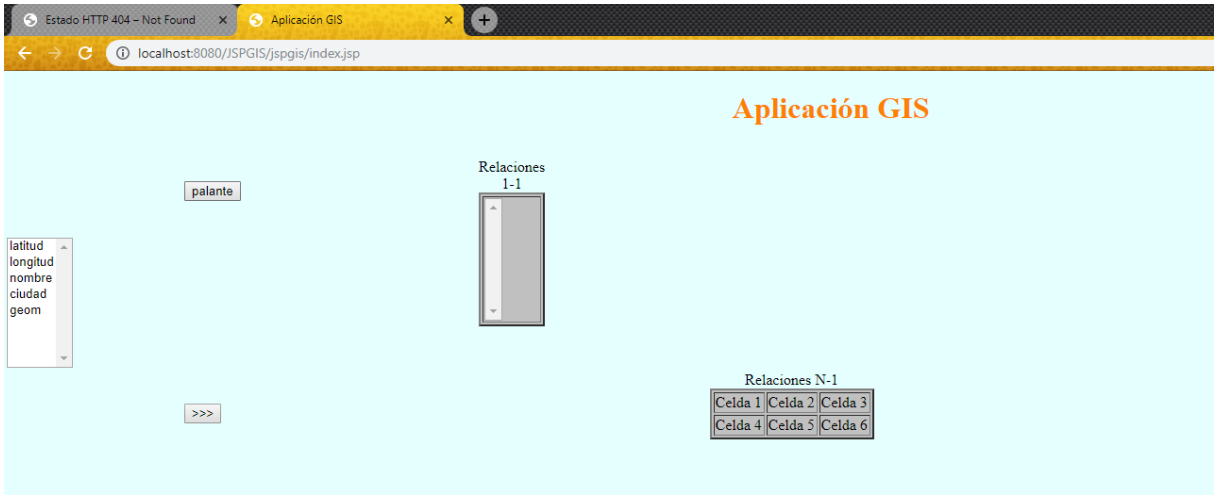


Figura 56: Primera versión interfaz visual de la aplicación web.

En la Figura 56 se puede ver lo primitiva que era la interfaz inicial, y como en una primera idea se pretendía diferenciar campos en relaciones 1-1 y 1-muchos

4.2.3. Tercer Sprint

Se codifica una primera versión en la cual están implementadas varias de las funcionalidades necesarias de la aplicación web, pero el usuario aún no puede introducir datos. La base de datos, el nombre de la tabla, los campos que se seleccionan, el nombre de la nueva tabla, y la creación de triggers no varían de una ejecución a otra.

Además, se hacen pruebas para, visualmente, cambiar un elemento entre dos tablas, lo que en un futuro supondrá escoger qué campos de la tabla origen tendrá la nueva tabla. Esta funcionalidad por el momento únicamente es visual, los campos de la nueva tabla están de manera predeterminada en el código.

En la Figura 57 y Figura 58 se ve un ejemplo de una prueba en la que un elemento se elimina de la tabla original y pasa a otra tabla.

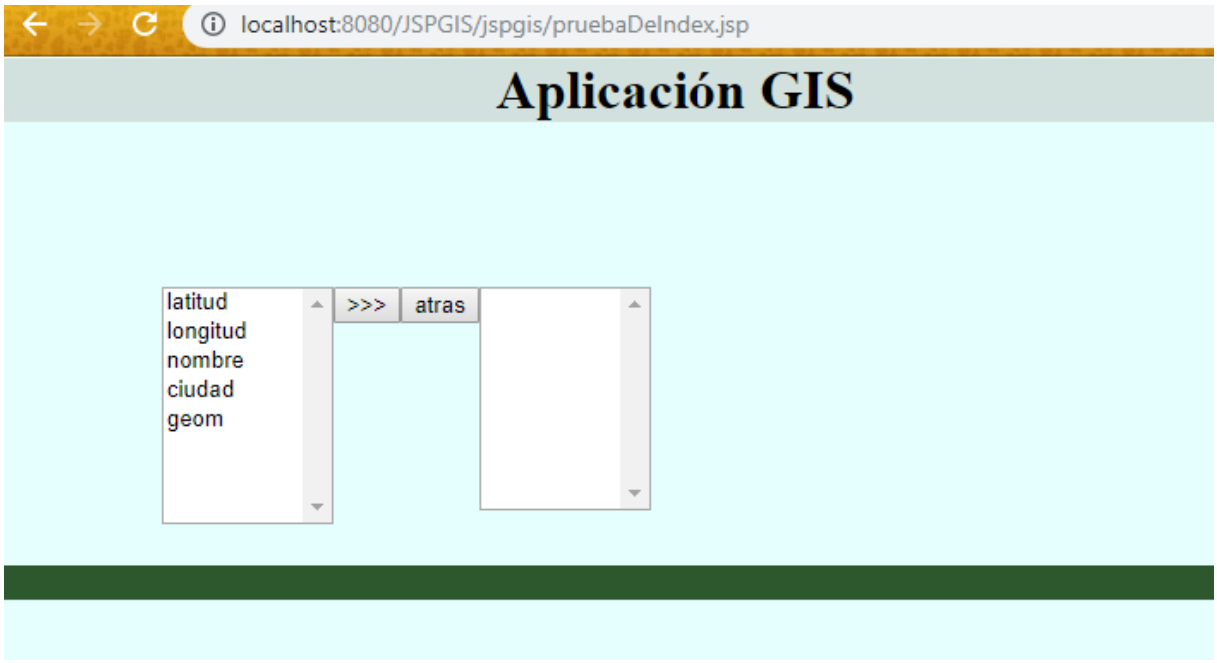


Figura 57: Pruebas para cambiar elemento de tabla.

Figura 58: Elemento cambiado de tabla.

A continuación, se muestra el código para cambiar un elemento de tabla:

```
function sendItemRight() {
    var option = document.createElement("option");
    var x = tableLoadReg.selectedIndex;
    var sel = tableLoadReg.options[tableLoadReg.selectedIdex];
    option.text = tableLoadReg.options.item(x).text;
    tableEmpty.add(option, sel);
    tableLoadReg.remove(x);
}
```

Una vez hechas las pruebas y comprobado que funciona correctamente el movimiento de elementos entre tablas, se añade al funcionamiento de la aplicación web. En esta versión, como se ha comentado anteriormente, aunque de manera visual puedes cambiar elementos de tabla, no supone ninguna modificación a la hora de crear la nueva tabla. Se hace únicamente para preparar la funcionalidad y tenerla lista y funcionando cuando se necesite, tal como se puede ver en la Figura 59.

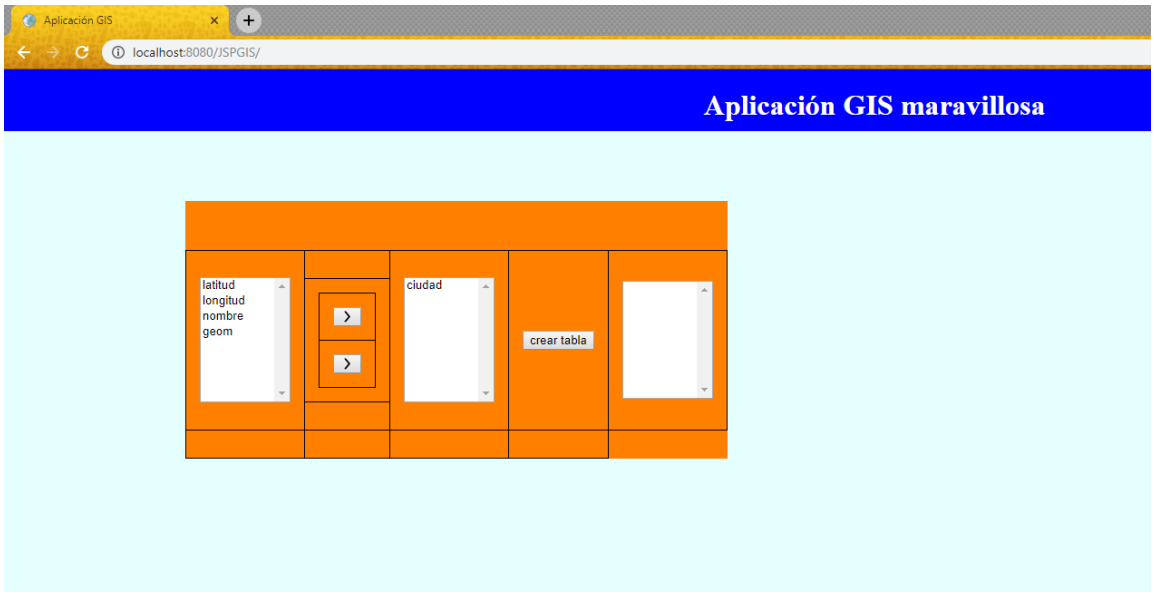


Figura 59: Nueva versión con tres divisiones diferenciadas.

Para esas diferenciaciones entre secciones en la interfaz visuales se ha hecho uso de div en html,

```
<body>
    <div class="cabecera">
        <h1>Aplicación GIS maravillosa</h1>
    </div>
    <div class="cuerpo">
        <br> <br> <br>
        <div>
        <table>
            .
            .
```

```
            .
        </table>
    </div>
```

A cada división se le ha puesto un nombre distinto, ya que no son iguales y debemos poder diferenciarlas para cambiar sus propiedades. A continuación, se muestra cómo se diferencian las divisiones en CSS para darle un estilo distinto a cada una.

- **Fondo de la página web.**

```
body {
        background: #e4fffd;
}
```

- **División hecha para la cabecera.**

```
div.cabecera {
        position: absolute;
        top: 0px;
        left: 0px;
        right: 0px;
        background-color: #0000ff;
        color: #ffffff;
        text-align: center;
        vertical-align: middle;
        height: 7%;
}
```

- **Cualquier texto del título h1.**

```
h1 {
        text-align: center;
        vertival-align: middle;
}
```

- **División hecha en la que se incluye la tabla.**

```
div.cuerpo {
        position: absolute;
        top: 15%;
        margin-right: 0px;
        margin-left: 200px;
        background-color: #ff8000;
}
```

Con este estilo se consigue una interfaz con elementos más suavizados, se cambia la manera de mostrar los datos y se deshecha la idea de diferenciar las relaciones 1-1 y 1-muchos. Simplemente se seleccionarán campos de la tabla origen que se incluirán en la nueva tabla.

Con esta primera versión no se le da opción al usuario de interacción salvo para mover elementos de tablas y pulsar el botón de crear tabla, pero con ella se comprueba que la

automatización de la creación de una nueva tabla a partir de otra, con sus respectivos triggers se consigue de manera satisfactoria.

En esta versión los botones están poco estilizados y con nombres provisionales. Es necesario escribir la funcionalidad en ellos para saber cuál es porque visualmente son poco intuitivos.

4.2.4. Cuarto Sprint

Se comienza el cuarto sprint con varios cambios con respecto al sprint anterior, se indica que el diseño de la interfaz visual está muy anticuado. Hay que darle un nuevo enfoque. Al cargar la tabla de la base de datos carga toda la información; se pide cambiar eso para que muestre únicamente el nombre de los campos, y así poder seleccionarlos y crear la nueva tabla con esos campos, por lo que ver los datos que hay en ellos no es necesario. Y además se decide hacer la aplicación web dividida en pasos, añadiendo (a) una interfaz para poner usuario y contraseña, (b) otra para elegir la base de datos, (c) otra para la tabla con la que trabajar, etc., realizando un paso por cada acción que se desea ejecutar, para que de esta manera resulte más sencillo al usuario.

Para dar una sensación de más fluidez, se hace una aplicación paso a paso, en la que se pide la información necesaria poco a poco; primero el tipo de BD y el nombre de la BD, usuario y contraseña, tabla sobre la que quieres trabajar, nombre de la nueva tabla, etc.

En la Figura 60, Figura 61 y Figura 62 se muestran algunas de las pantallas añadidas con un nuevo estilo, más simple y moderno.

Figura 60: Pantalla selección base de datos en una versión más moderna.

Figura 61: Pantalla selección campo geométrico en una versión más moderna.

Figura 62: Pantalla coordenadas en una versión más moderna.

Modificado esto, ya se le da la posibilidad al usuario de elegir la tabla origen y los campos de esta que va a utilizar, pero no funciona. Al iniciar la aplicación se muestra un error de que el array mencionado en el punto 4.2.2 está vacío. Esto se debe a que, en JavaScript, al iniciar la aplicación se ejecutan todos los scripts, por lo que el array realmente está vacío al inicio. No ha tenido oportunidad el usuario de introducir datos cuando ya ha habido una función en JavaScript que le ha pasado ese array a una clase Java en la que se ha intentado acceder a elementos del array, por lo que siempre dará error.

A continuación, se muestra el fragmento de código que incluye la función en la que se le pasa a la clase Java un array con los datos para crear la nueva tabla, pero que realmente está vacío.

```
<%!PostgreConexion pc = new PostgreConexion();%>
        function createNewTable() {
                <%try {
                        PostgreConexion();
                        pc.createTable(nuevaTablaJ);
                } catch (Exception e) {
                        //Error en algun momento.
                        out.println("Excepcion " + e);
                        e.printStackTrace();
                }%>
        }
```

Este error es debido a la **asincronía**, es el problema que surge dar la opción a elegir la tabla, los scripts se ejecutan todos al cargar la web por lo que, si simplemente escribes un parámetro del nombre de la tabla en un método, la aplicación fallará. Al cambiar el diseño a una aplicación por pasos, los datos no estarán hasta que el usuario los introduzco, y como los scripts se ejecutan todos al cargar la web y no se recargan los resultados, está ahí el problema.

La solución a este problema son los métodos asíncronos, un método no se ejecuta hasta que recibe una llamada, se hace una petición y devuelve un resultado, una vez llamada a una clase java que tiene el método necesario para la funcionalidad a la que se quiere dar uso. Este resultado se puede añadir a la parte de la web que más convenga, con el método getElementByID(), el cual aprovecharemos para llevar la respuesta de nuestro método asíncrono al elemento que necesitemos.

Para resolver este problema se ha optado por utilizar Ajax[23], Asynchronous JavaScript And XML[24] por sus siglas en inglés.

```
function <nombreFuncion>() {
        var v = document.vinform.t1.value;
```

[23] Async/await. (2019). Disponible en https://javascript.info/async-await
[24] ls.foundation, J. (2019). Ajax | jQuery API Documentation. Disponible en https://api.jquery.com/category/ajax/

```
        var url = "<nombreClase>.jsp?<nombreVariable>=" + v;

        if (window.XMLHttpRequest) {
                request = new XMLHttpRequest();
        } else if (window.ActiveXObject) {
                request = new ActiveXObject("Microsoft.XMLHTTP");
        }

        try {
                request.onreadystatechange = <nombFuncRespuesta>;
                request.open("GET", url, true);
                request.send();
        } catch (e) {
                alert("Unable to connect to server");
        }
}

function <nombFuncRespuesta>() {
        if (request.readyState == 4) {
                var val = request.responseText;
                document.getElementById('<id>').innerHTML = val;
        }
}
```

Ha sido el mayor problema encontrado, el que más ha retrasado el proyecto, los métodos asíncronos, en los que se consigue una asincronía cuando es dentro de un mismo método, pero no cuando la asincronía se necesita para llamar a otra clase Jsp que tiene código java y conecta con la BD, por lo que siempre ejecuta todo el código nada más compilar la aplicación web. Se busca información y consultas con profesores, se da con la respuesta y tras diversas pruebas se consigue que funcione. Las conexiones en la base de datos a partir de este momento se hacen al pulsar los botones correspondientes y no al cargar el programa.

Por otra parte, se añade una funcionalidad que consiste en, dados dos campos distintos en la tabla inicial (latitud y longitud), unificarlos en un único campo georreferenciado de coordenadas. El siguiente script consigue transformar dos campos sin información georreferenciada en un campo de tipo punto, un punto geográfico ubicado según el sistema de referencia World Geodetic System 1984 (WGS 84), el sistema de referencia más usado hoy en día.

```
UPDATE <nombreTabla> SET <nombreCampo> =
ST_SetSRID(ST_MakePoint(<campo1>, <campo2>), 4326);
```

Durante este sprint, y una vez solucionados los problemas con la asincronía, se procede a parametrizar el método de conexión, al que se le pasa una cadena de texto que se convierte en array y, al ser el orden de inserción siempre el mismo, se establece la posición que corresponde a cada elemento del array en el lugar que corresponde

```
public static Connection getConexion(String prueba) {
        Connection con = null;
        String[] elementos = prueba.split(",");

        try {
           // Driver
           Class.forName("org.postgresql.Driver");

           // localhost/NombreBD
           String url = "jdbc:postgresql://localhost:5432/" +
elementos[1];
           String usuario = elementos[2];
           String pass = elementos[3];
           con = DriverManager.getConnection(url, usuario, pass);

        } catch (ClassNotFoundException e) {
           System.out.println("Error al cargar el driver");
           e.printStackTrace();
        } catch (SQLException e) {
           System.out.println("Error con la BD");
           e.printStackTrace();
        }
        return con;
}
```

Durante este sprint se parametriza todo, incluidos los triggers, que quedarían tal y como se muestra a continuación:

```
String sqlDelete = "CREATE OR REPLACE FUNCTION delete_trigger" +
elementos[5] + "() RETURNS TRIGGER AS $$ " + "BEGIN " + "DELETE FROM
" + elementos[5] + " " + "WHERE " + cadenaSupletoria+ "; " + "RETURN
NULL; " + "END; " + "$$ LANGUAGE 'plpgsql'; " + "DROP TRIGGER IF
EXISTS eliminar_eliminados" + elementos[5] + " " + "ON
"+elementos[5]+";"


String sqlInsert = "CREATE OR REPLACE FUNCTION insert_trigger" +
elementos[5] + "() RETURNS TRIGGER AS $$ " + "BEGIN " + "INSERT INTO
" + elementos[5] + "(" + cadenaSelect + ") " + "VALUES (" +
cadenaInsert + "); " + "RETURN NULL; " + "END; " + "$$ LANGUAGE
'plpgsql'; "


String sqlUpdate = "CREATE OR REPLACE FUNCTION update_trigger" +
elementos[5] + "() RETURNS TRIGGER AS $$ " + "BEGIN " + "UPDATE " +
elementos[5] + " " + "SET " + cadenaUpdate + " " + "WHERE " +
cadenaSupletoria + "; " + "RETURN NULL; " + "END; " + "$$ LANGUAGE
'plpgsql'; "
```

Una vez tenemos los scripts en un campo de texto, debemos ejecutarlos en la clase Java. Para ello, debemos crear una conexión con la base de datos y ejecutarlos.

```
Connection con = getConexion(nombres);
Statement s = con.createStatement();

s.executeUpdate(sqlDelete);
s.executeUpdate(sqlInsert);
s.executeUpdate(sqlUpdate);
```

4.2.5. Quinto Sprint

El diseño visual de la web, pese a haber mejorado enormemente, no termina de convencer, y durante la reunión mensual se piden varios cambios, como, por ejemplo, el tamaño de los botones, que todos sean iguales. Se pide simplificarlo todo, incluso los pasos existentes simplificarlos más para facilitar la comprensión y el uso de la aplicación web.

En la Figura 63 se pueden observar los cambios, con botones homogéneos y con un menú principal añadido, que nos permite la acción que queremos hacer. Para este proyecto se dejará la opción de fusionar tablas sin implementar, pero todo preparado para que en un futuro se implemente.

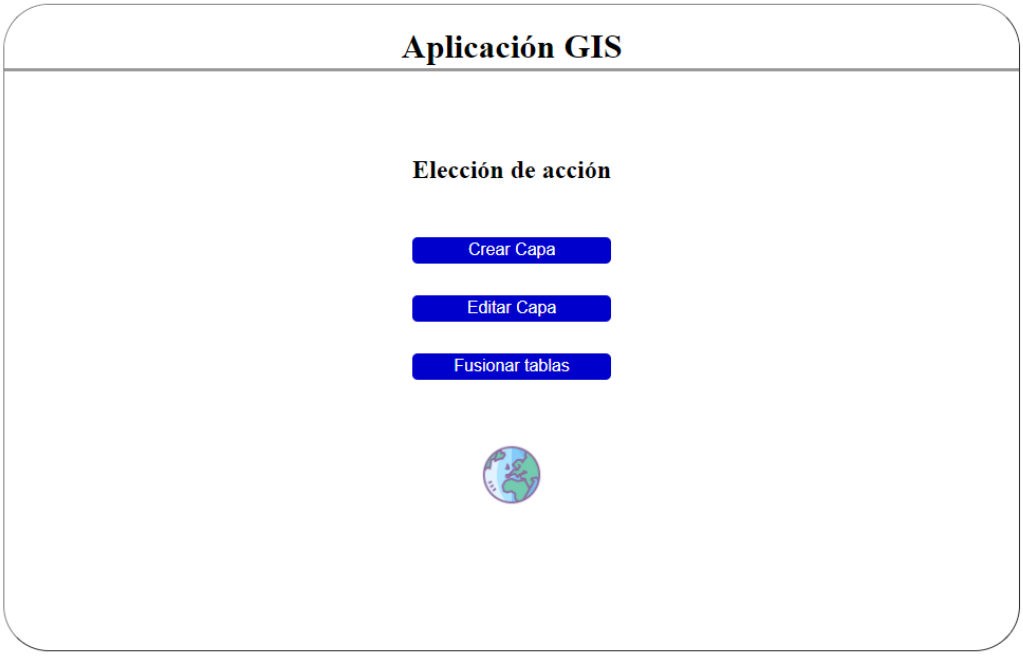


Figura 63: Menú inicial aplicación web.

En la Figura 64 tenemos el menú que aparece tras pulsar el botón de crear capa, que nos permite elegir el tipo de capa a crear, entre capa de puntos, líneas y polígonos

Figura 64: Menú de creación de capa

Se añade un botón que permite seleccionar distintos tipos de bases de datos. Para este proyecto únicamente se utilizará PostgreSQL, pero se deja implementada para futuras ampliaciones de la aplicación. También se ha cambiado el color de los botones para pasar a la siguiente acción, al igual que se añade otro botón para volver a la pantalla anterior, tal y como se muestra en la Figura 65.

Figura 65: Seleccionar base de datos

Llegado a este punto, la aplicación web tiene más de 10 pantallas distintas por lo que, para evitar constantes recargas, se opta por cambiar el modo de navegar en la web. Siempre se muestra la misma página, y lo que se hace con los botones es mostrar y ocultar los elementos convenientes, parte de los elementos y su estado al inicio de la aplicación.

```
<div id="menu" class="cuerpo" style="visibility: visible">
       <%@include file="zero.jsp"%>
</div>

<div id="capSelection" class="cuerpo" style="visibility: hidden">
       <%@include file="capSelection.jsp"%>
</div>

<div id="capEdit" style="visibility: hidden" align="center">
       <%@include file="capEdit.jsp"%>
</div>

<div id="selectBD" class="cuerpo" style="visibility: hidden">
       <%@include file="firstStep.jsp"%>
</div>

<div id="authentication" class="cuerpo" style="visibility: hidden">
       <%@include file="secondStep.jsp"%>
</div>

<div id="selectTable" class="cuerpo" style="visibility: hidden">
       <%@include file="thirdStep.jsp"%>
</div>

<div id="tableName" class="cuerpo" style="visibility: hidden">
       <%@include file="fourthStep.jsp"%>
</div>

<div id="geometry" class="cuerpo" style="visibility: hidden">
       <%@include file="fifthStep.jsp"%>
</div>
```

Esto supone muchos elementos para tener en cuenta mientras se navega, por lo que se crean scripts únicamente pensados en la navegabilidad, como el que se ejecuta al pulsar el botón de menú, el que tiene la imagen de la tierra. Este debe ocultar todos los elementos existentes menos el menú principal, ya que hacer un script distinto dependiendo de la pantalla desde la que se pulsa el botón sería una gran pérdida de tiempo y recursos, por lo que se ha generado el siguiente script:

```
function mostrarMenu() {
     document.getElementById('menu').style.visibility = 'visible';
     document.getElementById('capEdit').style.visibility = 'hidden';
     document.getElementById('capSelection').style.visibility ='hidden';
```

```
        document.getElementById('selectBD').style.visibility = 'hidden';
        document.getElementById('authentication').style.visibility = 'hidden';
        document.getElementById('selectTable').style.visibility = 'hidden';
        document.getElementById('tableName').style.visibility = 'hidden';
        document.getElementById('geometry').style.visibility = 'hidden';
        document.getElementById('coordinates').style.visibility = 'hidden';
        document.getElementById('createTable').style.visibility = 'hidden';
        document.getElementById('newTable').style.visibility = 'hidden';
        document.getElementById('seeMap').style.visibility = 'hidden';
        document.getElementById('errorBD').style.visibility = 'hidden';
        document.getElementById('errorInit').style.visibility = 'hidden';
        document.getElementById('errorTable').style.visibility = 'hidden';
        document.getElementById('polData').style.visibility = 'hidden';
        document.getElementById('polInit').style.visibility = 'hidden';
}
```

Una vez hechos los scripts necesarios para la navegabilidad, se prueban para validar su funcionalidad. La navegabilidad funciona correctamente.

4.2.6. Sexto Sprint

El diseño visual nuevo es más convincente, pero se pide cambiar el color y diseño de los botones porque resaltan demasiado y deben estar más integrados.

Hasta el momento se podían añadir dos campos geográficos, uno de datos tipo punto si tenías coordenadas geográficas en dos campos, latitud y longitud y, aparte, añadir otro campo a elegir entre datos tipo punto, polilíneas o polígonos. Es decir, se elegía el tipo de capa a crear y en medio del proceso, independientemente del tipo de capa elegido, había un paso en el que podías transformar dos campos de la tabla en un punto. Esto decide cambiarse, podría resultar demasiado confuso.

Se decide únicamente añadir un campo georreferenciado a la tabla. Si se elige crear una capa de tipo punto, se dispondrá de la posibilidad de convertir dos campos en una coordenada geográfica; en otra elección esa posibilidad no está disponible.

Durante la reunión previa al sexto sprint, en una de las pruebas los datos se introducen mal y aparece un error. No existe la base de datos, pero visualmente no se informa al usuario. Se pide añadir mensajes de error cuando esto suceda para que de esta manera pueda el usuario saber cuál ha sido su error y tener la posibilidad de modificarlo.

Comenzamos el sprint modificando el aspecto de los botones, para que visualmente sean más modernos, que tengan un poco de sombra para dar una sensación de relieve, cambiar la fuente del texto, pequeños detalles que se agradecen en el aspecto visual, como podemos apreciar en la Figura 66.

Figura 66: Nuevo aspecto de los botones, más moderno.

A continuación, se muestra el estilo de los botones, todos tienen el mismo a excepción del color de fondo por lo que, para diferenciarlos, a cada estilo de input se le ha llamado de manera distinta, los cinco botones existentes son *selection, previous, next, jump y menu*.

```
input.selection {
        display: inline-block;
        float: middle;
        background-color: #0000CC;
        border: none;
        border-radius: 5px 5px 5px 5px;
        font-size: 16px;
        width: 100%;
        min-width: 90px;
        min-height: 25px;
        color: #FFFFFF;
        font-family: 'Open Sans', sans-serif;
        border-width: 0;
        outline: none;
        border-radius: 2px;
        box-shadow: 0 1px 4px rgba(0, 0, 0, .6);
        background-color: #3065AC;
        color: #ecf0f1;
        transition: background-color .3s;
}
```

En la Figura 67 podemos encontrar la creación de un punto a partir de dos campos que se encuentran en la tabla origen. Esta pantalla estará disponible si seleccionamos crear una capa de tipo punto,

Figura 67: Ejemplo de creación de un punto.

Podemos ver en la Figura 68 la inicialización de una capa de tipo polígono, que constará de 4 vértices.

Figura 68: Inicialización de la capa polígonos.

Los valores de los 4 puntos se guardan en un array que será enviado a la clase java para ejecutar el siguiente script SQL:

```
String actCoordenadas = "UPDATE " + elementos[5] + " SET geom_p =
ST_GeomFromText('POLYGON((" + elementos[6] + ", " + elementos[7] + ",
" + elementos[8] + ", " + elementos[9] + ", "    +    elementos[6]    +
"))',4326);";
s.executeUpdate(actCoordenadas);
```

En la Figura 69, mostrada a continuación, se muestra uno de los mensajes de error que informan al usuario que ha de modificar algo.

Figura 69: Mensaje de error.

La función en JavaScript para mostrar ese mensaje es la presentada a continuación:

```
function addBD() {
     var option = document.createElement("option");
     var x = tableBD.selectedIndex;
     if (x == 0) {
          document.getElementById('errorBD').style.visibility =
'visible';
     } else {
          var sel = tableBD.options[tableBD.selectedIdex];
          option.text = tableBD.options.item(x).value;
          nuevaTabla.splice(0, 0, tableBD.options.item(x).value);
          var z = bdName.value;
          nuevaTabla.splice(1, 0, z);
```

```
            document.getElementById('authentication').style.visibility
= 'visible';
            document.getElementById('selectBD').style.visibility =
'hidden';
            document.getElementById('errorBD').style.visibility =
'hidden';
        }
}
```

4.2.7. Séptimo Sprint

En este sprint, lo que se hace es, dada una tabla con un campo georreferenciado, seleccionarla y mostrar los datos en un mapa, viendo también los datos del campo georreferenciado en un espacio bajo el mapa, de tal manera que si seleccionas uno de los puntos en el mapa, se marcará en el espacio donde se muestra, y si lo seleccionas en el espacio donde están todos los puntos de la tabla, se te señalará en el mapa cuál de ellos es.

En la Figura 70 se puede observar el mapa en la web mostrando la capa de información seleccionada, puntos blancos con borde azul.

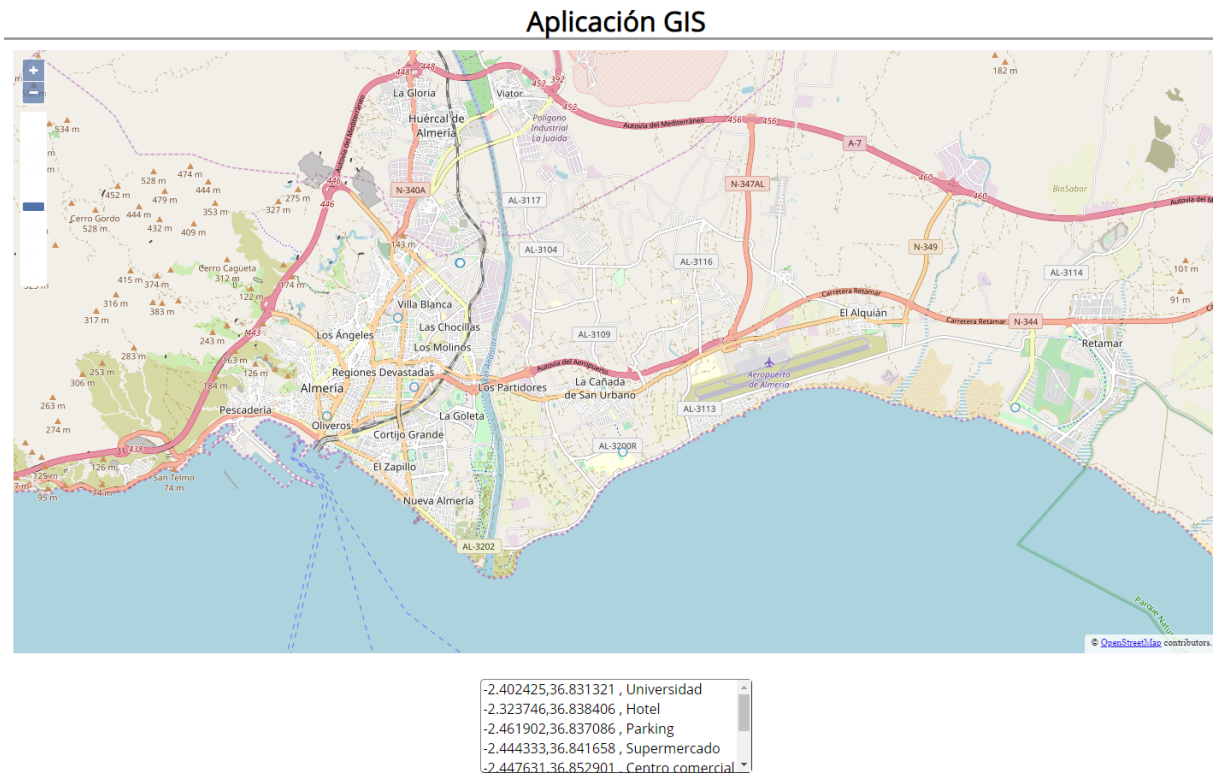


Figura 70: Mapa de OpenLayers en la aplicación web.

Para poder trabajar con OpenLayers ha de referenciarse los estilos y el repositorio del cual obtiene la información.

```
<script src="https://openlayers.org/en/v5.3.0/build/ol.js"
        type="text/javascript">

<link rel="stylesheet"
        href="https://openlayers.org/en/v5.3.0/css/ol.css"
type="text/css">
```

A continuación, se mostrará la inicialización en JavaScript de un mapa de OpenLayers, es una variable tipo OpenLayers Map, en la que hay que inicializar la capa, el fondo que se mostrará. En este caso es un mapa callejero y el objetivo, la división en html en la que se colocará el mapa. Se centra la imagen, que por defecto será el punto 0, 0.

El punto se modificará al cargar cada capa para centrar la vista en el primero de los datos

```
var map = new ol.Map({

    layers : [ new ol.layer.Tile({
        source : new ol.source.OSM()
    }) ],
    overlays: [overlay],
    target: 'showmap',
    view: new ol.View({
        center: [0, 0],
        zoom: 2
    })
});
```

Una vez tenemos inicializado el mapa, hemos de añadir los puntos de la capa. Para ello tenemos una función que se encarga únicamente de recorrer el resultado de obtener los datos de la capa e ir añadiendo elemento a elemento a una capa[25] de OpenLayers. Para que no se sobrescriban los datos, se guardan en arrays los nombres de las variables, tantas como elementos tenga la capa, para que cada uno tenga un nombre distinto en sus variables correspondientes.

```
function addCap(){
    .
    .
    .
    for(name of geoArr){
        nuevoArr.push(name);
    }
    .
    .
    .
```

[25]Overlay. (2019). Disponible en https://openlayers.org/en/latest/examples/overlay.html

```
    var marker = [];
    var vectorSource = [];
    var markerVectorLayer = []
    for (var m = 5; m < cantidad; m++) {
        marker = "marker"+m;
        vectorSource = "vectorSource"+m;
        markerVectorLayer = "markerVectorLayer"+m;
    }
    for (var i = 5; i < cantidad; i++) {
        var pruebon = nuevoArr[i].split(",");
        .
        .
        .
        this["marker"+i] = new ol.Feature({type: 'click', desc:
nuevoArr[i], geometry: new
ol.geom.Point(ol.proj.fromLonLat([lati,longi])),});
        map.addControl(new ol.control.ZoomSlider());

        this["vectorSource"+i] = new ol.source.Vector({features:
[this["marker"+i]]});

        this["markerVectorLayer"+i] = new ol.layer.Vector({source:
this["vectorSource"+i],});

        map.addLayer(this["markerVectorLayer"+i]);
    }
}
```

Se incluye en el mapa la funcionalidad en la que pinchando en el punto que se ve en el mapa, muestra la información de este y marca sus datos correspondientes en la tabla que se muestra bajo el mapa. La acción inversa también está contemplada, clicando en la tabla se abre un cuadro de diálogo que muestra la información de ese elemento y dónde está localizado, tal y como se puede ver en la Figura 71.

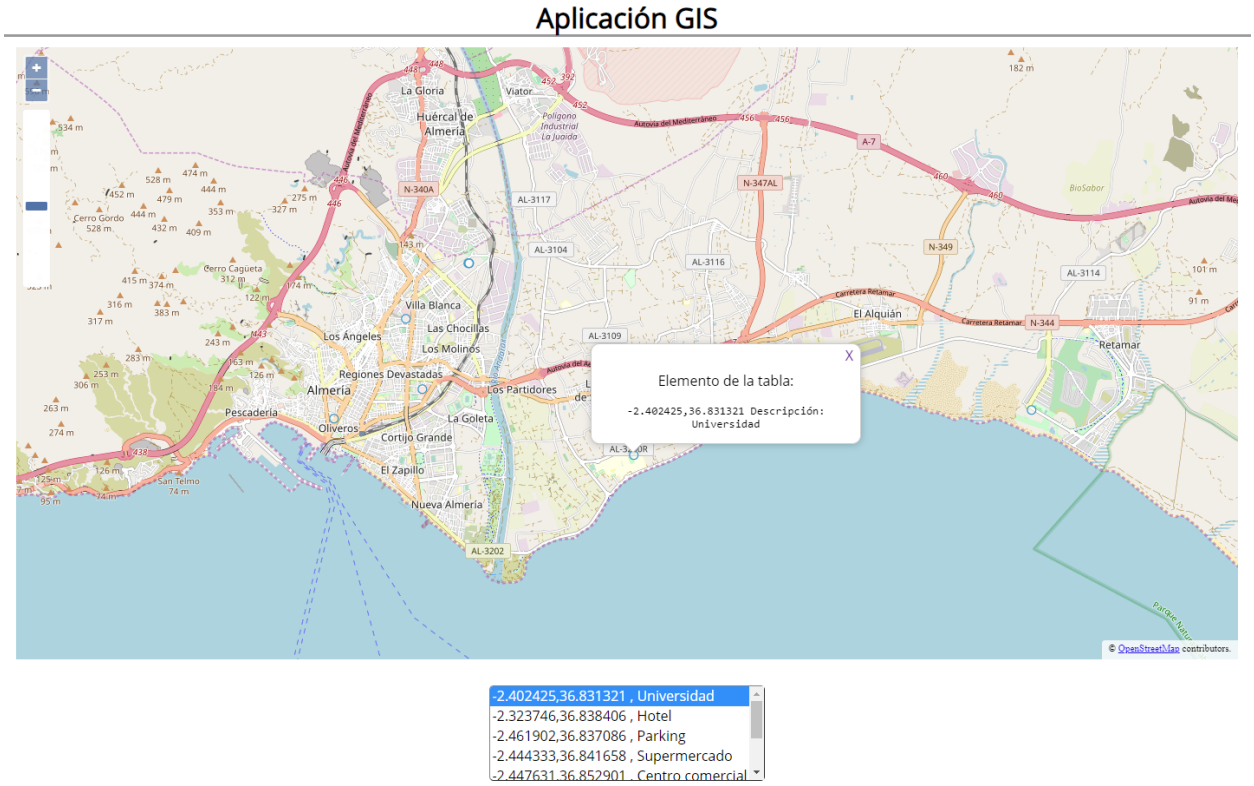


Figura 71: Cuadro de diálogo mostrando información asociada a ese elemento.

4.2.8. Octavo Sprint

Se comienza el octavo y último sprint en el que se procederá a subir la capa creada a GeoServer y desplegar la aplicación web creada en la máquina virtual proporcionada.

Durante la implementación de la publicación de la capa surge el problema de CORS policy, Figura 72, por lo que será imprescindible desplegar la aplicación en la máquina virtual para probar la conexión entre la aplicación web creada y GeoServer.

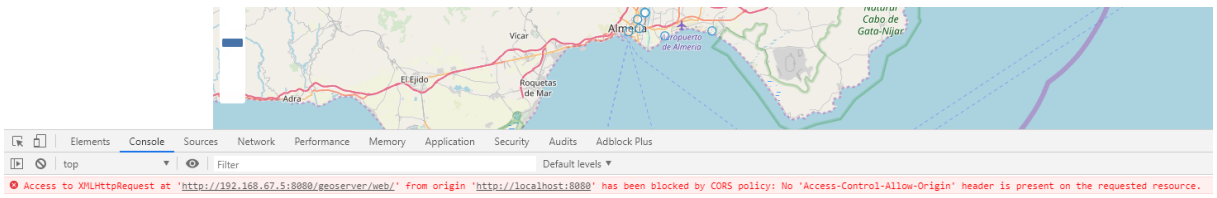


Figura 72: Error CORS policy.

Para el despliegue de la aplicación en la máquina virtual, necesitamos el archivo .war que se genera tras exportar el proyecto.

Se añade este archivo a la máquina virtual, para ello se ha usado GitHub, se ha subido el archivo a un repositorio que se ha descargado en la máquina virtual, y una vez descargado se ha movido el archivo al directorio desde el que Tomcat lanza las aplicaciones.

Para mover el archivo utilizamos el siguiente comando
`sudo mv JSPGIS.war /var/lib/tomcat8/webapps/`

Se reinicia Tomcat y de esta manera ya está disponible la aplicación en a la que podemos acceder desde un navegador por la IP de la máquina virtual.

Con la aplicación y GeoServer en la máquina virtual, se procede a subir la capa creada con los datos introducidos por el usuario a GeoServer. El proceso para subir la capa consiste en, dado un workspace predeterminado de GeoServer, añadirle una capa directamente desde la base de datos. Se selecciona la tabla que se ha creado para y mediante un json enviarse con un POST a la siguiente dirección:
http://192.168.67.5:8080/geoserver/rest/workspaces/tfg/datastores/pruebatfg/featuretypes.json

En el json que se envía a esa dirección hay que parametrizar el nombre de la base de datos, el nombre de la tabla y los campos y datos que hay en ella para poder localizar la capa en la sección de previsualización de capas de GeoServer Figura73.

Spearfish elevation	sf:sfdem	OpenLayers KML	Seleccionar una
Spearfish streams	sf:streams	OpenLayers KML GML	Seleccionar una
despachosgeometricos	tfg:despachosgeometricos	OpenLayers KML GML	Seleccionar una
puntoscargaautomatizado	tfg:puntoscargaautomatizado	OpenLayers KML GML	Seleccionar una
Spearfish	spearfish	OpenLayers KML	Seleccionar una
Tasmania	tasmania	OpenLayers KML	Seleccionar una
TIGER New York	tiger-ny	OpenLayers KML	Seleccionar una

Figura 73: Capas despachosgeometricos y puntocargaautomatizado creadas por el usuario.

Pulsando en OpenLayers KML se abrirá una nueva ventana en la que se mostrarán los elementos georreferenciados que se han publicado, sin ningún fondo, únicamente los elementos, tal como se ve en la Figura 74.

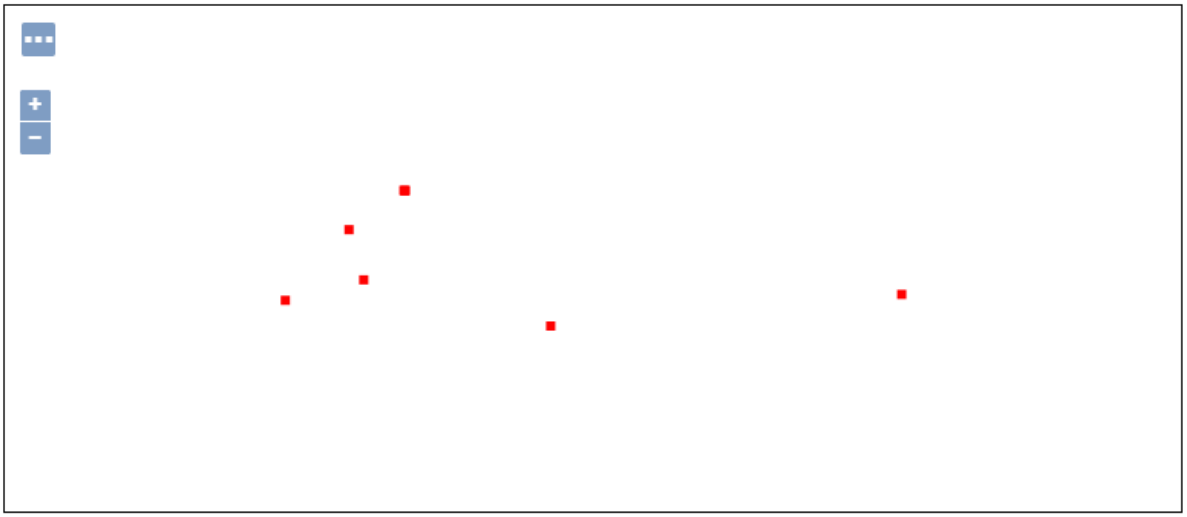


Figura 74: Capa puntocargaautomatizado mostrada.

Capítulo 5
Escenario de ejemplo

5. Escenario de ejemplo

En este capítulo se verá el funcionamiento de la aplicación web mostrando en un escenario de ejemplo la principal funcionalidad de esta aplicación web: crear una capa con información geográfica y mostrar esa capa creada.

5.1. Creación capa polígonos

En primer lugar, accedemos a la aplicación y la pantalla de inicio nos dará la opción de crear capas, mostrar una o fusionar tablas (no implementada), Figura 75.

Figura 75: Menú principal.

Tras seleccionar crear capa, aparecerá otro menú en el que se debe seleccionar el tipo de capa que se va a crear, en este caso crearemos una capa de polígonos. Figura 76.

Figura 76: Selección de capa a crear.

El siguiente paso es introducir los datos de la base de datos en la cual se va a trabajar. Como se ve en la Figura 77, hay que introducir la dirección completa, ya que no solo trabaja con bases de datos locales.

Figura 77: Tipo y dirección de la base de datos.

Una vez introducida la dirección de la base de datos, es el turno de introducir las credenciales, Figura 78.

Figura 78: Introducción usuario y contraseña de la base de datos.

La aplicación comprobará si la conexión es correcta, si no lo es aparecerá un mensaje de error indicándolo, si es correcta mostrará la siguiente pantalla, en la que se ha de introducir el nombre de la tabla sobre la que se va a trabajar. Figura 79.

Figura 79: Introducción de latabla sobre la que se va a trabajar.

Se vuelve a comprobar que la tabla existe; si no existe, se mostrará un mensaje indicando al usuario que esa tabla no existe para que cambie el nombre a uno de una tabla existente. Si existe, se continuará mostrando la siguiente pantalla, en la que se debe introducir el nombre de la tabla nueva que se va a crear, preferiblemente en minúsculas porque PostgreSQL no siempre funciona bien con letras mayúsculas. Figura 80.

Figura 80: Introducción nombre de la nueva tabla.

Tras introducir el nombre y pulsar el botón “Siguiente”, mostrará la pantalla en la que se inicializará el polígono. En esta pantalla han de introducirse las coordenadas de 4 puntos, los 4 vértices del polígono que se inicializará. A la hora de inicializar el polígono hay que tener en cuenta que las coordenadas del punto 1 se unirán a las del punto 2, estas a las del punto 3, el punto 4 al punto 4 y las coordenadas del punto 4 cerrarán el polígono uniéndose a las del punto 1. Figura 81.

Figura 81: Inicialización de la capa de elementos tipo polígono.

Una vez introducidos los puntos que definirán el polígono, se mostrará una pantalla en la que podremos elegir qué campos de la tabla seleccionada en el tercer paso de este escenario de ejemplo añadiremos a la tabla que va a ser creada. Figura 82.

Figura 82: Seleccionar campos para la nueva tabla.

Cuando se hayan seleccionado todos los campos que se necesiten, se pula el botón de crear tabla, el cual ejecutará la función que crea la tabla y avanza a la siguiente pantalla, en la que se muestra los campos que contiene la nueva tabla. Figura 83.

Figura 83: Campos que contiene la nueva tabla.

La tabla con información geográfica ya está creada, es el turno de comprobar esa información en un mapa. El siguiente paso en la aplicación es poder ver en un mapa los datos creados, y bajo este mapa una tabla en la que se muestran los elementos que hay en la base de datos. Figura 84.

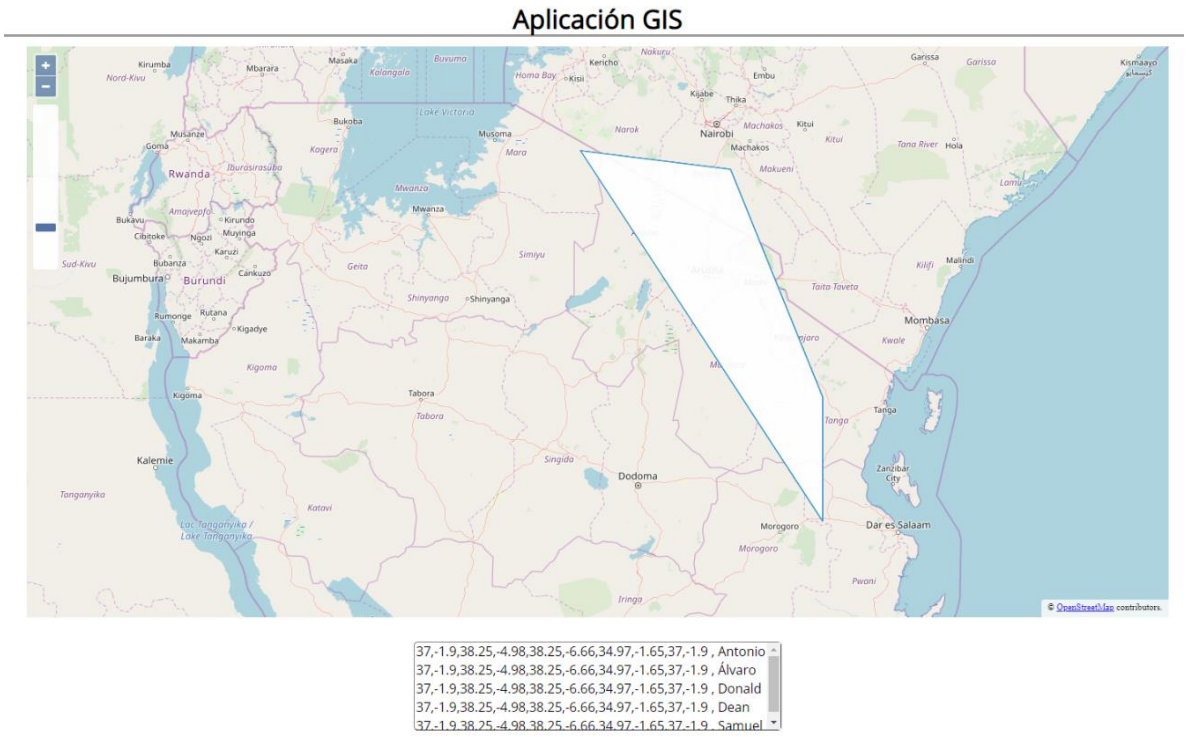


Figura 84: Mapa con la información geográfica creada.

En esta pantalla, pulsando sobre los datos en la tabla, se obtendrá más información sobre el elemento que se ha pinchado, tal y como se muestra en la Figura 85.

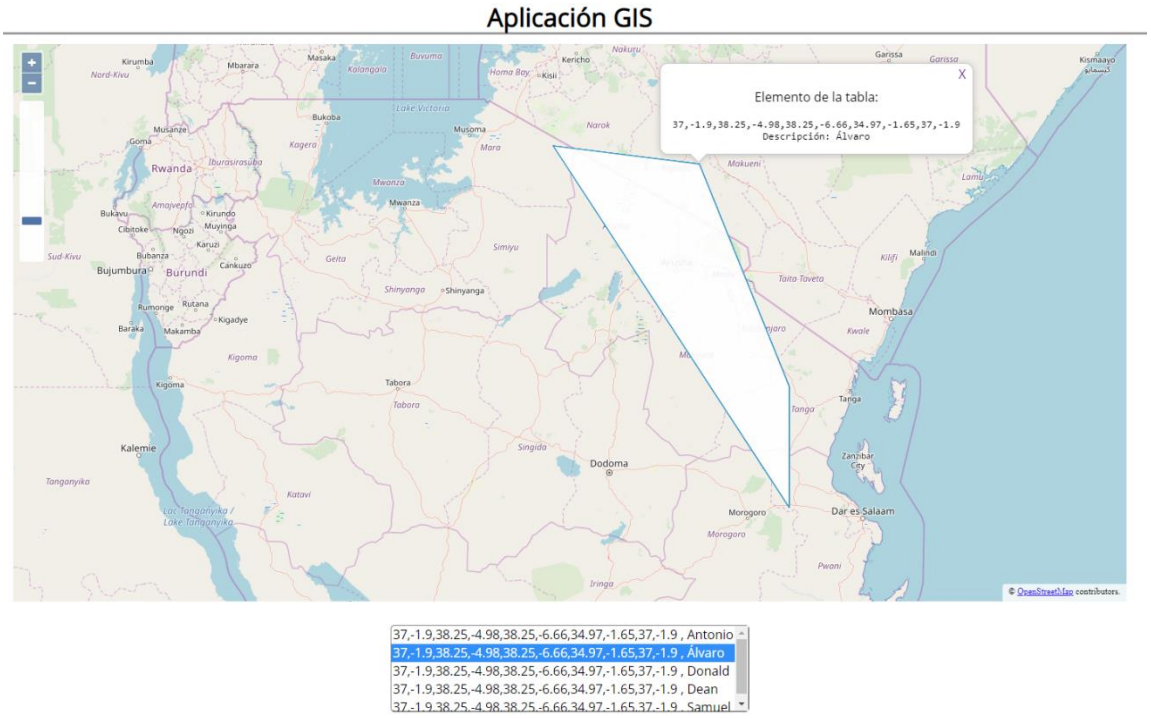


Figura 85: Información en el mapa sobre el dato seleccionado.

Capítulo 6
Conclusiones

6. Conclusiones

En este capítulo se presentan las conclusiones sobre el desarrollo de la aplicación web y futuras líneas de trabajo asociadas a aumentar su funcionalidad. El desarrollo de este proyecto de final de grado se ha prolongado durante un año, desde el planteamiento de la idea hasta las pruebas finales, incluyendo todo el proceso de documentación, estudio, investigación y desarrollo de la aplicación. Es un plazo razonable teniendo en cuenta que al comienzo de este proyecto no se tenía conocimiento alguno sobre un SIG, información georreferenciada, mostrar información en un mapa o su almacenamiento en una base de datos.

La implementación de la automatización de creación de capas georreferenciadas a partir de datos no georreferenciados supone la simplificación del proceso para usuarios que no estén familiarizados con el tema, al igual que para los usuarios que sí están familiarizados, pero prefieren agilizar el proceso mediante el uso de esta aplicación.

Al ser una aplicación web desplegada en Tomcat, nos da la libertad de ejecutarla en cualquier sistema operativo, tanto si queremos ejecutarlo en local como si queremos dedicar una máquina virtual en el que ejecutar la aplicación y que de esta manera muchos más usuarios tengan acceso al uso de las aplicaciones.

Otra virtud de esta aplicación es disponer una interfaz sencilla e intuitiva, siempre se sabe en qué pantalla se está y la acción que se debe hacer en ella, y se pueden deshacer las acciones y moverse entre pantallas con total facilidad.

Como trabajos futuros, se podría ampliar el funcionamiento, habilitando e implementando el apartado de fusionar tablas para, dada la circunstancia en que tengas dos tablas que compartan uno o varios campos, crear una nueva tabla en esa base de datos que contenga la información de ambas tablas. Podría ser otra manera de añadir información georreferenciada a la ya existente, si una de las tablas no tiene un campo georreferenciado, pero contiene más información descriptiva, y la otra contiene un campo georreferenciado pero puede que solo contenga uno o dos campos más con información descriptiva. Una última propuesta de ampliación y para la que también está prepara esta aplicación es la posibilidad de conectarse a distintos tipos de bases de datos SQL, y pasar toda esa información a la base de datos PostgreSQL y trabajar en esta última base de datos.

El desarrollo de este proyecto de fin de grado ha sido muy exigente. Programar aplicaciones web, por sencillas que parezcan en un principio, siempre suponen un gran esfuerzo, tiempo y dedicación, por ello se han dejado bastantes funcionalidades preparadas, pero no terminadas y funcionando. La parametrización para la creación tanto de tablas, como de *triggers*, como de campos geométricos ha supuesto gran parte del esfuerzo, pero el gran contratiempo, el problema que más tiempo y esfuerzo a dedicado ha sido la asincronía entre la web y la base de datos.

En conclusión, podríamos decir que el desarrollo de este proyecto ha servido para adentrarse en el campo del SIG, poner a prueba las habilidades de desarrollo web estilizando la aplicación a una visualización siempre lo más moderna posible, aumentar los conocimientos de bases de datos y aprender sobre conexiones de este tipo de aplicaciones web y, por último, trabajar con mapas y capas de información georreferenciada para mostrar todo esto en una web.

Además, se ha cumplido con el objetivo principal de crear una aplicación web en la que se creen tablas en una base de datos con campos georreferenciados, estos campos siempre a elección del usuario. También se ha cumplido el otro objetivo principal de poder mostrar el campo georreferenciado de una tabla en un mapa. Siempre dándole la máxima posibilidad de elección al usuario.

.

Capítulo 7
Referencias

7. Referencias

- [1] Una Ligera Introducción a GIS. (2019). Disponible en https://docs.qgis.org/2.8/es/docs/gentle_gis_introduction/index.html ... **pág. 6**
- [2] Almería. (2019). Disponible en https://www.google.com/maps/place/Almer%C3%ADa/@36.8274171,-2.4388171,273m/data=!3m1!1e3!4m5!3m4!1s0xd7a9e00ecccf2c1:0x8d9da01f8ebc485e!8m2!3d36.834047!4d-2.4637136 . **pág. 7**
- [3] Datos Vectoriales. (2019). Disponible en https://docs.qgis.org/2.8/es/docs/gentle_gis_introduction/vector_data.html **pág. 8**
- [4] Datos Raster. (2019). Disponible en https://docs.qgis.org/2.8/es/docs/gentle_gis_introduction/raster_data.html **pág. 10**
- [5] Mapa mundial temperatura (2019). Disponible en https://rapidainformacion.files.wordpress.com/2015/03/mapa-mundial-temperatura.jpg ... **pág. 11**
- [6] Datos Raster. (2019). Disponible en https://docs.qgis.org/2.8/es/docs/gentle_gis_introduction/raster_data.html **pág. 12**
- [7] PostGIS manual. (2019) Disponible en http://www.dcc.fc.up.pt/~michel/TABD/postgis.pdf **pág. 13**
- [8] Developers, Postgis–Spatial and Geographic Objects PostgreSQL- (2019). Disponible en https://postgis.net/ **pág. 13**
- [9] Openlayers.org. (2019) OpenLayers-Wlcome. Disponible en https://openlayers.org **pág. 15**
- [10] Openlayers.org. (2019) Select Features. Disponible en https://openlayers.org/en/latest/examples/select-features.html **pág. 15**
- [11] GeoServer. (2019). Disponible en http://geoserver.org/ **pág. 16**
- [12] López, B. (2019). GeoServicios y cómo administrarlos en GeoServer. Disponible en https://www.cursosgis.com/que-son-los-geoservicios-y-como-administrarlos-en-geoserver-opengeo-suite/ **pág. 16**
- [13] Foundation, E. (2019). Eclipse Downloads | The Eclipse Foundation. Disponible en https://www.eclipse.org/downloads/ **pág. 38**
- [14] PostgreSQL: Windows installers. (2019). Disponible en https://www.postgresql.org/download/windows/ **pág. 38**

- [15] Developers, P. (2019). PostGIS — Windows Downloads. Disponible en https://postgis.net/windows_downloads/ ..... **pág. 38**
- [16] Apache Tomcat® - Apache Tomcat 8 Software Downloads. (2019). Disponible en https://tomcat.apache.org/download-80.cgi ..... **pág. 38**
- [17]GeoServer Installation On Ubuntu Server 18.04 LTS. (2019). Disponible en https://gist.github.com/falu/1c077f67f009ac487305983c78c767b7 ..... **pág. 52**
- [18] PostgreSQL: Documentation: 10: CREATE TRIGGER. (2019). Disponible en https://www.postgresql.org/docs/10/sql-createtrigger.html ..... **pág. 59**
- [19] JavaScript Tutorial. (2019). Disponible en https://www.w3schools.com/js ..... **pág. 62**
- [20] Java Platform Standard Edition 8 Doc, *Docs.oracle.com*, (2019). Disponible en https://docs.oracle.com/javase/8/docs/ ..... **pág. 62**
- [21] Jsp, Mendoza, L. (2019). Accessing java variable from javascript jsp. Disponible en https://stackoverflow.com/questions/18990700/accessing-java-variable-from-javascript-on-same-jsp ..... **pág. 62**
- [22]CSS Reference, W3schools.com. (2019). Disponible en https://www.w3schools.com/cssref/ ..... **pág. 62**
- [23] Async/await. (2019). Disponible en https://javascript.info/async-await ..... **pág. 69**
- [24] Is.foundation, J. (2019). Ajax | jQuery API Documentation. Disponible en https://api.jquery.com/category/ajax/ ..... **pág. 69**
- [25] Overlay, OpenLayers. (2019). Disponible en https://openlayers.org/en/latest/examples/overlay.html .....**pág. 80**

UNIVERSIDAD DE ALMERÍA

Resumen/Abstract

En este trabajo de fin de grado se desarrollará e implementará una herramienta web con un conjunto de utilidades orientadas a crear y visualizar capas con información georreferenciada.

Con esta herramienta web, se pretende automatizar la construcción de capas con información georreferenciada a partir de bases de datos que no disponen de dicha información. De esta forma, se pretende agilizar todo el proceso de construcción, incluida la creación de disparadores o *triggers* para que la nueva tabla esté siempre actualizada y acorde a la información que existe en la original.

GRADO EN INGENIERÍA INFORMÁTICA, 2018/2019