

E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Aplicación Web-Mapping para la gestión de Parques y Jardines



Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor: Iñigo Rezusta Iglesias

Tutor: César Arriaga Egüés

Pamplona, 10 de Junio 2019

Agradecimientos:

*A mis padres y a mi hermano por
su apoyo durante todos estos años.*

A mis familiares y amigos.

Y por supuesto a mi tutor, César.

2. ÍNDICE

2. ÍNDICE DE FIGURAS	7
3. PALABRAS CLAVE.....	9
3.1. Keywords.....	10
4. RESUMEN	11
4.1. Abstract.....	12
5. INTRODUCCIÓN.....	13
5.1. Objetivos de la aplicación	13
5.2. Resultados del aprendizaje	14
6. ANÁLISIS DE REQUISITOS	15
6.1. Requisitos funcionales.	15
6.2. Requisitos no funcionales	17
6.2.1. Requisitos de Carga y Tiempo	17
6.2.2. Requisitos Software	17
6.2.3. Requisitos Hardware	19
7. DISEÑO	20
7.1. Modelo de datos.....	20
7.1.1. Diagrama Entidad-Relación	20
7.1.2. Modelo relacional.....	22
7.2. Arquitectura de la solución.....	27
7.2.1. Arquitectura de mapas	27
7.2.2. Arquitectura de texto	27
7.2.3. Componentes	28
7.3. Casos de uso.....	29
7.4. Diseño Interfaz.....	31
8. IMPLEMENTACIÓN	38
8.1. Base de datos.....	38
8.2. Conexión, consulta y manejo de información de la base de datos:.....	39
8.3. Servidor API REST.....	41
8.4. OpenLayers	42
8.5. Obtención de geometrías de las poblaciones de Aranguren	44
8.6. Construcción del Front-End e integración en el servidor.	45
9. PROBLEMAS ENCONTRADOS	47

9.1.	Problemas de interacción con el mapa	47
9.2.	Problemas de CORS.....	49
9.3.	Obtención de datos geográficos y almacén en la base de datos.	50
10.	TECNOLOGÍAS.....	52
10.1.	NodeJS + Express.....	52
10.2.	Angular	52
10.3.	QGIS.....	52
10.4.	Geoserver	53
10.5.	PostgreSQL	53
10.6.	PostGIS	53
10.7.	Angular Material	54
10.8.	Lenguajes de programación.	54
10.8.1.	JavaScript.....	54
10.8.2.	HTML.....	54
10.8.3.	CSS	54
10.8.4.	TypeScript	55
11.	MANUAL DE USUARIO.....	56
11.1.	Instalación de PostgreSQL + PostGIS.....	56
11.2.	Instalación de GeoServer	57
11.3.	Instalacion de QGIS y conexión con PostgreSQL.....	58
11.4.	Dibujar geometrías en QGIS y almacenarlas en la base de datos.....	59
11.5.	Conexión Geoserver - PostgreSQL	61
12.	VALORACIONES Y CONCLUSIONES	63
12.1.	Valoraciones	63
12.2.	Conclusiones.....	63
12.3.	Líneas futuras	64
13.	BIBLIOGRAFÍA	65

3. ÍNDICE DE FIGURAS

Figura 1. Diagrama Entidad – Relación.	20
Figura 2. Arquitectura de la solución	27
Figura 3. Diagrama de casos de uso	29
Figura 4. Página principal Gestión Zonas Verdes Aranguren.	31
Figura 5. Menú lateral Gestión Zonas Verdes Aranguren.	32
Figura 6. Elemento seleccionado Gestión Zonas Verdes Aranguren.	32
Figura 7. Listado de trabajos Gestión Zonas Verdes Aranguren.	33
Figura 8. Ficha de trabajo Gestión Zonas Verdes Aranguren.	34
Figura 9. Añadir parte de trabajo Gestión Zonas Verdes Aranguren.	34
Figura 10. Listado de personal Zonas Verdes Aranguren.	35
Figura 11. Editar empleado Gestión Zonas Verdes Aranguren.	35
Figura 12. Añadir empleado Gestión Zonas Verdes Aranguren	36
Figura 13. Listado de material Gestión Zonas Verdes Aranguren	36
Figura 14. Añadir maquinaria Gestión Zonas Verdes Aranguren	37
Figura 15. Ficha de elemento Gestión Zonas Verdes Aranguren	37
Figura 16. Gestor de base de datos: pgAdmin 4	38
Figura 17. Editor de datos geoespaciales: QGIS	39
Figura 18. Conexión servidor – base de datos	39
Figura 19. Consulta de servidor a base de datos	40
Figura 20. Llamada del cliente a la API del servidor.	40
Figura 21. Creación de servidor con Node.js + Express	41
Figura 22. API REST del servidor	41
Figura 23. Creación de capa de fondo en OpenLayers	42
Figura 24. Creación de capa WMS en OpenLayers	42
Figura 25. Creación de mapa en OpenLayers	43
Figura 26. Creación de capa WFS en OpenLayers	43
Figura 27. Consulta de almacén de lugares	44
Figura 28. Consulta de introducción de información relativa a lugares	44
Figura 29. Servidor de desarrollo proporcionado por Angular CLI	45
Figura 30. Envío de la SPA desde el servidor Node.js + Express	46
Figura 31. Proyección EPSG:25830 (Incluye Navarra)	48
Figura 32. Declaración de servidores externos	49

Figura 33. Creación de un Proxy inverso en Node.js	49
Figura 34. Redirección de peticiones según sus parámetros	49
Figura 35. Creación de función y trigger instead of insert	51
Figura 36. Servicio en ejecución del servidor de bases de datos.	56
Figura 37. Consola de arranque de GeoServer	58
Figura 38. Página de inicio del software QGIS	59
Figura 39. Ventana de conexión de QGIS con PostGIS	60
Figura 40. Conexión QGIS con servicio WMS.	61
Figura 41. Guardado de capas editadas en QGIS.	61
Figura 42. Parámetros de la conexión GeoServer – PostgreSQL	62
Figura 43. Capas disponibles en GeoServer tras la conexión con PostgreSQL.	62

4. PALABRAS CLAVE

Sistema de Información Geográfica (GIS), Open data, Gestión de Parques y Jardines, Web-Mapping, SPA, Single Page Application, OGC (Open GIS), PostGIS, GeoServer, QGIS, OL (Open Layers).

4.1. Keywords

Geographic Information System (GIS), Open data, Parks and Gardens Management, Web-Mapping, SPA, Single Page Application, OGC (Open GIS), PostGIS, GeoServer, QGIS, OL (Open Layers).

5. RESUMEN

En este trabajo se ha desarrollado una **aplicación para la gestión** del departamento encargado del mantenimiento **de las zonas verdes** del Valle de Aranguren.

Se trata de un proyecto para el cual no se ha dispuesto de ningún tipo de información ni sistema previo, por lo que ha sido necesario realizar toda la implementación y poblado de datos desde el principio. Para la representación de datos se han empleado los mapas que ofrece el Gobierno de Navarra de manera pública a través del portal SITNA.

Se ha realizado una implementación de una base de datos geográfica y una **herramienta de explotación Web-Mapping** mediante un desarrollo **Full Stack** basado en interfaces adaptativos y **paradigma SPA** (Single Page Application). La principal novedad de la aplicación es que ofrece un mapa como elemento central, sobre el que se podrá navegar, buscar y localizar elementos, y realizar distintas operaciones sobre éstos. Desde el mismo mapa o desde los accesos situados en el menú podremos acceder al área de gestión.

En el apartado de gestión, dispondremos de un **gestor de trabajos**, que permitirá listar todos los trabajos, planificar nuevos trabajos, añadir sobre cada trabajo los partes de trabajo correspondientes, extraer la información en documentos para su almacenaje o análisis posterior, ver trabajos y sus partes en detalle, etc.

También será posible gestionar tanto recursos humanos del departamento como el stock de maquinaria en cada momento, siendo posible controlar las disponibilidades en distintas fechas.

Para su desarrollo se han empleado **tecnologías actuales**, tales como Angular para el desarrollo de una Single Page Application en el Front-End o Node.js + Express en el lado del servidor para programar un API REST en JavaScript. GeoServer, PostgreSQL, PostGIS y QGIS para el manejo de información geográfica y la librería OpenLayers para su representación final en un mapa.

5.1. Abstract

In this project an application has been developed for the management of the maintenance department of the green areas of the Aranguren Valley.

An implementation of a **geographic database and a Web-Mapping exploitation tool** was carried out through a **Full Stack** development based on adaptive interfaces and the **Single Page Application paradigm**. The main novelty of the application is that it offers a map as a central element, on which you can navigate, search and locate elements, and perform different operations on them. From the same map or from the accesses located in the menu we can access the management area.

In the management section, we will have a **job manager**, which will allow you to list all the jobs, plan new jobs, add the corresponding work items on each job, extract the information in documents for later storage or analysis, see jobs and their parts in detail, etc.

It will also be possible to manage both the department's **human resources** and the stock of **machinery** at any given time, being possible to control the availabilities on different dates.

It is a project for which no information or previous system has been provided, so it has been necessary to carry out the entire implementation and data populated from the beginning. For the representation of data, the maps offered by the Government of Navarra in a public way through the SITNA portal have been used.

For its development, **current technologies** have been used, such as Angular for the development of a Single Page Application in the Front-End or Node.js + Express on the server side to program a REST API in JavaScript. GeoServer, PostgreSQL, PostGIS and QGIS for the management of geographic information and the OpenLayers library for its final representation on a map.

6. INTRODUCCIÓN

Los municipios de la comarca de Pamplona destinan una parte importante de sus presupuestos al mantenimiento de Parques y Jardines, lo que implica un gran esfuerzo de gestión de las personas, actividades y recursos implicados que es necesario optimizar, para lo cual disponer de un **sistema de información** bien estructurado y unas buenas aplicaciones de gestión es fundamental.

El servicio de parques y jardines del Valle de Aranguren actualmente **no dispone** de una estructura informática de gestión de dicho mantenimiento, sino que toda la planificación y control de trabajos, y la gestión de recursos se hace en papel. Esto puede ocasionar infinidad de problemas, tanto problemas de coordinación, control de stock de maquinaria en cada momento, recuperación información y su posterior análisis...

Tomando como referencia este modelo se ha desarrollado una aplicación de gestión departamental que evita todos los problemas anteriormente mencionados y ofrece nuevas funcionalidades para poder optimizar recursos, tiempo y trabajo al máximo.

6.1. Objetivos de la aplicación

Teniendo en cuenta la estructura de la que dispone el departamento de Parques y Jardines del Ayuntamiento del Valle de Aranguren, los objetivos principales del proyecto serán:

- Disponer de un sistema de **búsqueda y localización** de los distintos elementos que conforman las zonas verdes del Valle de Aranguren a través de un mapa y acceso rápido al apartado de gestión de éstos.
- Disponer de un sistema de **gestión de trabajos** planificables, sobre los que cada trabajador pueda añadir partes de trabajo.
- Disponer de un sistema de gestión de **empleados**.
- Disponer de un sistema de gestión y control de **maquinaria** y herramientas.
- Disponer de un sistema de gestión que permita listar y recuperar información en ficheros pdf de trabajos realizados, en proceso y planificados, además de sus partes de trabajo correspondientes.

6.2. Resultados del aprendizaje

El proyecto desde un inicio fue pensado para desarrollarse con las **tecnologías modernas y los paradigmas actuales**, lo que ha favorecido que tras el desarrollo haya ampliado conocimientos y mejorado competencias en las siguientes áreas:

- **JavaScript/TypeScript:** como lenguajes de programación orientados a objetos, débilmente tipados y dinámicos. Permiten mejoras en la interfaz y páginas web dinámicas. En nuestro caso se ha empleado tanto en el lado del cliente como en la API REST del servidor, por lo que ha estado presente en todo momento en el desarrollo. Su amplio uso y la interacción con la librería de OpenLayers para la gestión de mapas ha derivado en una mejora de los conocimientos en esta área exponencial.
- **Angular:** como plataforma o framework para el desarrollo de aplicaciones web que siguen el paradigma Single-Page application. Es una tecnología relativamente nueva que divide el desarrollo en componentes, clases y servicios, por lo que ha sido necesario llevar a cabo un proceso de formación para realizar una implementación sólida y robusta.
- **Gestión de información geoespacial:** Conocer los distintos sistemas de proyección de coordenadas y saber cómo transformarlos, tanto para su correcta representación en mapas de OpenLayers como para su correcto almacenaje en la base de datos.
- **PostgreSQL y PostGIS:** como sistema gestor de bases de datos relacional y orientada a objetos, con extensión para datos geoespaciales y operaciones de tipo espacial. Fundamental para proveer de datos a la API REST del servidor y para almacenar los datos espaciales que serán servidos por Geoserver.
- **Geoserver:** como servidor de datos geoespaciales alojados en la base de datos y consumibles por el mapa de lado del cliente. Necesario para servir capas en formato tanto WMS (Web Map Services) como WFS (Web Features Services).
- **IDENA:** como mapa Open-Source de fondo en la aplicación. Ofrece un servicio de acceso a todas las capas de información del SITNA publicadas vía WMS de manera gratuita y sin licencia necesaria: (http://idena.navarra.es/descargas/WMS_WFS_IDENA.pdf)

7. ANÁLISIS DE REQUISITOS

Aplicación Web-Mapping para la gestión de Parques y Jardines es un proyecto que toma como referencia el modelo de gestión implementado a día de hoy en el Ayuntamiento del Valle de Aranguren.

Tras un estudio del documento de requisitos y una primera reunión con el encargado de dicho departamento, obtenemos el siguiente listado de requisitos funcionales:

7.1. Requisitos funcionales.

Todo el sistema va a estar dirigido y preparado para el uso por parte del departamento de Parques y Jardines desde un sitio web local, por lo que en primera instancia no será necesario un control de acceso.

El **encargado** dispondrá de las siguientes funcionalidades:

- 1- Visualización del mapa con todos los elementos del valle. Consideraremos elementos:
 - a. Parques
 - b. Jardines
 - c. Masas arboladas
 - d. Senderos
 - e. Motas acústicas
 - f. Árboles
 - g. Alineaciones
 - h. Plantaciones de monte
- 2- El mapa podrá minimizarse para acceder a la parte de gestión, quedando minimizado, pero siempre accesible para poder abrirse e interactuar con él.
- 3- El mapa permitirá mostrar y ocultar capas, seleccionables desde la **leyenda** que aparecerá en la parte baja cuando el mapa esté maximizado.
- 4- Navegación lateral para buscar y localizar los distintos elementos en el mapa o abrir su ficha de detalle.

5- **Interacción** con los elementos del mapa. Podrá consultar sus datos, los históricos de trabajo e incluso añadir trabajos a cada elementos.

6- Gestión de **personal**:

- a. Controlar listado de los empleados.
- b. Añadir empleados nuevos.
- c. Eliminar empleados.
- d. Actualizar empleados.

7- Gestión de **maquinaria**

- a. Controlar listado de la maquinaria.
- b. Añadir máquinas nuevas.
- c. Eliminar máquinas.
- d. Comprobar disponibilidad de cada máquina en una fecha determinada.

8- Gestión de **trabajos**

- a. Comprobar listado de trabajos.
- b. Comprobar listado de trabajos programados el día en el que se realiza la consulta.
- c. Comprobar listado de trabajos de cada elemento.
- d. Ordenar listado de trabajos por fecha de inicio, prioridad, nombre.
- e. Abrir ficha de detalle de cada trabajo.
- f. Descargar listado de trabajos en PDF.
- g. Marcar trabajo como finalizado.
- h. Crear trabajos.
- i. Descargar ficha de trabajo en PDF (Incluyendo los partes de trabajo correspondientes).

9- Gestión de **partes de trabajo**

- a. Crear parte de trabajo.

Por otra parte, la aplicación está pensada para que el resto de los empleados puedan llevar a cabo las siguientes actividades:

1. Consultar trabajos pendientes para cada día.
2. Añadir partes de trabajo sobre cada trabajo asignado.
3. Navegar y localizar elementos en el mapa.

7.2. Requisitos no funcionales

7.2.1. Requisitos de Carga y Tiempo

El sistema está pensado para un uso departamental, por lo que la concurrencia **no será demasiado elevada**, siendo siempre inferior a tres usuarios concurrentes y habitualmente uno o ninguno.

En cuanto a tiempos de carga se requiere que el sistema realice transacciones de manera rápida, y que la carga de datos en el mapa sea **lo más rápida posible** para una conexión a internet media.

7.2.2. Requisitos Software

Desarrollo:

Todo el desarrollo del proyecto se ha llevado a cabo en un sistema operativo Windows 10, empleando Visual Studio para el desarrollo del Front-End y Sublime Text 3 para el desarrollo del back-end.

Durante todo el proyecto nos hemos apoyado en Express para el desarrollo de nuestra API REST, servidor estándar de facto de Node, y en Angular CLI, el interfaz de línea de comandos para Angular. Como sistema gestor de bases de datos hemos requerido de PostgreSQL + PostGIS como extensión espacial, y Geoserver para la representación de datos espaciales en mapas de OpenLayers.

En resumen, el listado de aplicaciones software necesarias para el desarrollo completo del proyecto ha sido el siguiente:

1. Visual Studio
2. Sublime Text 3
3. GeoServer
4. QGIS
5. PostgreSQL + PostGIS (Gestión a través de pgAdmin 4)

6. Node.js
7. Npm (Manejador de paquetes de Node.js)
8. Angular CLI
9. Angular Material

Despliegue:

Desde el punto de vista de la integración de la aplicación en un entorno real, sería necesario disponer del siguiente material software:

- **Navegador web:** El cliente debe disponer de un navegador web estándar, preferiblemente Chrome, Firefox, Edge o Safari, ya que permiten la interacción correcta de JavaScript y Ajax, permiten el correcto dibujo de los elementos con Material Design y están constantemente actualizados.
- **Instancia de AWS:** En caso de querer alojar el servicio REST en un servidor en la **nube**. Será necesario instalar como sistema operativo Linux y añadir el servidor de bases de datos, las extensiones geográficas y el servidor de datos geográficos GeoServer.
- **Conexión a internet: Obligatoria** en el lado del cliente para obtener los mapas de fondo de IDENA y realizar peticiones al servidor en caso de que éste sea externo.
- **Servicio de Dominio:** de acceso al servidor en caso de seleccionar la solución de servidor remoto.

7.2.3. Requisitos Hardware

Desarrollo:

Para el desarrollo del proyecto se ha empleado un portátil con las siguientes especificaciones técnicas:

- Modelo: Asus X555LDB
- Sistema Operativo: Microsoft Windows 10 Home
- Procesador: Intel Core i7-5500U CPU @ 2.40GHz, 2397 MHz
- Memoria RAM: 8GB
- Almacenamiento: 1TB
- Conexión a internet

Despliegue:

Desde el punto de vista de la **integración** de la aplicación en un **entorno real**, sería necesario disponer del siguiente material hardware:

- Terminal en el cliente con conexión a **Internet** vía inalámbrica o cableada.
- En caso de incorporar el servicio en el propio entorno de trabajo, necesario **servidor** dedicado y unidad de **almacenamiento** u ordenador que mantenga el servicio activo. Dado que la concurrencia del servicio será baja y las peticiones no serán demasiado frecuentes es posible que la última solución sea la más factible.
- Router para la interacción servicio-red, carga de mapas y carga de datos en caso de tener los servidores en la nube.

8. DISEÑO

8.1. Modelo de datos

8.1.1. Diagrama Entidad-Relación

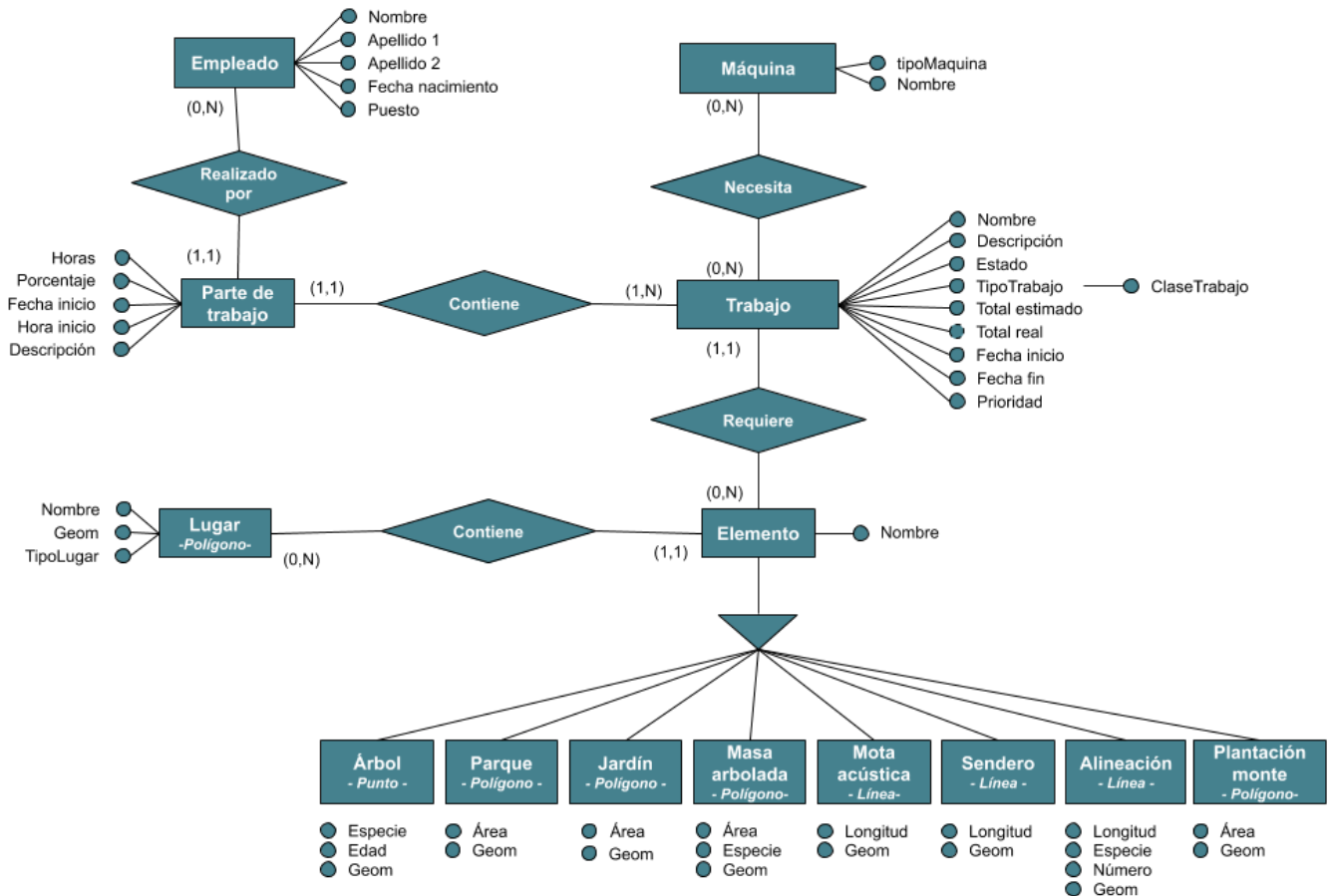


Figura 1. Diagrama Entidad – Relación.

Las **entidades** disponen de la siguiente información:

- *Empleado:* La información personal de cada trabajador.
- *Parte de trabajo:* La información completa de cada parte de trabajo realizado sobre un trabajo.
- *Trabajo:* La información completa de los trabajos planificados y ejecutados.
- *Máquina:* La información de cada máquina disponible en el sistema.
- *Lugar:* La información de cada uno de los lugares que conforman el valle de Aranguren.

- *Elemento*: El nombre de los distintos elementos que serán mantenidos y modificados.
- *Árbol*: La información completa de cada árbol del sistema.
- *Parque*: La información completa de cada parque del sistema.
- *Jardín*: La información completa de cada jardín del sistema.
- *Masa arbolada*: La información completa de cada masa arbolada del sistema.
- *Mota acústica*: La información completa de cada mota acústica del sistema.
- *Sendero*: La información completa de cada sendero del sistema.
- *Alineación*: La información completa de cada alineación del sistema.
- *Plantación de monte*: La información completa de cada plantación de monte del sistema.

Las **relaciones** entre entidades representan la siguiente información:

- *Realizado por*: Un parte de trabajo es realizado por un empleado y un empleado puede realizar ninguno o varios partes de trabajo.
- *Contiene*: Un parte de trabajo pertenece a un trabajo y un trabajo puede contener ninguno o varios partes de trabajos.
- *Necesita*: Un trabajo puede necesitar ninguna o varias máquinas mientras que una máquina puede ser necesitada por ninguno o varios trabajos.
- *Requiere*: Un trabajo es requerido por un elemento, mientras que un elemento puede no requerir trabajos o requerir varios.
- *Contiene*: Un lugar puede contener ninguno o varios elementos y un elemento es contenido en un único lugar.
- *Herencia*: Todos los hijos heredan atributos y relaciones de la entidad padre elemento.

El modelo Entidad/Relación es un **modelo conceptual** en el que se representan la estructura e ideas iniciales. Es por ello que no añadimos atributos identificadores en cada una de las entidades, debido a que es algo necesario para el modelado *real* y su implementación correcta en la base de datos.

Asimismo, al tratarse de un modelo conceptual, establecemos determinados dominios a algunos atributos que posteriormente pasaran a modelarse como tablas auxiliares. Listamos a continuación el dominio de dichos atributos:

Atributo: Máquina.tipoMaquina
Dominio: 'Desbrozadora' o 'Cortacésped' o 'Motosierra' o 'Tijeras' o 'Podadora' o 'Cortadora' o 'Otro'

Atributo: Trabajo.tipoTrabajo
Dominio: ‘Poda’ o ‘Siega’ o ‘Riego manual’ o ‘Abono’ o ‘Plantación’ o ‘Eliminación’

Atributo: Trabajo.tipoTrabajo.claseTrabajo
Dominio: ‘Mantenimiento’ o ‘Modificación’

Atributo: Lugar.tipoLugar
Dominio: ‘Población’ o ‘Polígono Industrial’ o ‘Planta’ o ‘Granja’

Atributo: Empleado.Puesto
Dominio: ‘Encargado’ o ‘Empleado’

Atributo: Trabajo.estado
Dominio: ‘Pendiente’ o ‘En proceso’ o ‘Finalizado’

Atributo: Trabajo.prioridad
Dominio: ‘Baja’ o ‘Media’ o ‘Alta’

8.1.2. Modelo relacional

El modelo conceptual anterior deriva en un modelo relacional en el que, a partir de las relaciones, dominios de atributos y entidades del modelo entidad relación se obtienen las tablas indicadas a continuación. Añadimos ahora identificadores pues con este modelo realizamos la transición de un modelado conceptual a un modelado físico, donde es necesario identificar a cada elemento. Las claves primarias aparecerán subrayadas y las claves extranjeras o foráneas aparecerán *en color azul*, siguiendo la notación del modelo relacional:

- 1- **Empleado** (idEmpleado, Nombre, Apellido1, Apellido2, FechaNacimiento, *Puesto*)
 - a. idEmpleado: Código único que identificará a cada empleado.
 - b. Nombre: Nombre del empleado.
 - c. Apellido1: Primer apellido del empleado.

- d. Apellido2: Segundo apellido del empleado.
- e. FechaNacimiento: Fecha de nacimiento del empleado.
- f. Puesto: Clave extranjera del puesto al que pertenece el empleado.

2- **Puesto** (idPuesto, Puesto)

- a. idPuesto: Código único que identificará a cada puesto.
- b. Puesto: Nombre de cada puesto (Empleado, encargado...).

3- **Parte_trabajo** (idParte, Horas, Porcentaje, Fecha_inicio, Notas, idEmpleado, idTrabajo)

- a. idParte: Código único que identificará a cada parte de trabajo.
- b. Horas: Número de horas que habrá costado realizar cada parte.
- c. Porcentaje: Porcentaje del total de trabajo que se ha realizado en ese parte.
- d. Fecha_inicio: Fecha en la que se ha llevado a cabo el parte.
- e. Notas: Anotaciones sobre el parte de trabajo, incidencias, etc.
- f. idEmpleado: Clave extranjera del empleado que ha realizado el parte de trabajo.
- g. idTrabajo: Clave extranjera del trabajo sobre el que se añadirá el parte.

4- **Máquina** (idMaquina, Nombre, idTipoMaquina)

- a. idMaquina: Código único que identificará a cada máquina.
- b. Nombre: Nombre de la máquina.
- c. idTipoMáquina: Clave extranjera de tipo_maquina.

5- **Tipo_maquina** (idTipoMaquina, NombreTipo)

- a. idTipoMaquina: Código único que identificará a cada tipo de máquina.
- b. NombreTipo: Nombre de cada tipo de máquina (Desbrozadora, cortacésped...).

6- **Trabajo** (idTrabajo, Nombre, Descripción, idPrioridad, Total_Estimado, Total_Real, Fecha_Inicio, Fecha_Fin, Estado, idTipoTrabajo, idElemento)

- a. idTrabajo: Código único que identificará a cada trabajo.
- b. Nombre: Nombre asignado a cada trabajo.
- c. Descripción: Descripción breve del objetivo del trabajo.
- d. idPrioridad: Clave extranjera de prioridad, representará la prioridad del trabajo respecto al resto.

- e. Total_estimado: Horas estimadas que llevará completar el trabajo.
- f. Total_real: Horas reales que supone completar el trabajo. Se calcula a partir de los partes de trabajo asignados a dicho trabajo.
- g. Fecha_inicio: Fecha en la que comienza a realizarse el trabajo.
- h. Fecha_fin: Fecha en la que el trabajo se concluye y da por finalizado.
- i. Estado: Clave extranjera de *estado* que representa el estado del trabajo en un momento determinado.
- j. idTipoTrabajo: Clave extranjera de *tipo_trabajo* que representa el tipo de trabajo asignado.
- k. idElemento: Clave extranjera del *elemento* sobre el que se va a realizar el trabajo.

7- **Estado** (idEstado, Estado)

- a. idEstado: Clave única que identificará cada estado.
- b. Estado: Nombre de cada estado.

8- **Tipo_trabajo** (idTipoTrabajo, Descripción, idClase)

- a. idTipoTrabajo: Clave única que identificará cada tipo de trabajo.
- b. Descripción: Nombre del tipo de trabajo.
- c. idClase: Clave extranjera de la *clase_trabajo* a la que va a pertenecer el tipo.

9- **Clase_trabajo** (idClase, NombreClase)

- a. idClase: Clave única que identificará cada clase de trabajo.
- b. NombreClase: Descripción de la clase.

10- **Prioridad** (idPrioridad, Prioridad)

- a. idPrioridad: Clave única que identificará cada prioridad.
- b. Prioridad: Nombre de la prioridad.

11- **Tipo_lugar** (idTipoLugar, Descripción)

- a. idTipoLugar: Clave única que identificará a cada tipo de lugar.
- b. Descripción: Descripción de cada tipo de lugar.

12- **Lugar** (idLugar, Nombre, Geom, idTipoLugar)

- a. idLugar: Clave única que identificará a cada lugar.
- b. Nombre: Nombre del lugar.
- c. Geom: Geometría del lugar. Atributo espacial de tipo polígono.

- d. idTipoLugar: Clave extranjera del *tipo_lugar* al que pertenecerá el lugar.

13- **Elemento** ([idElemento](#), Nombre, [idLugar](#))

- a. idElemento: Clave única que identificará a cada elemento.
- b. Nombre: Nombre de cada elemento.
- c. idLugar: Clave extranjera del *lugar* al que pertenece cada elemento.

14- **Máquina_trabajo** ([idMaquina](#), idTrabajo)

- a. idMáquina: Clave extranjera de la *maquina* asignada al trabajo.
- b. idTrabajo: Clave extranjera del *trabajo* asignado a la máquina.

15- **Árbol** ([idElemento](#), Especie, Edad, Geom)

- a. idElemento: Clave extranjera del *elemento* padre.
- b. Especie: Nombre de la especie a la que pertenece el árbol.
- c. Edad: Número de años que tiene el árbol.
- d. Geom: Geometría del árbol. Atributo espacial de tipo punto.

16- **Parque** ([idElemento](#), Área, Geom)

- a. idElemento: Clave extranjera del *elemento* padre.
- b. Área: Metros cuadrados de extensión del parque.
- c. Geom: Geometría del parque. Atributo espacial de tipo polígono.

17- **Jardín** ([idElemento](#), Área, Geom)

- a. idElemento: Clave extranjera del *elemento* padre.
- b. Área: Metros cuadrados de extensión del jardín.
- c. Geom: Geometría del jardín. Atributo espacial de tipo polígono.

18- **Masa_arbolada** ([idElemento](#), Especie, Geom)

- a. idElemento: Clave extranjera del *elemento* padre.
- b. Especie: Nombre de la especie de árbol presente en la masa arbolada.
- c. Geom: Geometría de la masa arbolada. Atributo espacial de tipo polígono.

19- **Mota_acústica** ([idElemento](#), Longitud, Geom)

- a. idElemento: Clave extranjera del *elemento* padre.

- b. Longitud: Longitud en metros de la mota acústica.
- c. Geom: Geometría de la mota acústica. Atributo espacial de tipo línea.

20- **Sendero** ([idElemento](#), Longitud, Geom)

- a. idElemento: Clave extranjera del *elemento* padre.
- b. Longitud: Longitud en metros del sendero.
- c. Geom: Geometría del sendero. Atributo espacial de tipo línea.

21- **Alineación** ([idElemento](#), Longitud, Especie, Número, Geom)

- a. idElemento: Clave extranjera del *elemento* padre.
- b. Longitud: Longitud en metros de la alineación.
- c. Especie: Especie de los árboles que pertenecen a la alineación.
- d. Número: Número de árboles que conforman la alineación.
- e. Geom: Geometría de la alineación. Atributo espacial de tipo línea.

22- **Plantación_monte** ([idElemento](#), Área, Geom)

- a. idElemento: Clave extranjera del *elemento* padre.
- b. Área: Metros cuadrados de extensión de la plantación.
- c. Geom: Geometría de la plantación. Atributo espacial de tipo polígono.

8.2. Arquitectura de la solución

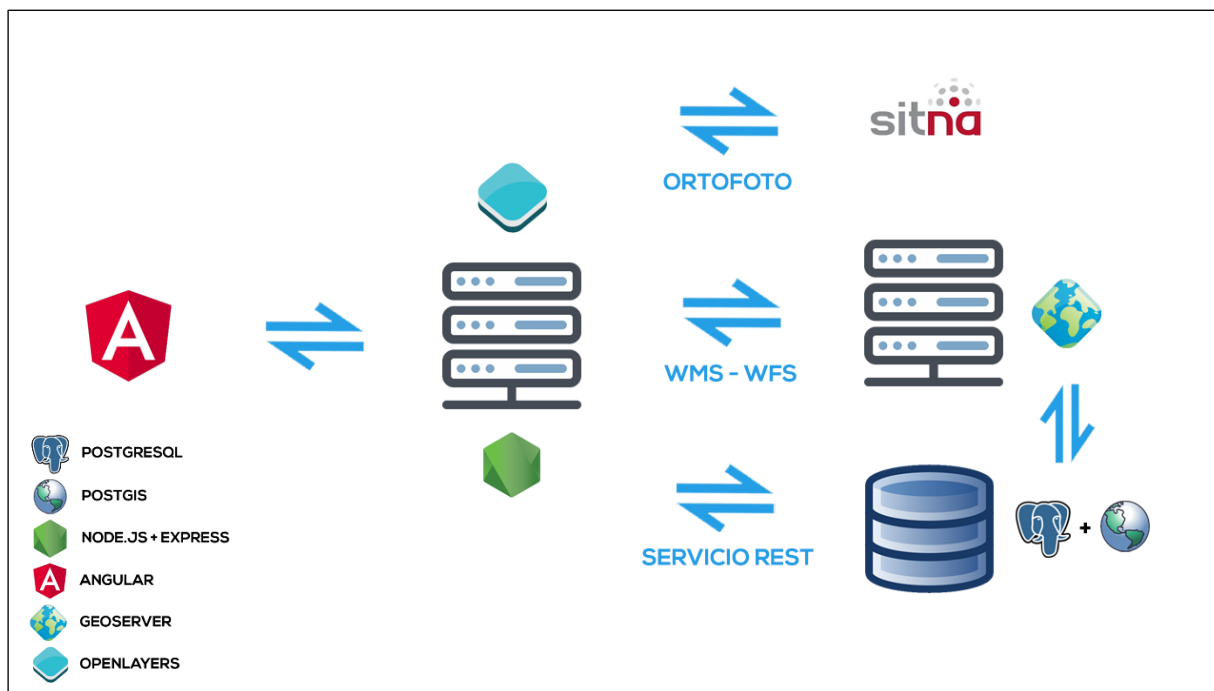


Figura 2. Arquitectura de la solución.

La arquitectura de la solución está dividida en 2 componentes especializados. El primero está orientado al servicio de información geográfica mientras que el segundo se especializa en la información literal, orientado a toda la parte de gestión de la aplicación.

8.2.1. Arquitectura de mapas

Se obtiene información geográfica desde Geoserver y desde SITNA.

- OpenLayers: Librería en el lado del cliente que muestra mapas web interactivos.
- Geoserver: Información geográfica de la base de datos.
- SITNA: El servicio WMS nos proporciona la imagen de fondo para nuestro mapa.

Ambos servicios son consumidos por OpenLayers en el lado del cliente.

8.2.2. Arquitectura de texto

Se obtiene y se actualiza información de la base de datos PostgreSQL a través de un API REST establecido en el servidor que será consumido por el cliente.

8.2.3. Componentes

Servidor de Bases de Datos: Proporciona información geográfica a GeoServer gracias a la extensión geográfica PostGIS y a demás proporciona y almacena los datos requeridos por el API REST ubicado en el servidor.

Servidor web: Se ha empleado un servidor Node.js + Express como servidor web. Trabaja en JavaScript y además de ofrecer un API REST para el área de gestión de la aplicación sirve como proxy inverso redirigiendo peticiones a SITNA y a GeoServer.

WMS: Servicio web empleado tanto en SITNA como en GeoServer que ofrece mapas de datos de forma dinámica a partir de información geográfica.

Angular: Framework para el desarrollo del FrontEnd, que mediante TypeScript ofrece una web de una sola página, también denominada Single Page Application, que carga el contenido de manera dinámica sin necesidad de recargar la página entera.

8.3. Casos de uso

En los casos de uso de gestión, se incluyen implícitamente los casos de uso de creación, eliminación, actualización y consulta, las denominadas operaciones **CRUD**.

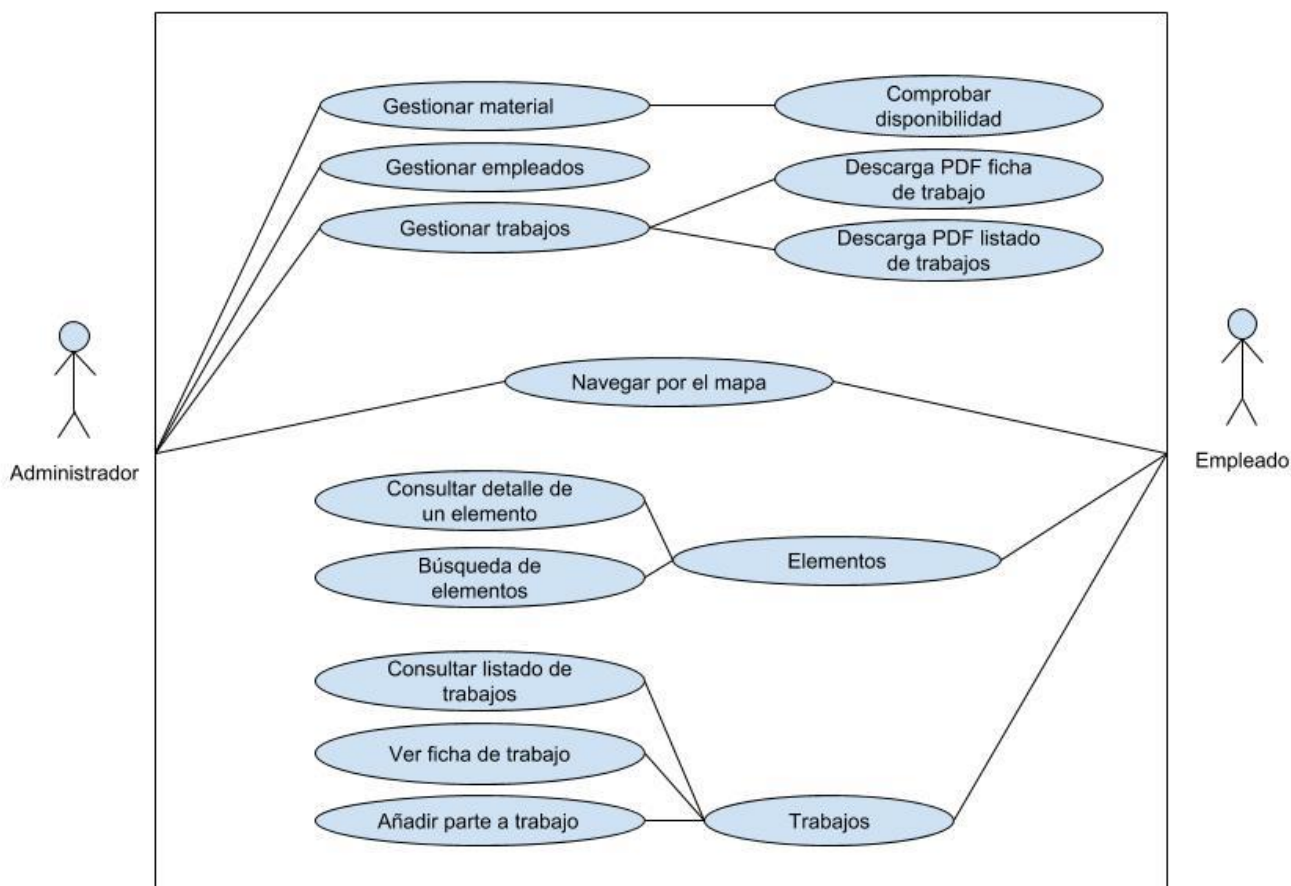


Figura 3. Diagrama de casos de uso

Administrador

Gestión de material: Comprobar listado de materiales, creación de material, eliminación de material, actualización de material, comprobación de material disponible y ocupado a partir de una fecha.

Gestión de empleados: Comprobar listado de empleados, añadir nuevos empleados, modificar empleados existentes, eliminar empleados del sistema.

Gestión de trabajos: Creación de trabajos introduciendo fechas estimadas, horas, prioridad, material necesario, descarga de listado de trabajos en PDF, descarga de ficha de trabajos en PDF, marcar trabajos como finalizados.

Administradores y Empleados

Navegación por el mapa: Búsqueda y localización de elementos en el mapa desde un buscador lateral o desde el propio mapa con el ratón. Acceso al menú de opciones rápidas de cada elemento seleccionado. Filtro de capas desde la leyenda inferior. Minimización/Maximización del mapa, alternándolo con la sección de gestión.

Empleados

Elementos: Acceso a la ficha en detalle de los distintos elementos, búsqueda de elementos desde el menú lateral.

Trabajos: Consulta de listado de trabajos realizados, pendientes y en proceso, consulta de ficha en detalle de los trabajos, posibilidad de añadir partes de trabajo sobre cada trabajo en proceso.

8.4. Diseño Interfaz

Analizados los requisitos, realizamos el desarrollo de la web tratando en todo momento de que sea práctica en términos de **usabilidad y accesibilidad**. Para ello estableceremos un menú superior con acceso rápido a las principales categorías de la aplicación que estará presente en todo momento, al igual que la barra de búsqueda lateral. Será el mapa el que varíe de tamaño para dejar espacio a toda la parte de gestión.

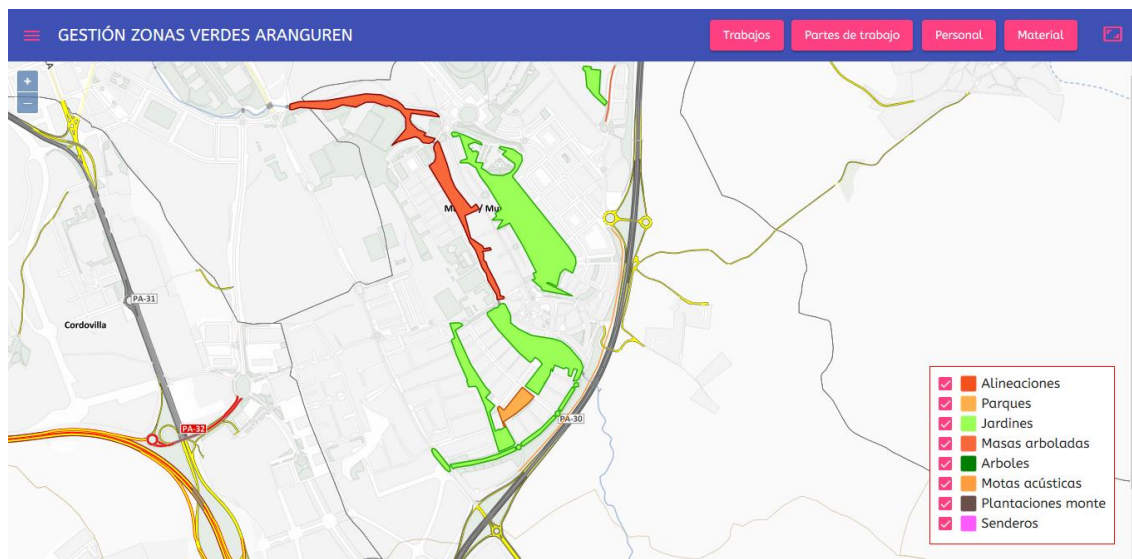


Figura 4. Página principal Gestión Zonas Verdes Aranguren.

Tal y como se aprecia en la Figura 4, en la parte inferior derecha dispondremos de una **leyenda** con todos los tipos de elementos del mapa, que nos permitirá elegir las capas que queremos que sean visibles. En la parte superior izquierda dispondremos de un botón que abrirá y cerrará el menú de búsqueda que se muestra a continuación.

Este menú lateral se denomina *sideNav* y es uno de los elementos ya implementados que ofrece **Angular Material**.

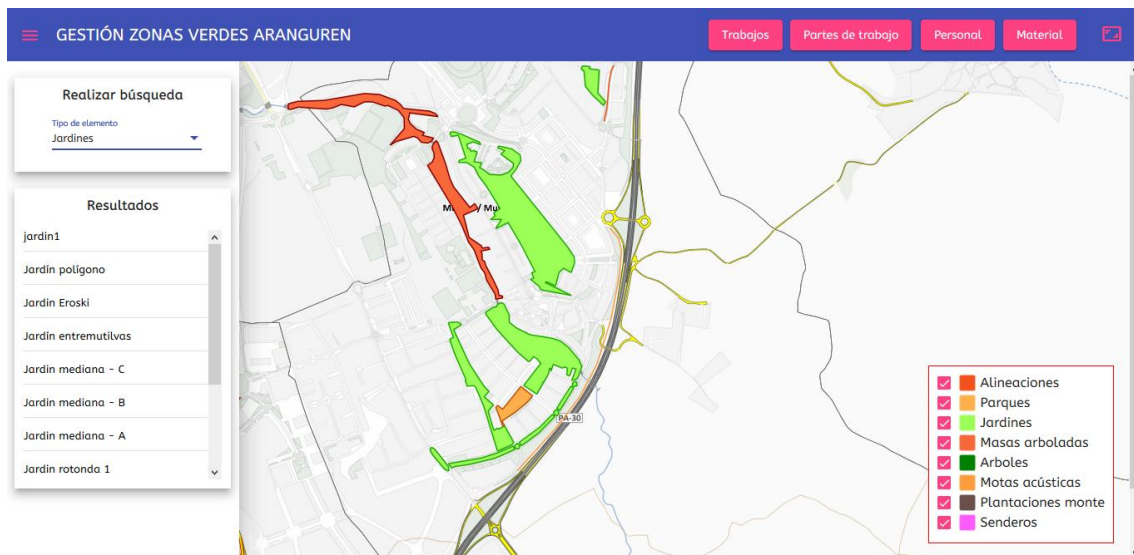


Figura 5. Menú lateral Gestión Zonas Verdes Aranguren.

El botón situado en la esquina superior derecha nos permitirá **alternar** entre una vista del mapa a tamaño completo, o minimizado en la esquina inferior de la pantalla. En la vista minimizada es posible hacer clic sobre el mapa para maximizarlo, facilitando así la interacción con el mapa.

Al seleccionar un elemento en el buscador lateral o al pulsar sobre cualquiera de los elementos directamente en el mapa, éste se resaltará y se situará en el centro de la pantalla, abriendo una pequeña ficha de información y ofreciendo accesos rápidos a distintas funcionalidades para el elemento en cuestión.

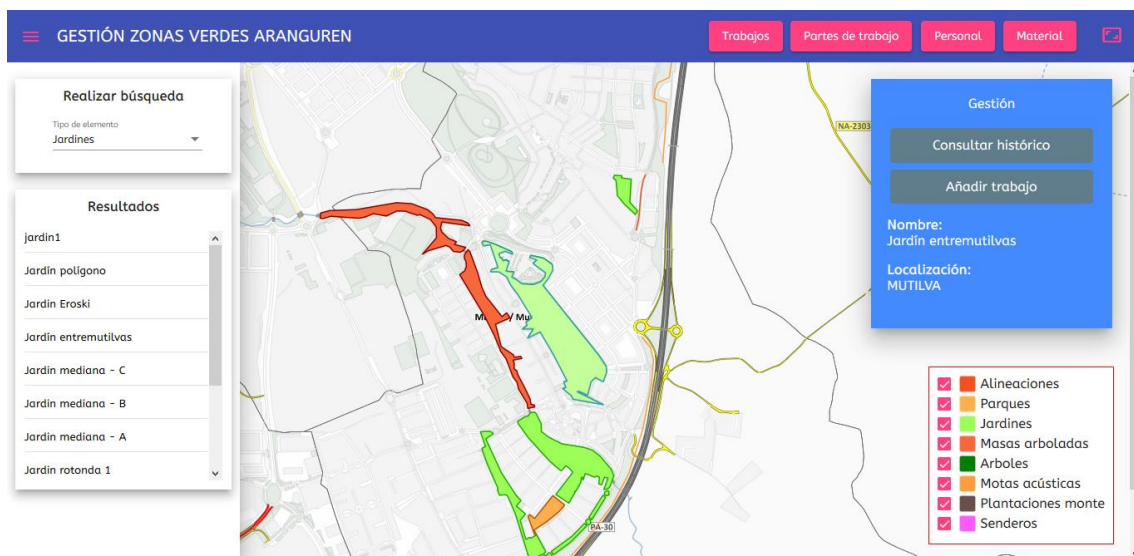


Figura 6. Elemento seleccionado Gestión Zonas Verdes Aranguren.

Analizaremos ahora las distintas interfaces para los elementos del menú superior.

Al pulsar sobre **trabajos** el mapa se minimizará y encontraremos dos listados: En el listado superior aparecerán los trabajos que están programados para el día en el que se realiza la consulta, y en el inferior aparecerá un listado tanto de trabajos planificados como completados. El botón de *Descargar en PDF* permitirá descargar en un fichero de tipo PDF un único listado con todos los trabajos.

Esta pantalla es equivalente a la que aparecería si pulsamos *Consultar histórico* en el detalle de un elemento en el mapa, salvo que los trabajos que aparecerían en ese caso serían los del elemento seleccionado.

The screenshot shows the 'Gestión Zonas Verdes Aranguren' web application. At the top, there is a navigation bar with the title and several menu items: 'Trabajos', 'Partes de trabajo', 'Personal', and 'Material'. Below the navigation bar, there is a search section on the left with a search bar and a dropdown menu for 'Tipo de elemento' (set to 'Jardines'). Below the search bar is a list of search results under the heading 'Resultados', including 'jardin1', 'Jardín polígono', 'Jardín Eroski', 'Jardín entremutilvas', 'Jardín mediana - C', 'Jardín mediana - B', 'Jardín mediana - A', and 'Jardín redonda 1'. The main content area is titled 'TRABAJOS' and contains two tables. The first table, 'Programados para hoy', has columns for 'Id.', 'Nombre', 'Fecha de inicio', 'Prioridad', 'Estado', and 'Detalles'. It lists two jobs: 'Siega regular de césped' (Id. 43, Apr 22, 2019, Media, Pendiente) and 'Poda hoy' (Id. 42, Mar 31, 2019, Media, Finalizado). The second table, 'Listado completo', has the same columns and lists three jobs: 'Siega regular de césped' (Id. 43, Apr 22, 2019, Media, Pendiente), 'Poda hoy' (Id. 42, Mar 31, 2019, Media, Finalizado), and 'timestamp' (Id. 41, Mar 30, 2019, Media, Finalizado). A map is visible in the bottom right corner, showing a location with a red box around it. There are also buttons for 'AÑADIR TRABAJO' and 'DESCARGAR EN PDF' at the top right of the main content area.

Figura 7. Listado de trabajos Gestión Zonas Verdes Aranguren.

Es posible ver cada uno de los trabajos en detalle para añadir partes, consultar información, etc. accediendo a través del botón *Detalles*, que nos llevará a la siguiente página:

GESTIÓN ZONAS VERDES ARANGUREN

Trabajos Partes de trabajo Personal Material

Volver al listado Descargar PDF

Poda diaria (Jardín entremutilvas) En proceso

Comenzar por la zona norte

Prioridad	Fecha de inicio	Fecha fin	Horas estimadas	Horas reales
Alta	Apr 29, 2019	Apr 29, 2019	8 horas.	2 horas.

Marcar como finalizado

Partes de trabajo Añadir parte

Fecha	Empleado	Horas
Apr 29, 2019	Juan	2 horas.
Horas totales		2 horas

Figura 8. Ficha de trabajo Gestión Zonas Verdes Aranguren.

La ficha de trabajo nos ofrece información básica, y además **calcula** el tiempo real invertido en el trabajo a partir de los partes que se le han añadido. Es posible también obtener un fichero PDF con la información del trabajo y todos los partes de éste. Para cerrar un trabajo bastará con pulsar el botón de *Marcar como finalizado*.

El botón *Partes de trabajo* del menú superior es un acceso rápido a la creación de un parte de trabajo. Es necesario buscar el trabajo en un listado, aunque también es posible añadir un parte directamente desde la ficha del trabajo, evitando tener que buscarlo en el desplegable.

GESTIÓN ZONAS VERDES ARANGUREN

Trabajos Partes de trabajo Personal Material

Realizar búsqueda

Tipo de elemento Jardines

Resultados

- Jardin1
- Jardin poligono
- Jardin Eroski
- Jardin entremutilvas
- Jardin mediana - C
- Jardin mediana - B
- Jardin mediana - A
- Jardin rotonda 1

Añadir parte trabajo

1. Trabajo y empleado

Trabajo

Empleado

2. Datos básicos

Horas

Porcentaje (%)

Fecha dd / mm / aaaa

3. Anotaciones (Opcional)

Notas

AÑADIR CANCELAR

Figura 9. Añadir parte de trabajo Gestión Zonas Verdes Aranguren.

En el apartado de *Personal*, encontraremos un listado con todos los empleados que se encuentran en ese momento en activo.

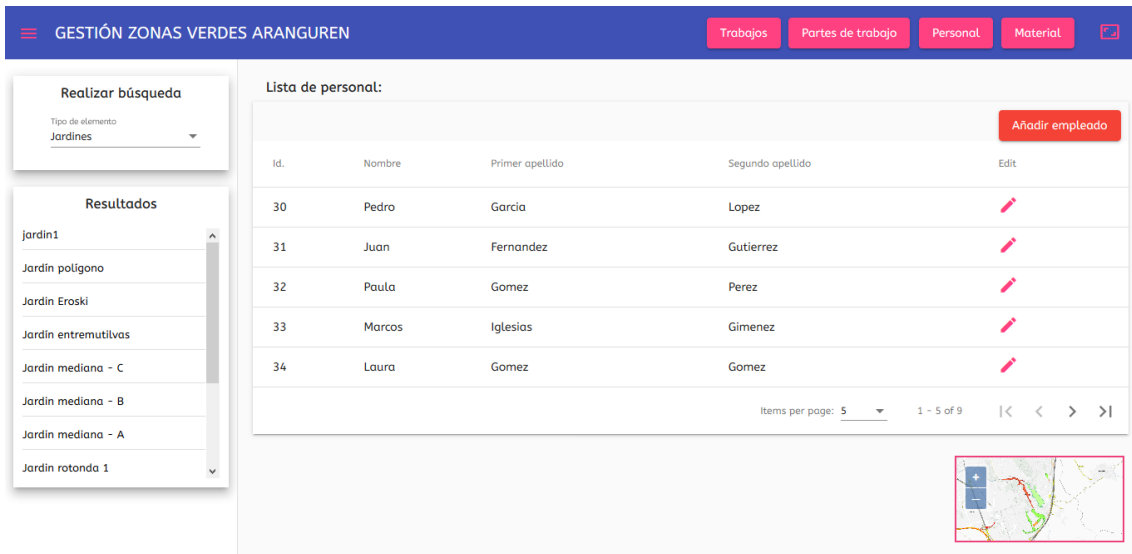


Figura 10. Listado de personal Zonas Verdes Aranguren.

Podremos editar a los empleados (En la ficha de editado también será posible eliminarlos):

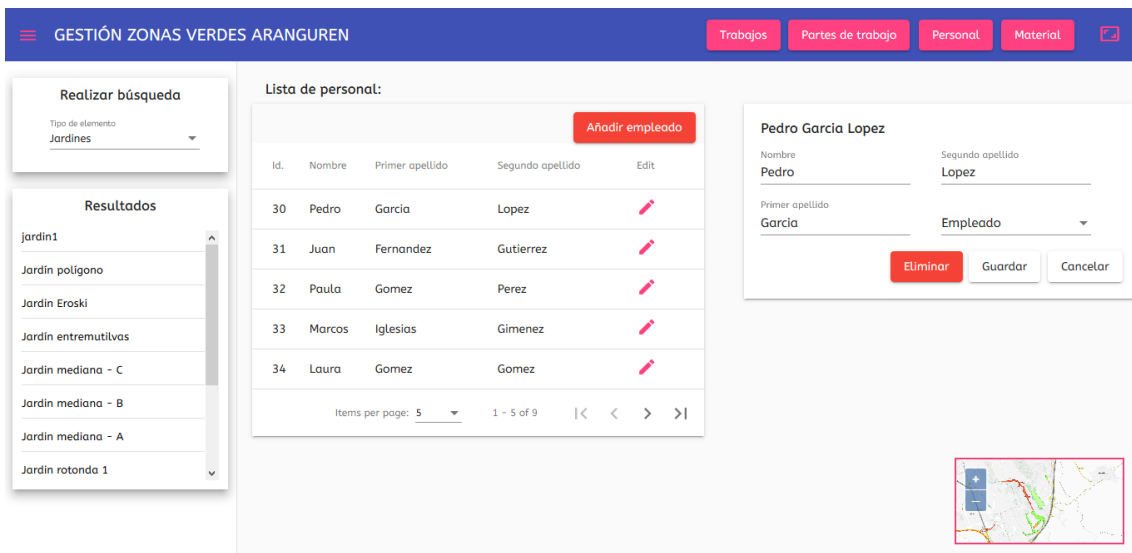


Figura 11. Editar empleado Gestión Zonas Verdes Aranguren.

Y añadir empleados será un proceso análogo al de editar:

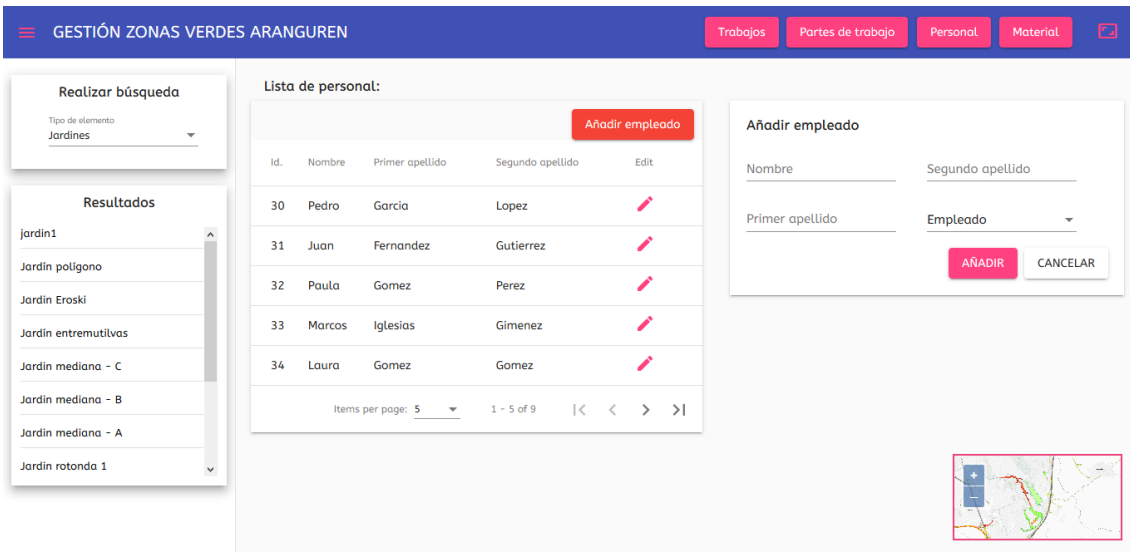


Figura 12. Añadir empleado Gestión Zonas Verdes Aranguren.

Por último, en la sección de **material** encontraremos otros dos listados, el primero con todo el material disponible en la fecha en la que se realiza la consulta y en el segundo el material que está siendo utilizado en los trabajos que se están llevando a cabo el día de la consulta.

Es posible comprobar la disponibilidad del material para distintas fechas introduciendo el día a consultar en el datepicker correspondiente.

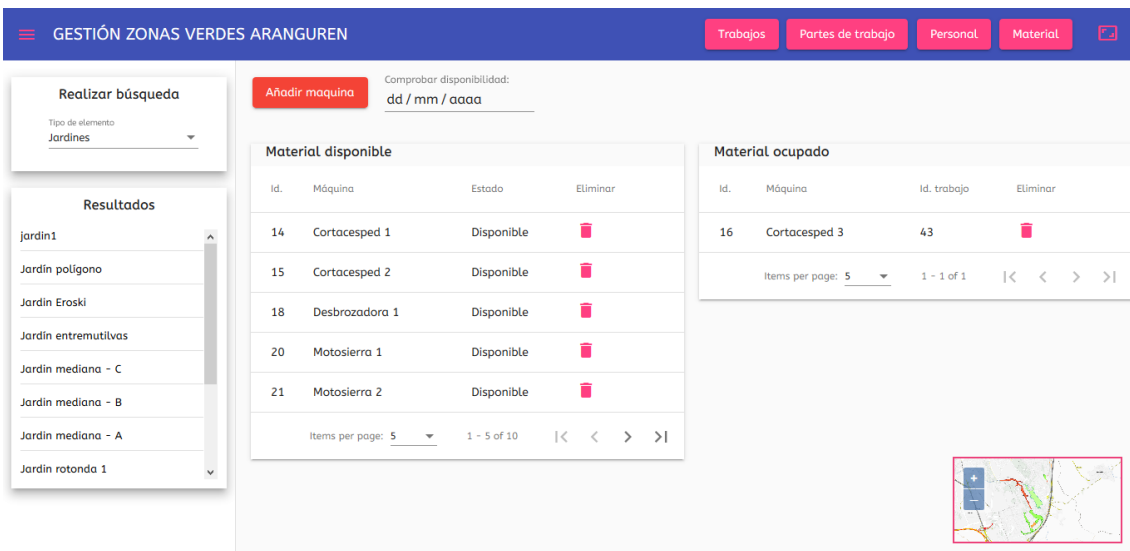


Figura 13. Listado de material Gestión Zonas Verdes Aranguren.

La eliminación de material se realiza a través del icono de borrado y es posible añadir maquinaria nueva mediante el botón de *Añadir máquina*.

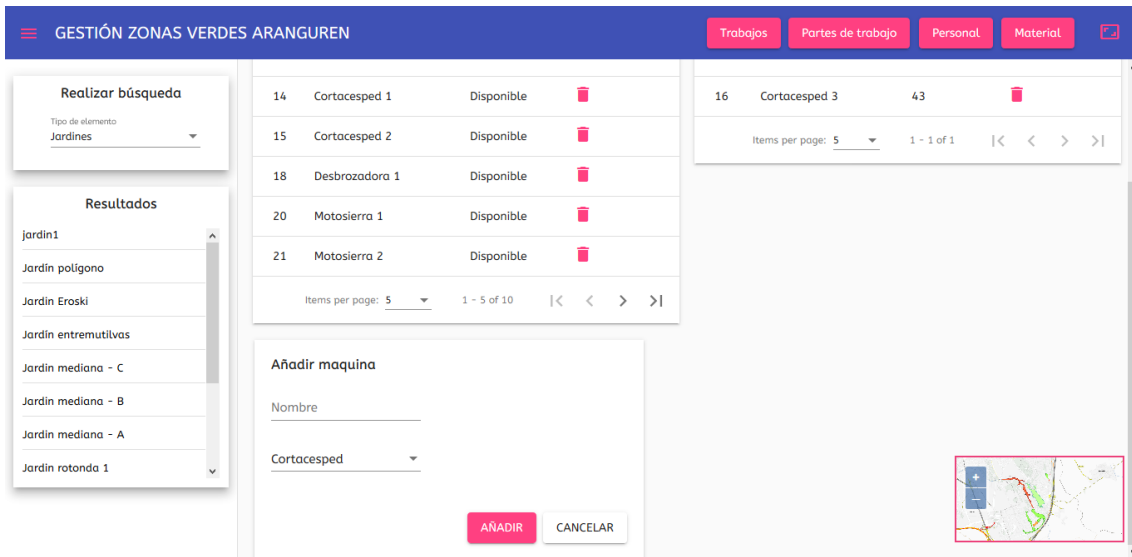


Figura 14. Añadir maquinaria Gestión Zonas Verdes Aranguren.

Cabe destacar que en todas las pantallas anteriores es posible cerrar el menú lateral y todos los elementos restantes se acondicionan al nuevo tamaño, haciendo los interfaces adaptativos y la navegación mucho más satisfactoria.

Si el mapa está minimizado y realizamos una búsqueda en dicho menú lateral, se abrirá una ficha del elemento seleccionado con su información y los trabajos que se le han realizado y que están en proceso.

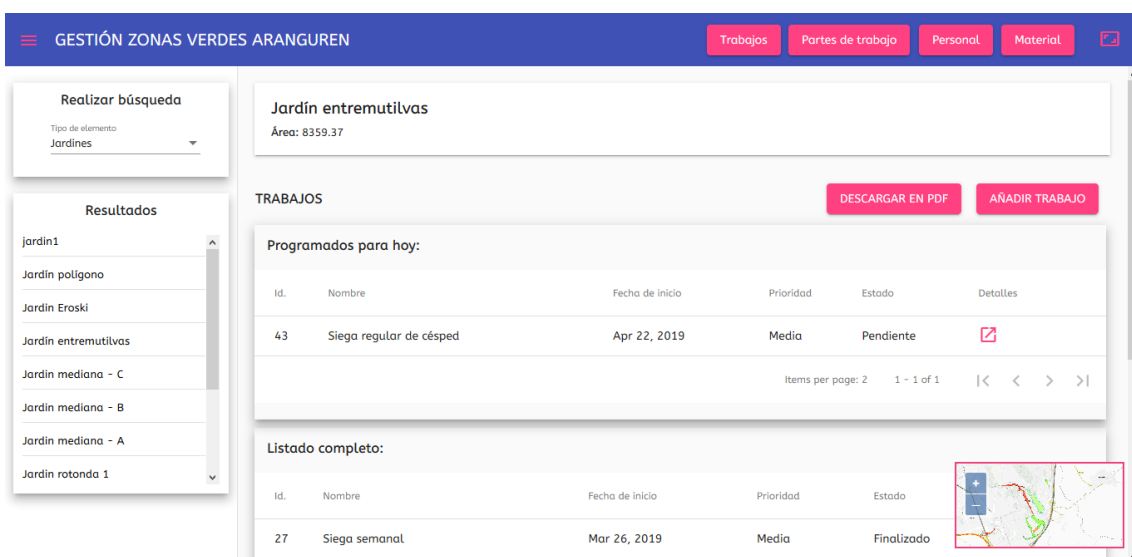


Figura 15. Ficha de elemento Gestión Zonas Verdes Aranguren.

9. IMPLEMENTACIÓN

En este capítulo se recogen las partes del desarrollo del proyecto más críticas e importantes.

9.1. Base de datos

La base de datos se ha desarrollado empleando el sistema gestor de bases de datos **PostgreSQL** con la extensión para datos de tipo espacial **PostGIS**. Para la gestión de la base de datos se ha utilizado la interfaz de PostgreSQL, **pgAdmin 4**, a la que se accede a través del navegador.

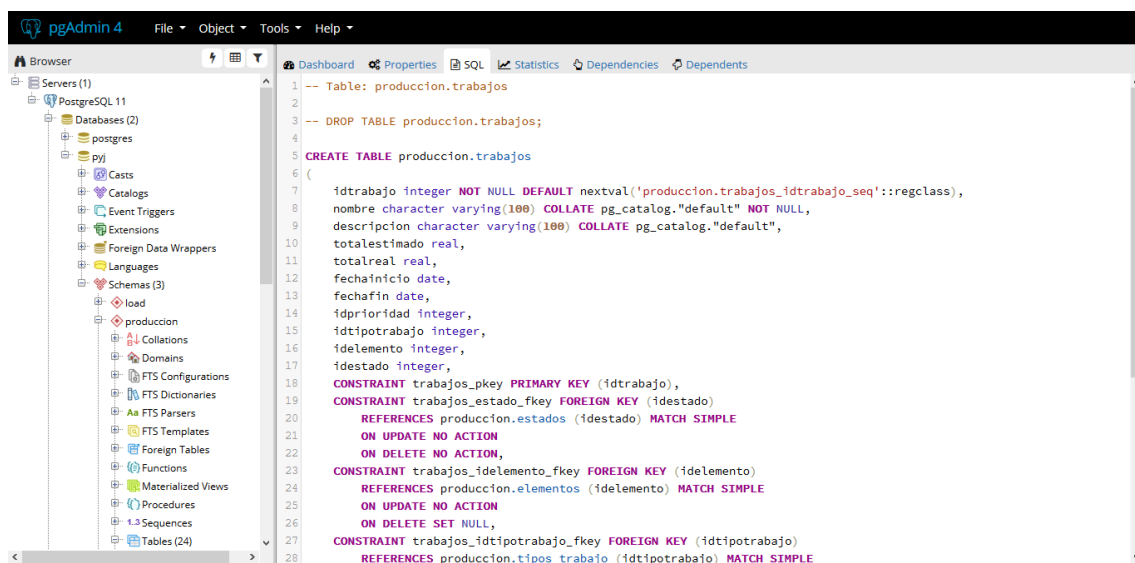


Figura 16. Gestor de base de datos: pgAdmin 4.

Para poblarla con datos geospaciales se ha empleado la herramienta **QGIS**, que permite realizar una conexión con la extensión *PostGIS* de PostgreSQL, permitiéndonos dibujar directamente sobre cualquier tabla de nuestra base de datos que disponga de una columna de tipo espacial.

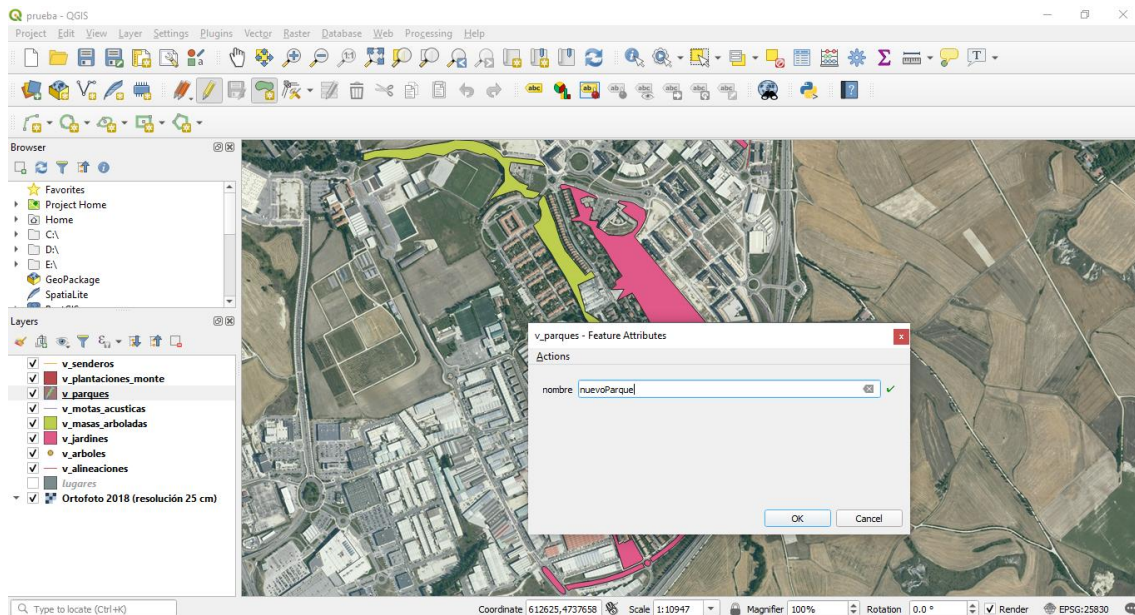


Figura 17. Editor de datos geoespaciales: QGIS.

9.2. Conexión, consulta y manejo de información de la base de datos:

Las consultas a la base de datos se realizan desde nuestro servidor **Node.js express** desarrollado en JavaScript. Node.js dispone de un módulo de conexión específico para conexiones con postgres, por lo que el proceso de conexión es bastante sencillo:

```
const Pool = require('pg').Pool
const pool = new Pool({
  user: 'postgres',
  host: 'localhost',
  database: 'pyj',
  password: 'admin',
  port: 5432,
})
```

Figura 18. Conexión servidor – base de datos.

Y para las consultas, siendo pool la variable que contiene la conexión con la base de datos, emplearemos el siguiente procedimiento:

```
const getEmpleados = (request, response) => {
```

```

pool.query('SELECT * FROM produccion.empleados', (error,
results) =>{
  if (error) { throw error }
  response.status(200).json(results.rows)
})
}

```

Figura 19. Consulta de servidor a base de datos.

Devolviendo el resultado en JSON ya que estamos ofreciendo un servicio REST que devuelve los datos en este formato.

Para las peticiones desde el lado del cliente, Angular permite crear servicios, componentes que generan un servicio http y realizan peticiones GET, POST de manera muy sencilla y que, a partir de una clase predefinida por el usuario, parsean automáticamente la respuesta al tipo de objeto que se le haya indicado en la petición simplificando enormemente el proceso de comunicación con la base de datos:

```

private partesUrl = 'http://localhost:3000/partes';
constructor(private http: HttpClient) { }

addParte(parte: ParteTrabajo): Observable<ParteTrabajo>{
  return this.http.post<ParteTrabajo>(this.partesUrl,
parte, httpOptions).pipe(

catchError(this.handleError<ParteTrabajo>('addParte'))
);
}

```

Figura 20. Llamada del cliente a la API del servidor.

9.3. Servidor API REST

El desarrollo de nuestra **API Rest** se ha llevado a cabo con Node.js y Express, que a través de las siguientes líneas de código generan un servidor sobre el que posteriormente añadiremos nuestra API REST.

```
const express = require("express");
const app = express();

app.listen(3000, () => {
  console.log("El servidor está inicializado en el puerto 3000");
});
```

Figura 21. Creación de servidor con Node.js + Express

Ahora tendremos un servidor activo en nuestra dirección localhost:3000 sobre el que podremos realizar peticiones REST que nos devolverán información en formato JSON. En el manejo de peticiones, estableceremos una acción para cada tipo de petición recibida, distinguiéndolas por el tipo de petición y por sus parámetros:

```
app.get('/empleados', db.getEmpleados)
app.get('/empleados/puestos', db.getPuestos)
app.get('/empleados/:idempleado', db.getEmpleadoById)
app.post('/empleados', db.createEmpleado)
app.put('/empleados', db.updateEmpleado)
app.delete('/empleados/:idempleado', db.deleteEmpleado)
```

Figura 22. API REST del servidor.

En este caso, *db* será un fichero JavaScript en el que estarán la conexión a la base de datos y las distintas operaciones sobre esta, tal y como se indica en la *Figura 19*. Para emplearlo es necesario añadir en el servidor la siguiente línea:

```
const db = require('./queries')
```

9.4. OpenLayers

OpenLayers es una biblioteca de JavaScript de código abierto bajo una derivación de la licencia BSD para mostrar **mapas interactivos en los navegadores web**. En nuestro caso, el mapa es parte fundamental de la aplicación y contiene varios tipos de capas:

- **Capa base:** o mapa de fondo, ortofoto procedente de IDENA. Es necesario declarar una Imagen WMS, aplicarla a una capa y añadirla al mapa instanciado anteriormente:

```
idena = new ImageWMS({
  url: 'https://idena.navarra.es/ogc/wms',
  params: {'LAYERS': 'mapaBase'}
})

imageLayer = new ImageLayer({
  source: idena
})
```

Figura 23. Creación de capa de fondo en OpenLayers.

- **Capa con los elementos:** Capa que contiene una imagen WMS del grupo de capas procedente de GeoServer con todos los elementos del mapa. Los estilos son proporcionados por GeoServer.

```
wmsLayer = new ImageLayer({
  source: new ImageWMS({
    url: 'http://localhost:3000/capas',
    params: {'LAYERS': 'pyj:parquesyjardines'},
    ratio: 1,
    serverType: 'geoserver'
  })
})
```

Figura 24. Creación de capa WMS en OpenLayers.

Ambas capas se introducen en el atributo layers de la instancia de *Map* correspondiente, a la que se le aplica una vista con el centro y el zoom deseados.

```

layers = [imageLayer, wmsLayer];
view = new View({
  center: ol.proj.fromLonLat([-1.616041, 42.788064]),
  zoom: 15
});

map = new Map({
  view: view,
  layers: layers,
  target: 'map',
});

```

Figura 25. Creación de mapa en OpenLayers.

A la hora de hacer clic en cualquiera de los elementos, obtenemos el Feature correspondiente a partir de las coordenadas del clic del evento. Mediante el método *getFeatureInfoUrl* obtenemos la URL a la cual realizar la petición que nos devolverá la Feature en cuestión.

Tras esto y con los datos recibidos ya podemos pintar la nueva capa resaltada en el mapa:

```

var source = new VectorSource({
  features : [new Feature({
    name : a.features[0].id,
    geometry : geometria
  })]
});
wfsLayer.setSource(source);

```

Figura 26. Creación de capa WFS en OpenLayers.

9.5. Obtención de geometrías de las poblaciones de Aranguren

Al almacenar distintos datos espaciales el modelo de datos obliga a que se indique la población de Aranguren a la que pertenece. Para evitar introducir los datos de todo el valle manualmente, se ha realizado un **proceso automatizado** que explicaremos a continuación:

En postgres generamos un **nuevo esquema de tipo carga** en el que almacenaremos los valores del catastro agrupados según nuestras necesidades. En él creamos una tabla de tipo lugar, que contendrá los campos id, nombre, y geometría de tipo multi poligonal.

Descargamos los datos de las poblaciones de Aranguren desde <https://catastro.navarra.es/descargas/>

Desde QGIS, en el esquema de carga, importamos el **fichero tipo shape** CATAST_Pol_CascoUrbano.shp que contiene los datos que necesitamos en nuestro modelo. Estos datos estarán contenidos ahora en una nueva tabla CATAST_Pol_CascoUrbano, pero existen varias entradas por cada población (Varios polígonos independientes).

Ahora, mediante una consulta en QGIS, cargamos los datos agrupados por población con la consulta:

```
insert into load.lugares (id,nombre,geom)
select centidadc, entidadc, ST_UNION(geom) geom
from load.CATAST_Pol_CascoUrbano
group by centidadc, entidadc
order by centidadc, entidadc;
```

Figura 27. Consulta de almacén de lugares.

Esto nos permitirá tener en nuestra tabla de carga los valores con los todos los polígonos agrupados por municipio (la función ST_UNION los agrupará en el campo geom de tipo multi poligonal).

Lo almacenamos ahora en nuestra tabla lugares de producción con la consulta:

```
insert into produccion.lugares (idlugar,nombre,geom)
select id,nombre,geom from load.lugares
```

Figura 28. Consulta de introducción de información relativa a lugares.

De manera que, si nuestro cliente decide modificar algún lugar, simplemente habrá que cambiar la consulta de la tabla de carga y volcar los nuevos datos a la tabla en producción.

9.6. Construcción del Front-End e integración en el servidor.

Tal y como se ha explicado en apartados anteriores, el Front-End de la aplicación ha sido desarrollado con Angular CLI, el interfaz de comandos para Angular.

Para comprobar el estado de la aplicación durante el desarrollo, Angular CLI nos ofrece un comando especial, *ng serve --open*, que permite construir la aplicación e iniciar un servidor de desarrollo para poder acceder a la aplicación a través del navegador en una dirección aleatoria (*localhost:4200*).

```
C:\Users\rezus\Desktop\AngularEmp\angular>ng serve
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
Date: 2019-05-18T09:35:27.118Z
Hash: 79ede6d1fb18eac03837
Time: 47022ms
chunk {es2015-polyfills} es2015-polyfills.js, es2015-polyfills.js.map (es2015-polyfills) 284 kB [initial] [rendered]
chunk {main} main.js, main.js.map (main) 207 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 236 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.08 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 180 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 10.1 MB [initial] [rendered]
@wdm@: Compiled successfully.
^C¿Desea terminar el trabajo por lotes (S/N)?
```

Figura 29. Servidor de desarrollo proporcionado por Angular CLI.

Una vez finalizado el desarrollo es necesario **unificar** la parte del Front-End con el Back-End, introduciendo ambas partes en el servidor Node.js + express, de forma que cuando el cliente acceda a la dirección de la aplicación (En este caso *localhost:3000*) el servidor responda con la SPA construida con Angular, y tras esto, todas las peticiones posteriores se hagan a la API REST situada en el mismo servidor.

Para **construir la aplicación** se emplea el comando de Angular CLI: *ng build --prod*. Este comando generara en el directorio dist una carpeta con la SPA construida, que copiaremos al directorio raíz de nuestro servidor express con la API REST.

Estableceremos ahora en el servidor express una regla para las peticiones *localhost:3000*, de modo que devuelva la SPA al cliente. La SPA no es más que un fichero *index.html* con referencias a código JavaScript y a hojas de estilos. Debemos por tanto responder a dichas peticiones con el fichero *html* correspondiente, siendo necesario activar el envío de contenido estático en el servidor.

En resumidas cuentas, el proceso necesario para obtener la SPA a través del servidor express es el mostrado en la siguiente figura:

```
var index = "http://localhost:3000/public/index.html"

app.use(express.static('public'));
app.use('/public', express.static('public'));

app.get('/', (req, res) => {
  apiProxy.web(req, res, {target: index});
})

app.listen(port, () => {
  console.log(`App running on port ${port}.`)
})
```

Figura 30. Envío de la SPA desde el servidor Node.js + Express

10. PROBLEMAS ENCONTRADOS

En este apartado comentaremos los principales problemas encontrados y las soluciones que se han adoptado para solventarlos.

10.1. Problemas de interacción con el mapa

A la hora de crear el mapa con OpenLayers, es necesario obtener desde GeoServer la capa base de todos los elementos con los que se interactuará en el mapa. Se trata de una capa que está formada por un grupo de capas base, las cuales representan las distintas tablas geográficas de nuestra base de datos, es decir, los distintos elementos a gestionar (Parques, jardines...).

Este grupo de capas se importa en el mapa como una **imagen WMS**, por lo que cada vez que se realiza un movimiento en el mapa, geoserver nos devuelve un png con la imagen correspondiente de nuestro grupo de capas adecuándola al mapa.

Nuestra aplicación busca poder interactuar con el mapa y sus elementos mediante clics de ratón. Por ello, al pulsar sobre cualquier elemento del mapa se obtiene información de la capa pulsada a partir de una llamada Ajax a una URL generada con la proyección, resolución, formato de la respuesta y coordenada del evento clic mediante el método `getFeatureInfoUrl`.

Una vez recibimos la respuesta al llamar a dicha URL, obtenemos las coordenadas de la Feature recibida y añadimos su geometría a la capa WFS del mapa, centrándolo con las coordenadas de ésta.

Esta solución generó una gran cantidad de problemas a la hora de traer una Feature, debido a que las geometrías de los elementos se almacenan en proyección EPSG:25830¹, adecuándose a Navarra, mientras que el mapa de OpenLayers sólo incluye por defecto las definiciones de las proyecciones EPSG:3857 (Siendo esta la proyección por defecto) y EPSG:4326. Esta **discordancia de proyecciones** impedía que las features se dibujasen en el mapa a pesar de que la petición se llevaba a cabo satisfactoriamente.

¹ SRID: Identificador de Referencia Espacial, es un identificador estándar único que hace referencia a un Sistema de Coordenadas concreto.

EPSG: Organismo que desarrolló un repositorio de parámetros geodésicos que contiene información sobre sistemas de referencia antiguos y modernos, proyecciones cartográficas y elipsoides de todo el mundo.



Figura 31. Proyección EPSG:25830 (Incluye Navarra).

Tratamos de solucionarlo **adecuando el mapa a la proyección** EPSG:25830. Para ello era necesario instalar PROJ4, un transformador de coordenadas genérico, definir nuestra proyección con unos parámetros específicos (+proj=utm +zone=30 +ellps=GRS80 +towgs84=0,0,0,0,0,0 +units=m +no_defs) y aplicarla a la vista del mapa.

A pesar de parecer un proceso sencillo, en la práctica da infinidad de problemas pues OpenLayers no está directamente preparado para trabajar en este tipo de proyecciones. Por ello nos vimos obligados a cambiar de estrategia, esta vez con un resultado satisfactorio:

Al declarar una capa en geoserver, añadimos un Sistema de Referencia de coordenadas nativo, en nuestro caso el EPSG:25830, que es el EPSG con el que las capas fueron originalmente dibujadas en QGIS. Sin embargo, podemos hacer que esa capa aparezca disponible en otra proyección, **añadiendo en la especificación de SRS Declarado el EPSG:3857**, que es el que nos interesa para nuestro mapa de OpenLayers. Esto hará que al pedir nuestra capa jardín, ésta se reprojete al sistema de coordenadas declarado, haciendo de la reproyección un proceso transparente para el programador y el uso de OpenLayers mucho más eficiente y sencillo.

10.2. Problemas de CORS

Nuestro cliente a la hora de utilizar la aplicación requiere de información de distintos servidores como pueden ser IDENA para la obtención del mapa de fondo, Geoserver para la obtención de capas, o el propio servidor que contiene la API REST. Estos servidores tienen distintas direcciones, lo que hace que los recursos que llegan al navegador sean de distintos orígenes.

Por seguridad los navegadores **bloquean solicitudes HTTP de origen cruzado** iniciadas dentro de un script para evitar suplantaciones de identidad, robos de información, etc., haciendo que nuestra aplicación no funcione correctamente.

Para evitar dichos bloqueos, ha sido necesario crear un **proxy inverso**, de forma que todas las peticiones del cliente vayan a nuestro servidor (localhost:3000) y en función de la petición realizada se le redirija al servidor correspondiente.

Para implementar esta función, express nos ofrece una extensión *http-proxy* mediante la cual podemos redirigir las peticiones a los destinos más adecuados.

Definimos los servidores externos que necesitamos en variables.

```
var serverCapas = "http://localhost:8080/geoserver/pyj/wms"  
var serverMapa = "https://idena.navarra.es/ogc/wms"
```

Figura 32. Declaración de servidores externos

Agregamos la extensión del proxy:

```
var apiProxy = httpProxy.createProxyServer();
```

Figura 33. Creación de un Proxy inverso en Node.js.

Y ahora redirigimos las peticiones que nos interesan de la siguiente forma:

```
app.get("/capas*", function(req, res) {  
  apiProxy.web(req, res, {target: serverCapas});  
});  
  
app.get("/wfs*", function(req, res) {  
  apiProxy.web(req, res, {target: serverWFS});  
});  
  
app.get("/idena*", function(req, res) {  
  apiProxy.web(req, res, {target: serverMapa});  
});
```

Figura 34. Redirección de peticiones según sus parámetros.

De esta manera evitamos bloqueos por parte de la mayoría de los navegadores, siendo despreciable la pérdida de tiempo ocasionada por el proxy.

10.3. Obtención de datos geográficos y almacén en la base de datos.

Para la obtención de datos, dado que el ayuntamiento del valle de Aranguren no dispone de ninguna información de tipo digital, nos vemos obligados a ofrecerles un **sistema de dibujo espacial** que les permita añadir la información de forma fácil e intuitiva.

Elegimos el software QGIS, al que se le añade fácilmente la extensión PostGIS, lo que permite incluir tablas con campos de tipo espacial de PostgreSQL al editor de mapas y dibujar y añadir datos de manera sencilla. QGIS permite rellenar la columna geométrica de una tabla dibujando sobre el mapa, y una vez acabado el dibujo es necesario rellenar el resto de los campos de la tabla a mano antes de introducir en la base de datos la tupla generada.

Esto en nuestro caso supone un gran problema, ya que el modelo de datos hace que todos los elementos espaciales hereden de la tabla *Elemento* que contiene los campos *nombre* y *lugar*. Con este modelo de datos, si dibujamos un elemento hijo en QGIS, esos campos de la tabla padre quedan vacíos y por lo tanto pierde el sentido la herencia.

A demás, todos los elementos hijo tienen una columna que es clave primaria (Y extranjera del elemento padre) y que debe ser única. Al ser un campo más de la tabla, QGIS nos obligará a rellenarlo una vez finalizado el dibujo.

Todo esto hace que la herencia pierda el sentido y que la introducción de datos sea muy lenta y pesada, siendo una solución inviable en la práctica.

Para solucionar estos impedimentos, implantamos una solución eficiente basada en crear en la base de datos **una vista por cada elemento hijo** con únicamente los campos que queremos rellenar al dibujar cada elemento (nombre, especie, edad...) y la geometría, independientemente del modelo de datos de la base de datos.

Ahora en QGIS es posible dibujar esta vista de manera transparente a la herencia, como si de una tabla normal se tratase. Una vez acabado el dibujo y rellenados los datos, mediante un **trigger INSTEAD OF INSERT** sobre la vista descrita, se obtienen los datos necesarios y se realizan 2 inserts independientes en *elemento* y *árbol* con sus campos correspondientes, obteniendo la clave

común a partir de una secuencia aleatoria, resultando en una solución robusta y eficaz:

Ejemplo al dibujar un parque en QGIS:

```
-- Creación de la función a ejecutar por el trigger
CREATE FUNCTION produccion.trig_parque() RETURNS trigger AS $trig_parque$

    DECLARE i bigint;
    DECLARE idl bigint;

    BEGIN
        -- Obtenemos el id común y único
        i:= (SELECT nextval('produccion.secuenciaElementos'));
        -- Obtenemos el id de la ubicación del nuevo elemento
        idl:=(select idlugar from produccion.lugares l where
ST_CONTAINS(l.geom,NEW.geom));
        -- Insertamos el padre
        INSERT INTO produccion.elementos(idelemento,nombre,idlugar)
VALUES (i,NEW.nombre,idl);
        -- Insertamos el hijo (Calculando el área)
        INSERT INTO produccion.parques(idelemento,area,geom) VALUES
(i,(select ST_AREA(NEW.geom)*POWER(0.3048,2) As sqm),NEW.geom);

        RETURN NULL;
    END;
$trig_parque$ LANGUAGE plpgsql;

-- Creación del trigger
CREATE TRIGGER trig_parque INSTEAD OF INSERT OR UPDATE ON
produccion.v_parques
FOR EACH ROW EXECUTE PROCEDURE produccion.trig_parque();
```

Figura 35. Creación de función y trigger instead of insert.

Para rellenar en cada *elemento* la ubicación a la que pertenece, simplemente hacemos una consulta que nos devuelve el identificador de dicha ubicación, y en la que establecemos como condición que la geometría del elemento esté contenida en la geometría de una de las ubicaciones almacenadas, mediante la función de PostGIS: ST_CONTAINS.

Para rellenar el área de los elementos poligonales hacemos uso de la función que ofrece PostGIS: ST_AREA, y las longitudes las obtenemos a partir de la función ST_LENGTH.

11. TECNOLOGÍAS

En esta sección se recopilan las principales tecnologías empleadas para el correcto desarrollo del proyecto, incluyendo:

Programas de visualización de datos geoespaciales, gestor de bases de datos con capacidad geoespacial, servidores que ofrecerán los datos de la aplicación, lenguajes empleados en todo el proceso de desarrollo del proyecto.

11.1. NodeJS + Express

NodeJS es un entorno de ejecución de JavaScript. Está principalmente orientado a eventos asíncronos, lo que lo convierte en un entorno ideal para construir aplicaciones en red escalables del lado del servidor.

Express se trata de una infraestructura de aplicaciones web Node.js que proporciona un conjunto sólido de características para la creación de aplicaciones web. Permite trabajar con el protocolo http y disponer de sistemas de rutas. Comúnmente es denominado el servidor estándar de facto para Node.js

La combinación de ambas tecnologías permite construir de forma sencilla un servidor programado en JavaScript que disponga de una API REST, o lo que es equivalente, un conjunto de funciones sobre las que se pueden realizar solicitudes y obtener respuestas a través del protocolo HTTP en nuestro caso en formato JSON, aunque también es posible realizarlo en XML.

Cabe destacar que ambas tecnologías son OpenSource bajo licencia del MIT.

11.2. Angular

Angular es una plataforma o framework para el desarrollo de aplicaciones web que siguen el paradigma SPA (Single-Page application, aplicación web que carga todo el contenido de una vez, y los recursos necesarios se cargan de manera dinámica sin necesidad de recargar la página) desarrollado en TypeScript. Angular basa su comportamiento en la creación de componentes, que no son más que clases con variables y métodos que se relacionan con la vista. Dispone de licencia de código abierto y es mantenido por Google.

11.3. QGIS

QGIS se trata de una herramienta de escritorio con la que es posible visualizar y editar datos geoespaciales. Incluye soporte para la extensión espacial de

PostgreSQL, PostGIS, lo que permite poblar los campos geoespaciales de una base de datos simplemente importando la tabla sobre la que se introducirá la geometría y dibujando sobre un mapa las distintas formas a introducir.

QGIS es software libre y opera bajo licencia GNU.

11.4. Geoserver

Geoserver es un servidor que permite compartir datos geoespaciales. Dado que la interoperabilidad es una de sus bases, está diseñado para publicar datos con los principales estándares abiertos y es compatible con las especificaciones WMS, WCS y WFS.

Admite PostGIS como fuente de entrada de datos, lo que permite generar un servicio de los datos geoespaciales almacenados en una base de datos. Soporta peticiones GET y POST, configuración de estilos de capas y como formatos de salida soporte JPEG, PNG, GIF, SVG y GML.

Geoserver está distribuido bajo la licencia GNU, General Public License.

11.5. PostgreSQL

PostgreSQL es un sistema de gestión de base de datos objeto-relacional (orientado a objetos). Permite una alta concurrencia dado que mientras un proceso escribe en una tabla, otros procesos pueden acceder a la misma sin bloqueos, manteniendo la consistencia.

Incluye una herramienta de administración de las bases de datos denominada pgAdmin, que permite realizar operaciones de gestión, lectura y escritura sobre la base de datos directamente a través de la interfaz de usuario en un navegador.

PostgreSQL es un proyecto de código abierto publicado bajo la licencia PostgreSQL, similar a la MIT.

11.6. PostGIS

PostGIS es una extensión espacial para las bases de datos relacionales orientadas a objetos de PostgreSQL. Añade soporte para objetos geográficos permitiendo realizar consultas espaciales en SQL, convirtiendo la base de datos en una base de datos de tipo espacial para su posterior utilización en sistemas de información geográfica (GIS).

Es una herramienta certificada por el OGC y ha demostrado ser superior a las extensiones geográficas de MySQL y a la versión geográfica de la base de datos de Oracle.

PostGIS se publica bajo licencia pública general de GNU.

11.7. Angular Material

Librería de componentes web con un diseño Material Design, permite crear interfaces de usuario modernas, escalables, adaptativas, rápidas y consistentes sobre proyectos de Angular. Esto posible a través de objetos predefinidos sobre los que se puede añadir comportamiento y control sin necesidad de implementarlos desde cero. Sigue la normativa de diseño *Material Design* creada por Google para Android y para sus aplicaciones.

11.8. Lenguajes de programación.

Los lenguajes empleados en el desarrollo del proyecto ha sido los siguientes:

11.8.1. JavaScript

JavaScript es un lenguaje de programación interpretado (no requiere de compilador), orientado a objetos, débilmente tipado y dinámico. Se emplea principalmente en el lado del cliente permitiendo mejoras en la interfaz y páginas web dinámicas, aunque también existe en el lado del servidor, como es nuestro caso, en el que el API REST está programado únicamente con JavaScript.

11.8.2. HTML

HTML es el lenguaje de marcado para la elaboración de páginas web. Define una estructura básica y un código para la definición del contenido de la web. Esta estructura en formato de texto es interpretada por el navegador, que es el que se encarga de unir los elementos y mostrar la visualización final de la página.

Es el estándar a cargo del W3C para la visualización de páginas web y ha sido adoptado por todos los navegadores.

11.8.3. CSS

Es un lenguaje de diseño gráfico escrito en lenguaje de marcado para definir la presentación de un documento HTML. Permite mejorar la accesibilidad y usabilidad de la web, a la par de reducir la complejidad y repetición del código en la estructura del documento HTML.

11.8.4. TypeScript

TypeScript es un lenguaje de programación que extiende de JavaScript y añade tipado estático y objetos basados en clases. Dado que extiende de JavaScript, cualquier código JavaScript funcionará sin problemas. Sin embargo, no es un lenguaje interpretado y requiere de compilador, que traducirá el código a JavaScript original.

12. MANUAL DE USUARIO

12.1. Instalación de PostgreSQL + PostGIS

Para la instalación de PostgreSQL accedemos a la web oficial y descargamos el instalador en este caso para Windows x86-64:

<https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>

Para el desarrollo del proyecto se ha empleado PostgreSQL 11.1. Ejecutando el instalador seguiremos los siguientes pasos:

- En la selección de componentes es importante que estén seleccionados PostgreSQL Server, pgAdmin 4 (Será nuestra interfaz para la gestión de la base de datos) y Stack Builder (Nos permitirá instalar PostGIS más adelante)
- Seleccionaremos la ruta de instalación deseada.
- Seleccionaremos el directorio de almacenamiento de las bases de datos.
- Añadimos la contraseña de acceso del administrador, se requerirá para la gestión de la base de datos
- El resto de los valores se dejarán en predeterminado ya que su configuración no es relevante.

Una vez instalado PostgreSQL, y antes de cerrar el instalador, Stack Builder nos permitirá descargar programas adicionales. En ese momento seleccionaremos PostGIS para disponer de extensión geográfica. Tras descargarse e instalarse tendremos disponible nuestra base de datos espacial, accesible a través de pgAdmin 4 en el navegador.

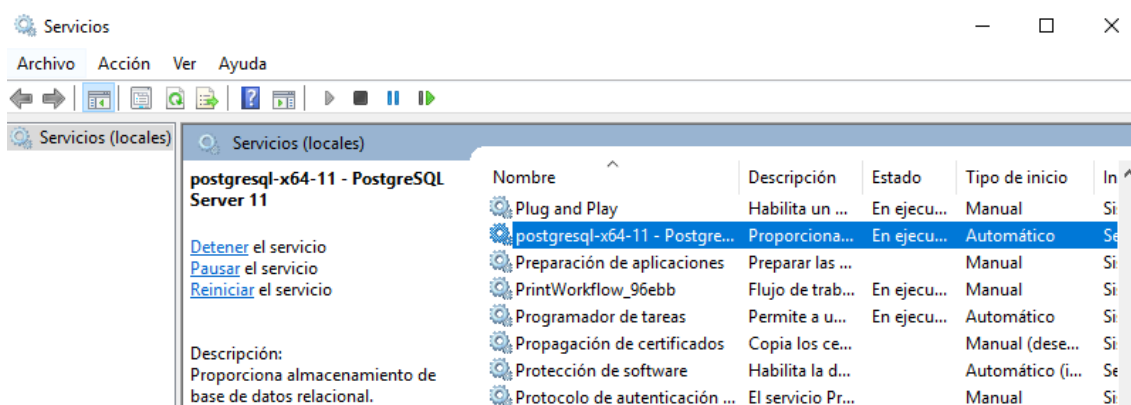


Figura 36. Servicio en ejecución del servidor de bases de datos.

Si ejecutamos services.msc en Windows podremos comprobar que la base de datos está activa y en caso de fallos podremos reiniciarla desde esta ubicación.

12.2. Instalación de GeoServer

La instalación de GeoServer se lleva a cabo desde el siguiente enlace:

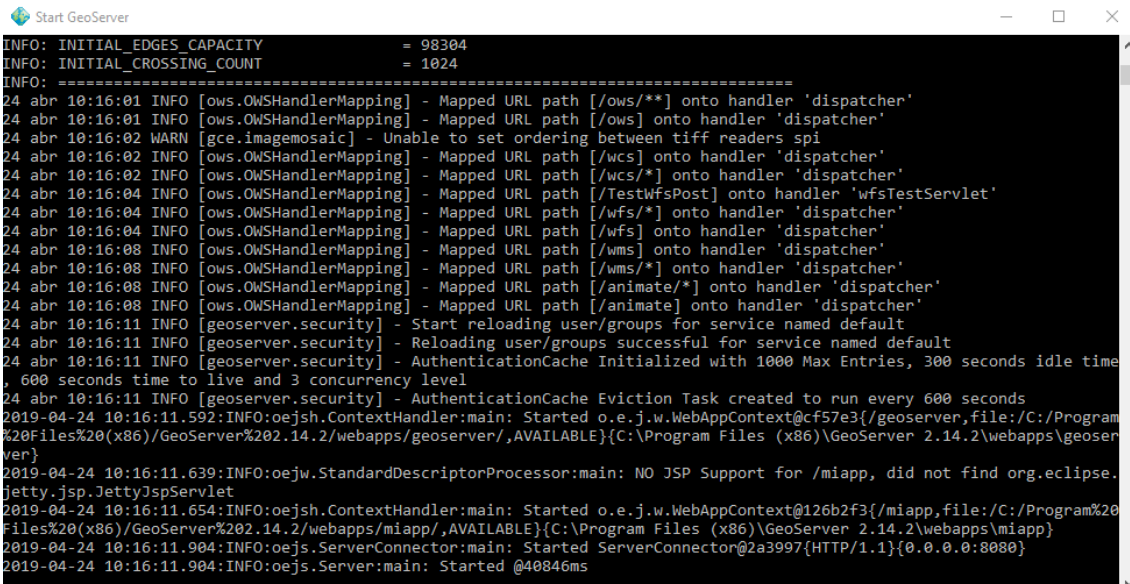
<http://geoserver.org/download/>

Para el desarrollo del proyecto se ha empleado la versión de GeoServer 2.14.2. Al seleccionar el instalador para Windows se descargará un fichero .exe que será necesario ejecutar como administrador para instalar GeoServer.

Los pasos para la instalación son los siguientes:

- Seleccionamos la carpeta de instalación de GeoServer
- Seleccionamos la ruta al Java Runtime Environment (JRE)
- Seleccionamos el puerto en el que trabajará GeoServer (En nuestro caso el 8080)
- Seleccionamos la opción *Run Manually*. Esto hará que tengamos que ejecutar GeoServer cada vez que queramos utilizarlo, lo que evitará que el ordenador consuma recursos innecesarios y aumentará el nivel de seguridad. También es posible instalarlo como servicio, al igual que hicimos con PostgreSQL.
- Hacemos clic en *instalar* y esperamos a que concluya la instalación.

Una vez instalado dispondremos de un icono de nombre *Start Geoserver* mediante el cual arrancaremos el servidor, que será accesible desde la dirección localhost:8080/Geoserver de un navegador.



```
Start GeoServer
INFO: INITIAL_EDGES_CAPACITY = 98304
INFO: INITIAL_CROSSING_COUNT = 1024
INFO: =====
24 abr 10:16:01 INFO [ows.OWSHandlerMapping] - Mapped URL path [/ows/**] onto handler 'dispatcher'
24 abr 10:16:01 INFO [ows.OWSHandlerMapping] - Mapped URL path [/ows] onto handler 'dispatcher'
24 abr 10:16:02 WARN [gce.imagemosaic] - Unable to set ordering between tiff readers spi
24 abr 10:16:02 INFO [ows.OWSHandlerMapping] - Mapped URL path [/wcs] onto handler 'dispatcher'
24 abr 10:16:02 INFO [ows.OWSHandlerMapping] - Mapped URL path [/wcs/*] onto handler 'dispatcher'
24 abr 10:16:04 INFO [ows.OWSHandlerMapping] - Mapped URL path [/TestWfsPost] onto handler 'wfsTestServlet'
24 abr 10:16:04 INFO [ows.OWSHandlerMapping] - Mapped URL path [/wfs/*] onto handler 'dispatcher'
24 abr 10:16:04 INFO [ows.OWSHandlerMapping] - Mapped URL path [/wfs] onto handler 'dispatcher'
24 abr 10:16:08 INFO [ows.OWSHandlerMapping] - Mapped URL path [/wms] onto handler 'dispatcher'
24 abr 10:16:08 INFO [ows.OWSHandlerMapping] - Mapped URL path [/wms/*] onto handler 'dispatcher'
24 abr 10:16:08 INFO [ows.OWSHandlerMapping] - Mapped URL path [/animate/*] onto handler 'dispatcher'
24 abr 10:16:08 INFO [ows.OWSHandlerMapping] - Mapped URL path [/animate] onto handler 'dispatcher'
24 abr 10:16:11 INFO [geoserver.security] - Start reloading user/groups for service named default
24 abr 10:16:11 INFO [geoserver.security] - Reloading user/groups successful for service named default
24 abr 10:16:11 INFO [geoserver.security] - AuthenticationCache Initialized with 1000 Max Entries, 300 seconds idle time
, 600 seconds time to live and 3 concurrency level
24 abr 10:16:11 INFO [geoserver.security] - AuthenticationCache Eviction Task created to run every 600 seconds
2019-04-24 10:16:11.592:INFO:oejsh.ContextHandler:main: Started o.e.j.w.WebAppContext@cf57e3{/geoserver,file:/C:/Program
%20Files%20(x86)/GeoServer%202.14.2/webapps/geoserver/,AVAILABLE){C:\Program Files (x86)\GeoServer 2.14.2\webapps\geoser
ver}
2019-04-24 10:16:11.639:INFO:oejw.StandardDescriptorProcessor:main: NO JSP Support for /miapp, did not find org.eclipse.
jetty.jsp.JettyJspServlet
2019-04-24 10:16:11.654:INFO:oejsh.ContextHandler:main: Started o.e.j.w.WebAppContext@126b2f3{/miapp,file:/C:/Program%20
Files%20(x86)/GeoServer%202.14.2/webapps/miapp/,AVAILABLE){C:\Program Files (x86)\GeoServer 2.14.2\webapps\miapp}
2019-04-24 10:16:11.904:INFO:oejs.ServerConnector:main: Started ServerConnector@2a3997{HTTP/1.1}{0.0.0.0:8080}
2019-04-24 10:16:11.904:INFO:oejs.Server:main: Started @40846ms
```

Figura 37. Consola de arranque de GeoServer

12.3. Instalación de QGIS y conexión con PostgreSQL

La instalación de QGIS se lleva a cabo desde el siguiente enlace:

<https://qgis.org/es/site/forusers/download.html>

Para el desarrollo del proyecto se ha empleado la versión de QGIS 3.4.4. Al seleccionar el instalador para Windows 64bits independiente de los paquetes de OSGeo4W se descargará un fichero .exe que será necesario ejecutar como administrador para instalar QGIS.

El proceso de instalación no requiere de ningún comentario adicional ya que los valores por defecto son más que suficientes para el uso que daremos a continuación.

Conexión con PostgreSQL: Esta conexión se realiza con la extensión geográfica de PostgreSQL, PostGIS y se lleva a cabo de la siguiente manera.

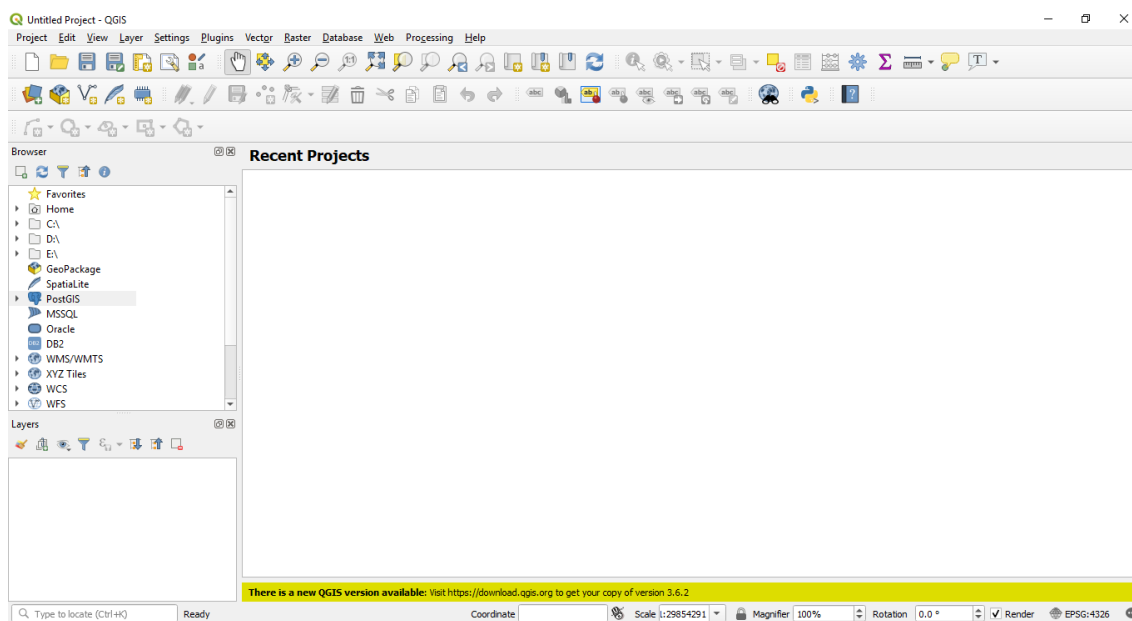


Figura 38. Página de inicio del software QGIS.

En el buscador del lado izquierdo de QGIS aparecerá el icono de PostgreSQL con el texto *PostGIS*. Haremos clic derecho – new connection y rellenaremos los datos con la información de nuestra base de datos en PostgreSQL:

Los **campos críticos** para la conexión serán los siguientes:

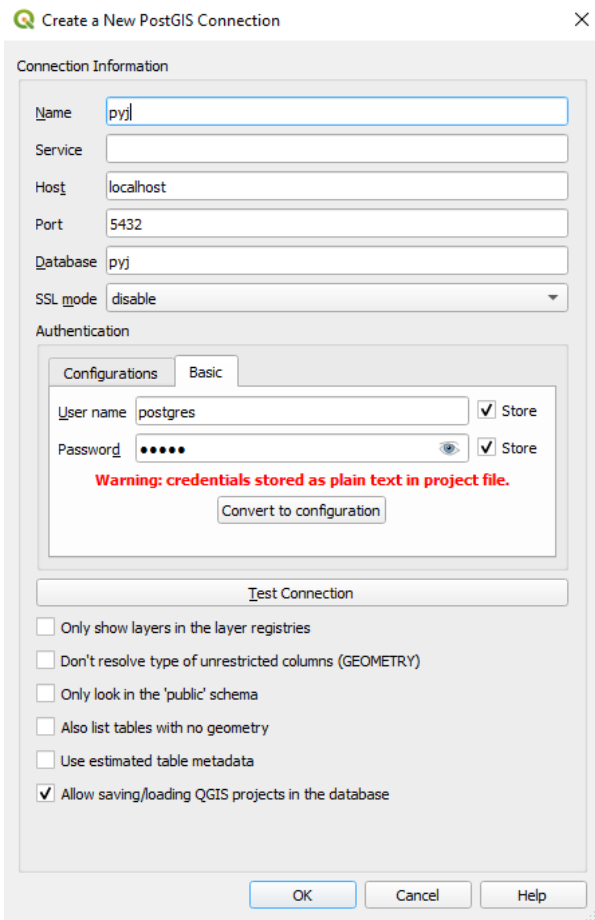


Figura 39. Ventana de conexión de QGIS con PostGIS

Host: PostgreSQL está operando como servicio local, luego añadiremos la dirección localhost.

Port: Tal y como se ha configurado en la instalación, el puerto en el que opera PostgreSQL es el 5432

Database: Nombre de la base de datos que queremos vincular a QGIS.

User name / Password : Credenciales de acceso del administrador. Son las mismas que se emplean en el acceso a la consola de administración pgAdmin 4.

Allow saving/loading QGIS projects in the database: Esta opción debe estar marcada para poder dibujar sobre QGIS y almacenar las geometrías en la base de datos PostgreSQL.

12.4. Dibujar geometrías en QGIS y almacenarlas en la base de datos

Una vez tenemos vinculado QGIS con PostGIS será posible dibujar directamente geometrías sobre las tablas de la base de datos que dispongan de una columna de tipo geometry.

Pero para poder tener una referencia de fondo a la hora de dibujar, añadimos una conexión WMS en el explorador de la misma forma que añadimos la conexión PostGIS, y la rellenamos con los datos de IDENA:

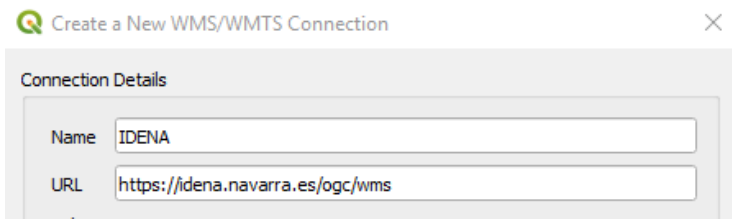


Figura 40. Conexión QGIS con servicio WMS.

Una vez tenemos todas las capas necesarias, las arrastramos desde el explorador a la sección *layers* haciendo doble clic sobre ellas. Importaremos las capas sobre las que queremos añadir información y la capa Ortofoto de IDENA.

Para dibujar sobre una capa importada debemos seleccionarla y hacer clic sobre el icono:



Seguidamente, podremos dibujar haciendo clic sobre el icono:



En este caso el dibujo será un polígono. Para dibujar simplemente basta con ir haciendo clic izquierdo en los puntos que generarán el polígono, y una vez terminado, haciendo clic con el botón derecho nos aparecerá la información alfanumérica del resto de campos de la tabla a dibujar.

Tras rellenarlos y pulsar OK tendremos los datos de una nueva entidad pero no estarán todavía almacenados en la base de datos. Esto se logrará haciendo clic el icono de los lápices rojos, seleccionando la opción *Save For Selected Layers*

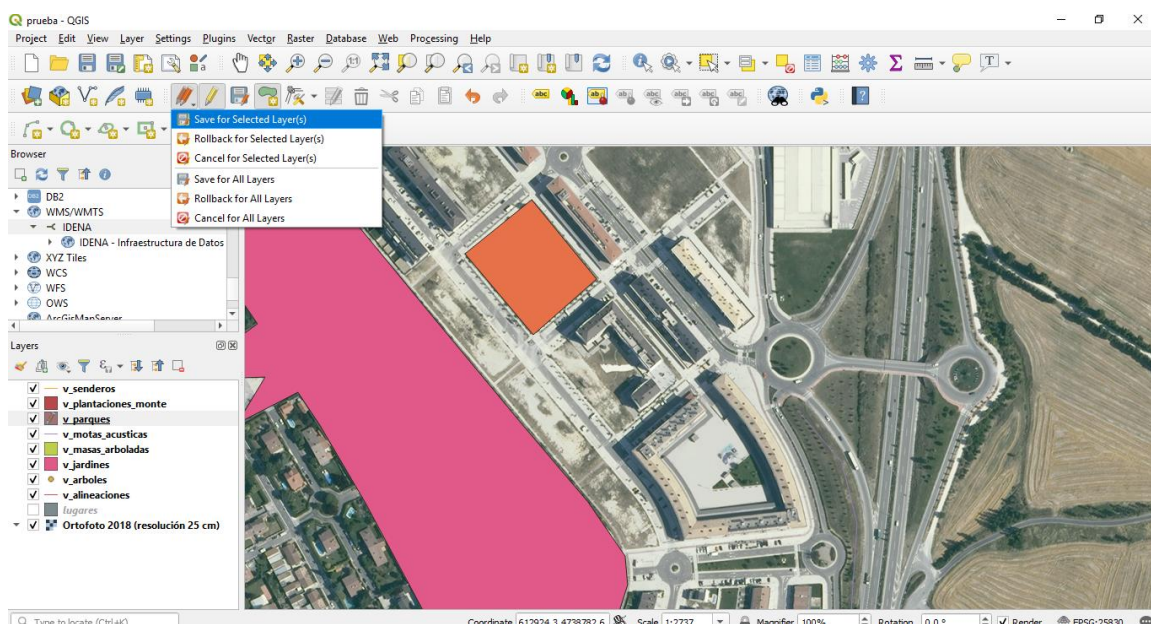


Figura 41. Guardado de capas editadas en QGIS.

12.5. Conexión Geoserver - PostgreSQL

La conexión de GeoServer y PostgreSQL se realiza desde la consola de administración de GeoServer, en nuestro caso en la dirección local localhost:8080/Geoserver.

Para poder agregar un almacén de datos será necesario disponer de un espacio de trabajo, cuya creación no requiere de ninguna anotación especial.

Una vez lo tenemos, desde la sección *Almacenes de datos* seleccionaremos la opción de Agregar un nuevo almacén, y rellenaremos los datos tal y como se indica a continuación:

Parámetros de conexión

host *	<input type="text" value="localhost"/>
port *	<input type="text" value="5432"/>
database	<input type="text" value="pyj"/>
schema	<input type="text" value="produccion"/>
user *	<input type="text" value="postgres"/>
passwd	<input type="password" value="•••••"/>

Los parámetros de conexión son los campos requeridos para realizar la conexión. Al igual que en QGIS, introduciremos la dirección localhost y puerto 5432 de PostgreSQL, el nombre de la base de datos a vincular, el esquema y las claves de acceso.

El resto de los parámetros son opcionales y darán nombre al almacén y al resto de componentes.

Figura 42. Parámetros de la conexión GeoServer - PostgreSQL



Figura 43. Capas disponibles en GeoServer tras la conexión con PostgreSQL.

Si la conexión se realiza de manera exitosa, a la hora de añadir nuevas capas, seleccionando el almacén de trabajo introducido anteriormente, obtendremos todas las capas geográficas de nuestra base de datos que GeoServer podrá servir posteriormente a OpenLayers para su representación en el mapa de la aplicación.

13. VALORACIONES Y CONCLUSIONES

En este capítulo analizaremos las conclusiones obtenidas tras la implementación de la aplicación. Se realizarán valoraciones a nivel personal y líneas futuras del desarrollo del proyecto.

13.1. Valoraciones

El desarrollo de esta aplicación me ha servido para conocer el proceso de desarrollo de un proyecto desde la especificación de requisitos hasta las pruebas finales.

La elaboración de una página web que sigue el paradigma Single Page Application y su implementación con las últimas tecnologías como Angular y Node.js + Express ha sido de gran interés, pues era un sector que prácticamente desconocía.

El manejo de datos e información de tipo espacial, y la necesidad de operar con ellos empleando diferentes funciones geoespaciales y manejando distintas proyecciones, así como la necesidad de almacenarlos y dibujarlos ha ampliado enormemente mis conocimientos del área GIS (Geographic Information Systems), aspecto indudablemente positivo.

En líneas generales, *Aplicación Web-Mapping para la gestión de Parques y Jardines* ha sido un **proyecto muy completo y enriquecedor** a nivel de conocimientos técnicos, a nivel de desarrollo y de gestión de proyectos, y a nivel personal.

13.2. Conclusiones

Tras el desarrollo completo de la aplicación, ha quedado claro que se trata de un proyecto de **envergadura considerable** y es por ello que habría sido interesante disponer de más tiempo para poder profundizar completamente en todos los aspectos de su desarrollo.

Habría sido interesante disponer de un mayor contacto con el cliente ya que en un principio la idea original era implantarlo en el ayuntamiento del valle de Aranguren, siendo la fase de requisitos, reuniones con el cliente, puesta en marcha, integración en el entorno de cliente, pruebas, etc. un aspecto más a tener en cuenta durante todo el ciclo de vida del proyecto.

El tema de la aplicación y su desarrollo con tecnologías y paradigmas modernos han hecho que me haya implicado mucho en el proyecto a nivel de

formación y posteriormente en su implementación, por ello me gustaría continuar su desarrollo en un futuro, incluyendo nuevas funcionalidades que no han podido ser implementadas todavía y mejorando la estructura ya existente.

13.3. Líneas futuras

Algunas de las ramas en las que se podría trabajar en un futuro son las siguientes:

- Subida de imágenes: Tanto de partes de trabajo, incidencias, etc.
- Posibilitar el dibujado directamente en el mapa de OpenLayers para añadir nuevos elementos, en lugar de añadirlos desde QGIS.
- Incorporación de información geográfica directamente a través del mapa OpenLayers, prescindiendo así de QGIS.
- Diseño responsive: Adaptable a teléfonos móviles y tablets, para disponer de la aplicación en cualquier dispositivo y lugar.
- Añadir un control de acceso de distintos usuarios.
- Desarrollo de área ciudadana: Sección para la ciudadanía donde puedan indicar incidencias, fotografías, quejas...
- Añadir más información al mapa, no únicamente elementos del área de parques y jardines.
- Extender el servicio a otros departamentos (Limpieza, obras, carreteras).

14. BIBLIOGRAFÍA

<https://victorroblesweb.es/2018/01/31/configurar-acceso-cors-en-nodejs/>

<https://malcoded.com/posts/angular-backend-express>

https://es.wikipedia.org/wiki/Web_Map_Service

<https://angular.io/>

[https://es.wikipedia.org/wiki/Angular_\(framework\)](https://es.wikipedia.org/wiki/Angular_(framework))

<https://nodejs.org/es/>

<https://openlayers.org/>

<https://blog.logrocket.com/setting-up-a-restful-api-with-node-js-and-postgresql-d96d6fc892d8>

<https://malcoded.com/posts/angular-backend-express/>

<https://idena.navarra.es/Portal/Inicio>

https://www.alibabacloud.com/blog/building-a-restful-api-with-express-postgresql-and-node-using-es6_594137