Departamento de Ciencias da Computación e
Tecnoloxías da Información
Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑERÍA INFORMÁTICA
MENCIÓN EN SISTEMAS DE INFORMACIÓN

Aplicación Web de gestión de información geográfica

Estudiante: Manuel Ángel Ameneiros Perille
Dirección: Miguel Ángel Rodríguez Luaces

A Coruña, 14 de noviembre de 2019

A mis padres Ana y Manel, porque siempre han luchado por mí.

Agradecimientos

En primer lugar quería agradecer a mi tutor, Miguel, por la ayuda y los consejos que me brindó durante todo el proceso, permitiendo que el proyecto llegase a buen puerto.
A mi padre, porque lamentablemente falleció durante el transcurso de la carrera, y siempre tuvo fe en que lograría acabar la carrera.
A mi madre y hermano, porque siempre han estado ahí cuando más los necesitaba y siempre conté con su apoyo.
Y finalmente a Patricia, porque desde el momento en que nos conocimos siempre has estado a mi lado y me has apoyado en todo, y sin ti seguramente no habría tenido los ánimos para acabar la carrera.

Resumen

Actualmente la información geográfica está presente en el día a día de nuestra rutina, ya sea cuando usamos el GPS, al consultar el tiempo atmosférico, al consultar el censo de nuestro ayuntamiento, etc. Tal es la importancia de la información geográfica, que se estima que un 70% de los datos corporativos que existen en el mundo incluyen algún componente geográfico. Esta aplicación nace con el objetivo de simplificar el acceso y aumentar los tipos de visualización que se pueden aplicar sobre esta información geográfica, por ello además de mapas la aplicación permitirá visualizar los datos como tablas o gráficos variados.

La aplicación la puede usar cualquier usuario que no esté registrado, pero solo los usuarios administradores podrán almacenar información geográfica en la base de datos, crear indicadores a partir de esta información o crear a otros administradores.

Para llevar a cabo el proyecto, fue necesario en primer lugar realizar un estudio de las tecnologías que se iban a emplear en el trabajo, ya que la gran mayoría eran novedosas. A continuación se realizó un estudio de necesidades que debía satisfacer la aplicación y se obtuvo una lista de historias de usuario así como un modelo de datos. Después se realizaron unos prototipos de pantallas y se implementaron todas las historias de usuario. Y finalmente se terminó haciendo las pruebas de unidad para comprobar que todo funcionaba adecuadamente.

La aplicación web está compuesta por un cliente web creado con Vue.js, un servidor REST construido con Java y Geotools y un SGBD implementado sobre PostreSQL.

El trabajo de fin de grado se gestionó aplicando conceptos de Scrum pero sin implementar todas las características de Scrum. En concreto se emplearon historias de usuario para gestionar los requisitos y se usaron sprints para llevar a cabo la planificación y seguimiento del proyecto.

Abstract

Today the geographic information is present in our daily life, whether when we use the GPS, when checking the weather, when consulting the municipal census, etc. Such is the importance of geographic information, that an estimated 70 % of the corporate data that exists in the world include some geographical component. This application was born with the aim of simplifying access and increasing the types of visualization that can be applied to this geographic information, so in addition to maps, the application will allow the visualization of data as tables or varied graphs.

The application can be used by any user who is not registered, but only administrator users can store geographic information in the database, create indicators from this information or create other administrators.

To build the project, it was first necessary to conduct a study of the technologies that were to be used at work, since the vast majority were new technologies. Next, a needs assessment that the application had to satisfy was carry out and a list of user stories was obtained, as well a data model. Then some prototype screens were made and all user stories were implemented. And finally the unit tests were finished to verify that everything was working properly.

The web application is composed of a web client created with Vue.js, a REST server built with Java and Geotools and a DBMS implemented on PostreSQL.

The final degree project was managed by applying Scrum concepts but without implementing all the Scrum features. Specifically, user stories were used to manage the requirements and sprints were used to carry out the planning and monitoring of the project.

Palabras clave:

- Shapefile
- Leaflet
- Vue.js
- Postgis
- Indicador geográfico
- Geotools

Keywords:

- Shapefile
- Leaflet
- Vue.js
- Postgis
- Geographical indicator
- Geotools

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En este capítulo se hablará de la motivación principal para la realización de este proyecto y los objetivos que ha de cumplir para realizarse con éxito.

1.1 Motivación

Hoy en día los Sistemas de Información Geográficos se encuentran en auge y cada vez se emplean en más disciplinas y campos como pueden ser navegadores GPS, estudio y análisis de áreas inundables, tratamiento de la información meteorológica, etc. Además debido al aumento de la popularidad y funcionalidad de Internet, las aplicaciones web son una de las maneras más comunes utilizadas para publicar información geográfica. Recientemente han aparecido numerosas librerías[1] que permiten la integración de funcionalidad de visualización de información geográfica en una aplicación web.

Sin embargo, no es fácil para un usuario sin conocimientos avanzados publicar información geográfica en la web. Mucha veces se encuentran con aplicaciones que solamente permiten crear mapas sobre una información geográfica predeterminada pero que no les permiten gestionar su propia información geográfica.

Por ello, el objetivo principal de este proyecto es utilizar esta tecnología existente e integrarla para crear una aplicación web que permita visualizar, gestionar información geográfica en distintos formatos y facilitar la publicación de información geográfica en la web. Por otra parte, el trabajo de fin de grado también tiene como objetivo permitir al estudiante profundizar en el uso de Java, las bases de datos, el desarrollo de aplicaciones web, y la integración de sistemas.

[1] https://leafletjs.com/index.html

1.2 Objetivos

A partir del objetivo principal que se ha descrito en la sección anterior, este trabajo de fin de grado debe alcanzar los siguientes objetivos específicos:

- **Publicar información geográfica mediante ficheros shapefile.** La aplicación permite a los usuarios convertir los ficheros shapefile [2] en tablas de la base de datos sobre los que se pueden aplicar consultas para gestionar los indicadores geográficos. Estos archivos incluyen toda la información geográfica que se necesita para poder crear los indicadores. Disponen de coordenadas, nombres de lugares y datos cuantitativos que sirven para darle valor a los indicadores. Actualmente existen shapefile de todo tipo, como pueden ser accidentes de tráfico, porcentaje de viviendas secundarias, entidades bancarias, centros comerciales, población, etc.

- **Crear indicadores geográficos.** A partir de unos datos geográficos obtenidos de una tabla de base de datos los administradores pueden crear indicadores geográficos que reflejen una de las columnas con valor numérico de la tabla. Se deben permitir distintos tipos de visualización y no solo como un mapa. Las visualizaciones que se permiten son en diagrama de barras, como una tabla, en diagrama de sectores y en diagrama de cajas y bigotes.

- **Visualizar indicadores geográficos.** Cualquier usuario que no sea administrador podrá visualizar la información reflejada en un indicador geográfico, pudiendo modificar el tipo de vista que quiere del indicador a voluntad.

- **Flexibilidad de la aplicación.** En un futuro la aplicación debería permitir incrementar los tipos de visualización o las funcionalidades sobre los indicadores fácilmente. Por ello la aplicación tiene que ser muy flexible.

[2] https://es.wikipedia.org/wiki/Shapefile

Capítulo 2

Fundamentos tecnológicos

2.1 Estado del arte

Basándonos en los objetivos del proyecto, se ha realizado una búsqueda de aplicaciones que proporcionen una funcionalidad similar o que a pesar de no ser puramente aplicaciones, su uso esté tan extendido que no se puedan pasar por alto. Las 3 principales serían Carto, Mapbox y Arcgis, siendo Mapbox la única que no es una aplicación de por sí.

- **Carto**. [2]. CARTO (anteriormente CartoDB) es una plataforma Software como Servicio (SaaS) de computación en nube que proporciona herramientas SIG y de mapeo web para visualizar en un navegador de web. La compañía está orientada como plataforma de Inteligencia de Localización dado que dispone de herramientas con capacidad para el análisis y visualización de datos y que no requiere experiencia previa en el desarrollo SIG. Los usuarios de CARTO pueden utilizar la plataforma libre de la compañía o desplegar su propia instancia del software de código abierto. CARTO se ofrece como un servicio freemium, donde se pueden crear cuentas libres para su uso con un límite de tamaño. Para cuentas más grandes hay un coste que incluye características avanzadas. CARTO es un software de código abierto construido sobre PostGIS y PostgreSQL. Utiliza herramientas Javascript en el front-end de aplicaciones web, Node.js en el back-end basado APIs, y bibliotecas de clientes. Ofrece dos propuestas principales:

  - CARTO Builder. Es la aplicación web donde los usuarios pueden gestionar datos, realizar análisis en el lado del usuario y diseñar mapas personalizados. CARTO Builder está enfocado a usuarios no desarrolladores y a principiantes que tengan acceso a herramientas geoespaciales avanzadas. En CARTO Builder los usuarios avanzados también pueden acceder una interfaz de web donde utilizar el lenguaje SQL para manipular los datos y trabajar también con CartoCSS, un lenguaje de marcado para la definición de semiología cartográfica, similar a CSS en el diseño

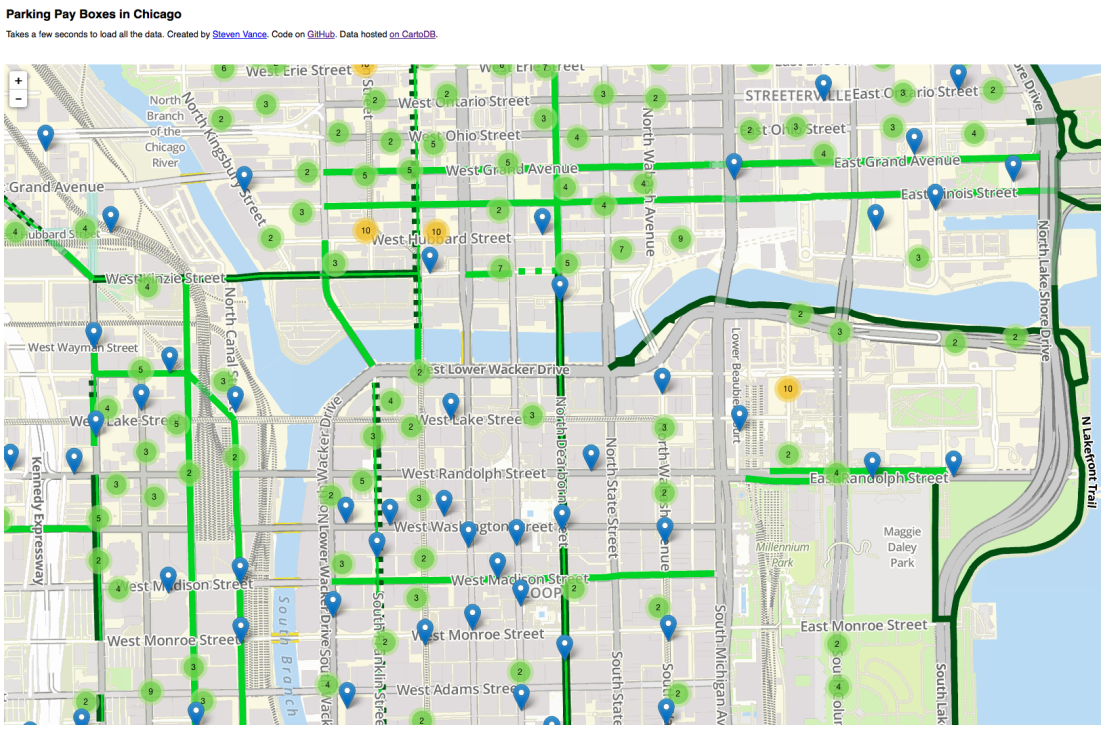


Figura 2.1: Mapa de Chicago que muestra los parquímetros creado con Carto

web.

– CARTO Engine. Un conjunto de API y bibliotecas para desarrolladores que facilitan construir interfaces de mapas personalizados y visualizaciones de los datos.

El precio de la aplicación premium está en los 199€ al mes. En Figura 2.1 se puede ver un mapa creado con Carto.

- **Mapbox.** [3]. Mapbox es un proveedor de mapas on-line realizados por encargo para páginas webs como Foursquare, Pinterest, Evernote, Financial Times, EThe Weather Channel y Uber Tecnologías. Desde 2010, ha expandido rápidamente su nicho de mapas por encargo como respuesta a la limitada elección que ofrecen otros proveedores como Google Maps y OpenStreetMap. Mapbox es el creador, o un colaborador significativo, de algunas bibliotecas de mapeo de código abierto y aplicaciones, entre ellas la especificación MBTiles, la cartografía TileMill IDE, la biblioteca de Javascript de Leaftlet, y el estilo de mapas y analizador sintáctico (parser) CartoCSS. Para poder usar los mapas de Mapbox hay que pagar por número de visualizaciones que hagan los usuarios.

- **Arcgis.** [4]. ArcGIS es el nombre de un conjunto de productos de software en el campo de los Sistemas de Información Geográfica o SIG. Producido y comercializado por ESRI, bajo el nombre genérico ArcGIS se agrupan varias aplicaciones para la captura, edición,

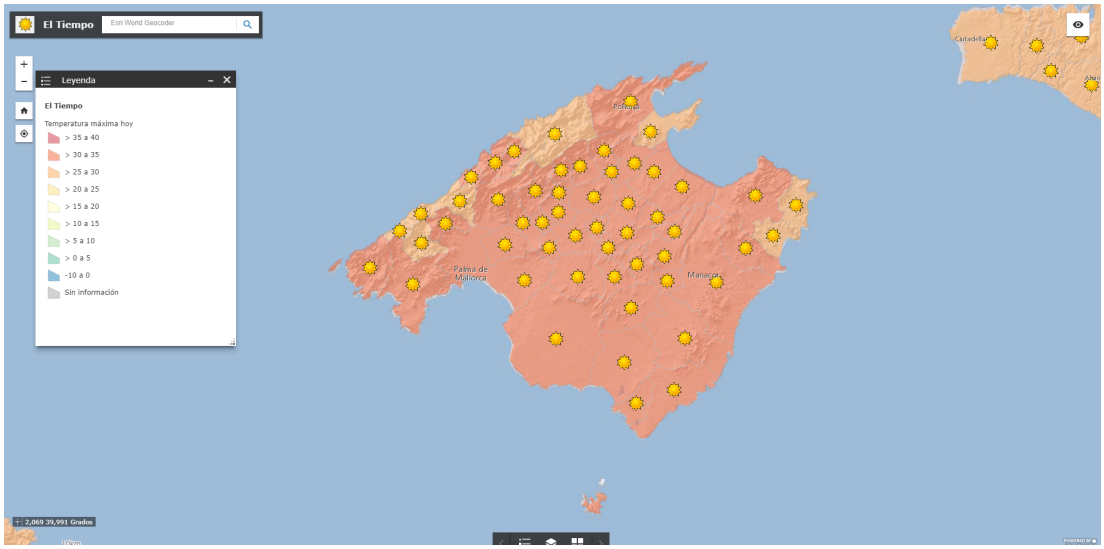


Figura 2.2: Mapa creado con Arcgis

análisis, tratamiento, diseño, publicación e impresión de información geográfica. Estas aplicaciones se engloban en familias temáticas como ArcGIS Server, para la publicación y gestión web, o ArcGIS Móvil para la captura y gestión de información en campo. Aunque dispone de una versión Pro de pago también está disponible la versión online gratuita.

Aunque cualquiera de las 3 aplicaciones proporcionan funcionalidades muy completas, ninguna ofrece tipos de visualización a mayores de los mapas, por ello, tomando como punto de partida la personalización que ofrecen a los mapas, pudiendo añadirle colores y distintos rangos a los mapas, se desarrollará una aplicación que permita crear mapas a partir de información geográfica y se centre en ofrecer una gran variedad de tipos de visualización para la información geográfica y permita publicar información geográfica de manera sencilla.

2.2 Tecnologías utilizadas

- **PostgreSQL.** [5] Es un sistema de gestión de bases de datos relacional orientado a objetos y de código abierto.
- **PostGIS.** [6]. Extensión de PostgreSQL para almacenar información geográfica en la base de datos.
- **Java.** [7] Es un lenguaje de programación de propósito general, concurrente, orientado a objetos. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo.

- **Hibernate.** [8] Es una herramienta de mapeo objeto-relacional (ORM) para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.
- **Geotools.** [9]. Kit de herramientas geoespaciales Java de código abierto para trabajar con datos vectoriales. Se emplea para almacenar los shapefiles en base de datos.
- **Spring.** [10] Es un framework para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java.
- **JavaScript.** [11] Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos,3 basado en prototipos, imperativo, débilmente tipado y dinámico.
- **Node.js.** [12] Plataforma de desarrollo para la creación de aplicaciones destinadas a la Web, orientada a redes y centrada en la velocidad y la escalabilidad.
- **Vue.js.** [13] Framework progresivo para construir interfaces de usuario. Su núcleo es bastante pequeño y se escala a través de plugins. Engloba en un mismo archivo HTML, CSS y JavaScript.
- **Bootstrap.** [14] Es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales.
- **Leaflet.** [15]. Librería JavaScript para visualizar mapas.

Capítulo 3

Metodología y planificación

En este capítulo se va a comentar la metodología de desarrollo seguida en este proyecto y la planificación realizada junto con sus desviaciones finales y costes.

3.1 Metodología de desarrollo

Para llevar a cabo el desarrollo del proyecto se optó por un desarrollo que adaptaba algunas de las prácticas Scrum. Scrum [16] es una metodología ágil que se emplea para trabajar en equipo a partir de iteraciones o Sprints. Su objetivo será controlar y planificar proyectos con un gran volumen de cambios de última hora. Se suele planificar por semanas. Al final de cada Sprint o iteración, se va revisando el trabajo validado de la anterior semana. En función de esto, se priorizan y planifican las actividades en las que invertiremos nuestros recursos en el siguiente Sprint. Teniendo en cuenta esto, en este proyecto se aplicaron dos de las prácticas de Scrum.

- Historias de usuario. Son descripciones, siempre muy cortas y esquemáticas, que resumen la necesidad concreta de un usuario al utilizar un producto o servicio, así como la solución que la satisface. En este proyecto se obtienen 24 historias de usuario. Al completar una historia de usuario se llevan a cabo las pruebas unitarias para determinar que todo funciona correctamente.
- Sprints fijos. Son cada uno de los ciclos o iteraciones que vamos a tener dentro de un proyecto Scrum. Nos permiten tener un ritmo de trabajo con un tiempo prefijado, siendo la duración habitual de un Sprint unas cuatro semanas, aunque lo que la metodología dice es que debería estar entre dos semanas y un máximo de dos meses. Para llevar a cabo este proyecto se establecen 4 sprints que contienen 5 historias de usuario y una duración de 3 semanas, y un sprint final con 4 historias de usuario y una duración de 2 semanas. Al finalizar cada sprint se lleva a cabo una reunión para determinar los objetivos del siguiente sprint.

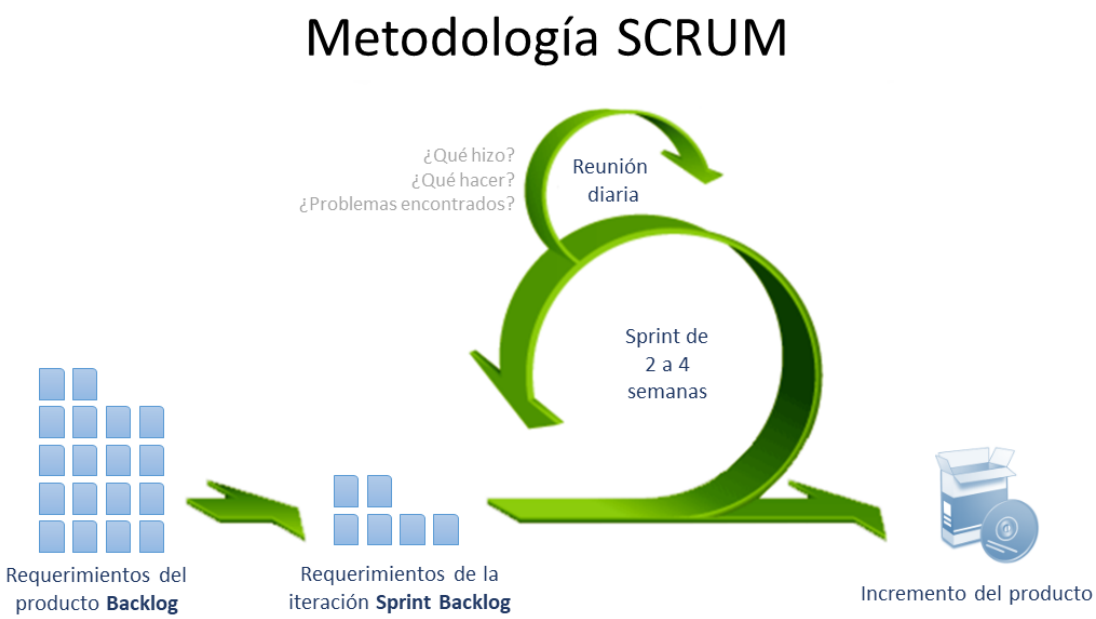


Figura 3.1: Metodología scrum.

En el proyecto se emplearon los sprints para el seguimiento del proyecto y las historias de usuario para el análisis, diseño e implementación.

A pesar de la utilidad y flexibilidad de Scrum, hubo prácticas que no se pudieron poner en práctica porque el tamaño del proyecto y los recursos disponibles no lo permitían.

- Uso de roles. La metodología Scrum tiene 3 roles definidos: Scrum master, Jefe de proyecto y desarrolladores. En el proyecto no se pudieron usar roles porque el alumno se tendría que encargar de ser los 3 roles al mismo tiempo.
- Reuniones diarias. En la metodología Scrum se efectúan reuniones diarias para determinar el plan del día que se va a seguir en el proyecto. Como este proyecto no tenía un tamaño muy grande simplemente se optó por llevar a cabo las reuniones al acabar cada sprint.

3.2 Planificación y seguimiento

3.2.1 Planificación

Primero se describen brevemente las tareas necesarias para realizar el proyecto y de los recursos necesarios para hacerlo (humanos y técnicos). Para llevar a cabo el proyecto se consideraron las siguientes tareas.

- Estudio de la tecnología. Antes de empezar con los requisitos se hace un estudio previo

de las herramientas novedosas que se van a emplear en el desarrollo, ya que así se evita estar gastando el tiempo en leer documentación una vez empezado el desarrollo. Duración estimada: 3 semanas.

- Historias de usuario. Se obtiene una lista con 24 funcionalidades básicas que debe satisfacer la aplicación. Estas historias de usuario se dividen en 5 sprints. Duración estimada: 3 días.
- Diseño de mocks. Una vez se obtiene la lista de funcionalidades se pasa al prototipado de pantallas. Duración estimada: 2 días.
- Diseño del modelo de datos. Terminadas las pantallas se construye el modelo de datos que da soporte a la aplicación. Duración estimada. 1 día.
- Desarrollo de la aplicación. Cuando se tiene el modelo de datos se empiezan a implementar las historias de usuario dividas en los sprints antes mencionados. Duración estimada: 14 semanas.
- Realización de la memoria. Una vez terminado el desarrollo de la aplicación se procede a escribir la memoria del proyecto. Duración estimada: 5 semanas.

En el cronograma de la figura 3.2 se puede ver la distribución de tareas en el tiempo. La fecha inicio del proyecto corresponde con el 6 de abril y la fecha fin del proyecto corresponde con el 6 de septiembre.

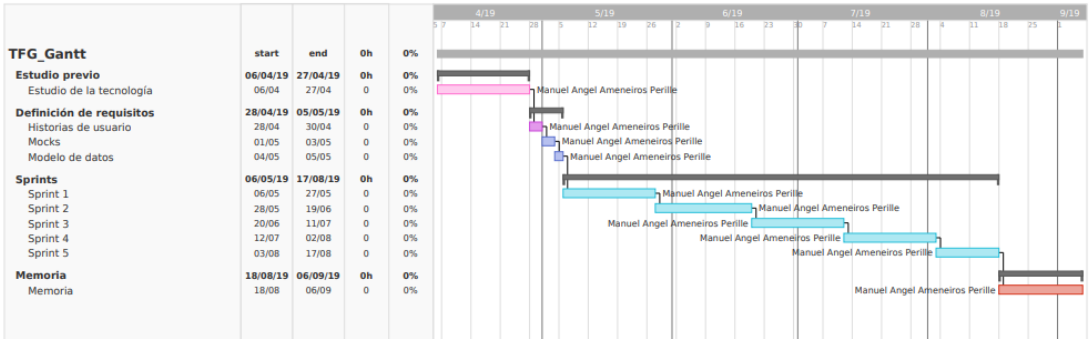


Figura 3.2: Diagrama de Gantt del proyecto.

Los recursos humanos disponibles para este proyecto son de dos tipos diferentes: Director y Programador.

- **Director:** Se encarga de la planificación y supervisión del proyecto, así como de las iteraciones. El seguimiento del proyecto se producía aproximadamente cada 4 semanas pudiendo ser alguna semana más debido a que parte del desarrollo se realizó durante el verano.

- **Programador:** Rol realizado por el autor, se encarga del análisis, diseño, desarrollo y pruebas de cada uno de los sprints indicados.

En cuanto a los recursos técnicos, se disponía de dos ordenadores: uno para realizar el desarrollo y otro para llevar a cabo las pruebas.

3.2.2 Seguimiento

En esta sección se detalla el seguimiento que se llevó durante el desarrollo del proyecto. En concreto, se establece la asignación de historias a sprints, se actualiza el cronograma con detalle y se detallan las horas empleadas y los costes.

La asignación de historias a sprints se produce de la siguiente manera:

- Sprint 1. Primer sprint del desarrollo. Incluye las historias de usuario 19, 22, 13, 1 y 2. Duración real: 28 días.
- Sprint 2. Segundo sprint del desarrollo. Incluye las historias de usuario 23, 16, 14, 7 y 3. Duración real: 10 días.
- Sprint 3. Tercer sprint del desarrollo. Incluye las historias de usuario 20, 18, 17, 5 y 4. Duración real: 16 días.
- Sprint 4. Cuarto sprint del desarrollo. Incluye las historias de usuario 21, 15, 8, 11 y 6. Duración real: 15 días.
- Sprint 5. Quinto sprint del desarrollo. Incluye las historias de usuario 9, 10, 12 y 24. Duración real: 17 días

Una vez asignadas las historias y llevado a cabo el desarrollo, el cronograma queda como se puede ver en la figura 3.3 Comparando este cronograma con el de la figura 3.2 vemos que solo existe desviación en tiempo durante el primer sprint puesto que las historias de este sprint son más complejas de lo que inicialmente se estima, pero esta desviación se compensa durante los otros sprints y se aprovecha el tiempo ganado para empezar a trabajar en la memoria. Debido a problemas de causa mayor se produce un retraso en la finalización de la memoria y la fecha fin real del proyecto es el 27 de septiembre.

Respecto a las horas dedicadas, el autor trabaja 4 horas al día todos los días de la semana, lo que suponen 28 horas semanales aproximadamente. Además, la duración en días del proyecto se establece en 174 días por lo que la duración total del proyecto es de 696 horas. Por otra parte, el director del proyecto dedica unas 25 horas en total entre reuniones y revisión de la memoria. Se puede ver un resumen de las horas dedicas en la tabla 3.1

Por último, el detalle de los costes del proyecto quedaría como se puede ver a continuación:

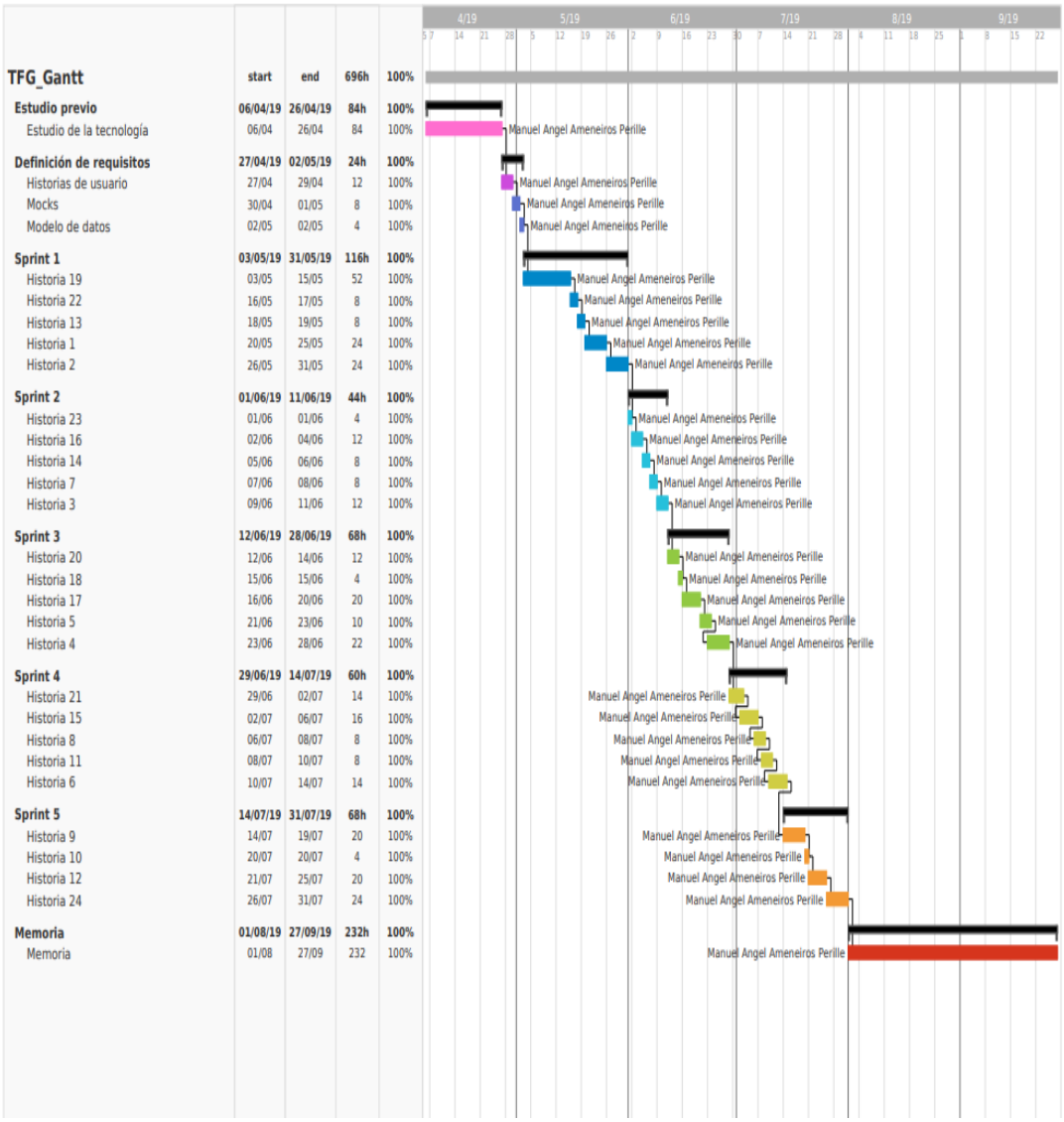


Figura 3.3: Diagrama de Gantt en detalle del proyecto.

Recurso	Horas/persona
Manuel Ángel Ameneiros Perille	696
Miguel Ángel Rodríguez Luaces	25

Cuadro 3.1: Horas dedicadas al proyecto.

- **Director:** Se parte de un salario bruto de 36000€/año. Teniendo en cuenta que un año laboral tiene 1764 horas, nos quedarían 20.54€/hora.
- **Programador:** Se parte de un salario bruto de 15000€/año. Teniendo en cuenta que un año laboral tiene 1764 horas, nos quedarían 8.50€/hora.

3.2.3 Herramientas

Las herramientas empleadas durante el desarrollo han sido las siguientes:

- **Latex** [17]: Es un sistema de composición de textos. Se ha empleado para realizar la memoria.
- **Eclipse** [18]: Entorno de desarrollo integrado (IDE) empleado para codificar la parte Java de la aplicación.
- **Sublime** [19]: Es un editor de código multiplataforma y ligero. Fue empleado para codificar la parte cliente de la aplicación.
- **MagicDraw** [20]: Herramienta CASE compatible con el estándar UML. Se ha empleado para elaborar los diagramas de casos de uso.
- **Dia** [21]: Herramienta de propósito general para la creación de diagramas. Se utilizó para dibujar el modelo de datos y los diagramas de clases.
- **Trello** [22]: Es un software de administración de proyectos con interfaz web. Se utilizó para dibujar el diagrama de Gantt.
- **Google Docs** [23]: Es el gestor de documentos de Google. Se empleó una hoja de cálculo para llevar la cuenta de las horas dedicadas al proyecto.
- **Postman** [24]: Herramienta que permite probar operaciones REST sobre distintas urls.

Además de estas herramientas se hizo uso de Gitlab [25] para llevar a cabo el seguimiento del proyecto. Gitlab nos permite crear milestones para cada sprint y dentro de cada Milestone se crean tantas issues como historias vaya a implementar el sprint. A continuación se establecen las fechas de comienzo y entrega para la Milestone y se van desarrollando las historias individualmente. Para facilitar el desarrollo, Gitlab nos proporciona la opción de crear una rama por cada issue, de esta manera cada desarrollo es independiente y una vez se termina se mergea todo sobre la rama maestra, que será la que se entregue al final de todo. En la Figura 3.4 se puede ver una issue en detalle y en la Figura 3.5 se puede ver una milestone en detalle.

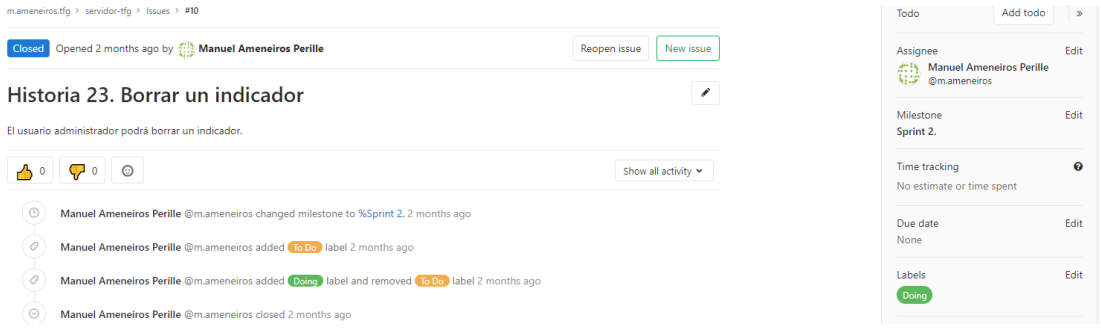


Figura 3.4: Issue en detalle.

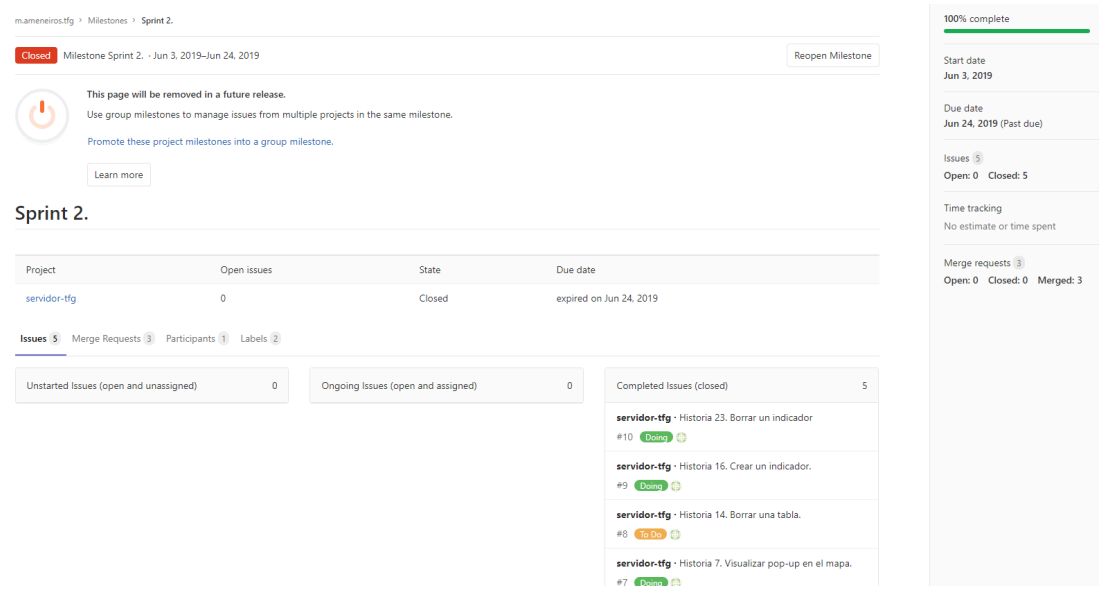


Figura 3.5: Milestone en detalle.

Capítulo 4

Análisis

4.1 Actores

Un actor es una entidad que participa en una historia de usuario, y que puede tener relaciones de herencia con otros actores. En esta sección se mostrarán los dos actores presentes en la aplicación.

- **Usuario anónimo.** Puede logearse en la aplicación como usuario administrador y puede visualizar la información de los indicadores geográficos.
- **Usuario administrador.** Puede crear, modificar y agrupar indicadores. Puede dar de alta o de baja a otros administradores. Y puede publicar o borrar de base de datos la información geográfica.

4.2 Requisitos

4.2.1 Requisitos del cliente

Uno de los pasos que debemos realizar antes del desarrollo para describir el comportamiento del sistema es la Especificación de Requisitos. A continuación se detallan los distintos requisitos del cliente agrupados por tipo de funcionalidad.

Autenticación

- Autenticarse en la aplicación. Los administradores tienen que poder autenticarse en la aplicación.
- Cerrar la sesión. Los administradores tienen que poder cerrar la sesión en la aplicación.

Indicadores

- Consultar los indicadores disponibles. En la pantalla principal de la aplicación los usuarios que no estén logeados podrán ver un listado con los indicadores geográficos.
- Visualizar un indicador como un mapa. Al seleccionar un indicador, el usuario anónimo puede ver la información asociada en un mapa con distintos colores para determinar los rangos del valor del indicador.
- Visualizar un indicador como una tabla. El usuario anónimo puede ver la información del indicador como una tabla compuesta por las columnas 'lugar' y 'valor'.
- Visualizar un indicador como un gráfico de barras. El usuario anónimo puede ver la información del indicador como un gráfico de barras.
- Visualizar un indicador como un gráfico circular.
- Visualizar un indicador como un gráfico de cajas y bigotes.
- Ver un popup al pasar el ratón por encima de un lugar en el mapa. Si el usuario anónimo pasa el ratón por encima de alguno de los lugares del mapa se actualizará un popup que indica el valor del indicador para el lugar indicado.
- Ver un popup al pasar el ratón por encima del gráfico de barras. Si el usuario anónimo pasa el ratón por encima de alguna de las barras del gráfico podrá ver un popup con información del lugar y el valor del indicador.
- Ver un popup al pasar el ratón por encima del gráfico circular. Si el usuario anónimo pasa el ratón por encima de alguno de los sectores del gráfico circular podrá ver un popup con información del lugar y el valor del indicador.
- Comparar dos indicadores. El usuario anónimo puede comparar dos indicadores creados sobre la misma tabla de datos.
- Filtrar indicadores por nombre. En el listado general de indicadores se podrá filtrar por nombre.
- Aplicar filtros sobre el mapa. El usuario anónimo puede filtrar la información sobre el mapa por nombre o por valor, así solo verá dibujados en el mapa aquellos lugares que cumplan los filtros aplicados.
- Crear indicadores. Un administrador puede crear indicadores. A la hora de crear un indicador debe elegir la tabla de la que quiere obtener los datos, la columna de valores que quiere representar con el indicador, así como los distintos tipos de visualización que quiere para el indicador. Se enviará un email al administrador con los datos del indicador.

- Modificar indicadores. Un administrador puede modificar los valores de un indicador. Se enviará un email al administrador con los datos del indicador.
- Eliminar indicadores. Un administrador puede eliminar indicadores.
- Ocultar indicadores. Un administrador puede ocultar aquellos indicadores que no le interesa que vean los usuarios anónimos.
- Agrupar indicadores. Un administrador puede agrupar los indicadores siguiendo algún tipo de criterio.

Tablas

- Publicar una tabla. Un administrador puede publicar archivos shapefile en la base de datos.
- Borrar una tabla. Un administrador puede borrar una tabla de datos de la base de datos.

Usuarios

- Dar de alta administradores. Un administrador puede crear a otros administradores. Se enviará un email al administrador registrado.
- Dar de baja administradores. Un administrador puede dar de baja a otros administradores.
- Actualizar el perfil de usuario. Un administrador puede modificar sus datos personales.

4.2.2 Requisitos del sistema

A partir de los requisitos del cliente, detallaremos en esta sección los requisitos con suficiente detalle, describiendo la secuencia de iteraciones entre un actor y el sistema, como para que se puedan implementar. En nuestro caso disponemos de los actores citados en la Sección 4.1

R1-Usuario anónimo

El usuario que accede a la página web y puede loguearse (Figura 4.1) o consultar información de los indicadores (Figura 4.2).

R1.1-Loguearse

Un usuario anónimo puede entrar en la aplicación como administrador si otro administrador lo ha registrado como tal.

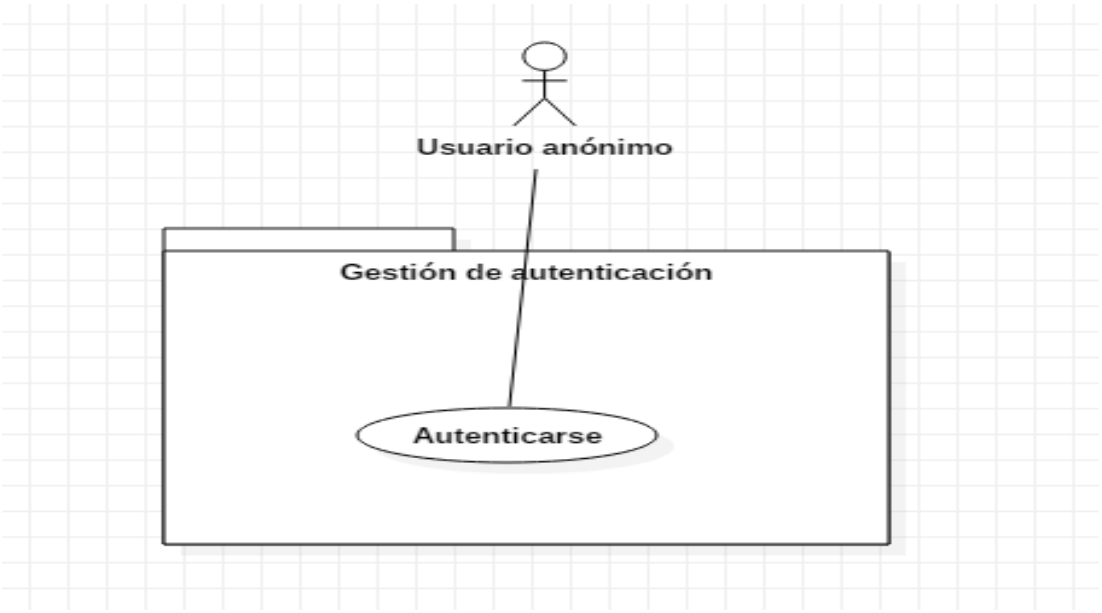


Figura 4.1: Historia de usuario: Autenticación.

R1.2-Consulta de indicadores

Al entrar en la aplicación, el usuario anónimo tiene acceso a un listado formado por los grupos de indicadores que puede visualizar. Si hace click en cualquier indicador podrá acceder a las visualizaciones disponibles para el indicador (R1.2.1, R1.2.2, R1.2.3, R1.2.4, R1.2.5), también puede filtrar los indicadores de los grupos (R1.2.6), aplicar filtros en el mapa de un indicador (R1.2.7, R1.2.8) o comparar 2 indicadores (R1.2.9).

R1.2.1-Visualizar mapa

Al acceder al detalle de un indicador, lo primero que vemos es el mapa que genera ese indicador. El mapa representará los lugares asociados al indicador así como el valor del indicador para cada lugar. En función de los rangos y colores establecidos durante la creación, cada lugar tendrá un color. En el mapa podremos hacer click sobre cada lugar para hacer zoom y al pasar el ratón por encima de un lugar cambiará de valor un pop-up de la esquina superior.

R1.2.2-Visualizar tabla

Se genera una tabla con tantas filas como lugares haya en la tabla asociada al indicador y con una columna para el lugar y otra para el valor. Las columnas de esta tabla se pueden ordenar y por defecto tendrán la ordenación que se haya establecido durante la creación. También se podrán filtrar resultados de la tabla.

R1.2.3-Visualizar gráfico de barras

Se generará un gráfico de barras con tantas barras como lugares, y cada barra indicará el valor del indicador. Si pasamos el ratón por encima de una de las barras obtendremos el valor para el indicador en el lugar correspondiente.

R1.2.4-Visualizar gráfico circular

Se generará un gráfico circular o diagrama de sectores que puede incluir 3 conjuntos de datos diferentes en función de lo que se haya elegido durante la creación. Si el administrador elige ”Los 20 mayores” solo tendremos 20 sectores en el gráfico, si elige ”> 5%” tendremos tantos sectores como lugares que superen el 5% del valor total y si elige todos tendremos tantos sectores como lugares. Al pasar el ratón por encima de un sector obtendremos un pop-up con el valor del indicador en el lugar.

R1.2.5-Visualizar gráfico de cajas y bigotes

Se generará un gráfico de cajas que mostrará el primer cuartil, la mediana, el tercer cuartil, la mediana, la media y los valores externos. Si pasamos el ratón por encima de alguno de los valores externos obtendremos el valor y el nombre de ese punto externo.

R1.2.6-Filtrar indicadores

En el listado de grupos un usuario puede aplicar un filtro por nombre sobre los indicadores. El filtro determinará que nombres de indicador contienen la palabra del filtro y mostrará los resultados adecuados. Estos resultados no se dividen en grupos.

R1.2.7-Filtrar por nombre en el mapa

El usuario puede introducir cualquier palabra en un filtro situado al lado del mapa y la aplicación se encargará de mostrar en el mapa aquellos lugares cuyo nombre incluya la palabra del filtro.

R1.2.8-Filtrar por valor en el mapa

El usuario puede introducir cualquier valor en un filtro situado al lado del mapa y además elegir el signo que quiere, pudiendo ser >, <, >=, <=, y la aplicación se encargará de mostrar en el mapa aquellos lugares cuyo valor cumpla las condiciones del filtro.

R1.2.9-Comparar dos indicadores

Se podrán comparar dos indicadores creados sobre la misma tabla para que incluyan los mismos lugares. La comparación da lugar a una tabla formada por 3 columnas: nombre del lugar, valor del primer indicador y valor del segundo indicador.

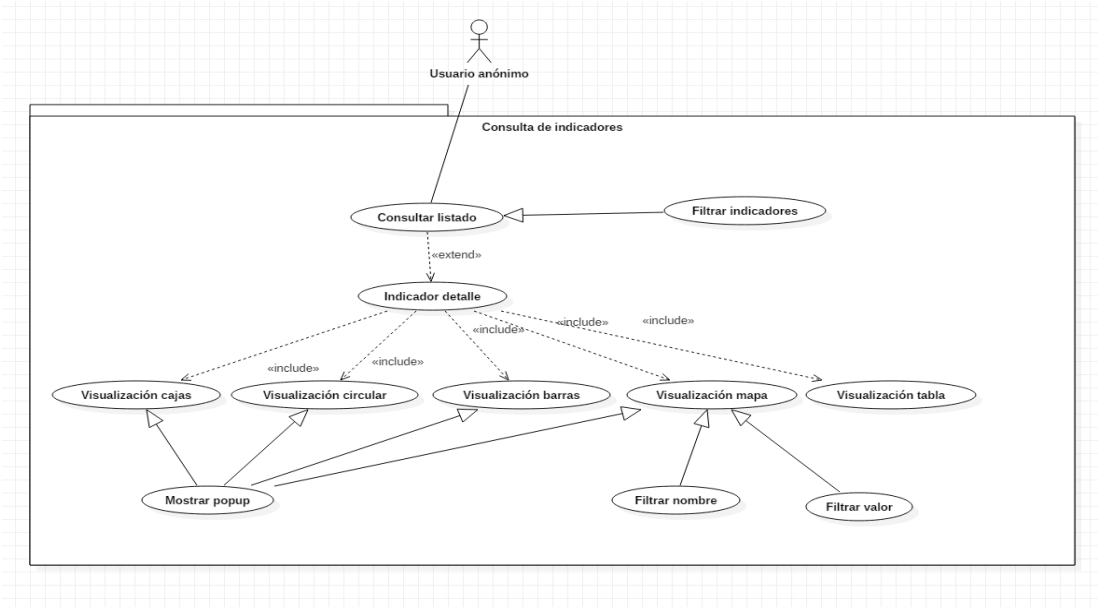


Figura 4.2: Historia de usuario: Consulta de indicadores.

R2-Usuario administrador

Es el actor encargado de gestionar los indicadores, los grupos de indicadores, las tablas de datos y a otros administradores. Las historias de usuario se pueden ver en la Figura 4.3

R2.1-Gestión indicadores

El administrador podrá visualizar el conjunto de indicadores así como crear (R2.1.1), editar (R2.1.2), eliminar (R2.1.3) u ocultar indicadores (R2.1.4).

R2.1.1-Crear indicador

Se usará un formulario para rellenar los campos del indicador formado por el nombre, la tabla asociada, la columna de donde obtendremos los valores del indicador, la columna de donde obtendremos los nombres de los lugares, la unidad de medida, el grupo si fuese necesario y a mayores las visualizaciones que queremos darle a nuestro indicador. Cada visualización

tendrá un checkbox al lado para elegirla, siendo la de mapa obligatoria, además las visualizaciones de mapa, tabla y sectores incluyen datos adicionales. Al elegir la de mapa tendremos que introducir los rangos y colores que queremos, al elegir la de tabla tendremos que elegir la columna de ordenación y al elegir la de sectores tendremos que elegir el conjunto de datos que queremos. Al crear un indicador se envía un email al administrador que lo ha creado con un resumen de los campos que tiene el indicador.

R2.1.2-Editar indicador

Se usará un formulario con los mismos campos que en la creación pero cubiertos con los datos del indicador. Además no se podrá desmarcar la casilla de mapa. Al modificar un indicador se envía un email al administrador que lo ha modificado con un resumen de los campos que tiene el indicador.

R2.1.3-Eliminar indicador

Desde la tabla de indicadores podremos eliminar un indicador clicando en el botón eliminar de cualquiera de las filas. Si aceptamos se eliminará el indicador así como las visualizaciones que se hayan creado para ese indicador para evitar saturar la base de datos de visualizaciones sin uso.

R2.1.4-Ocultar indicador

Desde la tabla de indicadores podremos ocultar o mostrar un indicador si pulsamos en el radio button que se encuentra en cada fila. Al ocultar un indicador impedimos que los usuarios anónimos lo puedan ver.

R2.2-Gestión grupos

El administrador podrá visualizar el conjunto de grupos de la aplicación así como crear (R2.2.1), editar (R2.2.2) o eliminar cualquier grupo de indicadores (R2.2.3).

R2.2.1-Crear grupo

Se usará un formulario compuesto por los campos nombre del grupo, indicadores sin grupo que podemos añadir al grupo y subgrupos que queremos añadir al grupo. Por temas de rendimiento solo se permite que un grupo tenga un grupo padre, y solo se permite que un indicador pertenezca a un grupo.

R2.2.2-Editar grupo

Se usará el mismo formulario de la creación pero con los datos del grupo rellenados.

R2.2.3-Eliminar grupo

Se podrá eliminar un grupo desde la tabla con todos los grupos haciendo click en el botón eliminar de cada fila. Al eliminar un grupo con indicadores tendríamos que darle un grupo a esos indicadores para que se pudiesen ver en el menú principal.

R2.3-Gestión tablas

El administrador podrá visualizar el conjunto de tablas de datos así como crear (R2.3.1) o eliminar tablas (R2.3.2) de base de datos.

R2.3.1-Crear tabla

Se usará un formulario compuesto por el nombre de la tabla, un selector de archivos y una barra de progreso. El usuario deberá seleccionar un archivo .zip con todos los ficheros shapefile necesarios para crear la tabla en base de datos.

R2.3.2-Eliminar tabla

Se podrá eliminar una tabla desde la tabla con todos los nombres de las tablas haciendo click en el botón eliminar de cada fila. Al darle se mostrará una ventana modal que nos informará de que se borrarán todos los indicadores que se hayan creado sobre la tabla, siendo imposible su recuperación.

R2.4-Gestión administradores

El administrador podrá visualizar el conjunto de administradores que hay en la aplicación así como crear (R2.4.1) o eliminar (R2.4.2) administradores. Cada administrador podrá actualizar sus datos personales (R2.4.3).

R2.4.1-Crear administrador

Se usará un formulario compuesto por los campos nombre, lugar de nacimiento, fecha de nacimiento, login, email, contraseña y repetir contraseña. Al crear un administrador se enviará un correo electrónico a la dirección especificada durante la creación. A mayores, si se intenta usar un login que ya existe se avisará mendiante una alerta de que no se ha podido crear el usuario.

R2.4.2-Eliminar administrador

Se podrán eliminar los administradores que no seamos nosotros desde la tabla con todos los administradores haciendo click sobre el botón de eliminar de cada fila.

R2.4.3-Actualizar datos personales

Se usará el mismo formulario empleado durante la creación pero con los campos del administrador rellenados.

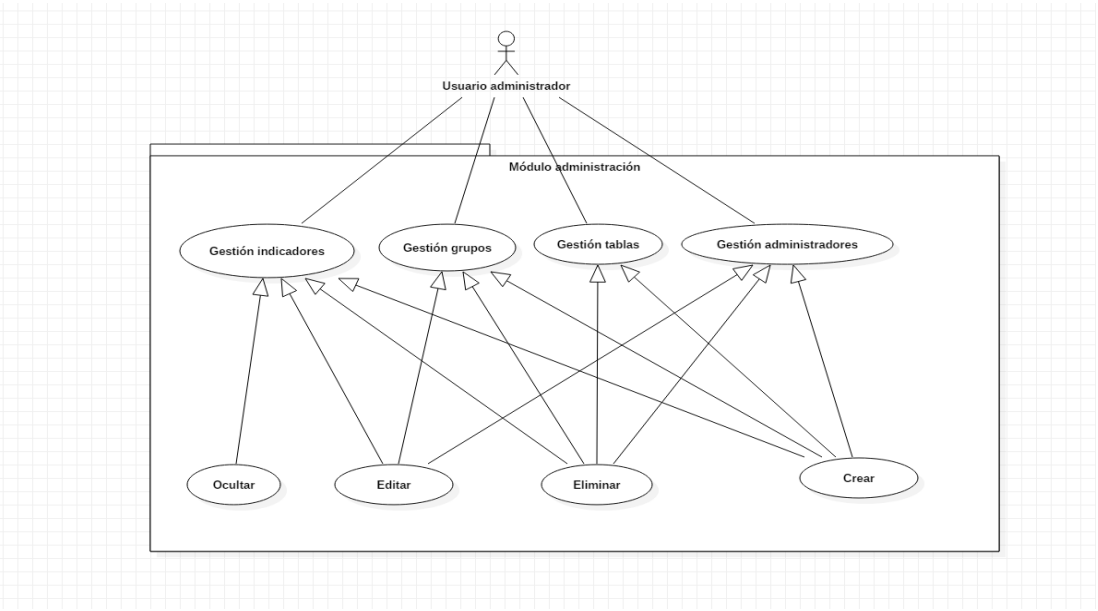


Figura 4.3: Historias de usuario administrador.

4.3 Arquitectura del sistema

La arquitectura empleada en el sistema es una arquitectura en 3 capas. Figura 4.4

- Capa de acceso a datos (Modelo).
- Capa de lógica de negocio (Controlador).
- Capa de presentación al usuario (Vista).

Solo se conoce la capa inmediatamente inferior consiguiendo así abstracción, aislamiento de cambios, incremento de escalabilidad, tolerancia a fallos y rendimiento.

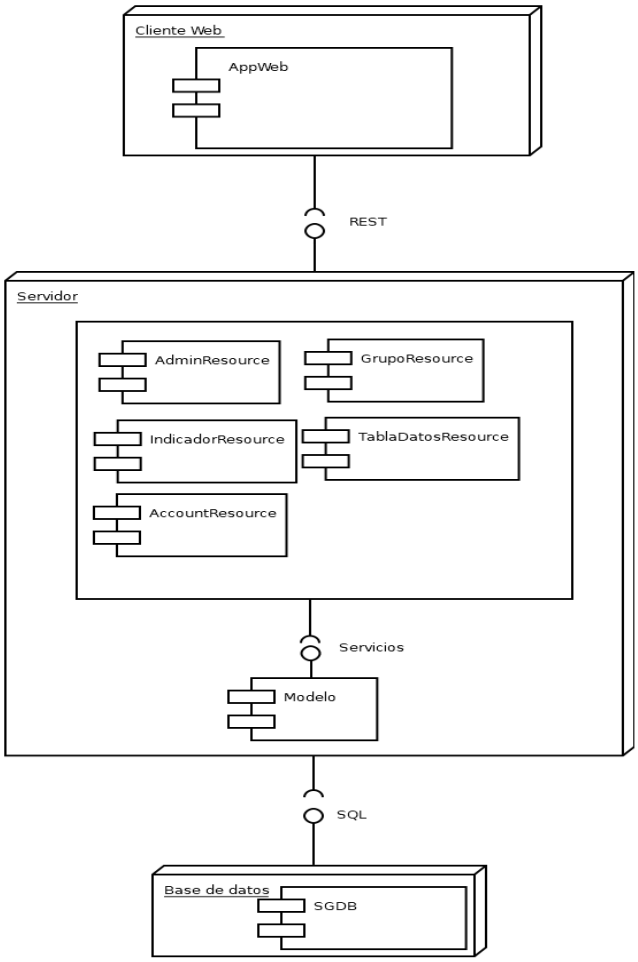


Figura 4.4: Arquitectura del proyecto.

4.4 Interfaz de usuario

4.4.1 Componentes del sistema

En este apartado se muestran todos los componentes que forman parte del cliente y la navegación entre ellos. Se omiten LoadingPage, NotFound y AppVue porque no tienen un comportamiento relacionado con la funcionalidad de la aplicación. Los componentes de la aplicación se pueden ver en los cuadros 4.1, 4.2, 4.3, 4.4, 4.5 y 4.6

Componentes comunes	Descripción
Home	Es la pantalla de inicio que solo sale para usuarios anónimos. Aquí el usuario verá un menú lateral con los grupos de indicadores que puede visualizar, al hacer click en cualquier grupo se despliega y podrá ver los indicadores que forman parte de él. Al hacer click en el indicador irá a la visualización en mapa del indicador.
Login	La pantalla en la que los usuarios administradores pueden logearse en la aplicación. Es un formulario compuesto por un campo nombre y un campo contraseña. Al introducir correctamente sus credenciales el administrador irá a la pantalla de gestión de indicadores.
MenuBar	Carga una barra superior en todas las pantallas. Esta barra dispone del botón de login si no estamos logeados o un botón de salir de la aplicación si estamos como administradores.

Cuadro 4.1: Componentes comunes.

4.4.2 Mockups de componentes

La aplicación está compuesta por 3 pantallas principales que varían su contenido en función de las acciones realizadas. La primera pantalla sería el menú principal para usuarios anónimos. En esta pantalla tendremos un menú lateral en el que se muestran los grupos de indicadores y podremos seleccionar cualquier indicador para ir a la pantalla de visualización de indicador. Figura 4.5 La segunda pantalla principal sería la de visualización de indicador. Esta pantalla está compuesta por un menú lateral en el que podemos seleccionar la visualización que queremos y la parte restante de la pantalla en la que se muestra la visualización que hayamos seleccionado. En el prototipo no sale este menú lateral sino unos botones para cambiar la visualización, se realizó el cambio para aprovechar mejor el espacio de la pantalla.

Componentes administrador	Descripción
AdminForm	Formulario para dar de alta a los administradores o modificar los datos personales. Tiene los campos nombre, login, fecha de nacimiento, lugar de nacimiento, email y contraseña. Si pulsamos el botón aceptar iremos a la gestión de administradores.
AdminList	Pantalla en la que disponemos de una tabla con los nombres de los administradores y un botón para eliminarlos. Si pulsamos el botón de eliminar nos saldrá una ventana modal que nos pedirá confirmación para eliminar al administrador.

Cuadro 4.2: Componentes de las pantallas para gestionar administradores.

Componentes grupos	Descripción
GrupoForm	Formulario para agrupar indicadores. Tiene un campo nombre, un selector multivaluado para seleccionar los indicadores que queremos en el grupo y un selector multivaluado para seleccionar subgrupos si queremos que sea un grupo padre. Al darle al botón aceptar iremos a la gestión de grupos.
GrupoList	Pantalla en la que disponemos de una tabla con los nombres de los grupos, un botón para eliminar el grupo y una celda con la lista de indicadores que tiene el grupo. Si le damos al botón de borrar grupo nos saldrá una ventana modal que nos pedirá confirmación para borrar el grupo.

Cuadro 4.3: Componentes de las pantallas para gestionar los grupos.

Componentes tablas	Descripción
TablaForm	Formulario para subir archivos shapefile. Tiene el campo nombre y un selector de archivos. Si el archivo se sube correctamente nos enviará directamente a la pantalla de gestión de tablas, si hay algún problema sale una alerta de bootstrap indicando que no se pudo almacenar el fichero en base de datos.
TablaGestion	Pantalla en la que disponemos de una tabla con los nombres de las tablas de datos geográficos y el botón para eliminarlas de base de datos. Si pulsamos el botón de eliminar saldrá una ventana modal para pedirnos confirmación de la eliminación, avisando también de que se eliminarán todos los indicadores geográficos que estén asociados con la susodicha tabla.

Cuadro 4.4: Componentes de las pantallas para gestionar las tablas.

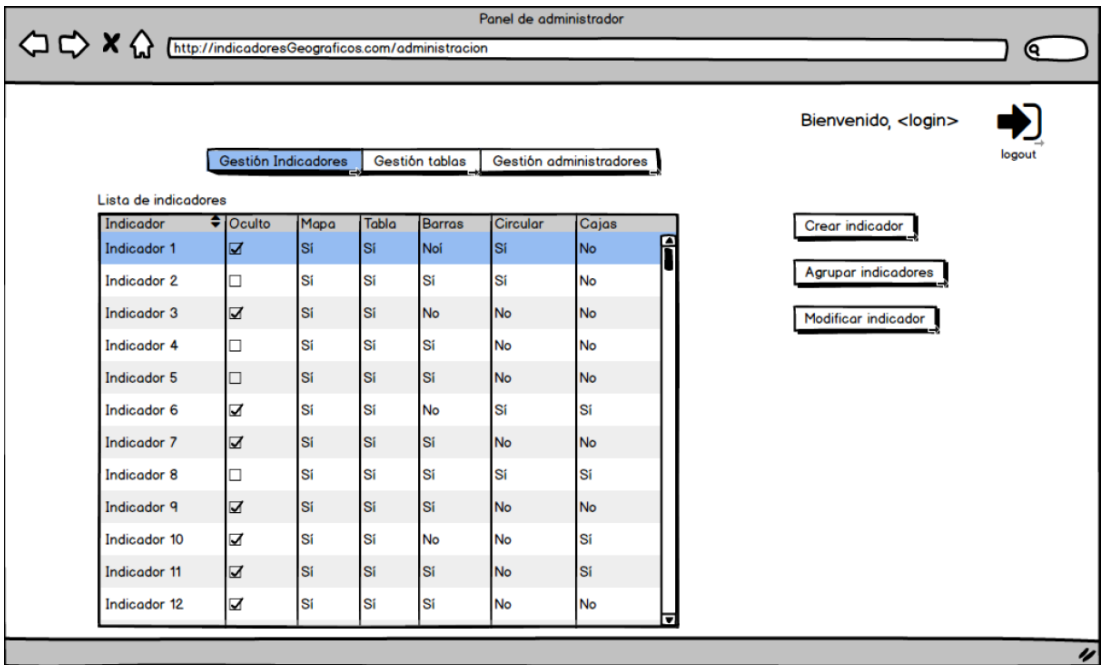


Figura 4.5: Pantalla principal

Componentes de indicadores	Descripción
IndicadorGestión	Pantalla en la que disponemos de una tabla con los nombres de los indicadores, el botón para eliminarlos y una celda para ocultar o mostrar los indicadores. Si le damos al botón de eliminar nos saldrá una ventana modal que nos pedirá confirmación para borrar el indicador.
IndicadorForm	Formulario para la creación de indicadores. Está compuesto por los campos nombre, tabla, atributo, lugar, unidad y la sección tipos de visualización donde podremos configurar los tipos de visualización que queremos para nuestro indicador así como personalizarlos. En concreto: podemos elegir mapa y establecer los rangos y colores que queremos para nuestro mapa, podemos elegir tabla y establecer la columna que queremos que determine el orden, podemos elegir diagrama de barras, podemos elegir diagrama de cajas y bigotes y podemos elegir diagrama de sectores y elegir si queremos que salgan todos los valores, solo los 20 mayores o aquellos cuyo valor sea superior al 5% del valor total. Al pulsar el botón aceptar iremos a la pantalla de gestión de indicadores.
IndicadorUpdate	Formulario para actualizar los indicadores. Su contenido es idéntico al de creación pero se ha separado en otro componente porque el comportamiento difería bastante de la creación.

Cuadro 4.5: Componentes de las pantallas para gestionar los indicadores.

Componentes de visualización	Descripción
IndicadorMapa	Pantalla en la podemos visualizar la información del indicador como un mapa. En la zona central sale el mapa que incluye una leyenda en la parte inferior derecha y un popup informativo en la parte superior derecha. A la derecha de este mapa tendremos un par de inputs para poder filtrar la información del mapa por valor o por nombre.
IndicadorTabla	Pantalla en la que podemos visualizar la información del indicador como una tabla con dos columnas: Lugar y valor. Esta tabla incluye un filtro, un paginador y la opción de ordenador los valores al hacer click en las cabeceras de la tabla.
IndicadorBarras	Pantalla en la que podemos visualizar la información del indicador como un gráfico de barras.
IndicadorSectores	Pantalla en la que podemos visualizar la información del indicador como un gráfico de sectores. Podemos ocultar valores del gráfico si hacemos click en alguno de los nombres que salen encima del propio gráfico. Si elegimos las opciones ”Solo 20 mayores” o ”Solo mayores 5 %” tendremos un sector Otros para representar los valores que no cumplen los requisitos.
IndicadorCajas	Pantalla en la que podemos visualizar la información como un gráfico de cajas y bigotes. Este gráfico se puede extender y descargar como una imagen si hacemos click en la esquina superior derecha.
IndicadorCompare	Pantalla en la que podemos comparar dos indicadores creados sobre la misma tabla. Al principio disponemos de un formulario con dos campos para seleccionar los dos indicadores a comparar. Una vez elegidos los dos nos saldrá una tabla con el nombre del lugar, el valor para el primer indicador y el valor para el segundo indicador.

Cuadro 4.6: Componentes de las pantallas para visualizar los indicadores.

Figura 4.6 La tercera y última pantalla sería el panel del administrador. Este panel está com-

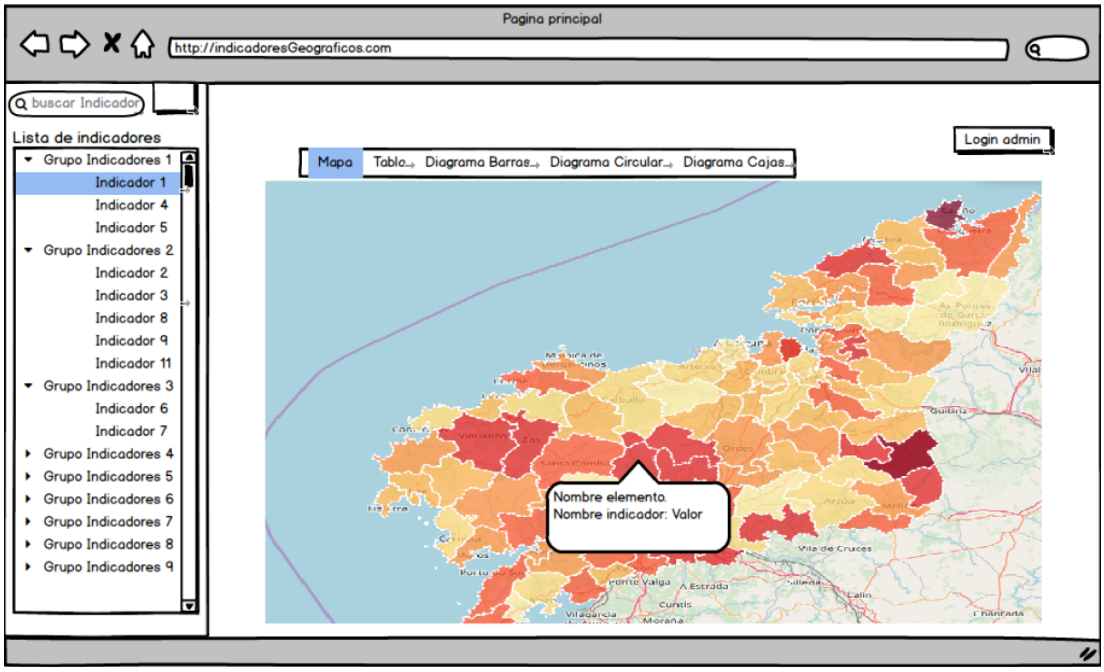


Figura 4.6: Pantalla de visualización de indicador

puesto por un menú lateral que permite cambiar de gestión, pudiendo así pasar de gestión de tablas de datos a gestión de indicadores o a gestión de administradores. En esta pantalla ocurre lo mismo que en la pantalla dos y salen los botones en lugar del menú lateral. Figura 4.7 Todas las pantallas comparten una barra superior que incluye el botón para autenticarse si se está como usuario anónimo o un botón para salir de la aplicación si se está como administrador.

4.5 Modelo conceptual de datos

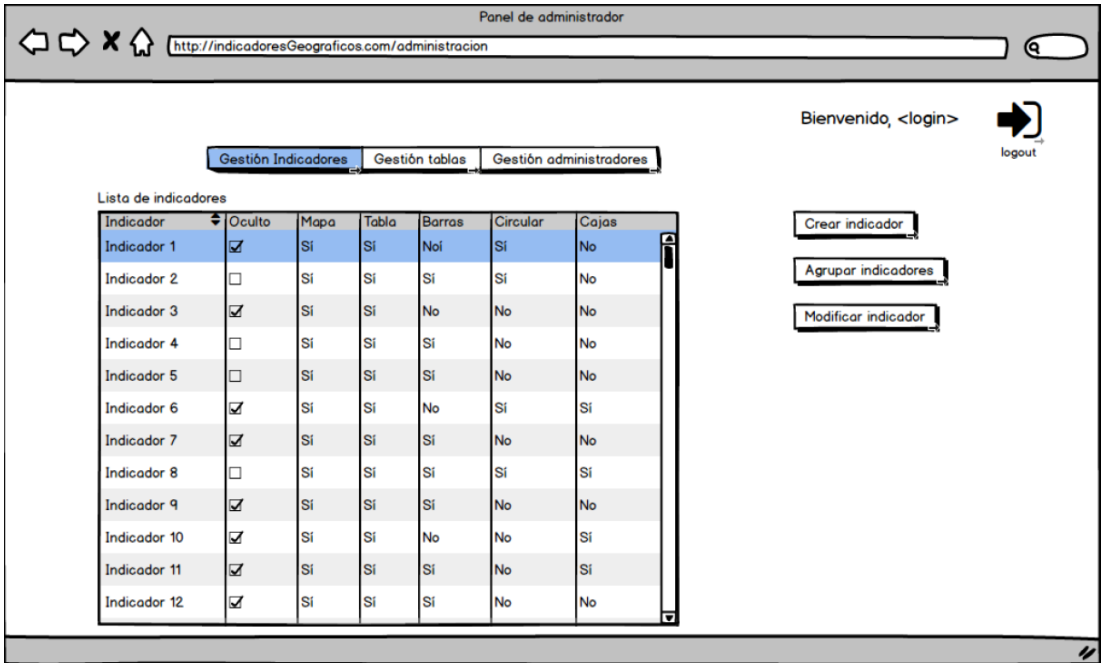
En esta sección se va a hablar de la capa de modelo, encargada de implementar las historias de usuario y la lógica de negocio. Para ello se mostrará el diseño de la Base de Datos, con sus entidades y atributos.

IndicadorComponente: Representa los atributos comunes de los grupos y los indicadores.

- *id:* Identificador del indicador componente.
- *nombre:* Nombre del indicador componente.

Indicador: Representa los atributos propios de los indicadores.

- *oculto:* Booleano que indica si el indicador es visible o no.



Indicador	Oculto	Mapa	Tabla	Barras	Circular	Cajas
Indicador 1	☑	Sí	Sí	Noí	Sí	No
Indicador 2	☐	Sí	Sí	Sí	Sí	No
Indicador 3	☑	Sí	Sí	No	No	No
Indicador 4	☐	Sí	Sí	Sí	No	No
Indicador 5	☐	Sí	Sí	Sí	No	No
Indicador 6	☑	Sí	Sí	No	Sí	Sí
Indicador 7	☑	Sí	Sí	Sí	No	No
Indicador 8	☐	Sí	Sí	Sí	Sí	Sí
Indicador 9	☑	Sí	Sí	Sí	No	No
Indicador 10	☑	Sí	Sí	No	No	Sí
Indicador 11	☑	Sí	Sí	Sí	No	Sí
Indicador 12	☑	Sí	Sí	Sí	No	No

Figura 4.7: Panel de administrador

- *unidad:* Unidad de medida del indicador.

GrupoIndicador: Representa los atributos propios de los grupos de indicadores.

- *idHijos:* Identificadores de los grupos hijos.
- *idPadre:* Identificador del grupo padre.

Administrador: Representa los atributos de los adminsitradores.

- *id:* Identificador del administrador.
- *name:* Nombre completo del administrador.
- *fechaNacimiento:* Fecha de nacimiento del administrador.
- *lugarNacimiento:* Lugar de nacimiento del administrador.
- *login:* Login del administrador.
- *mail:* Mail del administrador.
- *fechaCreacion:* Fecha de creación del administrador.

Visualización: Representa los atributos de las visualizaciones de los indicadores.

- *id:* Identificador de la visualización.
- *configuración:* JSON con los parámetros de la visualización.

TipoVisualización: Enumerado que representa los tipos de visualización que tiene la aplicación

- *MAPA:* Visualización en mapa.
- *TABLA:* Visualización en tabla.
- *BARRAS:* Visualización en diagrama de barras.
- *CIRCULAR:* Visualización en diagrama de sectores.
- *CAJAS* Visualización en cajas.

Configuración: Representa los parámetros de las visualizaciones. No se almacenaría en base de datos como entidad, sino que permite construir un JSON con los atributos del objeto.

- *id:* Identificador de la configuración.
- *atributo:* Atributo de la tabla de datos que va a representar el indicador geográfico.
- *lugar:* Atributo de la tabla de datos que contiene los nombres de los lugares que empleará el indicador.
- *columnaOrden:* Columna que determina la ordenación en la visualización en tabla.
- *visualizacionSectores:* Indica que rango de datos se empleará en la visualización en sectores.
- *rangos:* Lista de rangos para la visualización en mapa.
- *colores:* Lista de colores para la visualización en mapa.

TablaDatos: Representa la configuración básica de las tablas de datos geográficos.

- *id:* Identificador de la tabla de datos.
- *nombre:* Nombre de la tabla de datos.

Atributos: Representa los valores de cada columna de las tablas de datos geográficos.

- *id:* Identificador del atributo.
- *nombre:* Nombre de la columna.
- *tipo:* Tipo de la columna.

Diagrama de clases de los datos persistentes. Figura 4.8

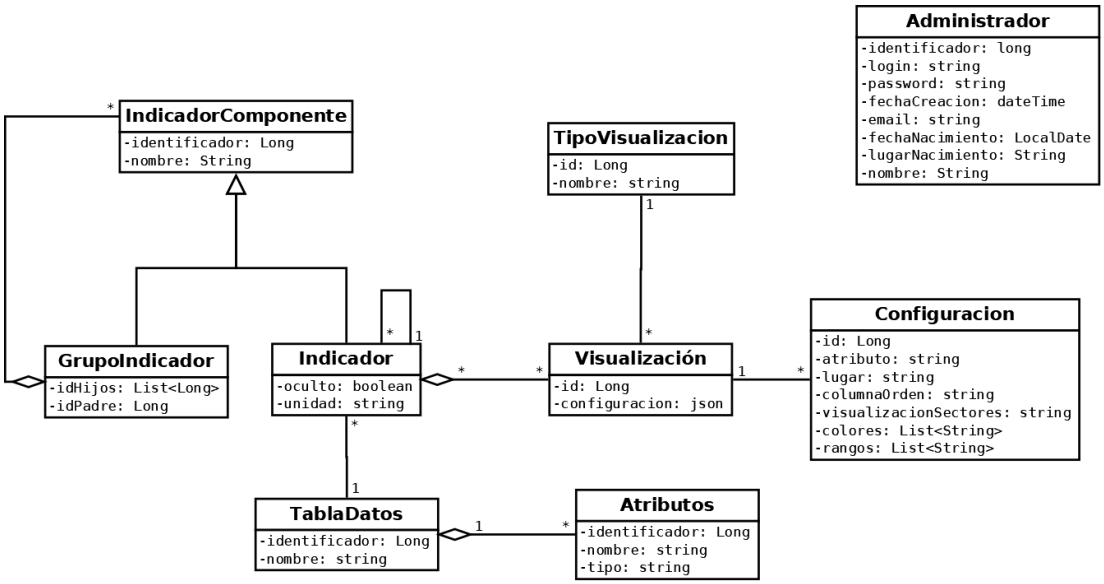


Figura 4.8: Diagrama del modelo de datos del proyecto.

Capítulo 5

Diseño

5.1 Arquitectura tecnológica del sistema

Para implementar los componentes del sistema se empleó la arquitectura mencionada en 4 complementada con las tecnologías indicadas en la Figura 5.1.

- Vista implementada con Vue y Leaflet. Se empleó Vue para desarrollar la parte principal de la vista porque es un framework bastante sencillo de utilizar y con mucha potencia, permitiendo escribir en un solo archivo todo el código html, javascript y css. Aunque el punto fuerte es que se puede personalizar al máximo mediante plugins que simplifican o mejoran el funcionamiento del código. Para dibujar los mapas en la aplicación se hizo uso de Leaflet ya que se trata de una librería gratuita y con facilidad de uso que proporcionaba todo lo necesario para crear mapas y personalizarlos.

- Controlador implementado con Java y Geotools. Se usó Java porque es el lenguaje de programación que más se ha trabajado durante la carrera y porque a día de hoy sigue siendo el lenguaje número uno para programar backend. A mayores se utilizó geotools para gestionar los ficheros shapefile que se suben desde la aplicación. Con geotools podemos convertir los archivos directamente en tablas de la base de datos y al estar completamente integrado con java resultó muy sencillo de usar.

- Modelo de datos implementado con hibernate. Se empleó Hibernate principalmente por ser el único ORM visto durante la carrera y además porque simplifica mucho el paso de objetos Java a entidades persistentes ya que con solo un par de anotaciones podemos convertir cualquier objeto de nuestro modelo en una entidad de base de datos.

- Base de datos creada sobre PostgreSQL con la extensión postgis. Aunque se podría haber trabajado sobre cualquier base de datos para almacenar los datos normales de la aplicación, se utilizó PostgreSQL porque dispone de la extensión postgis que nos permite

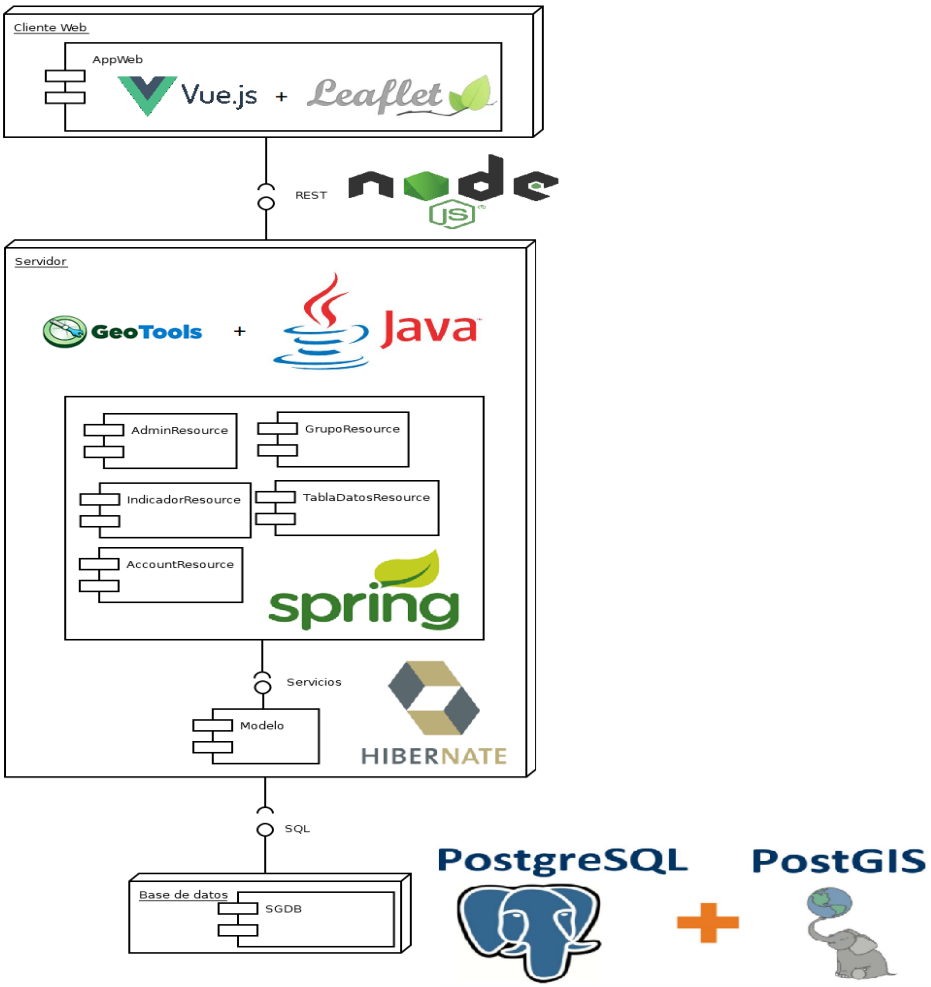


Figura 5.1: Arquitectura tecnológica del proyecto.

almacenar los datos geográficos con más facilidad que otras extensiones para las otras bases de datos. Por su parte postgis nos permite trabajar con las coordenadas de muchas maneras, ya sea obteniendo el área, cambiando el sistema de coordenadas, obteniendo un geojson con la información geográfica, etc.

La aplicación se divide en dos partes. La parte front-end, o lado cliente, es la parte con la que pueden interactuar los usuarios, y la parte back-end, es la parte oculta de la aplicación que se encarga de gestionar los datos así como las peticiones del cliente.

5.2 Diseño de la aplicación

En esta sección se explicará la estructura de las dos partes principales de la aplicación.

5.2.1 Back-end

DAOs

Los Data Access Object (DAO), permiten separar la lógica de acceso a datos de los Objetos de negocio, de forma que el DAO encapsula toda la lógica de negocio al resto de la aplicación. Así el DAO proporciona los métodos necesarios para insertar, actualizar, eliminar o buscar la información (CRUD), mientras que por su parte la capa de negocio se preocupa por la lógica de negocio y hace uso de los DAOs para acceder a la capa de datos.

En el proyecto si accedemos al paquete es.udc.lbd.tfg.model.repository podremos acceder a los 3 DAOs de la aplicación: es.udc.lbd.tfg.model.repository.AdminDAO, es.udc.lbd.tfg.model.repository.IndicadorComponenteDAO y es.udc.lbd.tfg.model.repository.VisualizacionDAO. A mayores de los métodos CRUD el DAO de IndicadorComponente incluye un método findAllNombresGrupos que se emplea para recuperar solo los nombres de los grupos y no enviar toda la información al cliente.

Servicios

La capa de servicios implementa la lógica que realiza las funciones principales de la aplicación. Estos servicios hacen uso de los DAOs para comunicarse con los datos y hacen uso de estos datos para proporcionar funcionalidades al cliente a través de los distintos controladores de la aplicación. En el proyecto tenemos el paquete es.udc.lbd.tfg.model.service que incluye los servicios es.udc.lbd.tfg.model.service.AdminService, es.udc.lbd.tfg.model.service.MailService, es.udc.lbd.tfg.model.service.TablaDatosService y es.udc.lbd.tfg.model.service.IndicadorComponenteService. Los servicios disponibles en el proyecto se muestran en los siguientes diagramas: Figura 5.3, Figura 5.4, Figura 5.6 y Figura 5.5

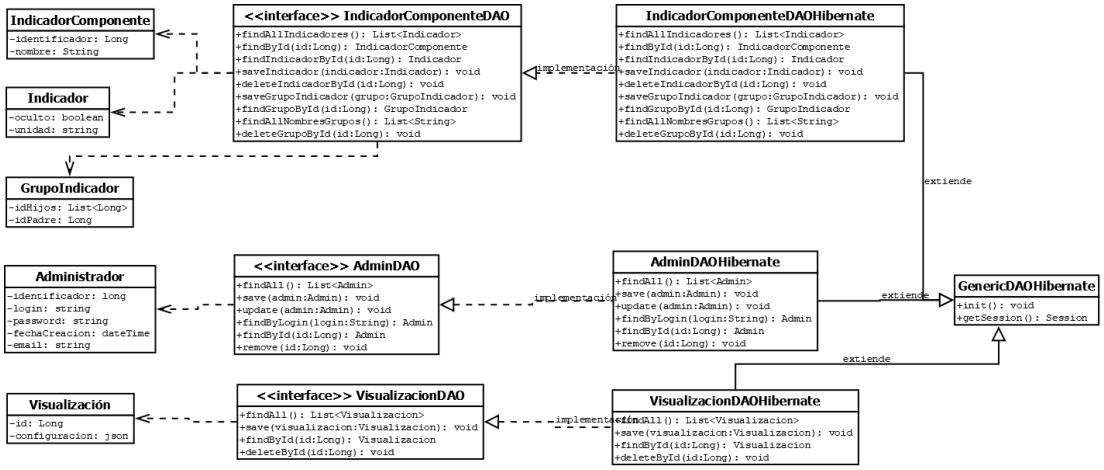


Figura 5.2: DAOs del proyecto.

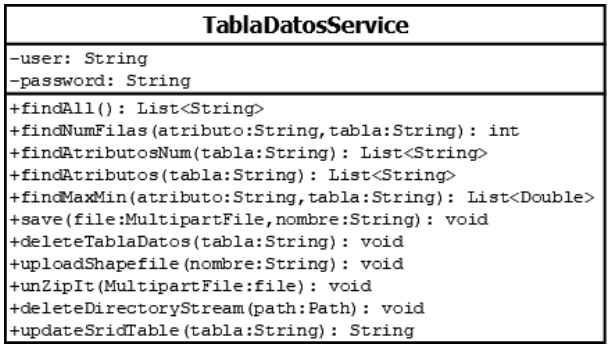


Figura 5.3: Servicio de tablas.

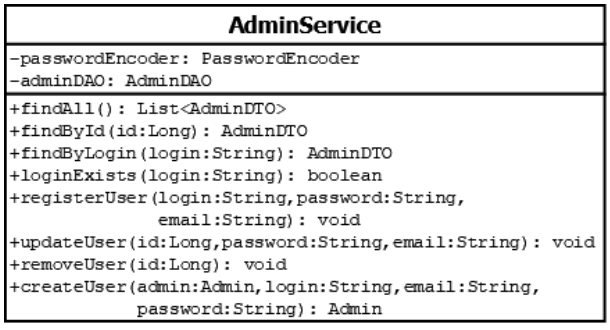


Figura 5.4: Servicio de administrador.

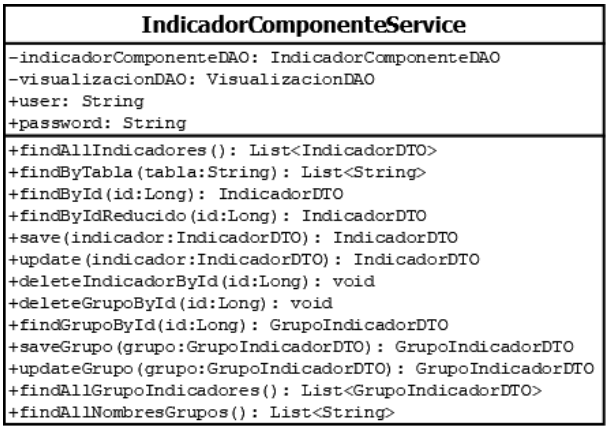


Figura 5.5: Servicio de indicadorComponente.

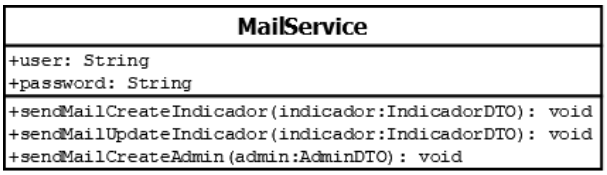


Figura 5.6: Servicio de mailService.

Controladores

Se encuentran en el paquete es.udc.lbd.tfg.web y se encargan de procesar las peticiones REST que llegan desde el cliente. En esta sección se mostrarán las rutas, operaciones REST y parámetros necesarios en cada operación de los recursos. (Cuadro 5.1, Cuadro 5.2, Cuadro 5.3, Cuadro 5.4, Cuadro 5.5)

URL	Operación	Método
/api/tablas	GET	findAll
/api/tablas/findAtributosNum&tabla=tabla	GET	findAtributosNum
/api/tablas/findAtributos&tabla=tabla	GET	findAtributos
/api/tablas/findNumFilas&tabla=tabla&atributo=atributo	GET	findNumFilas
/api/tablas/findByAtributo&tabla=tabla&atributo=atributo	GET	findMaxMin
/api/tablas/upload	POST	save
/api/tablas/tabla	DELETE	deleteTablaDatos

Cuadro 5.1: TablaDatosResource.

URL	Operación	Método
/api/indicadores	GET	findAll
/api/indicadores/id	GET	findById
/api/indicadores/findCompatibles&id=id	GET	findCompatibles
/api/indicadores/findByTabla&tabla=tabla	GET	findByTabla
/api/indicadores	POST	save
/api/indicadores/id	PUT	update
/api/indicadores/id	DELETE	deleteIndicadorById

Cuadro 5.2: IndicadorResource.

URL	Operación	Método
/api/grupos	GET	findAll
/api/grupos/id	GET	findGrupoById
/api/grupos/findAllNombresGrupos	GET	findAllNombresGrupos
/api/grupos	POST	save
/api/grupos/id	PUT	update
/api/grupos/id	DELETE	deleteIndicadorById

Cuadro 5.3: GrupoResource.

URL	Operación	Método
/api/authenticate	POST	authenticate
/api/account/	GET	getCurrentUserWithAuthority
/api/register/	POST	registerUser

Cuadro 5.4: AccountResource.

URL	Operación	Método
/api/admins	GET	findAll
/api/admins/	POST	create
/api/admins/id	GET	findById
/api/admins/findOneAdmin&login=login	GET	findByLogin
/api/admins/loginExists&login=login	GET	loginExists
/api/admins/id	PUT	updateUser
/api/admins/id	DELETE	removeUser

Cuadro 5.5: AdminResource.

5.2.2 Front-end

Para la parte Front-end se ha decidido utilizar Vue.js. Vue se basa en una implementación muy ligera del patrón VMMV lo que nos permite relacionar la capa cliente con la capa servidor de forma simple y eficaz. Se trata de un framework fácilmente aplicable a cualquier proyecto existente y sin la necesidad de emplear librerías obligatorias; a diferencia de frameworks como Backbone que están obligados a emplear jQuery y Underscore para funcionar.

Componentes Vue

La vista de la aplicación está formada por distintos componentes de Vue que se encargan de hacer las peticiones al servidor, manejar los datos obtenidos y renderizar en la vista esta información. Una de las ventajas de Vue es que los componentes pueden incluir el código HTML, javascript y CSS en un solo fichero, por lo tanto los componentes de nuestra aplicación quedarían como se puede ver en la Figura 5.7:

- **Template:** Se trata de la parte para incluir el código HTML. Puede acceder a los datos definidos en la sección de scripts y renderizar dinámicamente si estos datos varían.
- **Script:** Es la sección donde se incluye el código javascript. Se definen métodos que se pueden emplear desde la template, así como invocar al servicio para recuperar los datos necesarios.
- **Style:** Es la parte en la que se aplica el código CSS sobre la vista.

Los componentes de la aplicación se van a dividir entre los accesibles para los administradores y aquellos que solo pueden acceder los usuarios anónimos. También se incluyen los componentes comunes a todos los usuarios: Home.vue, MenuBar.vue y Login.vue. La distribución de componentes se puede apreciar en la Figura 5.8

Rutas

Para poder movernos entre distintos componentes tenemos que disponer de un fichero con los enrutamientos. En el caso de esta aplicación las rutas se encuentran en AppRouter.js. Al acceder a una ruta determinada App.vue hará uso de AppRouter.js para saber que componente tiene que utilizar en la vista. Las rutas de las que dispone la aplicación se pueden ver en los cuadros 5.6 y 5.7

Comunicación con el servidor

Para poder obtener los datos necesarios para pintar las vistas los componentes de Vue realizan peticiones REST al servidor emplean las peticiones HTTP de Axios. Con Axios sim-

```vue
<template>
  <div id="app">
    <div id="menu">
      <MenuBar/>
    </div>
    <router-view class="content"/>
  </div>
</template>

<script>
import auth from './common/auth'
import MenuBar from './components/MenuBar'

export default {
  name: 'App',
  components: { MenuBar },
  data() {
    return {
      // enlazamos el objeto donde vamos a guardar los datos de autenticación
      // de esta forma se activa el data-binding y los podemos usar como
      // propiedades públicas en los componentes
      storeAuth: auth
    }
  },
  created() {
    // nada más ejecutar la aplicación comprobamos si estamos autenticados
    if (auth.getToken()) {
      auth.authenticate().catch(() => auth.logout())
    }
  }
}
</script>

<style scoped lang="scss">
  #app {
    color: white;
  }

  #menu {
    background: #10171E;
  }

  .error {
    background: red;
  }
</style>
```

Figura 5.7: Componente Vue.

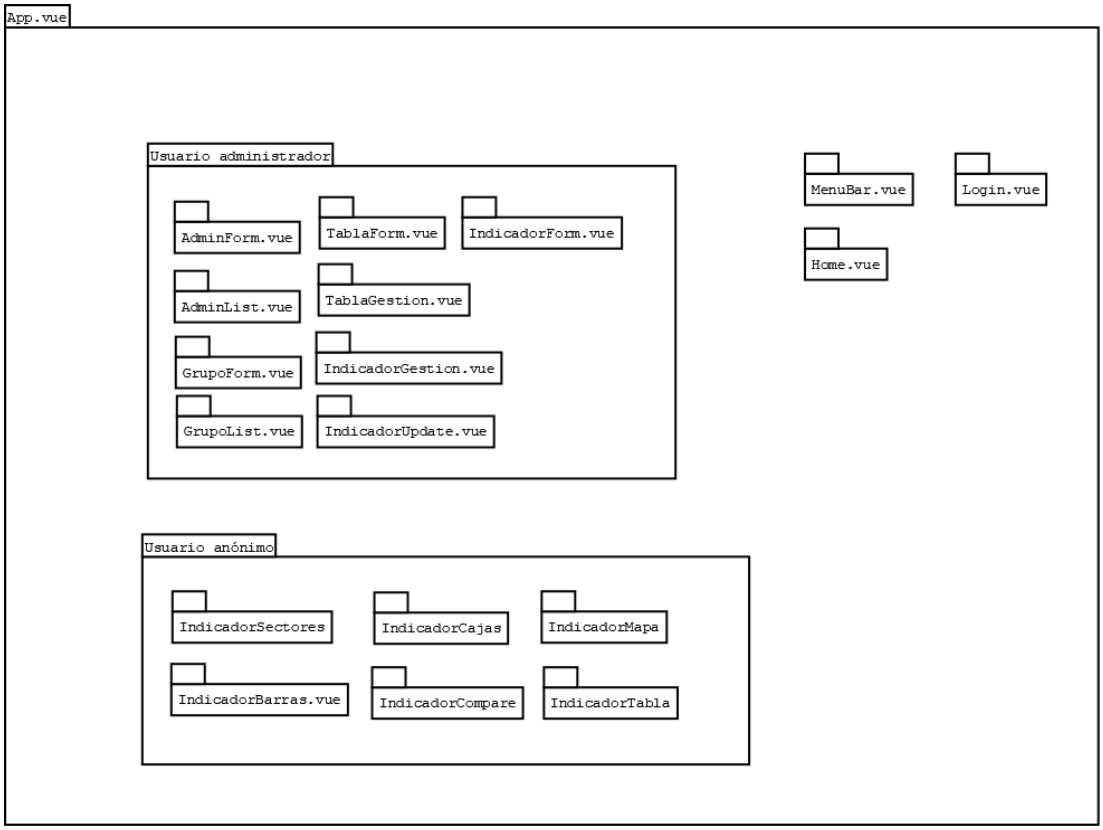


Figura 5.8: Componente de la aplicación.

Nombre	Ruta	Componente
Home	/	Home.vue
Login	/login	Login.vue
IndicadorMapa	/indicadores/:id/mapa	IndicadorMapa.vue
IndicadorTabla	/indicadores/:id/tabla	IndicadorTabla.vue
IndicadorBarras	/indicadores/:id/barras	IndicadorBarras.vue
IndicadorSectores	/indicadores/:id/sectores	IndicadorSectores.vue
IndicadorCajas	/indicadores/:id/cajas	IndicadorCajas.vue
IndicadorCompare	/comparar	IndicadorCompare.vue

Cuadro 5.6: Rutas comunes y usuario anónimo.

Nombre	Ruta	Componente
AdminCreate	/admins/new	AdminForm.vue
AdminUpdate	/admins/:login/update	AdminForm.vue
AdminList	/admins	AdminList.vue
Tablas	/tablas	TablaGestion.vue
TablaCreate	/tablas/new	TablaForm.vue
Indicadores	/indicadores	IndicadorGestion.vue
IndicadorCreate	/indicadores/new	IndicadorForm.vue
IndicadorUpdate	/indicadores/:id/update	IndicadorUpdate.vue
GrupoCreate	/grupos/new	GrupoForm.vue
GrupoUpdate	/grupos/:id/update	GrupoForm.vue
GrupoGestion	/grupos	GrupoList.vue

Cuadro 5.7: Rutas usuario administrador.

plemente tenemos que escribir el método que queremos emplear, la ruta sobre la que queremos lanzar la petición y los parámetros que fuesen necesarios. En la Figura 5.9 se muestran las peticiones lanzadas desde los componentes comunes y componentes de administrador contra el servidor. Así mismo en la Figura 5.10 se muestran las peticiones lanzadas desde los componentes del usuario anónimo contra el servidor.

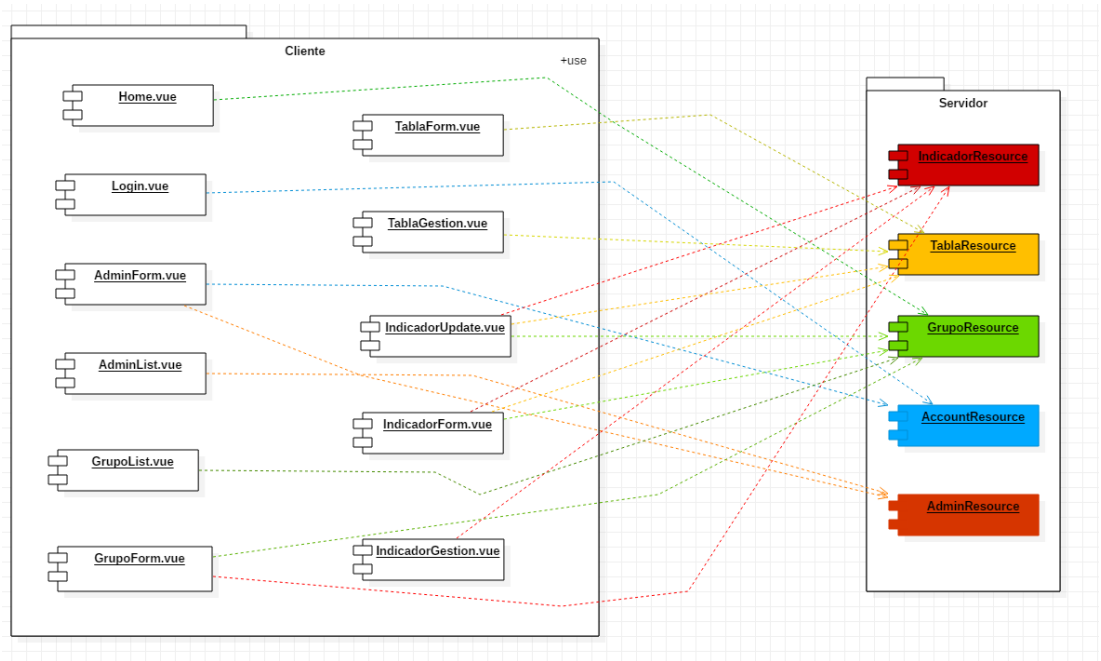


Figura 5.9: Comunicación entre componentes comunes / administrador y servidor.

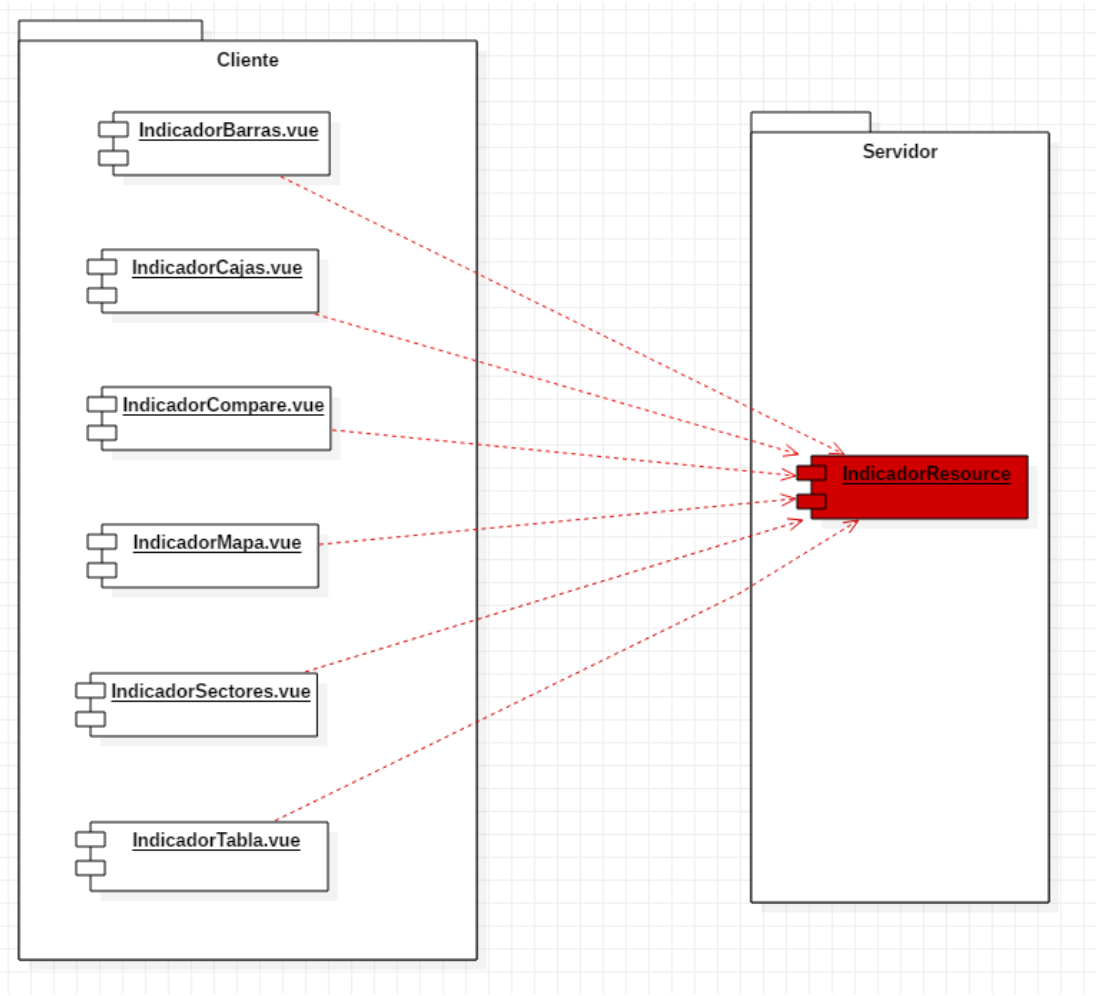


Figura 5.10: Comunicación entre componentes usuario anónimo y servidor.

Capítulo 6

Implementación y pruebas

En esta sección se detallarán los algoritmos complejos y los detalles de la implementación que no se han explicado en las secciones anteriores. También se mencionarán las pruebas llevadas a cabo.

6.1 Implementación

6.1.1 Envío de emails a administradores

Con el objetivo de proporcionar mayor información a los administradores se ha desarrollado un pequeño sistema de notificaciones mediante correo electrónico. Para ello se ha creado el servicio de mail que se puede ver en la Figura 5.6. Se trata de un pequeño servicio que se encarga de enviar un email al administrador correspondiente en función de la operación que llame al servicio. Actualmente el servicio funciona a la hora de dar de alta un administrador, a la hora de crear un indicador y a la hora de modificar un indicador. Cuando damos de alta un administrador enviará el email al mail que hayamos indicado para el usuario registrado, cuando creemos un indicador enviará un email al mail del administrador que haya creado el indicador y cuando modifiquemos un indicador se enviará el email al mail del administrador que esté modificando el indicador. Si se quisieran implementar nuevos mails simplemente habría que añadir la función al servicio e invocarla desde donde la necesitemos. Se han configurado tanto el usuario como la contraseña en el fichero application.yml que se encuentra en /src/main/resources. Se puede ver un ejemplo de función para enviar mails en Enviar mail

Función para enviar mail

```
public static void sendMailCreateIndicador(IndicadorDTO indicador) {

    final String username = username;
    final String password = password;
      Properties prop = new Properties() ;
       prop.put("mail.smtp.host", "smtp.gmail.com");
```

```
        prop.put("mail.smtp.port", "587");
        prop.put("mail.smtp.auth", "true");
        prop.put("mail.smtp. starttls .enable", "true"); //TLS
        Session session = Session.getInstance(prop,
                new javax.mail.Authenticator() {
                    protected PasswordAuthentication getPasswordAuthentication() {
                        return new PasswordAuthentication(username, password);
                    }
                });
        try {
            Message message = new MimeMessage(session);
            message.setFrom(new InternetAddress(EMAIL_FROM));
            message.setRecipients(Message.RecipientType.TO,
                    InternetAddress.parse(indicador.getMailCreador(), false));
            message.setSubject("Creacion del indicador geográfico " + indicador.getNombre());
            message.setText("Hola, se ha creado el indicador geográfico " + indicador.getNombre() + " con los
                siguientes atributos:"
                + "\n-Tabla del indicador: " + indicador.getTabla()
                + "\n-Unidad de medida: " + indicador.getUnidad()
                + "\n-Visualizaciones asociadas: " + indicador.getVisualizaciones().toString());
            message.setSentDate(new Date());
            Transport.send(message);
        } catch (MessagingException e) {
            e.printStackTrace();
        }

    }
```

6.1.2 Algoritmo de subida de ficheros shapefile

Para poder convertir los ficheros shapefile en tablas de la base de datos se desarrolló un algoritmo para descomprimir los ficheros y almacenarlos en base de datos como tablas. El algoritmo está formado por 3 funciones individuales, cada una con una funcionalidad bien definida. La primera de ellas es la que se encarga de descomprimir los archivos, ya que obligatoriamente es necesario subir los shapefile en un archivo .zip para poder trabajar con ellos porque además del .shp necesitamos otros ficheros para tener la información geográfica al completo. Esta función hace uso de la clase zipFile de java [26] que nos proporciona el método extracAll para extraer el contenido del .zip a una carpeta temporal. Una vez extraído el contenido, la función buscará el fichero con la extensión .shp y se quedará con su nombre para que lo use la función que almacena en base de datos la información. La función en detalle se puede observar en Descomprimir archivo. La segunda función es la función principal del algoritmo y es la que se encarga de almacenar los datos de los ficheros extraídos en la base de datos. Esta función se encarga de crear las columnas necesarias a partir de los archivos que tenemos y va almacenando datos obtenidos de los archivos. También se recalcula el sistema de coordenadas de la tabla ya que Leaflet solo puede trabajar con EPSG:4326 y muchos de los

ficheros que nos suban podrían no tener el sistema que quiere Leaflet. Para poder pasar de un fichero a una tabla la función crea un objeto SimpleFeatureTypeBuilder en el que añadirá las columnas que va a tener la tabla y el nombre de la tabla. A partir de este objeto creará otro objeto de tipo SimpleFeatureCollection al que le añadirá los datos que necesita la tabla, y establecerá una conexión con la base de datos para almacenar este objeto ya que Postgis es capaz de entenderlo. La función en detalle se puede ver en Convertir shapefile en tabla. Por último tenemos una función que se encarga de eliminar la carpeta temporal creada durante la extracción. Esta carpeta solo se puede eliminar una vez han sido subidos todos los ficheros o en caso de fallo, ya que las carpetas se crean localmente en el equipo del usuario y tenemos que liberar el espacio que ocupen estas carpetas temporales. La función en detalle se puede ver en Borrar carpeta.

Función para descomprimir archivos

```
public void unZipIt( MultipartFile file ) throws IOException{
    File zip = File.createTempFile(UUID.randomUUID().toString(), "temp");
    FileOutputStream o = new FileOutputStream(zip);
    IOUtils.copy(file.getInputStream(), o);
    o.close();
    OUTPUT_FOLDER = "C:\\destination";
    try {
        ZipFile zipFile = new ZipFile(zip);
        zipFile.extractAll(OUTPUT_FOLDER);
    } catch (ZipException e) {
        e.printStackTrace();
    } finally {
        zip.delete();
    }
    File dir = new File(OUTPUT_FOLDER);
    File[] directoryListing = dir.listFiles();
    if (directoryListing != null) {
      for (File child : directoryListing ) {
          if (FilenameUtils.getExtension(child.getAbsolutePath()).equals("shp")) {
              if (!SHP_FILE.equals("")) SHP_FILE = "";
              SHP_FILE = child.getName();
              }
      }
    }
}
```

Función para convertir shapefile en una tabla

```
public void uploadShapefile(String nombre) throws IOException, NoSuchAuthorityCodeException,
      FactoryException, MismatchedDimensionException, TransformException{

  FileDataStore ds = FileDataStoreFinder.getDataStore(new File(OUTPUT_FOLDER + "/" + SHP_FILE));
   SimpleFeatureType schema = ds.getSchema();
   CoordinateReferenceSystem dataCRS = schema.getCoordinateReferenceSystem();
```

```
CoordinateReferenceSystem targetCRS = CRS.decode("EPSG:4326",true);
SimpleFeatureTypeBuilder builder = new SimpleFeatureTypeBuilder();
builder.setName(schema.getName());
builder.setName(nombre); // aquí ponemos el nombre que nos inserten
builder.setSuperType((SimpleFeatureType) schema.getSuper());
builder.addAll(schema. getAttributeDescriptors ());
SimpleFeatureType nSchema = builder. buildFeatureType ();
List<SimpleFeature> features = new ArrayList<SimpleFeature>();
SimpleFeatureCollection ft = ds.getFeatureSource (). getFeatures ();
ReprojectingFeatureCollection rfc = new ReprojectingFeatureCollection (ft, targetCRS);
SimpleFeatureIterator itr = rfc. features ();
try {
    while (itr.hasNext()) {
        SimpleFeature f = itr.next();
        SimpleFeature f2 = DataUtilities .reType(nSchema, f);
        features.add(f2);
    }
}
finally {
  itr. close ();
  ds. dispose ();
}
Map<String, Object> params = new HashMap<>();
params.put("dbtype", "postgis");
params.put("host", "localhost");
params.put("port", 5432);
params.put("schema", "public");
params.put("database", "tfgdb");
params.put("user", user);
params.put("passwd", password);
params.put("charset", "UTF-8");
DataStore dataStore = DataStoreFinder.getDataStore(params);
String tableName = nSchema.getTypeName();
boolean exists = false;
String [] names = dataStore.getTypeNames();
for (String name : names) {
  if (name.equalsIgnoreCase(tableName)) {
    exists = true;
    break;
  }
}
if (! exists ) {
  dataStore.createSchema(nSchema);
}
SimpleFeatureSource source = dataStore.getFeatureSource(tableName);
if (source instanceof SimpleFeatureStore) {
  SimpleFeatureStore store = (SimpleFeatureStore) source;
  try {
      store.addFeatures( DataUtilities . collection (features));
  } finally {
      System.err. println ("Fallo");
  }
} else {
```

```
            System.err . println ("Unable to connect to database");
        }
        dataStore . dispose () ;
    }
```

Función para borrar carpetas

```
    void deleteDirectoryStream (Path path) throws IOException {
      Files . walk(path)
        . sorted (Comparator.reverseOrder())
        .map(Path:: toFile )
        .forEach( File :: delete ) ;
    }
```

6.1.3 Uso de topojson en lugar de geojson

Para pintar los mapas de la aplicación en un principio se optó por emplear Geojson ya que leaflet puede trabajar directamente con este formato sin necesitar transformaciones especiales. Pero a la hora de dibujar mapas complejos con muchos polígonos y figuras, el rendimiento bajaba considerablemente y las pantallas con mapas tardaban mucho en cargar, por ello se cambiaron los geojson por topojson [27].

Topojson es una extensión de Geojson desarrollada para mejorar el rendimiento ya que en lugar de pintar dos veces las intersecciones de los polígonos crea unos arcos con las coordenadas y solo pinta cada arco una sola vez, mejorando así la velocidad de carga de los mapas.

Para poder usar topojson primero tenemos que recuperar los datos de nuestro indicador como un json y enviarlos desde el servidor al cliente. Recuperar los datos como json se hace con la siguiente consulta:

Consulta para obtener geojson

```
ResultSet resultSet = statement.executeQuery("SELECT row_to_json(fc)\r\n" +
  " FROM ( SELECT 'FeatureCollection' As type, array_to_json(array_agg(f)) As features \r\n" +
  " FROM (SELECT 'Feature' As type\r\n" +
  "     , ST_AsGeoJSON(ST_SnapToGrid(lg.the_geom, 0.001))::json As geometry\r\n" +
  "     , row_to_json((SELECT l FROM (SELECT "+ atributo +", " + lugar + " as nombre) As l\r\n" +
  "       )) As properties \r\n" +
  "   FROM "+ table +" As lg ) As f ) As fc;");
```

Y en el servicio se añaden al dto del indicador en el campo 'datos', así ya se podrían emplear en el cliente. Este json está compuesto por numerosos elementos y cada elemento incluye las coordenadas completas de un lugar, el valor del indicador para ese lugar y el nombre del lugar. Una vez en el cliente, primero tendríamos que crear un geojson a partir del json que hemos recibido y después convertir este geojson a topojson para que leaflet pueda trabajar con él ya que de base no entiende el formato si no haces las transformaciones. Una vez tenemos el

topojson, podemos pasarle los datos a nuestro mapa y asignarle comportamientos, esto es, al pasar el ratón, quitar el ratón o hacer click sobre alguno de los lugares pintados en el mapa.

Con esto ya tendríamos un mapa pintado con el comportamiento asociado al click o a pasar el ratón por encima de un lugar pero nos faltarían la leyenda y los colores que hemos establecido durante la creación del indicador. Para crear la leyenda se usará una función que mediante un bucle for irá iterando por la lista de rangos del indicador e irá añadiendo el rango y su color asignado al html, de esta manera tendremos en un div la leyenda creada con los colores y rangos. Para darle color al mapa se usará un bucle for que iterará por todos los elementos del json mirando el valor del indicador y comparando con los rangos que tenemos definidos para el indicador, si el valor es menor que el que está comparando le asignará el color asociado a ese rango al lugar en el mapa. A mayores de usar el topojson se redujo el número de decimales en las coordenadas para mejorar el rendimiento, esto provoca que tengamos algo menos de detalle en los polígonos pero no se aprecia en zoom medio o largo, así que se ha dejado así para tener una aplicación lo más rápida posible.

El resultado final del mapa se puede apreciar en Figura 6.1

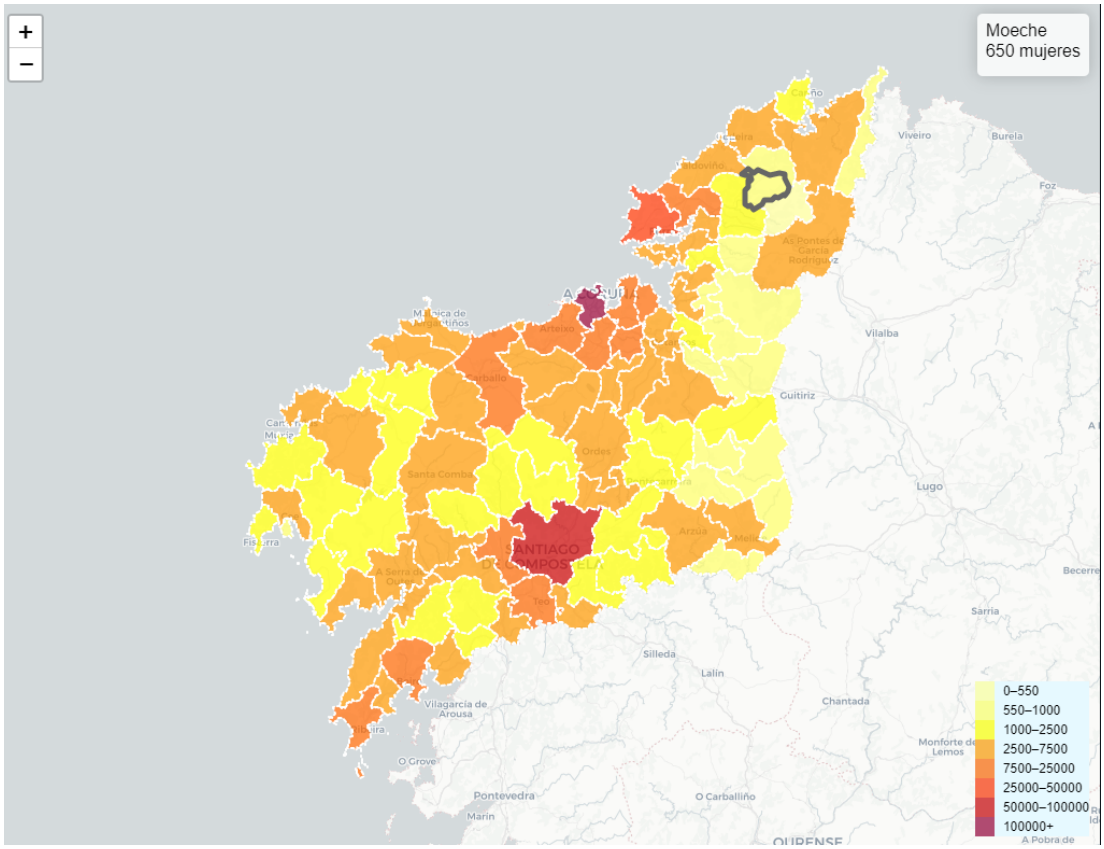


Figura 6.1: Mapa final.

6.2 Pruebas

6.2.1 Pruebas unitarias

Consisten en probar elementos individuales del software. Para llevar a cabo estas pruebas se hará uso del soporte para tests de Springboot. Para ello se tiene que añadir la dependencia spring-boot-starter-test en el pom.xml. Para poder llevar a cabo las pruebas será necesaria la anotación @RunWith(SpringRunner.class): Se emplea para proporcionar un puente entre las funciones de prueba SpringBoot y JUnit. Se han realizado pruebas sobre los servicios para comprobar que las funcionalidades que ofrecen trabajan correctamente. Se pueden ver algunas de las pruebas en las Figura 6.2, Figura 6.3, Figura 6.4, Figura 6.5

```
@Test
public void findAllAndRemoveTest() throws UserLoginExistsException {
    adminService.registerUser("patricia", "patricia", "plorenzo@hotmail.com");
    adminService.registerUser("manuel", "manuel", "ASDF@hotmail.com");
    adminService.registerUser("ana", "ana", "asffsbfdhd@hotmail.com");
    adminService.registerUser("migue", "migue", "ploreerwtyweafsadnzo@hotmail.com");

    assertEquals(4, adminService.findAll().size());

    adminService.removeUser(adminService.findByLogin("patricia").getId());
    adminService.removeUser(adminService.findByLogin("manuel").getId());

    assertEquals(2, adminService.findAll().size());

    adminService.removeUser(adminService.findByLogin("ana").getId());
    adminService.removeUser(adminService.findByLogin("migue").getId());

    assertEquals(0, adminService.findAll().size());

}

@Test
public void findByLoginTest() throws UserLoginExistsException {
    adminService.registerUser("ana", "ana", "asffsbfdhd@hotmail.com");
    adminService.registerUser("migue", "migue", "ploreerwtyweafsadnzo@hotmail.com");

    AdminDTO ana = adminService.findByLogin("ana");
    AdminDTO migue = adminService.findByLogin("migue");
    assertEquals("ana", ana.getLogin());
    assertEquals("migue", migue.getLogin());

    adminService.removeUser(adminService.findByLogin("ana").getId());
    adminService.removeUser(adminService.findByLogin("migue").getId());


}
```

Figura 6.2: Test AdminService - 1.

```java
@Test
public void updateUserTest() throws UserLoginExistsException {
    adminService.registerUser("ana", "ana", "asffsbfdhd@hotmail.com");

    AdminDTO ana = adminService.findByLogin("ana");
    ana.setLogin("anita");
    Admin anita = new Admin();
    anita.setId(ana.getId());
    anita.setEmail(ana.getEmail());
    anita.setLogin(ana.getLogin());
    adminService.updateUser(anita);
    assertEquals("anita", anita.getLogin());

    adminService.removeUser(adminService.findByLogin("anita").getId());


}
```

Figura 6.3: Test AdminService - 2.

```java
@Test
public void createAndFindIndicadorTest() throws SQLException {
    // given
    Indicador p_mujeres = new Indicador("Poblacion femenina", false, "municipios", "mujeres");
    IndicadorDTO saved = new IndicadorDTO(p_mujeres);

    indicadorComponenteService.save(saved);

    // when
    IndicadorDTO found = indicadorComponenteService.findById(saved.getId());
    assertEquals(saved, found);
}

@Test
public void findAllAndDeleteTest() {
    // given
    Indicador p_1 = new Indicador("Poblacion femenina", false, "municipios", "mujeres");
    IndicadorDTO saved_1 = new IndicadorDTO(p_1);
    Indicador p_2 = new Indicador("Poblacion masculina", false, "municipios", "hombres");
    IndicadorDTO saved_2 = new IndicadorDTO(p_2);
    Indicador p_3 = new Indicador("Poblacion total", false, "municipios", "total");
    IndicadorDTO saved_3 = new IndicadorDTO(p_3);

    indicadorComponenteService.save(saved_1);
    indicadorComponenteService.save(saved_2);
    indicadorComponenteService.save(saved_3);
    // when
    List<IndicadorDTO> found = indicadorComponenteService.findAllIndicadores();
    assertEquals(3, found.size());

    indicadorComponenteService.deleteIndicadorById(saved_1.getId());
    indicadorComponenteService.deleteIndicadorById(saved_2.getId());
    indicadorComponenteService.deleteIndicadorById(saved_3.getId());

}
```

Figura 6.4: Test IndicadorComponenteService - 1.

```java
@Test
public void createAndFindGruposTest() {
    // given
    Indicador p_1 = new Indicador("Poblacion femenina", false, "municipios", "mujeres");
    IndicadorDTO saved_1 = new IndicadorDTO(p_1);
    Indicador p_2 = new Indicador("Poblacion masculina", false, "municipios", "hombres");
    IndicadorDTO saved_2 = new IndicadorDTO(p_2);
    List<Indicador> indicadores = new ArrayList<Indicador>();
    indicadores.add(p_1);
    indicadores.add(p_2);
    GrupoIndicador g_1 = new GrupoIndicador("grupo_1", indicadores);
    GrupoIndicadorDTO g_dto = new GrupoIndicadorDTO(g_1);


    indicadorComponenteService.save(saved_1);
    indicadorComponenteService.save(saved_2);
    indicadorComponenteService.saveGrupo(g_dto);

    GrupoIndicadorDTO g_found = indicadorComponenteService.findGrupoById(g_dto.getId());
    // when
    assertEquals(2, g_dto.getIndicadores().size());
    assertEquals(g_dto, g_found);

    indicadorComponenteService.deleteIndicadorById(saved_1.getId());
    indicadorComponenteService.deleteIndicadorById(saved_2.getId());
    indicadorComponenteService.deleteGrupoById(g_1.getId());

}
```

Figura 6.5: Test IndicadorComponenteService - 2.

6.2.2 Pruebas REST

Usando Postman se han lanzando peticiones HTTP contra las urls que tenemos disponibles en la aplicación para comprobar que devolvían el resultado esperado. Algunas de estas pruebas se pueden ver en las Figura 6.6, Figura 6.7, Figura 6.8, Figura 6.9, Figura 6.10

6.2.3 Pruebas de integración

Estas pruebas se han realizado mediante la ejecución de los distintos casos de uso de la aplicación, simulando a los dos tipos de usuario y probando que el funcionamiento de la aplicación era el deseado. Como usuario administrador se probó a crear, editar y eliminar indicadores, mirando que los tipos de visualización se guardaban adecuadamente, también se crear y eliminaron administradores, se subieron tablas a la base de datos, con especial énfasis en que la base de datos se crease adecuadamente. Como usuario anónimo se consultaron todos los grupos de indicadores disponibles y se miraron los indicadores en detalle, prestando especial atención al rendimiento en la vista de mapa y controlando que los rangos y colores establecidos durante la creación se pintaban correctamente.

findAllIndicadores

GET http://localhost:8080/api/indicadores

Params Authorization Headers Body Pre-request Script Tests

KEY VALUE

Key Value

Body Cookies Headers (8) Test Results

Pretty Raw Preview JSON


```
[
    {
        "id": 1,
        "nombre": "Poblacion masculina",
        "oculto": false,
        "hasGrupo": true,
        "tabla": "municipios_2",
        "unidad": "hombres",
        "visualizaciones": [
            {
                "id": 2,
                "configuracion": {
                    "atributo": "p_hombres",
                    "lugar": "nombre",
                    "columnaOrden": "atributo",
                    "colores": [
                        "#990033",
                        "#cc0000",
                        "#ff3300",
                        "#ff6600",
                        "#ff9900",
                        "#ffff00",
                        "#ffff66",
                        "#ffff99"
                    ],
                    "rangos": [
                        "100000",
                        "50000",
                        "25000",
                        "7500",
```


Figura 6.6: Test findAllIndicadores.

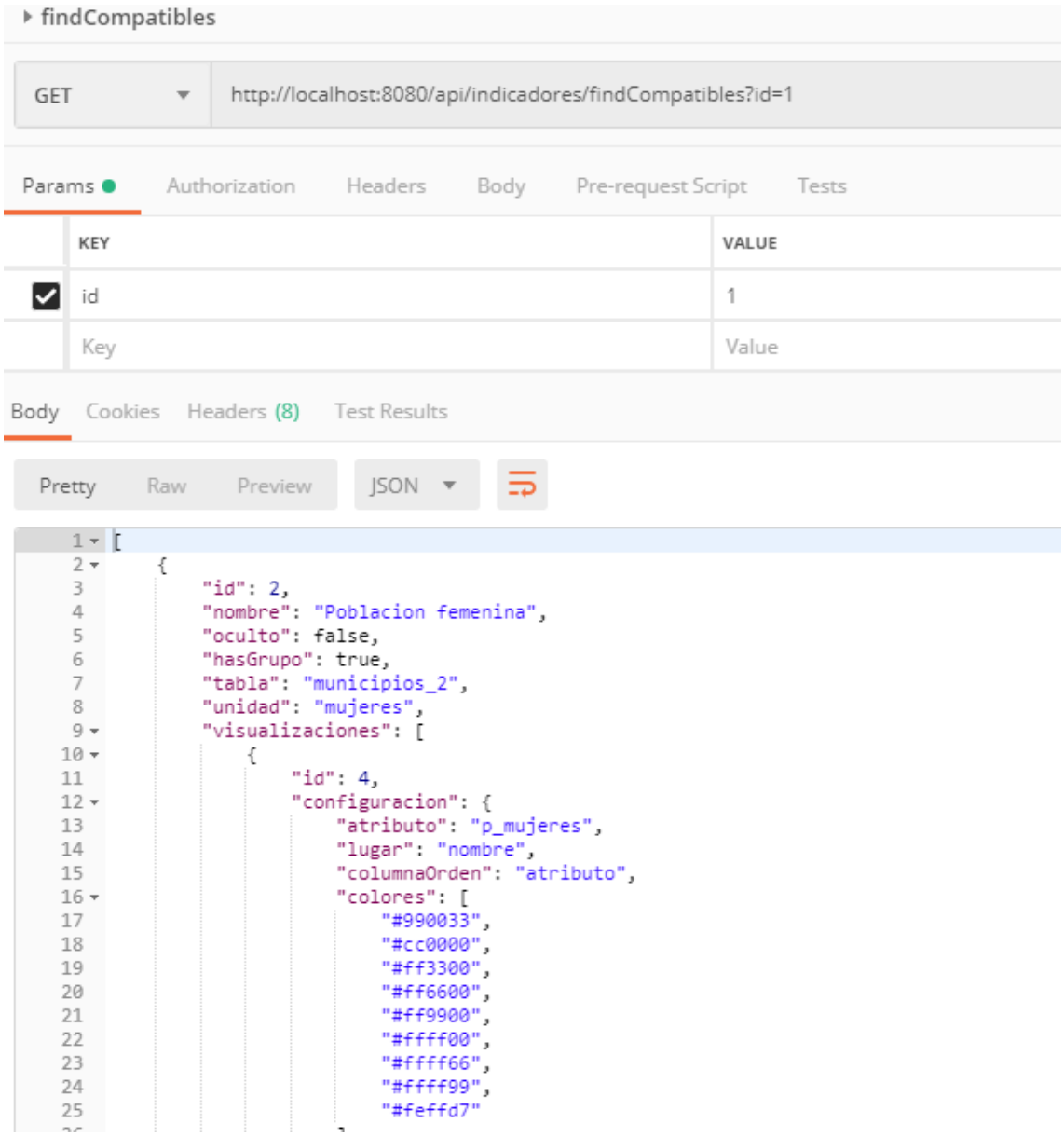


Figura 6.7: Test findCompatibles.

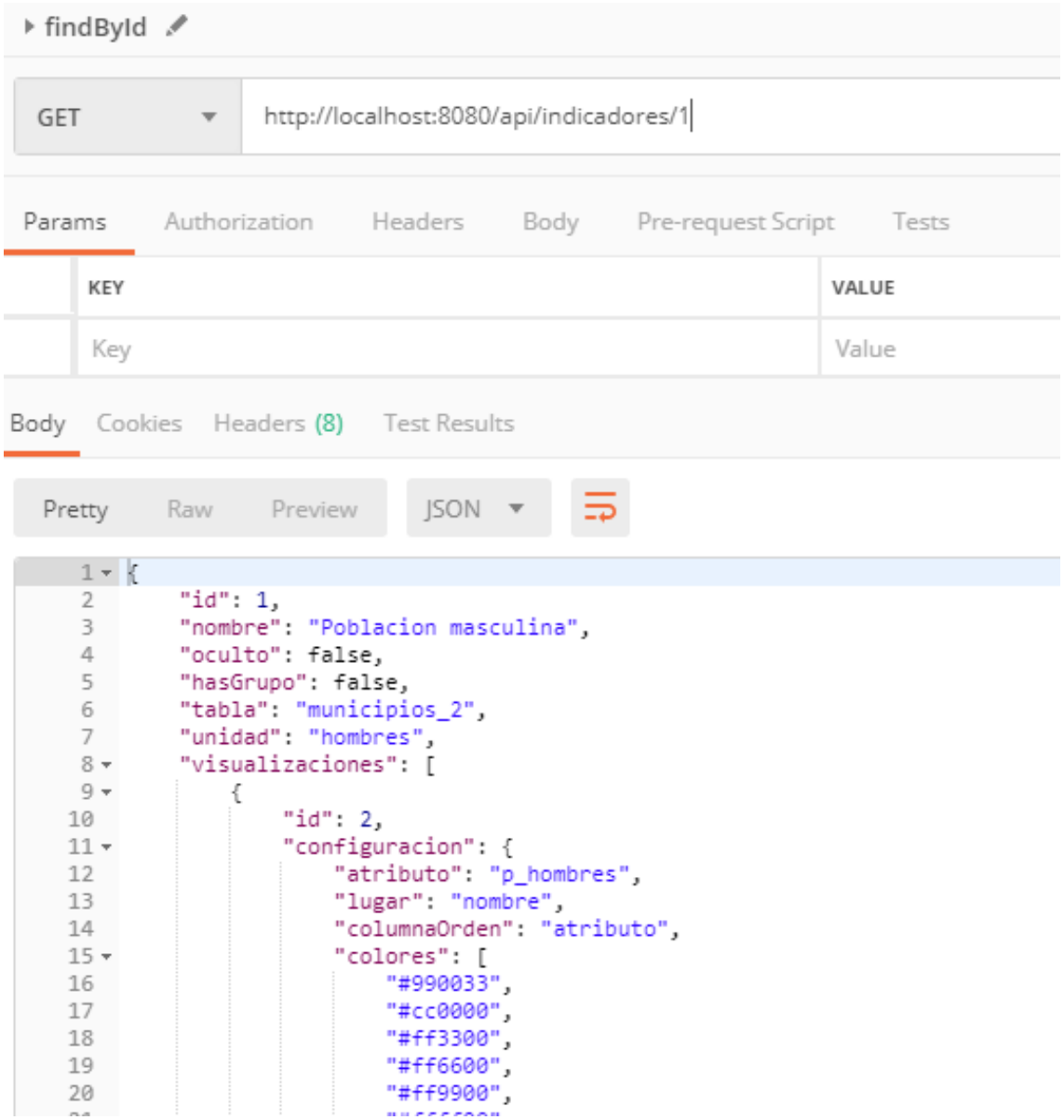


Figura 6.8: Test findIndicador.

createIndicador

POST http://localhost:8080/api/indicadores

Params Authorization Headers (1) Body Pre-request Script

none form-data x-www-form-urlencoded raw binary

```
{
    "nombre": "Poblacion prueba",
    "oculto": false,
    "hasGrupo": false,
    "tabla": "municipios_2",
    "unidad": "mujeres",
    "visualizaciones": [
        {
            "id": 4,
            "configuracion": {
                "atributo": "p_mujeres",
                "lugar": "nombre"
```

Body Cookies Headers (8) Test Results

Pretty Raw Preview JSON

```
{
    "id": 116,
    "nombre": "Poblacion prueba",
    "oculto": false,
    "hasGrupo": false,
    "tabla": "municipios_2",
    "unidad": "mujeres",
    "visualizaciones": [
        {
            "id": 274,
            "configuracion": {
                "atributo": "p_mujeres",
                "lugar": "nombre",
                "columnaOrden": "atributo",
                "colores": [
                    "#990033",
                    "#cc0000",
```

Figura 6.9: Test createIndicador.

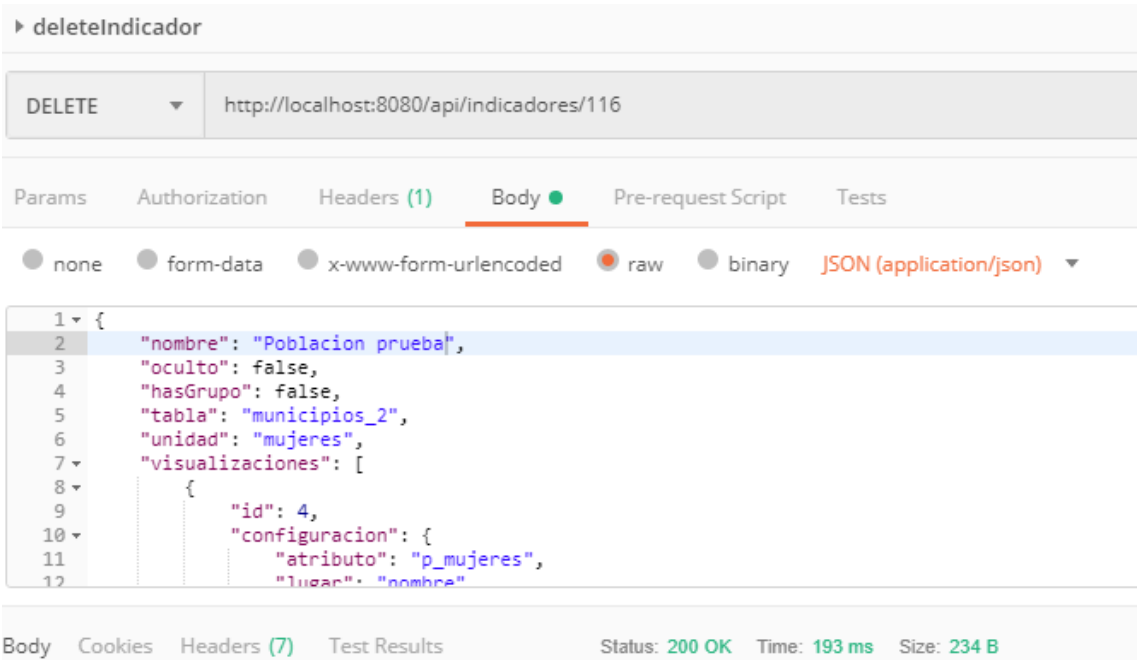


Figura 6.10: Test deleteIndicador.

Capítulo 7

Solución desarrollada

En este capítulo vamos a mostrar las características principales de la aplicación desarrollada. Se van a dividir las pantallas de la aplicación entre las del usuario administrador y las del usuario anónimo.

7.1 Usuario administrador

El usuario administrador se encarga de la gestión de indicadores, grupos de indicadores, tablas y otros administradores. El administrador puede crear, editar o eliminar tanto indicadores como grupos, puede crear o eliminar administradores y tablas, y a mayores puede ocultar los indicadores que desee. Para cambiar de sección puede emplear el menú lateral y hacer click en la gestión que quiera.

En la figura 7.1 podemos ver la pantalla de login en la que el administrador tiene que introducir sus credenciales para poder acceder al panel de administrador.

Una vez ha entrado en el panel de administrador tendrá disponible un menú lateral con los distintos módulos en los que puede entrar: Indicadores, Grupos, Tablas y Administradores. Si selecciona el módulo de Indicadores verá una pantalla como la de la figura 7.2 y aquí tendrá disponible una tabla con todos los indicadores que hayan sido creados, pudiendo eliminar, ocultar o acceder a la edición de cualquier indicador. Si pulsa en la sección indicadores del menú lateral saldrá una nueva opción para crear indicadores con la que podrá acceder al formulario que se puede ver en la figura 7.3. En este formulario tiene que introducir el nombre que quiere para el indicador, la tabla sobre la que consultará los datos, que atributo va a representar el indicador, que atributo se usará para consultar los nombres de los lugares, la unidad de medida, un grupo, y que tipos de visualización quiere para el indicador.

Se sigue el mismo esquema para los grupos y en la figura 7.4 podemos ver la tabla con todos los grupos creados en la aplicación así como los indicadores que incluyen.

Para gestionar las tablas tenemos que acceder al módulo de tablas y como se puede ver en

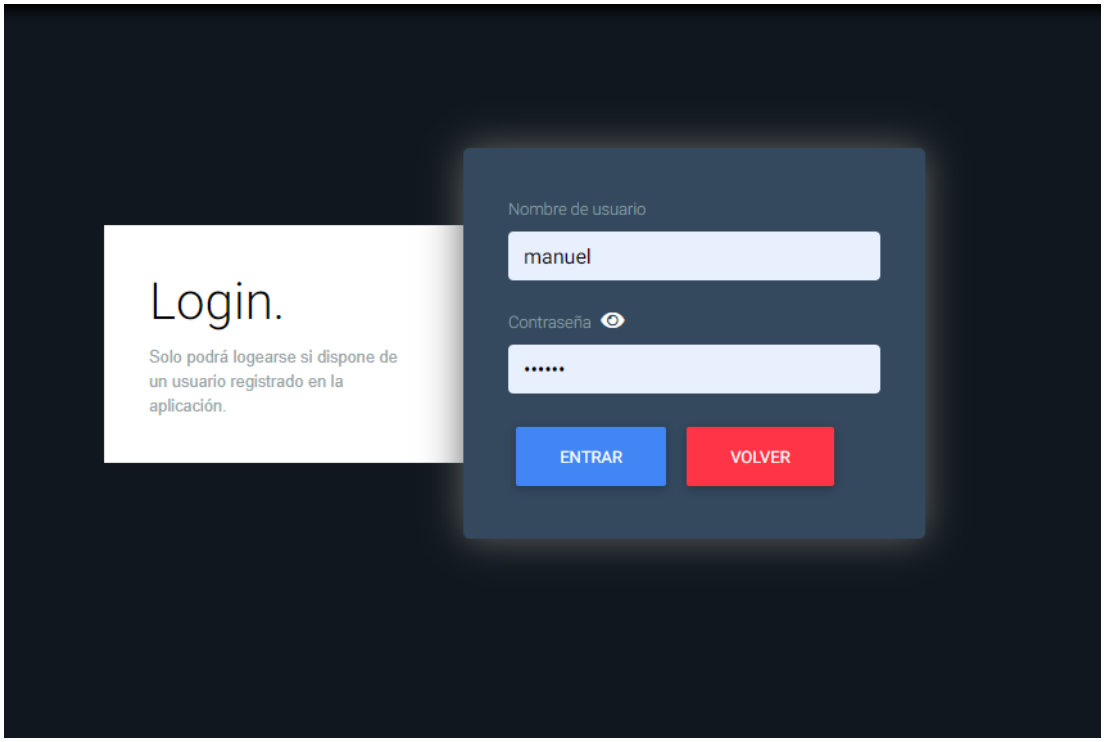


Figura 7.1: Página de login.

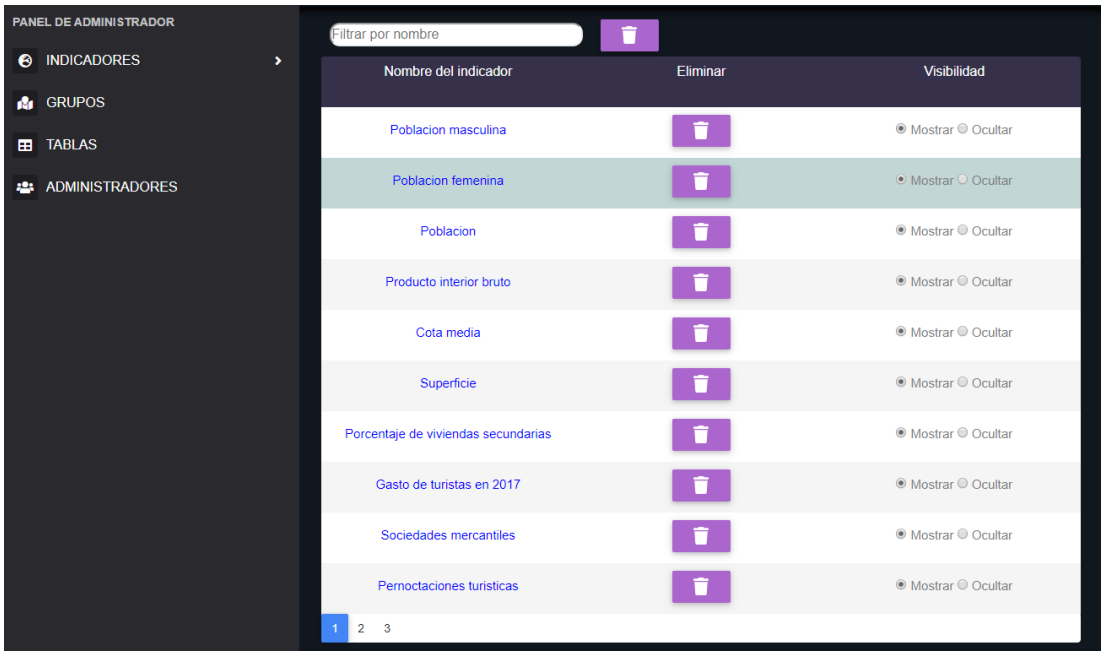


Figura 7.2: Página de gestión de indicadores.

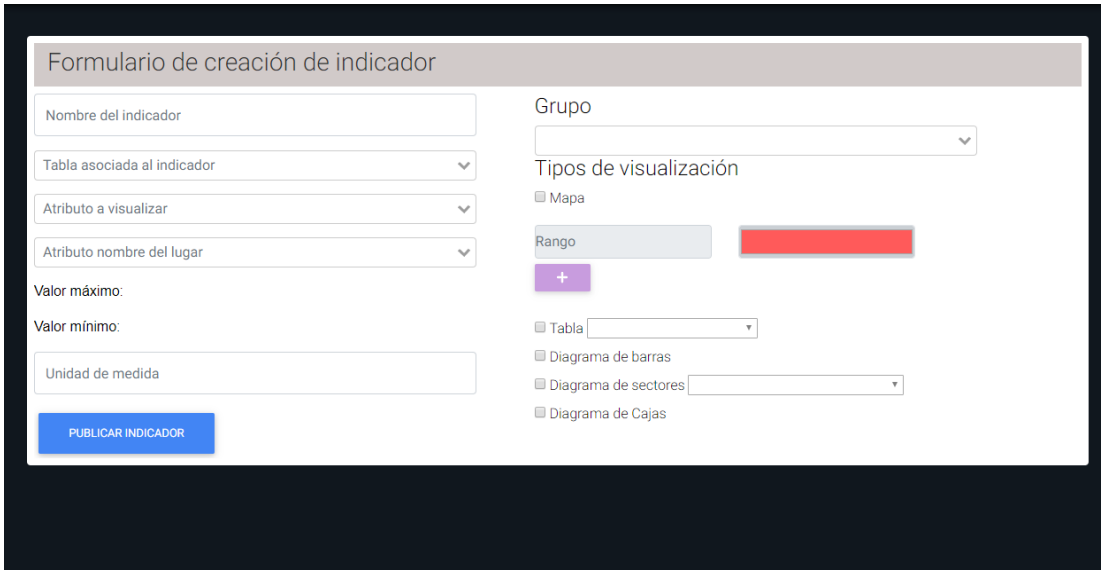


Figura 7.3: Página de creación de indicadores.

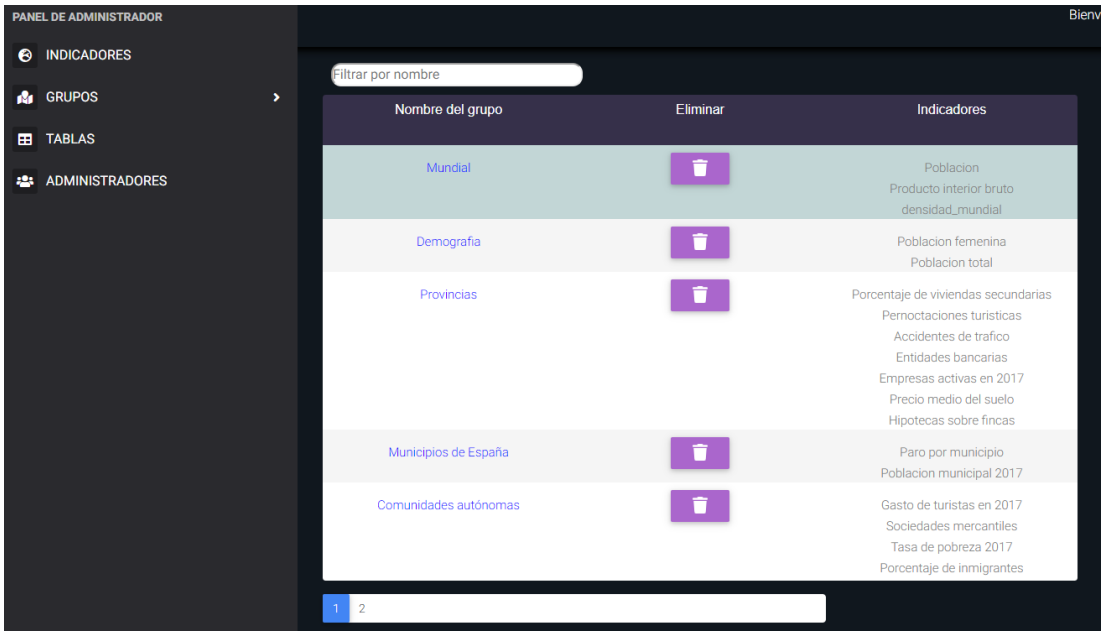


Figura 7.4: Página de gestión de grupos.

la figura 7.5 tendremos disponible una tabla que incluye todas las tablas con datos geográficos que se han creado en la base de datos, así como la opción para eliminar las tablas que queramos. En la figura 7.6 podemos ver el formulario que se usa para publicar las tablas. Aquí el usuario tiene que elegir un nombre para la tabla y subir un fichero .zip con el archivo shapefile y distintos ficheros adicionales que servirán para crear la tabla en base de datos.

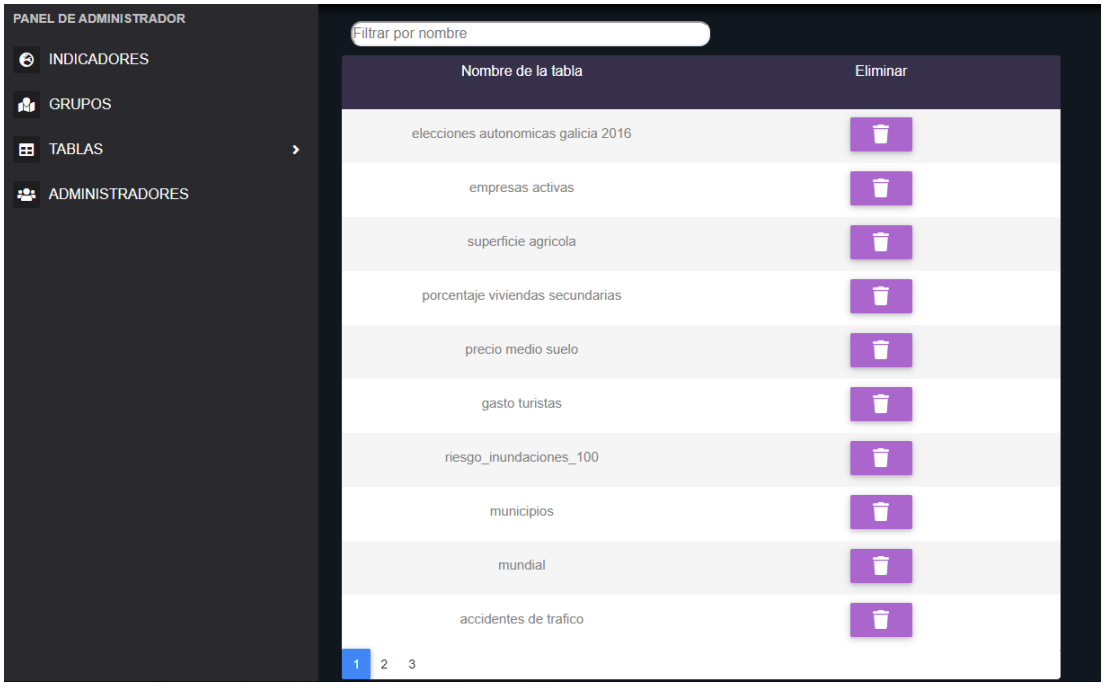


Figura 7.5: Página de gestión de tablas.

Los administradores se gestionan de la misma manera que hemos visto hasta ahora y al acceder al módulo tendremos una pantalla como la de la figura 7.7 con todos los administradores que pueden acceder al panel de administrador. En la figura 7.8 podemos ver el formulario para dar de alta administradores. En él se tienen que introducir el nombre del usuario, el login, el lugar y la fecha de nacimiento, un email válido y una contraseña. Una vez creado el administrador se envía un email al email introducido durante la creación con los datos de acceso.

Por último, si accedemos a nuestro perfil accederemos a un formulario similar al de crear administrador pero con nuestros datos actuales y podremos actualizar cualquier campo salvo el login.

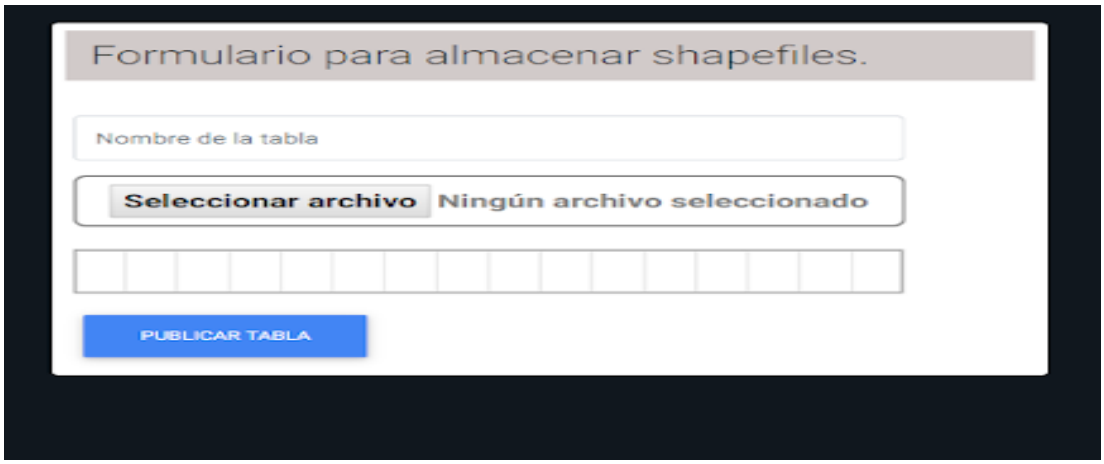


Figura 7.6: Página de creación de tablas.

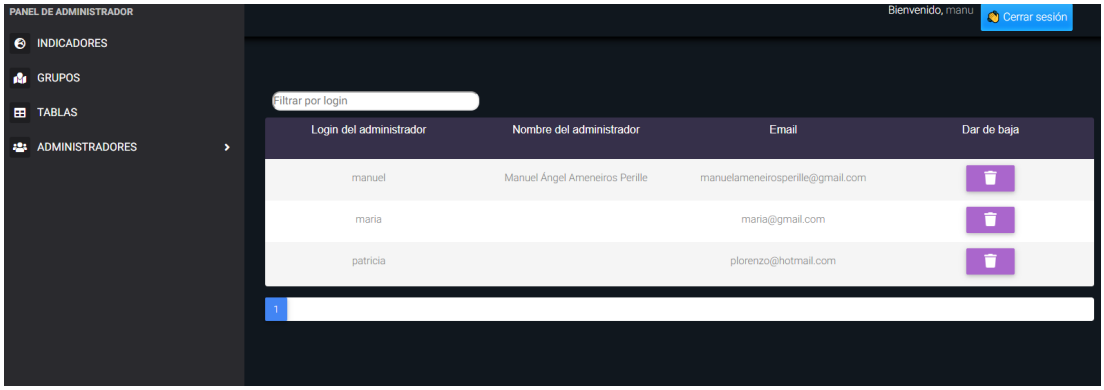


Figura 7.7: Página de gestión de administradores.

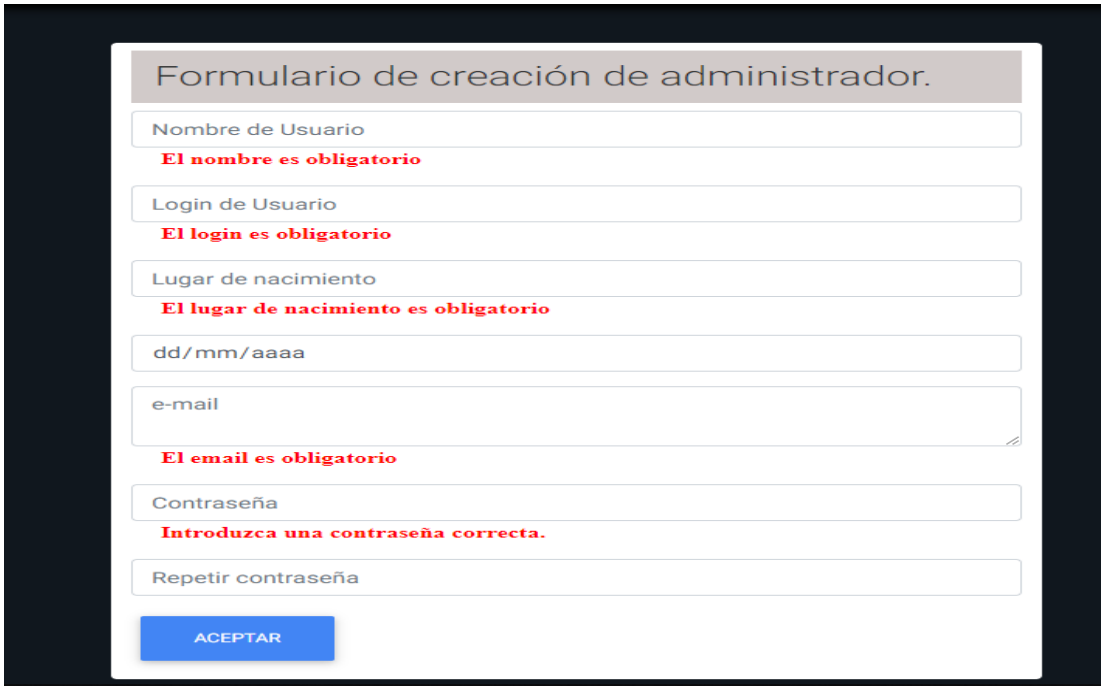


Figura 7.8: Página de creación de administradores.

7.2 Usuario anónimo

Los usuarios anónimos pueden realizar operaciones de consulta sobre los indicadores. Al entrar en la aplicación verán una pantalla como la de la figura 7.9 en la que tendrán disponible una lista de grupos creados y al hacer click en alguno de los grupos podrán ver que indicadores o subgrupos incluye. También pueden filtrar los indicadores por nombre y pueden acceder al comparador de indicadores.

Si accede a cualquier indicador se abrirá la visualización de mapa como podemos ver en la figura 7.10. En el mapa se pintarán los lugares con el color que corresponda con el valor que tiene el indicador, y podremos filtrar los resultados del mapa por nombre o por valor, así se repintará el mapa solo con los valores que cumplan el filtro.

Desde el menú lateral se puede acceder a las distintas visualizaciones que tenemos para los indicadores. En la figura 7.11 se puede ver la vista en tabla, que está compuesta con una columna para el lugar y otra para el valor del indicador. En la figura 7.12 se puede ver la vista como diagrama de barras, teniendo tantas barras como lugares y que al pasar el ratón por encima de cualquiera de ellas nos indicará el valor que tiene. En la figura 7.13 se puede ver la vista como diagrama de sectores. Tendremos un número de sectores dependiendo de la configuración elegida al crear el indicador, pudiendo tener solo los 20 mayores, solo los que superan el 5% del valor total o todos los valores. Al pasar el cursor por encima de un sector

Figura 7.9: Página de Home.

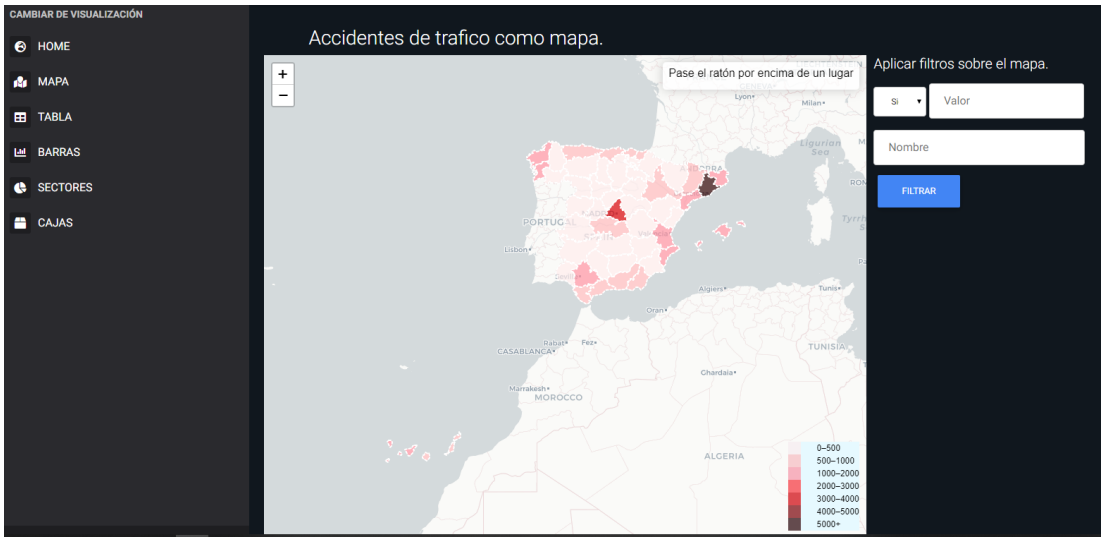


Figura 7.10: Vista en mapa de un indicador.

nos indicará su valor. Por último, en la figura 7.14 tenemos la vista como diagrama de cajas y bigotes. Aquí disponemos de una caja en la que podemos ver los cuartiles, la media y la mediana, y además los valores exteriores en los que podremos ver el valor y el lugar con el que se corresponde.

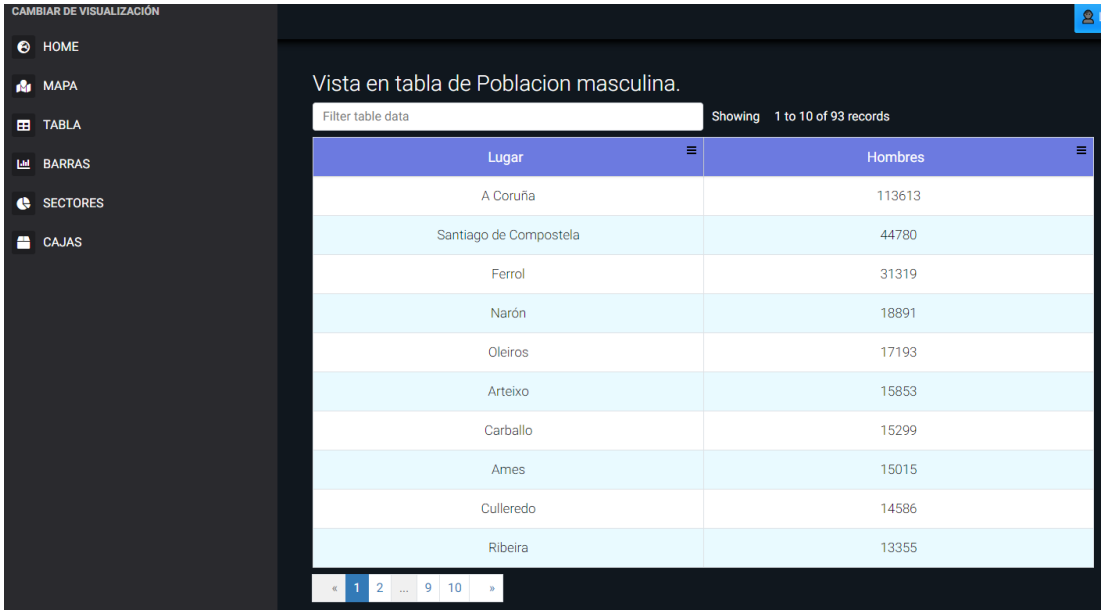


Figura 7.11: Vista en tabla de un indicador.

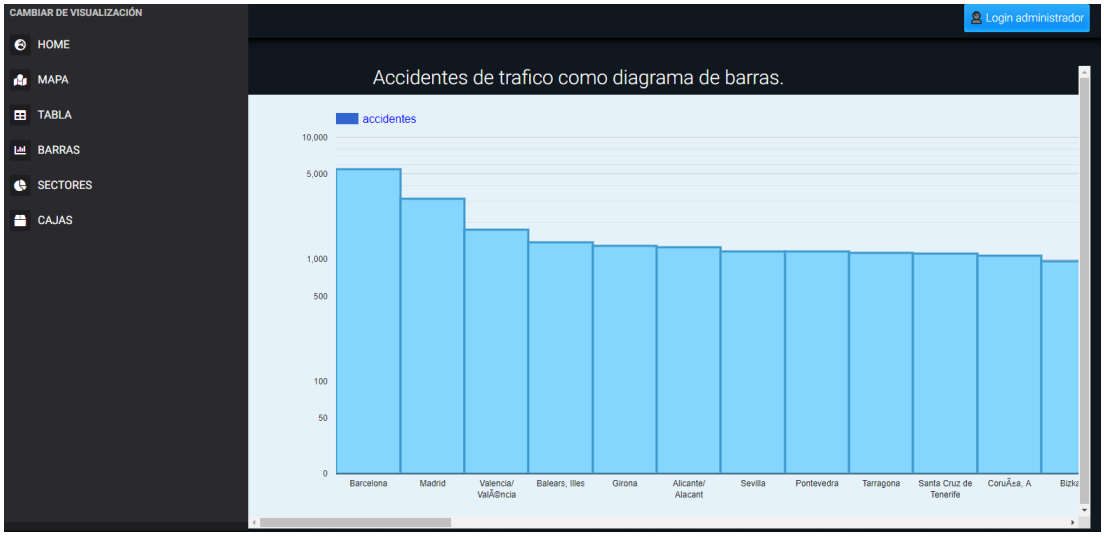


Figura 7.12: Vista en diagrama de barras de un indicador.

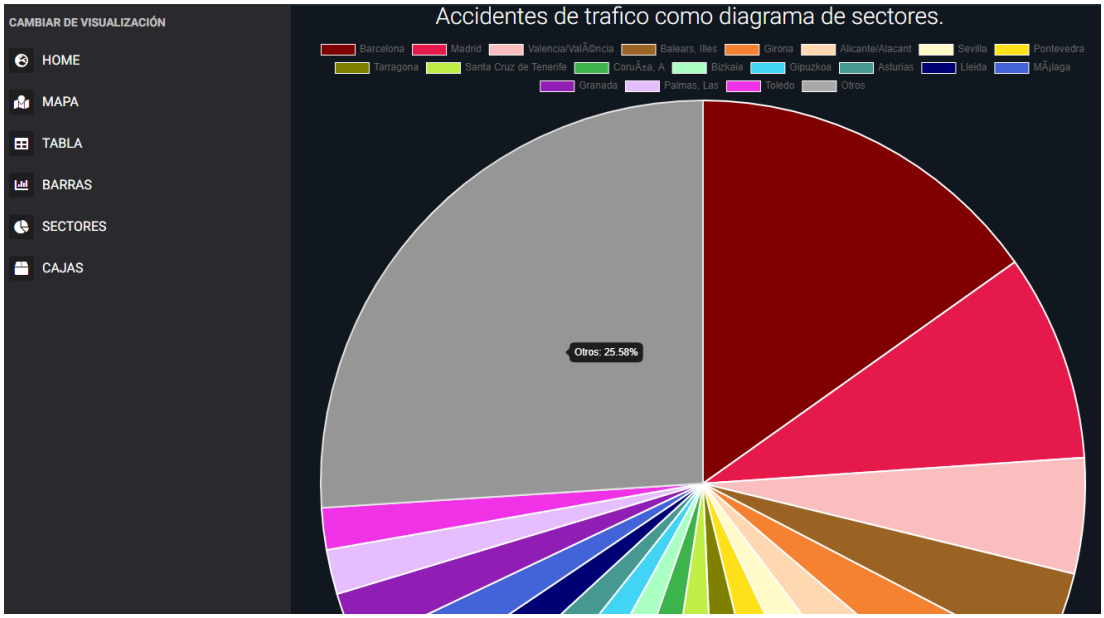


Figura 7.13: Vista en diagrama de sectores de un indicador.

Figura 7.14: Vista en diagrama de cajas de un indicador.

Capítulo 8

Conclusiones y trabajo futuro

8.1 Conclusiones

Al realizar este Trabajo de Fin de Grado se han obtenido las siguientes conclusiones:

- Se han implementado correctamente todos los objetivos establecidos en Sección 1.2. Todas las funcionalidades se han desarrollado y probado correctamente.
- Relacionado con la formación académica se han conseguido ampliar los conocimientos relacionados con tecnologías novedosas y muy presentes en el ámbito profesional. Se ha adquirido experiencia a la hora de trabajar con Spring, Hibernate, REST, desarrollar un cliente con Vue.js y Leaflet para trabajar con mapas.
- Finalmente, relacionado con la gestión de proyectos, se ha aprendido a trabajar siguiendo una metodología de desarrollo, con una planificación adecuada y llevando a cabo un seguimiento constante de los objetivos planteados.

8.2 Planes de futuro

Aunque la aplicación presenta funcionalidades interesantes y se puede usar sin problema, existen varias funcionalidades que se podrían añadir a mayores si se dispusiese del tiempo necesario.

- **Exportar indicadores como un fichero en excel.** Una funcionalidad interesante sería la que nos permita descargar los datos de un indicador como un fichero excel, así el usuario tendría disponibles los datos del indicador de forma local sin necesidad de conectarse a la aplicación. El fichero excel incluiría todos los datos del indicador, incluyendo las coordenadas de los lugares. También se podría añadir que el usuario pueda descargar varios indicadores a la vez, quedando cada indicador en una página del fichero excel.

- **Visualizar todos los indicadores asociados a un lugar.** Actualmente no disponemos de ninguna funcionalidad que nos permita visualizar todos los valores de indicadores que se han creado sobre un lugar. Se crearía una tabla con el nombre del indicador y el valor, y se accedería haciendo click en el lugar sobre el mapa. La complejidad de la operación reside en que existen varias tablas que pueden contener el lugar en cuestión, por lo que se necesitarían múltiples JOIN para recuperar toda la información.
- **Nuevos tipos de visualización.** Aunque actualmente la variedad de visualizaciones es grande, todavía se podrían añadir nuevos tipos de visualizaciones como gráficos lineares, gráfico de areas circulares o combinar varios gráficos al mismo tiempo.
- **Histórico de mapa mediante slide.** Una mejora para la visualización en mapa sería disponer de varios indicadores creados sobre los mismos datos pero para distintas fechas, de forma que pudiésemos variar el contenido del mapa modificando la fecha con un slide. Con esto podríamos crear un indicador con el clima en distintas épocas y ver la variación rápidamente al cambiar la fecha.
- **Sistema de newsletter.** Se podría permitir a los usuarios anónimos que estén al tanto de los indicadores que se vayan creando. Para ello el usuario introduciría su mail y pasaría a estar incluído en una lista de mails a los que se les enviaría el aviso de que se han creado nuevos indicadores.
- **Mejorar el sistema de notificaciones.** Aumentar las funcionalidades que generan email de 3 a todas. Tendríamos mails cuando se crean grupos de indicadores, al añadir una nueva tabla en la base de datos, al borrar cualquier elemento de base de datos, etc. Con esto se podría llevar un control exhaustivo de los datos de la aplicación.

Apéndices

Apéndice A

Glosario de acrónimos

API *Integrated Development Environment.*

BD *Base de Datos.*

CRUD *Create Read Update Delete.*

CSS *Cascading Style Sheets.*

DAO *Data Access Object.*

DTO *Data Transfer Object.*

GeoJSON *Geographic JavaScript Object Notation.*

HTML *HyperText Markup Language.*

HTTP *Hypertext Transfer Protocol.*

HTTPS *Hypertext Transfer Protocol Secure.*

IDE *Application Programming Interface.*

IoC *Inversion of Control.*

JDBC *Java Database Connectivity.*

MVC *Model View Controller.*

MVVM *Model View ViewModel.*

REST *Representational State Transfer.*

SIG *Sistema de Información Geográfico.*

URL *Uniform Resource Locator.*

Apéndice B

Patrones de diseño

B.1 Back-end

B.1.1 DAO

El patrón DAO propone separar por completo la lógica de negocio de la lógica para acceder a los datos, de esta forma, el DAO proporcionará los métodos necesarios para insertar, actualizar, borrar y consultar la información; por otra parte, la capa de negocio solo se preocupa por lógica de negocio y utiliza el DAO para interactuar con la fuente de datos.

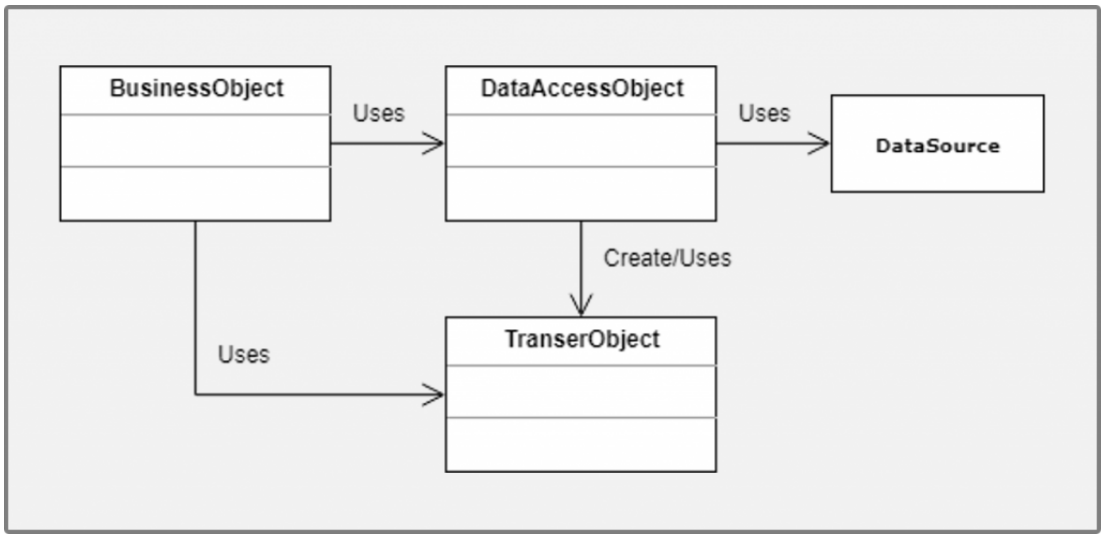


Figura B.1: Diagrama de clases del patrón DAO.[1]

Los compones que conforman el patrón son:

BusinessObject: representa un objeto con la lógica de negocio. DataAccessObject: representa una capa de acceso a datos que oculta la fuente y los detalles técnicos para recuperar los datos. TransferObject: este es un objeto plano que implementa el patrón Data Transfer Object

(DTO) Apéndice B.1.2, el cual sirve para transmitir la información entre el DAO y el Business Service. DataSource: representa de forma abstracta la fuente de datos, la cual puede ser una base de datos, Webservices, LDAP, archivos de texto, etc.

B.1.2 DTO

El patrón DTO tiene como finalidad de crear un objeto plano (POJO) con una serie de atributos que puedan ser enviados o recuperados del servidor en una sola invocación, de tal forma que un DTO puede contener información de múltiples fuentes o tablas y concentrarlas en una única clase simple.

Si bien un DTO es simplemente un objeto plano, sí que tiene que cumplir algunas reglas para poder considerar que hemos creado un DTO correctamente implementado:

- **Solo lectura:** Dado que el objetivo de un DTO es utilizarlo como un objeto de transferencia entre el cliente y el servidor, es importante evitar tener operaciones de negocio o métodos que realicen cálculos sobre los datos, es por ello que solo deberemos de tener los métodos GET y SET de los respectivos atributos del DTO.

- **Serializable:** Es claro que, si los objetos tendrán que viajar por la red, deberán de poder ser serializables, pero no hablamos solamente de la clase en sí, sino que también todos los atributos que contenga el DTO deberán ser fácilmente serializables. Un error clásico en Java es, por ejemplo, crear atributos de tipo Date o Calendar para transmitir la fecha u hora, ya que estos no tienen una forma estándar para serializarse por ejemplo en Webservices o REST.

B.1.3 MVC

El MVC o Modelo-Vista-Controlador es un patrón de arquitectura de software que, utilizando 3 componentes (Vistas, Modelos y Controladores) separa la lógica de la aplicación de la lógica de la vista en una aplicación.

- **Modelo.** Se encarga de los datos, generalmente (pero no obligatoriamente) consultando la base de datos. Actualizaciones, consultas, búsquedas, etc. todo eso va en el modelo.

- **Controlador.** Se encarga de. recibir las órdenes del usuario y se encarga de solicitar los datos al modelo y de comunicárselos a la vista.

- **Vista.** Es la representación visual de los datos, todo lo que tenga que ver con la interfaz gráfica va aquí. Ni el modelo ni el controlador se preocupan de cómo se verán los datos, esa responsabilidad es únicamente de la vista.

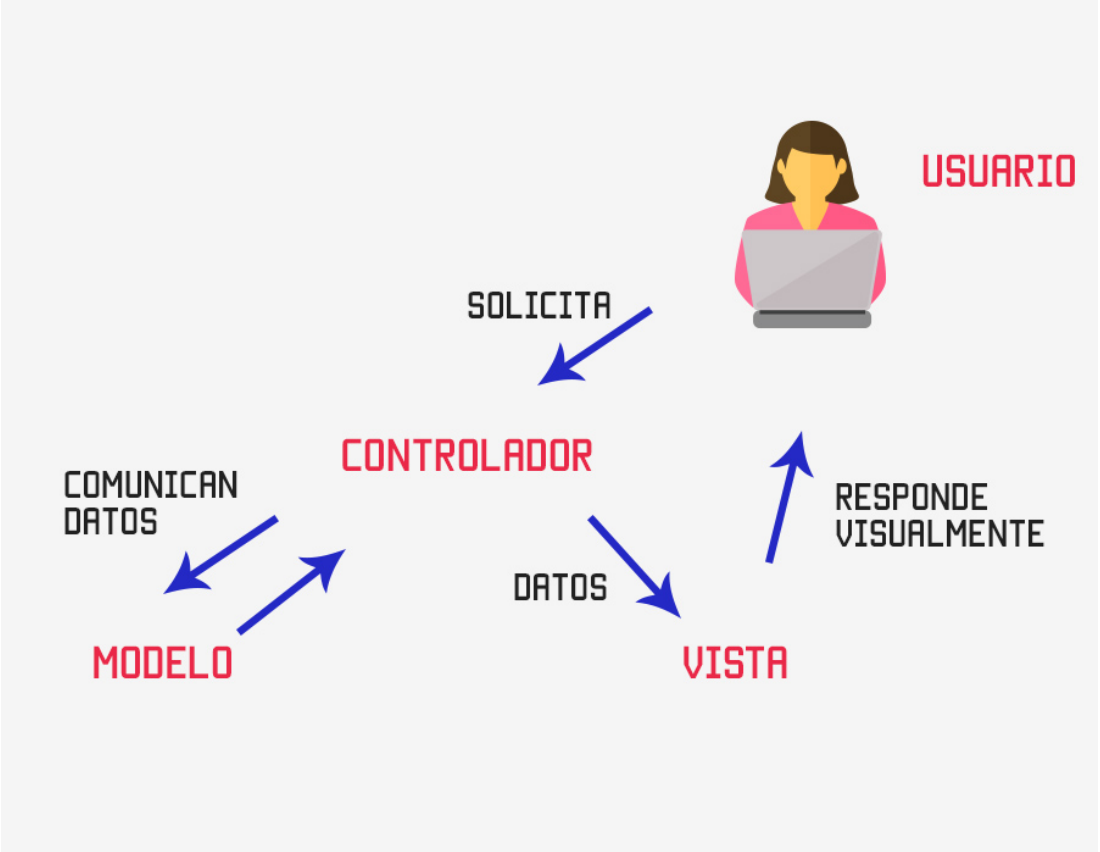


Figura B.2: Patrón MVC

B.1.4 Inyección de dependencias

El patrón de Inyección de Dependencias, también conocido como de Inversión de Control es un patrón que tiene como finalidad conseguir un código mas desacoplado, que nos facilitará las cosas a la hora de hacer Tests y además nos permite cambiar partes del sistema más fácilmente en caso de que fuese necesario.

Tener el código desacoplado nos permite cambiar las dependencias en tiempo de ejecución basándonos en cualquier factor que considerásemos, para ello necesitaríamos un Inyector o Contenedor que sería el encargado de inyectar las dependencias correctas en el momento necesario.

Siguiendo el patrón de Inyección de Dependencias (DI, Dependency Injection) los componentes declaran sus dependencias, pero no se encargan de conseguirlas, ahí es donde entra el Contenedor de Spring, que en nuestras aplicaciones de Spring será el encargado de conseguir e inyectar las dependencias a los objetos.

B.2 Front-end

B.2.1 MVVM

Este patrón de diseño se caracteriza por su utilidad para pasar datos desde el código javascript -modelo- al código html -vista-, de una manera sencilla, sin tener que tratar directamente con las etiquetas html desde javascript.

La vista estaría formada por el código HTML y las directivas de vue que se encargan de detectar los cambios sobre la vista, y la vista-modelo sería el código javascript que se encarga de transformar los datos acorde a los cambios que recibe de la vista y enviarle estos datos a la vista, o efectuar llamadas al lado servidor.

B.2.2 Promise y Callback

Un callback es una función que se ejecutará una vez termine una llamada asíncrona. Mientras tanto, JavaScript continúa su ejecución normal. Una vez acabó la llamada asíncrona se volverá al punto en el que está el callback y se procederá a su ejecución. Un buen ejemplo del patrón callback es el uso de EventListeners en Javascript, que nos permite definir el listeners para una vista que llamarán a un método cuando una acción concreta ocurra, esto puede ser un click en alguna parte concreta de la vista, que pase un tiempo concreto, etc.

Una promesa es un objeto que envuelve una operación asíncrona y notifica cuándo se hace. Es muy similar a los callbacks, pero las diferencias importantes están en el uso de Promesas. En lugar de proporcionar un callback, una promesa tiene sus propios métodos que llamamos para decirle a la promesa lo que sucederá cuando tenga éxito o cuando falle. Los métodos que proporciona una promesa se invocan con .then() cuando la promesa devuelve algo correcto y ".catch()" para cuando algo salió mal.

Apéndice C

Manual de instalación

Para poder utilizar la aplicación en local es necesario tener la siguiente configuración:

- Eclipse Neon instalado y desde el que se lanzará el servidor de la aplicación
- Java versión 1.8
- El gestor de bases de datos de PostgreSQL
- Apache Maven 3.5.4
- Node.js versión 6.10.3

Una vez tengamos instalado y configurado todo lo que se indica en la lista, tenemos que seguir los siguientes pasos para lanzar la aplicación:

- Crear una base de datos llamada tfgdb con la extensión postgis, un usuario llamado tfg con la contraseña tfg y que pueda acceder a la base de datos creada. Al lanzar la aplicación todas las tablas de entidades se crean debido a que se lo indicamos así a postgres en el fichero application.yml con la opción hibernate.ddl-auto: create, aunque es recomendable cambiarlo a validate una vez lancemos la aplicación por primera vez, ya que así no borraría la base de datos cada vez que lancemos la aplicación. En DatabaseLoader.java se encuentran los datos iniciales que cargará la aplicación.
- Lanzar servidor. Iniciamos eclipse e importamos el proyecto servidor-tfg. Una vez importado creamos una configuración de maven que tenga por meta spring-boot:run y la ejecutamos.
- Lanzar el cliente. Abrimos un terminal y vamos a cliente-tfg, y aquí ejecutamos npm install y a continuación npm run debug.

- Abrir en un navegador http://localhost:1234 y apareceremos como usuario anónimo en el menú Home. En un primer momento no saldrán indicadores ya que no tienen las tablas de datos geográficos creadas. Para ello nos tenemos que logear como administrador, pudiendo usar los pares usuario/contraseña manuel/manuel, maria/maria o patricia/patricia, e ir a la gestión de tablas y municipios.zip poniéndole de nombre municipios a la tabla de base de datos.

Apéndice D

Mockups

Se incluyen todos los mockups que se dibujaron en las primeras etapas del proyecto antes de empezar con el desarrollo. Al tratarse de una versión preliminar de las pantallas, existen bastantes cambios respecto a las versiones finales de estas.

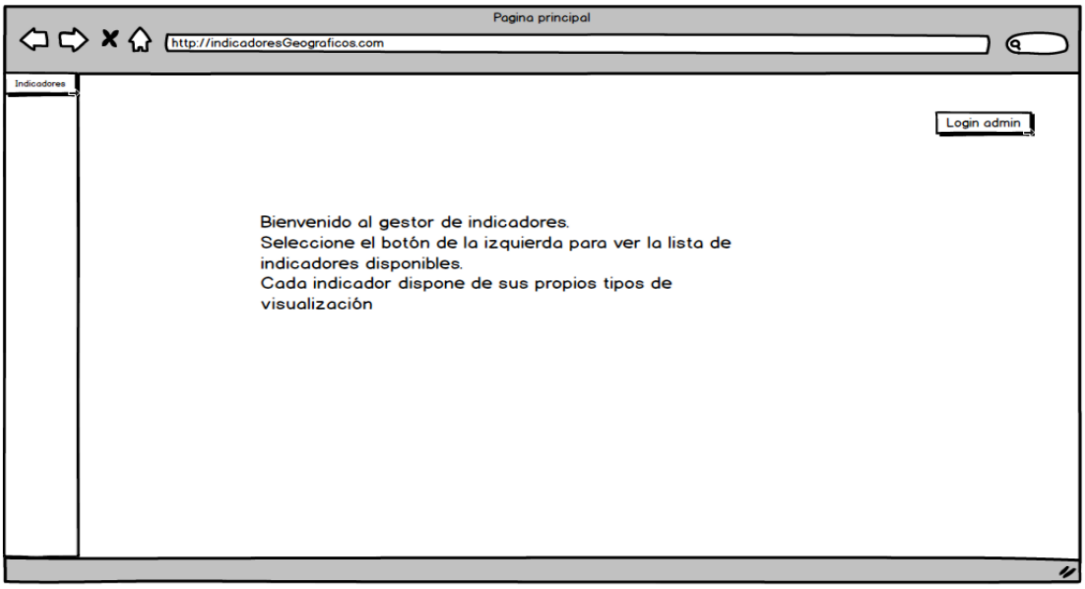


Figura D.1: Pantalla de Home

Figura D.2: Pantalla con el listado de grupos.

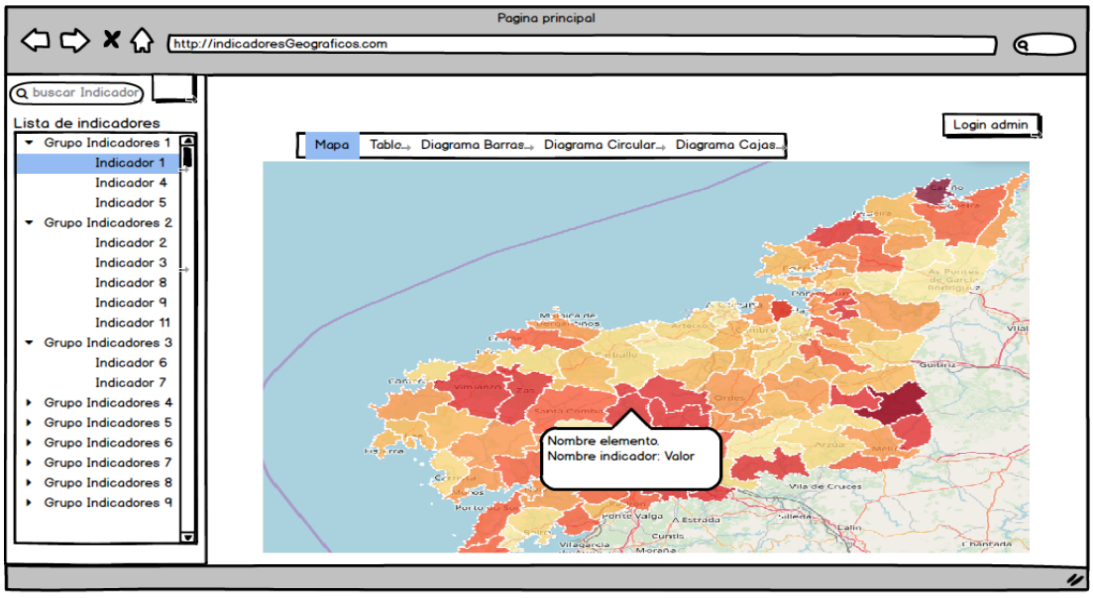


Figura D.3: Pantalla con vista de mapa.

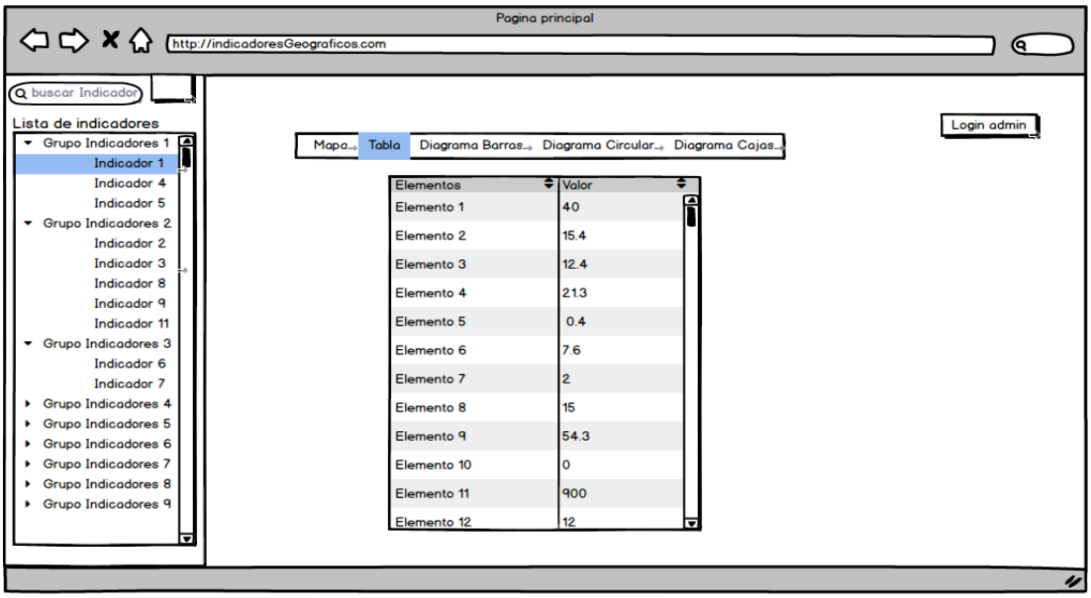


Figura D.4: Pantalla con vista de tabla.

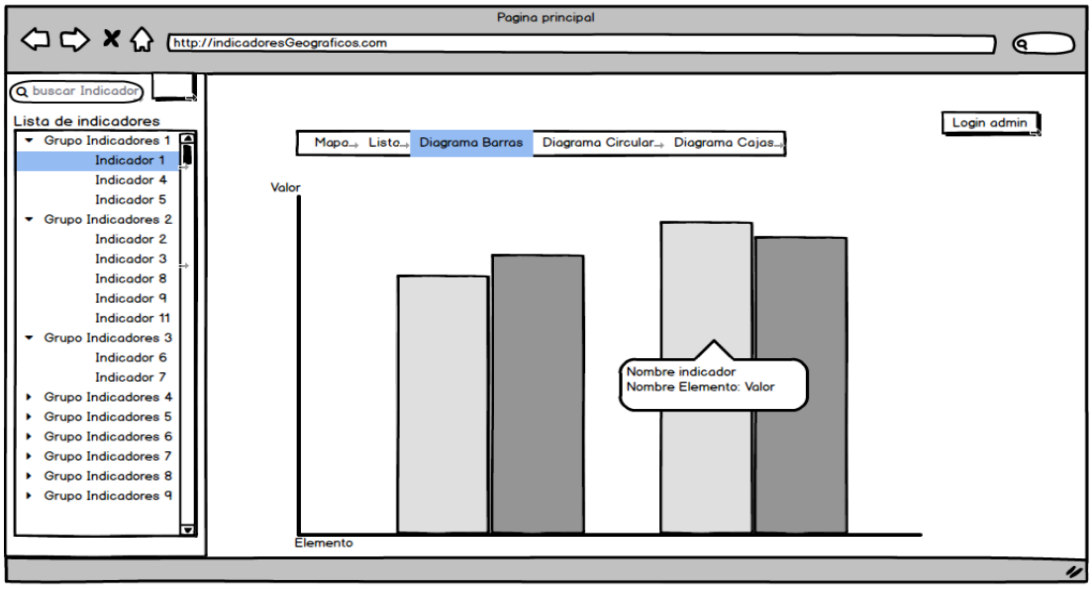


Figura D.5: Pantalla con vista de barras.

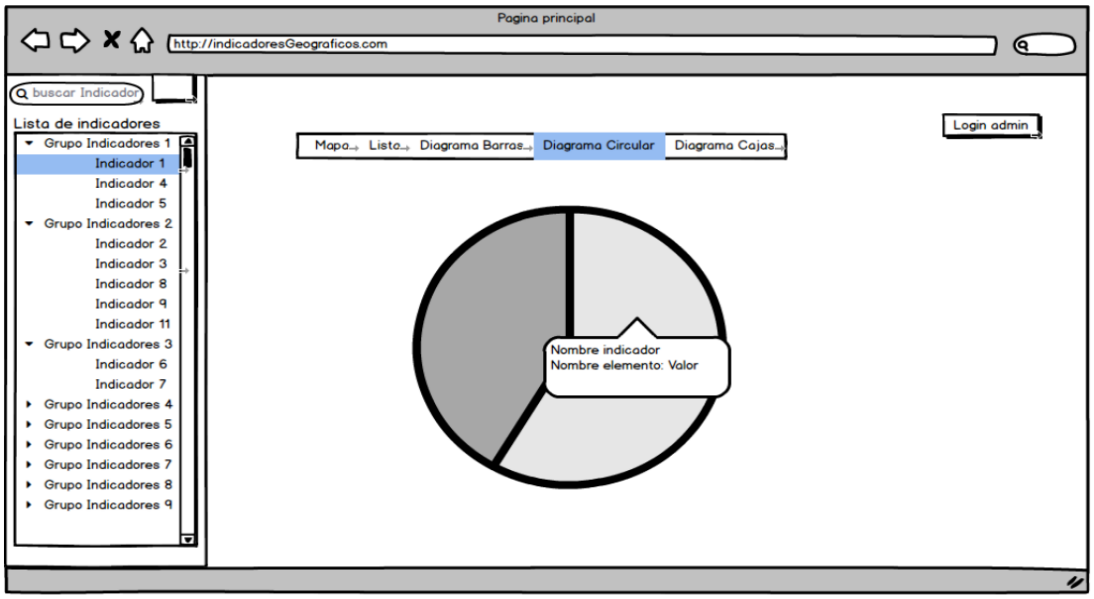


Figura D.6: Pantalla con vista de diagrama circular.

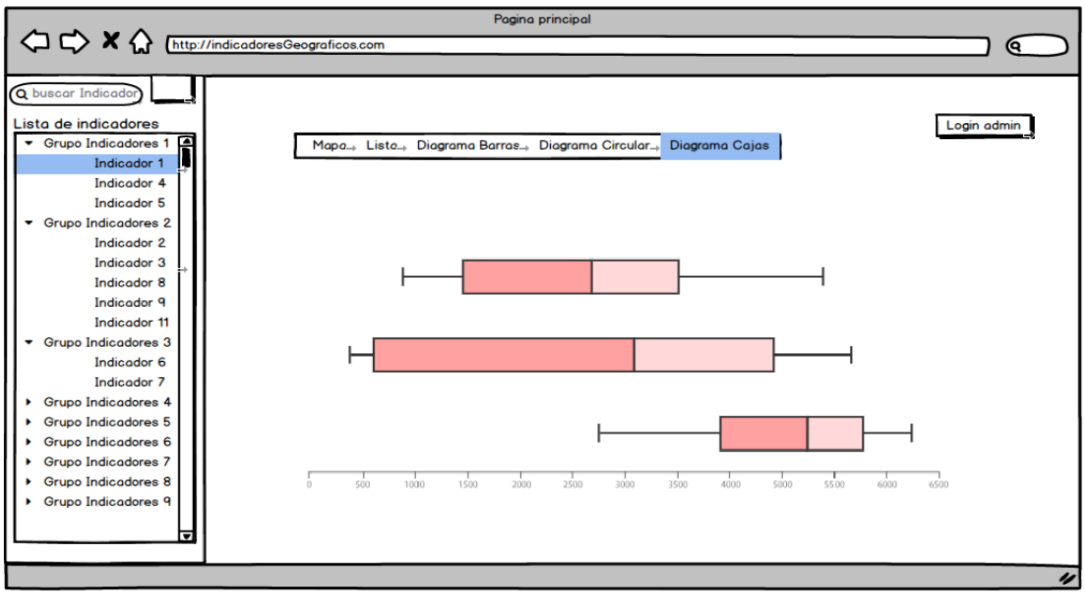


Figura D.7: Pantalla con vista de cajas.

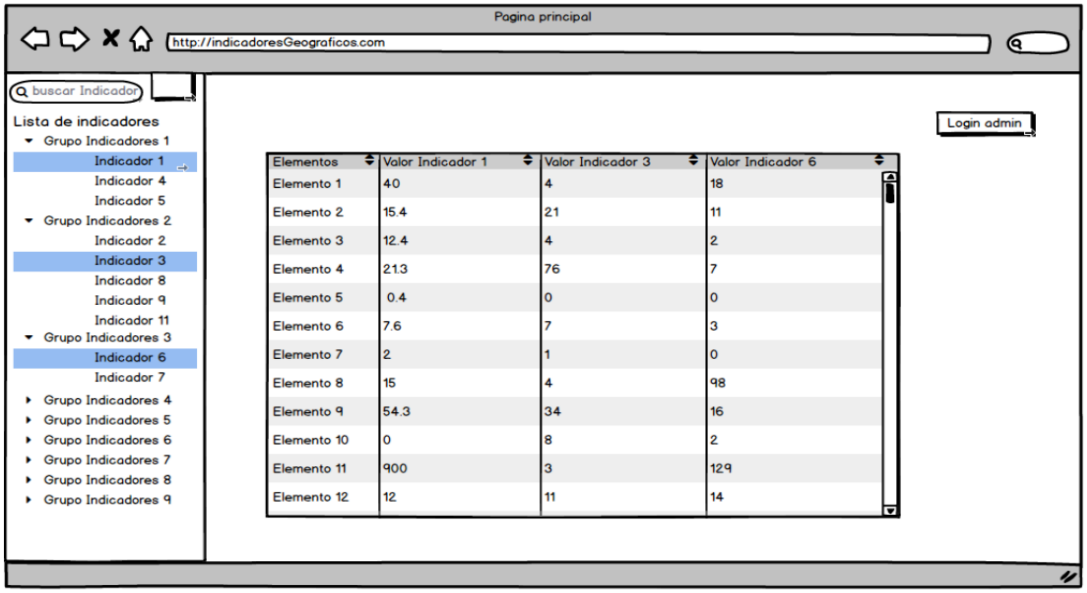


Figura D.8: Pantalla para comparar indicadores.

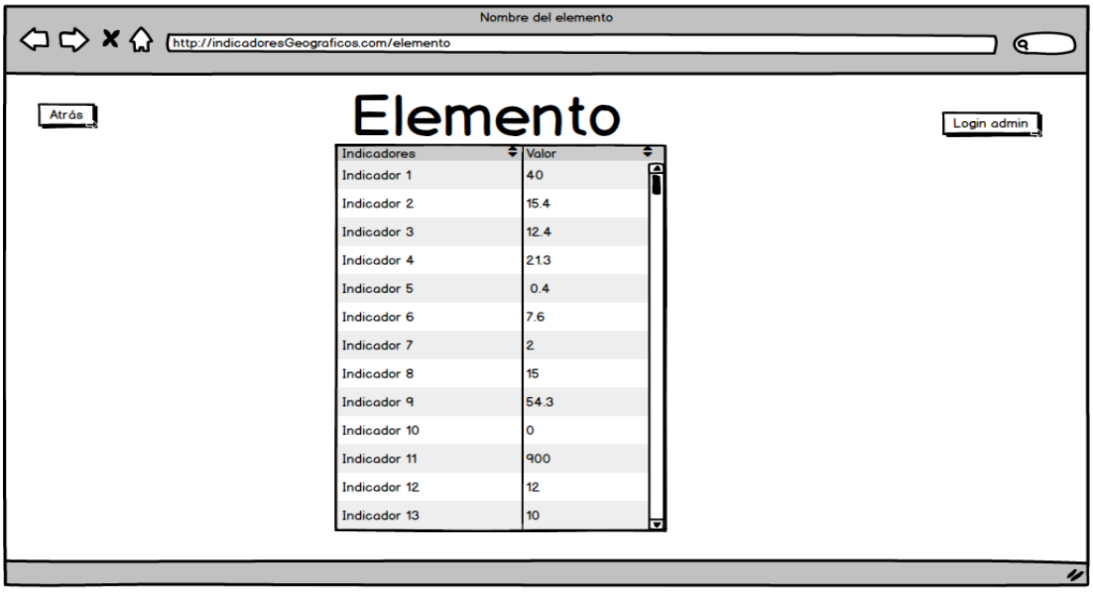


Figura D.9: Pantalla con los indicadores de un lugar.

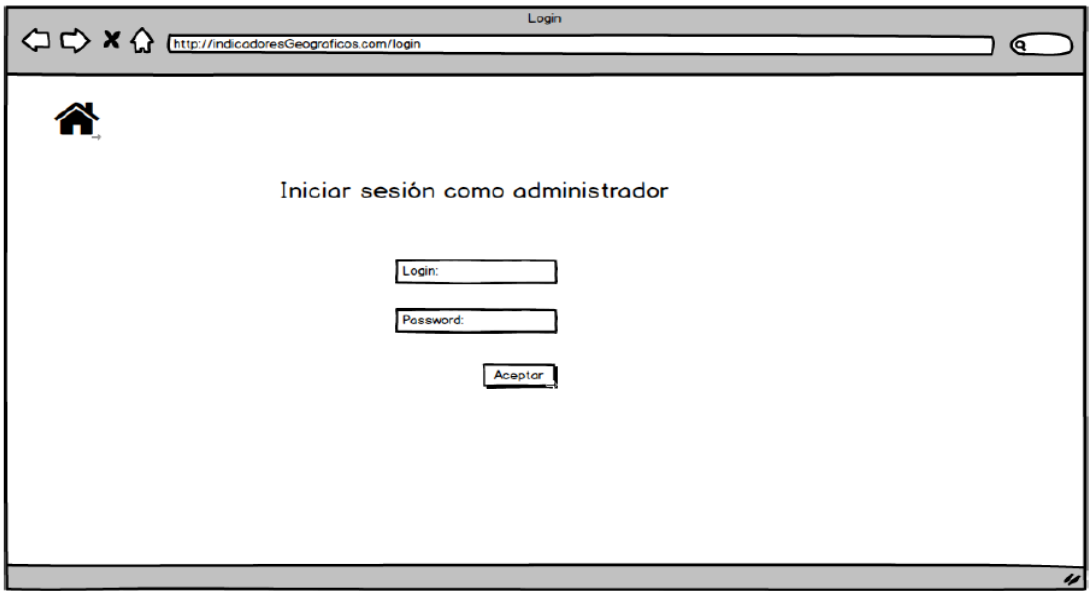


Figura D.10: Pantalla de login.

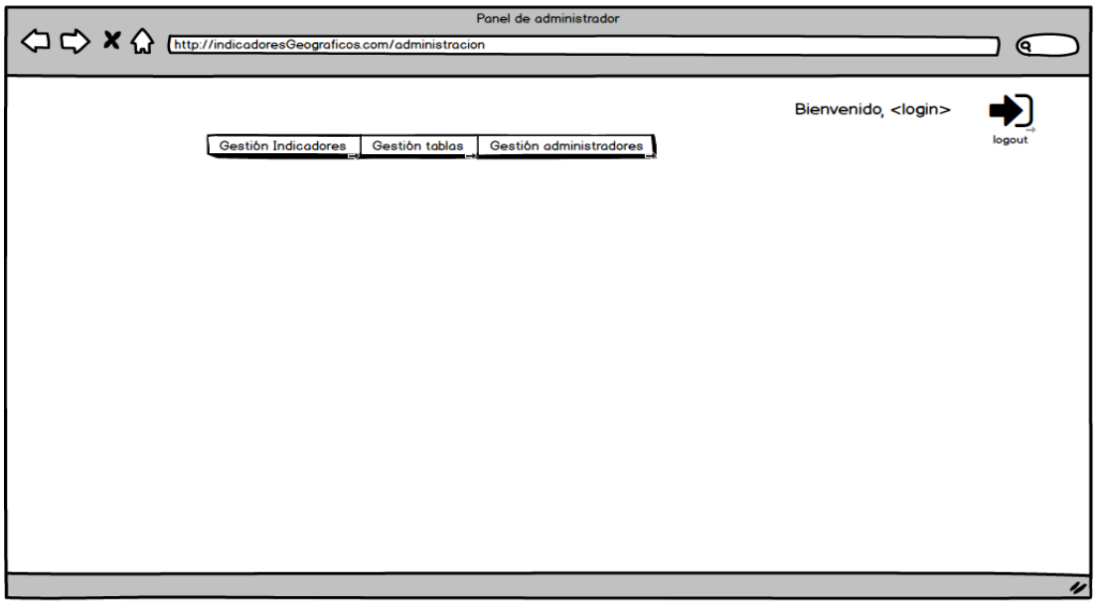


Figura D.11: Pantalla de Home para administradores.

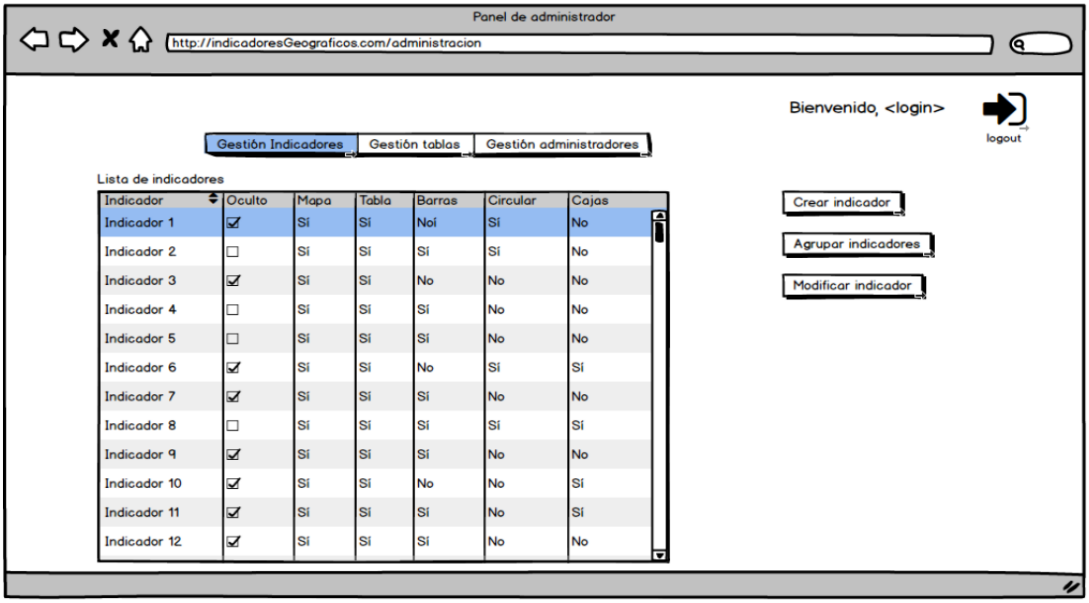


Figura D.12: Pantalla con el listado de indicadores.

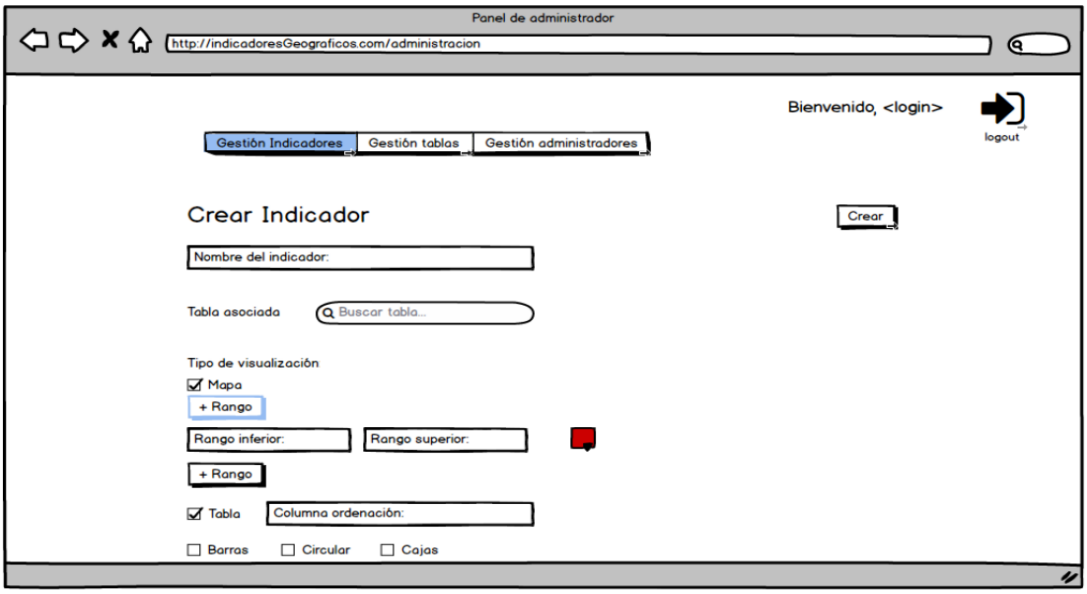


Figura D.13: Pantalla de creación de indicadores.

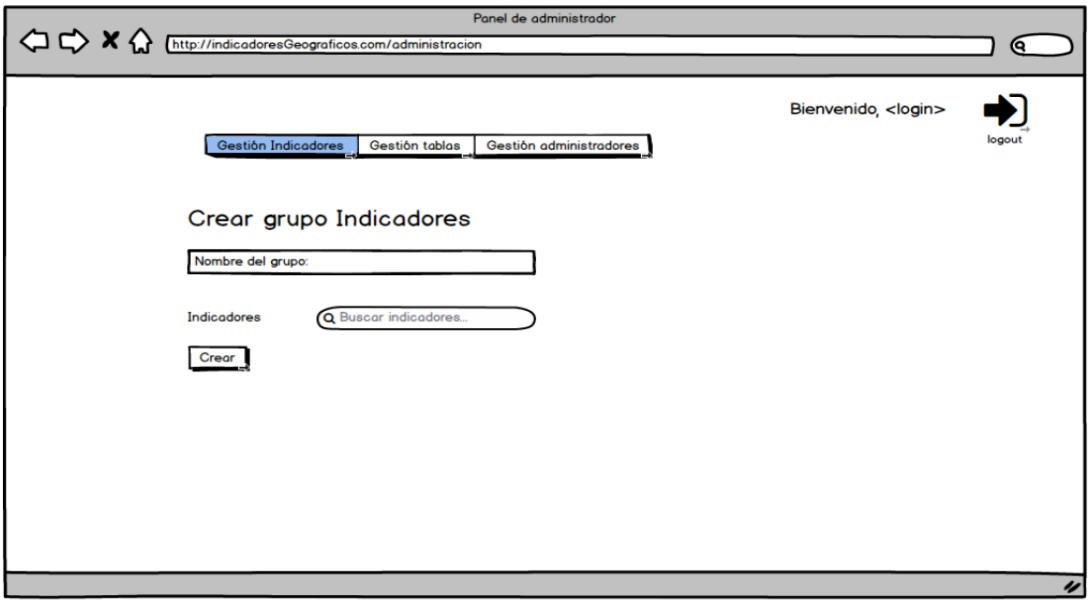


Figura D.14: Pantalla para crear grupo de indicadores.

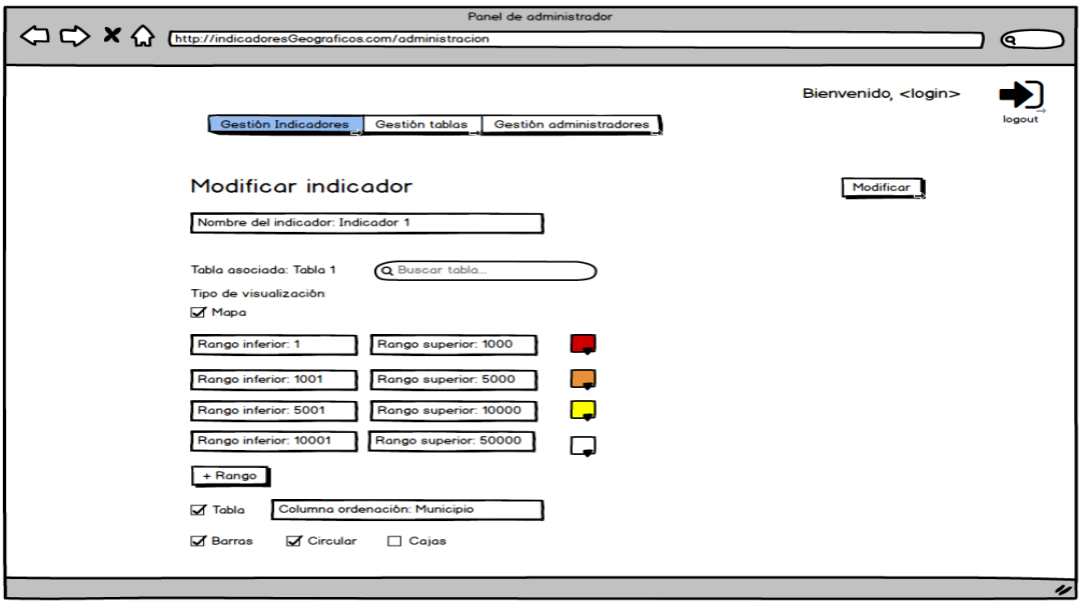


Figura D.15: Pantalla para actualizar un indicador.

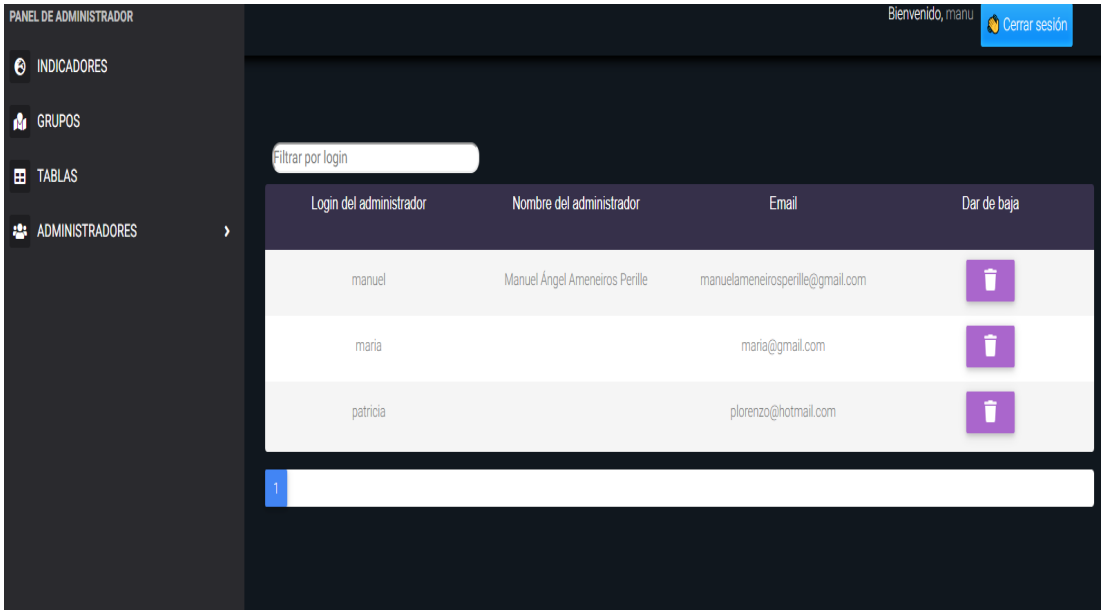


Figura D.16: Pantalla con el listado de administradores.

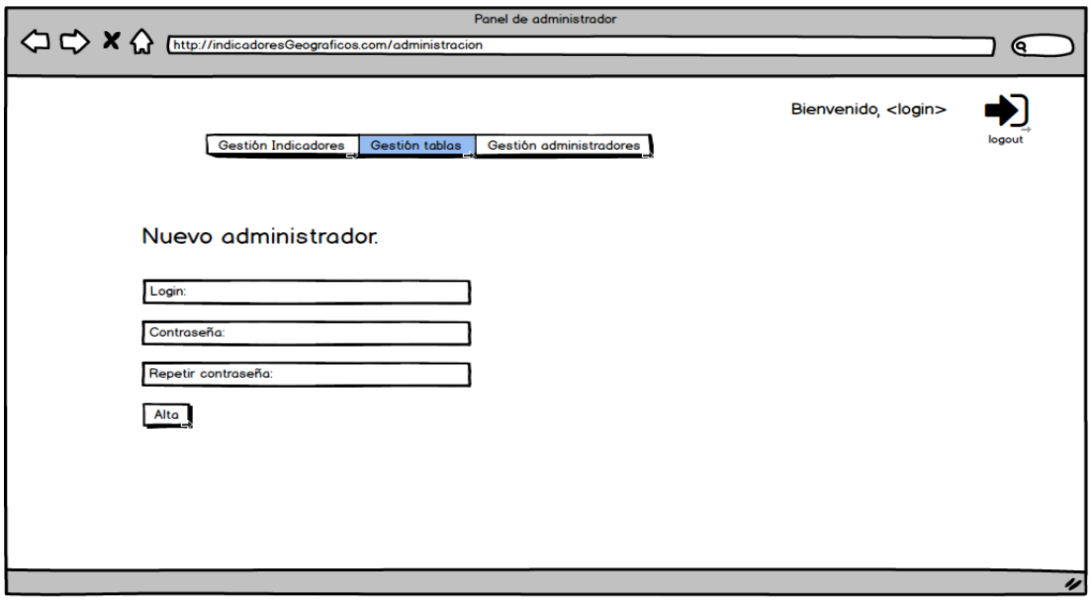


Figura D.17: Pantalla para dar de alta un administrador.

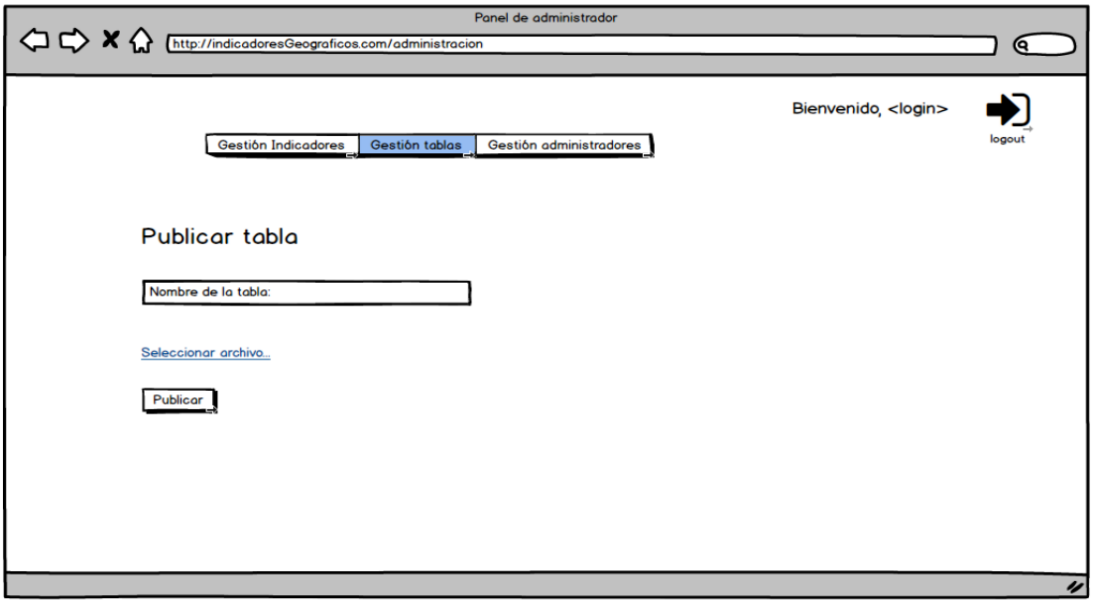


Figura D.18: Pantalla para publicar una tabla.

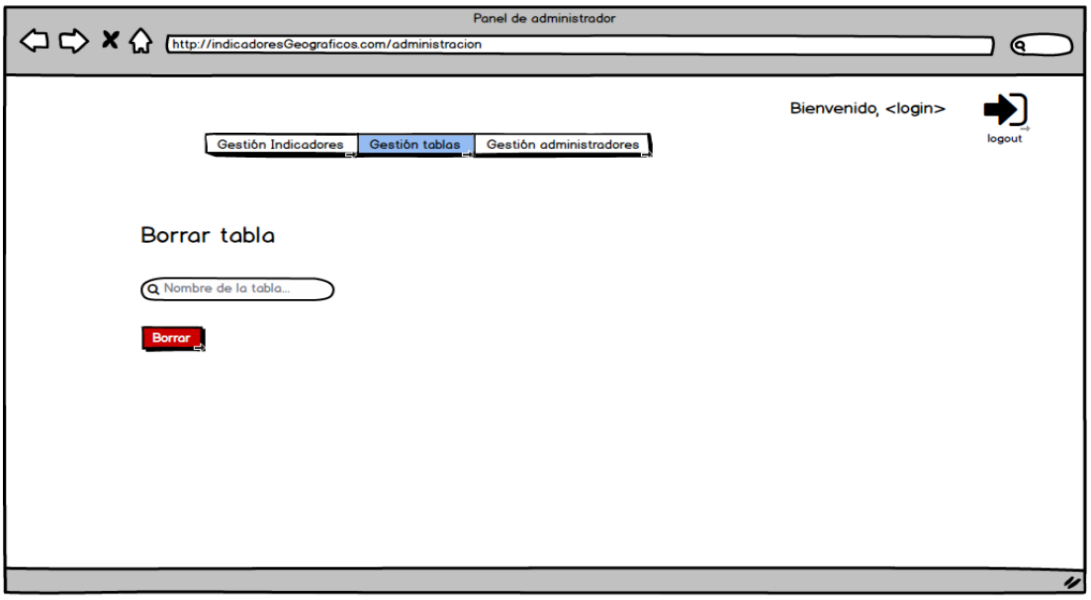


Figura D.19: Pantalla para borrar una tabla.

Bibliografía

[1] "Patrón DAO," 2019, (14 de noviembre de 2019). [Online]. Available: https://www.oscarblancarteblog.com/2018/12/10/data-access-object-dao-pattern/

[2] "Página web de Carto," 2019, (14 de noviembre de 2019). [Online]. Available: https://carto.com/

[3] "Página web de Mapbox," 2019, (14 de noviembre de 2019). [Online]. Available: https://www.mapbox.com/about/

[4] "Página web de Arcgis," 2019, (14 de noviembre de 2019). [Online]. Available: https://www.arcgis.com/home/index.html

[5] "Página web de PostgreSQL," 2019, (14 de noviembre de 2019). [Online]. Available: https://www.postgresql.org/

[6] "Página web de Postgis," 2019, (14 de noviembre de 2019). [Online]. Available: https://postgis.net/

[7] "Página web de Java," 2019, (14 de noviembre de 2019). [Online]. Available: https://www.java.com/es/

[8] "Página web de Hibernate," 2019, (14 de noviembre de 2019). [Online]. Available: https://hibernate.org/

[9] "Página web de Geotools," 2019, (14 de noviembre de 2019). [Online]. Available: https://docs.geotools.org/

[10] "Página web de Spring," 2019, (14 de noviembre de 2019). [Online]. Available: https://spring.io/

[11] "Página web de Javascript," 2019, (14 de noviembre de 2019). [Online]. Available: https://www.javascript.com/

[12] Ryan Dahl, "Página web de Node," 2019, (14 de noviembre de 2019). [Online]. Available: https://nodejs.org/es/

[13] Evan You, "Página web de Vue," 2019, (14 de noviembre de 2019). [Online]. Available: https://vuejs.org/

[14] "Página web de Bootstrap," 2019, (14 de noviembre de 2019). [Online]. Available: https://getbootstrap.com/docs/4.3/getting-started/introduction/

[15] Vladimir Agafonkin, "Página web de Leaflet," 2019, (14 de noviembre de 2019). [Online]. Available: https://leafletjs.com/

[16] J.J. Sutherland, Jeff Sutherland, "Scrum: The Art of Doing Twice the Work in Half the Time," 2014, (14 de noviembre de 2019).

[17] "Página web de LaTex," 2019, (14 de noviembre de 2019). [Online]. Available: https://www.latex-project.org/

[18] "Página web de Eclipse," 2019, (14 de noviembre de 2019). [Online]. Available: https://www.eclipse.org/

[19] "Página web de Sublime," 2019, (14 de noviembre de 2019). [Online]. Available: https://www.sublimetext.com/

[20] "Página web de MagicDraw," 2019, (14 de noviembre de 2019). [Online]. Available: https://www.nomagic.com/products/magicdraw

[21] "Página web de Dia," 2019, (14 de noviembre de 2019). [Online]. Available: http://dia-installer.de/

[22] "Página web de Trello," 2019, (14 de noviembre de 2019). [Online]. Available: https://trello.com/

[23] "Página web de Google Docs," 2019, (14 de noviembre de 2019). [Online]. Available: https://www.google.es/intl/es/docs/about/

[24] "Web de postman," 2019, (14 de noviembre de 2019). [Online]. Available: https://www.getpostman.com/

[25] "Página web de Gitlab," 2019, (14 de noviembre de 2019). [Online]. Available: https://about.gitlab.com/

[26] "Clase zipFile," 2019, (14 de noviembre de 2019). [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/util/zip/ZipFile.html

[27] “Repositorio de topojson,” 2019, (14 de noviembre de 2019). [Online]. Available: https://github.com/topojson/topojson