



INSTITUTO DE EDUCACIÓN SECUNDARIA SERPIS

Gestor de de datos multiplataforma

**Ciclo Formativo Desarrollo de Aplicaciones
Multiplataforma**

Departamento de informática

Autor: Sanz López, Enrique

Tutor: Saura Lloria, Maria

Curso: 2024/2025

Modalidad: Semipresencial



Gestor de de datos multiplataforma

Dedicado a mi familia, que siempre me ha ayudado a seguir adelante.

Resumen

En este trabajo se busca crear una aplicación multiplataforma que sirva para centralizar la gestión de diferentes tipos de bases de datos. En la actualidad existe muchos gestores de bases de datos que se encargan de realizar esta clase de trabajos para tipos de bases específicas, como pueda ser phpMyAdmin para MySQL o pgAdmin para PostgreSQL, sin embargo, hay muy pocos gestores multiplataforma que sirvan para trabajar con múltiples tipos de bases de datos y a la hora de emplear nuevos tipos de bases de datos, requieren hacer instalaciones manuales de drivers para las nuevas conexiones que no siempre son sencillas ya que sus métodos de instalación cambian dependiendo del sistema operativo y características del dispositivo. Es por esto que mediante la tecnología Docker se busca crear un gestor de bases de datos que sea multiplataforma y de rápida instalación en el que en caso de necesitarse nuevos drivers, permita una instalación común en los sistemas operativos en los que opere. Para lograrlo, se ha implementado una aplicación multiplataforma mediante los frameworks Django, y Vue.js, que emplea Python, JavaScript, CSS, y HTML capaz de realizar los procesos requeridos en los gestores de bases de datos y con una gran capacidad para ser escalado a futuro. Por último cabe destacar, que la filosofía de desarrollo de la aplicación ha buscado en todo momento dar una gestión lo más gráfica posible al usuario para facilitar su uso en el entorno de trabajo, y otorgar al usuario la capacidad de modificar la apariencia de la aplicación para que la pueda ajustar en la medida de lo posible a sus preferencias.

Resum

En aquest treball es busca crear una aplicació multiplataforma que serveixi per centralitzar la gestió de diferents tipus de bases de dades. En l'actualitat hi ha molts gestors de bases de dades que s'encarreguen de realitzar aquesta classe de treballs per a tipus de bases específiques, com pugui ser phpMyAdmin per a MySQL o pgAdmin per a PostgreSQL, no obstant això, hi ha molt pocs gestors multiplataforma que serveixin per treballar amb múltiples tipus de bases de dades i a l'hora d'emprar nous tipus de bases de dades amb execucions manuals els mètodes d'instal·lació canvien segons el sistema operatiu i les característiques del dispositiu. És per això que mitjançant la tecnologia Docker es busca crear un gestor de bases de dades que sigui multiplataforma i de ràpida instal·lació on en cas de necessitar-se nous drivers, permeti una instal·lació comuna en els sistemes operatius en què operi. Per aconseguir-ho, s'ha implementat una aplicació multiplataforma mitjançant els frameworks Django, i Vue.js, que utilitza Python, JavaScript, CSS, i HTML capaç de realitzar els processos requerits als gestors de bases de dades i amb una gran capacitat per ser escalat a futur. Finalment, cal destacar que la filosofia de desenvolupament de l'aplicació ha buscat en tot moment donar una gestió tan gràfica com sigui possible a l'usuari per facilitar-ne l'ús en l'entorn de treball, i atorgar a l'usuari la capacitat de modificar l'aparença de l'aplicació perquè la pugui ajustar en la mesura que sigui possible a les seves preferències.

Abstract

This work aims to create a cross-platform application that centralizes the management of different types of databases. Currently, there are many database managers that handle this type of work for specific database types, such as phpMyAdmin for MySQL or pgAdmin for PostgreSQL. However, there are very few cross-platform managers that can work with multiple types of databases, and when using new types of databases, they require manual driver installations for new connections, which are not always simple since their installation methods vary depending on the operating system and device characteristics. For this reason, using Docker technology, the project aims to create a cross-platform and quick-to-install database manager that, if new drivers are needed, allows for a common installation across the operating systems on which it operates. To achieve this, a cross-platform application was implemented using the Django and Vue.js frameworks, employing Python, JavaScript, CSS, and HTML, capable of performing the processes required by database managers and with great capacity for future scalability. Finally, it should be noted that the application's development philosophy has always sought to provide the most graphical user experience possible to facilitate its use in the work environment, and to give users the ability to modify the application's appearance to best suit their preferences.

Índice

1.	Justificación.....	1
2.	Gestión del proyecto.....	1
3.	Herramientas utilizadas.....	2
4.	Descripción del proyecto.....	4
4.1.	Análisis.....	4
4.1.1.	Requisitos.....	5
4.1.2.	Criterios de aceptación.....	6
4.2.	Diseño.....	6
4.2.1.	Guía de estilos.....	6
4.2.2.	Iconografía y tono.....	7
4.2.3.	Diagrama de entidad relación.....	7
4.2.4.	Casos de uso.....	9
4.2.5.	Site map.....	12
4.3.	Desarrollo.....	13
4.4.	Pruebas.....	16
4.4.1.	Pruebas de usuario.....	16
4.4.2.	Pruebas de personalización.....	20
4.4.3.	Pruebas de gestión de bases de datos.....	22
4.4.4.	Pruebas de control de accesos.....	28
4.5.	Documentación.....	29
5.	Trabajos futuros.....	30
6.	Conclusiones.....	30
7.	Bibliografía y Webgrafía.....	31
8.	Glosario.....	32
9.	Anexos.....	34
	Anexo I: Diagrama de casos de uso.....	34
	Anexo II: Maquetación del proyecto.....	36

Índice de imágenes

Figura 1: Ejemplo de modelo organizativo Kanban.....	2
Figura 2: Figura 2: logotipos de las tecnologías empleadas en el desarrollo del proyecto.....	2
Figura 3: Logotipo de la aplicación.....	7
Figura 4: Esquema de la base de datos.....	8
Figura 5: Diagrama de casos de uso para la gestión de sesiones.....	10
Figura 6: Diagrama de casos de uso para la personalización de la aplicación.....	10
Figura 7: Diagrama de casos de uso para la gestión de usuarios, grupos y permisos.....	11
Figura 8: Diagrama de métodos de la pantalla principal.....	12
Figura 9: Site map de la aplicación.....	13
Figura 10: Diagrama de la distribución de componentes Vue.....	15
Figura 11: Pantalla de inicio de sesión de la aplicación.....	17
Figura 12: Pantalla principal de la aplicación.....	17
Figura 13: Pantalla de gestión de usuarios, grupos, y permisos.....	18
Figura 14: Formulario de creación de usuario.....	18
Figura 15: Pantalla de gestión de usuarios con el nuevo usuario generado.....	19
Figura 16: Modal de edición de usuario.....	19
Figura 17: Pantalla de gestión de usuarios con usuario editado.....	19
Figura 18: Modal de eliminación de usuario.....	20
Figura 19: Pantalla de personalización.....	21
Figura 20: Pantalla de personalización con los cambios de CSS.....	21
Figura 21: Pantalla principal con cambios en el CSS.....	22
Figura 22: Pantalla principal de la aplicación.....	23
Figura 23: Modal de generación de conexión.....	23
Figura 24: Pantalla principal con varias conexiones generadas.....	24
Figura 25: Menú contextual de acciones de tabla.....	24
Figura 26: Pantalla principal con acción de búsqueda.....	25
Figura 27: Pantalla principal con una búsqueda filtrada.....	25
Figura 28: Ejemplo de JSON descargado de los resultados de una búsqueda.....	26
Figura 29: Ejemplo de CSV descargado de los resultados de una búsqueda.....	26
Figura 30: Pantalla principal con la consola abierta.....	27
Figura 31: Pantalla principal tras la ejecución de un comando de consola.....	27
Figura 32: Menú contextual de conexión.....	28
Figura 33: Pantalla principal tras la eliminación de una conexión.....	28
Figura 34: Diagrama de casos de uso.....	35
Figura 35: Maquetación original de la pantalla de login.....	36
Figura 36: Maquetación original de la pantalla principal.....	36

Figura 37: Maquetación original de la pantalla de gestión de usuarios.....	37
Figura 38: Maquetación original de la pantalla de personalización.....	37

1. Justificación

En la actualidad en la actividad empresarial de muchas empresas informáticas, se requiere el uso de múltiples bases de datos para los diferentes proyectos que desarrollan. Esto típicamente conlleva la necesidad de instalar múltiples gestores de bases de datos con las dificultades que esto conlleva, como por ejemplo cumplir requisitos de instalación que en ocasiones pueden llevar a limitar los sistemas operativos disponibles. Todo esto ralentiza y encarece el desarrollo de proyectos, genera duplicidades y problemas de compatibilidad, y complica el mantenimiento de la información en las empresas.

En los últimos años, el desarrollo de frameworks multiplataforma y de Docker, ha facilitado el desarrollo de aplicaciones multiplataforma que regularmente conectan con bases de datos, y como tal, están preparados para emplear varios tipos de bases de datos, son flexibles, y escalables lo cual puede ser empleado para realizar operaciones de gestión en bases de datos de forma gráfica.

De los frameworks más modernos que se utilizan a día de hoy, Django es un framework de Python, un lenguaje moderno, con una gran comunidad que lo respalda, bien documentado, de rápido desarrollo, escalable, extensible, seguro, y que presenta ya un soporte para múltiples tipos de bases de datos. Siendo un framework muy extendido.

Es por este motivo por el que se ha determinado que existe la necesidad de la generación de un gestor de bases de datos multiplataforma universal que otorgue un entorno gráfico con el que poder trabajar diferentes tipos de bases de datos. Con lo que se espera reducir la dificultad del mantenimiento de las bases de datos y la cantidad de datos duplicados, así como acelerar el desarrollo de proyectos en las empresas utilizando Django y Docker como tecnologías principales en su desarrollo.

2. Gestión del proyecto

En el desarrollo de este proyecto se ha empleado la metodología Kanban por tratarse de una metodología de trabajo ágil y flexible para la organización de proyectos, lo cual encaja con las necesidades de generar una aplicación flexible y escalable.

Dicha metodología se basa en el control del flujo de trabajo mediante la división de los proyectos en tareas más pequeñas y fáciles de controlar, de las que se va controlando su progreso para controlar el desarrollo del proyecto.

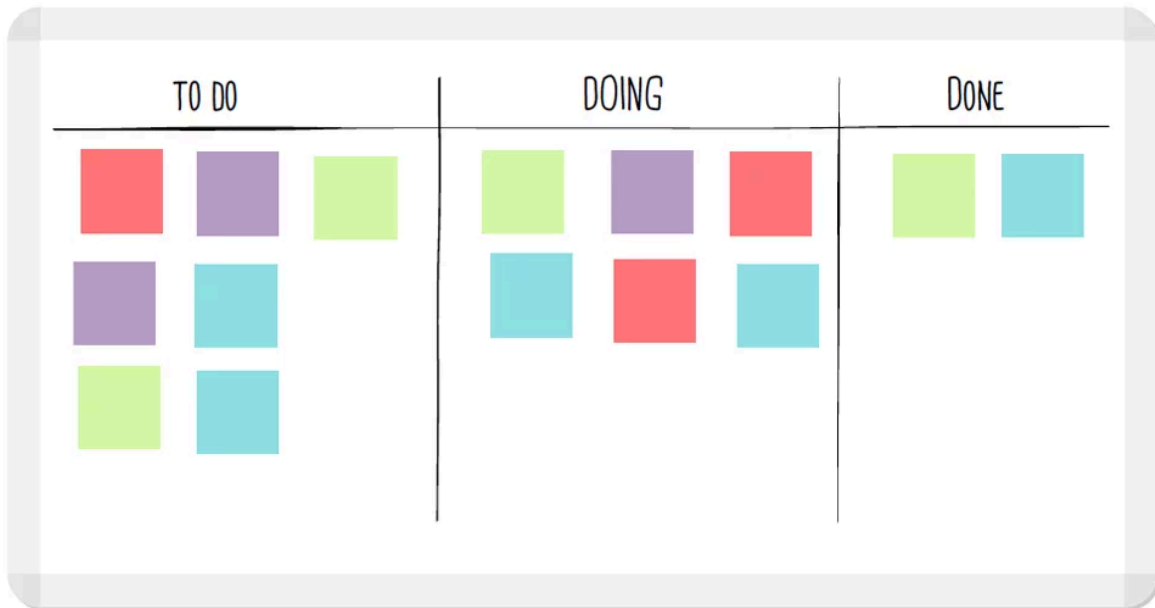


Figura 1: Ejemplo de modelo organizativo Kanban obtenido de <https://smartway.es/guia-kanban/>

3. Herramientas utilizadas



Figura 2: logotipos de las tecnologías empleadas en el desarrollo del proyecto obtenidos de:
<https://upload.wikimedia.org/wikipedia/commons/thumb/c/c3/Python-logo-notext.svg/1200px-Python-logo-notext.svg.png>
<https://static.djangoproject.com/img/logos/django-logo-positive.png>
https://upload.wikimedia.org/wikipedia/commons/thumb/9/99/Unofficial_JavaScript_logo_2.svg/640px-Unofficial_JavaScript_logo_2.svg.png
https://upload.wikimedia.org/wikipedia/commons/thumb/9/95/Vue.js_Logo_2.svg/1200px-Vue.js_Logo_2.svg.png
https://images.seeklogo.com/logo-png/38/2/bootstrap-5-logo-png_seeklogo-386607.png
https://www.docker.com/app/uploads/2023/05/symbol_blue-docker-logo.png
https://upload.wikimedia.org/wikipedia/commons/thumb/2/29/Postgresql_elephant.svg/1200px-Postgresql_elephant.svg.png

https://upload.wikimedia.org/wikipedia/commons/thumb/d/d5/CSS3_logo_and_wordmark.svg/1452px-CSS3_logo_and_wordmark.svg.png
<https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRmLME0hpAJOqBGhaVjcgkk8hIKS3S4GAqrLq&s>
<https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSjoAY1yvW2TRRXFVU1Hcf0h6MfFH0AXDS2Jq&s>

Para el desarrollo de este proyecto se han empleado las siguientes tecnologías:

Como lenguaje de programación principal se ha empleado Python en su versión 3.13.2 junto al framework Django en su versión 5.2 para el control del backend de la aplicación, el renderizado de los componentes estáticos de las pantallas, y la gestión de las conexiones.

Para la generación de componentes dinámicos se ha hecho uso de JavaScript con el framework Vue.js en su versión 3 dado que permite el renderizado dinámico de componentes, es fácil de mantener y es compatible como framework de frontend con la mayoría de frameworks más seguros via CDN.

Para la gestión de peticiones desde el frontend se ha empleado JQuery versión 3.7.1 mediante CDN por ser compatible con JavaScript y Vue.js, tener un uso bastante intuitivo, y permitir la inclusión de las cabeceras necesarias para realizar peticiones seguras en Django.

Adicionalmente para el entorno de frontend se ha empleado Bootstrap en su versión 5.3, por la librería de estilos para objetos HTML que se han empleado en el entorno gráfico.

Como librería de iconos se ha empleado FontAwesome en su versión 6.7.2 por su facilidad de uso, su compatibilidad con el lenguaje HTML, y el hecho de que posee una gran variedad de iconos de uso gratuito.

Como gestor de paquetes se ha hecho uso de pip por tratarse de un gestor tradicional de Python que es compatible con la mayoría de sus librerías y muy estable.

Para facilitar su portabilidad se ha hecho uso de Docker en su versión 27.2.0 que permite hacer pequeñas virtualizaciones de diferentes aplicaciones siendo rápidas, eficientes y exportables a diferentes sistemas operativos.

Como base de datos se ha empleado PostgreSQL versión 17.4 dado que es un sistema de bases de datos gratuito que puede ser adaptado a las necesidades del proyecto en caso de necesidad, presenta un gran rendimiento, está preparado para integrarse con Python, y al ser un tipo de SQL garantiza la integridad de datos.

Como gestor de la base de datos durante el desarrollo se ha empleado pgAdmin versión 8.14, por la facilidad que aporta en el desarrollo de la aplicación poder ver de forma gráfica el estado de su base de datos y ser compatible con PostgreSQL.

Como sistema de control de versiones se ha empleado Git versión 2.44.0, el cual permite controlar los cambios de código en los archivos, colaborar con otros desarrolladores, e incluso revertir a versiones previas.

4. Descripción del proyecto

El objetivo de este proyecto es desarrollar un gestor de proyectos multiplataforma capaz de gestionar diferentes tipos de bases de datos de forma segura, distinguiendo entre los diferentes tipos de bases de usuarios para garantizar la seguridad de las operaciones.

Para garantizar esta seguridad, la aplicación divide a los usuarios en grupos, a los cuales se les puede asignar y eliminar permisos, de esta forma permitiendo una configuración de la seguridad que tienen los usuarios muy flexible y extensible.

Todos los usuarios identificados tienen la capacidad de realizar operaciones de conexión a base de datos y lectura de registros, pero a partir de este punto, el resto de operaciones que se pueden realizar en el gestor dependen de los permisos del grupo al que pertenezca el usuario.

Los usuarios no identificados solo tienen acceso a la pantalla de inicio de sesión y al método de inicio de sesión.

Dentro de la aplicación, se tiene dos grupos especiales, que son el grupo de administradores, que tiene todos los permisos para realizar operaciones en la aplicación, y el grupo de lectores, que solo puede realizar las operaciones de lectura de datos y conexión a bases de datos.

4.1. Análisis

Para lograr los objetivos especificados para este proyecto, es necesario cumplimentar una serie de requisitos para garantizar que la aplicación pueda ser usada en un entorno de trabajo para la gestión de proyectos. Estos requisitos se han indicado a continuación.

4.1.1. Requisitos

Categoría	Requisito	Descripción
Requisitos funcionales	Autenticación de usuario	La aplicación tiene que tener un inicio de sesión seguro basándose en su nombre de usuario y contraseña.
Requisitos funcionales	Gestión de permisos	La aplicación debe gestionar las opciones del usuario según sus datos de sesión, limitando de esta forma las capacidades que posee en la gestión de las bases de datos y añadiendo una capa de seguridad adicional al proceso de gestión de las mismas.
Requisitos funcionales	Interfaz de usuario	El diseño de la interfaz de usuario debería ser intuitiva y fácil de navegar, con nombres y botones claros para las acciones comunes.
Requisitos funcionales	Gestión de bases de datos	La aplicación debe permitir la gestión de diferentes tipos de bases de datos en sus operaciones de CRUD así como permitir la importación y exportación de datos de datos.
Requisitos no funcionales	Rendimiento	La aplicación deberá cargarse rápidamente, y no presentar retrasos a los eventos de usuario.
Requisitos no funcionales	Seguridad	La aplicación debe ser diseñada teniendo la seguridad en cuenta, incluyendo protección contra ataques por inyección de código, y secuencias de comandos entre sitios.
Requisitos no funcionales	Compatibilidad	La aplicación debe ser compatible con navegadores modernos como Google Chrome, Mozilla Firefox, y Microsoft Edge, así como con los sistemas operativos Windows, Linux, y Mac.

4.1.2 Criterios de aceptación

La aplicación debe ser funcional tanto en los requisitos funcionales como en los no funcionales y haber sido testeada para garantizar el cumplimiento de todos los criterios ya indicados a un nivel en el que pueda ser empleada en un entorno laboral.

4.2. Diseño

Como ya se ha visto, el objetivo del diseño de la aplicación es garantizar la legibilidad de la misma, y que su navegación sea intuitiva. Para garantizar la legibilidad, se ha empleado un tamaño de letra mediano con una fuente estándar, y para hacer la navegación intuitiva se ha optado por hacer uso de modales, para reducir la cantidad de veces que se debe recargar la pantalla. De esta forma, se ha generado una aplicación con pocas pantallas, pero fáciles de mantener y que contienen todos los métodos necesarios para el funcionamiento de la aplicación.

Adicionalmente, dado que esta aplicación puede ser empleada por diferentes empresas, se ha incluido un campo de inclusión de CSS para que los usuarios con los permisos adecuados puedan modificar la apariencia de la aplicación en caso de ser necesario.

4.2.1. Guía de estilos

El objetivo del diseño de la aplicación es transmitir un diseño profesional y controlado, fomentando la sensación de seguridad, y centrándose en la accesibilidad para garantizar la comodidad del usuario. Para lo cual se ha usado colores azules para la navegación y los botones de confirmación y edición, blanco para el texto de la navegación, el texto del footer y el fondo del resto de la página para que destaquen del resto de secciones, negro para el footer, rojo para los botones de eliminar, y gris para los botones de cancelación. Con esto se ha buscado dar una imagen formal, mientras se usa una paleta de colores que puede ser empleada fácilmente en futuras páginas.

La paleta de colores ha sido usada en la aplicación ha sido:

- #0d6efd para los botones de confirmación y edición.
- #DC3545 para los botones de eliminación.
- #fff para el color de texto de botones y navegación.
- #000000 para el texto general de la página.
- #ddd8d8 para el fondo del formulario de login.
- #004e97 para el color de fondo de la navegación.

Por otra parte se ha empleado la fuente Arial Helvetica sans-serif como fuente para los textos de la aplicación por aportar un tono profesional y tener una fácil lectura para el usuario promedio.

4.2.2. Iconografía y tono

Dado que el objetivo de la aplicación es facilitar la gestión de diferentes bases de datos en el entorno laboral, en toda la aplicación se ha optado por un tono directo, claro, neutro y formal, apropiado para un entorno laboral.

Para la iconografía, se ha empleado las librerías de fontawesome versión 6.7.2, por ser una librería de iconos compatible con HTML que ya contiene una enorme cantidad de iconos de significados claros.

Por otra parte, el icono de la aplicación empleado ha sido:

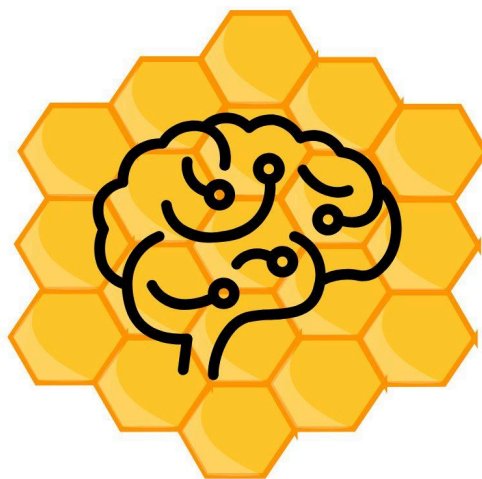


Figura 3: Logotipo de la aplicación

El fondo de panal de abeja busca dar una impresión de flexibilidad por su color mientras una sensación de organización y coordinación. Mientras que por otra parte, el cerebro en el centro busca enlazar lo anterior con la memoria y la gestión de la información.

4.2.3. Diagrama de entidad relación

En el desarrollo de esta aplicación se ha empleado una base de datos relacional PostgreSQL, por su fiabilidad, facilidad de uso y extensión en el mercado laboral.

Las tablas empleadas en la base de datos se han visto dadas por las necesidades de seguridad y personalización de la aplicación, y por las tablas predeterminadas de Django. Dando como resultado el siguiente diagrama:

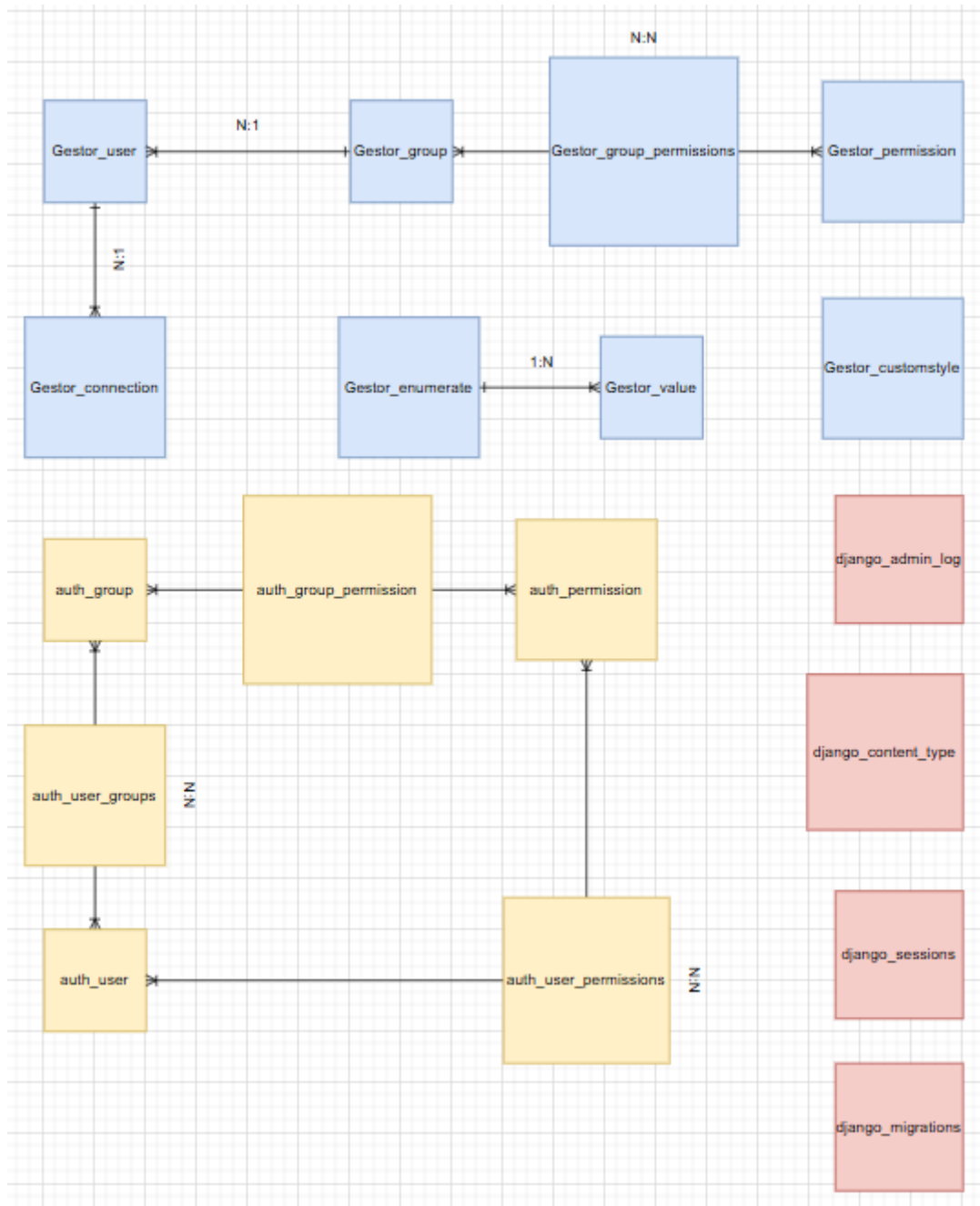


Figura 4: Esquema de la base de datos

En este esquema, se ha dividido por colores las tablas según el esquema al que pertenecen. El esquema azul corresponde a las tablas de la aplicación, que se dedican a la gestión de la seguridad de la aplicación, a guardar valores predeterminados, y a la personalización de su estética. El esquema rojo corresponde a las tablas de Django encargadas del control de las migraciones,

sesiones, el tipo de contenido y logs. Finalmente las tablas en amarillo son las tablas que Django emplea por defecto para la gestión de su seguridad.

Para el caso de esta aplicación, se ha generado una seguridad separada de la de Django debido a que de esta manera, se puede ganar un mayor control sobre los permisos de acceso y ejecución que tienen los usuarios y se puede gestionar mejor los medios por los que se almacena la información de los usuarios. Sin embargo, se ha optado por dejar las tablas de seguridad predeterminada de Django por si en un futuro pudieran ser necesarias para otros accesos en futuras expansiones de la aplicación.

Las tablas `Gestor_user`, `Gestor_group`, `Gestor_group_permissions`, y `Gestor_permission` son tablas encargadas de almacenar la información de los usuarios de la aplicación, y los permisos que tienen para acceder y ejecutar diferentes métodos de la aplicación.

La tabla `Gestor_connection` guarda datos de conexiones para cada usuario, como por ejemplo el host, el puerto y el nombre de la base de datos, de forma que el usuario tenga una mayor facilidad para el acceso y la gestión de las bases de datos.

Las tablas `Gestor_enumerate` y `Gestor_value` sirven para guardar valores de tipo enumerados y existe para facilitar el mantenimiento de este tipo de valores, por ejemplo, aquí se guarda la lista de comparadores para filtros de búsqueda.

La tabla `Gestor_customstyle` sirve para guardar datos relacionados a la estética de la aplicación como pueda ser el icono de la aplicación, o el tamaño de letra. No se relaciona con ninguna otra tabla, por que no contiene datos ni de usuarios, ni de permisos, ni de enumerados, el acceso y gestión de estos datos, en su lugar se realiza por código en la aplicación.

4.2.4. Casos de uso

Para poder realizar un correcto desarrollo de la aplicación ha sido necesario desarrollar un diagrama de casos de uso que explique de una forma esquematizada explique las acciones que el usuario pueda hacer desde la aplicación.

Para esto, ha sido necesario distinguir las capacidades del usuario según si ha iniciado sesión o no, y los permisos que tenga el grupo en el que se ubique. Dado que todo esto combinado con las funciones de gestión de base de datos han dado lugar a un diagrama de casos de uso muy extenso, se ha optado por dividir el diagrama en tramos para visualizarlo mejor aquí. Una versión completa del diagrama se encuentra en el anexo I.

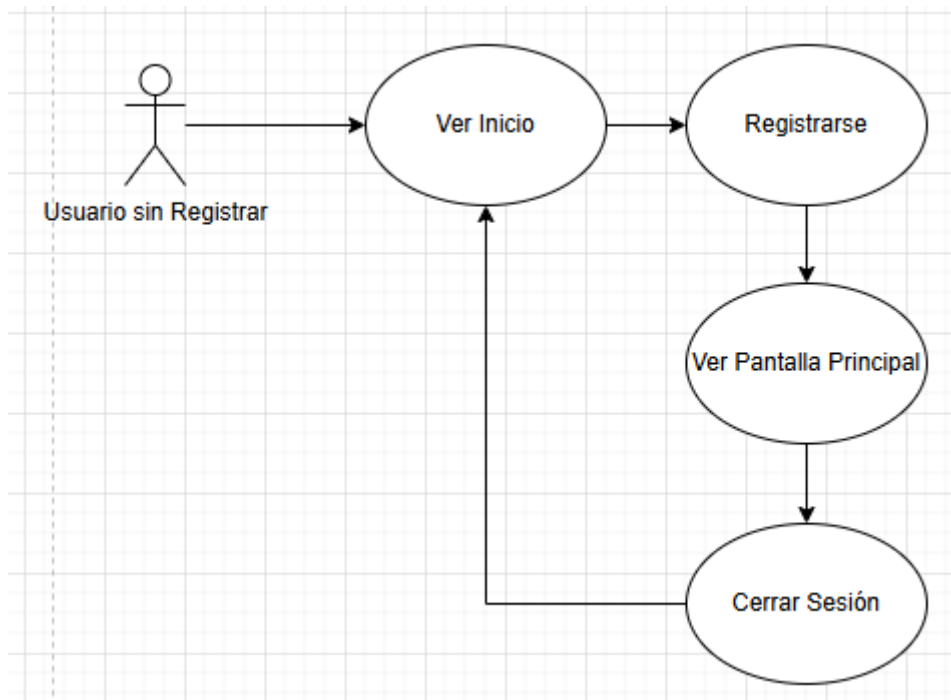


Figura 5: Diagrama de casos de uso para la gestión de sesiones

Los primeros casos de uso de la aplicación son los responsables de la gestión de la sesión, en estos, el usuario ve una pantalla de inicio con el formulario de inicio de sesión, se registra, y pasa a ver la pantalla principal, desde la que pueda operar el programa y finalmente cerrar sesión.

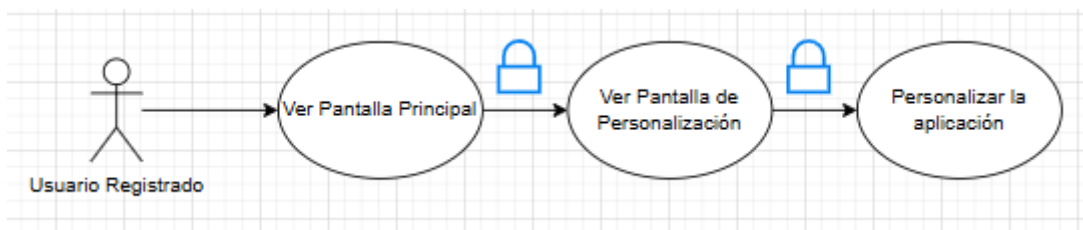


Figura 6: Diagrama de casos de uso para la personalización de la aplicación

En este segundo diagrama, se representan los pasos que sigue un usuario registrado para personalizar la estética de la aplicación. Los candados representan que el usuario solo puede acceder a dichas acciones si tiene los permisos adecuados para ello, en este caso el permiso de personalización.

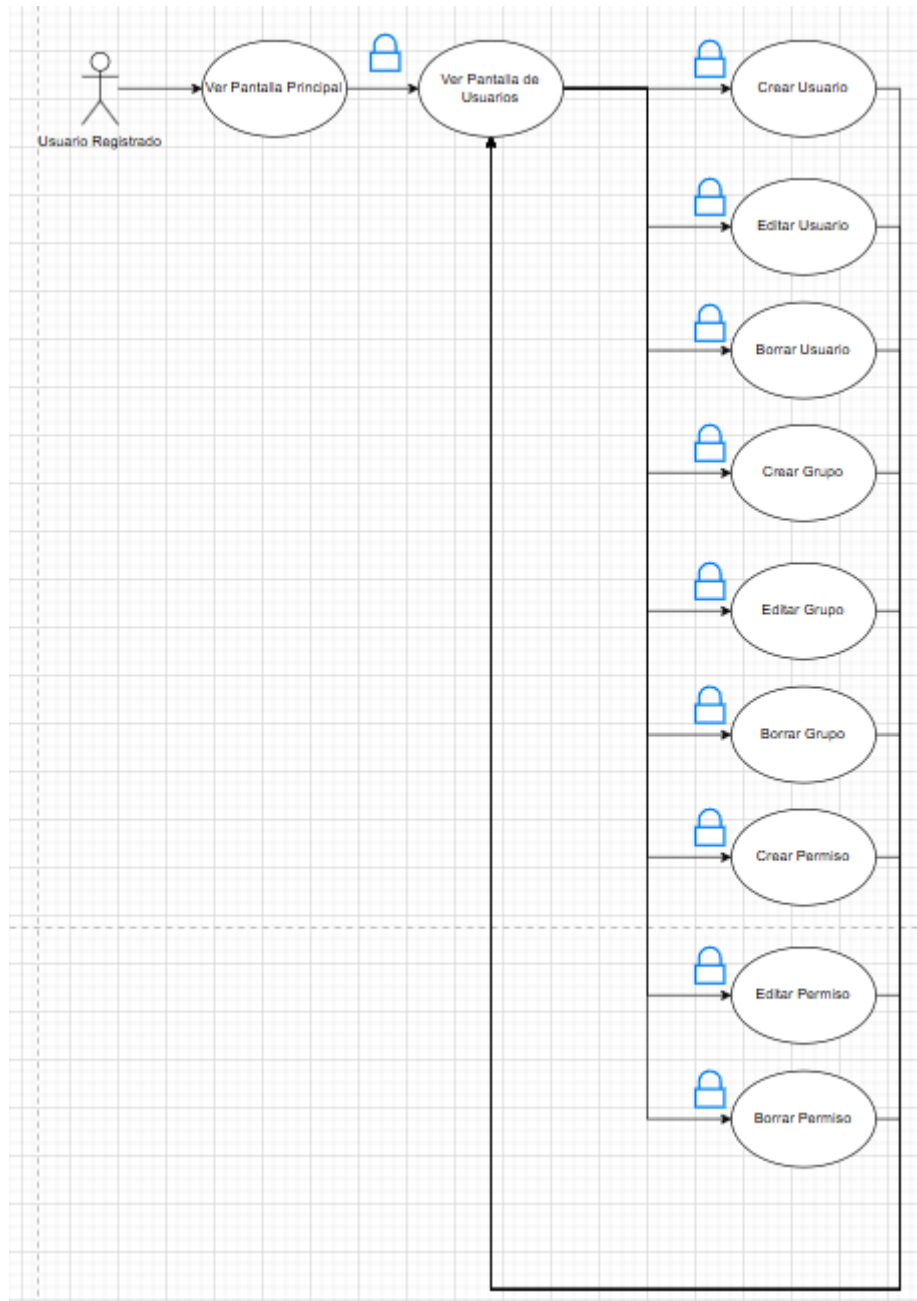


Figura 7: Diagrama de casos de uso para la gestión de usuarios, grupos y permisos

En este diagrama, se tratan los métodos que tiene el usuario para gestionar usuarios, grupos, y permisos. Aquí, de forma similar al caso de la personalización, si el usuario tiene permiso, accede a una pantalla a la pantalla de gestión, y desde esta tiene las acciones necesarias para todas estas gestiones.

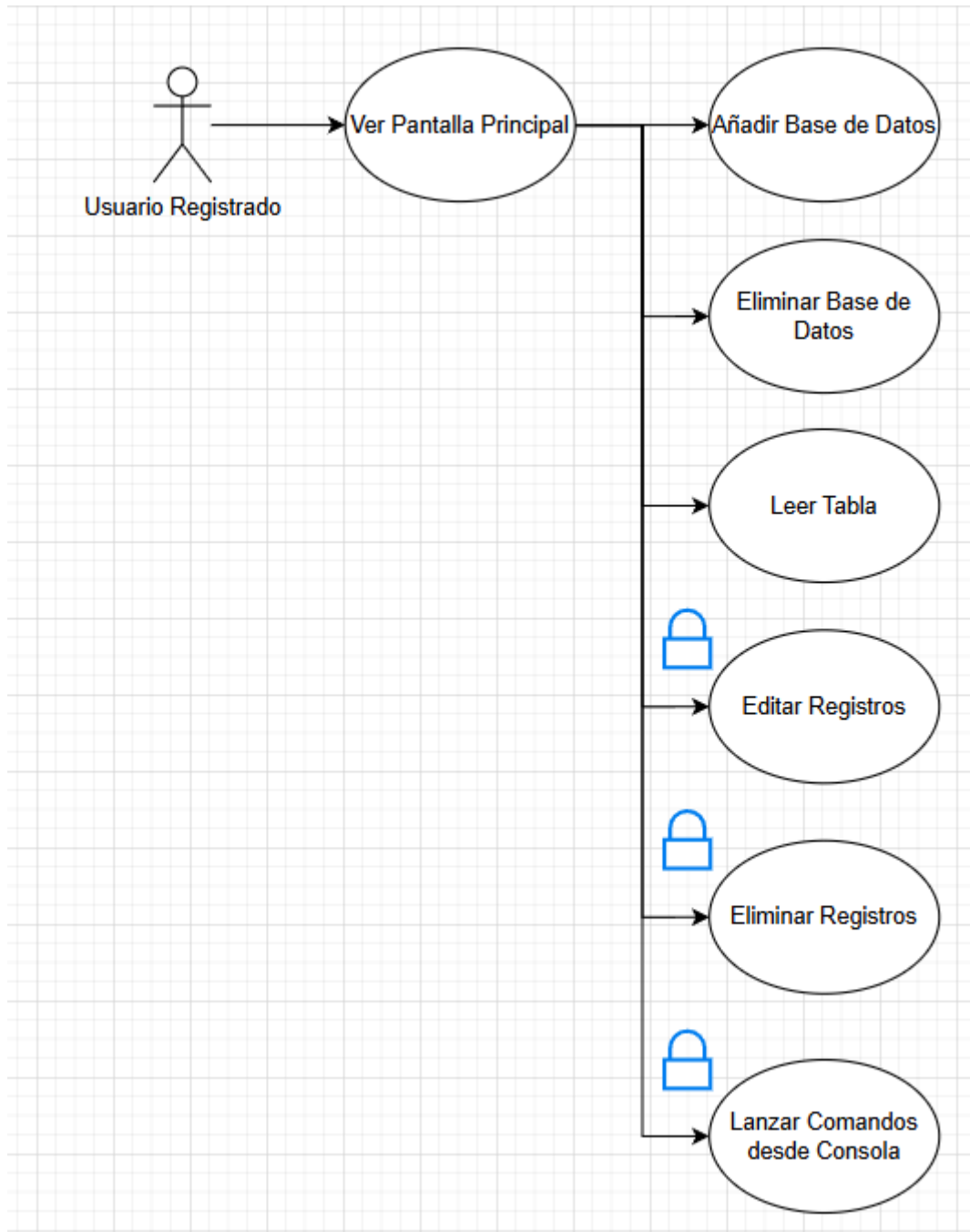


Figura 8: Diagrama de métodos de la pantalla principal

En esta pantalla, el usuario puede realizar las gestiones de base de datos que necesite siempre y cuando se tenga los permisos necesarios.

4.2.5. Site map

Entendidas las acciones que puede realizar el usuario en la aplicación, se ha considerado necesaria la creación de un diagrama que explique la distribución de las páginas web de la aplicación para simplificar la navegación por la misma, cosa que podemos ver en el siguiente diagrama:

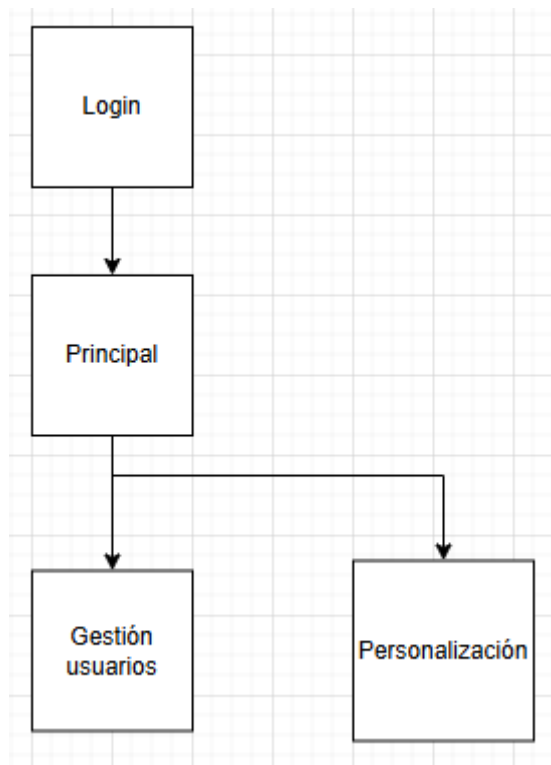


Figura 9: Site map de la aplicación

Como se puede observar, muchos de los métodos indicados en el diagrama de casos de uso no se contemplan en site map, esto se debe a que se ha optado por trabajarlas por medio de API Restful y formularios en formato de modal y así reducir el número de cargas de pantalla durante la ejecución de la aplicación, haciéndola más dinámica.

4.3. Desarrollo

Para la división de responsabilidades en la aplicación, se ha empleado el modelo vista controlador, dado que es el sistema de desarrollo más extendido en el desarrollo de las aplicaciones web, y facilita la extensibilidad y el control de errores de las aplicaciones. En adición, para el apartado visual de la aplicación, se ha optado por emplear Bootstrap, como librería de clases que otorgan un aspecto uniforme a la aplicación, Font Awesome, que proporciona una librería de iconos fácilmente reconocibles e implementables, scripts de JavaScript para tareas menores como el cambio de idioma, o métodos sueltos, y Vue.js, para componentes más complejos y que requieren una reactividad directa a las acciones del usuario.

Además de esto, se ha tenido en cuenta que no todas las interacciones con la base de datos se van a hacer pasando directamente por el entorno servidor, sino que algunas se hacen por medio de peticiones API Restful, para lo cual se ha empleado la librería JQuery, generalmente en métodos de los componentes de Vue.js.

Finalmente, se ha dividido el trabajo de la aplicación por responsabilidades, dividiendo las pantallas de la aplicación según las responsabilidades que puedan tener los usuarios.

Al principio, se generaron las plantillas, migraciones, rutas, y métodos para la gestión de sesiones. Durante este proceso, se optó por generar un sistema de gestión de sesiones personalizado de la aplicación para aumentar la seguridad de la misma. Posteriormente se procedió a diseñar el sistema de permisos de usuario. Cuando esto ya funcionaba se generó lo necesario para la personalización de la aplicación, y finalmente, se trabajó en los recursos necesarios para la gestión de la base de datos. Durante este proceso, se tuvo en cuenta la necesidad de encriptar los datos de conexiones y las contraseñas para evitar posibles robos de información.

Una vez estaban funcionando las pantallas básicas y el sistema de permisos, se procedió a generar las API's necesarias para la pantalla de gestión de bases de datos. Aquí, se tuvo que lidiar con el problema de que Django cachea las conexiones a base de datos, y se tuvo que generar un método específico que evitar que genera una conexión temporal para las API's, de esta forma, evitando que se dé el caso de que se la caché sobreescriba la conexión con la base de datos.

Posteriormente, se procedió a la generación de los componentes Vue.js necesarios para el trabajo con las bases de datos. Para el funcionamiento gráfico de las conexiones, se optó por trabajar con Vue.js, por ser un framework centrado en el entorno cliente y de fácil integración via CDN y se empleo mediante la creación de cuatro componentes, uno principal que se monta en la página principal y se hace cargo de mostrar los errores que se puedan producir y de distribuir los otros tres, otro encargado de mostrar las conexiones que tiene el usuario guardadas en la aplicación y las acciones disponibles en cada conexión, otro encargado de las acciones las búsquedas en tabla o en el caso de MongoDB, en colección. Y otro encargado de trabajar como consola para que el usuario pueda lanzar comandos. Todo lo cual, se puede observar mejor en el siguiente diagrama:

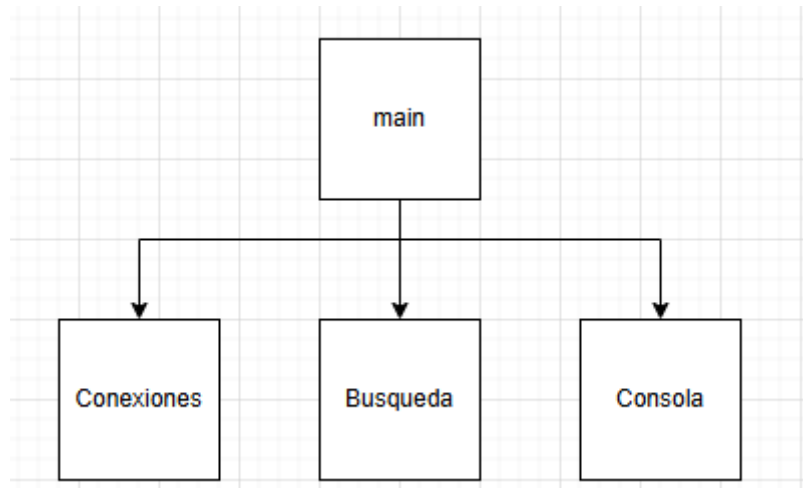


Figura 10: Diagrama de la distribución de componentes Vue

Cuando ya estaba funcionando la gestión de la base de datos, se hizo la aplicación multilingüe. Para lograr esto, Django, tiene una librería por defecto que permite la traducción de plantillas, sin embargo, se comprobó, que dicha librería no funciona correctamente en todos los sistemas operativos, y en algunos requiere de programas adicionales para complementarla. Para solucionar esto, se instaló la librería babel en la propia aplicación, la cual puede hacer lo mismo por medio de archivos de traducción .po, y un proceso de compilación.

Entonces, se procedió a integrar la aplicación en Docker. Esto se hizo con la intención de que la instalación de los drivers para SQL Server y MongoDB fueran automáticos y simplificara la instalación de la aplicación, y procedió por un proceso de prueba y error en el que se fueron añadiendo las librerías necesarias conforme las iba requiriendo el framework para conectarse a las diferentes bases de datos. Así mismo, también hace que la instalación de la aplicación en diferentes sistemas operativos se estandarice. Para lograr esto, se generó un archivo Dockerfile encargado de generar la imagen de la aplicación, y un archivo docker-compose.yml, encargado de generar un contenedor mediante la imagen y generar un contenedor de PostgreSQL con la base de datos de la aplicación, que se coordine con la aplicación mediante una red interna y solo sea accesible desde el propio docker o la red interna de los dos servicios, así como un archivo .env que guarda los datos de configuración de la base de datos y la aplicación para que puedan ser configurados previa la generación del contenedor.

Durante el proceso de integración, se comprobó, que existía un problema no contemplado originalmente durante el planteamiento del proyecto, y ese problema, era que las bases de datos SQLite, no trabajan por host, como lo harían otras bases de datos SQL, en su lugar, trabajan con archivos .db que se guardan en el sistema, para solucionar este problema, se habilitó dentro de la carpeta static, una nueva carpeta llamada sqlite, en la que el usuario, puede almacenar sus archivos .db y acceder a ellos mediante el panel de configuración de las conexiones. Para que el

usuario pueda acceder, debe alojar el archivo dentro de la carpeta indicada, y cuando configura la conexión, no necesita indicar el host ni el puerto, y el nombre de la base de datos debe indicarse como el nombre del archivo.

Cuando este proceso terminó, se realizó una batería de pruebas sobre toda la aplicación para comprobar su correcto funcionamiento.

4.4. Pruebas

Para demostrar el correcto funcionamiento de la aplicación, ha sido necesario realizar una serie de pruebas para comprobar que la aplicación se encuentra en una fase en la que puede ser empleada en un entorno de producción. Sin embargo, debido a la naturaleza de las múltiples funcionalidades del proyecto, se ha optado por dividir las pruebas de funcionalidad en varios apartados: pruebas de usuario, pruebas de personalización, pruebas de gestión de bases de datos, y control de acceso.

4.4.1. Pruebas de usuario

Las pruebas de usuario, se refieren a las pruebas de control de usuarios, es decir, su creación, su visualización, su uso en sesiones, su edición y su eliminación. Es necesario comprobar que solo los usuarios con permiso de usuarios pueden acceder a las rutas relacionadas con la gestión de usuarios, que todos los usuarios pueden iniciar y cerrar sesión, y que la aplicación es capaz de distinguir a un usuario que ha iniciado sesión de un usuario que no ha iniciado sesión.

Con respecto al inicio de sesión, cuando un usuario entra por primera vez a la aplicación, ve la pantalla de inicio de sesión como se muestra a continuación:

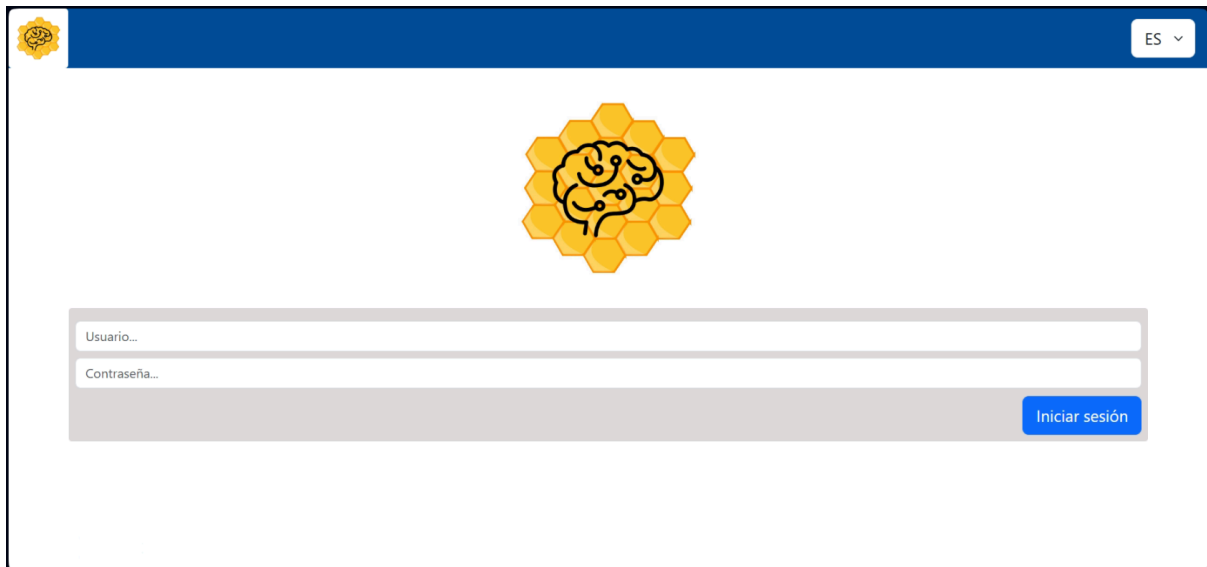


Figura 11: Pantalla de inicio de sesión de la aplicación

Por defecto, cuando se instala la aplicación, viene con un usuario inicial llamado admin, de contraseña 1234, con todos los permisos de la aplicación. Este usuario se crea para que se pueda realizar un primer acceso a la aplicación, cambiar la contraseña y empezar la gestión de usuarios y el trabajo. Si se rellena el formulario de la pantalla y se pulsa el botón de inicio de sesión, se inicia la sesión y se redirige al usuario a la pantalla principal de la aplicación como se puede observar a continuación:

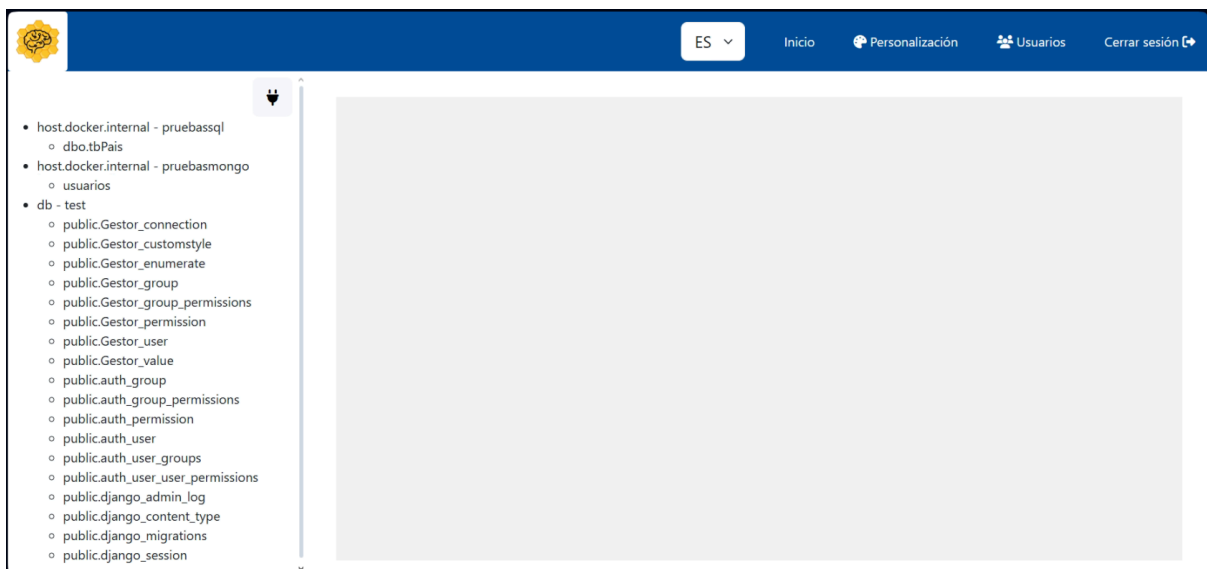


Figura 12: Pantalla principal de la aplicación

Esta es la pantalla en la que el usuario va a pasar más tiempo trabajando, dado que es la pantalla en la que se gestionan las bases de datos. En la navegación, se puede observar las opciones que tiene el usuario a la hora de navegar por la

aplicación. Estas opciones varían según los permisos que tenga el usuario. Si se pulsa cerrar sesión en la navegación, se cierra la sesión, y se retorna a la pantalla de inicio de sesión.

Para gestionar los usuarios, grupos y permisos de la aplicación, el usuario debe pulsar la opción usuarios de la navegación, lo cual lo lleva a la siguiente pantalla:

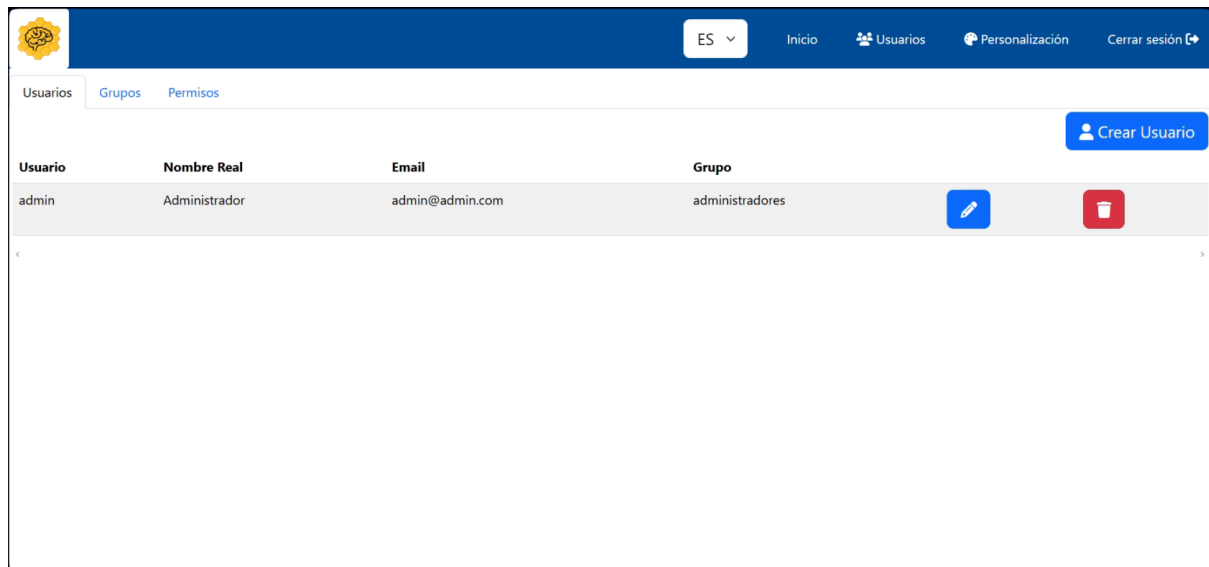


Figura 13: Pantalla de gestión de usuarios, grupos, y permisos

En esta pantalla se gestionan las operaciones de creación, edición, y eliminación de los usuarios, grupos, y permisos. Para cada uno de los tipos de gestión que permite la aplicación tiene habilitada una pestaña, en la cual lista todos los registros de dicho tipo, así pues, en la pestaña de usuarios, se listan todos los usuarios, en la de grupos todos los grupos, y en la de permisos todos los permisos. Dentro de la propia pestaña, existe un botón para crear un nuevo registro del tipo de la pestaña indicada, al pulsarlo, se muestra un formulario vacío para que se rellene la información del registro a guardar, se puede ver el formulario aquí:

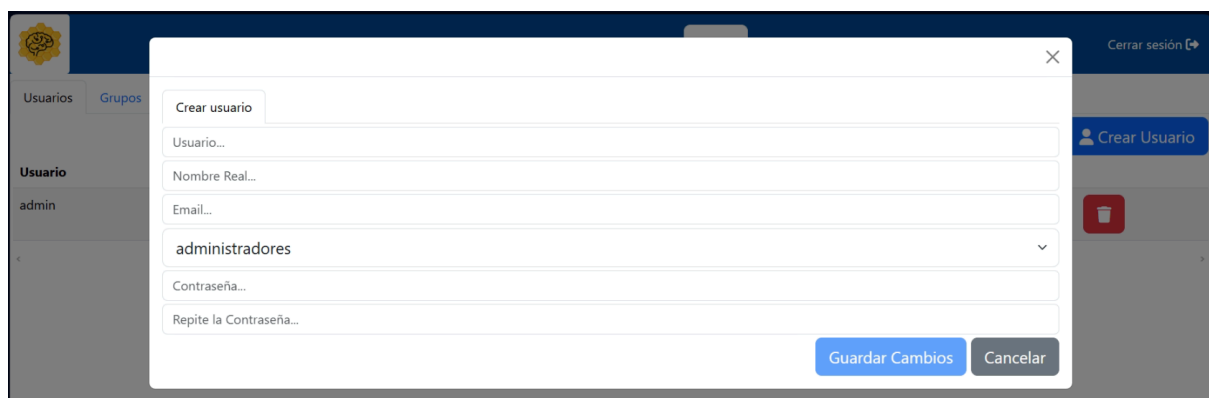


Figura 14: Formulario de creación de usuario

La información que se muestra en dichos formularios depende del tipo de registro que se vaya a generar, en este caso, se muestra el formulario de un usuario por ser el más extenso. Si en este modal se rellenan los datos y se pulsa guardar, se recarga la página y se puede observar el nuevo usuario generado:

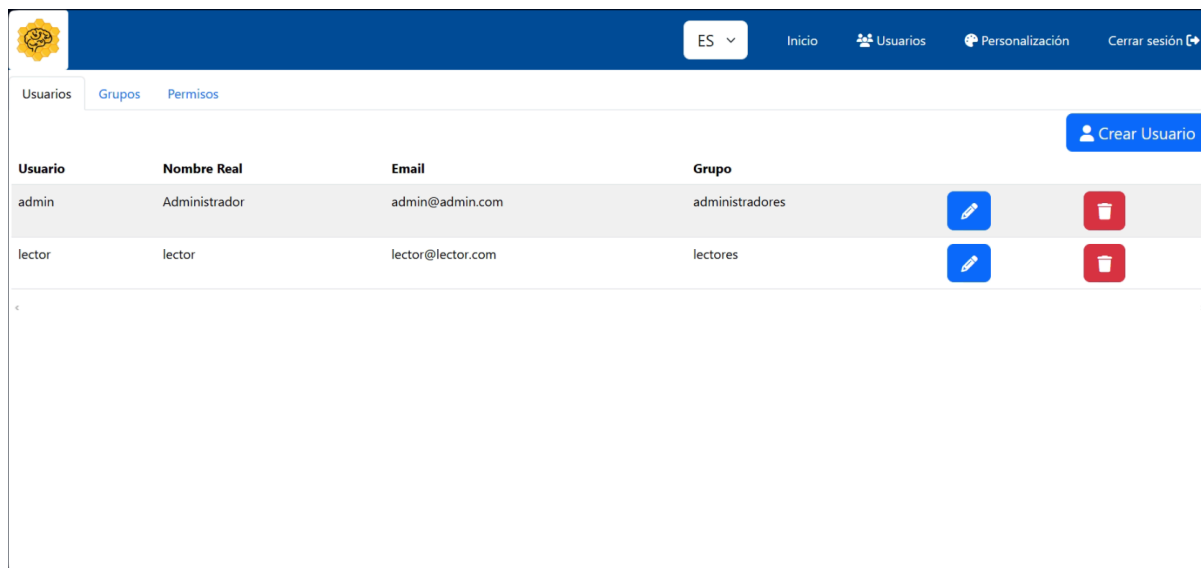


Figura 15: Pantalla de gestión de usuarios con el nuevo usuario generado

Para cualquier registro, si se pulsa el botón con el icono de lápiz, se abre un modal con un formulario de edición como se observa a continuación:



Figura 16: Modal de edición de usuario

El modal se autorrellena con la información del registro para que pueda ser modificada. En el caso de los usuarios, este modal tiene dos formularios, separados en pestañas, el primero es para modificar todos los datos con excepción de la contraseña, y el segundo, es para modificar la contraseña. Esto se hace así para evitar posibles cambios accidentales en la contraseña. Si se modifican los datos del

usuario aquí y se pulsa guardar cambios, se recarga la página y se puede observar que los cambios se han almacenado como se ve a continuación:

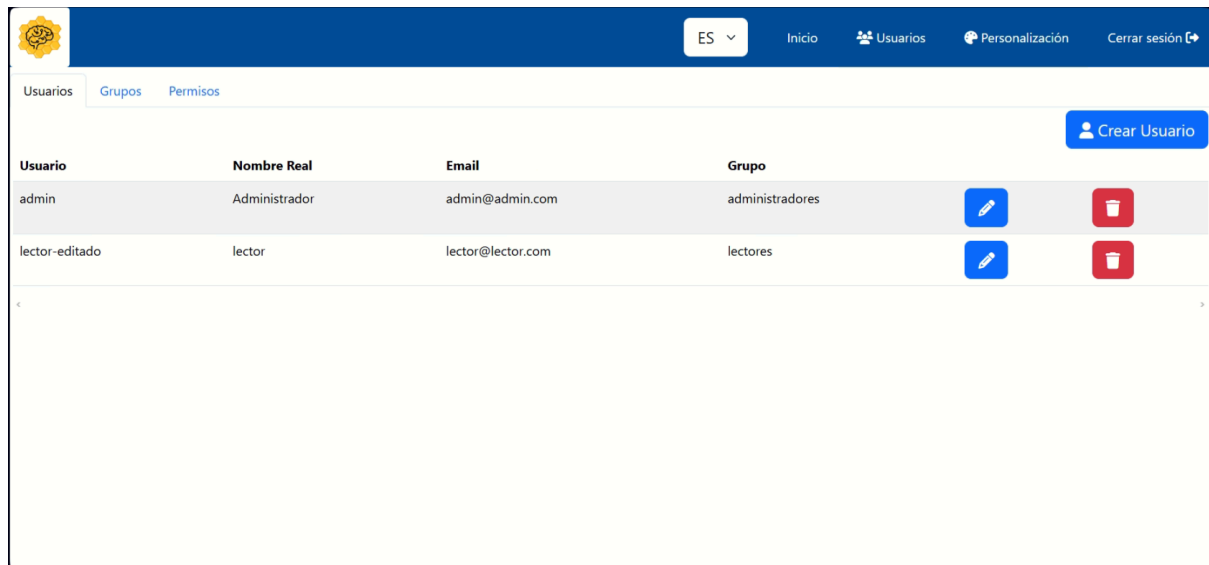


Figura 17: Pantalla de gestión de usuarios con usuario editado

A continuación, se prueba a borrar el usuario editado, para lo cual se pulsa el botón rojo con el icono de papelera, y se observa el siguiente modal de advertencia:

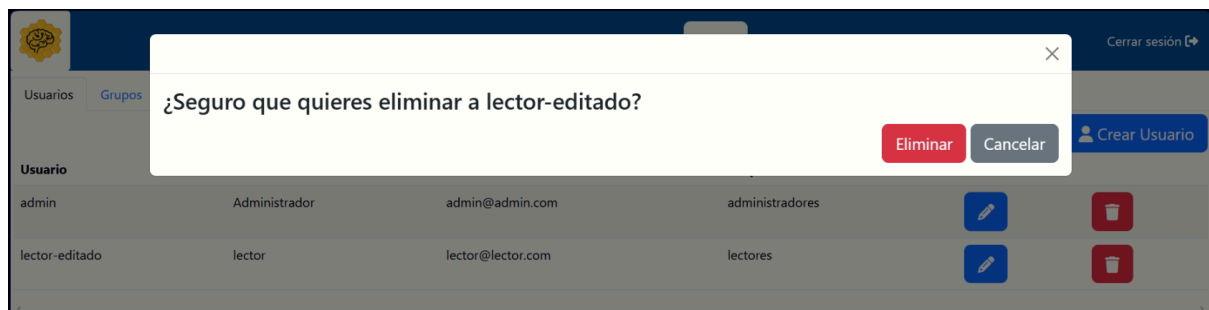


Figura 18: Modal de eliminación de usuario

En este modal se pulsa el botón de eliminar, se elimina al usuario, y se recarga la página para reflejar los cambios. Todos los métodos indicados para estas pruebas funcionan para grupos y permisos de la misma manera con una salvedad, si se elimina un grupo, también se eliminan los usuarios que pertenezcan a dicho grupo.

4.4.2. Pruebas de personalización

Las pruebas de personalización, se refieren a las pruebas que se han realizado para comprobar que la personalización estética de la aplicación funciona correctamente. Es necesario comprobar que solo se puede acceder en caso de que se tenga el

permiso de personalización, y que los cambios que se realicen sobre la estética de la aplicación se reflejen correctamente sobre la misma.

Si el usuario tiene permiso para acceder a la personalización, una vez ha iniciado sesión, en la navegación encontrará una opción llamada personalización, la cual muestra al usuario la siguiente pantalla:

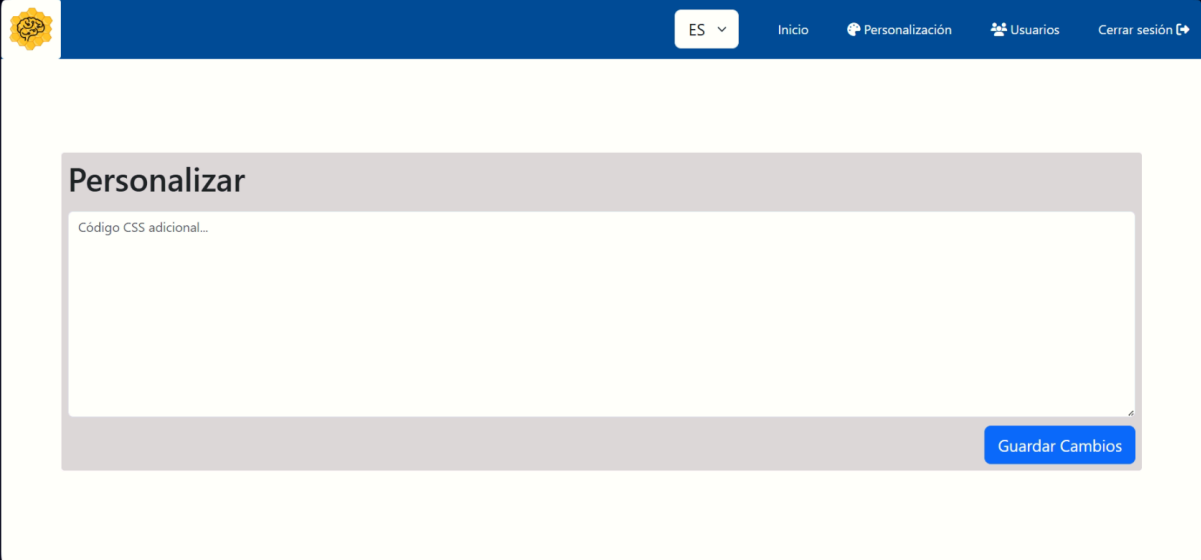


Figura 19: Pantalla de personalización

En esta pantalla, el usuario puede introducir el código CSS adicional que considere necesario, y este código CSS se aplicará a todas las páginas de la aplicación. Por ejemplo, si se desea cambiar el color de la navegación a fondo amarillo, basta con introducir el código CSS correspondiente y pulsar guardar cambios, lo cual recarga la página con los cambios tal y como se observan a continuación:



Figura 20: Pantalla de personalización con los cambios de CSS

Se puede observar, que los cambios, también se aplican al resto de páginas.

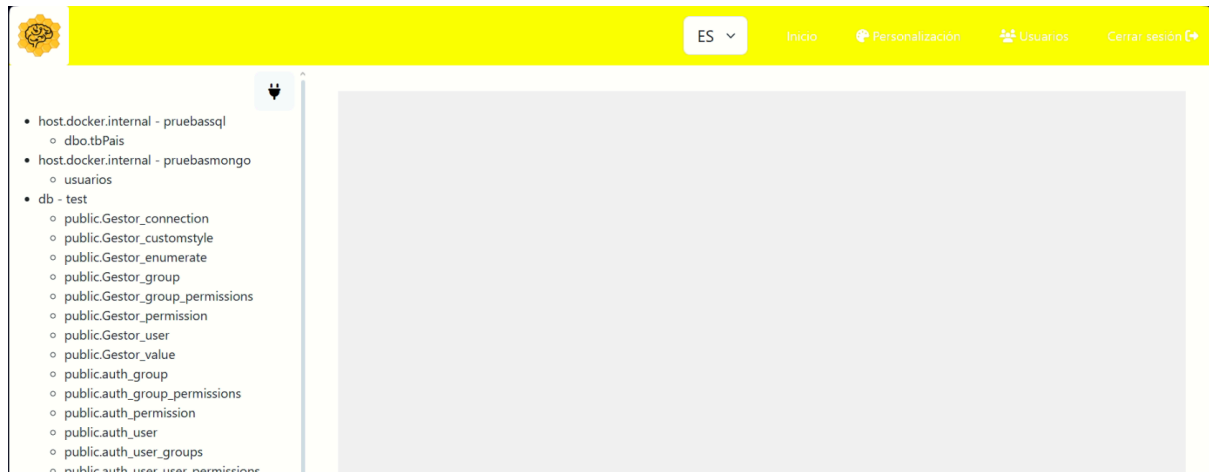


Figura 21: Pantalla principal con cambios en el CSS

Finalmente, si un usuario distinto abre la aplicación, puede ver los mismos cambios aplicados.

4.4.3. Pruebas de gestión de bases de datos

Las pruebas de gestión de bases de datos, hacen referencia a las pruebas realizadas para controlar diferentes bases de datos. Es necesario comprobar que se listan correctamente las bases de datos, las pantallas, y las colecciones, que se puede generar una nueva conexión, que se puede destruir una conexión, que se pueden realizar búsquedas sobre las tablas y las colecciones, que se puede emplear la consola para ejecutar queries manualmente, y que se puede importar los resultados en JSON o CSV.

Una vez iniciada la sesión, se accede a la pantalla principal de la aplicación, en la cual se tiene una columna a la izquierda, en la que se listan todas las conexiones de la aplicación, especificando para cada conexión su host y el nombre de la base de datos a la que se ha conectado, y para cada conexión listando todas sus tablas o conexiones. Adicionalmente, en esta columna se debe ver un botón con forma de enchufe que sirve para abrir el modal de generación de conexiones nuevas. En este caso, al entrar por primera vez en la aplicación, no se tienen conexiones establecidas, por lo que la lista de conexiones aparece vacía como se puede observar a continuación:

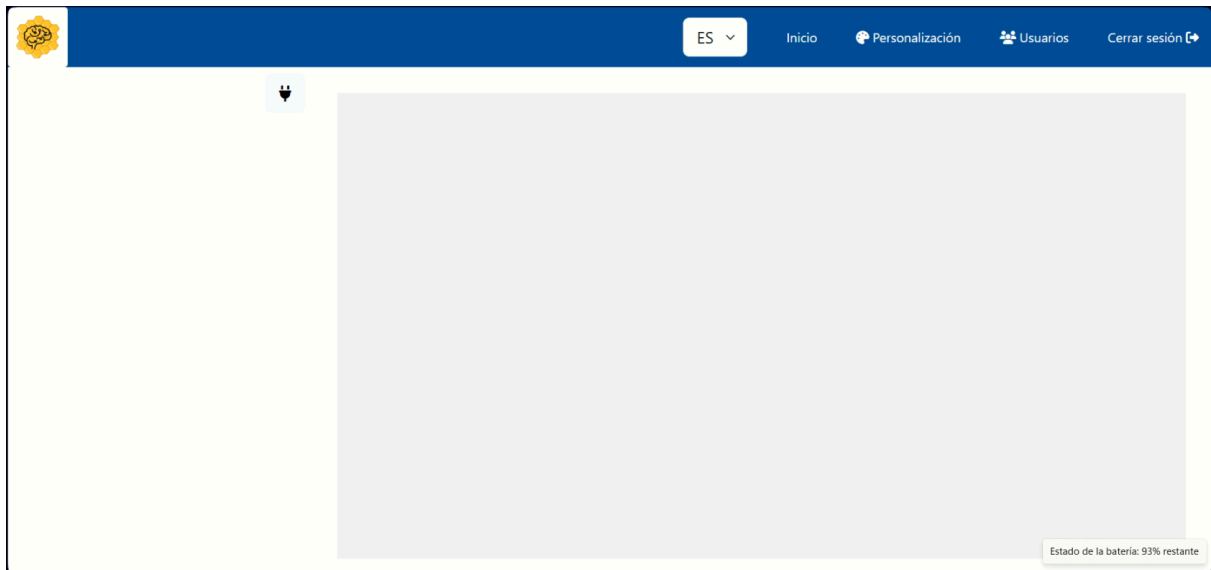


Figura 22: Pantalla principal de la aplicación

Para generar una conexión se debe pulsar el botón con el icono de enchufe, el cual muestra el siguiente modal:

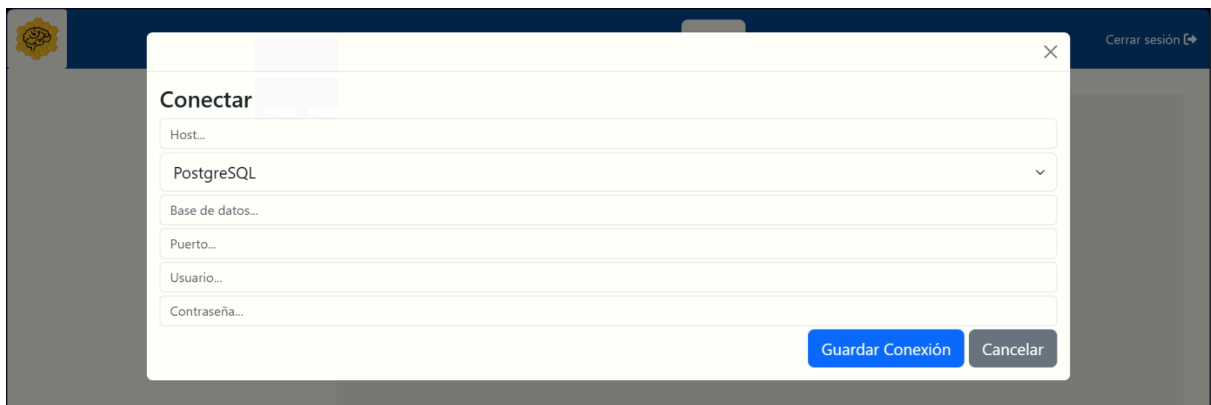


Figura 23: Modal de generación de conexión

En este modal se indican los datos de la conexión incluyendo host, tipo de base de datos, nombre de la base de datos, puerto, usuario y contraseña, y se debe pulsar guardar la conexión. Al hacerlo, si la conexión es correcta, se guarda en la aplicación, y procede a listarse en la columna izquierda para que el usuario pueda emplearla en sus consultas como se muestra a continuación:

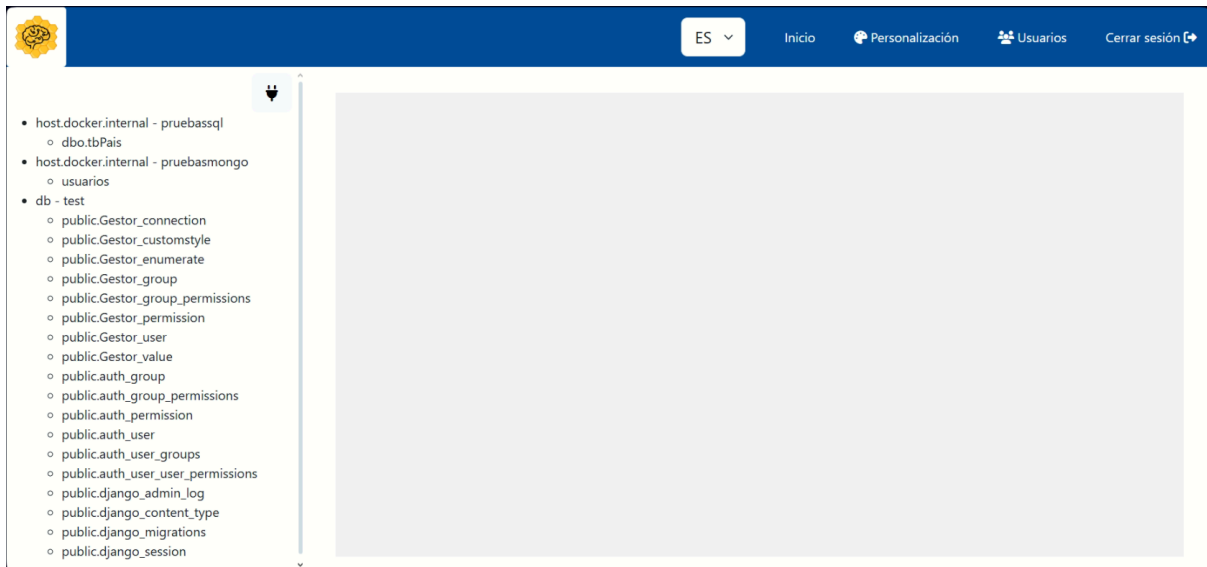


Figura 24: Pantalla principal con varias conexiones generadas

Aquí se muestran varias conexiones que se han generado para poder realizar diferentes pruebas sobre la aplicación. Un punto importante a tener en cuenta sobre la generación de conexiones, es que dado que la aplicación se despliega en Docker, se considera una máquina virtual, y para hacer conexiones a localhost, por ejemplo en casos como bases de datos en el mismo dispositivo, se tiene que emplear como host el host: `host.docker.internal`, el cual Docker se encarga de interpretar automáticamente como peticiones de localhost para el resto del dispositivo.

Para poder realizar una búsqueda sobre una tabla o colección de una base de datos, se debe pulsar el botón derecho del ratón sobre la tabla o conexión sobre la que interesa realizar las pruebas. Al hacerlo, se muestra el siguiente menú contextual:

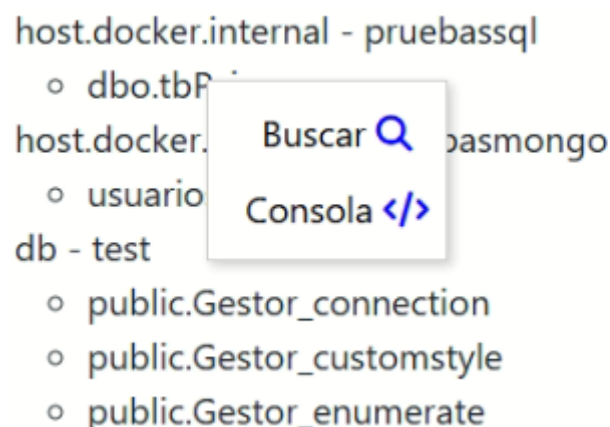


Figura 25: Menú contextual de acciones de tabla

Aquí se pulsa la opción de buscar, la cual modifica la pantalla principal de la siguiente manera:

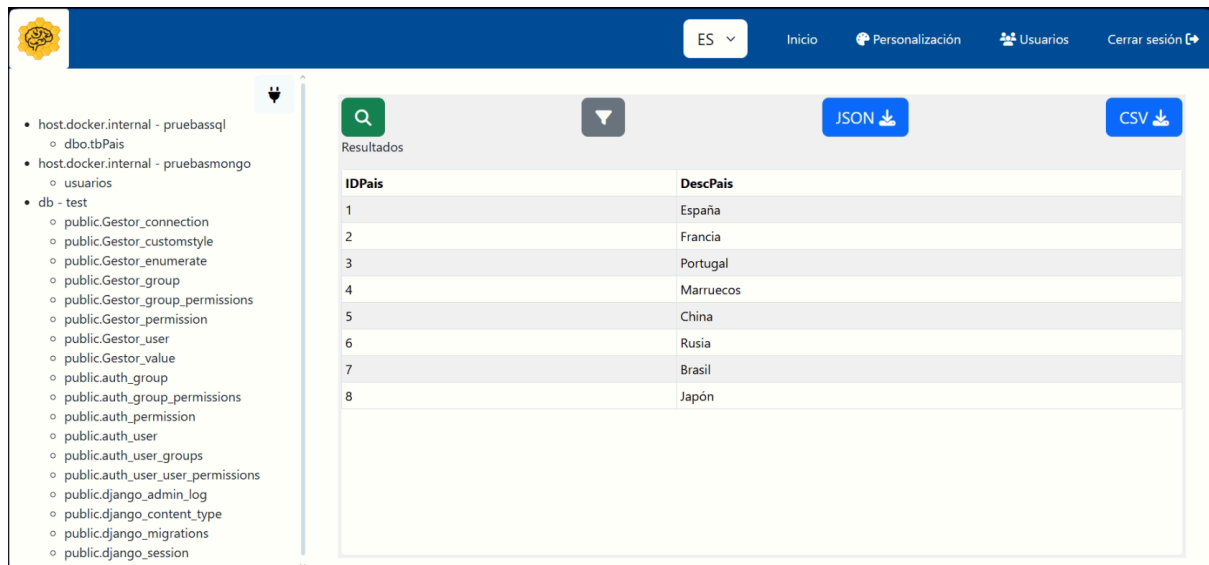


Figura 26: Pantalla principal con acción de búsqueda

Aquí se muestran los datos de la tabla, con un panel oculto de filtros de búsqueda, el cual se puede emplear para aumentar o reducir los datos obtenidos de la búsqueda. Para acceder a los filtros se debe pulsar el botón gris con el icono de filtro, y para poder lanzar una búsqueda, se debe pulsar el botón con el icono de lupa. A continuación, se muestra una búsqueda con filtros para ver un ejemplo:

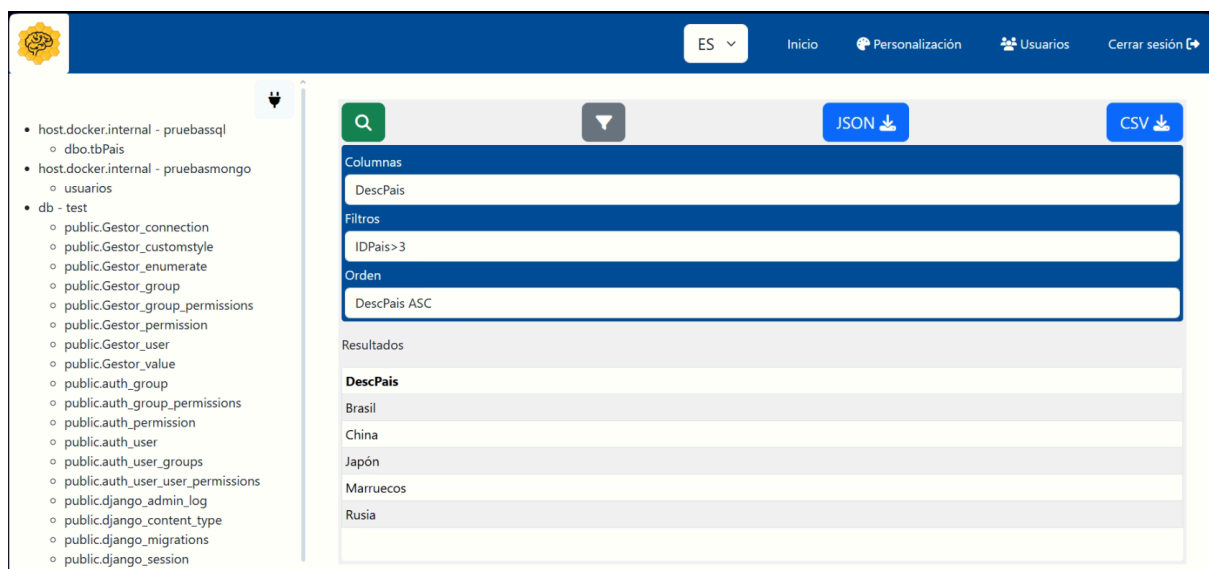


Figura 27: Pantalla principal con una búsqueda filtrada

Seguidamente, si se pulsa el botón de JSON, se descarga una versión JSON de los resultados de búsqueda, que si se abre, muestra algo como lo siguiente:

```
[
  {
    "DescPais": "Brasil"
  },
  {
    "DescPais": "China"
  },
  {
    "DescPais": "Japón"
  },
  {
    "DescPais": "Marruecos"
  },
  {
    "DescPais": "Rusia"
  }
]
```

Figura 28: Ejemplo de JSON descargado de los resultados de una búsqueda

Del mismo modo que con el botón de JSON, si se pulsa el botón de CSV, se descarga una versión CSV de los resultados como se muestra a continuación:

```
C: > Users > enriq > Downloads > registros_dbo.tbPais (1).csv > data
1  IDPais,DescPais
2  "7","Brasil"
3  "5","China"
4  "8","Japón"
5  "4","Marruecos"
6  "6","Rusia"
```

Figura 29: Ejemplo de CSV descargado de los resultados de una búsqueda

A continuación, se procede a acceder a la consola para lanzar queries manualmente. Para hacerlo, se pulsa consola en el menú contextual de la tabla, lo cual modifica la pantalla principal de la siguiente manera:

Gestor de de datos multiplataforma

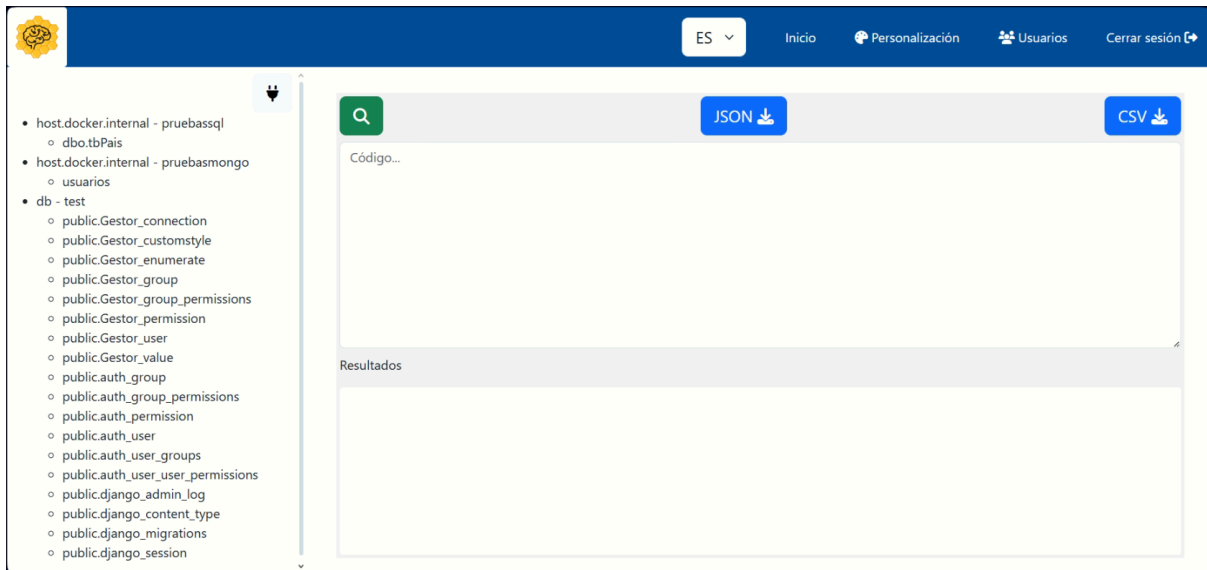


Figura 30: Pantalla principal con la consola abierta

En esta pantalla, se puede ejecutar código en el lenguaje de la base de datos que se esté empleando. Por ejemplo, una búsqueda de SQL Server como se observa a continuación:

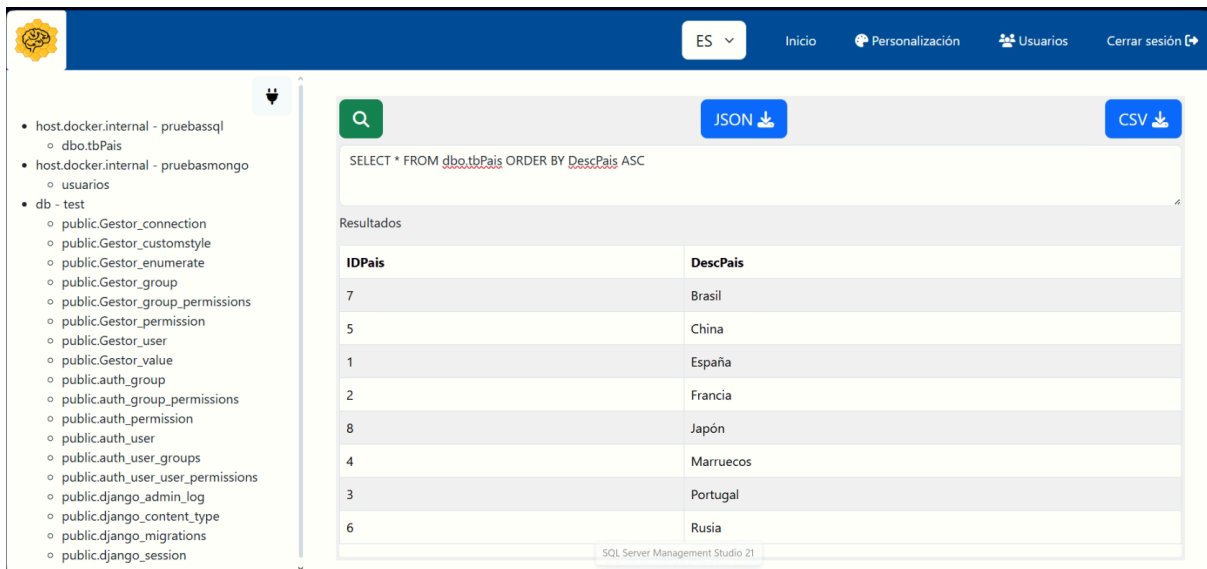


Figura 31: Pantalla principal tras la ejecución de un comando de consola

Como se puede observar, la pantalla cambia para mostrar los resultados de la ejecución del comando escrito en la consola. Del mismo modo que con la acción de búsqueda, se pueden descargar como JSON y CSV los resultados obtenidos.

Finalmente, para eliminar una conexión, se debe pulsar el botón derecho del ratón sobre la conexión que se desea borrar, para ver el siguiente menú contextual:

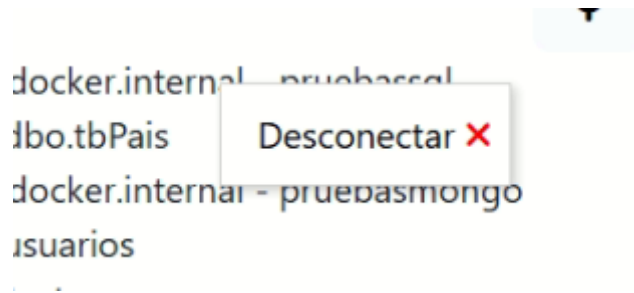


Figura 32: Menú contextual de conexión

Aquí, se pulsa desconectar, y al hacerse, la conexión es eliminada como se puede observar a continuación:

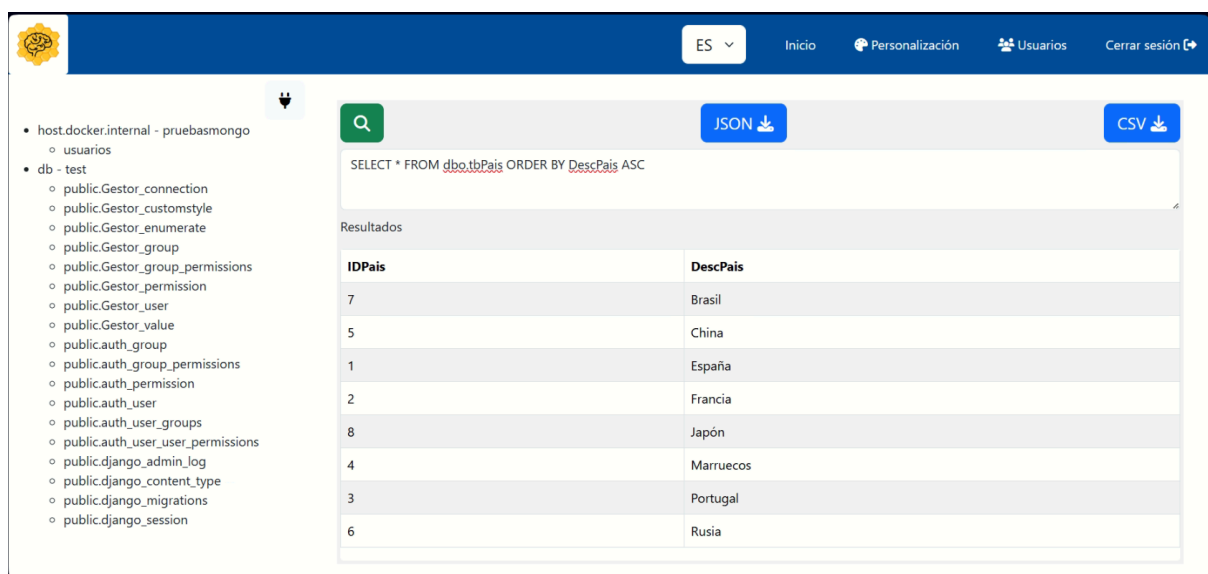


Figura 33: Pantalla principal tras la eliminación de una conexión

Finalmente, las conexiones registradas en la aplicación, solo pueden ser vistas por el usuario que las ha generado, mientras que otros usuarios no pueden visualizar las conexiones que haya generado el usuario. En caso de que otro usuario necesite conectarse a la misma base de datos, tiene que generar la conexión en su propia sesión.

4.4.4. Pruebas de control de accesos

Las pruebas de control de acceso se refieren a las pruebas realizadas para garantizar la seguridad de la aplicación. En estas pruebas, se comprueba que visualmente el usuario no puede ver opciones a las que no tenga acceso, y que en caso de intentar acceder por URL, se controlen los permisos del usuario, y se redirija al usuario de vuelta a pantalla en las que tenga permisos en caso de que no tenga acceso a la pantalla objetivo. Del mismo modo, para las API's, en caso

de que el usuario no tenga acceso a las API's, estas deben rechazar los intentos de acceso.

Primeramente, los usuarios solo tienen acceso a la pantalla de inicio de sesión, la cual ya se ha visualizado en la figura 11, en esta pantalla solo se tiene acceso al inicio de sesión, y cuando no se ha iniciado sesión, el intento de acceso a cualquier otra pantalla redirecciona a esta para evitar posibles accesos indebidos.

Cuando se inicia sesión, se accede a la pantalla principal, y según el grupo en el que esté asignado el usuario, se determinan los permisos que tiene dentro de la aplicación.

Dentro de la pantalla principal, las opciones de la navegación dependen de los permisos del grupo del usuario. Solo se visualiza la opción de usuarios, si el grupo del usuario, tiene el permiso de usuarios habilitado, y sólo se visualiza la opción de personalización, si se tiene el permiso de personalización. En ambos casos, si no se tiene el permiso, las opciones no aparecen, y si se intenta acceder, se retorna al usuario a la pantalla principal.

En la pantalla principal, además, la opción de consola del menú contextual, sólo aparece si el usuario tiene el permiso de edición. En caso de que el usuario intente lanzar una petición API a alguna API para la que no tenga permiso, se le devuelve un error de permisos insuficientes.

Finalmente, cabe destacar, que a todo esto se le añade la seguridad adicional que tenga la base de datos para el usuario que se emplee en la conexión.

4.5. Documentación

Junto a este archivo, el usuario puede encontrar la siguiente documentación relativa a la aplicación:

La carpeta GestorDDBB, que contiene el código fuente de la aplicación, así como los archivos necesarios para ser implementada en un sistema operativo Windows, Linux, o Mac que tenga instalado y funcionando Docker Engine.

El archivo Instrucciones de instalacion.pdf, que contiene las instrucciones necesarias para la instalación y ejecución de la aplicación.

5. Trabajos futuros

Si bien esta aplicación ha logrado cumplir los objetivos indicados en la especificación, como se ha podido comprobar, no deja de ser cierto, que todavía hay cosas en las que la aplicación puede mejorar y que están previstas como futuras actualizaciones de la aplicación.

En futuras actualizaciones se busca pasar la pantalla principal a trabajar con un modelo de pestañas, para así facilitar el trabajo en casos donde se necesita trabajar sobre múltiples tablas y bases de datos simultáneamente.

Se busca también ir aumentando los tipos de bases de datos a los que se puede conectar la aplicación. En la siguiente actualización, se espera preparar la aplicación para que pueda trabajar con bases de datos de Java.

Dado que en la actualidad, la aplicación solo permite la importación de datos por comandos de consola, se preparará un método para la importación de datos que permita el uso de archivos como sql.

Se pretende generar un método de inserción y actualización de registros que funcione de forma gráfica y se diferencie de la consola, para así aumentar la seguridad de la aplicación.

Se desea preparar un método gráfico para comprobar la seguridad que tiene configurada la base de datos en términos de usuarios y permisos.

Se va a preparar en actualizaciones futuras, un método para poder revisar gráficamente las conexiones de las tablas en la base de datos.

6. Conclusiones

En el transcurso de este trabajo final de ciclo, se ha desarrollado una aplicación multiplataforma para la gestión de diferentes tipos de bases de datos. A través de este proceso, se han logrado los siguientes objetivos: se han superado todos los objetivos funcionales de la aplicación, se ha desarrollado una interfaz de usuario intuitiva que facilita una gestión eficiente de las bases de datos, se han integrado exitosamente las herramientas de gestión de usuarios, personalización, búsqueda en tablas, y consola para todas las bases de datos planteadas inicialmente, se ha realizado un testeo riguroso del buen funcionamiento de la aplicación para los diferentes casos de uso contemplados en esta, y se ha logrado optimizar la aplicación para garantizar unos tiempos de carga rápidos.

Se puede decir, que este proyecto ha servido para ganar conocimiento en todas las materias asociadas al ciclo, especialmente en la forma de combinarlas para lograr el desarrollo de una aplicación funcional.

En conclusión, este proyecto ha sido una gran combinación de conocimientos teóricos y prácticos obtenidos a lo largo de todo el ciclo que ha servido para aprender a desarrollar aplicaciones multiplataforma basadas en requisitos prácticos y limitaciones de tiempo realistas y sirve como una buena demostración de las habilidades adquiridas a lo largo del ciclo.

7. Bibliografía y webgrafía

Para el desarrollo de este proyecto, se han empleado las siguientes páginas como referencia:

[Cristina Navarro Laboulais y Natalia Sastre Miralles]

Cómo citar la bibliografía en los trabajos académicos

<https://mpison.webs.upv.es/investigacion_aplicada/textos/como_citar_upv.pdf>

[Consulta: 26 de abril de 2025]

[Django]

Django documentation | Django documentation | Django

<<https://docs.djangoproject.com>> *[Consulta: 20 de marzo de 2025]*

[Smartway]

Arranca con Kanban en tu equipo en 6 pasos - SmartWay

<<https://smartway.es/guia-kanban/>> *[Consulta: 25 de marzo de 2025]*

[Bootstrap]

Get started with Bootstrap · Bootstrap v5.3

<<https://getbootstrap.com>> *[Consulta: 01 de abril de 2025]*

[Font Awesome]

Font Awesome

<<https://fontawesome.com>> *[Consulta: 01 de abril de 2025]*

[Vue.js]

Vue.js - The Progressive JavaScript Framework | Vue.js

<<https://vuejs.org>> *[Consulta: 15 de abril de 2025]*

[jQuery]

jQuery API Documentation

<<https://api.jquery.com>> *[Consulta: 17 de abril de 2025]*

[DockerDocs]

Docker Docs

<<https://docs.docker.com>> *[Consulta: 20 de abril de 2025]*

[Git]

Git - Documentation

<<https://git-scm.com/doc>> *[Consulta: 03 de abril de 2025]*

[Python]

3.13.3 Documentation

<<https://docs.python.org/3/>> [Consulta: 25 de marzo de 2025]

[DevDocs]

JavaScript documentation — DevDocs

<<https://devdocs.io/javascript/>> [Consulta: 10 de abril de 2025]

[MDN Web Docs]

JavaScript | MDN

<<https://developer.mozilla.org/en-US/docs/Web/JavaScript>> [Consulta: 10 de abril de 2025]

[PostgreSQL]

PostgreSQL: Documentation

<<https://www.postgresql.org/docs/>> [Consulta: 20 de marzo de 2025]

[pgAdmin]

Documentation

<<https://www.pgadmin.org/docs/>> [Consulta: 20 de marzo de 2025]

[MDN Web Docs]

HTML: Lenguaje de etiquetas de hipertexto | MDN

<<https://developer.mozilla.org/es/docs/Web/HTML>> [Consulta: 20 de marzo de 2025]

[MDN Web Docs]

CSS: Cascading Style Sheets | MDN

<<https://developer.mozilla.org/en-US/docs/Web/CSS>> [Consulta: 20 de marzo de 2025]

[Pip]

pip documentation v25.1.1

<<https://pip.pypa.io/en/stable/>> [Consulta: 20 de marzo de 2025]

[Ionos]

MongoDB y Python: todo sobre PyMongo - IONOS España

<<https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/mongodb-python/>>

[Consulta: 1 de mayo de 2025]

[StackOverflow]

¿Cómo me conecto a una Base de Datos MySQL con Python? - Stack Overflow en español

<<https://es.stackoverflow.com/questions/568/cómo-me-conecto-a-una-base-de-datos-mysql-con-python>> [Consulta: 1 de mayo de 2025]

8. Glosario

API RESTful: “La API RESTful es una interfaz que dos sistemas de computación utilizan para intercambiar información de manera segura a través de Internet. La mayoría de las aplicaciones para empresas deben comunicarse con otras aplicaciones internas o de terceros para llevar a cabo varias tareas. Por ejemplo, para generar nóminas mensuales, su sistema interno de cuentas debe compartir

datos con el sistema bancario de su cliente para automatizar la facturación y comunicarse con una aplicación interna de planillas de horarios. Las API RESTful admiten este intercambio de información porque siguen estándares de comunicación de software seguros, confiables y eficientes.” *¿Qué es una API de RESTful? - Explicación de API de RESTful - AWS*

<<https://aws.amazon.com/es/what-is/restful-api/>> [Consulta: 18 de mayo de 2023]

Diagrama entidad relación: “Un diagrama entidad-relación, también conocido como modelo entidad relación o ERD, es un tipo de diagrama de flujo que ilustra cómo las "entidades", como personas, objetos o conceptos, se relacionan entre sí dentro de un sistema. Los diagramas ER se usan a menudo para diseñar o depurar bases de datos relacionales en los campos de ingeniería de software, sistemas de información empresarial, educación e investigación. También conocidos como los ERD o modelos ER, emplean un conjunto definido de símbolos, tales como rectángulos, diamantes, óvalos y líneas de conexión para representar la interconexión de entidades, relaciones y sus atributos. Son un reflejo de la estructura gramatical y emplean entidades como sustantivos y relaciones como verbos.” *¿Qué es un diagrama entidad-relación? | Lucidchart*

<<https://www.lucidchart.com/pages/es/que-es-un-diagrama-entidad-relacion>>

[Consulta: 18 de mayo de 2023]

jQuery: “jQuery es una biblioteca multiplataforma de JavaScript, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. Fue presentada el 14 de enero de 2006 en el BarCamp NYC. De acuerdo a un análisis de la Web (realizado en 2017) jQuery es la biblioteca de JavaScript más utilizada, por un amplio margen.” *jQuery - Wikipedia, la enciclopedia libre*

<<https://es.wikipedia.org/wiki/JQuery>> [Consulta: 18 de mayo de 2023]

Modelo vista controlador (MVC): “En líneas generales, MVC es una propuesta de arquitectura del software utilizada para separar el código por sus distintas responsabilidades, manteniendo distintas capas que se encargan de hacer una tarea muy concreta, lo que ofrece beneficios diversos.” *Qué es MVC*

<<https://desarrolloweb.com/articulos/que-es-mvc.html>> [Consulta: 18 de mayo de 2023]

Sitemap: “Un Sitemap es un diagrama jerárquico que muestra la estructura de un sitio web, aplicación o producto. Los diseñadores UX y los arquitectos de

información lo utilizan para agrupar contenido similar y así definir la taxonomía del sitio.” *¿Qué es un Sitemap en diseño UX?*

<<https://formiux.com/sitemaps-en-ux/>> [Consulta: 18 de mayo de 2023]

9. Anexos

Anexo I: Diagrama de casos de uso

Gestor de de datos multiplataforma

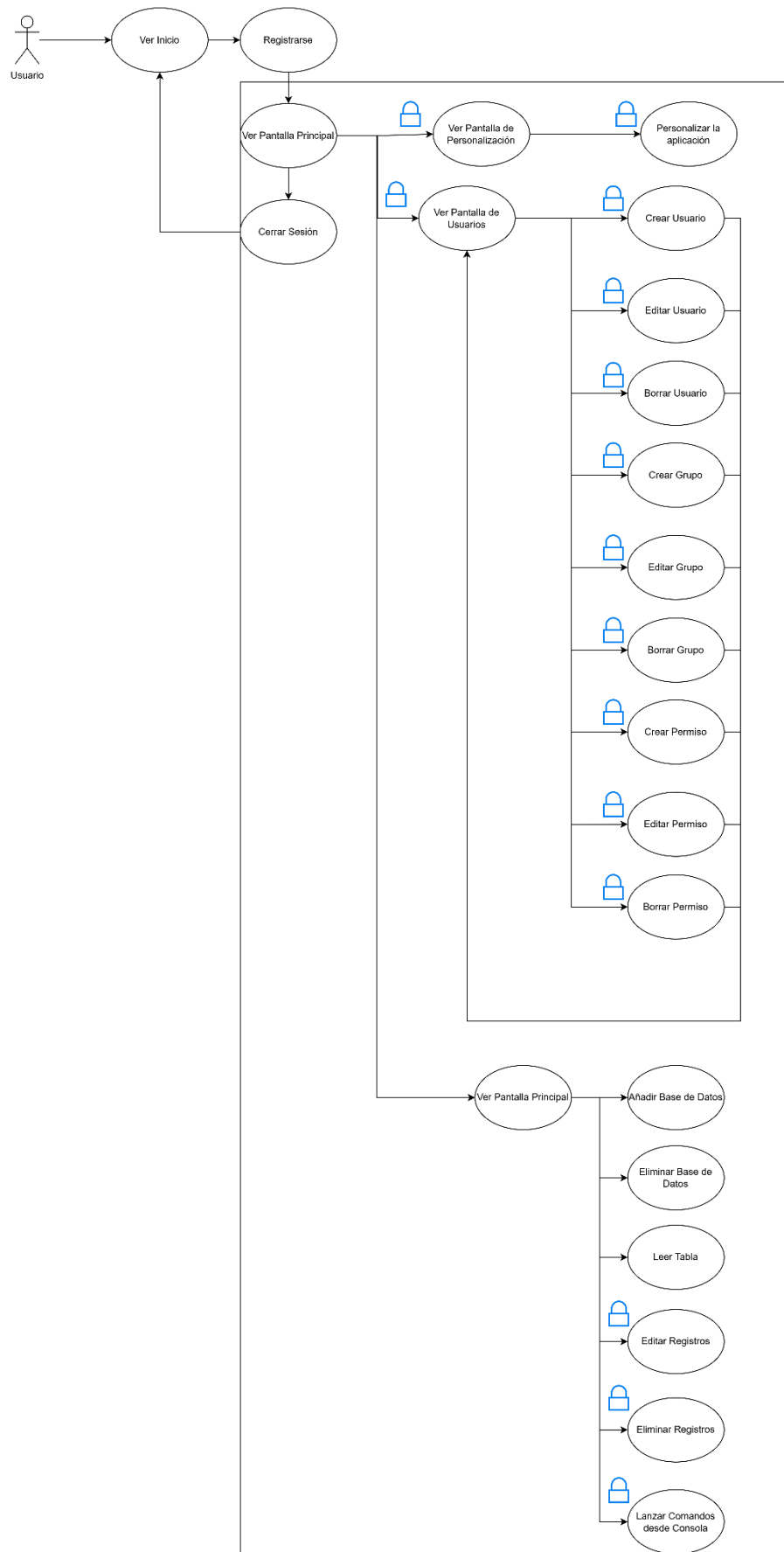


Figura 34: Diagrama de casos de uso

Anexo II: Maquetación del proyecto

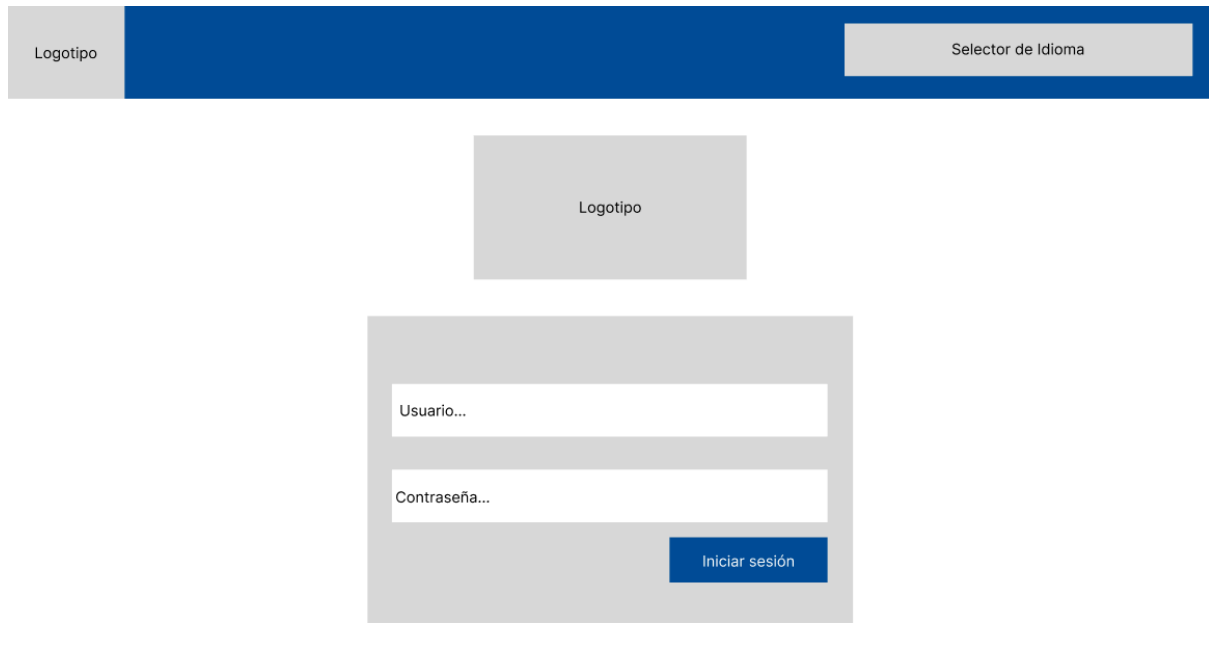


Figura 35: Maquetación original de la pantalla de login

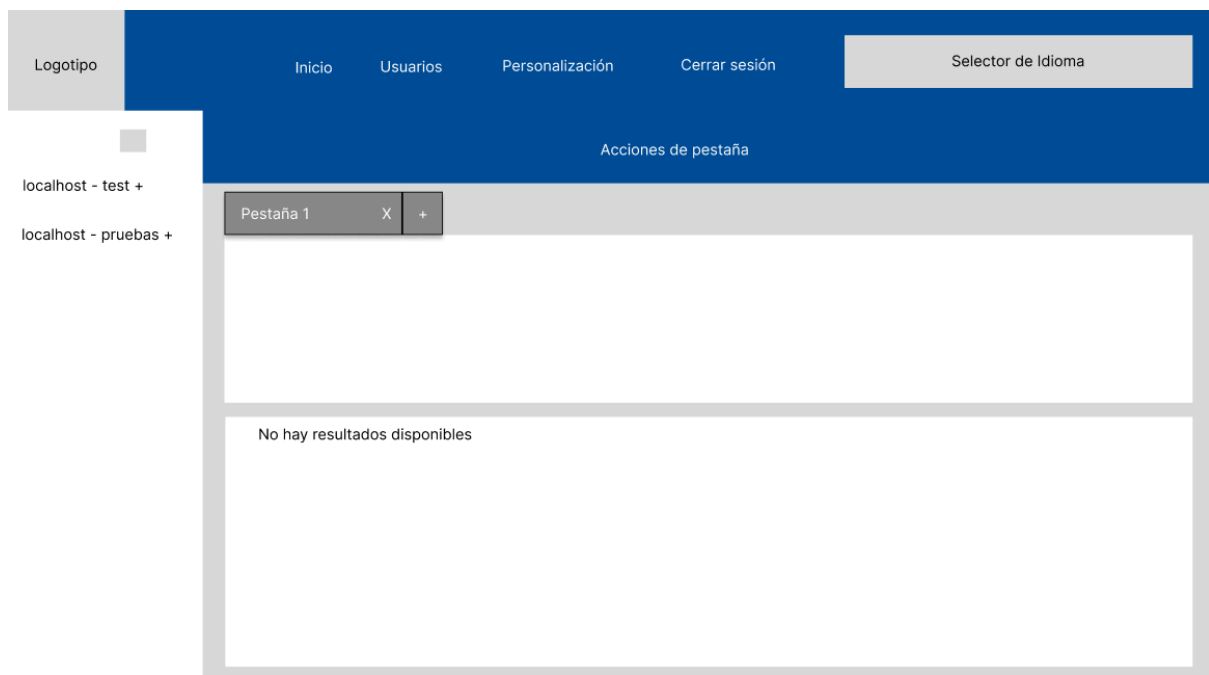


Figura 36: Maquetación original de la pantalla principal

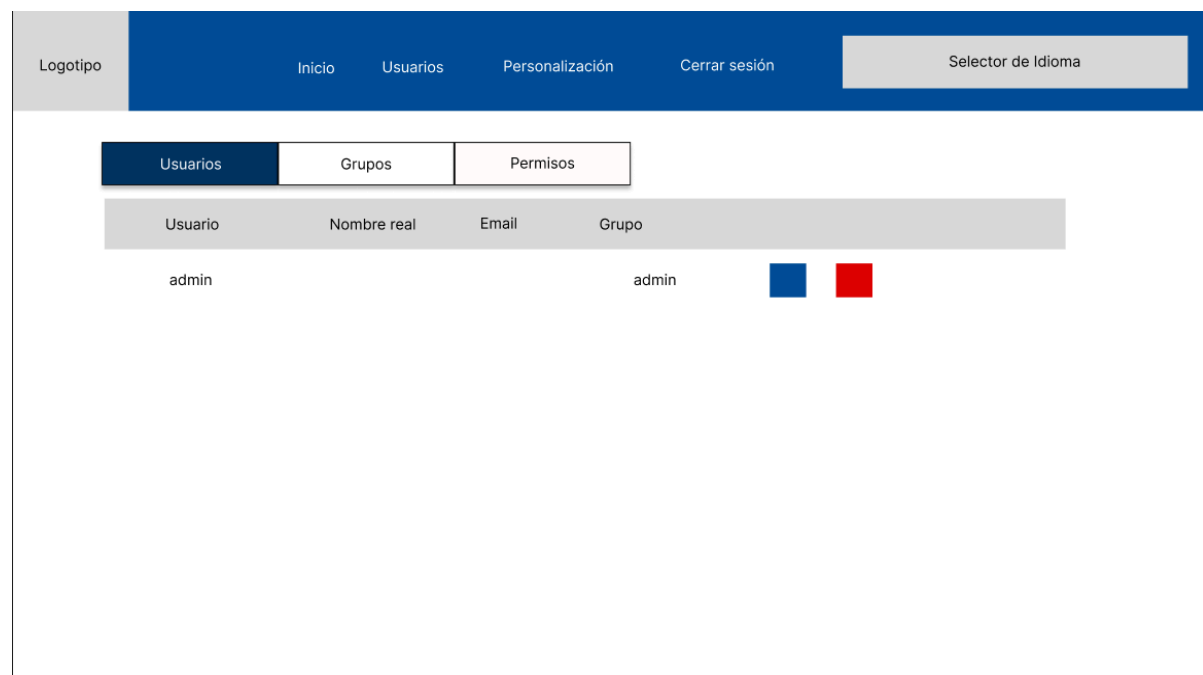


Figura 37: Maquetación original de la pantalla de gestión de usuarios

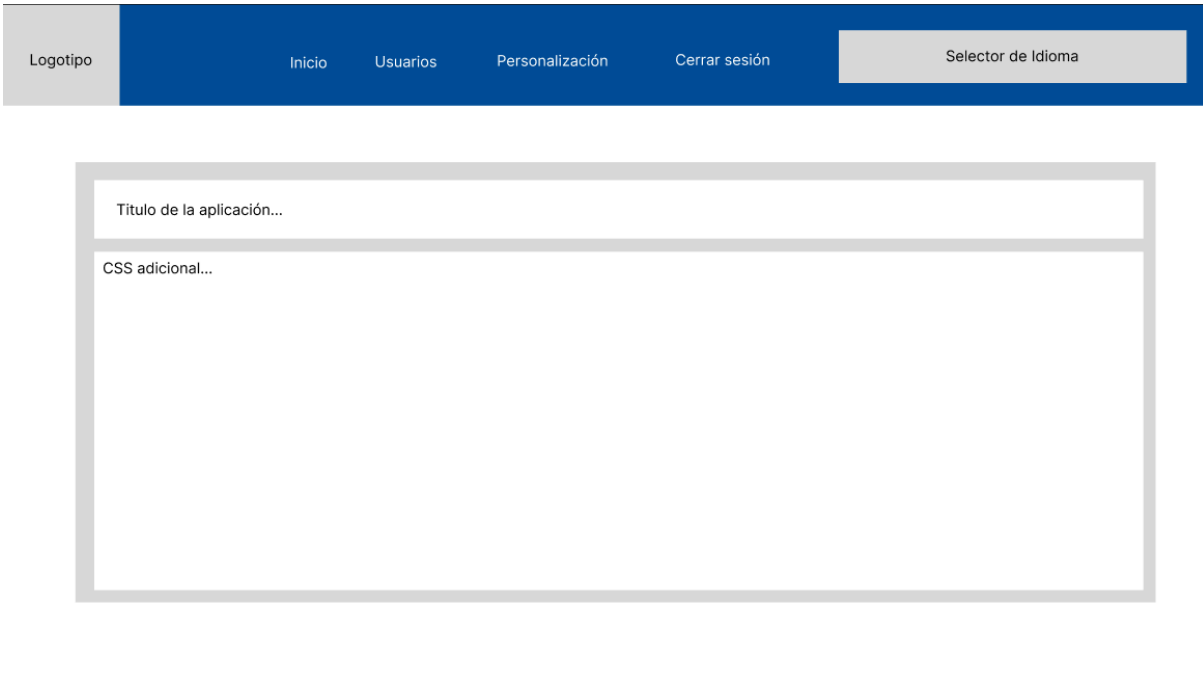


Figura 38: Maquetación original de la pantalla de personalización