

|  |
| --- |
| **Departamento: Tecnología** |
| **Responsables: Desarrollo Digital** |
| **Central Salamanca; España** |
| **Fecha: 29/08/2022** |

**Manual de Pruebas**

**– Directus 9 –**

**Page Builder**

**ÍNDICE**

[1. – INTRODUCCIÓN 2](#_Toc113281147)

[2. – PUNTOS DÉBILES 2](#_Toc113281148)

[3. – OBJETIVO 3](#_Toc113281149)

[4. – REQUISITOS 3](#_Toc113281150)

[5. – PRIMEROS PASOS 3](#_Toc113281151)

[5.1 – Instalación de Directus 3](#_Toc113281152)

[5.2 – Instalación de React 4](#_Toc113281153)

[5.3 – Instalación de extensiones 4](#_Toc113281154)

[5.4 – Instalación de dependencias 4](#_Toc113281155)

[6. – CASOS DE USO 5](#_Toc113281156)

[7. – CREACIÓN DEL MODELO DE DATOS 5](#_Toc113281157)

[7.1 – [CU-01] - Permisos 6](#_Toc113281158)

[8. – ADMINITRACIÓN DEL CONTENIDO 6](#_Toc113281159)

[8.1 – [CU-02] - Menús 6](#_Toc113281160)

[8.2 – [CU-03] - Contenido 7](#_Toc113281161)

[8.3 – [CU-04] - Secciones 8](#_Toc113281162)

[8.4 – [CU-05] – Páginas 8](#_Toc113281163)

[9. – BUG 9](#_Toc113281164)

# – INTRODUCCIÓN

Directus permite crear una página personalizada con un diseño libre, fuera de plantillas, en la que se gestiona el contenido de manera independiente a la interfaz. Por lo tanto, Directus permite administrar la información y consumirla en cualquier sitio en forma de API.

Lo que diferencia a Directus de cualquier CMS tradicional como Liferay, es que permite crear una aplicación con un diseño diferente y sin limitaciones en cuanto a herramientas y creatividad, ya que no está sujeto a los estrictos estándares de creación de themes y plantillas de estos CMS.

*Mientras un CMS tradicional combina el contenido y la capa de presentación, Directus se centra únicamente en el contenido.*

En la mayoría de departamentos dedicados al desarrollo web, el gestor de contenidos es la persona encargada de configurar el CMS y de maquetar el frontal. Con Directus, este perfil únicamente se centrará en la administración de esa información, sin tenerse que preocupar del diseño ni de la maquetación del prototipo del sitio web.

# – PUNTOS DÉBILES

Analizando Directus desde el punto de vista del rol de editor no todo son beneficios a la hora de usar esta arquitectura. Existen algunos inconvenientes a la hora de implementarla que es necesario tener en cuenta:

1. La persona encargada de administrar la información**[[1]](#footnote-1)** tendrá que adaptarse al nuevo CMS y enfrentarse a un nuevo paradigma de gestión de contenidos, es decir, tendrá que aprender a trabajar con la nueva estructura ‘pura’ de presentación y organización de la información.
2. No podrá obtener una vista previa de cómo se verá el contenido creado en las aplicaciones desde Directus para poder guiarse.
3. Este sistema fue desarrollado para que el proceso de creación, administración y modificación de la información fuese un proceso ágil, rápido y sencillo. Por el contrario, Directus no fue pensado para realizar personalizaciones complejas o actualizaciones del frontal del sitio, es decir, Directus no es la mejor opción si lo que se pretende es que el gestor de contenidos pueda modificar la capa de presentación de la aplicación.

# – OBJETIVO

La falta de plantillas, themes y funcionalidad en cuanto a la presentación del contenido, significa que hay que buscar tecnologías adicionales para servir como “cabeza” de este CMS. La integración de estas tecnologías que garanticen un óptimo resultado no es una tarea sencilla.

El objetivo de este desarrollo es proponer una solución alternativa que sea fácil de implementar y no rompa los principios de Directus. Esta solución consiste en extender la funcionalidad de Directus, en cuanto a la gestión del contenido, para que el GC pueda tener un impacto directo en la capa de presentación de la aplicación. De esta manera no solo podrá decidir qué información se visualizará en la aplicación, sino que también, con ciertas limitaciones, podrá decidir cómo y dónde se visualizará.

Se detallarán los pasos a seguir para crear un sencillo **Page Builder**.

# – REQUISITOS

* Se usará **Directus** como gestor de contenidos para la creación de la base de datos y de la API de conexión. La API que se usará será **GraphQL**
* Se usará **React** como Framework de frontend para creación de la interfaz de la aplicación.
* El gestor de base de datos[[2]](#footnote-2) de la aplicación creará el modelo de datos y dará los permisos oportunos para que el GC pueda gestionarlos de una manera eficiente.

Los permisos en este sentido serán:

* + Tendrá todos los permisos para la creación de nuevas páginas, secciones, contenidos y menús.
  + No tendrá permisos para la administración de las plantillas.
* Se limitará la estilización de contenidos desde Directus.
* Tanto los menús como las páginas se crearán dinámicamente desde Directus.

# – PRIMEROS PASOS

## – Instalación de Directus

* Crear el directorio del proyecto

mkdir <nombre del proyecto>

cd <nombre del proyecto>

* Instalar Directus 9

npm init directus-project <nombre del directorio de la api>

Durante el proceso de instalación hay que seleccionar la base de datos e introducir los datos de administración de la herramienta.

* Lanzar Directus 9

cd <ruta del directorio de la api>

npx directus start

Servidor lanzado en <http://localhost:8055>

## – Instalación de React

* Instalar React

npm create vite@latest <nombre del directorio de la app>

Durante el proceso de instalación hay que seleccionar **React** como framework del proyecto.

* Lanzar React

cd <ruta del directorio de la app>

npm install

npm run dev

Servidor lanzado en <http://127.0.0.1:5173>

## – Instalación de extensiones

Editor.Js es una biblioteca para editar contenido de manera visual/WYSIWYG basado en bloques de contenido y en add-ons para extender el funcionamiento del mismo plugin. Además, devuelve los datos en formato JSON limpio y legible, que se representará fácilmente como HTML limpio.

* Instalar Editor

cd <ruta del directorio de la api>

npm install directus-extension-editorjs

## – Instalación de dependencias

* Instalar React Query como manejador de estados en el servidor y optimizador del rendimiento de las solicitudes a la API:

cd <ruta del directorio de la app>

npm install @tanstack/react-query

* Instalar React Router para gestionar las rutas de nuestra aplicación:

cd <ruta del directorio de la app>

npm install react-router-dom

* Instalar Tailwind como framework CSS para estilizar prototipos y casos de testing al permitir un desarrollo ágil y optimizado:

cd <ruta del directorio de la app>

npm install -D tailwindcss postcss autoprefixer

npx tailwindcss init -p

# – CASOS DE USO

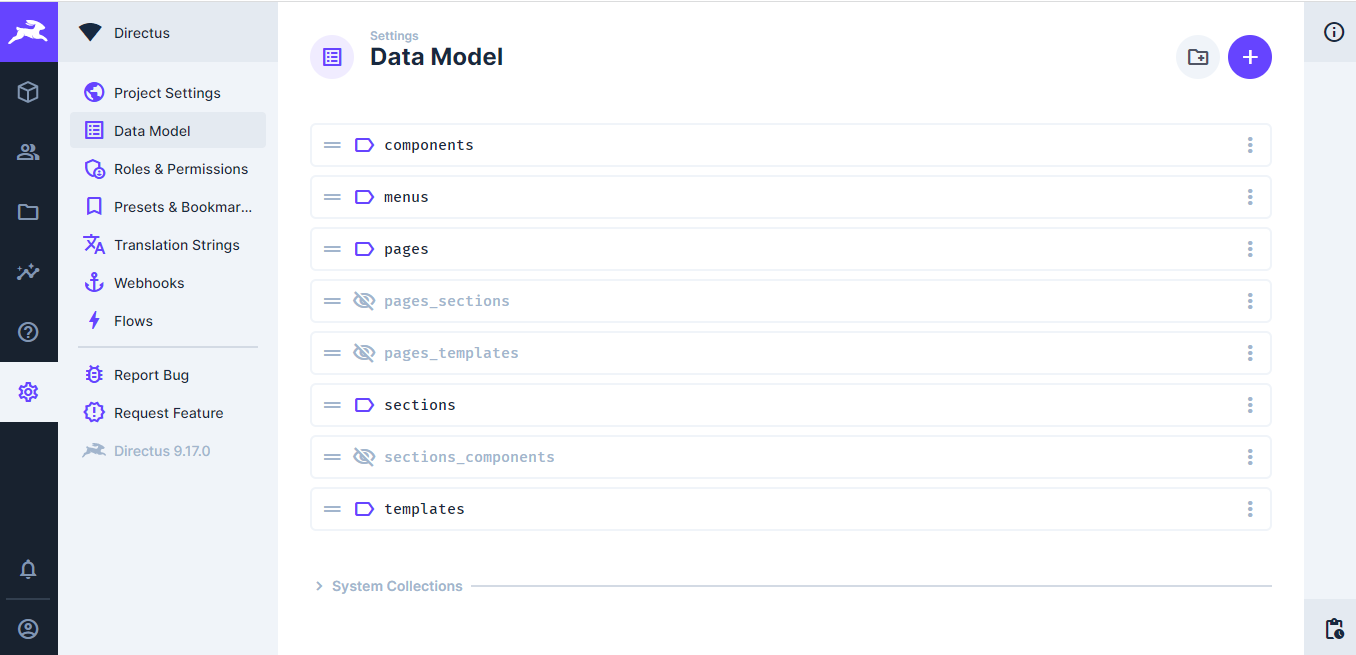
1. [CU-01] El GC tendrá todos los permisos para la administración de las páginas, secciones, contenidos y menús, pero únicamente tendrá permisos de lectura para poder visualizar las plantillas disponibles.
2. [CU-02] El GC administrará los menús de la aplicación de tal manera que una vez creados puedan visualizarse.
3. [CU-03] El GC administrará el contenido, pero no podrá darle estilos.
4. [CU-04] El GC podrá decidir qué contenido se publicará en cada sección.
5. [CU-05] El GC podrá decidir qué secciones se incluirán en cada página y además podrá aplicarle una plantilla para determinar su estructura.

# – CREACIÓN DEL MODELO DE DATOS

A través de la propia interfaz que nos proporciona Directus, el GBD creará las distintas colecciones necesarias para el desarrollo de la aplicación**.** Es aquí donde se almacenará la información que consumirá la capa de presentación por medio de su API. Estas colecciones por lo tanto serán tablas en la base de datos de Directus. Para la creación de las colecciones seguiremos estos pasos:

* Accedemos a las colecciones: **Settings > Data model**
* Añadimos una nueva colección y la nombramos
* Una vez creadas las tablas, añadiremos los campos personalizados correspondientes a cada una de ellas:
  + Templates: key (*string*) y value (*string*)
  + Pages: title (*string*), url (*slug*), section\_builder (*many-to-many*) y template\_builder (*many-to-many*)
  + Sections: title (*string*) y content\_builder (*many-to-many*)
  + Components: title (*string*) y content (*wysiwyg*)
  + Menus: title (*string*) y menú\_items (*repeater: label + url*)

[Ver vídeo de creación del modelo de datos de la colección menús](https://www.loom.com/share/e7047cbcaf894648b7ba1562c374f53b)



Teniendo el modelo de datos creado, el siguiente paso será crear el rol de Editor en la aplicación. Esto ser hará en:

**User directory > Create item**

## – [CU-01] - Permisos

Lo último que hará el GBD, será definir los permisos de esas colecciones que se acaban de añadir. Para que el GC pueda leer, actualizar, crear y borrar cualquier campo de las colecciones: *menus, pages, sections y components*, es necesario que se cambien esos permisos para que sean accesibles por ese rol. Para la colección *templates* únicamente se otorgarán permisos de lectura.

**Settings > Roles & Permissions > Editor**

# – ADMINITRACIÓN DEL CONTENIDO

Teniendo ya creado y configurado nuestro modelo de datos, el GC podrá gestionar la información que se mostrará en la aplicación.

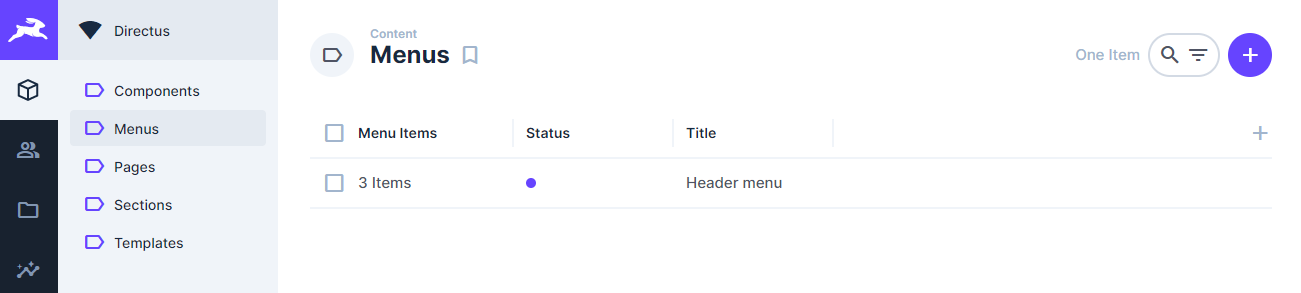
## – [CU-02] - Menús

El proceso para la creación de menús de la aplicación será el siguiente:

* Partiendo de los prototipos creados por el equipo de diseño, previamente validados antes del desarrollo, se determinarán los menús que formarán parte de la interfaz de usuario.
* Desde el frontal de nuestra aplicación se implementará el código necesario para que una vez que el GC cree un nuevo menú en Directus, este se publique automáticamente.
* El GC simplemente tendrá que acceder a la siguiente ruta para poder crearlos:

**Content > Menus> Create item**

* Se le dará un título significativo y para cada apartado se creará un nuevo item rellenado los campos *label* y *URL* del mismo.
* Si este apartado dispone de un submenú, además de estos dos campos anteriores, habría que añadir un campo *repetidor* que contendrá una lista de ítems formada por los campos label y URL igualmente. En el submenú se crearán tantos ítems como subapartados contenga el menú.
* Una vez creado el menú con sus apartados y subapartados, si fueran necesarios, estos se publicarán automáticamente en la aplicación.



[Vídeo Creación de un menú de navegación que contiene submenús](https://www.loom.com/share/d0f77e09833144ebbd54ddc6395b2c9e)

## – [CU-03] - Contenido

Cabe destacar que para la realización de estas pruebas el contenido que se va a publicar en la aplicación es un contenido genérico y por lo tanto solamente habrá un contenedor por defecto que contendrá la información. Cuando se implementen soluciones reales, este contenido se deberá componetizar, es decir, se deberá segmentar el contenido en unidades pequeñas que irán alojadas cada una en una colección (botones, cards, sliders, formularios…).

Antes de crear la colección correspondiente para el contenido genérico de la aplicación se analizarán los tres escenarios posibles para su desarrollo:

* Se usará el campo WYSIWYG para añadir el contenido
* Se creará un campo por cada tipo de contenido
* Se usará la extensión Editorjs para añadir el contenido

**Uso de WYSIWYG**

Usar este tipo de campo para añadir contenido a la aplicación es una buena estrategia por la sencillez y la rapidez del proceso. Además, nos da la opción de añadir multitud de tipos de contenido. El inconveniente de este campo es que el GC puede editar los estilos del contenido y según los requisitos no debería. Además, el formato en el que se exportan estos datos no es el más adecuado para posteriormente consumirlos a través de una API.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Facilidad de uso | Versatilidad | Formato exportar | Requisitos | Resultado |
| **POSITIVO** | **POSITIVO** | **NEGATIVO** | **NEGATIVO** | **DESCARTADO** |

**Creación de un campo por cada tipo de contenido**

En este escenario se cumplen los requisitos del proyecto puesto que el GC no podrá editar los estilos del contenido. Además, cumple con la filosofía de componetizar cada tipo de contenido. El inconveniente es que, si tenemos que introducir mucha información en la aplicación, se antoja una tarea bastante engorrosa tanto para GC como para el GBD que tiene que configurar el modelo de datos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Facilidad de uso | Versatilidad | Formato exportar | Requisitos | Resultado |
| **NEGATIVO** | **NEGATIVO** | **POSITIVO** | **POSITIVO** | **DESCARTADO** |

**Uso de Editorjs**

Editorjs es una biblioteca para editar contenido de código abierto que facilita la edición y la personalización de bloques de contenido. Además, devuelve los datos en formato JSON limpio y legible. Los datos limpios son más rápidos de cargar, más fáciles de depurar y mejoran el rendimiento general del proyecto.

Con esta extensión se puede agregar multitud de tipo de contenido como lo podíamos hacer con WYSIWYG usando el editor en línea, pero sin la opción de poder estilizarlo. Si queremos añadir más tipo de contenido y hacer uso de un editor avanzado se podría extender igualmente.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Facilidad de uso | Versatilidad | Formato exportar | Requisitos | Resultado |
| **POSITIVO** | **POSITIVO** | **POSITIVO** | **POSITIVO** | **ELEGIDO** |

## – [CU-04] - Secciones

Una vez que el GC haya añadido el contenido puramente dicho de la aplicación, haciendo uso de la extensión Editorjs, es momento de que agrupe dicha información en secciones. Para ello hará uso de la interfaz de Directus.

Por cada región de la aplicación se tendrá que crear una colección de tipo *section*, y en ella determinar qué contenido irá alojado en cada una de ellas.

## – [CU-05] – Páginas

De la misma manera que se crearon las colecciones de tipo *section*, se crearán las colecciones de tipo *page* que contendrán las diferentes secciones prefijadas en el prototipo de la aplicación. De esta manera, la estructura de cada página será mucho más limpia y coherente.

Este tipo de colección tiene una peculiaridad con respecto al tipo de colección *section* y es que el GC podrá determinar qué plantilla usar para que su aspecto se ajuste al diseño de la aplicación. Estas plantillas se han implementado previamente en el frontal.

Una vez perfiladas todas las páginas de la aplicación, solamente quedaría por cumplir el último requisito del proyecto: **las páginas se crearán dinámicamente desde Directus**.

Para el cumplimiento de este requisito se han seguido los siguientes pasos en el desarrollo del mismo:

* Se han configurado ruta raíz que agrupará el resto de rutas.
* Como no sabemos de antemano qué rutas creará el GC, y como se tienen que generar dinámicamente cuando esta persona las cree a través de Directus, se ha creado una ruta por defecto para todas ellas.
* En función del slug definido por el GC se creará la **url dinámicamente** para cada página.
* Al acceder a dicha url, la aplicación renderizará el componente de tipo funcional y pintará la información correspondiente a cada página.

Inicialmente la llamada a la API con React Query se hacía filtrando por la página que queríamos mostrar, pero existe un problema con React Query cuando queremos hacer un refetch (nueva llamada a la API) al cambiar de ruta, para que nos renderice el componente y nos muestre la nueva información. React Query no nos proporciona un método propio para hacer un refetch cuando usamos rutas dinámicas que renderizan un mismo componente.

Para solucionarlo, al realizar la llamada a la API en vez de filtrar por la página que queremos mostrar, en función de la url, tendremos que consumir la API sin hacer el filtrado, para traernos toda la información y luego mediante un manejador de estados hacer nosotros el filtrado.

La solución óptima será que, antes de que el GC cree las páginas de la aplicación, se definan las distintas rutas que va a tener la aplicación y en función de este requisito, el GC asocie cada página a cada una ellas.

Por requisitos de este proyecto en concreto se adoptará la solución inicial para que el GC tenga plena independencia. En proyectos reales, el GC no tendrá completa independencia para crear páginas, ya que se deberá seguir estrictamente los flujos de trabajo dentro del ciclo de vida de un proyecto, es decir, podrá crearlas y asignarles una ruta, secciones y contenidos, pero esa página no aparecerá publicada hasta que el desarrollador frontend haya definido previamente la ruta asociada a esa página.

# – BUG

Para mejorar la experiencia de usuario del GC, se ha modificado el modelo de datos.

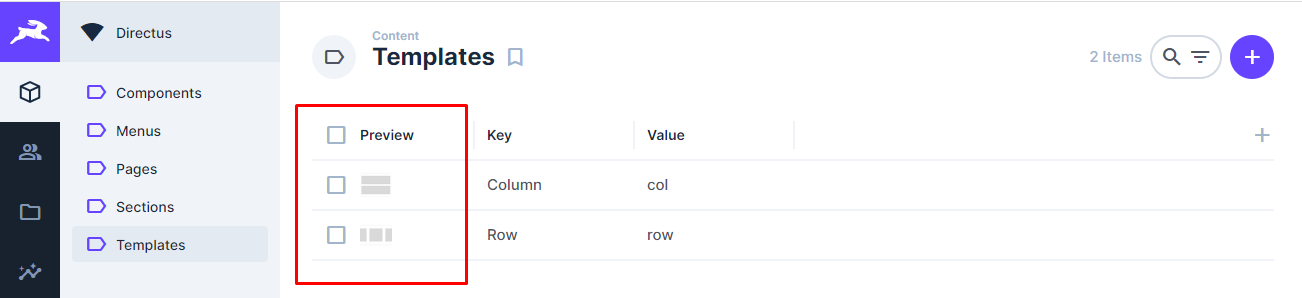
El GBD tendrá que añadir un nuevo campo a la colección *templates* para que el GC pueda previsualizar la plantilla antes de crearla y así de un vistazo pueda identificarla más rápidamente y evitar confusiones y minimizar los errores.

Con la implementación de este nuevo requisito hemos detectado un bug en Directus que ya está reportado:

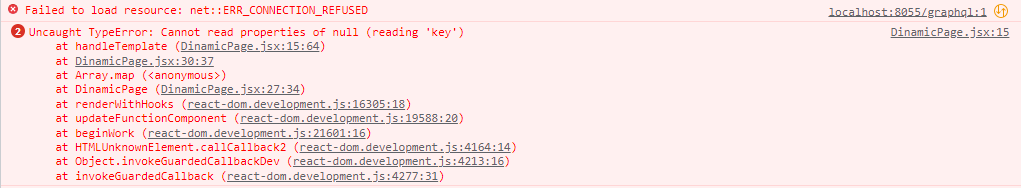
[Bug reportado a Directus](https://github.com/directus/directus/issues/15398)

Este error se produce de la siguiente manera:

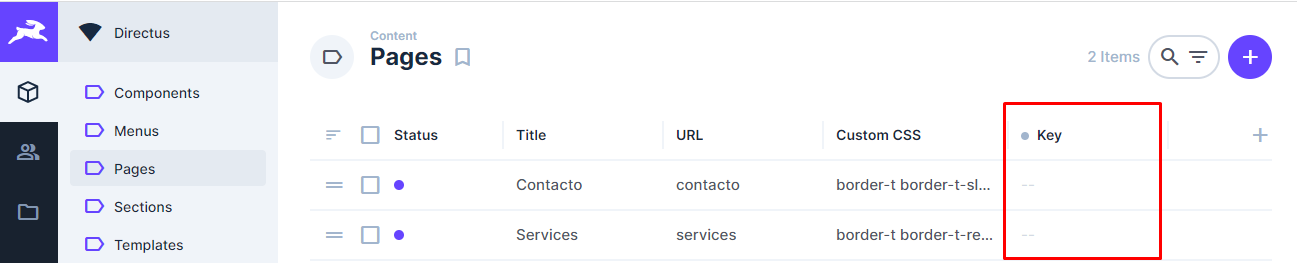
* El GBD creará un nuevo campo imagen en la colección *templates*.
* El GBD añadirá una imagen representativa a cada registro de esta tabla.



Una vez creado este campo, observamos que al cargar la aplicación nos da un error:



Analizando el error observamos que viene producido porque tras crear el nuevo campo, Directus ha eliminado las relaciones entre las tablas *pages y templates*, y ese campo que las enlace aparece vacío:



Se han vuelto a introducir manualmente los datos de correspondencia y la página ya funciona correctamente. El error ha sido reportado y corregido por el equipo técnico de Directus.

1. **A partir de ahora al gestor de contenidos se le nombrará como GC** [↑](#footnote-ref-1)
2. **A partir de ahora al gestor de bases de datos se le nombrará como GBD** [↑](#footnote-ref-2)