ChatGPT

Guía para crear páginas visuales en NocoBase mediante API

Introducción y Conceptos Clave

NocoBase permite construir **páginas visuales** (vistas de colección, formularios, páginas personalizadas, etc.) que actúan como contenedores de **bloques** de interfaz. Estas páginas se organizan en el menú de la app (rutas de la UI) y pueden contener bloques como tablas (listas de registros), formularios, gráficos, etc. En NocoBase existen tres tipos de ítems de menú (rutas) predeterminados: **Grupo**, **Página** y **Enlace externo** [1].

- **Grupo (Group)**: Carpeta o sección para agrupar varias páginas en el menú. Puede contener subrutas o páginas.
- **Página (Page)**: Página interna de NocoBase (un lienzo donde colocar bloques).
- **Enlace (Link)**: Enlace externo o redirección a una URL específica.

Internamente, NocoBase almacena las rutas de páginas en tablas especiales (`desktopRoutes` para la interfaz web de escritorio y `mobileRoutes` para móvil) [2]. Cada página (tipo "page") tiene asociado un esquema UI (un **layout** vacío por defecto) donde se ubicarán los bloques. Los **bloques** son componentes de interfaz (tabla, formulario, gráfico, etc.) que se pueden añadir a páginas, diálogos o paneles [3]. Por ejemplo, un bloque de tabla listado muestra datos de una colección (tabla de la base de datos), y un bloque de formulario permite capturar o editar datos.

La API REST de NocoBase sigue un estilo de **Resource & Action**. Para crear, leer o modificar recursos se utilizan endpoints con acciones. Por ejemplo, para crear un recurso se usa `POST /api/<recurso>:create` [4]. Es necesario autenticarse con un token API válido en **cada petición**, usando la cabecera HTTP `Authorization: Bearer <API_TOKEN>` [5]. Además, incluir `Content-Type: application/json` en las peticiones `POST/PUT` con cuerpo JSON.

A continuación, se detalla paso a paso cómo construir páginas visuales mediante la API, con ejemplos en JavaScript/TypeScript. Supondremos que ya tienes colecciones (tablas de datos) creadas y un token API de administrador.

1. Crear un grupo de menú (opcional)

Para organizar las páginas y hacerlas visibles en la interfaz, es recomendable crear primero un **grupo de menú**. Esto aparecerá como un ítem agrupador en el menú de NocoBase (por ejemplo, una sección en la barra lateral o superior). Si no creas un grupo, las páginas podrían quedar sin mostrarse en el menú (las páginas raíz suelen corresponder a rutas bajo `/admin` por defecto).

Endpoint: `POST /api/desktopRoutes:create`
Cuerpo JSON mínimo: indicar el `type` como `"group"` y un `title` (nombre visible). También puedes incluir `icon` (icono) o `sort` (orden) si lo deseas. Si este grupo es secundario dentro de otro, especifica `parentId` con el ID del grupo padre.

Ejemplo de creación de un grupo de menú llamado "CRM":

```
POST /api/desktopRoutes:create
Authorization: Bearer <API_TOKEN>
Content-Type: application/json

{
  "type": "group",
  "title": "CRM"
}
```

Si la petición es exitosa, la respuesta incluirá los datos del grupo creado, por ejemplo un identificador único. Un script de integración muestra este paso devolviendo un `groupId` [6] . Puedes usar ese `id` (o `uid`) en pasos posteriores para anidar páginas debajo de este grupo.

2. Crear una página de colección asociada a una tabla

El siguiente paso es crear la página visual en sí (tipo **page**), asociándola al grupo creado (u otro grupo existente). Esta página representará, por ejemplo, la lista de registros de una colección de datos.

Endpoint: `POST /api/desktopRoutes:create` (el mismo de rutas, porque la página es una ruta) **Cuerpo JSON:** Debe contener al menos `"type": "page"`, un `title` para la página, y `parentId` apuntando al grupo donde vivirá. También es conveniente especificar la ruta o nombre interno: - `name`: nombre interno de la ruta (único en la jerarquía). Suele usarse una notación jerárquica (`<padre>.<hijo>`) o un identificador simple si se anida mediante `parentId`. Por ejemplo `"name": "crm.leads"` podría representar una página "leads" bajo el grupo "crm". - `path`: ruta de URL para la página. Por ejemplo, `"/leads"` (NocoBase la montará posiblemente bajo `/admin` u otro prefijo según el layout). Si no se indica, NocoBase puede generarla automáticamente en base al nombre.

Ejemplo de cuerpo para crear una página "Leads" dentro del grupo "CRM" (asumiendo `parentId` obtenido arriba):

```
{
  "type": "page",
  "title": "Leads",
  "parentId": "<ID_del_grupo_CRM>",
  "name": "leads",
  "path": "/leads"
}
```

La respuesta de esta petición incluirá la información de la nueva página. En particular, debe proveer un identificador del esquema UI asociado a la página. NocoBase generalmente crea un **contenedor base (grid)** vacío dentro de la nueva página [7] . En la respuesta JSON esto puede aparecer como un `uid` de grid o un objeto `schema` con su `id`. Por ejemplo, un script devuelve `gridUid: "xyz789"` para referirse al contenedor principal de la página [8] . Tomaremos nota de este identificador de contenedor, ya que lo usaremos para agregar bloques dentro de la página.

> **Nota:** Hasta este punto, en la interfaz de NocoBase debería aparecer la nueva página "Leads" bajo el grupo de menú "CRM". Si refrescas la UI como administrador, verás el menú actualizado con la sección y página creada. Si no aparece, verifica que la página tenga un título y esté correctamente asociada a un grupo visible, y que tu usuario tenga permisos para verla (por defecto el admin ve todo).

3. Añadir un bloque de tabla (vista de colección) a la página

Una vez creada la página, ésta estará vacía. Debemos agregar un bloque de tipo **Tabla** que muestre los datos de la colección deseada. Usaremos el identificador de contenedor (`gridUid`) de la página como `parent` para el nuevo bloque.

Endpoint: `POST /api/uiSchemas:create` (endpoint genérico para crear nodos del esquema UI).
Cuerpo JSON: indicaremos el tipo de componente de interfaz y su configuración básica: - Un campo que indique el tipo de bloque/component (por ejemplo `componentName` o `x-component`). Para una tabla de datos suele ser `"Table"` (el bloque de tabla estándar de NocoBase). - Referencia a la **colección** cuyos datos mostrará: típicamente con la clave `collection` o `collectionName` y el nombre de la tabla (ej: `"collection": "leads"`). - `parentId` o `parentUid`: el identificador del contenedor padre donde insertar este bloque (el grid de la página obtenido antes).

Ejemplo de cuerpo para agregar una tabla vinculada a la colección "leads":

```
{
  "componentName": "Table",
  "collection": "leads",
  "parentUid": "<UID_contenedor_pagina>"
}
```

Al ejecutar esta petición, la respuesta contendrá los datos del bloque de tabla creado. Notablemente, la respuesta suele incluir: - Un identificador único del bloque de tabla (por ejemplo `tableBlockUid`). - Un identificador para la **columna de acciones** asociada al bloque (por ejemplo `actionsColumnUid`) [9]. NocoBase normalmente crea automáticamente una columna reservada para botones de acción (ej. ver, editar, borrar) al final de la tabla, y este ID nos permite referenciarla para agregar acciones específicas por fila más adelante.

En la interfaz, si refrescas en modo editor, ya deberías ver un bloque de tabla vacío en la página "Leads". Sin embargo, por defecto puede no mostrar columnas de datos hasta que las agreguemos en el esquema (siguiente paso).

4. Agregar columnas de campos a la tabla

Ahora popularemos la tabla con columnas correspondientes a los campos de la colección que queremos mostrar. Cada columna en la tabla es en sí un sub-bloque (generalmente de tipo **TableColumn** en el esquema UI) hijo del bloque tabla.

Endpoint: `POST /api/uiSchemas:create` (creación de nodo UI, similar al anterior).
Cuerpo JSON: debemos especificar: - El tipo de componente de columna (por ejemplo, `componentName`: `"TableColumn"`). - Referencia al campo de la colección que mostrará esa

columna. Esto suele indicarse con una propiedad como `"field"` o `"fieldPath"` que contenga el nombre del campo en la colección (ejemplo: `"field": "company_name"`). Algunos esquemas usan `"dataIndex"` o similar. Para mantenerlo simple, usaremos `fieldPath` con el nombre del campo. - El `parentUid` del bloque de tabla al que se añade la columna (el `tableBlockUid` obtenido antes).

Ejemplo para agregar dos columnas a la tabla "Leads", mostrando los campos "company_name" y "email":

```
{
  "componentName": "TableColumn",
  "fieldPath": "company_name",
  "parentUid": "<UID_bloque_tabla>"
}
```

```
{
  "componentName": "TableColumn",
  "fieldPath": "email",
  "parentUid": "<UID_bloque_tabla>"
}
```

Cada petición añadirá la columna correspondiente. Tras esto, la tabla de la página "Leads" ya tendrá visibles esas columnas con los datos de la colección. (NocoBase también permite configurar propiedades de columna adicionales como anchura, formato, etc., pero mediante API básica enviar solo el campo asegura que se muestre el valor del campo).

En el ejemplo de código de integración se ilustran estas llamadas repetidas para cada campo deseado [10].

5. Añadir acciones (botones) para crear, ver o editar registros

Para que la página sea realmente útil, es necesario habilitar acciones como **"Agregar nuevo registro"** o **"Ver/editar detalles"** por registro. En NocoBase, las acciones de interfaz (botones) también se configuran vía bloques/ esquema UI.

Hay dos lugares típicos para acciones en una vista de tabla: - **Acciones globales de la tabla:** botones en la cabecera del bloque tabla (por ejemplo, un botón *Agregar nuevo* que abra un formulario para crear un registro). - **Acciones por fila:** botones dentro de la columna de acciones de cada registro (por ejemplo, *Ver detalles*, *Editar*, *Eliminar* en cada fila).

NocoBase trae **acciones predefinidas** que se pueden configurar. Por ejemplo, la acción `"addNew"` abre un formulario para añadir un nuevo registro a la colección actual, y la acción `"view"` sirve para ver detalles de un registro (generalmente abriendo un popup con un formulario de solo lectura o editable) [11]. También existen acciones como `"edit"`, `"delete"`, `"export"`, etc., disponibles según los plugins activados.

Para añadir una acción mediante API, nuevamente usaremos la creación de un nodo UI:

Endpoint: `POST /api/uiSchemas:create`
Cuerpo JSON: Dependiendo de la acción: - Indicar un tipo de componente de acción. Podría ser genérico (p.ej. `componentName: "Action"`) con propiedades, o directamente un tipo como `"ActionBarItem"`. La documentación oficial sugiere usar inicializadores de esquema para acciones, pero vía API directamente podemos usar las propiedades. - Especificar el tipo de acción deseada, a veces con una propiedad como `"actionType"` o similar. - Especificar la colección objetivo si aplica (en acciones CRUD típicas, la colección es la misma de la tabla). - El `parentUid` donde irá el botón: - Para un botón global (ej. *Agregar nuevo*), el parent es el bloque tabla (`tableBlockUid`), ya que se coloca en la barra de herramientas de la tabla. - Para un botón por fila (ej. *Ver*), el parent es la **columna de acciones** de la tabla (el `actionsColumnUid` obtenido al crear la tabla).

Ejemplos: 1. **Botón "Agregar nuevo":** Añadir un botón que permita crear un nuevo registro de "leads". Se ubica en la cabecera de la tabla.

```
{
  "componentName": "Action",
  "actionType": "addNew",
  "collection": "leads",
  "parentUid": "<UID_bloque_tabla>"
}
```

Esto típicamente agrega un botón (por defecto con ícono "+") en la tabla. Al hacer clic, NocoBase abrirá un formulario de creación (en un modal o drawer) para la colección indicada. No hace falta que creemos manualmente la página de formulario; la acción *addNew* utiliza un formulario emergente generado automáticamente por el sistema para la colección [12]. *Nota:* Asegúrate de que la colección "leads" tenga permisos de escritura para tu rol, de lo contrario el formulario podría no guardar.

1. **Botón "Ver detalles":** Añadir un botón de acción por cada fila que abra la vista de detalle/edición del registro.

   ```
   {
     "componentName": "Action",
     "actionType": "view",
     "collection": "leads",
     "parentUid": "<UID_columna_acciones>",
     "isRowAction": true
   }
   ```

   Con `isRowAction: true` indicamos que esta acción se aplica a cada fila, usando el contexto del registro de esa fila. Por defecto, la acción *view* abrirá un popup (ej. un drawer) mostrando un formulario con los detalles del registro seleccionado [11]. Según la configuración, ese formulario puede permitir edición inmediata. Si se quisiera un botón *Editar* separado, se podría usar `"actionType": "update"` (o en versiones recientes, `"edit"`), que de forma similar abre un formulario en modo edición.

   Tip: NocoBase trae por defecto acciones de **duplicar**, **eliminar**, etc., que se pueden añadir de forma análoga especificando el `actionType` correspondiente (`duplicate`, `destroy`, etc.) y `isRowAction: true` bajo la columna de acciones. También existen

acciones para exportar/importar registros desde la cabecera de la tabla, etc., disponibles si los plugins pertinentes están activos.

Tras agregar las acciones deseadas, la página "Leads" estará completa: mostrará la tabla con los registros existentes, incluirá un botón para crear nuevos y botones en cada fila para ver/editar (y potencialmente borrar) registros. Todo ello replicando lo que se lograría configurando manualmente la página en la UI de NocoBase.

6. Confirmar cambios en la interfaz y posibles consideraciones

Una vez hechas las peticiones anteriores, los cambios en el backend deberían reflejarse en la interfaz web de NocoBase:

- **Refrescar la UI:** Es posible que necesites recargar la página del navegador (o limpiar caché de configuración) para ver las nuevas páginas y bloques añadidos. NocoBase tiene un modo de **Editor de UI** que actualiza la configuración visual; si estabas en modo edición, podrías ver aparecer los elementos casi inmediatamente, pero en modo normal puede requerir refresh.
- **Permisos:** Por defecto, las páginas creadas heredarán la visibilidad según los permisos de la colección y rol. Asegúrate de que el rol que usas (p.ej. Admin) tenga acceso a la colección vinculada, de lo contrario la tabla podría aparecer vacía o con errores. También, si utilizas roles personalizados, verifica en **Settings > Roles & Permissions** que la nueva página (ruta) no requiera ajustes adicionales de permiso.

- **Estructura JSON exacta:** Ten en cuenta que la estructura JSON puede variar ligeramente entre versiones de NocoBase. Los ejemplos anteriores asumen NocoBase 1.6+ o 1.7 (donde `desktopRoutes` y `uiSchemas` están separados). En versiones más nuevas (2.0 beta), los campos podrían llamarse ligeramente distinto. Consulta la documentación oficial o el plugin de documentación API (Swagger) de tu instancia para confirmar las propiedades aceptadas [13]. Por ejemplo, algunos campos podrían ser `x-component` en lugar de `componentName` en ciertos contextos de UI Schema.

- **Errores comunes:** Si al hacer las peticiones recibes errores 4XX/500:

- Revisa que la URL y acción sean correctas (`.../desktopRoutes:create`, `.../uiSchemas:create`, etc. **incluyendo** el sufijo de acción). Un error típico es usar mal el formato Resource:Action.
- Asegúrate de incluir `Authorization` y usar un token válido. Sin este, la API retorna 401.
- Si ves un error de `null value in column "id" of relation "desktopRoutes"`, puede indicar un bug de la versión o que falta un campo requerido en la creación de la ruta (por ejemplo, no enviar `type` correcto). Revisa que envías `type: "page"` o `"group"` según corresponda.
- Un error de *"value ... out of range for type integer"* al crear rutas puede deberse a IDs muy grandes (NocoBase usa identificadores numéricos grandes). En general, debes pasar los `parentId` exactamente como los recibes (numérico o string). No intentes modificar el formato de ID devuelto por la API.
- Si las páginas se crean pero no aparecen en la interfaz, verifica el campo `parentId`. Una página de nivel superior podría necesitar `parentId` de alguna ruta padre existente (por ejemplo, la ruta `admin` principal). Crear páginas huérfanas sin parent podría hacer que no se muestren. Lo más sencillo es siempre asignarlas a un grupo o a la ruta `admin`. *(Por ejemplo, podrías usar* `parentId` *de la ruta* `"admin"` *por defecto para una página top-level. La ruta*

`admin` *suele tener id 1 o 10001, pero es mejor buscarla con GET* `/api/desktopRoutes` *filtrando por name).*

- Consulta los foros oficiales de NocoBase; varios usuarios comparten soluciones a errores comunes al manipular páginas por API.

A continuación se proporcionan ejemplos en código JavaScript y TypeScript que integran todos estos pasos de forma secuencial.

Ejemplo paso a paso en JavaScript (Node.js o navegador)

En este ejemplo usaremos `fetch` (disponible en navegadores o en Node 18+) para realizar las peticiones a la API. Asegúrate de reemplazar `<TU_TOKEN_API>` y la URL base por los de tu instancia:

```
const BASE_URL = "http://<tu-servidor-nocobase>";  // sin la barra final
const API_TOKEN = "<TU_TOKEN_API>";

// Helper para hacer peticiones POST con fetch
async function post(endpoint, body) {
  const res = await fetch(`${BASE_URL}${endpoint}`, {
    method: "POST",
    headers: {
      "Authorization": `Bearer ${API_TOKEN}`,
      "Content-Type": "application/json"
    },
    body: JSON.stringify(body)
  });
  if (!res.ok) {
    const errorText = await res.text();
    throw new Error(`Error ${res.status}: ${errorText}`);
  }
  return res.json();
}

(async () => {
  try {
    // 1. Crear un grupo de menú "CRM"
    const grupo = await post("/api/desktopRoutes:create", {
      type: "group",
      title: "CRM"
    });
    console.log("Grupo creado:", grupo);
    // Suponiendo que el JSON devuelto contiene un campo "id" o similar:
    const groupId = grupo.id || grupo.uid;

    // 2. Crear una página "Leads" dentro del grupo CRM
    const pagina = await post("/api/desktopRoutes:create", {
      type: "page",
      title: "Leads",
      parentId: groupId,
      name: "leads",
```

```
      path: "/leads"
    });
    console.log("Página creada:", pagina);
    // Extraer identificador del contenedor de la página (grid principal)
    const gridUid = pagina.schemaUid || pagina.gridUid || pagina.uid;
    // Nota: La estructura exacta puede variar; ajusta según el campo real
devuelto.

    // 3. Agregar bloque de Tabla para la colección "leads"
    const tablaBlock = await post("/api/uiSchemas:create", {
      componentName: "Table",
      collection: "leads",
      parentUid: gridUid
    });
    console.log("Bloque tabla creado:", tablaBlock);
    const tableBlockUid = tablaBlock.id || tablaBlock.uid ||
tablaBlock.tableBlockUid;
    const actionsColUid = tablaBlock.actionsColumnUid ||
tablaBlock.actionsColumnId;

    // 4. Agregar columnas al bloque tabla (ejemplo con 2 campos)
    await post("/api/uiSchemas:create", {
      componentName: "TableColumn",
      fieldPath: "company_name",
      parentUid: tableBlockUid
    });
    await post("/api/uiSchemas:create", {
      componentName: "TableColumn",
      fieldPath: "email",
      parentUid: tableBlockUid
    });
    console.log("Columnas agregadas a la tabla.");

    // 5. Agregar acción "Agregar nuevo" (botón global en la tabla)
    await post("/api/uiSchemas:create", {
      componentName: "Action",
      actionType: "addNew",
      collection: "leads",
      parentUid: tableBlockUid
    });
    // Agregar acción "Ver detalle" (botón por fila en la columna de
acciones)
    await post("/api/uiSchemas:create", {
      componentName: "Action",
      actionType: "view",
      collection: "leads",
      parentUid: actionsColUid,
      isRowAction: true
    });
    console.log("Acciones agregadas.");
```

```
    console.log("¡Página 'Leads' configurada correctamente mediante la
API!");
  } catch (error) {
    console.error("Error durante la configuración:", error);
  }
})();
```

En el código anterior, usamos funciones auxiliares para hacer las peticiones. Hemos impreso en consola algunos resultados para visualizar los IDs devueltos. Una vez ejecutado, la página **Leads** debería aparecer en la interfaz de NocoBase con una tabla de la colección *leads*, mostrando las columnas *company_name* y *email*, un botón **"Agregar nuevo"** y un botón de **"Ver"** en cada fila. Al pulsar *Agregar nuevo*, la interfaz abrirá un formulario emergente para crear un registro; al pulsar *Ver*, se abrirá una vista (posiblemente en un drawer lateral) con los detalles del registro seleccionado [11].

Ejemplo en TypeScript (uso de Axios)

A modo de demostración, aquí está la misma lógica utilizando **TypeScript** y la librería Axios, con definiciones de tipos mínimas para mayor claridad:

```
import axios from 'axios';

const BASE_URL = 'http://<tu-servidor-nocobase>';
const API_TOKEN = '<TU_TOKEN_API>';

interface RouteResponse {
  id?: number | string;
  uid?: string;
  schemaUid?: string;
  gridUid?: string;
  // ... otros campos posibles de la ruta
}
interface SchemaResponse {
  id?: number | string;
  uid?: string;
  tableBlockUid?: string;
  actionsColumnUid?: string;
  // ... otros campos posibles del schema
}

// Configurar instancia de Axios con base URL y header de auth
const api = axios.create({
  baseURL: BASE_URL + '/api',
  headers: {
    'Authorization': `Bearer ${API_TOKEN}`
  }
});

async function crearPaginaLeads() {
  // 1. Crear grupo de menú "CRM"
  const grupoResp = await api.post<RouteResponse>('desktopRoutes:create', {
```

```
  type: 'group',
  title: 'CRM'
});
const grupo = grupoResp.data;
const groupId = grupo.id ?? grupo.uid;

// 2. Crear página "Leads" en el grupo
const pageResp = await api.post<RouteResponse>('desktopRoutes:create', {
  type: 'page',
  title: 'Leads',
  parentId: groupId,
  name: 'leads',
  path: '/leads'
});
const pagina = pageResp.data;
const gridUid = pagina.schemaUid ?? pagina.gridUid ?? pagina.uid;

// 3. Añadir bloque Tabla de la colección leads
const tableResp = await api.post<SchemaResponse>('uiSchemas:create', {
  componentName: 'Table',
  collection: 'leads',
  parentUid: gridUid
});
const tablaBlock = tableResp.data;
const tableBlockUid = tablaBlock.uid ?? tablaBlock.tableBlockUid;
const actionsColUid = tablaBlock.actionsColumnUid;

// 4. Añadir columnas "company_name" y "email"
await api.post('uiSchemas:create', {
  componentName: 'TableColumn',
  fieldPath: 'company_name',
  parentUid: tableBlockUid
});
await api.post('uiSchemas:create', {
  componentName: 'TableColumn',
  fieldPath: 'email',
  parentUid: tableBlockUid
});

// 5. Añadir acción global "addNew" y acción por fila "view"
await api.post('uiSchemas:create', {
  componentName: 'Action',
  actionType: 'addNew',
  collection: 'leads',
  parentUid: tableBlockUid
});
await api.post('uiSchemas:create', {
  componentName: 'Action',
  actionType: 'view',
  collection: 'leads',
  parentUid: actionsColUid,
```

```
    isRowAction: true
  });

  console.log('Página "Leads" y sus componentes creados exitosamente.');
}

crearPaginaLeads().catch(err => {
  console.error('Error en la configuración de la página:',
err.response?.data || err.message);
});
```

En este ejemplo TypeScript, usamos interfaces para tipar (muy básicamente) las respuestas esperadas de creación de rutas y schemas. La lógica es equivalente al ejemplo en JavaScript: primero creamos el grupo, luego la página, después el bloque tabla, las columnas, y finalmente las acciones. Axios simplifica el manejo de JSON, y concatenamos la base `'/api'` a la URL en la instancia. Observa que se utiliza el sufijo `:create` en los endpoints según el estilo de acciones de NocoBase.

Referencias y Documentación

- **Documentación de NocoBase – Menús y Rutas:** Explicación de los tipos de ítems de menú (*Group, Page, Link*) y cómo se estructuran las rutas en NocoBase [1] [2]. Esta separación de `desktopRoutes` / `mobileRoutes` se introdujo en NocoBase 1.6.0.
- **Documentación de NocoBase – Pages & Blocks:** Conceptos de páginas como contenedores de bloques y acciones de vista [11]. Las páginas actúan como un *canvas* donde podemos colocar bloques libremente.
- **Foros de NocoBase y ejemplos:** La comunidad ha compartido fragmentos de código y preguntas frecuentes. Un ejemplo de integración mediante API (NocoBase MCP FlowModels) ilustra la secuencia de crear grupo, página, tabla, columnas y acciones vía código [14] [15].
- **Swagger API (plugin API Docs):** NocoBase incluye un plugin para documentación de API (basado en Swagger) que se puede activar desde el panel de administración [13]. Es muy útil para descubrir la estructura exacta esperada en cada endpoint (por ejemplo, nombres de campos como `x-component`, `x-action`, etc., en las versiones más recientes).
- **Guía de uso de API (Deployalo):** Tutorial básico (en español) sobre cómo usar la API REST de NocoBase para operaciones CRUD en colecciones [16]. Aunque no cubre la parte de UI, es útil para repasar autenticación y formato de URLs.
- **NocoBase Handbook – UI Editor:** NocoBase permite alternar al modo editor de interfaz, lo cual *internamente* utiliza estos mismos endpoints para crear o modificar páginas y bloques. Puedes probar manualmente en la interfaz qué cambios realiza, e incluso inspeccionar las llamadas de red en tu navegador al agregar una página o bloque, para entender qué envía (esto ayuda a verificar la carga útil exacta requerida).

Con estos pasos y ejemplos, deberías poder **automatizar la creación de páginas visuales** en tu instancia auto-alojada de NocoBase. Esto es útil para aprovisionar dinámicamente vistas de colección y formularios sin tener que configurarlas manualmente, manteniendo la sincronización entre la estructura de datos (colecciones) y la interfaz visual. ¡Recuerda siempre probar en un entorno de desarrollo antes de aplicar cambios masivos en producción, y hacer respaldos de la base de datos/schema por seguridad!

Referencias: NocoBase Docs (menús, páginas, bloques) [1] [11]; Foro NocoBase (ejemplos de API UI) [14] [15]; Deployalo Docs (uso de API REST) [16].

1 Menus - NocoBase
https://docs.nocobase.com/manual/ui/menus/

2 NocoBase v1.6.0 正式版发布 - 新闻动态 - NocoBase Forum
https://forum.nocobase.com/t/nocobase-v1-6-0/3483

3 Blocks - Carriers of Data and Content
https://docs.nocobase.com/handbook/ui/blocks/

4 nocobase/actions
https://docs-cn.nocobase.com/api/actions

5 16 Usar la API REST de NocoBase | Deployalo Docs
https://deployalo.com/docs/recursos/nocobase/usar-la-api-rest-de-nocobase/

6 7 8 9 10 12 14 15 NocoBase MCP Server | MCP Servers · LobeHub
https://lobehub.com/mcp/tsxcorp-nocobase-mcp

11 View - NocoBase Documentation
https://v2.docs.nocobase.com/interface-builder/actions/types/view

13 Is it possible to add custom api endpoint? - NocoBase Forum
https://forum.nocobase.com/t/is-it-possible-to-add-custom-api-endpoint/393