


GUÍA | ¿Cómo empezar un proyecto individual?

 Antes de comenzar, te recomendamos leer el [README](#) y la Documentación de la API Externa para comprender la temática que debes desarrollar y lo que se considera como requisito obligatorio para estar en condiciones de presentar un PI.

☐ LISTADO DE REQUISITOS INDISPENSABLES

GENERAL

✦ TENER EN CUENTA

- Es requisito que el formulario de creación esté validado con JavaScript y no sólo con validaciones HTML.
- Para las funcionalidades de filtrado y ordenamiento **NO** puedeS utilizar los end-points de la API externa que ya devuelven los resultados filtrados u ordenados, sino que debes realizarlo tu mismo.
- Utilizar únicamente los end-points que están indicados en el [README](#).

BACK-END

✦ CONFIGURACIÓN ARCHIVO ".env"

```
DB_USER=usuarioDePostgres
DB_PASSWORD=passwordDePostgres
DB_HOST=localhost
API_KEY **sólo si tu proyecto lo necesita**
```

✦ GENERAR LA BASE DE DATOS

- Será necesario que crees, **desde psql (shell o PGAdmin)**, una la base de datos. Si no realizas este paso de manera manual no podrás avanzar con el proyecto.
- Colócale el mismo nombre que aparece en el archivo db.js. En el siguiente ejemplo, **pi** sería el nombre de la base de datos:

```
new Sequelize(`postgres://${DB_USER}:${DB_PASSWORD}@${DB_HOST}/pi`);
```

✦ Realizados estos pasos, en la línea de comandos debes posicionarte en la carpeta `/api`. Una vez allí, ya puedes ejecutar el comando `npm start`. Si el servidor está corriendo, deberías ver algo así:

```
[nodemon] starting `node index.js` %s listening at 3001
```

BASE DE DATOS

✦ MODELOS

- Debemos generar el código para ambos modelos y tener en cuenta que en el [README](#) nos especifica cuáles campos son obligatorios (lo que te ayudará a utilizar validaciones y restricciones en cada campo, de ser necesario).

[⚠ **IMPORTANTE**]: busca la forma de generar un `ID` que no te traiga conflictos con los IDs que tienen los elementos traídos de la API. Existe, por ejemplo, el identificador único universal o `UUID`. Investiga sobre esto 😊 .

- Luego del paso anterior, debemos aplicar "*destructuring*" de los modelos en el archivo `db.js`. En este archivo encontrarás comentarios que te indican dónde hacerlo y un ejemplo de cómo hacerlo.

[**NOTA**]: en todos los Proyectos Individuales se plantea la necesidad de generar una relación de tipo N:N. Investiga en la documentación de [sequelize](#) sobre cómo definirla en forma correcta.

✦ RUTAS

- Una vez realizados los modelos y las relaciones, podemos pensar en las rutas. Recuerda leer el [README](#), donde se indica cuáles son las rutas necesarias, además de si son de tipo `GET` o `POST`, y si necesitan `params` o `query params`.
 - La ruta `GET` que retorna todos los resultados debe devolver sólo los datos necesarios para la ruta principal (tanto los mostrados en cada Card, como los necesarios para realizar los filtros y ordenamientos).
 - La ruta `GET` por ID utilizada para mostrar el detalle de cada elemento debe traer sólo los datos pedidos en la ruta de detalle (según lo indicado en el [README](#)).

[**NOTA**]: recuerda que para usar librerías como `axios` deberás instalarlas previamente.

✦ TEST

- Luego de hacer cada ruta te conviene testearlas. Puedes utilizar algún cliente HTTP para realizar solicitudes como Postman, Insomnia o Thunder.

FRONT-END

✦ DISEÑO

- Intenta utilizar estilos uniformes en todo la SPA. Puedes buscar una [paleta de colores](#) y mantenerla.

- Es recomendable utilizar la misma fuente y el mismo tamaño de letra, botones con el mismo estilo y color para los que realizan la misma acción (por ejemplo, borrar).
- No se permitirá utilizar librerías externas para aplicar estilos a la aplicación.
- Los elementos deben estar centrados y estilizados.
- La **Landing Page** debe tener alguna imagen de fondo representativa al proyecto y un botón que redirija a la Home Page.

✈ RUTAS

- Crea rutas para cada una de las vistas que necesites (Landing Page, Home Page, Detail Page, etc...)

✈ STORE

- Configura el store para tener tu fuente de verdad y poder usarla donde la necesites.

✈ HOME PAGE

- Aquí vas a renderizar los resultados obtenidos; cada uno en una card. Además, existen otros elementos necesarios:
 - **Paginado**: con la cantidad de elementos mencionados en el [README](#).
 - **Search**: buscar por algún criterio. Lee en el [README](#) si la búsqueda debe ser exacta o no.
 - **Filtros**: los resultados deben estar paginados.
 - **Ordenamiento**: debe funcionar combinado con el/los filtro/s.

✈ DETAIL PAGE

- Se debe visualizar toda la información que se solicita en el [README](#).

✈ FORM PAGE

- Utilizar validaciones JavaScript.
- Utiliza las validaciones para que tu formulario sea reactivo y valide datos a medida que completas cada campo.
- Confirma si el elemento se ha creado correctamente.
- Si ocurre algún error en el backend debe comunicarlo a los usuarios de tu página.
- Al finalizar la creación limpia los campos de tu formulario.

PUNTOS EXTRA

✈ TESTS

- Al menos tener un componente del frontend con sus tests respectivos.
- Al menos tener dos ruta del backend con sus tests respectivos.
- Al menos tener un modelo de la base de datos con sus tests respectivos.

✈ BUENAS PRÁCTICAS

- Utilizar código modularizado. Reutilizar componentes en el front-end. Usar helpers en el Back-end.

✈ EXTRA FEATURES

- Agregar funcionalidades extras, que no fueron solicitadas en el README.