

## EXERCISES QUE TRABAJAREMOS EN LA CUE

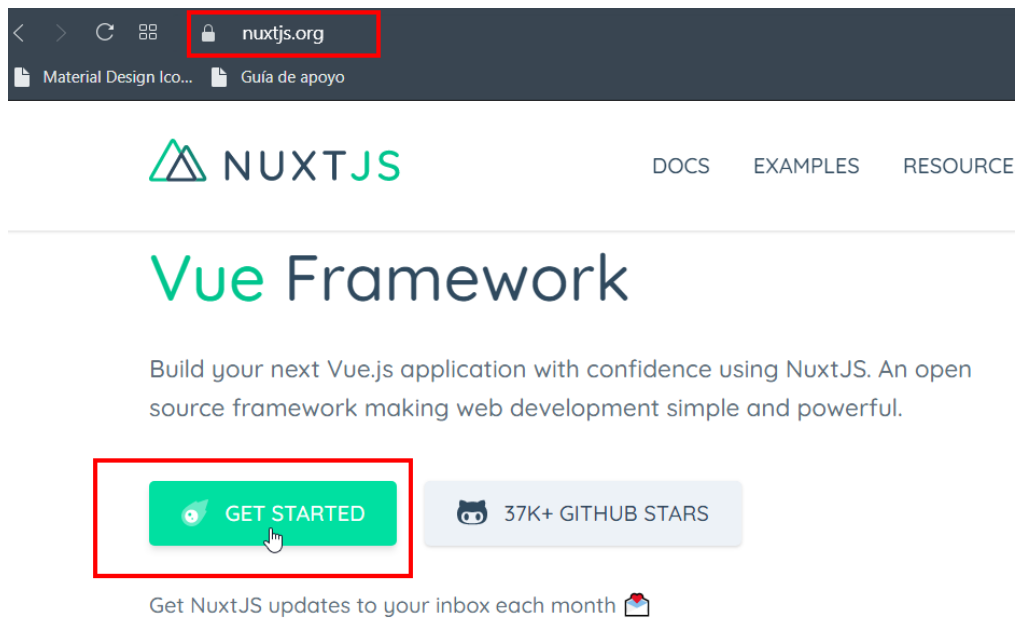
- EXERCISE 1: INSTALANDO NUXT JS EN NUESTRO PROYECTO.
- EXERCISE 2: UTILIZANDO NUXT JS PARTE 1.
- EXERCISE 3: UTILIZANDO NUXT JS PARTE 2.

## EXERCISE 1: INSTALANDO NUXT JS EN NUESTRO PROYECTO

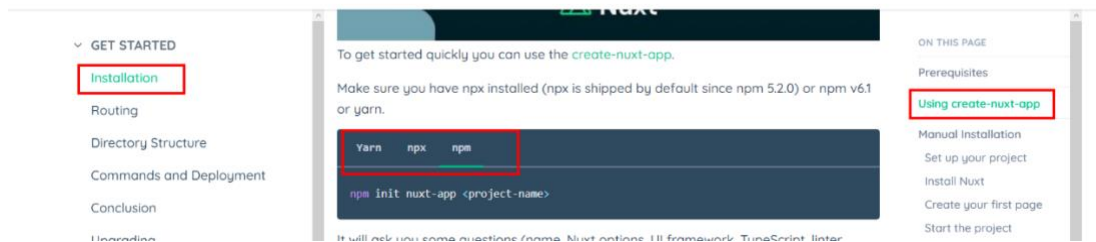
En este ejercicio trabajaremos **Nuxt JS**, que es un framework utilizado para el desarrollo de aplicaciones web, permitiendo crear aplicaciones estáticas **(Static Page)** de una sola página **(SPA)**, o de servidor **(SSR)**.

Trabaja sobre **VUE JS**, lo que quiere decir que tenemos todo lo bueno de **VUE**, pero contando con una organización y configuración establecida desde el principio, que ayuda a los desarrolladores a enfocarse sólo en el desarrollo.

Lo primero que haremos es ir al sitio oficial de **Nuxt**: <https://nuxtjs.org> y presionamos get started,



Vamos a **using create-nuxt-app**, y nos muestra qué comando podemos utilizar para crear un proyecto de **Nuxt**; regularmente, instalaríamos el cliente de **Nuxt** en el enlace **create-nuxt-app**, y con esto podríamos desarrollar cualquier proyecto, pero si no, lo podemos hacer como nos indica la página de Nuxt. Para no instalarlo globalmente, aquí podemos ver 3 opciones: **yarn, npx o npm**.



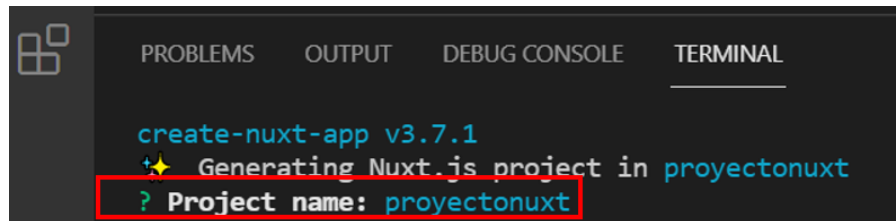
Ahora, lo que haremos será abrir VSC, iremos a la consola, verificamos que nos encontramos en la carpeta donde crearemos el proyecto, y ejecutaremos el comando **npx create-nuxt-app** junto al nombre, en este caso usaremos: "proyectonuxt".

```
$ npx-create-nuxt-app proyectonuxt
bash: npx-create-nuxt-app: command not found

/Exercises/ejercicio
$ npx create-nuxt-app proyectonuxt
[##.....] | loadDep:pify: sill resolveWithNewModule npm-conf@1.1.3 checking installable status
```

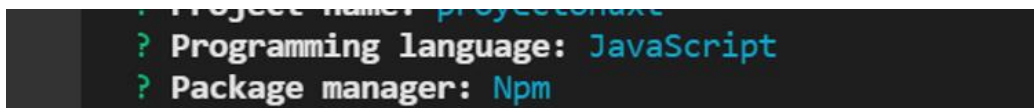
En primer lugar, lo que se está haciendo es descargar **create nuxt app**, una vez esté listo, se va a ejecutar ese comando, y va a crear nuestro proyecto. Debemos esperar que termine la descarga, el tiempo que demore sólo dependerá de nuestra conexión a internet.

Una vez descargado, nos pedirá información de cómo queremos configurar el proyecto. Lo primero que nos aparece es ponerle un nombre, lo dejaremos en: "proyectoNuxt", y teclearemos enter,



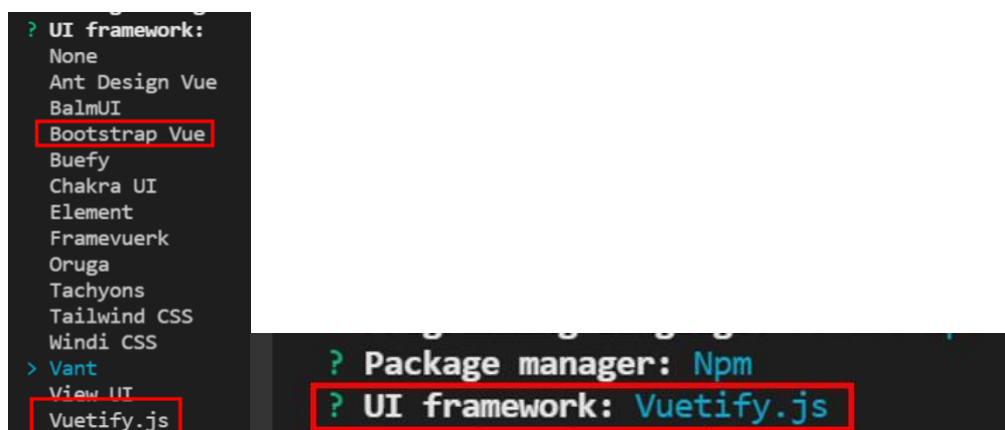
```
create-nuxt-app v3.7.1
🌟 Generating Nuxt.js project in proyectonuxt
? Project name: proyectonuxt
```

La siguiente pregunta es el lenguaje de programación que utilizaremos, pondremos **JavaScript**, y luego pregunta por el administrador de paquetes que queremos utilizar, en este caso será: **Npm**.



```
? Project name: proyectonuxt
? Programming language: JavaScript
? Package manager: Npm
```

Ahora, nos pregunta por algún framework de interfaz de usuario. Entre los conocidos, tenemos: **Bootstrap-Vue** y **Vuetify**; escogeremos el segundo.



```
? UI framework:
None
Ant Design Vue
BalmUI
Bootstrap Vue
Buefy
Chakra UI
Element
Framevuerk
Oruga
Tachyons
Tailwind CSS
Windi CSS
> Vant
View UI
Vuetify.js

? Package manager: Npm
? UI framework: Vuetify.js
```

Luego pregunta por los módulos de **nuxt** que queremos integrar para hacer peticiones, y escogeremos **axios**.

```
? Nuxt.js modules: (Press <space> to select, <a> to toggle all, <i> to invert selection)
> ( ) Axios - Promise based HTTP client
  ( ) Progressive Web App (PWA)
  ( ) Content - Git-based headless CMS
```

A continuación, escogeremos **eslint** para que nos ayuden a ordenar el código, y nos avise en caso de que no se cumplan algunas reglas, como: puntos y comas, si se he dejado una línea o algún espacio demás, entre otros.

```
? Linting tools: (Press <space> to select, <a> to toggle all, <i> to invert selection)
> ( ) ESLint
  ( ) Prettier
  ( ) Lint staged files
  ( ) StyleLint
  ( ) Commitlint
```

Como se puede observar, también nos da la opción de incluir un framework de testeo, allí se muestra **Jest** que es el que conocemos; en caso de necesitarlo, éste sería el elegido, pero como solo haremos un proyecto básico, seleccionaremos none.

```
? Linting tools: (Press <space> to select, <a> to toggle all, <i> to invert selection)
? Testing framework:
> None
  Jest
  AVA
  WebdriverIO
  Nightwatch
```

Luego, nos pregunta si queremos crear una aplicación universal, o **Single Page Application** típica; la diferencia es que la primera utiliza **Server Side Rendering**, es decir, el backend, y envía una

previsualización de cómo va a lucir la página, y, por ende, cómo va a comportarse para los navegadores o **crowlers** como cualquier tipo de aplicación.

Pero si utilizamos **Single Page Application**, vamos a perder esos beneficios; las aplicaciones universales tienen muchas más ventajas, pero es más complicado configurarlas, es por eso que estamos utilizando **Nuxt** para que lo haga por nosotros. Así es que, escogeremos la primera opción.

```
? Package manager: Npm
? UI framework: Vuetify.js
? Nuxt.js modules: (Press <space> to select, <a> to toggle all, <i> to invert selection)
? Linting tools: (Press <space> to select, <a> to toggle all, <i> to invert selection)
? Testing framework: None
? Rendering mode:
> Universal (SSR / SSG)
  Single Page App
```

Pondremos como servidor a **nodejs**, después nos notifica si queremos utilizar un archivo de configuración, y presionamos enter.

```
? Deployment target: Server (Node.js hosting)
? Development tools: (Press <space> to select, <a> to toggle all, <i> to invert selection)
> ( ) jsconfig.json (Recommended for VS Code if you're not using typescript)
  ( ) Semantic Pull Requests
  ( ) Dependabot (For auto-updating dependencies, GitHub only)
```

Nos pide nuestro nombre de usuario de **GitHub**, y si queremos utilizar nuestro control de versiones le pondremos none.

```
? What is your GitHub username? solange
? Version control system:
  Git
> None
```

Con esto, finalmente se crea el proyecto, y empieza a instalar las dependencias que hemos escogido.

```
? What is your GitHub username? solange
? Version control system: None
Warning: name can no longer contain capital letters
npm WARN deprecated core-js@2.6.12: core-js@<3.3 is no longer maintained
whims, feature detection in old core-js versions could cause a slowdown u
tual version of core-js.

\ Installing packages with npm
```

Una vez instalado, nos dará un mensaje de proyecto creado satisfactoriamente, junto al nombre, y luego muestra algunos comandos que se pueden utilizar para ejecutar el proyecto.

```
Successfully created project proyectoNuxt

To get started:

  cd proyectoNuxt
  npm run dev

To build & start for production:

  cd proyectoNuxt
  npm run build
  npm run start
```

Entonces escribimos el comando `cd proyectoNuxt`, y una vez dentro de nuestro proyecto, escribimos `npm run dev`, éste ejecutará un servidor de desarrollo con nuestra aplicación.

```
Edutecno@DESKTOP-LOFETOU MINGW64 ~/Documents/Front-end/M5CUE/CUE9
$ cd proyectoNuxt


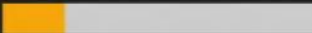
Edutecno@DESKTOP-LOFETOU MINGW64 ~/Documents/Front-end/M5CUE/CUE9/proyectoNuxt
$ npm run dev
```

Veremos la información de cómo está preparando el proyecto, y nos pregunta si queremos participar en el desarrollo de este framework, allí pondremos No.

```
i NuxtJS collects completely anonymous data about usage.  
This will help us improve Nuxt developer experience over time.  
Read more on https://git.io/nuxt-telemetry
```

```
? Are you interested in participating? No
```

Se puede ver la carga de la construcción, tanto del cliente como del servidor, esto nos sirve por si tenemos algún problema en una de las partes, y nos avisará.

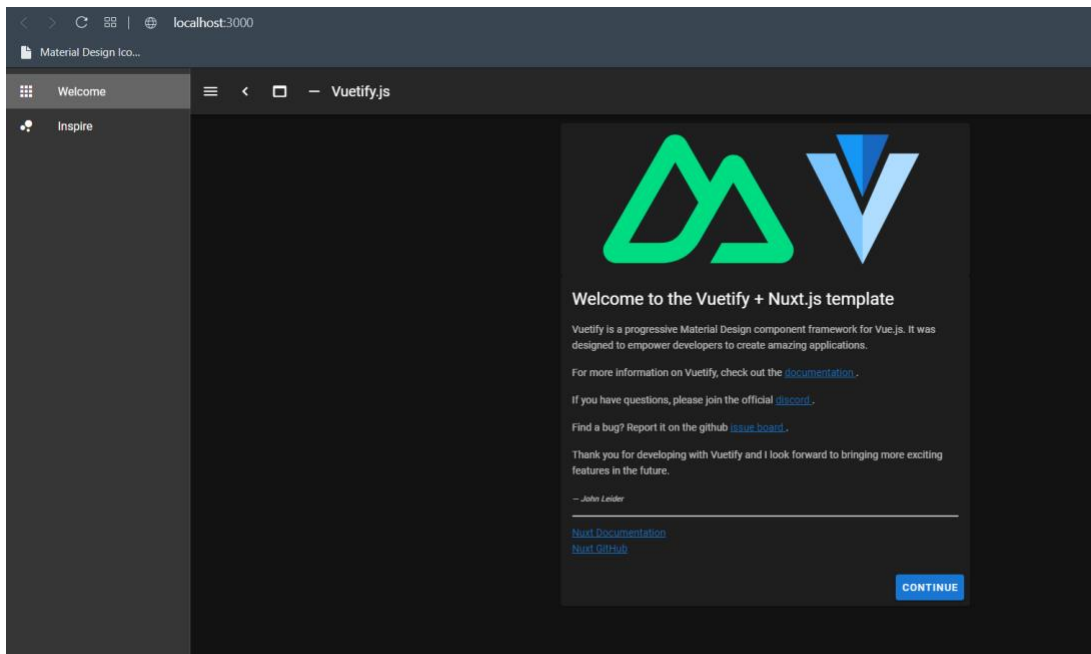
```
✓ Builder initialized  
✓ Nuxt files generated  
  
* Client  building (32%) 191/220 modules 29 active  
node_modules\vue-no-ssr\dist\vue-no-ssr.common.js  
  
* Server  building (18%) 73/77 modules 4 active  
babel-loader » vue-loader » .nuxt\components\nuxt-build-indicator.vue
```

Se observa que todo ha salido correctamente, y lo podemos visualizar en `localhost:3000`, damos clic , y se nos abrirá el navegador, podremos ver que nuestro proyecto con `Nuxt` y `Vuetify` fue creado exitosamente.

```
► Target: server
```

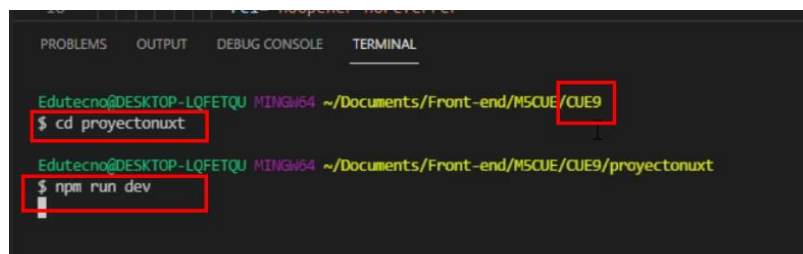
```
Listening: http://localhost:3000/
```

Al momento de crear un proyecto en Nuxt, es recomendable que el nombre de éste sea siempre en minúsculas, para evitar futuros errores con su creación.



## EXERCISE 2: UTILIZANDO NUXT JS PARTE 1

En este ejercicio utilizaremos **Nuxt**. Vamos a VSC, levantaremos el proyecto, nos dirigimos a la consola, nos ubicaremos en la carpeta donde lo tenemos, y una vez dentro, escribiremos el comando: **npm run dev**.



```
Edutecno@DESKTOP-LQFETQU MINGW64 ~/Documents/Front-end/MSCUE/CUE9
$ cd proyectonuxt
Edutecno@DESKTOP-LQFETQU MINGW64 ~/Documents/Front-end/MSCUE/CUE9/proyectonuxt
$ npm run dev
```



Veremos el proyecto en la ruta localhost:3000, y presionamos la URL.

```
Nuxt @ v2.15.7

> Environment: development
> Rendering:   server-side
> Target:      server

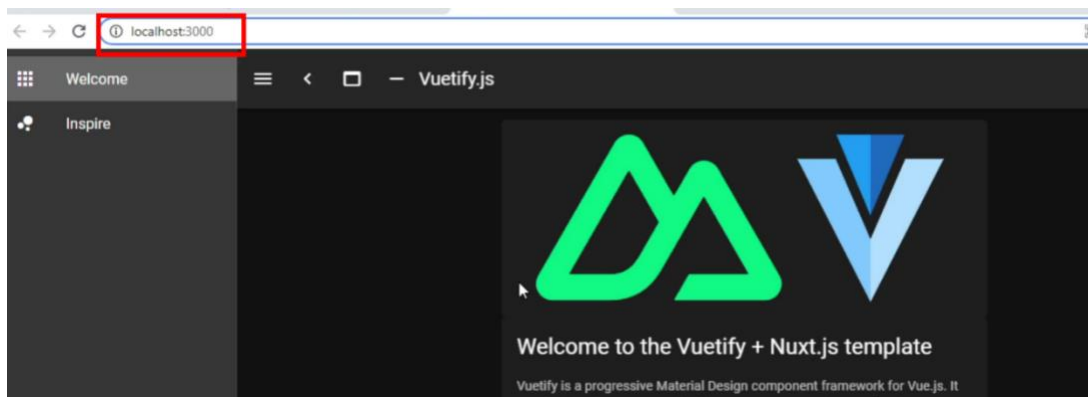
Listening: http://localhost:3000/

i Preparing project for development
i Initial build may take a while
i Discovered Components: .nuxt/components/readme.md
✓ Builder initialized
✓ Nuxt files generated

✓ Client
  Compiled successfully in 6.59s

✓ Server
  Compiled successfully in 5.41s
```

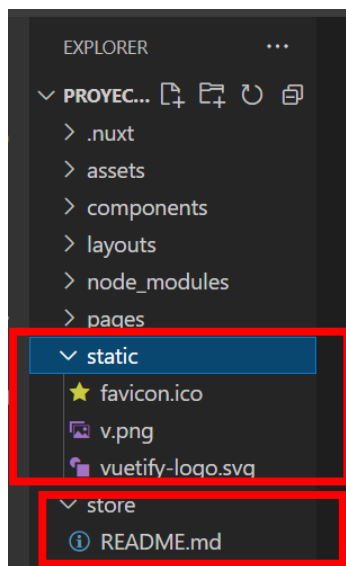
Nos vamos al navegador, allí todo está instalado y corriendo correctamente.



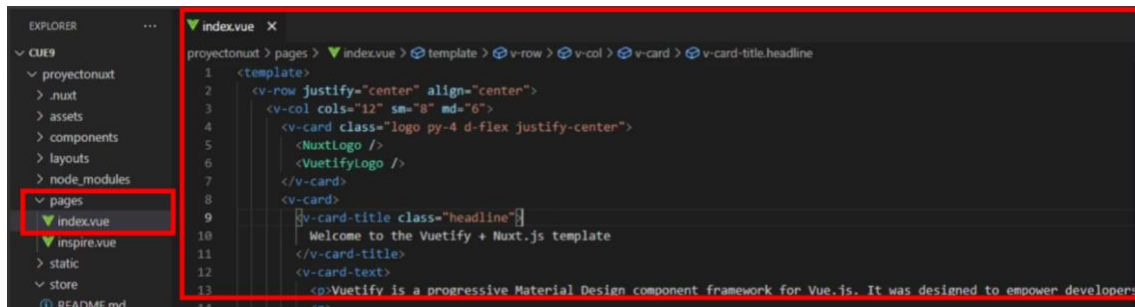
Ahora, vamos a revisar la estructura del proyecto en el archivo `package.json`, allí podemos ver los comandos ya ejecutados, por ejemplo: `npx run dev`; también está el `npx run build`, para construir la aplicación una vez hayamos terminado de desarrollar y convertir el código a producción; el comando `start`, para ejecutar finalmente el proyecto; y el `generate`, para poder generar la lista de dependencias.

```
{ } package.json X
{ } package.json > ...
1 {
2   "name": "proyectonuxt",
3   "version": "1.0.0",
4   "private": true,
5   "scripts": {
6     "dev": "nuxt",
7     "build": "nuxt build",
8     "start": "nuxt start",
9     "generate": "nuxt generate",
```

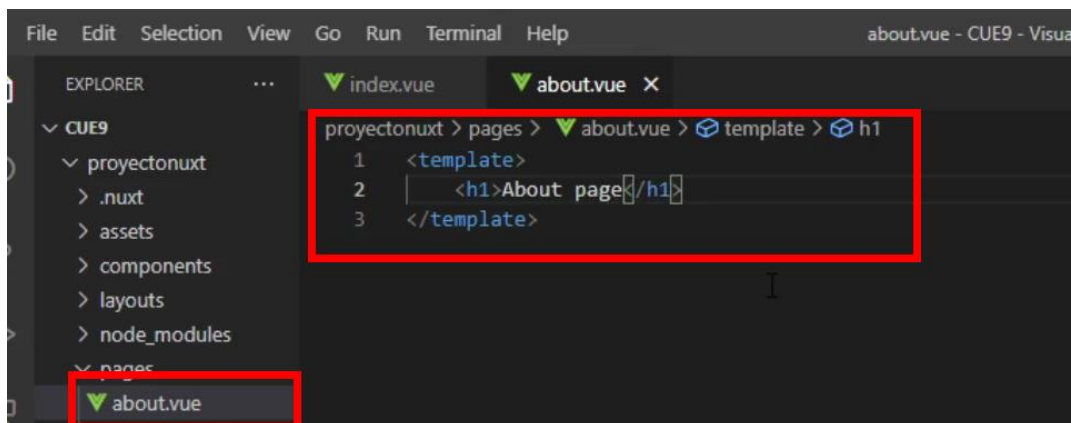
También tenemos la carpeta **store**, para utilizar **vuex** en **static**, pondremos los archivos estáticos como: imágenes, stylesheet, entre otros.



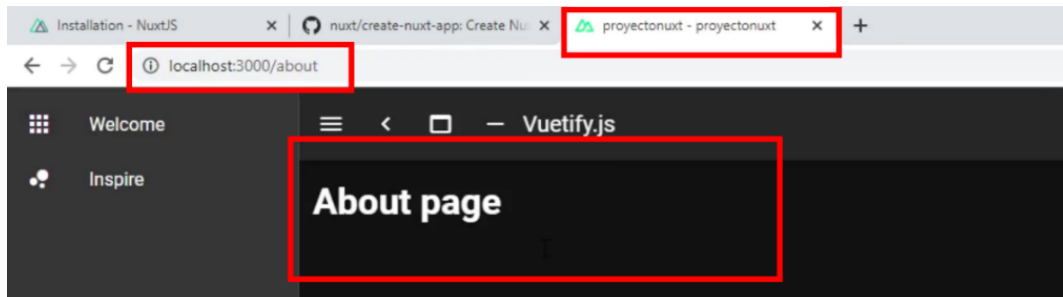
En la carpeta **page**, es donde se ve el archivo **index.vue**, éste es el que hace referencia a la página inicial cuando levantamos el proyecto.



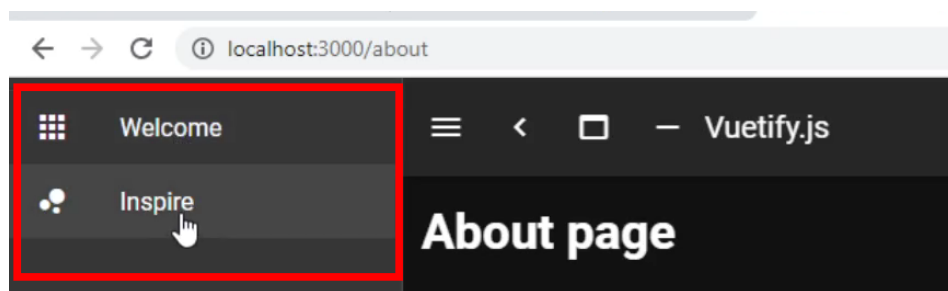
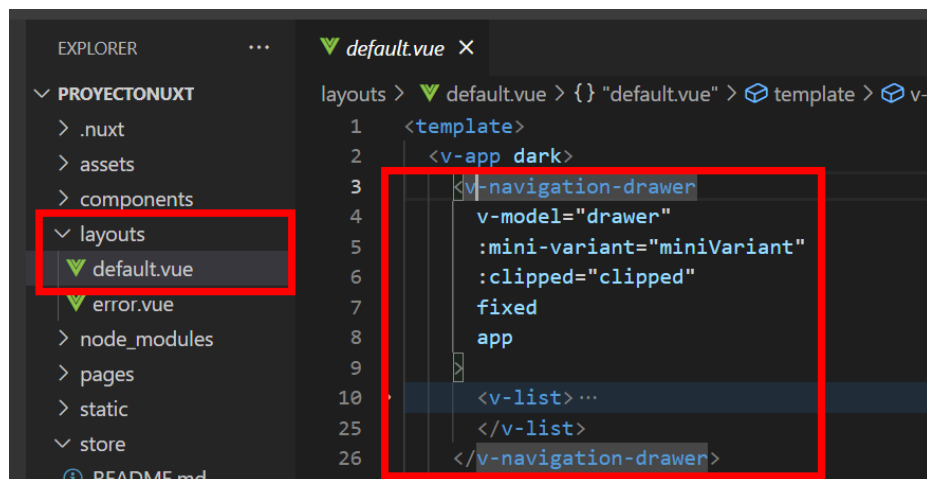
Lo interesante de **Nuxt** es que nosotros podemos crear archivos, y van a ser accedidos como si fueran una especie de servidor de archivos estáticos; es decir, al ser creado, ya tenemos una ruta que visitar, sin la necesidad de crear **router**, como lo hacíamos en **VUE Js**. Por ejemplo: si generamos un archivo **about.vue** en la carpeta **pages**, escribimos el **template** y un **h1** “About page”.



Nos dirigimos al navegador, y agregamos a la ruta `/about`, presionamos enter, y podremos ver inmediatamente la page que acabamos de crear.

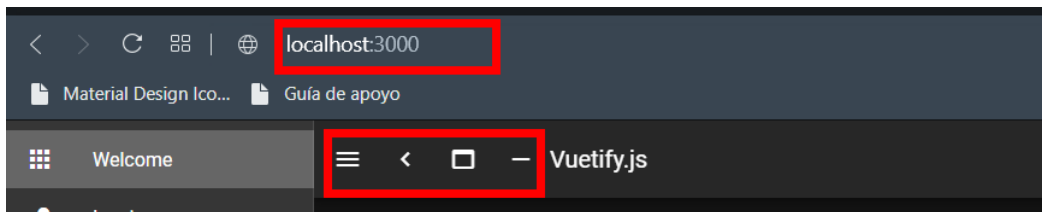


La carpeta **layouts** es muy utilizada si queremos tener una estructura típica en nuestra aplicación; por ejemplo, **default.vue**, vendría siendo la aplicación en sí. Aquí está el enrutador de qué páginas se van a mostrar, se puede ver la estructura de donde estará el navigation drawer, que sería welcome y el inspire.



El `v-app-bar`, es donde tenemos los elementos fijos, junto a la palabra `Vuetify`.

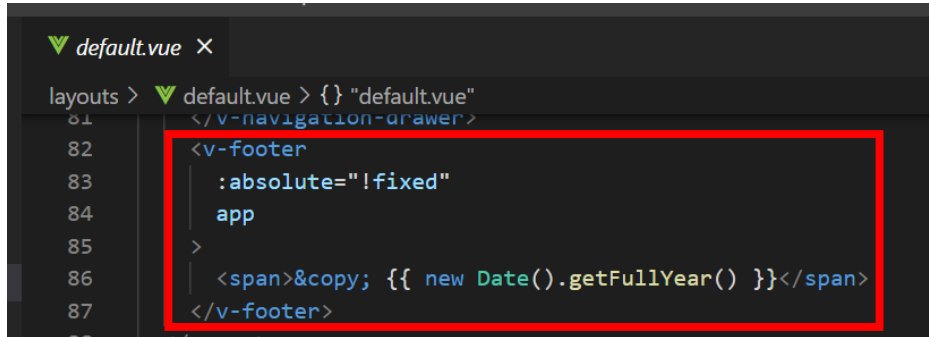
```
default.vue X
layouts > default.vue > {} "default.vue" > template > v-app > v-app-bar
26 </v-navigation-drawer>
27 <v-app-bar
28   :clipped-left="clipped"
29   fixed
30   app
31   <v-app-bar-nav-icon @click.stop="drawer = !drawer" />
32   <v-btn ...
33 >
38 </v-btn>
39 >
44 </v-btn>
45 >
50 </v-btn>
51 <v-toolbar-title v-text="title" />
52 <v-spacer />
53 >
58 </v-btn>
59 </v-app-bar>
```



Entremedio tenemos a `Nuxt` siendo invocado.

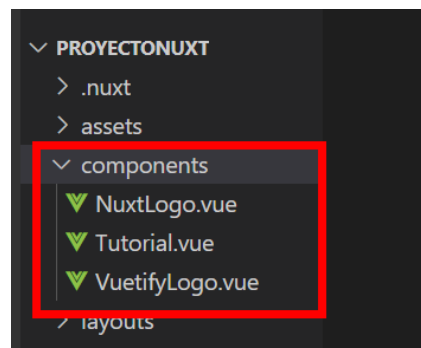
```
default.vue X
layouts > default.vue > {} "default.vue" > template
59 </v-app-bar>
60 <v-main>
61   <v-container>
62     <Nuxt />
63   </v-container>
64 </v-main>
```

Y por último, tenemos el **footer**.



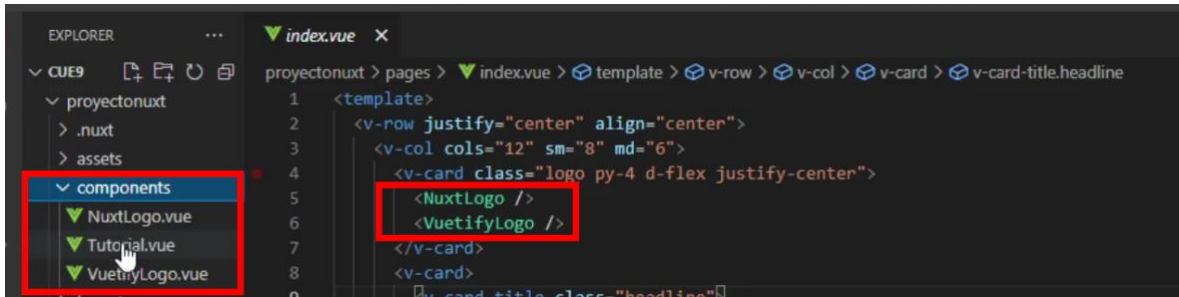
```
default.vue X
layouts > default.vue > {} "default.vue"
81 </v-navigation-drawer>
82 <v-footer
83   :absolute="!fixed"
84   app
85 >
86   <span>&copy; {{ new Date().getFullYear() }}</span>
87 </v-footer>
```

Después, encontramos la carpeta **components**, que es conocida por ser utilizada en **Vue Js**. Recordemos que se pueden crear componentes, que luego serán reutilizados en diferentes vistas; en el caso de **Nuxt**, en diferentes **pages**.

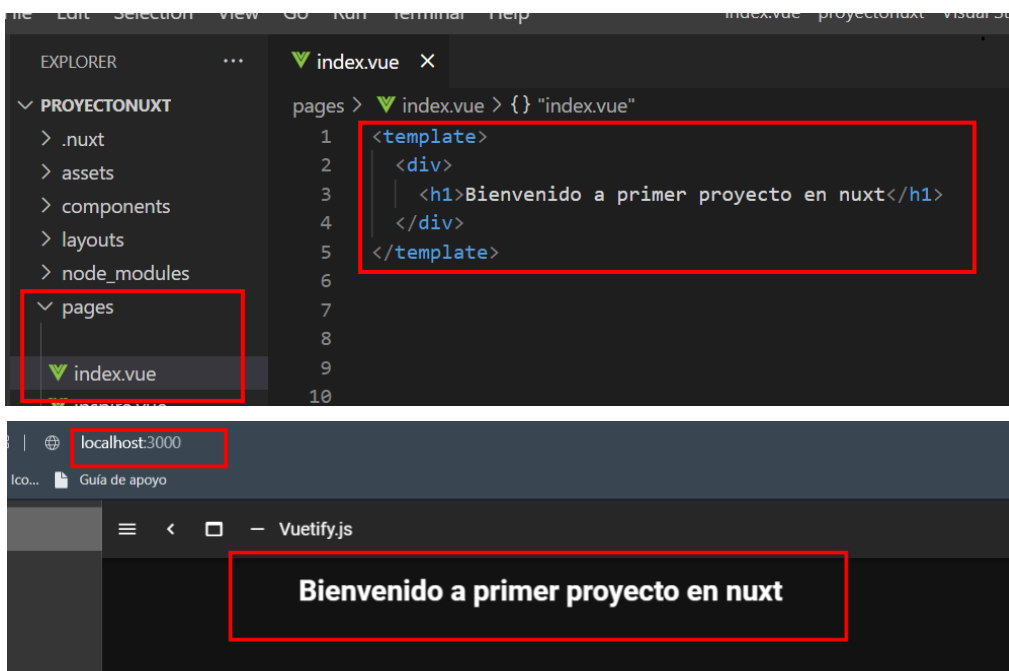


Ahora bien, ¿por qué **Nuxt** es mejor que una aplicación desarrollada solo con **Vue Js**? porque en cada página podemos agregar información que está relacionada no solo con la aplicación, sino que también con la meta información; es decir, que cuando creamos una página típica en otros lenguajes, como **Java**, **PHP**, entre otros, se están controlando todas las vistas desde el backend, y se pueden enviar cuáles fueron los títulos, la descripción, los tags, y al final, cada una de esas páginas serán leídas por los buscadores, y éstos sabrán cuál es el contenido, y ubicarán nuestra página en una buena posición al momento que un usuario busca una determinada palabra. Pero, con las apps SPA, como son generadas dinámicamente, los buscadores no los pueden leer completamente; esto lo resuelve **Server Side Rendering**, como **Nuxt**, ¿y cómo lo podemos hacer en la práctica?

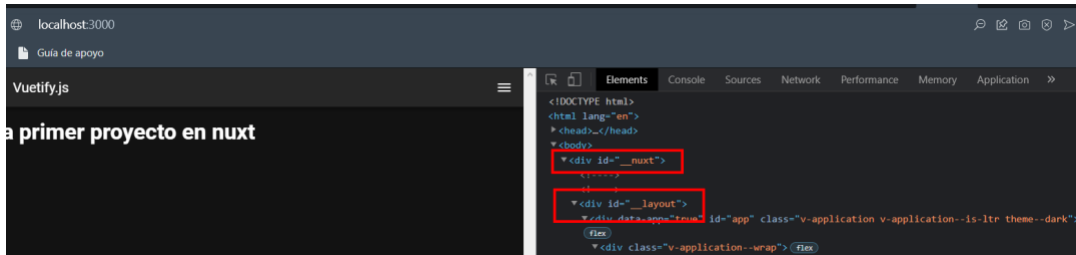
Modificaremos `index` de la carpeta `Pages`, allí estarán los logos, que vienen de la carpeta componentes.



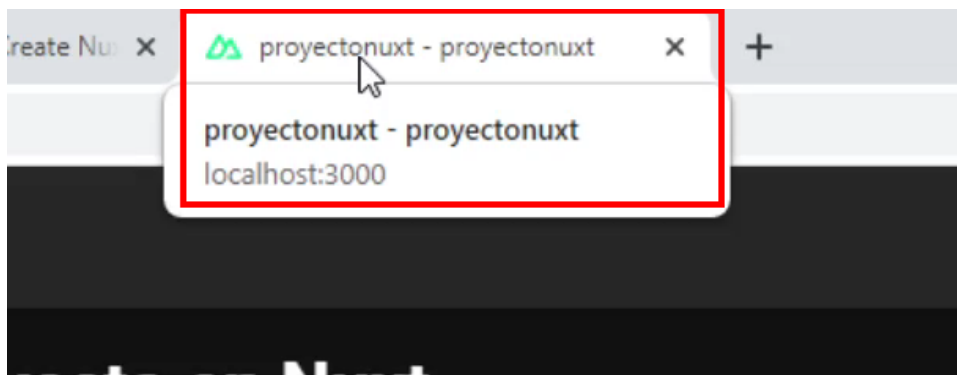
Borraremos todo, y vamos a crear una página inicial paso a paso, ahí colocaremos lo básico de un archivo `VUE`, y una etiqueta `h1` que diga: "Bienvenidos a primer proyecto en nuxt". Luego guardamos, y si revisamos el navegador y nos dirigimos a la ruta inicial, podemos ver el `h1` creado.



Inspeccionaremos el código, y notaremos que todo está dentro del `div` con `id nuxt`, y dentro está el `layout` que vamos a estar compartiendo con el `div` y el `h1` que creamos.



Ahora, añadiremos meta información. Por ejemplo, se puede ver que en la pestaña del navegador está el nombre de nuestro proyecto, éste también influye en los buscadores, por lo que agregaremos información para ser encontrados por ellos.



Entonces, en nuestro archivo `index` de la carpeta `Pages`, agregaremos las etiquetas `script` y empezaremos a crear una función. La primera será un `head`, que retornará un objeto, y éste es la meta información que estaremos pintando de la página.

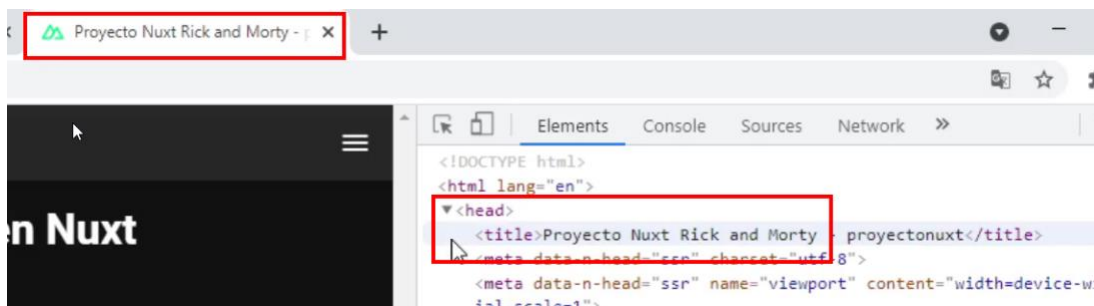
Para ejemplificarlo, agregaremos un título con un `title`, que será el nombre que tendría la etiqueta en `html`, y que tendrá el contenido tipo `string`: "Proyecto Nuxt Rick and Morty".



```

index.vue
pages > index.vue > {} "index.vue" > script > default > head >
4   </div>
5   </template>
6
7   <script>
8   export default {
9     head(){
10      return{
11
12        title: "Proyecto Nuxt Rick and Morty"
13      }
14    }
15  }
  
```

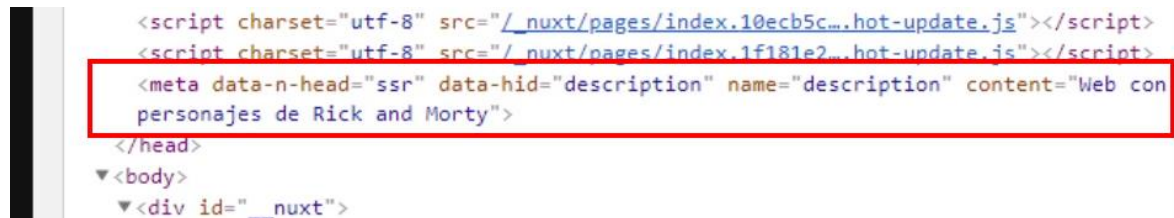
Revisamos el navegador, y aquí podemos ver que el nombre de la pestaña cambió, y si revisamos el código del navegador, también notaremos que se agregó el título como si fuera **html**.



Ahora, ¿qué pasa si queremos agregar una etiqueta **meta description**, que es muy útil para describir de qué se trata la página?: a continuación de **title**, agregaremos meta, que será un arreglo que contendrá un objeto, y éste tendrá una propiedad **hid**, que sirve para decirle a **nuxt** qué es lo que estamos agregando; escribiremos **description**, luego agregaremos **name: "description"**, que es el nombre de la etiqueta, y por último, el valor de dicha etiqueta; para esto ocuparemos: **content:** **"Web con personajes de Rick and morty"**

```
1 <template>
2   <div>
3     <h1>Bienvenido al primer proyecto en nuxt</h1>
4   </div>
5 </template>
6 export default {
7   head() {
8     return {
9       title: "Proyecto Nuxt Rick and Morty",
10      meta: [
11        {
12          hid: "description",
13          name: "description",
14          content: "Web con personajes de rick and morty"
15        }
16      ],
17    }
18  }
19 }
```

Vamos al navegador, y veremos el título, y también el **description**.

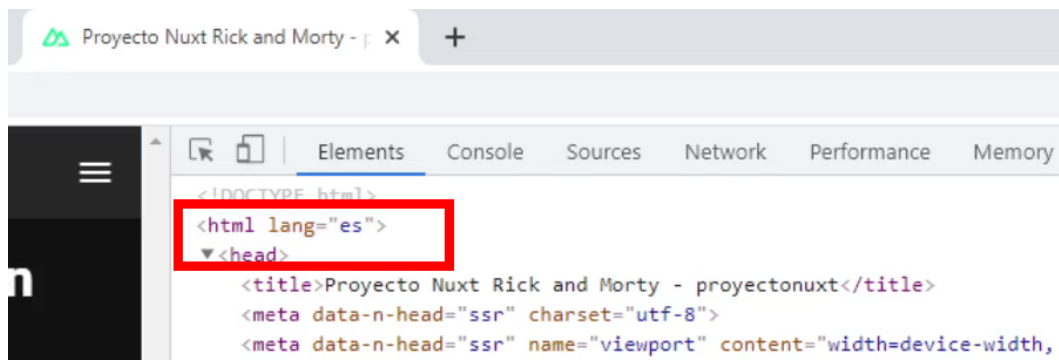


```
<script charset="utf-8" src="/_nuxt/pages/index.10ecb5c...hot-update.js"></script>
<script charset="utf-8" src="/_nuxt/pages/index.1f181e2...hot-update.js"></script>
<meta data-n-head="ssr" data-hid="description" name="description" content="Web con
personajes de Rick and Morty">
</head>
<body>
  <div id="__nuxt">
```

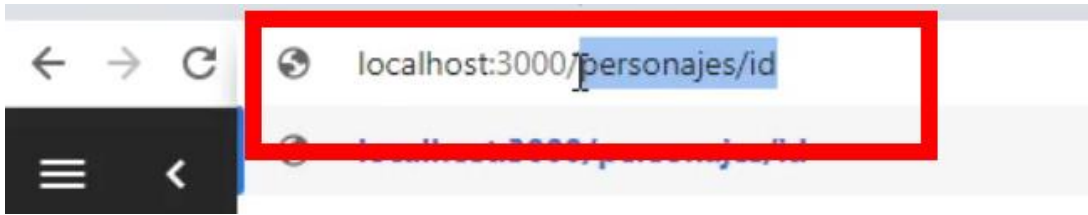
Lo siguiente que agregaremos será el idioma de nuestra página, para que se enlace con información, ya sea en español, inglés, entre otros. En las etiquetas script, después del arreglo meta, agregaremos **htmlAttrs** que será un objeto que contendrá: **Lang: 'es'**.

```
1 export default {
2   head() {
3     return {
4       title: "Proyecto Nuxt Rick and Morty",
5       meta: [
6         {
7           hid: "description",
8           name: "description",
9           content: "Web con personajes de rick and morty"
10        }
11      ],
12      htmlAttrs: {
13        lang: 'es'
14      }
15    }
16  }
17 }
```

Revisamos el navegador, y veremos que aparecerá el idioma en la cabecera.



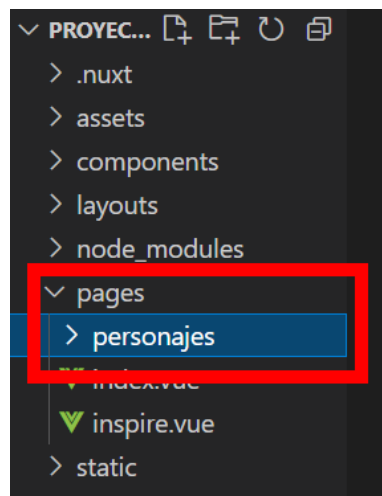
Ahora, ¿qué pasaría si tenemos una sección creada por múltiples páginas?: Por ejemplo, si hay una página con nombre "personajes", y escribimos en la **URL** de nuestra web: /personajes, al no estar creada en la carpeta **pages**, no nos mostrará nada, y seguido escribimos un **Id** donde queremos mostrar el detalle de ese personaje.



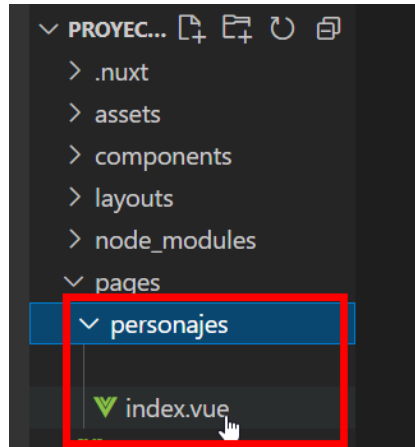
Lo que tenemos que hacer, es enlazar dos tipos de ruta, que pertenecen a una misma categoría; en este caso, a la categoría personajes.

### EXERCISE 3: UTILIZANDO NUXT JS PARTE 2

Para responder a la interrogante planteada, crearemos, dentro de la carpeta `page`, otra carpeta que se llamará "personajes".



Y aquí, para visitar el archivo principal de esta ruta, crearemos un archivo `index.vue`, lo que hará que al visitar la ruta `/personajes`, el navegador buscará dentro de la carpeta un archivo llamado `index`, y es lo que será mostrado.

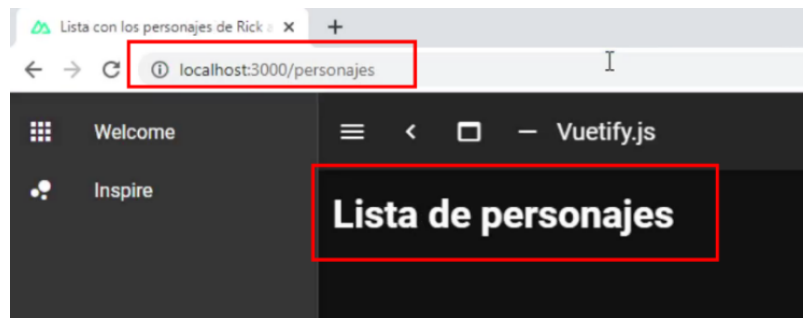


Entonces, agregaremos la estructura base con un **template**, una etiqueta **div** con un **h1** que dirá "Lista de personajes", y por último, la etiqueta **script**, donde pondremos lo mismo que en el archivo **index** principal del home, cambiaremos el **title** por "Lista con los personajes de Rick and Morty", y lo demás quedará igual.

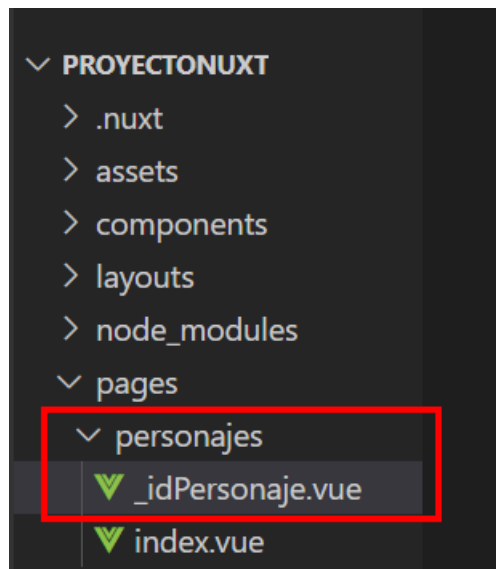
```

1  <template>
2    <div>
3      <h1>Lista de personajes</h1>
4    </div>
5  </template>
6  export default {
7    head() {
8      return {
9        title: "Lista con los personajes de Rick and Morty",
10       meta: [
11         {
12           hid: "description",
13           name: "description",
14           content: "Sitio con personajes de rick and morty"
15         }
16       ],
17       htmlAttrs: {
18         lang: 'es'
19       }
20     }
21   },
22 }
```

Iremos a la página principal del navegador, y en la **URL** agregaremos **/personajes**, como ésta es una carpeta, el primer archivo que buscará será **index**, presionamos enter, y podemos ver nuestro **h1**.



Ahora, si queremos pintar el **id**, crearemos un archivo **\_idPersonaje.vue** dentro de nuestra carpeta personajes.



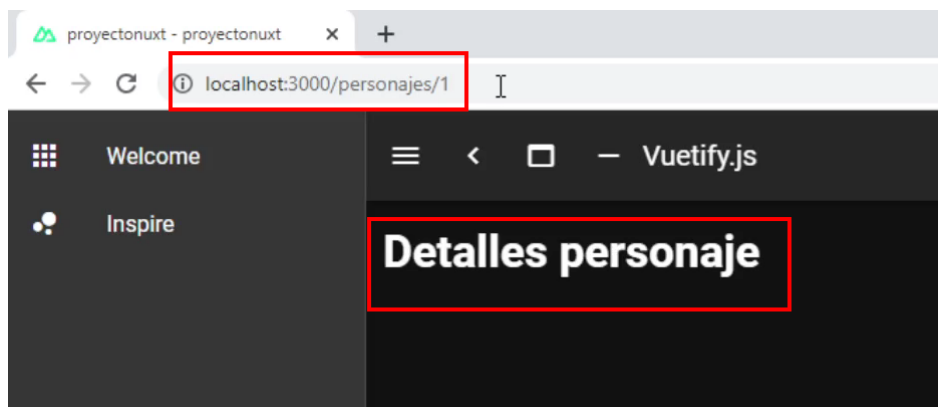
Luego, creamos la estructura base con un **h1** que dirá "Detalles personaje".

```

_idPersonaje.vue X
pages > personajes > _idPersonaje.vue > {} "_idPersonaje.vue" > 6
1 <template>
2   <div>
3     <h1>Detalles personaje</h1>
4   </div>
5 </template>
6

```

Vamos al navegador, y veremos que el `index` pinta la ruta personajes, y si a ésta le agregamos el `id` 1 (`/personajes/1`), nos mostrará el `h1` que escribimos en el archivo `_idPersonaje`.



Esto pasa porque estamos indicando que, dentro de una ruta, lo que le va a seguir será la ruta inicial de personajes (`index`), así como también la subruta (`id`), por ende, la estructura de nuestra carpeta personajes se ve reflejada en la `URL` final, todo para mantener un orden.

Ahora, lo que queremos es pintar datos de estos personajes, y lo haremos en el archivo `index` de su carpeta, al igual que en `VUE`, podemos traer datos de una `API`, para lo que agregaremos la propiedad `data` que retornará un objeto, el cual contendrá un arreglo vacío llamado personajes.

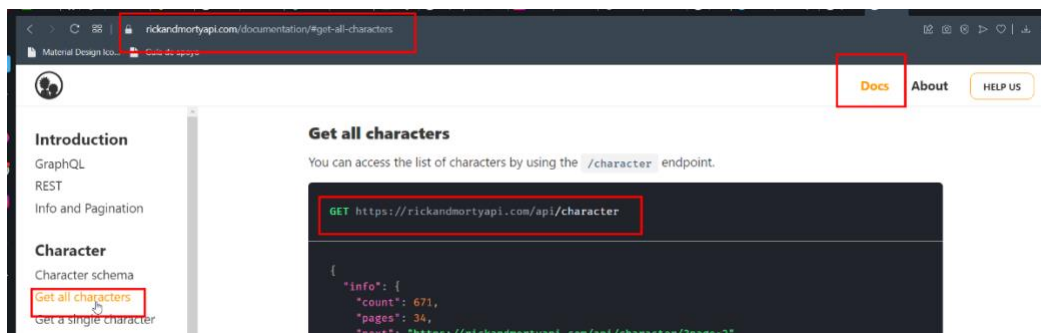
The screenshot shows the VS Code interface. On the left, the 'EXPLORER' sidebar shows a project structure with folders like .nuxt, assets, components, layouts, node\_modules, and pages. Under 'pages', there is a subfolder 'personajes' containing 'index.vue' and '\_idPersonaje.vue'. The 'index.vue' file is selected and highlighted with a red box. The main editor shows the content of 'index.vue', which includes a Vue component with 'htmlAttrs' and a 'data()' function. The 'data()' function is also highlighted with a red box and contains the following code:

```

data(){
  return{
    personajes: []
  }
},

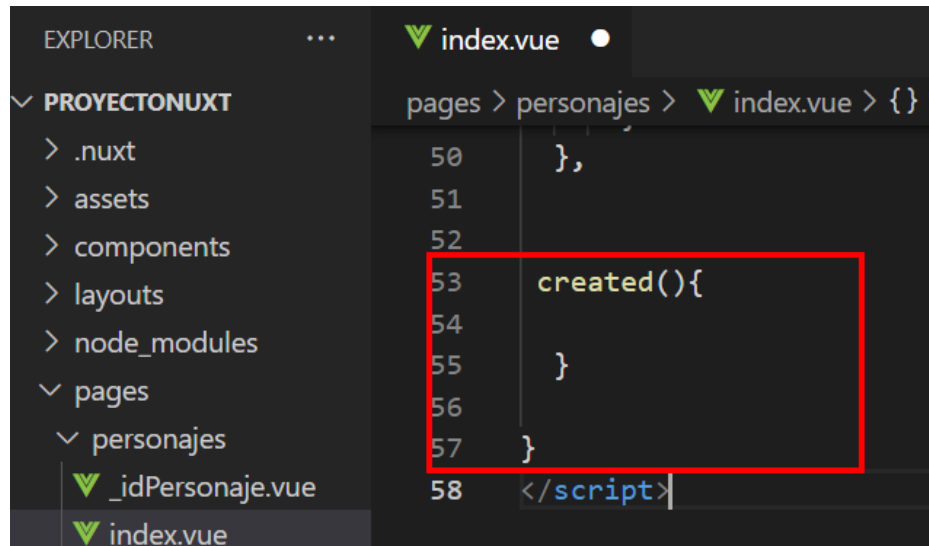
```

¿De dónde sacaremos toda esa información?: utilizaremos la **API** de Rick and Morty, nos dirigimos a la página <https://rickandmortyapi.com>, vamos a docs, y finalmente a "Get all characters", allí podemos ver la **URL** a la que haremos las peticiones **GET**.



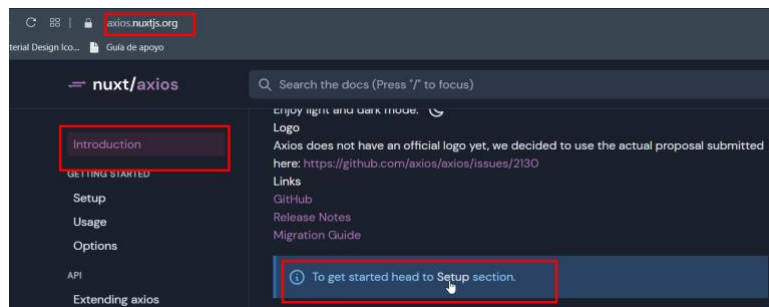
Volvemos a nuestro código, y agregaremos una función **created ()** para que nos traiga toda la información al cargar nuestro **index**.



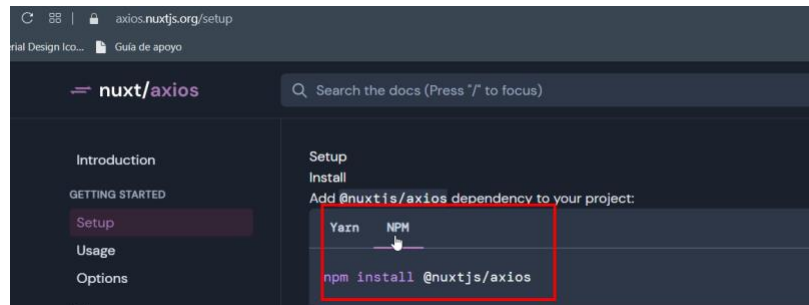


Para poder hacer los llamados a la **API**, tenemos que usar una biblioteca, en este caso **axios**, que es la que conocemos.

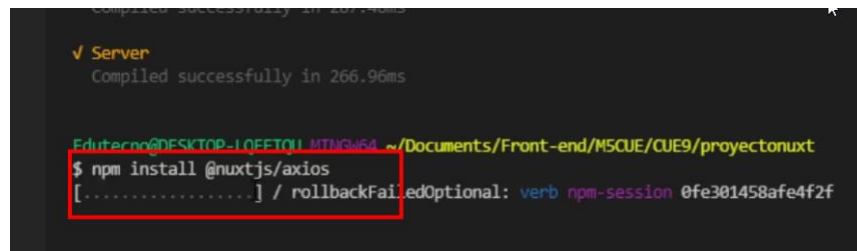
Existe una versión de **axios** para **Nuxt**, para ello nos dirigimos a la página <https://axios.nuxtjs.org>, y vamos a "To get started head to Setup section".



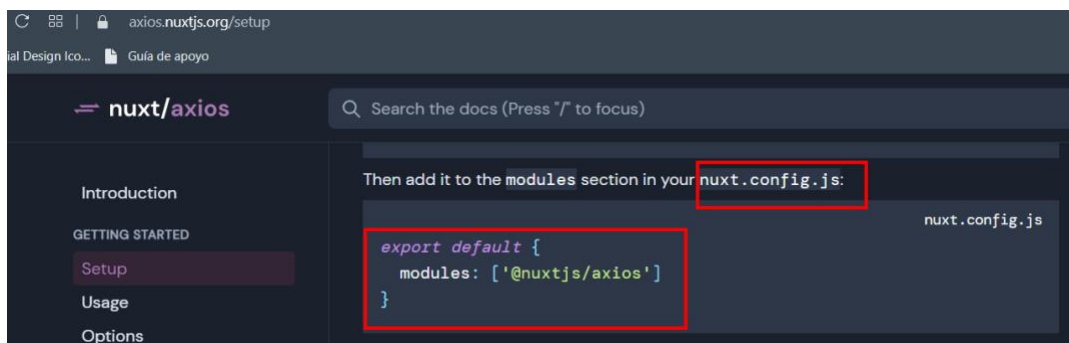
Presionamos allí, y nos dirigirá a las opciones para instalar **axios** en nuestro proyecto **Nuxt**, escogemos **npm** y copiamos el comando.



Nos dirigimos a la consola de VSC, detenemos nuestro proyecto con **control+c**, incluimos el comando y presionamos enter.



Mientras se instala **axios**, el otro paso que debemos seguir es agregar una línea de código en el archivo **nuxt.config.js**, en la sección de módulos.



Copiamos y pegamos donde nos indicó la página.

```

EXPLORER
PROYECTONUXT
  > .nuxt
  > assets
  > components
  > layouts
  > node_modules
  > pages
  > static
  > store
  > README.md
  > test
  > .babelrc
  > .editorconfig
  > .gitignore
  > nuxt.config.js
  > nuxt.config.js

JS nuxt.config.js
JS nuxt.config.js > default > vuetify > theme > themes
27 plugins: [
28   ],
29
30 // Auto import components: https://go.nuxtjs.dev/
31 components: true,
32
33 // Modules for dev and build (recommended): https
34 buildModules: [
35   // https://go.nuxtjs.dev/vuetify
36   '@nuxtjs/vuetify',
37 ],
38
39 // Modules: https://go.nuxtjs.dev/config-modules
40 modules: [
41   '@nuxtjs/axios'
42 ],
43
44 // Vuetify module configuration: https://go.nuxtjs
  
```

Volvemos a nuestro archivo `index` de la carpeta personajes, importamos `axios`, y lo utilizaremos en nuestra función `created()`. Como estamos empleando código moderno, utilizaremos `async` y `await`, por lo que agregaremos `async` a nuestra función `created()`, crearemos una constante `response`, y una vez termine la petición, guardaremos la respuesta en nuestro arreglo personajes.

```
1 import axios from 'axios'
2 export default {
3   head() {
4     return {
5       title: "Lista con los personajes de Rick and Morty",
6       meta: [
7         {
8           hid: "description",
9           name: "description",
10          content: "Sitio con personajes de rick and morty"
11        }
12      ],
13      htmlAttrs: {
14        lang: 'es'
15      }
16    },
17    data() {
18      return {
19        personajes: []
20      }
21    },
22    async created() {
23      const response = await
24      axios.get('https://rickandmortyapi.com/api/character')
25      this.personajes = response.data
26    }
27  }
28 }
29 }
```

Para probar la **API**, iremos a **Postman**, pegamos la **URL** donde haremos la petición de tipo **GET**, y veremos qué información nos trae. Si lo notamos, la descripción de los personajes está dentro del arreglo **results**.

```

6      "prev": null
7    },
8    "results": [
9      {
10       "id": 1,
11       "name": "Rick Sanchez",
12       "status": "Alive",
13       "species": "Human",
14       "type": "",
15       "gender": "Male",
16       "origin": { ...
17     },
18     "location": { ...
19   },
20 }
21 ]

```

```

71   {
72     "id": 2,
73     "name": "Morty Smith",
74     "status": "Alive",
75     "species": "Human",
76     "type": "",
77     "gender": "Male",
78     "origin": {
79       "name": "Earth (C-137)",
80       "url": "https://rickandmortyapi.com/api/location/1"
81     },
82     "location": { ...
83   },
84   "image": "https://rickandmortyapi.com/api/character/avatar/2.jpeg",
85 }
86 ]

```

Y para poder acceder a ésta, tendremos que agregar el `.results`, a continuación de `.data`.

```

//
async created(){
  const response = await axios.get('https://rickandmortyapi.com/api/character')
  this.personajes = response.data.results
}
}

```

Una vez terminada la petición, pintaremos en el `template` la información que queremos mostrar. Para esto crearemos una etiqueta `v-card` con la clase `mx-auto`, y a continuación, un `v-container`, un `v-row` con una propiedad `dense`, y un `v-col` donde pondremos un `v-for`.

```
1 <template>
2   <div>
3     <h1 class="pb-6">Lista de personajes</h1>
4     <v-card class="mx-auto">
5       <v-container>
6         <v-row dense>
7           <v-col v-for="personaje in personajes"
8 :key="personaje.id" cols="12" xl="4" lg="4" md="4">
9
10            </v-col>
11          </v-row>
12        </v-container>
13      </v-card>
14    </div>
15 </template>
```

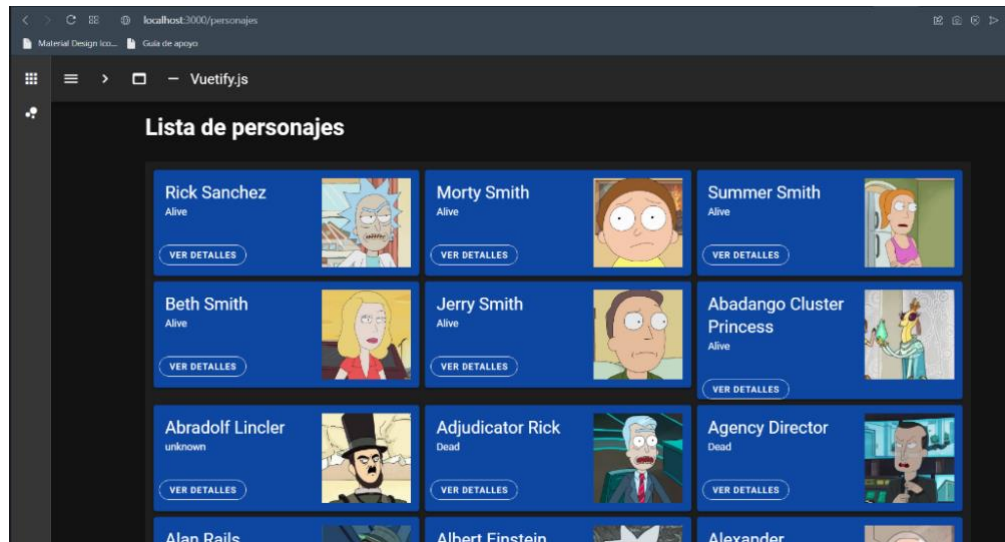
Utilizaremos una **card** simple de **Vuetify**, copiaremos y pegaremos el código, y continuamos; en este **v-card** se utilizará un color = **blue darken-4**.

```
1 <template>
2   <div>
3     <h1 class="pb-6">Lista de personajes</h1>
4     <v-card class="mx-auto">
5       <v-container>
6         <v-row dense>
7           <v-col v-for="personaje in personajes"
8 :key="personaje.id" cols="12" xl="4" lg="4" md="4">
9             <v-card color="blue darken-4" dark>
10
11              </v-card>
12            </v-col>
13          </v-row>
14        </v-container>
15      </v-card>
16    </div>
17 </template>
```

En **v-card title** agregaremos el **v-text personaje.name**, en **v-card-subtitle** pondremos **personaje.status**, y finalmente, en la imagen colocaremos **:src= "personaje.img"**, y al botón le daremos el nombre "VER DETALLES".

```
1 <template>
2   <div>
3     <h1 class="pb-6">Lista de personajes</h1>
4     <v-card class="mx-auto">
5       <v-container>
6         <v-row dense>
7           <v-col v-for="personaje in personajes" :key="personaje.id"
8 cols="12" xl="4" lg="4" md="4">
9             <v-card color="blue darken-4" dark>
10              <div class="d-flex flex-no-wrap justify-space-
11 between">
12                <div>
13                  <v-card-title class="text-h5" v-
14 text="personaje.name"></v-card-title>
15                  <v-card-subtitle v-text="personaje.status"></v-card-subtitle>
16                  <v-btn class="ml-2 mt-5" outlined rounded
17 small>VER DETALLES</v-btn>
18                </div>
19                <v-avatar class="ma-3" size="125" tile>
20                  <v-img :src="personaje.image"></v-img>
21                </v-avatar>
22              </div>
23            </v-card>
24          </v-col>
25        </v-row>
26      </v-container>
27    </v-card>
28  </div>
29 </template>
30
```

Levantomos nuestro proyecto, nos dirigimos al navegador, y podemos ver que nos trae información desde la **API**, correctamente.

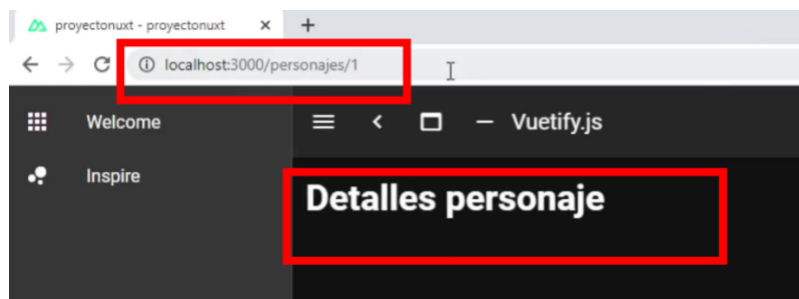
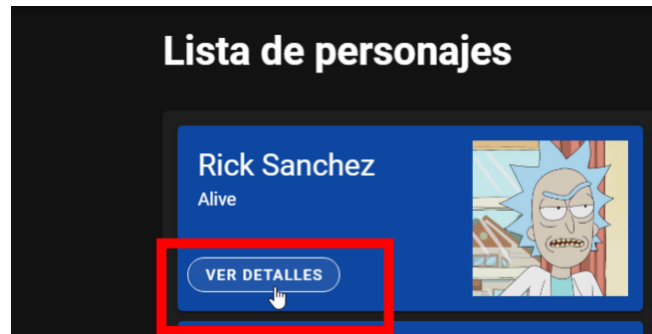


Ahora, le daremos funcionalidad al botón “Ver detalles”, para que al presionarlo podamos ver la descripción del personaje que queremos. Para esto, necesitamos una ruta dinámica que se linkee al **id** del personaje. Entonces, agregaremos la ruta al botón, y le concatenaremos el **id** correspondiente, colocaremos **+personaje.id**, quedando de la siguiente forma:

```
<div>
  <v-card-title class="text-h5" v-text="personaje.name"></v-card-title>
  <v-card-subtitle v-text="personaje.status"></v-card-subtitle>
  <v-btn class="m1-2 mt-5" :to="'/personajes/' + personaje.id" outlined rounded small>VER DETALLES</v-btn>
</div>
<v-avatar class="ma-3" size="125" tile>
  <v-img :src="personaje.image"></v-img>
</v-avatar>
```

Revisaremos el navegador, nos trae el **id** del personaje 1 que es donde presionamos.





Una vez comprobado que todo funciona bien, debemos agregar contenido en la página donde queremos mostrar la descripción de cada personaje. Vamos al archivo `_idPersonaje`, y aquí añadiremos una estructura para mostrar cada detalle, utilizaremos un componente `v-img` de **Vuetify** que contiene un texto, copiamos el código que deseamos utilizar, y al `v-row` le agregamos la clase `"justify-center"`.

```

1 <template>
2   <div>
3     <h1>Detalles personaje</h1>
4     <v-container class="fill-height mt-5" fluid style="min-height:
5 434px">
6       <v-row class="justify-center">
7
8       </v-row>
9
10    </v-container>
11  </div>
12 </template>

```

Ahora, es aquí donde haremos la petición a un solo elemento. Capturamos el número del `id` desde la `URL`; por ejemplo: si pongo `/personajes/2` o `3`, debemos capturar esos números, para eso,

agregamos las etiquetas `script`, crearemos la función `data ()` que retornará un objeto, y lo que queremos obtener será a un personaje como un `string` vacío.

```
1 export default {
2   data() {
3     return {
4       personaje: '',
5     };
6   },
7 };
```

Después, crearemos la función `created ()`, tal como lo hicimos en `index`, importaremos `axios`, incluiremos `async` a dicha función, y desarrollaremos la constante `response` con la petición `GET` y la `URL` a la que se hará, pero esta vez tenemos que agregar el `id`.

```
1 import axios from "axios";
2 export default {
3   data() {
4     return {
5       personaje: '',
6     };
7   },
8   async created() {
9     const response = await
10    axios.get("https://rickandmortyapi.com/api/character/");
11    this.personaje = response.data;
12  },
13 };
```

¿Cómo obtenemos ese `id`? de la misma `URL`, y para esto, vamos a concatenarla con el `id`.

```
1 import axios from "axios";
2 export default {
3   data() {
4     return {
5       personaje: '',
6     };
7   },
8   async created() {
9     const response = await
10    axios.get("https://rickandmortyapi.com/api/character/" +
11    this.$route.params.idPersonaje);
```

```

12   this.personaje = response.data;
13 },
14 };
15
16
17
18
19
  
```

El `params` hace referencia a los parámetros que le estamos pasando en la `URL`, y el archivo `idPersonaje` es el nombre que le dimos al número de `id`, y guardaremos en `personaje` la respuesta que nos traiga la petición.

Una vez obtenidos los datos de un solo personaje, empezaremos a pintar su información en el `template`. En la etiqueta `v-image`, agregaremos: `:src= personaje.image`, con un `max-height` de `400`, y un color `Brown darken-3`.

```

<v-card>
  <v-img :src="personaje.image" max-height="400" contain class="brown darken-3"></v-img>
  <v-card-title class="text-h4 pb-6">{{personaje.name}}</v-card-title>
  <v-card-subtitle class="text-h6 pb-0">{{personaje.status}}</v-card-subtitle>
  <v-card-subtitle class="text-h6 pb-0">{{personaje.gender}}</v-card-subtitle>
  <v-card-subtitle class="text-h6 pb-0">{{personaje.species}}</v-card-subtitle>
</v-card>
  
```

Al `v-card-title`, le pondremos el tamaño de letra `h4` y un `pb-6`, y entre estas etiquetas agregaremos `name: {{personaje.name}}`, así como la etiqueta `v-card-subtitle` con una clase `pb-0`, `text-h6`, y la tendremos 3 veces; en la primera pondremos `Status`, `Gender`, y en la última `Species`.

```
1 <template>
2   <div>
3     <h1>Detalles personaje</h1>
4     <v-container class="fill-height mt-5" fluid style="min-height:
5 434px">
6       <v-row class="justify-center">
7         <v-col cols="6">
8           <v-card>
9             <v-img :src="personaje.image" max-height="400" contain
10 class="brown darken-3"></v-img>
11             <v-card-title class="text-h4 pb-6">
12               Name: {{personaje.name}}
13             </v-card-title>
14             <v-card-subtitle class="pb-0 text-h6">
15               Status: {{personaje.status}}
16             </v-card-subtitle>
17             <v-card-subtitle class="pb-0 text-h6">
18               Gender: {{personaje.gender }}
19             </v-card-subtitle>
20             <v-card-subtitle class="text-h6">
21               Species: {{personaje.species }}
22             </v-card-subtitle>
23           </v-card>
24         </v-col>
25       </v-row>
26     </v-container>
27   </div>
28 </template>
```

Nos dirigimos al navegador, y si presionamos el botón “Ver detalle”, se desplegará la información del personaje seleccionado. De esta manera, hemos concluido la creación de un proyecto con **NUXTJS**.

