

## Proyectos de React: Rick and Morty API

Este documento presenta una colección de ejercicios prácticos realizados en React, enfocados en la creación de componentes, el manejo del estado, el enrutamiento con React Router y la interacción con una API externa. Se trata de un ejercicio práctico para produndizar en el curso de programación de Servidores Web. En este ejercicio, usaremos la API de [Rick and Morty](#). Cada ejercicio está diseñado para ayudar a mejorar las habilidades de programación y el entendimiento de conceptos clave en el desarrollo web con React.

Antes de realizar este ejercicio, se recomienda ver los siguientes videos en YouTube con formación inicial de React:

- a) Curso React JS – Aprende REACT desde cero

[https://www.youtube.com/watch?v=T\\_j60n1zgu0&list=PLV8x\\_i1fqBw0B008sQn79YxCjkHJU84pC&index=1](https://www.youtube.com/watch?v=T_j60n1zgu0&list=PLV8x_i1fqBw0B008sQn79YxCjkHJU84pC&index=1)

- b) Crea una app con React usando create-react-app:

[https://www.youtube.com/watch?v=QBLbXgeXMU8&list=PLV8x\\_i1fqBw0B008sQn79YxCjkHJU84pC&index=2](https://www.youtube.com/watch?v=QBLbXgeXMU8&list=PLV8x_i1fqBw0B008sQn79YxCjkHJU84pC&index=2)

Si deseas ver el código fuente de estos ejercicios, puedes acceder al repositorio en GitHub haciendo clic en el siguiente enlace.

<https://github.com/agorosti/React-Rick-Morty-Exercise>

### **1) Crear un Nuevo Proyecto en React:**

- a) **Instalar Node.js:** Asegúrate de tener instalado Node.js en tu máquina. Puedes descargarlo desde [aquí](#).
- b) **Crear un Proyecto de React:** Abre tu terminal y ejecuta el siguiente comando para crear un nuevo proyecto de React:

***npx create-react-app rick-and-morty-lab***

Esto creará una nueva carpeta llamada ***rick-and-morty-lab*** con todos los archivos necesarios para un proyecto de React.

- c) **Acceder a la Carpeta del Proyecto:** Navega a la carpeta creada:

***cd rick-and-morty-lab***

## 2) *Instalar Dependencias necesarias:*

Para trabajar con la *Rick and Morty API* y manejar el estado, necesitarás instalar algunas dependencias adicionales.

- a) **Instalar Axios:** Axios es una biblioteca para realizar solicitudes HTTP. Instálala usando npm:

***npm install axios***

- b) **Instalar React Router** (opcional, si quieres manejar rutas):

***npm install react-router-dom***

Este permite definir la navegación entre diferentes componentes o páginas de tu aplicación.

- **Router:** Envuelve tu aplicación y habilita la funcionalidad de enrutamiento.
  - **Routes:** Contiene todas las rutas de tu aplicación.
  - **Route:** Define una ruta específica.
    - path: La URL de la ruta.
    - element: El componente que se renderiza en esa ruta.
    - :id: Parámetro dinámico que puedes usar en el componente (como para detalles de recetas).
- c) Asegurate que tu archivo *package.json* tenga el script correcto para iniciar el servidor.

The screenshot shows the VS Code interface with the following details:

- EXPLORADOR**: Shows the project structure: **RICK-AND-MORTY-LAB** (expanded) containing **node\_modules**, **package-lock.json**, **package.json**, and **yarn.lock**.
- package.json** tab is active, displaying the contents of the **package.json** file.
- package.json** content (lines 1-33):

```
1  {
2    "name": "rick-and-morty-lab",
3    "version": "0.1.0",
4    "private": true,
5    "dependencies": {
6      "axios": "^1.7.7",
7      "react": "^18.3.1",
8      "react-dom": "^18.3.1",
9      "react-router-dom": "^6.27.0"
10     },
11    "devDependencies": {
12      "react-scripts": "5.0.1"
13    },
14    "scripts": {
15      "start": "react-scripts start",
16      "build": "react-scripts build",
17      "test": "react-scripts test",
18      "eject": "react-scripts eject"
19    },
20    "browserslist": {
21      "production": [
22        ">0.2%",
23        "not dead",
24        "not op_mini all"
25      ],
26      "development": [
27        "last 1 chrome version",
28        "last 1 firefox version",
29        "last 1 safari version"
30      ]
31    }
32  }
33 }
```

Ahora que tienes la base de tu aplicación React creada, el siguiente paso **es modularizarla y dividirla en secciones**, organizando el código para que sea fácil de mantener y ampliar. Lo haremos paso a paso.

Vamos a dividir la aplicación en **módulos**, como **CharacterList** (componente para mostrar una lista de personajes de Rick and Morty). Cada módulo tendrá su propio componente, y los integraremos en la aplicación principal (**App.js**).

### 3) *Estructura del Proyecto*

1. **Carpetas Principales:** *rick-and-morty-lab*.

- Contendrá todos los archivos y carpetas de tu proyecto.

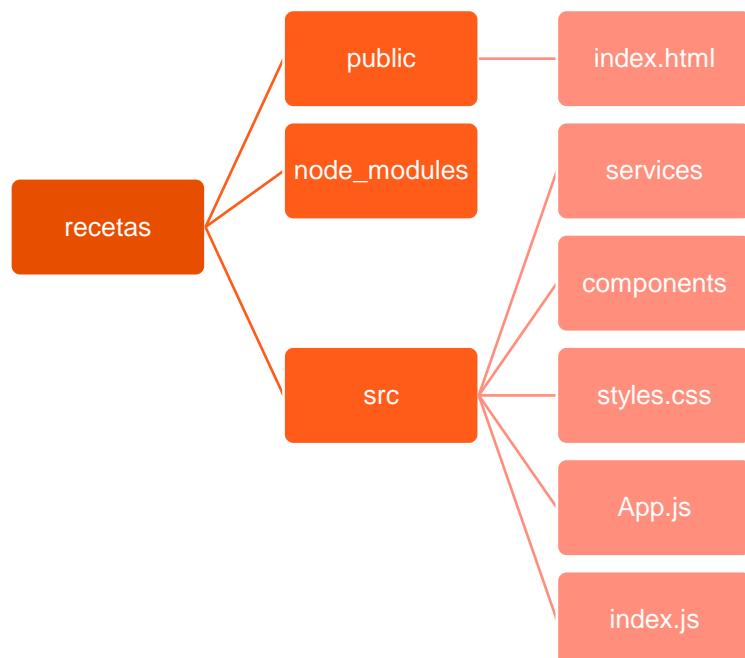
2. **Carpetas src:** Dentro de la carpeta principal vamos a crear la **carpeta src**, donde se guardan los archivos principales de tu aplicación.

- **Crea las carpetas components**
- **Crea la carpeta services.**
- **Crea el archivo App.js.**
- **Crea el archivo index.js.**
- **Crea el archivo styles.css**

3. **Carpetas public:**

- Se encuentra el archivo **index.html**.

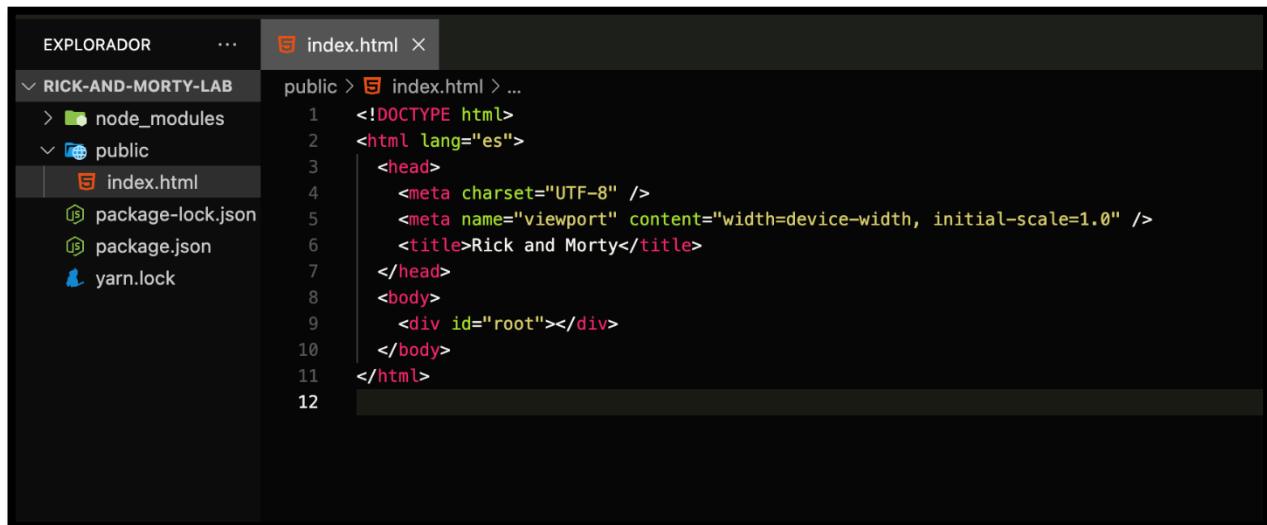
4. **Se debe crear la siguiente estructura:**



#### 4) **Paso a Paso para Crear la Estructura:**

##### 1. **index.html :**

Este archivo es el punto de entrada de tu aplicación web. Define la estructura básica de la página y contiene el elemento donde se montará tu aplicación React.

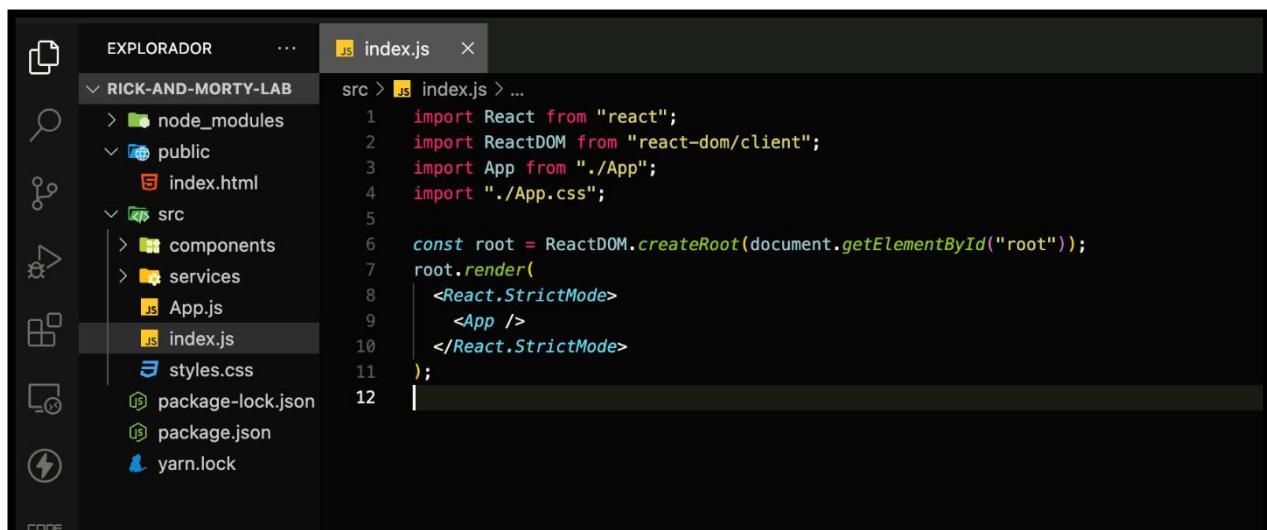


The screenshot shows the VS Code interface with the 'EXPLORADOR' (Explorer) view on the left and the code editor on the right. The project folder 'RICK-AND-MORTY-LAB' is expanded, showing 'node\_modules', 'public', and 'package-lock.json', 'package.json', 'yarn.lock'. The 'index.html' file is selected in the Explorer and is open in the editor. The code in the editor is:

```
public > index.html > ...
1   <!DOCTYPE html>
2   <html lang="es">
3     <head>
4       <meta charset="UTF-8" />
5       <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6       <title>Rick and Morty</title>
7     </head>
8     <body>
9       <div id="root"></div>
10    </body>
11   </html>
12
```

## 2. *index.js*.

Este archivo es el punto de entrada de tu aplicación React. Aquí es donde *ReactDOM* renderiza el componente principal (App) en el DOM. Utiliza *React.StrictMode* para ayudar a detectar problemas en la aplicación.



The screenshot shows the VS Code interface with the 'EXPLORADOR' (Explorer) view on the left and the code editor on the right. The project folder 'RICK-AND-MORTY-LAB' is expanded, showing 'node\_modules', 'public' (with 'index.html'), 'src' (with 'components', 'services', 'App.js', 'index.js', 'styles.css'), and 'package-lock.json', 'package.json', 'yarn.lock'. The 'index.js' file in the 'src' directory is selected in the Explorer and is open in the editor. The code in the editor is:

```
src > index.js > ...
1 import React from "react";
2 import ReactDOM from "react-dom/client";
3 import App from "./App";
4 import "./App.css";
5
6 const root = ReactDOM.createRoot(document.getElementById("root"));
7 root.render(
8   <React.StrictMode>
9     <App />
10   </React.StrictMode>
11 );
12
```

*React.StrictMode* es una herramienta útil en React que ayuda a identificar problemas potenciales en la aplicación. Se utiliza principalmente durante el desarrollo y no afecta el funcionamiento de la aplicación en producción.

## 3. *App.js*

Es el corazón de tu aplicación. En este archivo, vamos a definir los componentes principales que verán los usuarios, como el **Header**, el **CharacterList** y el **Footer**. Por ahora, vamos a comenzar con algo sencillo

- **Imports:**

- ***import React from 'react';***: Importa React para poder usar JSX.
- ***Import App.css***: Importa los estilos.

- **Estructura del Componente:**

- ***function App() { ... }***: Define el componente funcional App.
- Dentro del componente, se devuelve JSX, que es una mezcla de HTML y JavaScript

- **Encabezado:**

- Se utiliza un **<header>** para mostrar el título de la aplicación.

- **Cuerpo:**

- Se utiliza un **<main>** para mostrar el contenido de la pagina, en este caso nuestro componente **<CharacterList />** , que crearemos a continuación.

- **Pie de Página:**

- Se incluye un **<footer>** donde podemos poner cualquier tipo de información como el nombre de los desarrolladores o algún mensaje..

```
src > App.js > ...
1 import React from "react";
2 import CharacterList from "./components/CharacterList";
3
4 Qodo Gen: Options | Test this function
5 function App() {
6   return (
7     <div>
8       <header>
9         <h1>Rick and Morty Characters</h1>
10      </header>
11      <main>
12        <CharacterList />
13      </main>
14      <footer>
15        <p>Desarrollado por [Tu Nombre]</p>
16      </footer>
17    </div>
18  );
19
20 export default App;
21 |
```

4. **Styles.css**: puedes añadir las reglas CSS que deseas Si necesitas aplicar estilos generales a toda la aplicación (resetear estilos, fuentes, colores base). Que se rerenderizará en **App.js**.

Aca un ejemplo de algunos estilos que puedes agregar a tu página.

```

EXPLORADOR ... styles.css
RICK-AND-MORTY-LAB
  node_modules
  public
    index.html
  src
    components/C...
    services
      rickAndMorty...
      App.js
      index.js
    styles.css
  package-lock.json
  package.json
  yarn.lock

CODE GPT
  header {
    font-family: "Open Sans", sans-serif;
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    min-height: 100vh;
    background-color: #f7f7f7; /* Fondo claro */
  }

  .app-container {
    flex: 1;
    display: flex;
    flex-direction: column;
    align-items: center;
  }

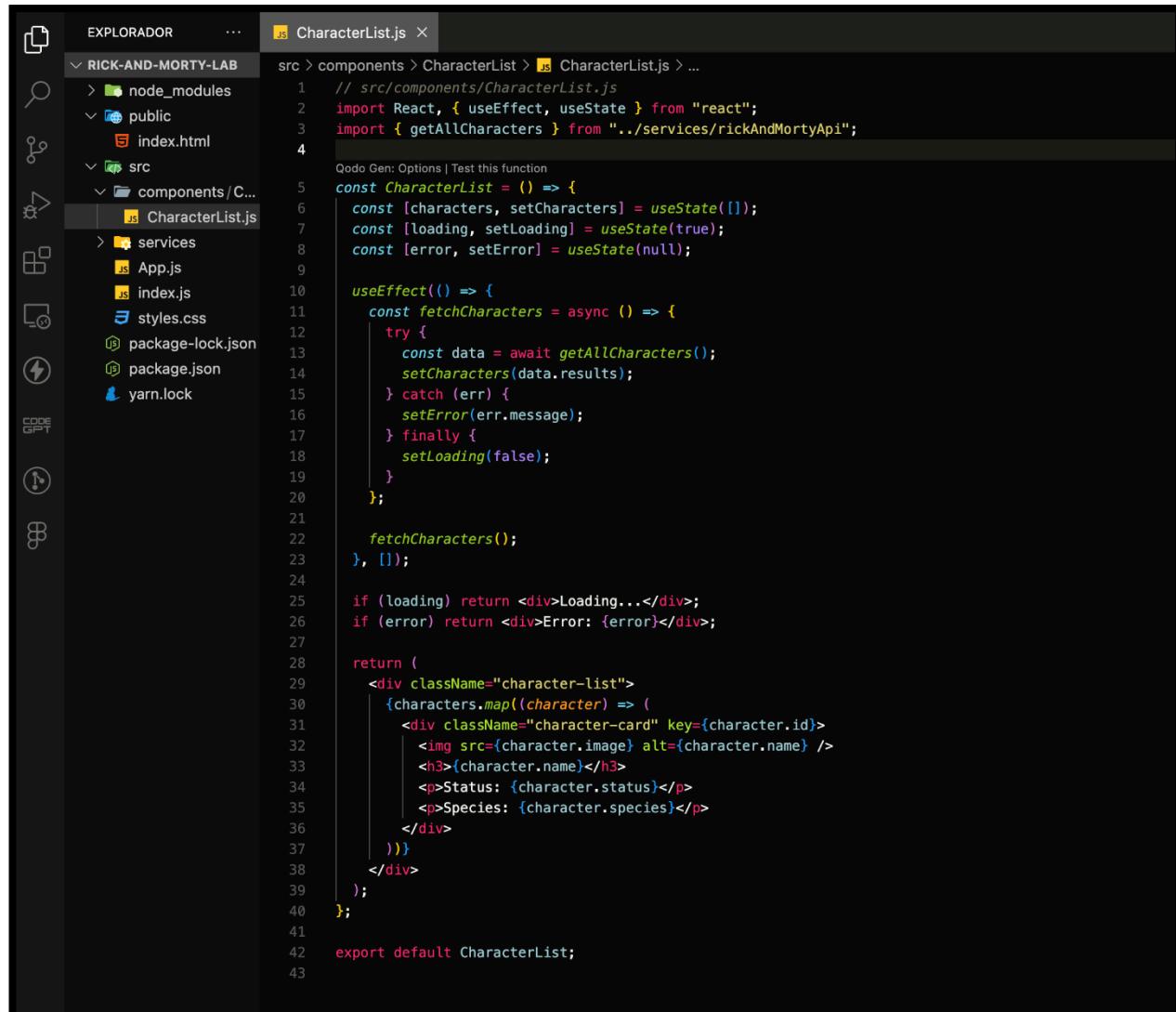
  footer {
    background-color: #02381c;
    color: #fff;
    padding: 20px 0;
    text-align: center;
    font-size: 14px;
    box-shadow: 0 -2px 10px rgba(0, 0, 0, 0.2);
    width: 100%; /* Asegura que ocupe el ancho completo */
    position: relative;
  }

  /* Estilos globales */
  h1,
  h2,
  h3,
  p {
    margin: 0 0 10px; /* Margen inferior para separación */
    align-items: center;
  }

```

5) **Crear el Componente CharacterList.js:** El componente **CharacterList** es el encargado de mostrar una lista de imágenes de personajes de Rick and Morty.

- **useState([]):** Guardamos una lista vacía de perros al principio.
- **useEffect(() => {}, []):** Esta función se ejecuta una vez cuando el componente se carga por primera vez. En este caso, estamos llamando a la **API** para obtener los personajes de Rick and Morty.
- **getAllCharacters():** Es la función que obtiene una lista de personajes de **Rick and Morty** (la crearemos a continuación en **rickAndMortyApi.js**).
- **error handling:** Muestra un mensaje de error si ocurre algún problema al cargar los datos.



```

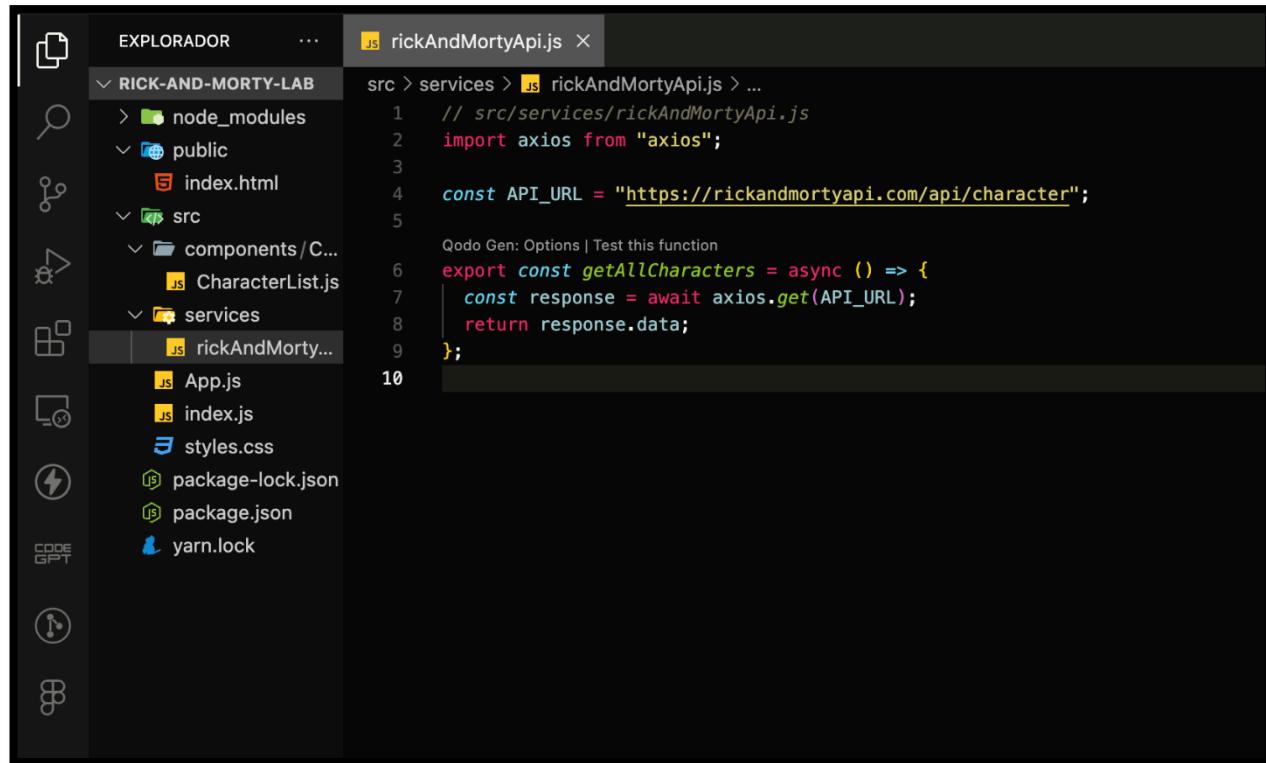
EXPLORADOR ... CharacterList.js
src > components > CharacterList > CharacterList.js > ...
1 // src/components/CharacterList.js
2 import React, { useEffect, useState } from "react";
3 import { getAllCharacters } from "../services/rickAndMortyApi";
4
5 Qodo Gen: Options | Test this function
6 const CharacterList = () => {
7   const [characters, setCharacters] = useState([]);
8   const [loading, setLoading] = useState(true);
9   const [error, setError] = useState(null);
10
11   useEffect(() => {
12     const fetchCharacters = async () => {
13       try {
14         const data = await getAllCharacters();
15         setCharacters(data.results);
16       } catch (err) {
17         setError(err.message);
18       } finally {
19         setLoading(false);
20       }
21     };
22     fetchCharacters();
23   }, []);
24
25   if (loading) return <div>Loading...</div>;
26   if (error) return <div>Error: {error}</div>;
27
28   return (
29     <div className="character-list">
30       {characters.map((character) => (
31         <div className="character-card" key={character.id}>
32           <img src={character.image} alt={character.name} />
33           <h3>{character.name}</h3>
34           <p>Status: {character.status}</p>
35           <p>Species: {character.species}</p>
36         </div>
37       )));
38     </div>
39   );
40
41   export default CharacterList;
42
43

```

6) **Estilos de *CharactersList.css*:** Agrega los estilos que deseas para mejorar tu renderizado.

7) **Crear el Servicio *rickAndMortyApi.js*:** En la carpeta **sevices** creamos el archive ***rickAndMortyApi.js***, este archivo se encarga de conectarse a la API y obtener la información de los personajes.

- ***axios.get***: Hacemos una solicitud a la API para obtener todos los personajes.
- ***getAllCharacters***: Es la función que utilizamos en ***CharacterList.js*** para obtener la lista de personajes.



```
// src/services/rickAndMortyApi.js
import axios from "axios";
const API_URL = "https://rickandmortyapi.com/api/character";
export const getAllCharacters = async () => {
  const response = await axios.get(API_URL);
  return response.data;
};
```

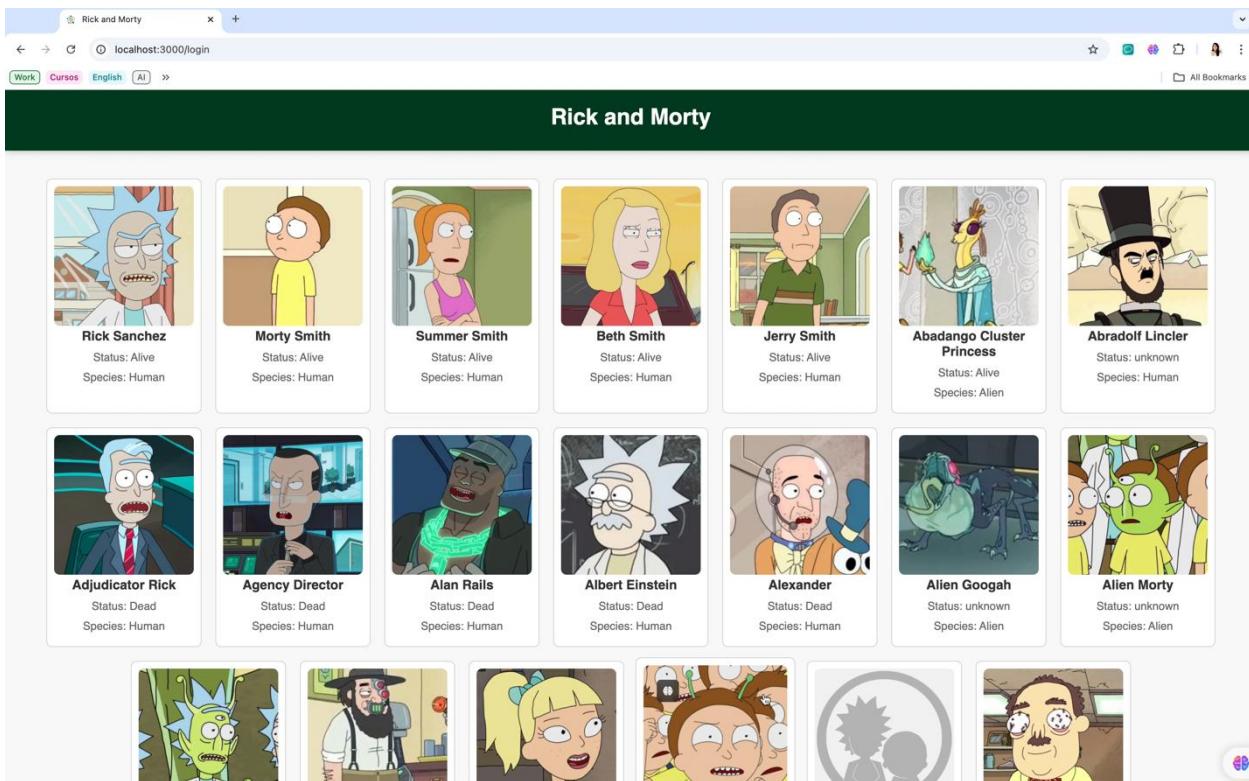
Inicia tu aplicación y deberías poder ver los cambios.

### Para iniciar la aplicación:

- Abre la terminal en la raíz de tu proyecto.
- Ejecuta el comando: **npm start**
- Esto lanzará la aplicación en tu navegador, y deberías ver los cambios aplicados.



Video demostrativo (Haz dole click).

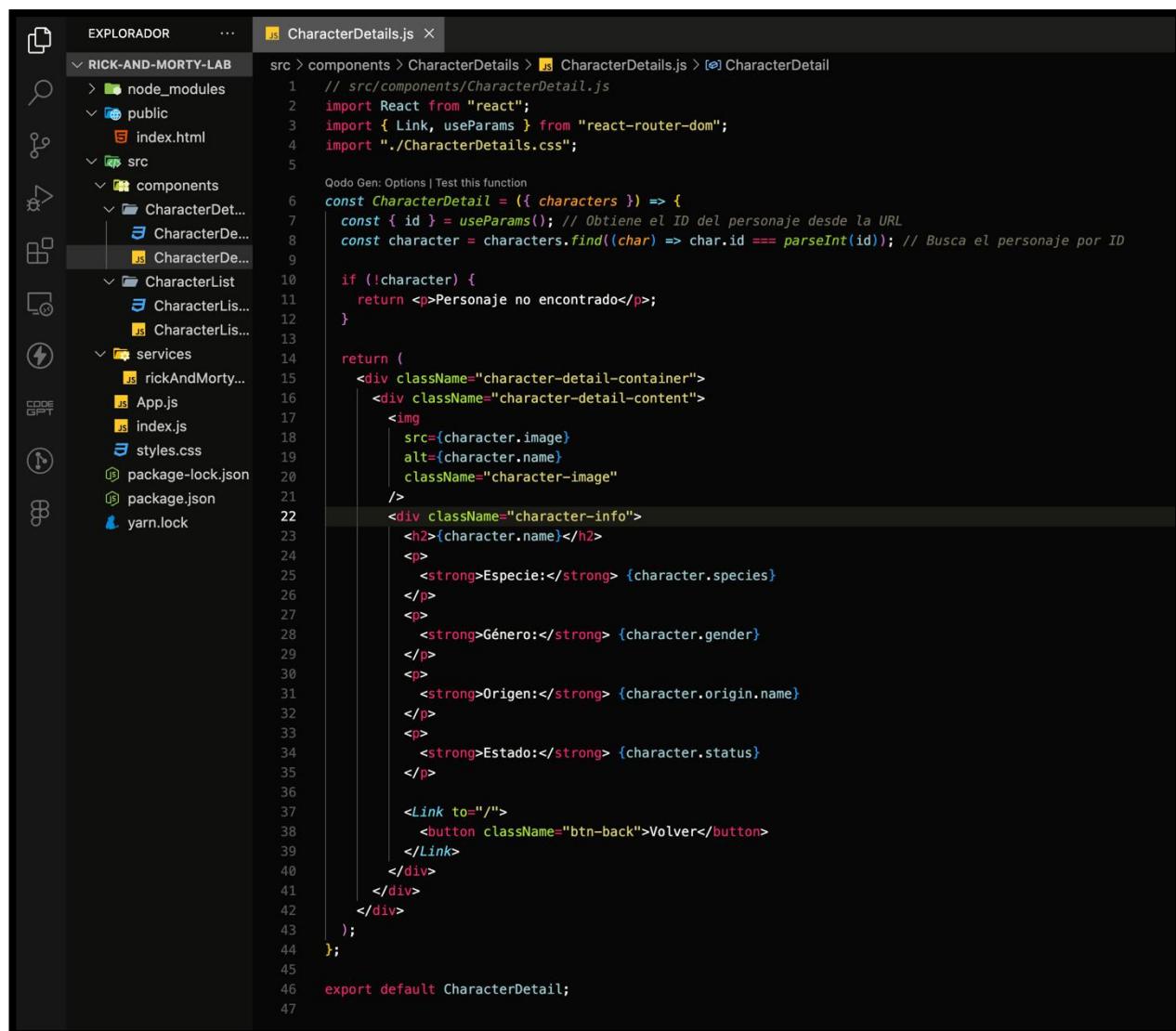


## 8) Crear el Componente *CharacterDetail.js*:

Primero, creamos un nuevo componente llamado *CharacterDetail.js*, que será el encargado de mostrar los detalles de un personaje específico cuando el usuario haga clic en una tarjeta.

### ¿Qué hace este componente?

- Muestra la información detallada de un personaje (imagen, nombre, especie, género, origen, estado).
- Incluye un **botón "Volver"** que lleva al usuario de regreso a la lista de personajes.
- Toma el **ID** del personaje desde la **URL usando useParams** de **react-router-dom**, lo busca en la lista de personajes y luego muestra la información.



```

src > components > CharacterDetails > CharacterDetails.js > CharacterDetail
1 // src/components/CharacterDetail.js
2 import React from "react";
3 import { Link, useParams } from "react-router-dom";
4 import "./CharacterDetails.css";
5
6 Qodo Gen: Options | Test this function
7 const CharacterDetail = ({ characters }) => {
8   const { id } = useParams(); // Obtiene el ID del personaje desde la URL
9   const character = characters.find(char) => char.id === parseInt(id); // Busca el personaje por ID
10
11   if (!character) {
12     return <p>Personaje no encontrado</p>;
13   }
14
15   return (
16     <div className="character-detail-container">
17       <div className="character-detail-content">
18         <img
19           src={character.image}
20           alt={character.name}
21           className="character-image"
22         />
23         <div className="character-info">
24           <h2>{character.name}</h2>
25           <p>
26             <strong>Especie:</strong> {character.species}
27           </p>
28           <p>
29             <strong>Género:</strong> {character.gender}
30           </p>
31           <p>
32             <strong>Origen:</strong> {character.origin.name}
33           </p>
34           <p>
35             <strong>Estado:</strong> {character.status}
36           </p>
37
38           <Link to="/">
39             <button className="btn-back">Volver</button>
40           </Link>
41         </div>
42       </div>
43     );
44   );
45
46 export default CharacterDetail;
47

```

## 9) Crear los Estilos para *CharacterDetail*:

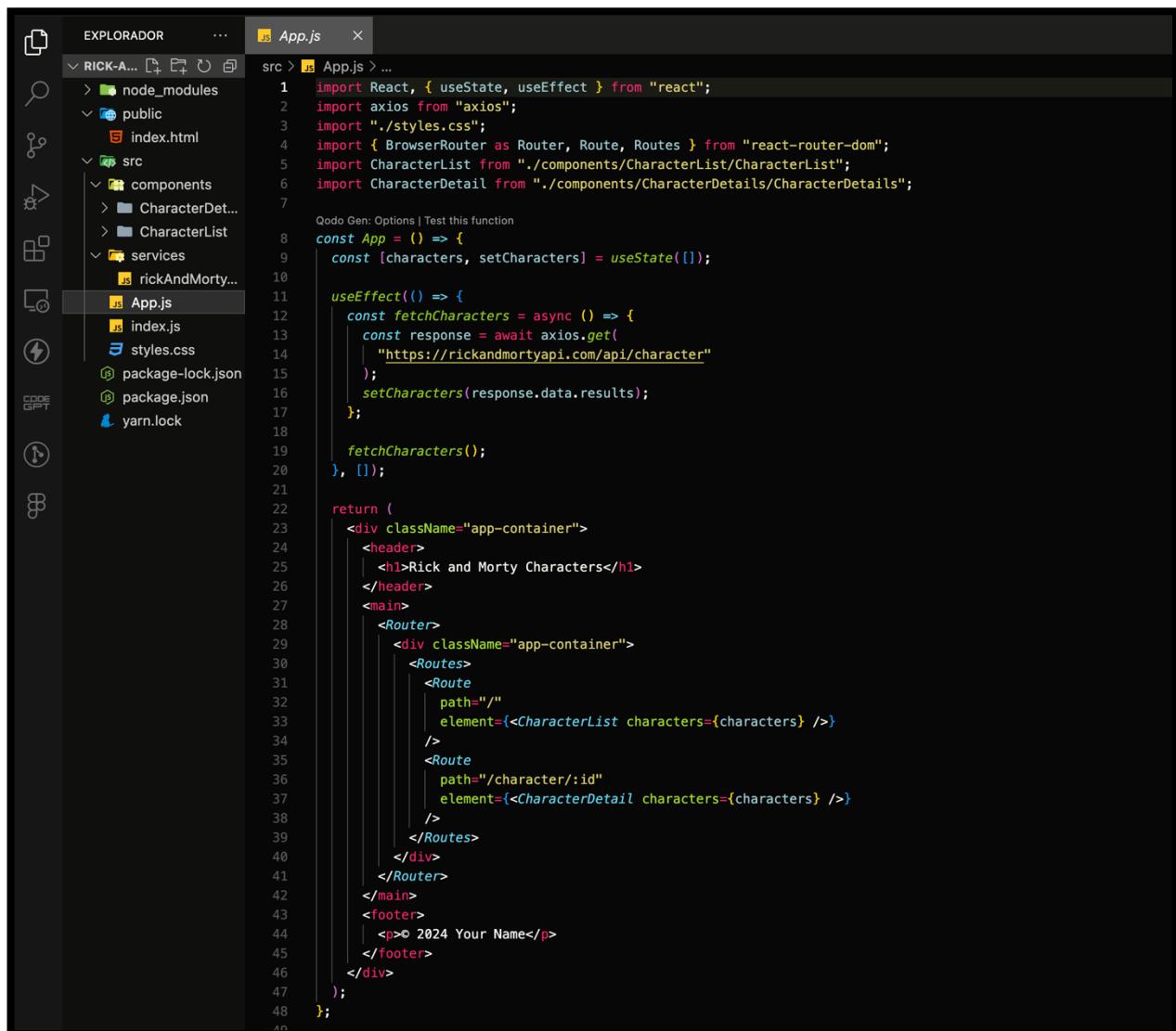
Una vez creado el componente *CharacterDetail*, necesitas agregar estilos específicos para que el diseño sea más atractivo. Creamos la hoja de estilos *CharacterDetail.css* para dar estilo a la página de detalles.

## 10) Definir la Ruta para los Detalles del Personaje:

En el archivo donde defines las rutas principales de la aplicación (en *App.js* o donde uses *react-router-dom*), necesitas agregar una ruta para que cuando el usuario haga clic en un personaje, sea dirigido a la página de detalles.

Pasos para agregar la ruta:

1. Importa el componente **CharacterDetail** en **App.js**.
2. Agrega una nueva ruta que apunte a los detalles del personaje, **usando :id** para pasar el ID del personaje seleccionado



The screenshot shows a code editor interface with the following details:

- File Structure:** The left sidebar shows a project structure with folders like node\_modules, public, and src. Inside src, there are components, services, and a rickAndMorty... folder containing App.js, index.js, styles.css, package-lock.json, package.json, and yarn.lock.
- Code Editor:** The main window displays the content of **App.js**. The code imports React, useState, useEffect from react, axios, and styles.css. It uses react-router-dom to handle routes. The **CharacterDetail** component is imported from components/CharacterDetails/CharacterDetails. The **App** component fetches characters from "https://rickandmortyapi.com/api/character" and sets them to state. It then defines two routes: one for the character list (path '/') and one for character details (path '/:id'). The details route uses the **CharacterDetail** component with the character ID as a prop.

```

    import React, { useState, useEffect } from "react";
    import axios from "axios";
    import "./styles.css";
    import { BrowserRouter as Router, Route, Routes } from "react-router-dom";
    import CharacterList from "./components/CharacterList/CharacterList";
    import CharacterDetail from "./components/CharacterDetails/CharacterDetails";

    const App = () => {
      const [characters, setCharacters] = useState([]);

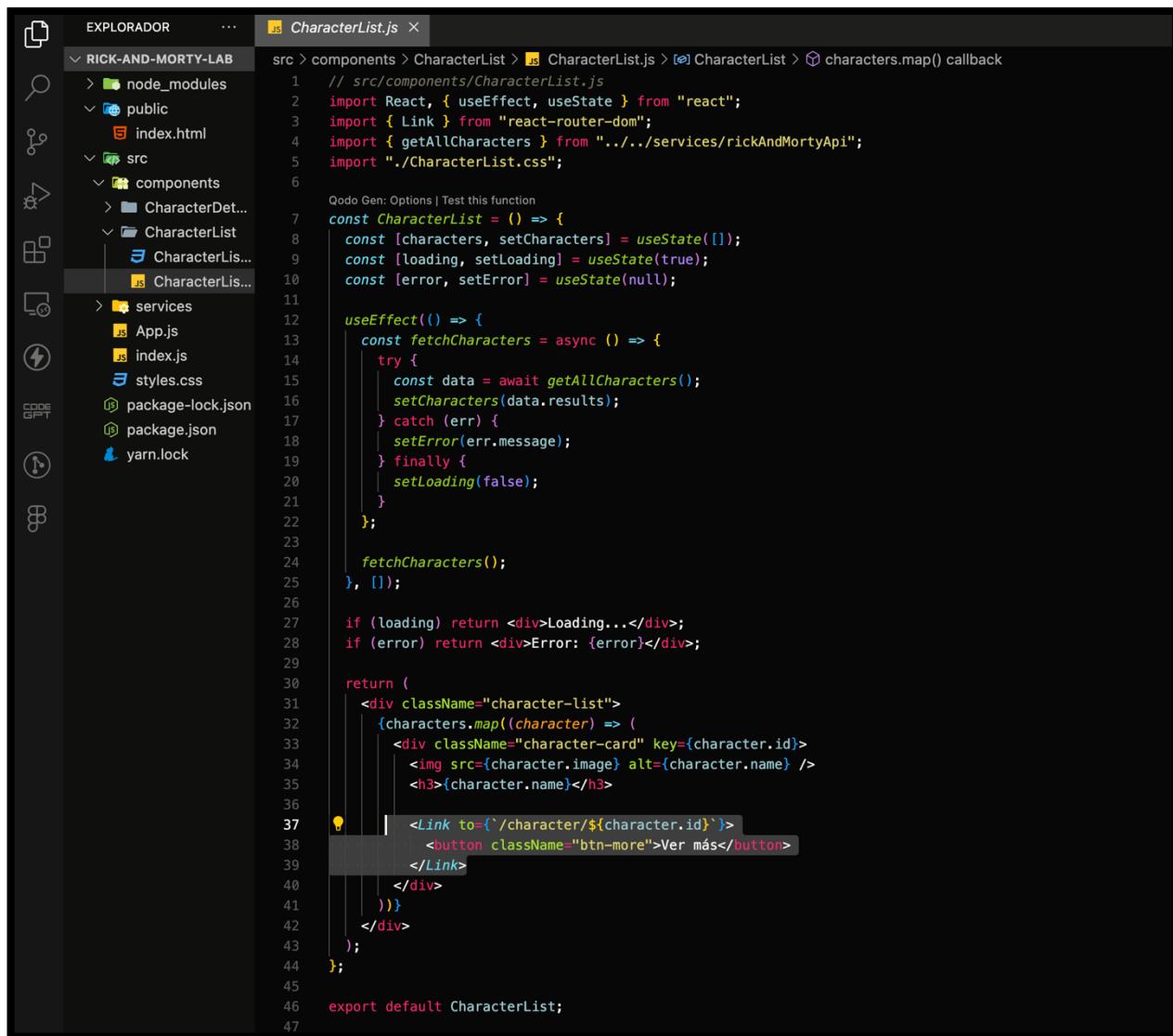
      useEffect(() => {
        const fetchCharacters = async () => {
          const response = await axios.get(
            "https://rickandmortyapi.com/api/character"
          );
          setCharacters(response.data.results);
        };
        fetchCharacters();
      }, []);

      return (
        <div className="app-container">
          <header>
            <h1>Rick and Morty Characters</h1>
          </header>
          <main>
            <Router>
              <div className="app-container">
                <Routes>
                  <Route
                    path="/"
                    element={<CharacterList characters={characters} />}
                  />
                  <Route
                    path="/character/:id"
                    element={<CharacterDetail characters={characters} />}
                  />
                </Routes>
              </div>
            </Router>
          </main>
          <footer>
            <p>© 2024 Your Name</p>
          </footer>
        </div>
      );
    };
  
```

## 11) Hacer que el Botón "Ver Más" Lleve a los Detalles

Ahora, en el componente **CharacterList.js**, necesitas asegurarte de que al hacer clic en un personaje o en un botón "Ver Más", se redirija a la página de detalles.

1. Importa **Link** de react-router-dom en **CharacterList.js**.
2. Añade el **Link** dentro de cada tarjeta para que el botón "Ver Más" o la tarjeta completa lleven a la **ruta /character/{id}**, pasando el ID del personaje seleccionado.



```

EXPLORADOR ... CharacterList.js
src > components > CharacterList > CharacterList.js > CharacterList > characters.map() callback
1 // src/components/CharacterList.js
2 import React, { useEffect, useState } from "react";
3 import { Link } from "react-router-dom";
4 import { getAllCharacters } from "../../services/rickAndMortyApi";
5 import "./CharacterList.css";
6
7 Qodo Gen: Options | Test this function
8 const CharacterList = () => {
9   const [characters, setCharacters] = useState([]);
10  const [loading, setLoading] = useState(true);
11  const [error, setError] = useState(null);
12
13  useEffect(() => {
14    const fetchCharacters = async () => {
15      try {
16        const data = await getAllCharacters();
17        setCharacters(data.results);
18      } catch (err) {
19        setError(err.message);
20      } finally {
21        setLoading(false);
22      }
23    };
24
25    fetchCharacters();
26  }, []);
27
28  if (loading) return <div>Loading...</div>;
29  if (error) return <div>Error: {error}</div>;
30
31  return (
32    <div className="character-list">
33      {characters.map((character) => (
34        <div className="character-card" key={character.id}>
35          <img src={character.image} alt={character.name} />
36          <h3>{character.name}</h3>
37          <Link to={`/character/${character.id}`}>
38            <button className="btn-more">Ver más</button>
39          </Link>
40        </div>
41      ))
42    </div>
43  );
44
45  export default CharacterList;
46
47

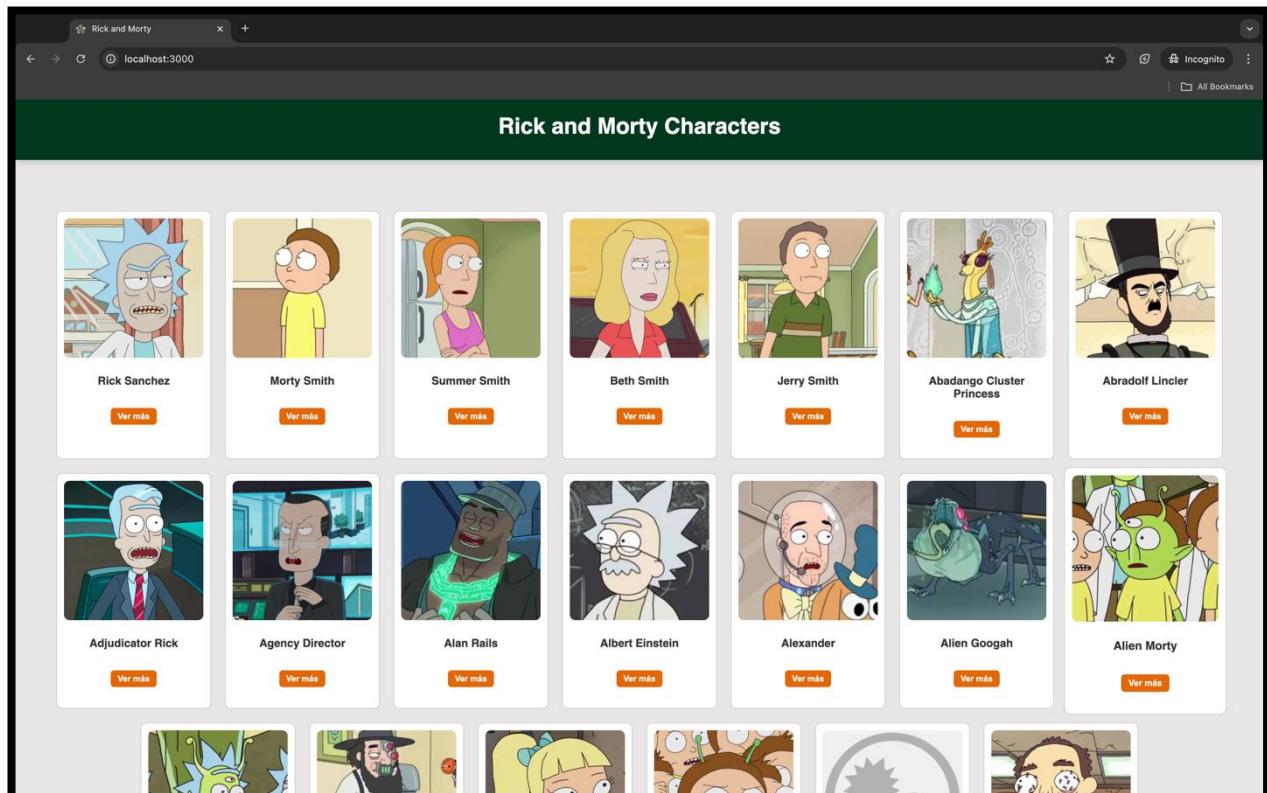
```

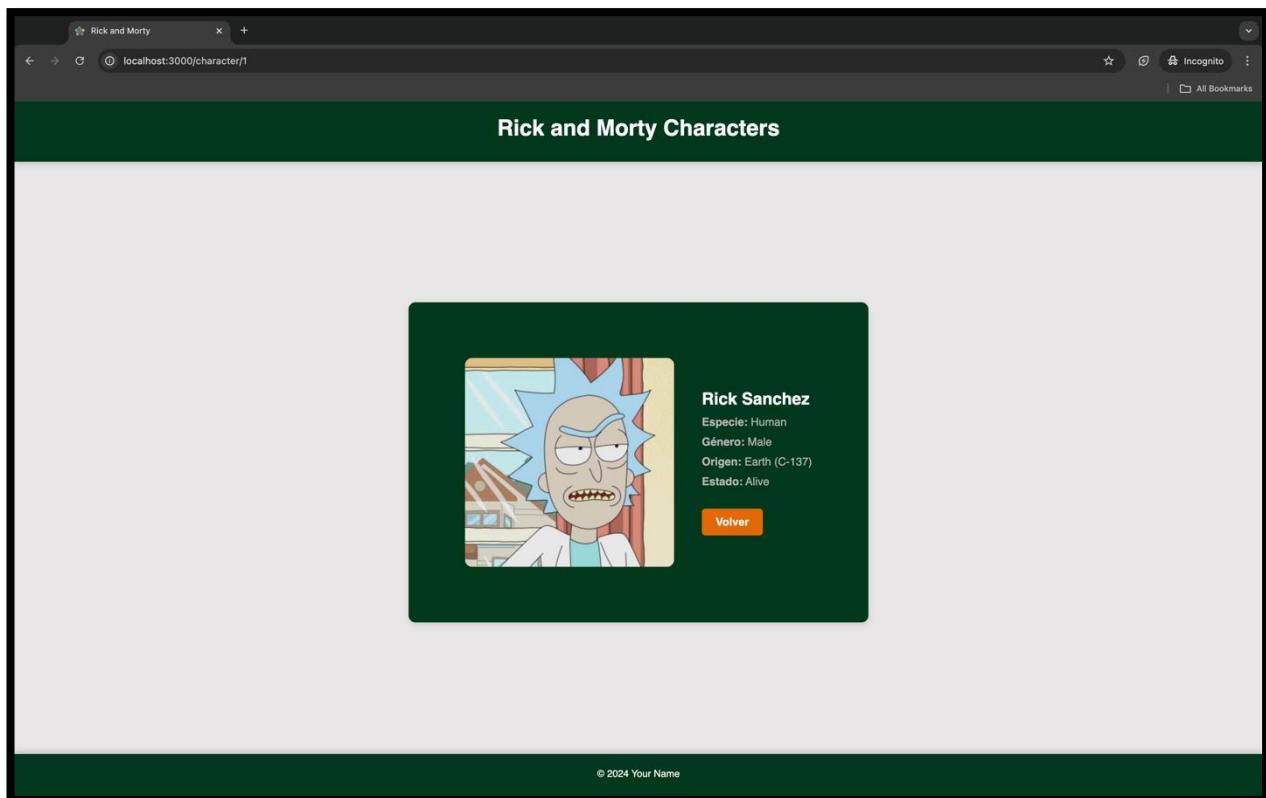
## 12) Verificar la Navegación:

1. Desde la página principal (**CharacterList**), el usuario ve la lista de personajes con un botón "Ver Más" en cada tarjeta.
2. Al hacer **clic en "Ver Más"**, el usuario es redirigido a la página de detalles del personaje correspondiente.

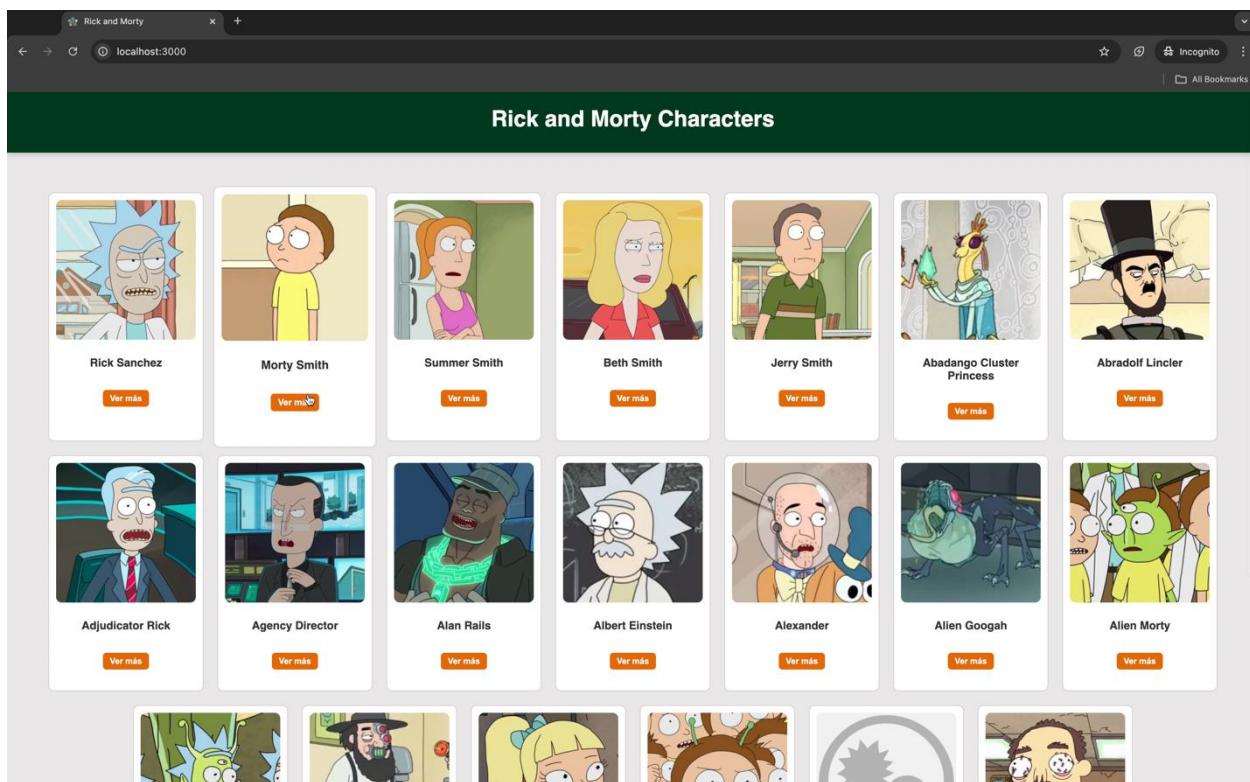
3. En la página de detalles (**CharacterDetail**), el usuario puede ver la imagen y la información del personaje, y el botón "Volver" les lleva de regreso a la lista.

Con estos pasos, ya tienes un flujo completo para mostrar la lista de personajes y los detalles de cada uno de forma organizada y estilizada.



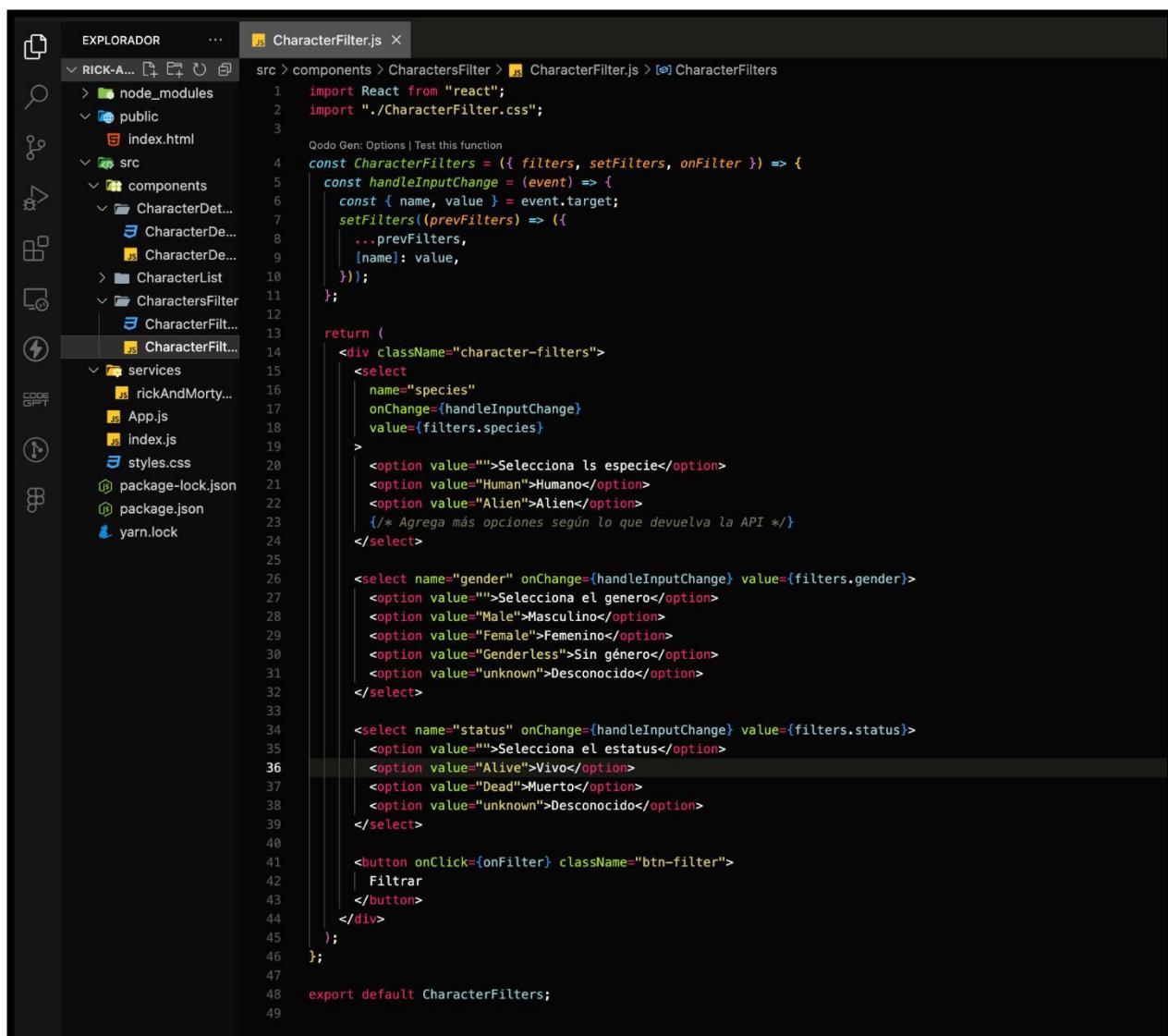


Video demostrativo (Haz dole click).



### 13) Crear el Componente CharacterFilter:

- Crea un archivo nuevo llamado **CharacterFilters.js** dentro de la carpeta de componentes.
- El componente CharacterFilters permite a los usuarios seleccionar diferentes criterios (especie, género, estatus) para filtrar los personajes.
- Contiene tres elementos **<select>**, uno para cada filtro: **species**, **gender**, y **status**.
- Al cambiar la selección en cualquiera de los campos, se actualizan los filtros en el estado de CharacterList usando **setFilters**, lo que mantiene el valor actual de los filtros.
- El botón "Filtrar" dispara la función **onFilter**, que aplica los filtros haciendo una nueva llamada a la API.



```

RICK-A...  EXPLORADOR  ...
src > components > CharactersFilter > CharacterFilter.js > CharacterFilters
  import React from "react";
  import "./CharacterFilter.css";

  const CharacterFilters = ({ filters, setFilters, onFilter }) => {
    const handleInputChange = (event) => {
      const { name, value } = event.target;
      setFilters((prevFilters) => {
        ...prevFilters,
        [name]: value,
      });
    };

    return (
      <div className="character-filters">
        <select
          name="species"
          onChange={handleInputChange}
          value={filters.species}>
          <option value="">Selecciona la especie</option>
          <option value="Human">Humano</option>
          <option value="Alien">Alien</option>
          /* Agrega más opciones según lo que devuelva la API */
        </select>

        <select name="gender" onChange={handleInputChange} value={filters.gender}>
          <option value="">Selecciona el género</option>
          <option value="Male">Masculino</option>
          <option value="Female">Femenino</option>
          <option value="Genderless">Sin género</option>
          <option value="unknown">Desconocido</option>
        </select>

        <select name="status" onChange={handleInputChange} value={filters.status}>
          <option value="">Selecciona el estatus</option>
          <option value="Alive">Vivo</option>
          <option value="Dead">Muerto</option>
          <option value="unknown">Desconocido</option>
        </select>

        <button onClick={onFilter} className="btn-filter">
          Filtrar
        </button>
      </div>
    );
  };

  export default CharacterFilters;

```

#### 14) Modificar el archivo *CharacterList.js*:

- Añade el componente **CharacterFilter** justo antes de mostrar la lista de personajes.
- El componente CharacterList maneja el estado de los personajes y los filtros. También controla la lógica de la obtención de personajes desde la API.
- En CharacterList, el estado filters se inicializa con los valores vacíos para las propiedades de **species**, **gender**, y **status**.
- Al renderizar el componente, se pasa el estado filters y la función setFilters como props a CharacterFilters.

```
useEffect(() => {
  fetchCharacters();
}, []); // Solo se ejecuta una vez al cargar el componente

const applyFilters = () => {
  fetchCharacters();
};

if (loading) return <div>Loading...</div>;
if (error) return <div>Error: {error}</div>;

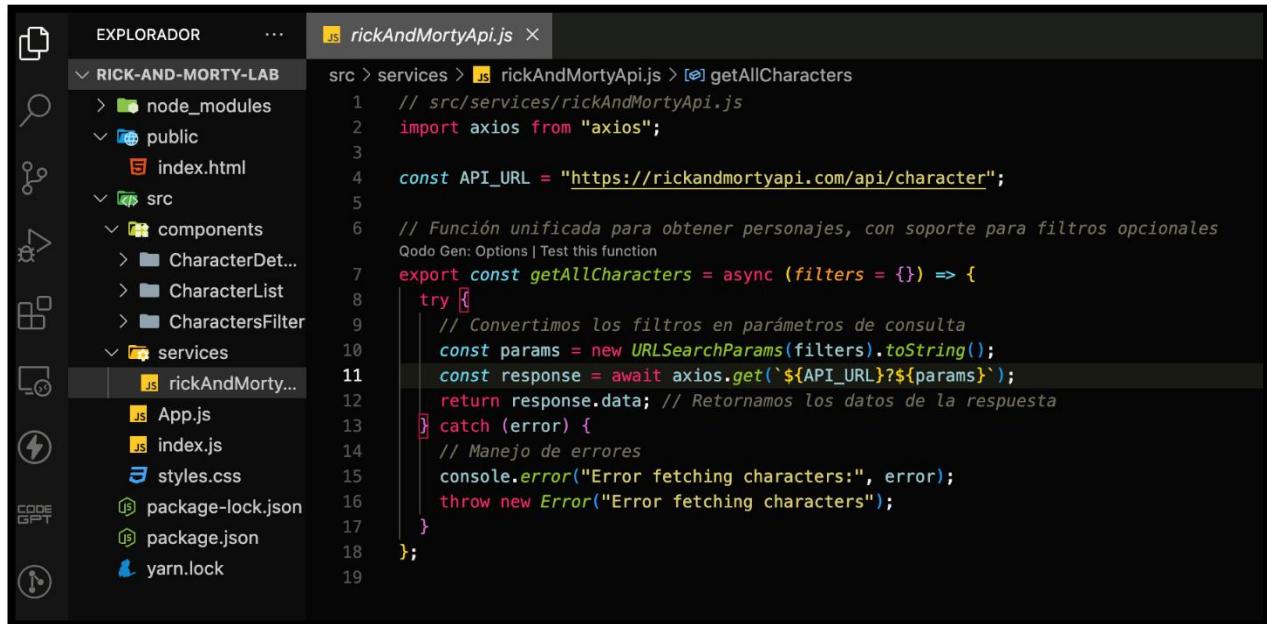
return [
  <div className="character-list">
    <CharacterFilter
      filters={filters}
      setFilters={setFilters}
      onFilter={applyFilters}
    />
```

- **applyFilters** es la función que realiza la llamada a la API con los filtros actuales. Se ejecuta cuando el usuario hace clic en el botón "Filtrar". Llama a **fetchCharacters** con los filtros aplicados.

#### 15) Función para obtener personajes filtrados (*getAllCharacters*)

- En el archivo *rickAndMortyApi.js*, la función getAllCharacters fue modificada para aceptar los filtros como parámetros.

- Los filtros se convierten en una cadena de consulta que se envía a la API de Rick and Morty.



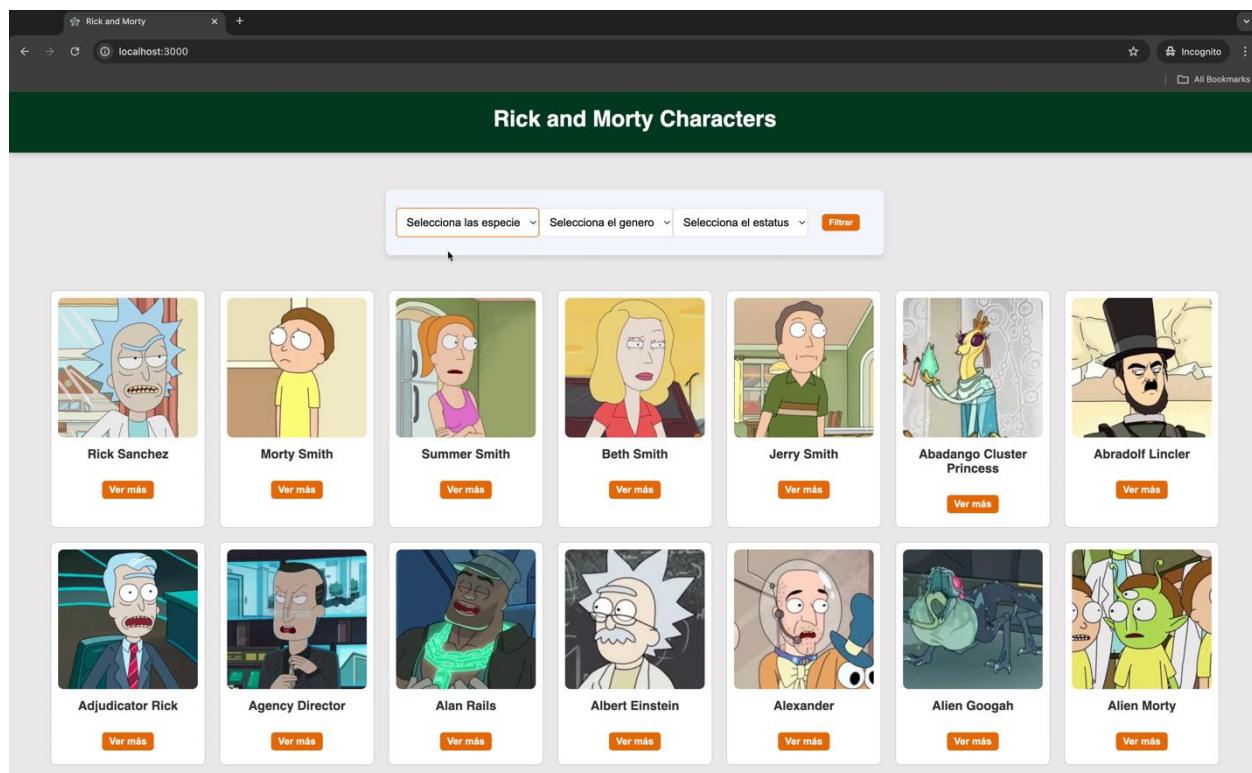
```

EXPLORADOR      rickAndMortyApi.js
RICK-AND-MORTY-LAB
  > node_modules
  > public
    > index.html
  > src
    > components
      > CharacterDetail
      > CharacterList
      > CharactersFilter
    > services
      > rickAndMorty...
        > App.js
        > index.js
        > styles.css
        > package-lock.json
        > package.json
        > yarn.lock

src > services > rickAndMortyApi.js > getAllCharacters
1   // src/services/rickAndMortyApi.js
2   import axios from "axios";
3
4   const API_URL = "https://rickandmortyapi.com/api/character";
5
6   // Función unificada para obtener personajes, con soporte para filtros opcionales
7   Qodo Gen: Options | Test this function
8   export const getAllCharacters = async (filters = {}) => {
9     try {
10       // Convertimos los filtros en parámetros de consulta
11       const params = new URLSearchParams(filters).toString();
12       const response = await axios.get(`${API_URL}?${params}`);
13       return response.data; // Retornamos los datos de la respuesta
14     } catch (error) {
15       // Manejo de errores
16       console.error("Error fetching characters:", error);
17       throw new Error("Error fetching characters");
18     }
19   };

```

Video demostrativo (Haz dole click).



Si deseas para generar una mejor experiencia de usuario, crea un botón que limpie los filtros aplicados, y vuelvan a su estado original.

#### **16) Crear el componente CharacterSearch:**

Este componente tendrá un campo de texto donde los usuarios podrán escribir el nombre de un personaje, y se aplicará la búsqueda cuando hagan clic en el botón o presionen la tecla Enter.

- **Estado (useState):**

- searchTerm es una variable que guarda lo que el usuario escribe en el campo de búsqueda.
- Usamos useState para crear y actualizar searchTerm.

- **Funciones principales:**

- **handleInputChange:** Esta función se ejecuta cada vez que el usuario escribe algo en el campo de texto. Lo que hace es actualizar el valor de searchTerm con el texto que se está escribiendo.
- **handleSearch:** Esta función se ejecuta cuando el usuario hace clic en el botón "Buscar". Lo que hace es llamar a la función onSearch que se le pasó como prop, enviándole el texto que el usuario escribió (el valor de searchTerm).
- **handleKeyDown:** Esta función se ejecuta cuando el usuario presiona una tecla dentro del campo de búsqueda. Si el usuario presiona "Enter", la búsqueda también se ejecuta (llamando a handleSearch)

```

EXPLORADOR ... js CharacterList.js js CharacterSearch.js ×
src > components > CharacterSearch > CharacterSearch.js > CharacterSearch > handleSearch
1 // src/components/CharacterSearch.js
2 import React, { useState } from "react";
3 import "./CharacterSearch.css";
4
5 Qodo Gen: Options | Test this function
6 const CharacterSearch = ({ onSearch }) => {
7   const [searchTerm, setSearchTerm] = useState("");
8
9   const handleInputChange = (event) => {
10     setSearchTerm(event.target.value);
11   };
12
13   const handleSearch = () => {
14     if (searchTerm.trim()) {
15       onSearch(searchTerm); // Llama a onSearch solo si searchTerm no está vacío
16     }
17   };
18
19   const handleKeyDown = (event) => {
20     if (event.key === "Enter") {
21       handleSearch(); // Ejecuta la búsqueda al presionar Enter
22     }
23   };
24
25   return (
26     <div className="character-search">
27       <input
28         type="text"
29         placeholder="Buscar por nombre..."
30         value={searchTerm}
31         onChange={handleInputChange}
32         onKeyDown={handleKeyDown} // Detecta cuando se presiona la tecla Enter
33       />
34       <button onClick={handleSearch} className="btn-search">
35         Buscar
36       </button>
37     );
38   };
39
40   export default CharacterSearch;
41

```

## 17) Integrar el componente de búsqueda en CharacterList.js:

Ahora que tenemos el componente de búsqueda, vamos a integrarlo en la lista de personajes. Modifica el archivo CharacterList.js para agregarlo junto a los filtros.

- **characters:** Almacena la lista de personajes.
- **loading:** Indica si los datos se están cargando.
- **error:** Guarda errores en la carga.
- **filters:** Incluye criterios de filtrado como especie, género, estado y nombre.

### Función fetchCharacters:

- Obtiene personajes de la API utilizando los filtros y actualiza el estado. Maneja errores y muestra un indicador de carga.

**Manejo de Eventos:**

- ***handleSearch***: Actualiza el filtro de nombre y llama a `fetchCharacters` para buscar personajes.
- ***applyFilters***: Se ejecuta al hacer clic en el botón de filtrado para aplicar los filtros seleccionado.

```
const [filters, setFilters] = useState({
  species: '',
  gender: '',
  status: '',
  name: "", // Agrega el campo para nombre
});

const fetchCharacters = async () => {
  setLoading(true);
  try {
    const data = await getAllCharacters(filters); // Usa todos los filtros
    setCharacters(data.results || []);
  } catch (err) {
    setError(err.message);
  } finally {
    setLoading(false);
  }
};

// Maneja la búsqueda de personajes
const handleSearch = (searchTerm) => {
  setFilters((prevFilters) => ({
    ...prevFilters,
    name: searchTerm, // Actualiza solo el nombre en los filtros
  }));
  fetchCharacters(); // Busca personajes después de actualizar el filtro
};

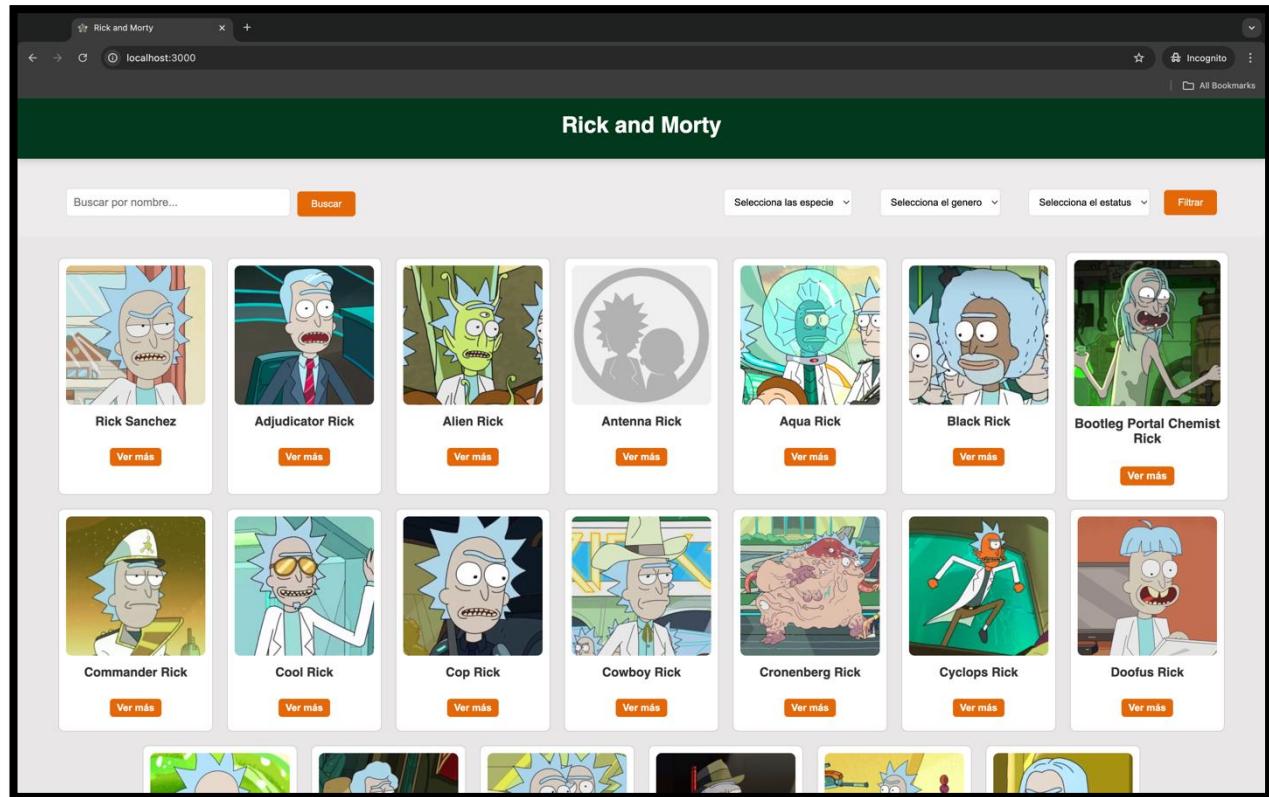
// Aplica los filtros al hacer clic en el botón
const applyFilters = () => {
  fetchCharacters(); // Solo se llama cuando el botón de filtro se presiona
};

useEffect(() => {
  fetchCharacters(); // Llama a fetchCharacters cuando se monta el componente
}, []); // Solo carga los personajes inicialmente

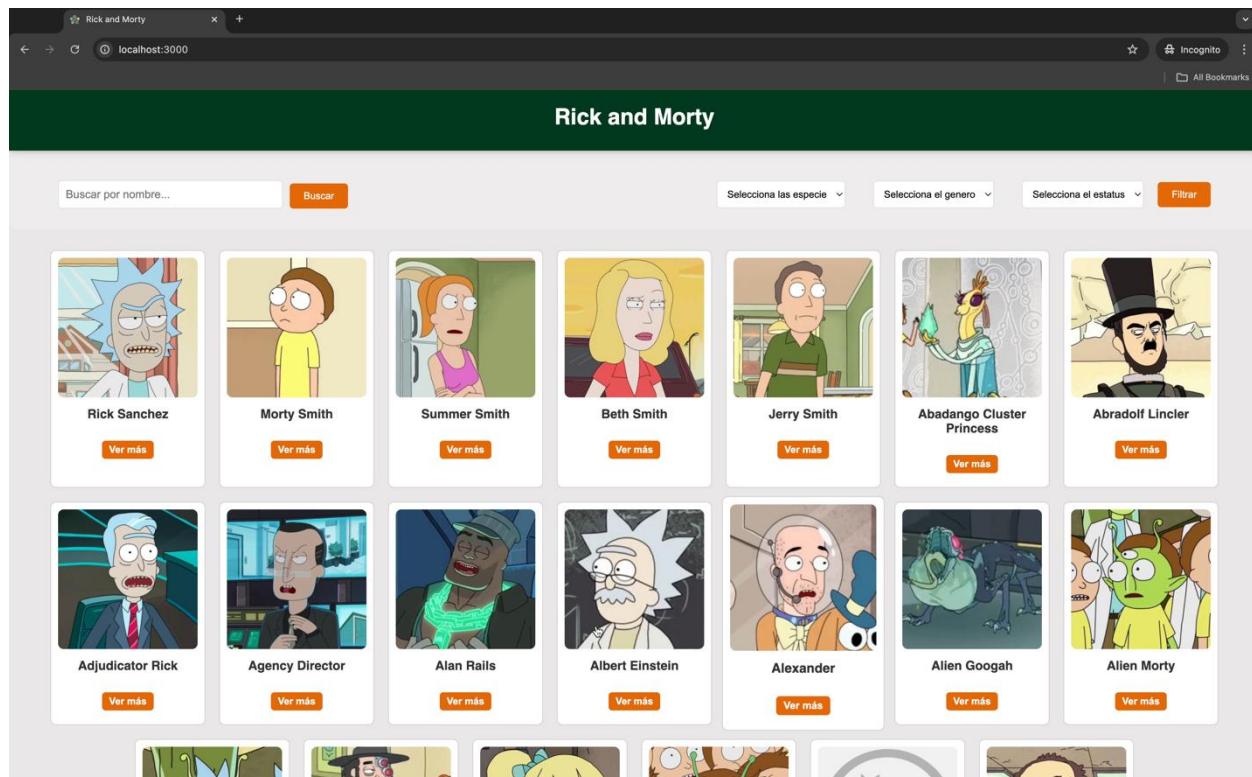
if (loading) return <div>Loading...</div>;
if (error) return <div>Error: {error}</div>;

return (
  <div>
    <div className="character-controls">
      <CharacterSearch onSearch={handleSearch} className="character-search" />{" "}
    
```

**18)** Agregar tus archivos css a cada componente para agregar estilos a tu página.



Video demostrativo (Haz dole click).

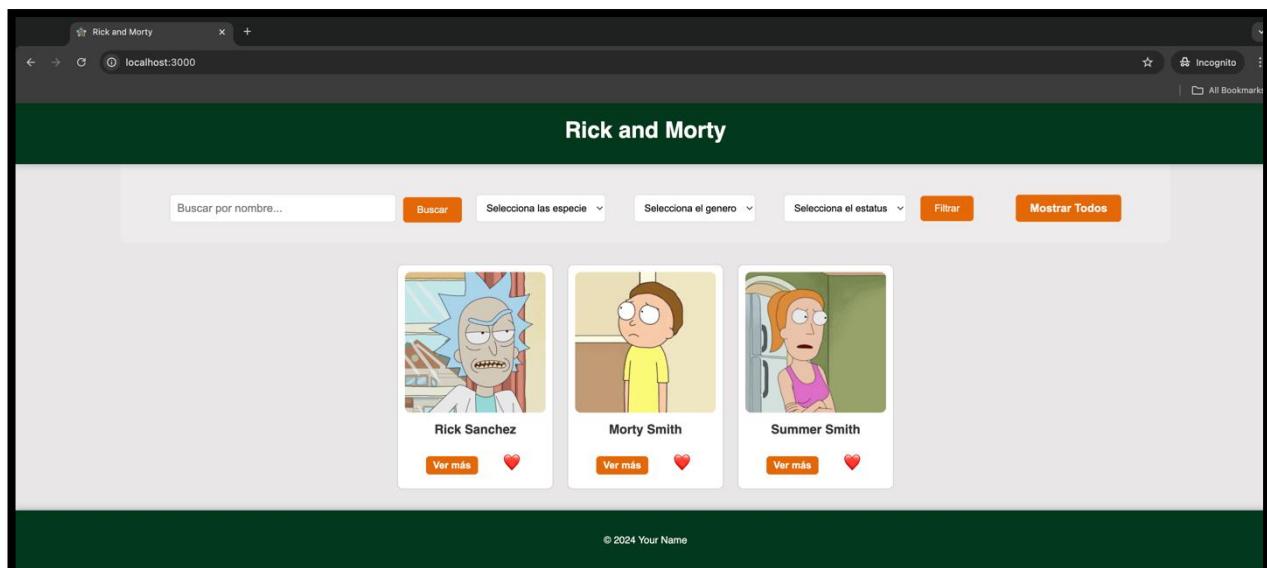
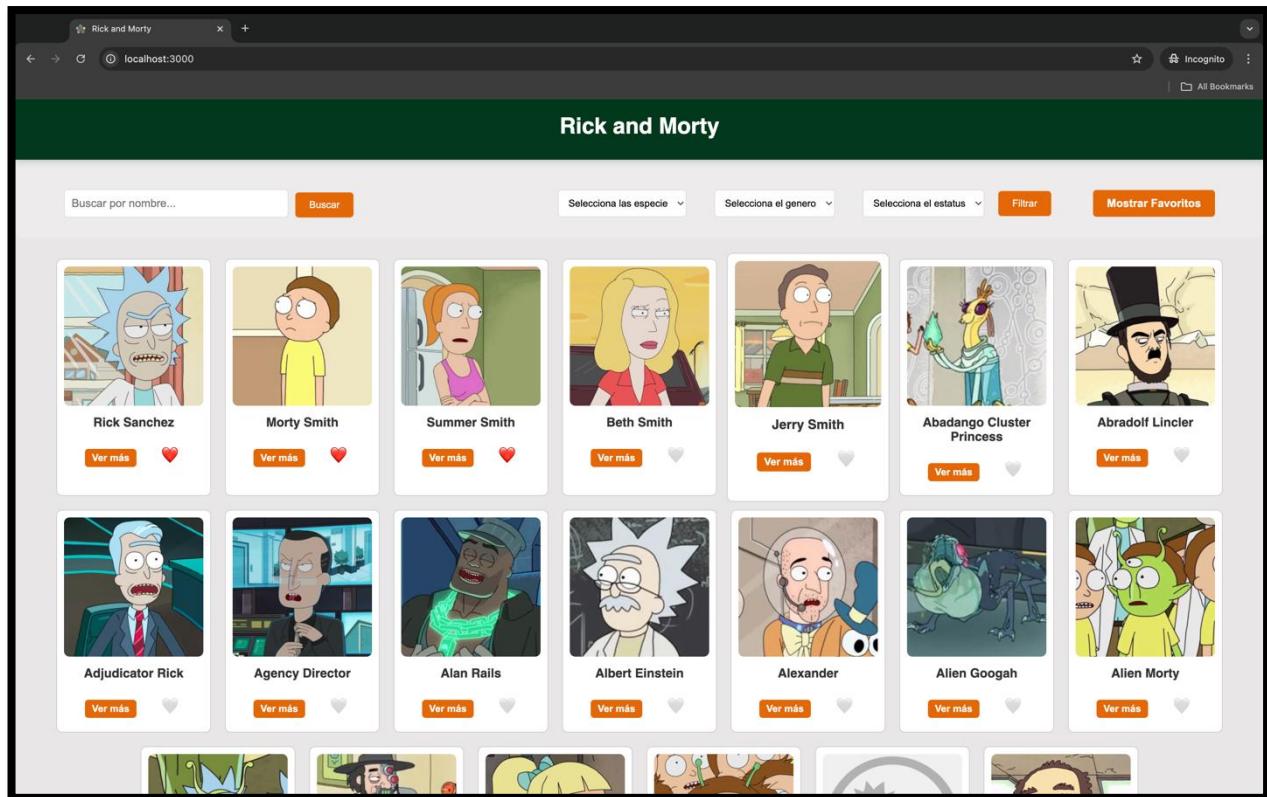


**19) Agregamos la funcionalidad de favoritos en el componente CharacterList;**

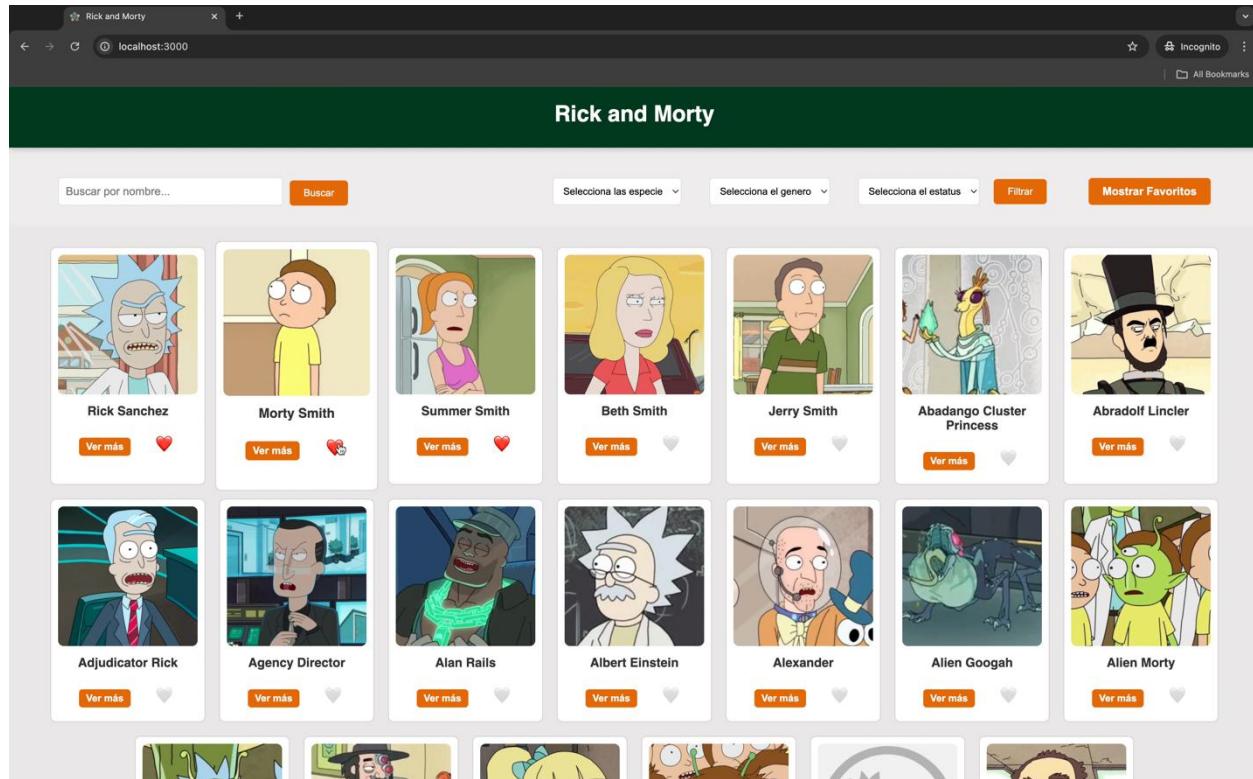
- **Estado para favoritos:**
  - **Añadimos** un estado `favorites`, que es un arreglo que almacenará los personajes que el usuario ha marcado como favoritos.
  - **Función:** Este estado permite realizar un seguimiento de qué personajes han sido seleccionados como favoritos.
- **Función toggleFavorite:**
  - **Descripción:** Esta función se encarga de activar o desactivar un personaje como favorito.
  - **Detalles:**
    - **Activar Favorito:** Si el personaje ya está en el estado `favorites` (usando `some` para verificar), se elimina de la lista utilizando `filter`.
    - **Desactivar Favorito:** Si el personaje no está en `favorites`, se agrega utilizando el operador de propagación (...).
  - **Función:** Permite que los usuarios añadan o eliminan personajes de su lista de favoritos al hacer clic en el botón del corazón.
- **Mostrar Favoritos u Otros Personajes:**
  - **Descripción:** Esta línea define qué personajes se mostrarán en la interfaz.
  - **Detalles:** Si `showFavorites` es verdadero, se mostrarán solo los personajes favoritos. De lo contrario, se mostrarán todos los personajes.
  - **Función:** Permite alternar entre la vista de todos los personajes y la vista de solo los favoritos.
- **Interfaz de Usuario:**
  - **Descripción:** Aquí es donde se muestra el botón del corazón.
  - **Detalles:**
    - **Corazón Lleno o Vacío:** El botón muestra un corazón lleno (❤) si el personaje está marcado como favorito y un corazón vacío (♡) si no lo está.
    - **Interacción:** Al hacer clic en el botón, se llama a la función `toggleFavorite`, lo que activa o desactiva el estado del favorito.
  - **Función:** Permite a los usuarios marcar personajes como favoritos directamente desde la lista.

- **Botón para Mostrar/Ocultar Favoritos:**

- **Descripción:** Este botón permite al usuario alternar entre mostrar todos los personajes y mostrar solo los favoritos.
- **Función:** Facilita la navegación en la lista, mejorando la experiencia del usuario.



Video demostrativo (Haz dole click).



Esta es una página dedicada a *Rick and Morty*, que incluye funciones básicas como filtrado, búsqueda y gestión de favoritos. Puedes añadir más funcionalidades y personalizar el estilo según tus preferencias