💪 HW8 | React Estado LifeCycle - Integration

🕓 DURACIÓN ESTIMADA

90 minutos

💻 RICK AND MORTY APP 💻

📝 INTRODUCCIÓN

Hasta el momento, en nuestra **Rick & Morty App** tenemos estos 3 componentes: **Card**, **Cards** y **SearchBar**.

Adicionalmente, crearemos otro componente denominado `Nav` que será nuestra barra superior de navegación, el cual envolverá a la `SearchBar`.

📋 INSTRUCCIONES

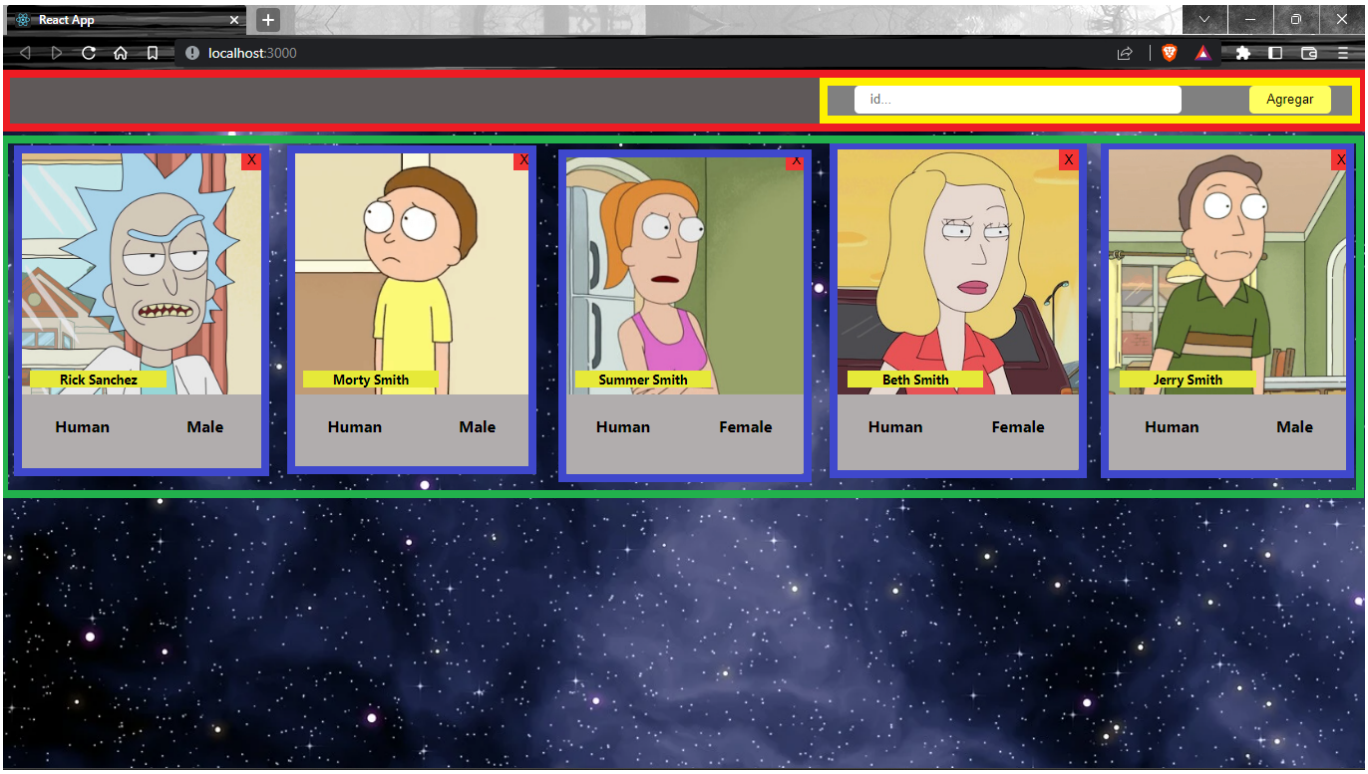
🤖 EJERCICIO 1 | Nav

1. Dirígete a tu archivo `App.js` y elimina tu SearchBar.
2. Crear el componente `Nav` dentro de la carpeta "***components***".
3. Renderiza la SearchBar dentro de este componente.

🤖 EJERCICIO 2 | Home

1. Elimina la **Card** "*suelta*" que se está renderizando.
2. Importa y renderiza el componente `Nav`.
3. ¡Aplica los estilos que más quieras!

Puedes guiarte a partir de la siguiente imagen cómo puede quedar tu aplicación:



> - **Recuadro rojo**: Nav
> - **Recuadro amarillo**: SearchBar
> - **Recuadro verde**: Cards
> - **Recuadro azul**: Card

🤖 EJERCICIO 3 | Estado

En este momento estamos dependiendo de un archivo `data.js` para recibir a los personajes. Lo que haremos ahora será crear un estado que nos permita almacenar personajes directamente.

Para esto, dirígete al componente `App.js` y:

1. Elimina el import y el archivo `data.js`. A partir de ahora ya no lo utilizaremos.
2. Importa el hook useState.
3. Crea un estado local llamado `characters` el cual se debe inicializar como un arreglo vacío.

🤖 EJERCICIO 4 | On Search

Ahora crearás una función llamada `onSearch` que te servirá para agregar nuevos personajes al estado que creaste en el ejercicio anterior.

1. Crea una función llamada **onSearch** en tu archivo `App.js`.
2. Cada vez que esta función sea ejecutada deberá agregar un nuevo personaje a tu estado local **characters**.

Como por el momento no vamos a recibir nuevos personajes, utilizaremos uno "*por default*". Es decir, cada vez que se ejecute la función anterior se debe agregar este personaje al estado local.

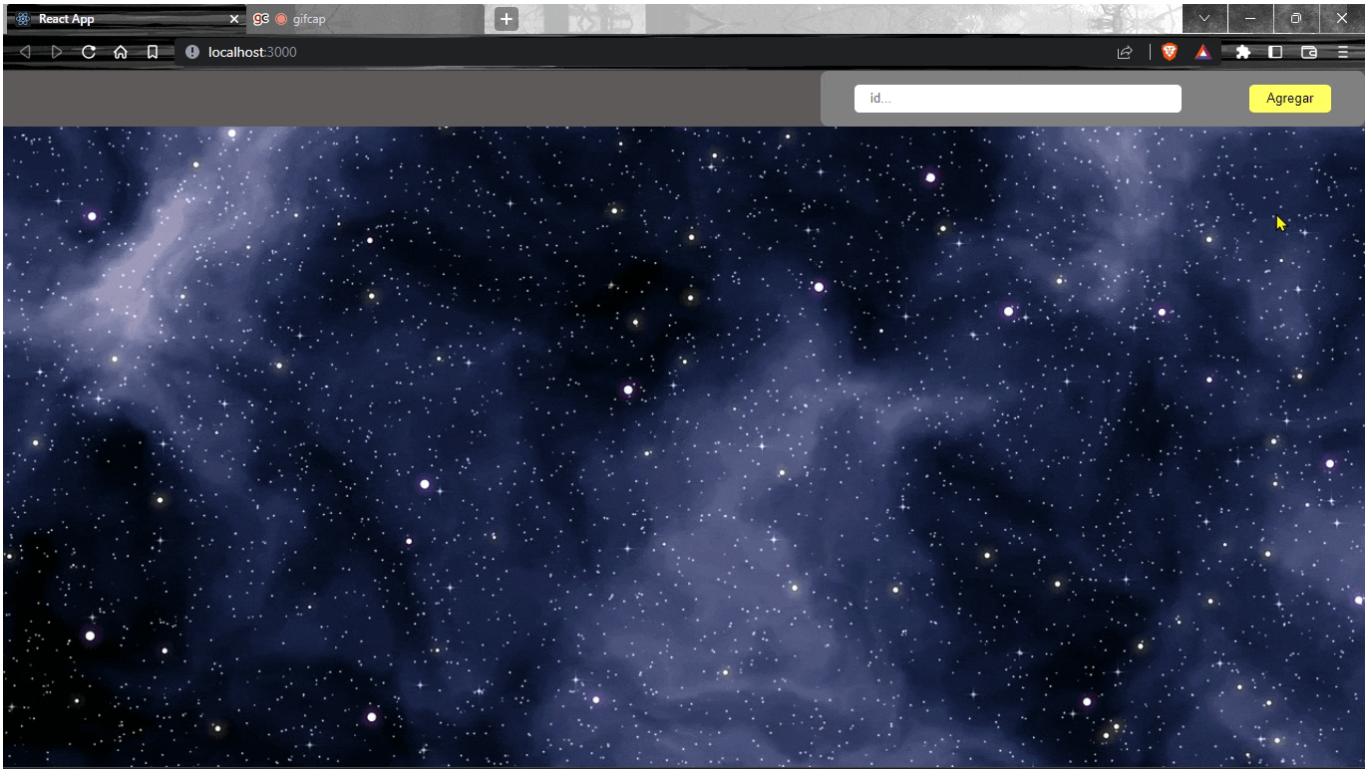
```
const example = {
   id: 1,
   name: 'Rick Sanchez',
   status: 'Alive',
   species: 'Human',
   gender: 'Male',
   origin: {
      name: 'Earth (C-137)',
      url: 'https://rickandmortyapi.com/api/location/1',
   },
   image: 'https://rickandmortyapi.com/api/character/avatar/1.jpeg',
};
```

🤖 EJERCICIO 5 | SearchBar & onSearch

Una vez que hayas creado la función `onSearch` deberás:

1. Pasársela como propiedad al componente `Nav`.
2. Pasársela como propiedad al componente `SearchBar`.

¡Listo! Si levantas tu proyecto y compruebas en tu navegador, cade vez que haces click sobre el botón "***Agregar***", se mostrará un nuevo personaje. Debería quedar algo así:



🔑 ALTO AQUÍ

Hasta el momento podemos agregar a un mismo personaje dentro de nuestra aplicación, pero... ¿Cómo podríamos agregar a distintos personajes?

🤓 Una buena idea sería utilizar nuestra SearchBar, ¿no te parece?

Podríamos escribir dentro de nuestra SearchBar el **ID** de un personaje, y que este se agregue automáticamente en nuestra aplicación.

✅ ¡Sigamos para descrubir cómo llevar esto a cabo!

🤖 EJERCICIO 6 | Parámetros

Ahora nos dirigiremos a la `SearchBar` para realizar algunas modificaciones. De esta forma podremos guardar el **ID** que escriba el usuario de nuestra aplicación.

1. Importa y crea un estado local llamado **id**. Debe inicializarse como un string vacío.
2. Crea una función **handleChange** de modo que, cada vez que el usuario escriba algo en el input, este se guarde en el estado local **id**.
3. No te olvides de pasarle esta función al input, y asignarle a este el estado local como su `value`.
4. Una vez que hayas cumplido con todos estos pasos, asegúrate de que cada vez que se ejecute la función `onSearch` esa reciba el estado **id** como argumento.

🤖 EJERCICIO 7 | API Connection

Ahora modificaremos la función `onSearch` para que busque nuevos personajes en la API de **Rick & Morty**. Para esto:

1. Instala la dependencia "**axios**". Una vez instala impórtala en el componente `App.js`.

2. Elimina la función `onSearch` que ya creaste y remplázala por esta nueva función:

```
function onSearch(id) {
   axios(`https://rickandmortyapi.com/api/character/${id}`).then(({ data }) => {
      if (data.name) {
         setCharacters((oldChars) => [...oldChars, data]);
      } else {
         window.alert('¡No hay personajes con este ID!');
      }
   });
}
```

> [**NOTA**]: como aún no hemos visto promesas, tienes este snippet para que copies la funcionalidad.

EJERCICIO 8 | On Close

En este momento, el componente `Cards` les está pasando al componente `Card` una función llamada **onClose**. Esta función no está realizando nada más que mostrar un aviso en el navegador. ¡Ahora le daremos la funcionalidad que estamos buscando! Para esto:

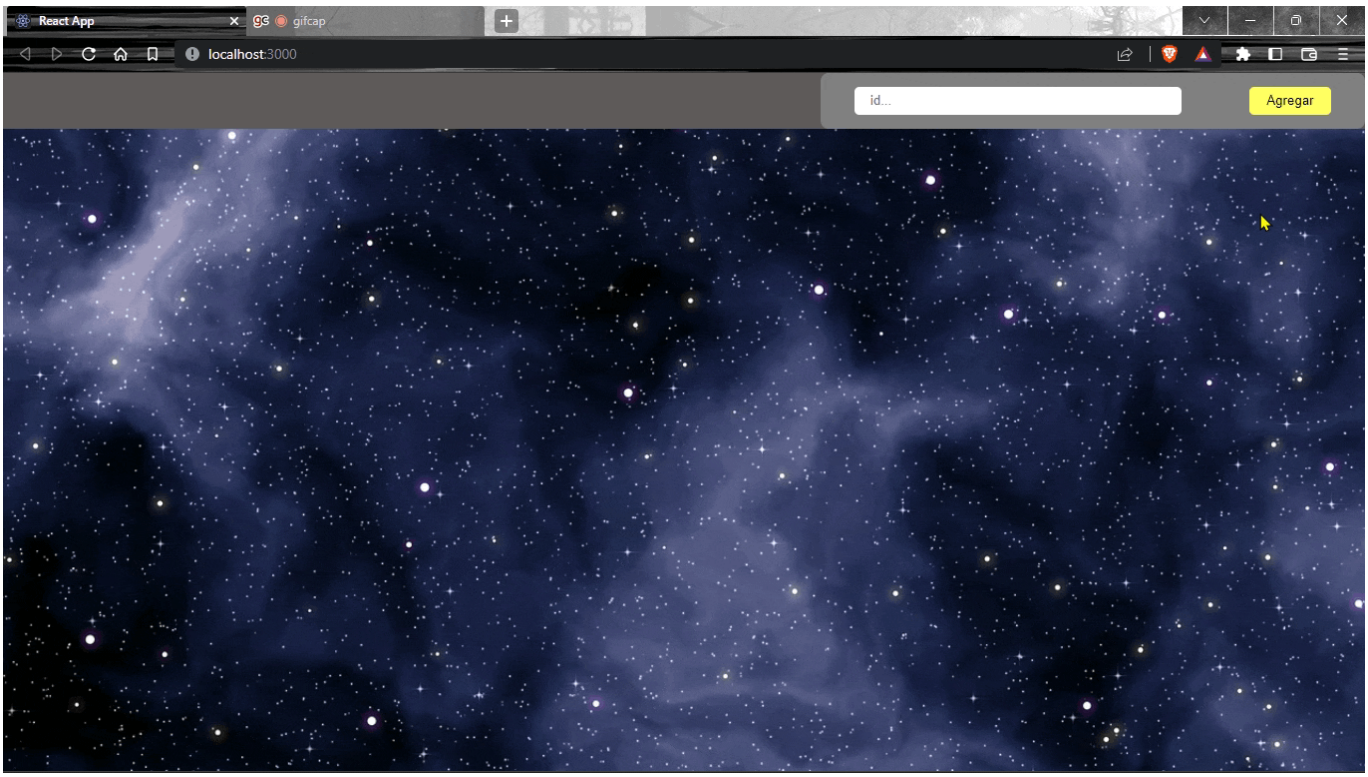
1. Dirígete a tu componente `App.js` y crea una función con el nombre **onClose**. Esta función recibirá por parámetro un **id**.
2. Dentro de la función deberás realizar un filtro de tu estado local en el que te quedes con todos aquellos personajes cuyo **id** sea distinto al que recibes por parámetro.

> [**NOTA**]: el id que recibes por parámetro es un string, pero el que debes comparar en tus personajes es un number. ¡Parséalo!

3. Setea este resultado en tu estado local **characters**.
4. Dirígete al componente `Cards` y envíale el **id** del personaje como propiedad al componente `Card`.
5. Finalmente dirígete al componente `Card` y pásale el **id** que recibes por props a la funcion **onClose** cuando se ejecuta.

Ahora solo queda que pases esta función al componente `Cards`, y que este se la pase al componente `Card`.

Este es el resultado esperado:



EJERCICIO EXTRA

1. Controla que no se puedan agregar personajes repetidos que ya se muestran en pantalla.

2. Crea un botón en tu componente `Nav` que te permita agregar un personaje random.

> [**NOTA**]: hay 826 personajes en total.