¡Excelente! Me alegra que te haya gustado.

Ya que dominas la modularización, el siguiente paso lógico y fundamental en React es aprender a manejar el **estado (state)** y los **eventos (events)**. Esto es lo que hace que tus aplicaciones cobren vida y sean interactivas.

Construiremos una aplicación clásica pero increíblemente útil para aprender: una **lista de tareas (To-Do List)**.

Tutorial: Interactividad con State y Eventos en React (useState)

En este tutorial, aprenderás a usar el hook useState para guardar y actualizar información dentro de un componente, y cómo responder a las interacciones del usuario, como escribir en un campo de texto y hacer clic en un botón.

Objetivo del Proyecto

Crearemos un componente TodoList que permitirá al usuario:

1. Ver una lista de tareas.
2. Escribir una nueva tarea en un campo de texto.
3. Añadir esa nueva tarea a la lista al presionar un botón.

Paso 1: Preparar el Entorno

Puedes continuar en el proyecto mi-app-modular que creamos antes.

1. Dentro de la carpeta src/components, crea una nueva carpeta llamada TodoList.
2. Dentro de TodoList, crea dos archivos: TodoList.js y TodoList.css.

Tu estructura ahora se verá así:

src/

└── components/

  ├── Header/

  ├── Welcome/

  └── TodoList/

    ├── TodoList.js

    └── TodoList.css

Paso 2: Crear el Componente TodoList y su Estado Inicial

Vamos a empezar por definir el componente y darle un "estado" inicial, que será nuestra lista de tareas.

1. Abre el archivo src/components/TodoList/TodoList.js y añade el siguiente código:

JavaScript

```
import React, { useState } from 'react';
import './TodoList.css';


const TodoList = () => {
  // 1. Hook useState para guardar la lista de tareas
  const [tasks, setTasks] = useState([
    { id: 1, text: 'Aprender React' },
    { id: 2, text: 'Construir una App' },
    { id: 3, text: 'Modularizar componentes' }
  ]);


  return (
    <div className="todo-list-container">
      <h2>Mi Lista de Tareas</h2>


      {/* 2. Aquí mostraremos la lista de tareas */}
      <ul>
        {tasks.map(task => (
          <li key={task.id}>{task.text}</li>
        ))}
      </ul>


      {/* 3. Aquí irá el formulario para añadir nuevas tareas */}
    </div>
```

```
  );
};

export default TodoList;
```

¿Qué está pasando aquí?

- **import React, { useState } from 'react';**: Importamos React y, lo más importante, el hook **useState**. Un "hook" es una función especial que te permite "engancharte" a las características de React, como el estado.
- **const [tasks, setTasks] = useState([...]);**: Aquí usamos el hook useState.
  - useState() devuelve un par de valores: la **variable de estado** actual (tasks) y una **función para actualizarla** (setTasks).
  - Le damos un valor inicial a nuestro estado: un arreglo de objetos que representan nuestras tareas.
- **tasks.map(task => ...)**: Usamos el método .map() de JavaScript para recorrer nuestro arreglo tasks. Por cada tarea en el arreglo, creamos un elemento de lista <li>.
- **key={task.id}**: **¡Muy importante!** Cuando creas una lista de elementos en React, debes darle a cada uno una prop key única. Esto ayuda a React a identificar qué elementos han cambiado, se han añadido o eliminado de manera eficiente.

Paso 3: Añadir Nuevas Tareas (Manejo de Eventos)

Ahora, vamos a hacer que la lista sea interactiva. Necesitamos un campo de texto para escribir la nueva tarea y un botón para añadirla.

1. Modifica TodoList.js para añadir un segundo estado que controle el valor del campo de texto y el formulario para añadir tareas.

JavaScript

```
import React, { useState } from 'react';

import './TodoList.css';


const TodoList = () => {

  const [tasks, setTasks] = useState([

    { id: 1, text: 'Aprender React' },
```

```
  { id: 2, text: 'Construir una App' },
  { id: 3, text: 'Modularizar componentes' }
]);

// Nuevo estado para el campo de texto
const [inputValue, setInputValue] = useState('');

// Función para manejar el envío del formulario
const handleAddTask = (e) => {
  e.preventDefault(); // Evita que la página se recargue
  if (inputValue.trim() === '') return; // No añadir tareas vacías

  const newTask = {
    id: Date.now(), // ID único basado en la fecha actual
    text: inputValue
  };

  setTasks([...tasks, newTask]); // Añadimos la nueva tarea a la lista
  setInputValue(''); // Limpiamos el campo de texto
};

return (
  <div className="todo-list-container">
    <h2>Mi Lista de Tareas</h2>

    {/* Formulario para añadir nuevas tareas */}
    <form onSubmit={handleAddTask} className="add-task-form">
      <input
        type="text"
```

```
        value={inputValue}
        onChange={(e) => setInputValue(e.target.value)}
        placeholder="Añade una nueva tarea..."
      />
      <button type="submit">Añadir</button>
    </form>

    <ul>
      {tasks.map(task => (
        <li key={task.id}>{task.text}</li>
      ))}
    </ul>
  </div>
 );
};

export default TodoList;
```

Desglose de las novedades:

- **const [inputValue, setInputValue] = useState('');**: Creamos un nuevo estado para almacenar lo que el usuario escribe en el <input>. Comienza como una cadena vacía.
- **<input ... />**:
  - **value={inputValue}**: Conectamos el valor del input a nuestra variable de estado inputValue. Esto se conoce como un **componente controlado**.
  - **onChange={(e) => setInputValue(e.target.value)}**: Este es el **manejador de eventos**. Cada vez que el usuario escribe algo (onChange), actualizamos el estado inputValue con el valor actual del campo de texto (e.target.value).
- **<form onSubmit={handleAddTask}>**: Cuando el formulario se envía (al presionar "Enter" o hacer clic en el botón), se ejecuta nuestra función handleAddTask.
- **handleAddTask function**:

- e.preventDefault(): Es crucial para evitar que el navegador recargue la página, que es el comportamiento por defecto de un formulario.
- setTasks([...tasks, newTask]): Esta es la forma correcta de actualizar un arreglo en el estado de React. Creamos un **nuevo arreglo** que contiene todos los elementos del arreglo antiguo (...tasks) más el nuevo elemento (newTask). **Nunca modifiques el estado directamente** (ej: tasks.push(newTask)).

Paso 4: Estilos y Montaje Final

1. Añade algunos estilos básicos en src/components/TodoList/TodoList.css:

CSS

```
.todo-list-container {
  margin: 20px auto;
  padding: 20px;
  max-width: 500px;
  background-color: #f9f9f9;
  border-radius: 8px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.add-task-form {
  display: flex;
  margin-bottom: 20px;
}

.add-task-form input {
  flex-grow: 1;
  padding: 10px;
  border: 1px solid #ddd;
  border-radius: 4px;
}
```

```
.add-task-form button {
  padding: 10px 15px;
  border: none;
  background-color: #007bff;
  color: white;
  border-radius: 4px;
  margin-left: 10px;
  cursor: pointer;
}

.todo-list-container ul {
  list-style-type: none;
  padding: 0;
}

.todo-list-container li {
  padding: 10px;
  background-color: #fff;
  border-bottom: 1px solid #eee;
}
```

2. Finalmente, importa y usa tu nuevo componente TodoList en src/App.js. Puedes reemplazar el componente Welcome.

JavaScript

```
import React from 'react';
import './App.css';
import Header from './components/Header/Header';
import TodoList from './components/TodoList/TodoList'; // Importamos el nuevo componente
```

```
function App() {
  return (
    <div className="App">
      <Header />
      <main>
        <TodoList />
      </main>
    </div>
  );
}

export default App;
```

Paso 5: ¡A Probarlo!

Si tu servidor de desarrollo sigue corriendo (npm start), la página se habrá actualizado automáticamente. Si no, ejecútalo de nuevo.

Ahora deberías ver tu lista de tareas. ¡Intenta añadir una nueva tarea y observa cómo la lista se actualiza al instante!

Resumen de Conceptos Clave Aprendidos

- **Hook useState**: Para dar "memoria" a tus componentes y guardar datos que cambian con el tiempo.
- **Eventos**: Cómo responder a las acciones del usuario, como onChange para los inputs y onSubmit para los formularios.
- **Componentes Controlados**: La práctica de vincular el valor de un campo de formulario al estado de React.
- **Actualización Inmutable del Estado**: La importancia de crear nuevas copias del estado (arreglos u objetos) en lugar de modificarlos directamente.

¡Felicidades! Ahora puedes crear aplicaciones de React que no solo se ven bien, sino que también son completamente interactivas.