Politécnico PIO
Internacional de Occidente

Crear un proyecto React con Vite utilizando useState y useEffect

Repasemos un poco

useState

El hook useState se utiliza para manejar el estado en un componente funcional. Permite agregar datos dinámicos a los componentes y actualizarlos con el tiempo.

Sintaxis básica:

```
const [state, setState] = useState(initialValue);
```

- State: representa el valor actual del estado
- setState: es una función para actualizar el estado
- initialvalue: valor inicial del estado

useEffect

El hook useEffect se utiliza para manejar **efectos secundarios** en los componentes funcionales, como:

- Fetch de datos desde una API.
- Suscripciones a eventos.
- Actualización del DOM.
- Limpieza de recursos (por ejemplo, detener un temporizador).

Sintaxis básica

```
useEffect(() => {
  // Código del efecto
  return () => {
    // Limpieza del efecto (opcional)
  };
}, [dependencies]);
```

- Efecto principal: Se ejecuta después de que el componente se renderiza.
- Función de limpieza: Limpia el efecto anterior (por ejemplo, elimina un evento).
- dependencies (dependencias): Determinan cuándo debe ejecutarse el efecto. Si se deja vacío ([]), el efecto solo se ejecutará una vez, cuando el componente se monte.

Dependencias en useEffect

Las dependencias determinan cuándo se ejecuta el efecto.

1. **Sin dependencias (useEffect(() => {...}))**
    - El efecto se ejecuta después de cada renderizado.
2. **Con dependencias vacías (useEffect(() => {...}, []))**
    - El efecto se ejecuta solo una vez, cuando el componente se monta.
3. **Con dependencias específicas (useEffect(() => {...}, [dependency]))**
    - El efecto se ejecuta solo cuando las dependencias cambian.

Politécnico PIO
Internacional de Occidente

Ahora… Crearemos una aplicación llamada "Contador de Clics con Historial". La aplicación contará los clics realizados y mostrará un historial dinámico de estos, gestionando los estados y efectos secundarios con los hooks useState y useEffect.

Para crear un proyecto o una web app con Vite deberemos seguir los siguientes pasos:
Crea la configuración necesaria para un proyecto con React

a. Ejecutaremos node –v en la terminal para revisar la versión de Node.js.

b. Crearemos un proyecto con Vite, para esto seguiremos los siguientes pasos:

- Desde la terminal ejecuta:
  npm create vite@latest contadorHistorial -- --template react

- En la terminal deberias de observar mensajes similares a estos:

```
C:\MERN\lp1\tallerReact>npm create vite@latest contador-historial -- --template react

> npx
> create-vite contador-historial --template react


Scaffolding project in C:\MERN\lp1\tallerReact\contador-historial...

Done. Now run:

  cd contador-historial
  npm install
  npm run dev
```

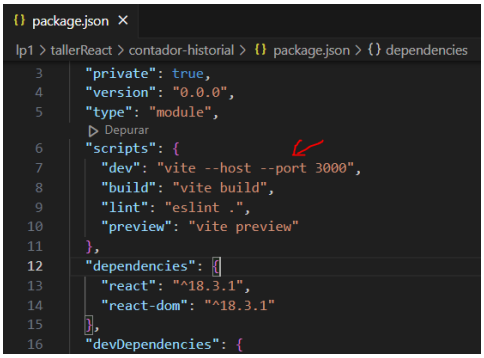
- En es línea estamos indicando que en nombre del proyecto es contador-historial y la plantilla es react

- Movernos al directorio del proyecto con el comando cd <nombre del proyecto>.

- Instalar las dependencias con el comando npm install.

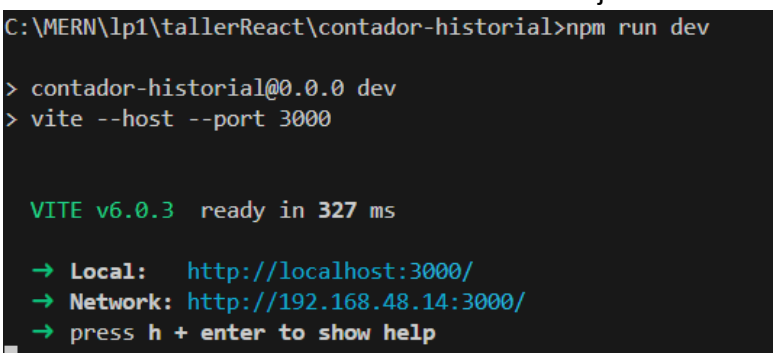
- Modificar nuestro script de desarrollo, para esto:

  Abrimos el archivo package.json
  En el scripts reemplazaremos "dev": "vite --host –host 3000",

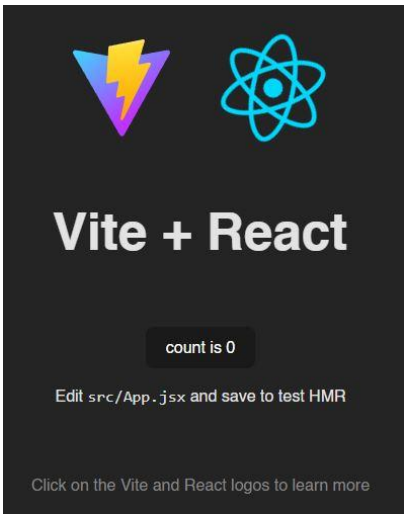


1. Finalmente, ejecutamos el comando **npm run dev** desde la terminal, para iniciar el servidor de desarrollo.

   En la terminal deberíamos de tener un mensaje similar a este:



Al dirigirnos a nuestro navegador web favorito, como por ejemplo Google Chrome, en la dirección http://localhost:3000/, deberíamos ser capaces de observar la página de cargue inicial del proyecto:



Politécnico PIO
Internacional de Occidente

Construcción del ejercicio

1. Limpiar el proyecto inicial

Elimina los contenidos predeterminados de src/App.jsx y deja una estructura básica:

App.jsx

lp1 > tallerReact > contador-historial > src > App.jsx > ...

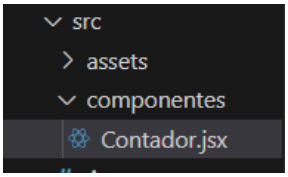
```jsx
import React from 'react'

function App() {

  return (
    <>
      <h1>Contador de clics con historial</h1>
    </>
  )
}

export default App
```

2. Crear el componente Contador
   a. Crea una nueva carpeta llamada **componentes** en la carpeta src/.
   b. Crea un archivo nuevo llamado **Contador.jsx** en la carpeta componentes/.



Añade el siguiente código en el archivo Contador.jsx:

1 > tallerReact > contador-historial > src > componentes > Contador.jsx > Contador

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

export default function Contador() {
    const [contador, setContador] = useState(0); // Estado del contador
    const [historial, setHistorial] = useState([]); // Estado para el historial

    // Efecto para actualizar el historial cada vez que cambia el contador
    useEffect(() => {
      if (contador !== 0) {
        setHistorial((prevHistorial) => [...prevHistorial, contador]);
      }
    }, [contador]);

    // Incrementar el contador
    const incrementar = () => {
      setContador(contador + 1);
    };

    // Reiniciar el contador y el historial
    const reiniciar = () => {
      setContador(0);
      setHistorial([]);
    };

```

```
    return (
      <div style={{ textAlign: 'center', marginTop: '20px' }}>
        <h2>Contador: {contador}</h2>
        <button onClick={incrementar} style={buttonStyle}>
          Incrementar
        </button>
        <button onClick={reiniciar} style={buttonStyle}>
          Reiniciar
        </button>
        <h3>Historial:</h3>
        <ul>
          {historial.map((valor, index) => (
            <li key={index}>Clic #{index + 1}: {valor}</li>
          ))}
        </ul>
      </div>
    );
  }

const buttonStyle = {
  margin: '10px',
  padding: '10px 20px',
  fontSize: '16px',
  cursor: 'pointer',
  borderRadius: '5px',
  backgroundColor: '#007bff',
  color: '#fff',
  border: 'none',
};
```

3. Incorporar el componente al proyecto
   En el archivo src/App.jsx, importa y utiliza el componente Contador:

```
p1 > tallerReact > contador-historial > src > App.jsx > App
import React from 'react'
import Contador from './componentes/Contador'

function App() {

  return (
    <>
      <div className='App' style={{ fontFamily: 'Arial, sans-serif', textAlign: 'center', padding: '20px' }}>
        <h1>React con Vite: Contador de Clics con Historial</h1>
        <Contador/>

      </div>
    </>
  )
}

export default App
```

4. Agregar estilos opcionales
   En el archivo llamado src/App.css y añade:

```css
body {
  font-family: 'Arial', sans-serif;
  background-color: #f9f9f9;
  color: #333;
  margin: 0;
  padding: 0;
}

h1 {
  color: #007bff;
}

ul {
  list-style: none;
  padding: 0;
}

li {
  margin: 5px 0;
  padding: 5px;
  background-color: #e9ecef;
  border-radius: 4px;
  font-size: 14px;
}
```

5. Importa los estilos en src/App.jsx:

```
lp1 > tallerReact > contador-historial > src > App.jsx > ...
1   import React from 'react'
2   import './App.css'
3   import Contador from './componentes/Contador'
4
5   function App() {
```

Funcionamiento

1. **useState:**
   - contador: Almacena el número actual de clics.
   - historial: Almacena un array con los valores del contador después de cada clic.
2. **useEffect:**
   - Se ejecuta cada vez que cambia el valor de contador.
   - Agrega el valor actual del contador al historial, siempre y cuando el contador no sea 0.
3. **Funciones:**
   - incrementar: Incrementa el valor del estado contador en 1.
   - reiniciar: Reinicia el contador y vacía el historial.

Este ejercicio te ayuda a practicar:

- La gestión de múltiples estados con useState.
- La dependencia de estados en useEffect.
- La renderización dinámica de listas en React.

6. Ahora, actualizaremos el componente Contador para incluir un botón en cada elemento del historial que permita eliminarlo. También actualizaremos la lógica para manejar la eliminación.

a. Inserta la función eliminar en el archivo **Contador.jsx**

```
19    // Reiniciar el contador y el historial
20    const reiniciar = () => {
21      setContador(0);
22      setHistorial([]);
23    };
24
25    // Eliminar un elemento específico del historial
26    const eliminarElemento = (indice) => {
27      setHistorial((prevHistorial) =>
28        prevHistorial.filter((_, i) => i !== indice)
29      );
30    };
31
32    return (
33      <div style={{ textAlign: 'center', marginTop: '20px' }}>
34        <h2>Contador: {contador}</h2>
35        <button onClick={incrementar} style={buttonStyle}>
```

- **indice**: Es un parámetro que representa el índice (posición) del elemento que queremos eliminar de la lista.
- **setHistorial**: Es la función proporcionada por useState para actualizar el estado historial.
- **prevHistorial**: Es el valor actual del estado historial. Se pasa automáticamente como argumento a la función de actualización.
- **prevHistorial.filter()**: Crea un nuevo array a partir de prevHistorial, incluyendo solo los elementos que cumplan una condición específica.
- **(_, i)**: El método filter pasa dos parámetros a su función de callback:
  - ✓ _: Representa el valor actual del elemento en el array. En este caso no lo necesitamos, así que lo dejamos como _.
  - ✓ **i**: Representa el índice del elemento actual en el array.
- **i !== indice**: Es la condición que evalúa si el índice del elemento actual es diferente del índice que queremos eliminar. Si la condición es true, el elemento permanece en el nuevo array; si es false, se elimina.

b. Modifica la lista con el siguiente código en el archivo **Contador.jsx**

```
    return (
      <div style={{ textAlign: 'center', marginTop: '20px' }}>
        <h2>Contador: {contador}</h2>
        <button onClick={incrementar} style={buttonStyle}>
          Incrementar
        </button>
        <button onClick={reiniciar} style={buttonStyle}>
          Reiniciar
        </button>
        <h3>Historial:</h3>
        <ul>
          {historial.map((valor, index) => (
            <li key={index} style={listItemStyle}>
              Clic #{index + 1}: {valor}{' '}
              <button
                onClick={() => eliminarElemento(index)}
                style={deleteButtonStyle}
              >
                Eliminar
              </button>
            </li>
          ))}
        </ul>
      </div>
    );
}
```

c. Añade los estilos de listItemStyle y deleteButtonStyle en el archivo **Contador.jsx.**

```
const listItemStyle = {
  margin: '5px 0',
  padding: '5px',
  backgroundColor: '#e9ecef',
  borderRadius: '4px',
  display: 'flex',
  justifyContent: 'space-between',
  alignItems: 'center',
};

const deleteButtonStyle = {
  marginLeft: '10px',
  padding: '5px 10px',
  fontSize: '14px',
  cursor: 'pointer',
  borderRadius: '3px',
  backgroundColor: '#dc3545',
  color: '#fff',
  border: 'none',
};
```

Cambios realizados

1. **Nueva función eliminarElemento:**
   - Toma el índice del elemento que se desea eliminar.
   - Utiliza el método filter para crear un nuevo array excluyendo el elemento seleccionado.
2. **Botón "Eliminar":**
   - En cada elemento del historial, se agrega un botón asociado con su índice.
   - Este botón llama a la función eliminarElemento al hacer clic.
3. Estilos adicionales:
   - listItemStyle: Mejora la presentación de cada elemento del historial.
   - deleteButtonStyle: Personaliza el botón "Eliminar".

Esta actualización demuestra cómo manejar eventos y actualizar estados en React con los hooks useState y useEffect. Además, el uso de filter permite manipular arrays de manera eficiente, mostrando cómo gestionar listas dinámicas en React.