Actividad de Aprendizaje N° 07A

Hooks en React

1. Hooks en React

Hooks fue agregado a React en la versión 16.8

Hooks permite a los componentes de tipo función tener acceso a la gestión de estado y otras características de React. Debido a esto, los componentes de clase ya casi no son necesarios.

¿Qué es un Hook?

> Es una *función de javascript* que permite crear y acceder al ***estado*** y al ***ciclo de vida del componente*** en React

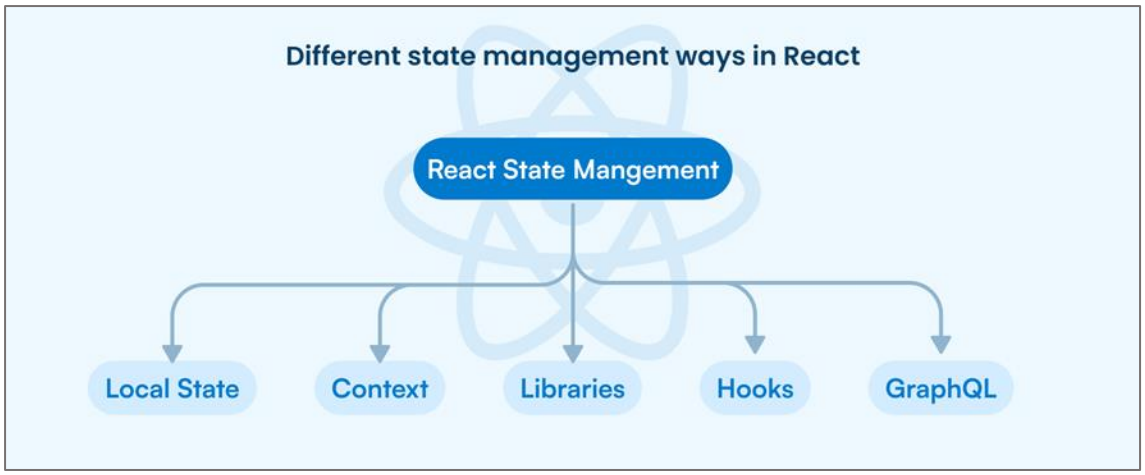
Tipos de Hooks

UseState
UseEffect
UseContext
UseReducer
UseRef
Lazy Loading
Suspense

2. State Management in React

La gestión de estados *es el proceso de seguimiento y actualización del estado de una aplicación*. El estado es *la condición actual de una aplicación*(la ubicación actual del usuario, los artículos en su carrito de compras o el estado de un juego).

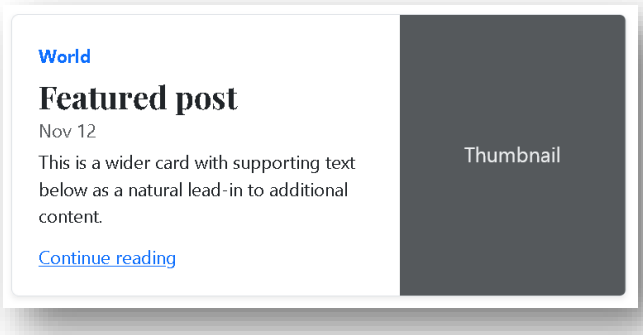
Depende del tamaño del proyecto. https://www.etatvasoft.com/blog/react-state-management/



Local State

> El ***estado de un componente*** es un Objeto que contiene *los datos (variables)* que necesitamos para ***representarla*** en *la interfaz u otro contexto*.

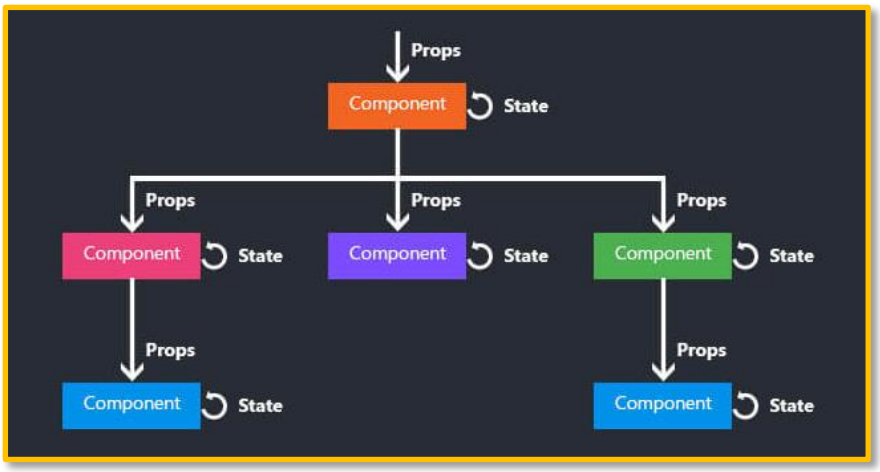
Ejemplo: *Class Component React*




```
const state = {
    blogId: "001",
    blogTitle: "Título del Blog",
    blogContent: "Contenido del Blog",
    blogActive: true,
    userId: "user101",
    createDate: "10/05/2024",
    updateDate: "30/05/2024",
    nameImage: "../images/img-blog001.png"
}
```

```
constructor(){
    super();
    this.state = { //Ejemplo de un estado
        blogId: "001",
        blogTitle: "Titulo de artículo",
        blogContent: "Contenido",
        blogActive: true,
        userID: "user101",
        createDate: "10/06/2022",
        updateDate: "24/06/2022",
        nameImage: "../images/img-blog001.jpg"
    }
```

¿Cuál es la diferencia entre state y props?



3. HOOK useState

El **Hook useState** *es una propiedad del componente* y su setter renderizable. El setter permite un renderizado automático en el DOM, sí el valor del useState cambió.

Sintaxis:

```
const [property, setProperty] = useState(initialValue)
```

useState acepta dos argumentos:

- El primero es la propiedad y su setter.
- El segundo argumento es el valor inicial de la propiedad.

¿Cómo usar un Hook useState?

1. **Importar useState**

```
import { useState } from "react";
```

2. **Inicializar useState:** ***ush***

```
const [color, setColor] = useState("");
```

3. **Leer el State**

```
return <h1>My favorite color is {color}!</h1>
```

4. **Actualizar el State**

```
<button
    type="button"
    onClick={() => setColor("blue")}
>Blue</button>
```

¿Cómo usar varios Hook useState?

1. **Importar useState**

```
import { useState } from "react";
```

2. **Inicializar useState**

Iniciar todas las propiedades

```
const [brand, setBrand] = useState("Ford");
const [model, setModel] = useState("Mustang");
const [year, setYear] = useState("1964");
const [color, setColor] = useState("red");
```

3. **Leer el State**

Se puede mostrar muchos elementos independientes

```
return (
    <>
      <h1>My {brand}</h1>
      <p>It is a {color} {model} from {year}.</p>
    </>
)
```

4. **Actualizar el State**

```
<button
    type="button"
    onClick={() => setColor("blue")}
>Blue</button>
```

¿Cómo usar un Hook useState con un objeto?

1. **Importar useState**

```
import { useState } from "react";
```

2. **Inicializar useState**

Asignar un objeto

```
const [car, setCar] = useState({
    brand: "Ford",
    model: "Mustang",
    year: "1964",
    color: "red"
});
```

3. **Leer el State**

Se puede mostrar desde un objeto

```
return (
    <>
      <h1>My {car.brand}</h1>
      <p>It is a {car.color} {car.model} from {car.year}.</p>
    </>
)
```

4. **Actualizar el State**

Cuando se *actualiza el estado*, se **sobrescribe** *todo el objeto*.

Ejemplo 01

```
const EjemUseState = () => {
    const [nombre, setNombre] = useState("");
    const [direccion, setDireccion] = useState("");
    const handleClick = () => {
        setNombre("Jaime")
        setDireccion("Junin 120")
    }
  return (
    <div>
        <p>Nombre: {nombre}</p>
        <p>Dirección: {direccion}</p>
        <button onClick={handleClick}>Cambiar Datos</button>
    </div>
  )
}
```

4. HOOK useEffect

El hook useEffect es una función que permite ejecutar código en los eventos:
Mount del componente
Update de una o varias propiedades
Post de un render.

```
useEffect (<function>, <dependency>)
```

useEffect acepta dos argumentos:

- El primero es una función a ejecutar.
- El segundo argumento es una dependencia pero es opcional .

¿Cómo usar un Hook useEffect?

1. **Importar useEffect**

```
import { useEffect} from 'react'
```

2. **Insertar useEffect:** ***ueh***

```
useEffect(() => {


}, []);
```

3. **Codificar useEffect según necesidad de ejecución.**

Eventos en el que useEffect ejecuta código.

a. Mount Component: Segundo argumentoarray vacío:

```
useEffect(() => {
    //Se ejecuta solo en el primer render
}, []);
```

b. Post Render: sin segundo argumento

```
useEffect(() => {
    //Se ejecutar en cada render
});
```

c. Update usState Segundo argumento listar useStates.

```
useEffect(() => {
    //Runs on the first render Se ejecuta en el prime
r render
    //Y algunas veces dependiendo del cambio de valor
}, [prop, state]);
```

Donde utilizar useEffect:

- Obtención de datos FETCHING
- Actualización directa del DOM
- Uso de Timers o Temporizadores.

Ejemplo 1: Usar useEffect en el mount y el render de un useState

Podemos cambiar a array vacio en useEffect.

```
const Ejem1UseEffect = () => {
    const [titulo, setTitulo] = useState("Sin titulo");
    let nombre="Jaime"
    useEffect(() => {
        document.title=titulo
    }, [titulo]);
    const handleClick = () => {
        setTitulo(`Hola ${nombre}`)
    }
  return (
    <div>
        <p>{titulo}</p>
        <button onClick={handleClick}>Cambiar Titulo</button>
    </div>
  )
}
```

Nota. Eso significa que cuando cambia el conteo con setCount, ocurre un renderizado, que desencadena otro efecto.

Ejemplo 2: Usar useEffect depende del render

La actualización depende del cambio de estado.

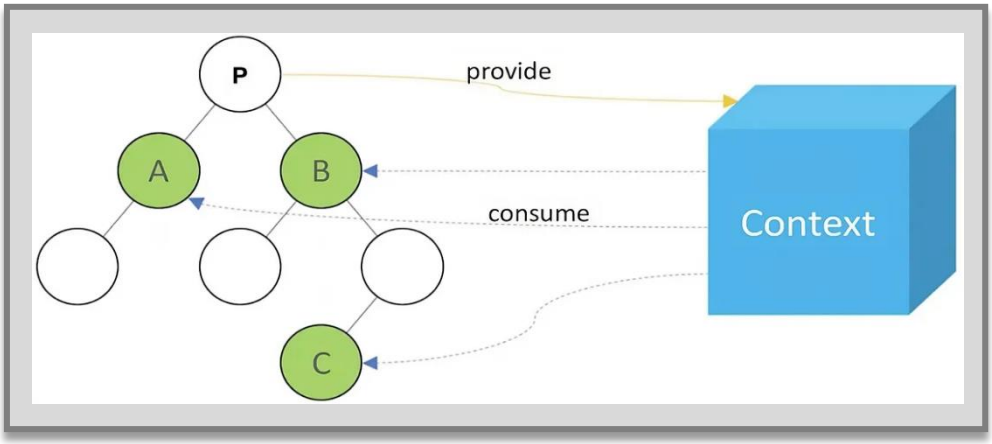
```
const Ejem2UseEffect = () => {
    const [contador, setContador] = useState(null);
    useEffect(() => {
        setTimeout(() => { //sto
            let c=contador
            setContador(c+1)
         }, 2000)
    });
    const handleClick = () => {
        setContador(0)
    }
  return (
    <div>
        <p>{contador}</p>
        <button onClick={handleClick}>Poner a 0 el Contador</button>
    </div>
  )
```

5. HOOK useContext

useContext es una unidad lógica que permite compartir información entre diversos componentes pueden acceder a datos y funciones en un Context compartido sin tener que pasar props.

Nota. React Context es una forma de administrar el estado globalmente.

Las propiedades de componentes que son compartidos deberían ser cambiando a un ámbito superior. *Ejemplo: Datos de un usuario*.



Sintaxis:

```
//Provider
export const myContext = React.createContext('Default value')

<myContext.Provider value={sharedData}>
    {/* children components */}
</myContext.Provider>
```

```
//Consomer
const sharedData = useContext(MyContext)
```

Crear el Context en el proveedor

1. **Crear contexto**
   a. Importar react
   b. Inicializar un contexto con createContext exportable

```
export const MyContext = React.createContext()
```

2. **Configurar el Proveedor de contexto**
   Con *<MyContext.Provider> </MyContext.Provider>* Envolver a los componentes secundarios con y *proporcionar el valor del estado*.

```
<MyContext.Provider value={/* algun valor */} >
    <Hijo />
</MyContext.Provider>
```

Ahora, todos los componentes dentro de MyContext tendrán acceso al contexto como valor global.

Usar el Context en el Consumer

Para usar el Contexto en un componente secundario, necesitamos acceder a él usando el el hook useContext.

1. **Importar useContext:**

```
import { useContext } from 'react'
import { MyContext } from './Padre'
```

2. **Obtener el Contexto guardado disponible para todos los componentes:**

```
const dato = useContext(MyContext)
```

3. **Mostrar datos obtenidos**

```
<p>{dato.nombre}</p>
<p>{dato.apellido}</p>
```

Ejemplo 1: Definir un objeto simple en useContext y Consumirlo

La actualización depende del cambio de estado.

Creamos el Componente Padre

```
export const MyContext = React.createContext()
const Padre = () => {
    let docente = {
        nombre: "Jaime",
        apellido: "Suasnabar"
    }
    return (
        <MyContext.Provider value={docente} >
            <Hijo />
        </MyContext.Provider>
    )
}
export default Padre
```

Creamos el componente hijo

```
import { useContext } from 'react'
import { MyContext } from './Padre'
const Hijo = () => {
    const dato = useContext(MyContext)
    return (
        <div>
            <p>{dato.nombre}</p>
            <p>{dato.apellido}</p>
        </div>
    )
}
export default Hijo
```

Ejemplo 2:

Definir un color simple en useContext y Consumirlo en varias instancias con diferentes colores.

```
export const MyContext = React.createContext("pink")
const Padre2 = () => {
    return (
        <>
            <h3>Componentes Colores</h3>
            <Hijo2 />
            <MyContext.Provider value="red">
                <Hijo2 />
            </MyContext.Provider>
            <MyContext.Provider value="green">
                <Hijo2 />
            </MyContext.Provider>
        </>
    )
}
export default Padre2
```

```
import React, { useContext } from 'react'
import { MyContext } from './Padre2';
const Hijo2 = () => {
    const color = useContext(MyContext);
    return (
             <h4 style={{color}}>Hola {color}</h4>
    )
}
export default Hijo2
```

6. Custom HOOK

Un Custom Hook es una función creada por el desarrollador que encapsula parte de la lógica y estado de un componente para que pueda ser reutilizable en otros.

Buenas prácticas Custom Hooks

- **Nombre de Hooks:**
  Comience el nombre con **use** seguido de un nombre que describe su propósito. Ejemplo **useEventListener**, **useFetch**.
- **Encapsulación:** Encapsular sólo una pieza de lógica por hook. Esto mejora la reutilizabilidad y mantiene su hook personalizado enfocado.

- **Minimizar el Estado:** Si su hook necesita manejar el estado con useState, tratar de mantener el estado mínimo.
- **Dependientes Array:** Preste mucha atención el array de dependencias de useEffect. Debe incluir todas las variables utilizadas dentro del efecto.
- **Pruebas:** Escriba pruebas de unidad para sus hooks personalizados. Esto asegura que funcionan correctamente y permanecen mantenibles.
- **Documentación:** Comente tus hooks personalizados, explicando lo que hacen y cómo usarlos.

Ejemplo 1: Desarrollar un Custom Hook para consumir datos de una API

```
import { useState, useEffect } from "react";
const useFetch = (url) => {
  const [data, setData] = useState(null);
  useEffect(() => {
    fetch(url)
      .then((res) => res.json())
      .then((data) => setData(data));
  }, [url]);

  return [data];
};
export default useFetch;
```

```
import useFetch from "./useFetch";
const Padre3 = () => {
  const [data] = useFetch("https://jsonplaceholder.typicode.com/todos");
  return (
    <>
      {data &&
        data.map((item) => {
          return <p key={item.id}>Title: {item.title}</p>;
        })}
    </>
  );
};
export default Padre3
```

7. HOOK Reducer

El hook useReducer es un administrador de estado en React es una alternativa al useState. Se utiliza cuando el estado de un componente es complejo y requiere más de una variable de estado.

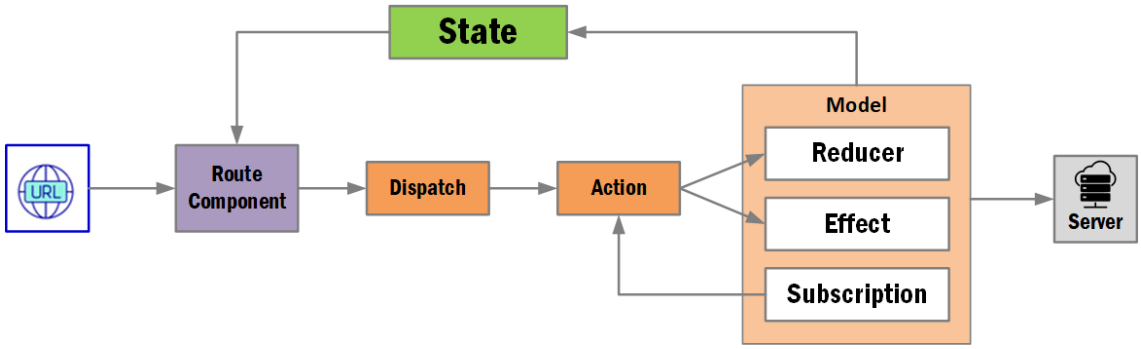
https://www.etatvasoft.com/blog/react-state-management/

Un estado complejo es aquel que tiene una estructura de varios niveles

Ejemplo: un objeto de objetos que contiene arrays.

```
{
  foo: {
    faa: {
      test: [],
      testB: ''
    }
  },
  fee: [],
  fii: {
    testC: [],
    testD: {}
  }
}
```

```
{
  "name": "Jo\u00E3o Soares",
  "age": 30,
  "isAlive": true,
  "languages": [
    "Portugues",
    "Ingles"
  ],
  "lastAccess": "2022-04-04T20:05:22.3840251+02:00",
  "addresses": {
    "Home": {
      "street": "Rua dos Lobos, 1",
      "city": "Lisboa",
      "state": null,
      "zip": null
    },
    "Office": {
      "street": "Avenida de Vigo, 34",
      "city": "Madrid",
      "state": null,
      "zip": "28080"
    }
  },
  "extraInfo": null
}
```



Sintaxis:

```
const reducerFunction = (state, action) => {
    switch (action.type) {
        case 'option1':
            return expression;
        case 'option2':
            return expression;
        default
            return state;
    }
};
```

```
const [state, dispatch] = useReducer(reducerFunction, initialState)
```

```
dispatch({ type: option2 });
```

Usar el useReducer

1. **Importar useReducer:**

```
import { useReducer } from 'react'
```

2. **Crear el reductor**

El reductor es una función que recibe el estado actual y una acción, y devuelve el nuevo estado.

```
function reducer(state, action) {
    // ...
}
```

3. **Inicializar el hook useReduce**

El hook useReducer recibe dos argumentos: una función reductor y un valor inicial. El valor inicial es el estado inicial de la aplicación.

```
const [state, dispatch] = useReducer(reducer, initialState, init?)
```

El hook useReducer devuelve un array con dos elementos: el estado actual y una función dispatch. La función dispatch recibe una acción, y utiliza el reductor para actualizar el estado.

4. **Ejecutar el useReduce**

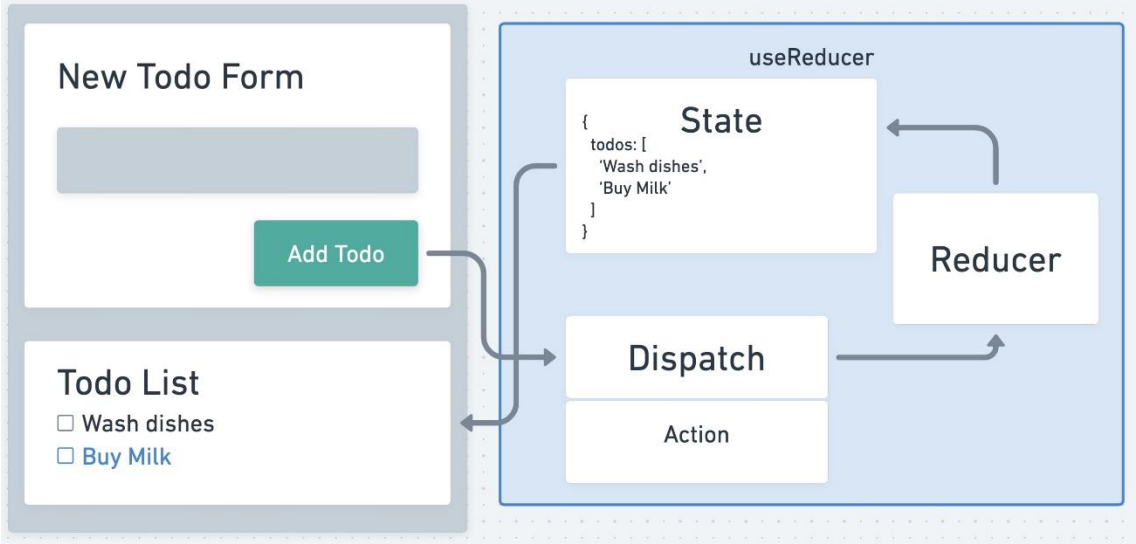
Se ejecuta con la función dispatch de la siguiente manera

```
dispatch({ type: 'increment' })
```

Diferencia entre useState y useReducer

```
const [loading, setLoading] = useState(false);
const [error, setError] = useState(false);
const [post, setPost] = useState({});
```

```
const STATE = {
  loading: false, error: false, post: {},
};
const [state, dispatch] = useReducer(reducer, STATE);
```




```
import React, { useReducer } from 'react'
const Ejem1UseReduce = () => {
    const [todos, dispatch] = useReducer(reducer, initialTodos);

    const handleComplete = (todo) => {
      dispatch({ type: "COMPLETE", id: todo.id });
    };

    return (
      <>
        {todos.map((todo) => (
          <div key={todo.id}>
            <label>
              <input
                type="checkbox"
                checked={todo.complete}
                onChange={() => handleComplete(todo)}
              />
              {todo.title}
            </label>
          </div>
        ))}
      </>
    );
  }
export default Ejem1UseReduce
```

```
const initialTodos = [
  {
    id: 1,
    title: "Todo 1",
    complete: false,
  },
  {
    id: 2,
    title: "Todo 2",
    complete: false,
  },
];
```

```
const reducer = (state, action) => {
  switch (action.type) {
    case "COMPLETE":
      return state.map((todo) => {
        if (todo.id === action.id) {
          return { ...todo, complete: !todo.complete };
        } else {
          return todo;
        }
      });
    default:
      return state;
  }
};
```

8. HOOK useNavigate

El hook useNavigate es una nueva incorporación a React Router Dom 6.

Es un reemplazo de los hooks *useHistory* y *useLocation* en versiones anteriores de React Router.

> El hook useNavigate proporciona una API simple e intuitiva para navegar entre páginas en su aplicación React y simplifica el proceso de manejo de cambios de una URL en su aplicación.

Usar useNavigate

1. **Importar useNavigate:**

```
import { useNavigate } from "react-router-dom"
```

2. **Crear useNavigate**

```
const navigateTo = useNavigate();
```

3. **Navegar**

```
useNavigate("/employee")
```

Usar -1 o +1 para avanzar o retroceder en el history.

9. HOOK useParams

El hook useParams devuelve un objeto de pares clave/valor de los parámetros dinámicos de la URL actual que coinciden con el <Route path>. Las rutas secundarias heredan todos los parámetros de sus rutas principales.

Usar useParams

1. **Importar useParams:**

```
import { useParams } from 'react-router-dom'
```

2. **Crear useParams**

```
let { userId } = useParams()
```

Debe coincidir con el <Route path>

```
<Route path="users">
    <Route path=":userId" element={<ProfilePage />} />
    <Route path="me" element={...} />
</Route>
```

Evaluación de competencia

1. ¿Cómo maneja React la persistencia del estado entre renders cuando se usa useState comparado con useRef?
2. ¿Por qué useEffect puede ejecutar su callback más veces de lo esperado y cómo puedes evitarlo?
3. ¿Cómo podrías replicar el comportamiento de componentDidUpdate con hooks sin generar efectos secundarios innecesarios?
4. ¿Qué problemas pueden surgir al usar useEffect para manejar llamadas a APIs y cómo los abordarías?
5. ¿En qué casos useCallback podría ser innecesario y potencialmente contraproducente en una aplicación?
6. ¿Cómo React maneja internamente la limpieza de efectos en useEffect al desmontar un componente?
7. ¿Cómo podrías evitar el re-render innecesario de un componente que usa useMemo con dependencias incorrectamente gestionadas?
8. ¿Cuál es la diferencia fundamental entre useReducer y useState, y cuándo preferirías uno sobre el otro?
9. ¿Cómo podrías compartir lógica entre múltiples componentes usando hooks personalizados sin afectar la reactividad de la aplicación?
10. ¿Cómo afecta el orden de los hooks dentro de un componente a su ejecución y qué problemas podrían ocurrir si se usan condicionalmente?
11. Implementar un formulario CRUD para una tabla productos (idproduct, nameproduct, description, price, photo). Utilizando UseContext y UseReducer.