REACT

Las Referencias Hook

0.- Introducción

Recordemos primeramente que un Hook en React es una función que permite utilizar el estado y otros métodos de React dentro de un componente funcional. Los Hooks permiten agregar funcionalidad a los componentes de React sin necesidad de utilizar clases. Se reconocen por tener todos el **prefijo `use`**. Los Hooks no funcionan dentro de las clases, permiten usar React sin clases. Veamos algunos de ellos importantes:

1.- useState

Permite agregar una variable de estado a tu componente. Se utiliza para agregar estado a componentes funcionales en React.

Se invoca con el estado inicial, y devuelve un par: el estado actual y una función para actualizarlo. Esta función de actualización, cuando se llama, desencadena un nuevo renderizado con el estado actualizado. `useState` sólo puede ser llamado en el nivel superior de tu componente o en tus propios Hooks.

Aquí tienes un ejemplo de cómo se usa:

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

function MyComponent() {
  const [age, setAge] = useState(28);
  const [name, setName] = useState('Taylor');

  // ...
}
```

En este ejemplo, `useState` se utiliza para declarar dos variables de estado: `age` y `name`. Cada llamada a `useState` devuelve un par de valores: el estado actual y una función que te permite actualizar ese estado[1].

La convención es nombrar las variables de estado como `[algo, setAlgo]` usando la desestructuración de arrays.

El valor que pasas a `useState` es el valor inicial del estado. React llamará a tu función de inicialización al inicializar el componente y almacenará su valor de devolución como el estado inicial.

La función `set` (como `setAge` o `setName` en el ejemplo) te permite actualizar el estado a un valor diferente y desencadenar un nuevo renderizado.

Es importante recordar que `useState` es un Hook, por lo que solo puedes llamarlo en el nivel superior de tu componente o en tus propios Hooks. No puedes llamarlo dentro de bucles o condiciones.

Ejemplo de formulario:

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

function Formulario() {
  const [nombre, setNombre] = useState('');
  const [correo, setCorreo] = useState('');

  const manejarCambioNombre = (evento) => {
    setNombre(evento.target.value);
  };

  const manejarCambioCorreo = (evento) => {
    setCorreo(evento.target.value);
  };

  const manejarEnvio = (evento) => {
```

```
    evento.preventDefault();
    console.log(`Nombre: ${nombre}, Correo: ${correo}`);
  };

  return (
    <form onSubmit={manejarEnvio}>
      <label>
        Nombre:
        <input type="text" value={nombre} onChange={manejarCambioNombre} />
      </label>
      <label>
        Correo:
        <input type="email" value={correo} onChange={manejarCambioCorreo} />
      </label>
      <input type="submit" value="Enviar" />
    </form>
  );
}
```

En este ejemplo, useState se utiliza para manejar el estado de los campos de entrada del formulario. Cuando el usuario escribe en los campos de entrada, las funciones manejarCambioNombre y manejarCambioCorreo se disparan, actualizando el estado con los nuevos valores introducidos por el usuario. Cuando el formulario se envía, la función manejarEnvio se dispara, mostrando los valores actuales del estado en la consola. Este es un patrón común para manejar formularios en React con useState.

2.- useEffect

Permite sincronizar un componente con un sistema externo. Se invoca con una función de configuración y una lista de dependencias. Cuando las dependencias cambian, React ejecuta primero la función de limpieza (si la proporcionaste) con los valores antiguos, y luego ejecuta tu función de configuración con los nuevos valores. Después de que tu componente se elimina del DOM, React ejecuta tu función de limpieza una última vez.

La forma de llamarla es:

```
useEffect(() => {
   <código>
  };
}, [<lista de dependencias>]);
```

Cada vez que alguna de las variables de [<lista de dependencias>] son modificadas en la página se ejecutará el <código> y se renderiza la página con esos nuevos valores.

¡Importante!
Sí [<lista de dependencias>] está vacío el userEffec sólo se ejecutará la primera vez que se carga el componente.

NOTA
En un componente puede hacer uso de varios userEffect.

Aquí tienes dos ejemplos de cómo se puede usar `useEffect`:

Ejemplo 1: Conexión a un sistema externo

```
import { useEffect, useState } from 'react';
import { createConnection } from './chat.js';

function ChatRoom({ roomId }) {
  const [serverUrl, setServerUrl] = useState('https://localhost:1234');

  useEffect(() => {
    const connection = createConnection(serverUrl, roomId);
    connection.connect();
    return () => {
      connection.disconnect();
    };
  }, [serverUrl, roomId]);
}
```

En este ejemplo, `useEffect` se utiliza para conectar y desconectar una sala de chat cuando cambian `serverUrl` o `roomId`.

Ejemplo 2: Actualización del título del documento

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

function App() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `Haz clic ${count} veces`;
  }, [count]);
```

```
}
```

En este ejemplo, `useEffect` se utiliza para actualizar el título del documento cada vez que cambia el estado `count`.

4.- useRef

`useRef` es un Hook de React que permite referenciar un valor que nos provoca un nuevo renderizado. Se utiliza para almacenar valores mutables y para acceder a elementos del Modelo de Objetos del Documento (DOM). `useRef` devuelve un objeto mutable con una única propiedad `current`. A diferencia del estado, `useRef` es mutable y su cambio no provoca un nuevo renderizado.

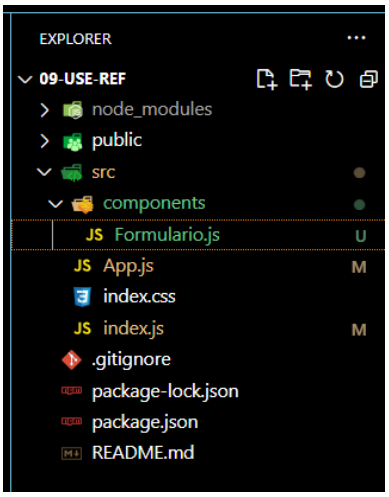
Hace una referencia con un elemento del DOM. Básicamente, al final lo que hace es devolver un objeto de referencia mutable que va a tener una propiedad.

Es decir, que mientras que el componente esté montado ese, esa referencia va a existir, va a persistir en el tiempo y cuando se desmonte esa referencia va a dejar de existir.

4.1.- Ejemplo con formulario

Vamos por ejemplo a predefinir un dato previamente de un formulario, por ejemplo que en nombre ponga David por defecto. Y vamos a comprobar que aunque yo en el input “nombre” ponga otro siempre saldrá el predefinido.

A.- Creo la estructura de carpetas:



B.- Crea también el fichero Formulario.js y añade un formulario simple:

```
import React from 'react'
```

```
const Formulario = () => {
  return (
    <div>
        <h1>Formulario</h1>

         <form>
            <input type="text" placeholder='Nombre' /><br/>
            <input type="text" placeholder='Apellidos' /><br/>
                  <input type="email" placeholder='Correo electrónico'
/><br/>
            <input type="submit" placeholder='Enviar' />
         </form>
    </div>
  )
}

export default Formulario
```

C.- Vamos hacer uso de useRef. Para ello creamos una constante nombre al que le pasamos el valor “David” al useRef:

```
const nombre = useRef(“David”)
```

D.- Al formulario le añadimos el onSubmit para que llame a una función, en este caso “mostrar”:

```
<form onSubmit={mostrar}>
```

E.- Creamos la función “mostrar” capturando el evento (e), añadiendo preventDefault() para que no recargue la pantalla y ponemos un console.log(nombre) para ver el valor de la variable.

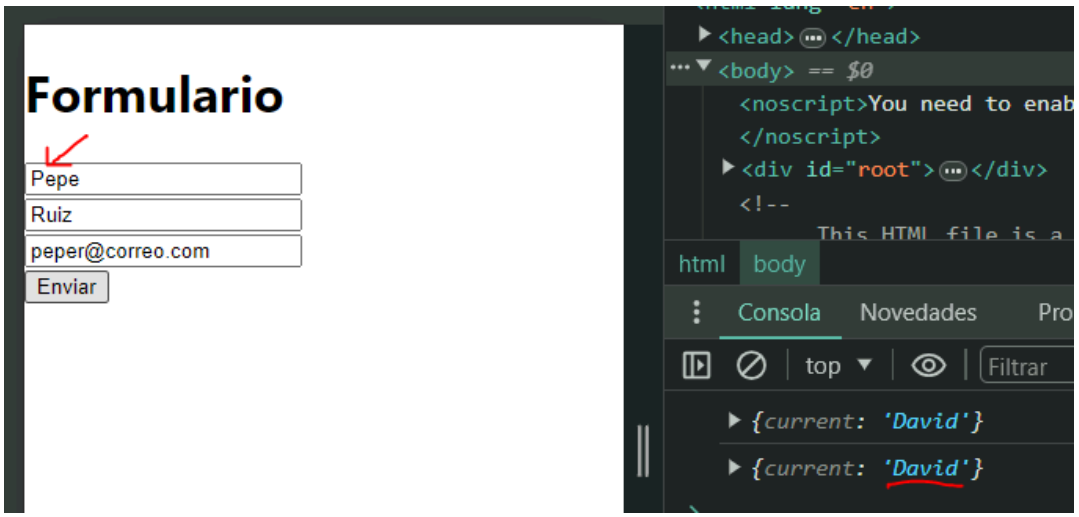
```
const mostrar = e => {
        e.preventDefault();
        console.log(nombre);
}
```

El código queda:

```
import React, { useRef } from 'react'
```

```
const Formulario = () => {

  const nombre = useRef("Victor");

  const mostrar = e => {
    e.preventDefault();
    console.log(nombre);
  }
  return (
    <div>
        <h1>Formulario</h1>

         <form onSubmit={mostrar}>
            <input type="text" placeholder='Nombre' /><br/>
            <input type="text" placeholder='Apellidos' /><br/>
                  <input type="email" placeholder='Correo electrónico'
/><br/>
            <input type="submit" placeholder='Enviar' />
         </form>
    </div>
  )
}
export default Formulario
```

Si ejecutamos el programa con npm start y tratamos de poner unos datos en el formulario:



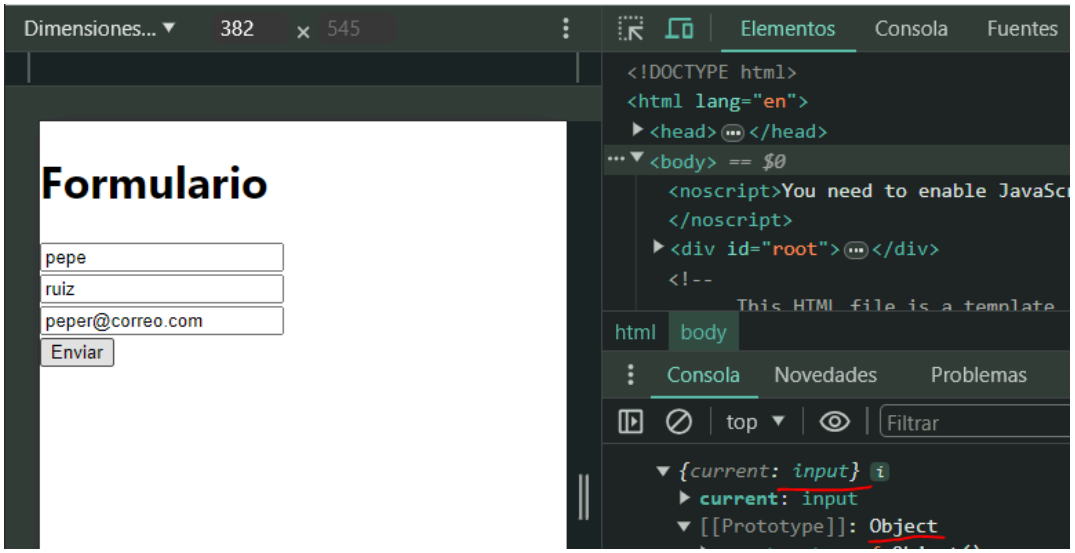
Por consola nos devuelve el nombre predefinido con useRef.

4.1.1.- ¿Qué ocurre si al campo input le paso el ref de la variable?

Si yo hago:

```
<input type="text" placeholder='Nombre'  ref={nombre}  /><br/>
```

¿Qué va a ocurrir?



Lo que ocurre es que la variable “nombre” tiene el objeto input.

Con lo cual yo podría acceder al objeto current y acceder al value:

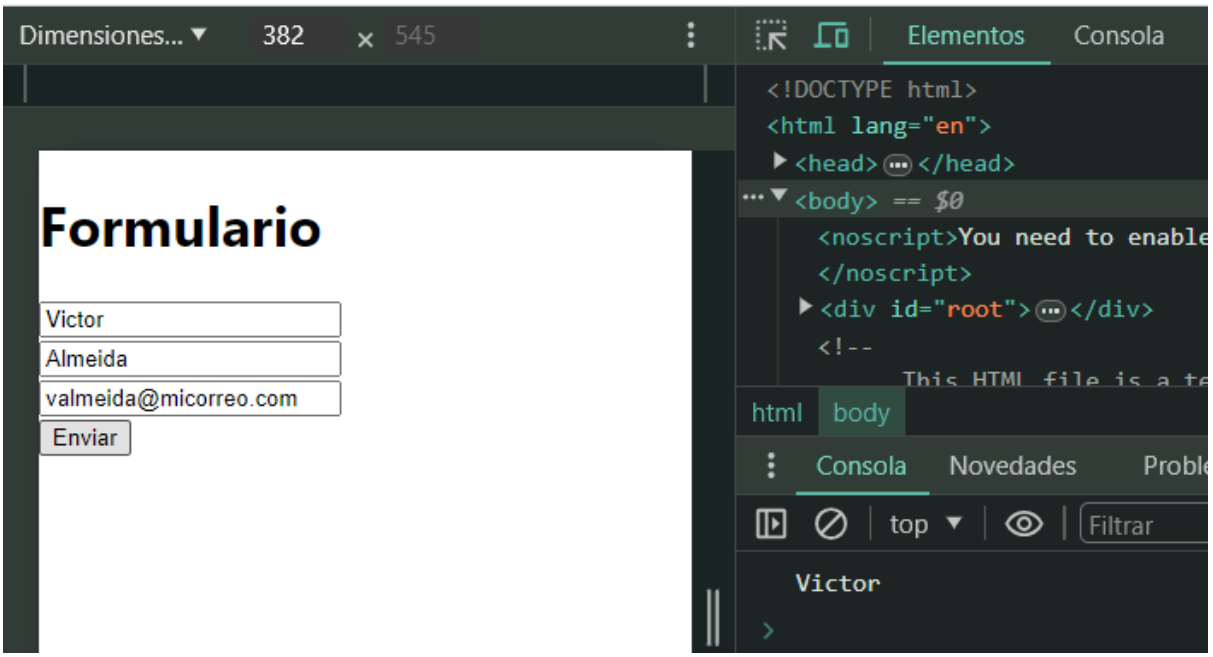
```
console.log(nombre.current.value);
```

El código queda como:

```
import React, { useRef } from 'react'

const Formulario = () => {

  const nombre = useRef("David");

  const mostrar = e => {
    e.preventDefault();
```

```
    console.log(nombre.current.value);
  }

  return (
    <div>
        <h1>Formulario</h1>

         <form onSubmit={mostrar}>
            <input type="text" placeholder='Nombre' ref={nombre}/><br/>
            <input type="text" placeholder='Apellidos' /><br/>
            <input type="email" placeholder='Correo electrónico'
/><br/>
            <input type="submit" placeholder='Enviar' />
         </form>
    </div>
  )
}

export default Formulario
```

Y la ejecución de la aplicación es:



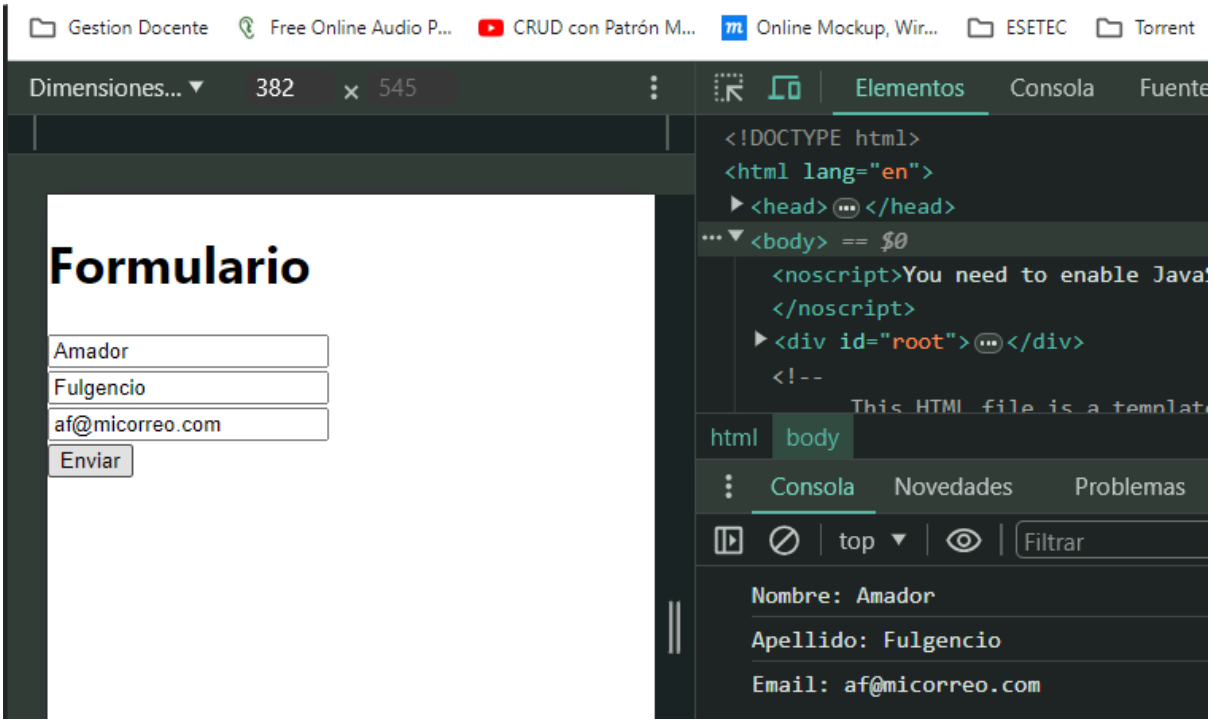
Observamos que ¡ahora sí! que tenemos el nombre puesto en formulario y no que usa useRef por defecto.

Esta, es pues, una manera de acceder a los campos de un formulario.

Creemos el resto de referencias, ya usando el useRef vacío:

```
import React, { useRef } from "react";

const Formulario = () => {
  const nombreValue = useRef();
  const apellidoValue = useRef();
  const emailValue = useRef();

  const mostrar = (e) => {
    e.preventDefault();
    console.log("Nombre: " + nombreValue.current.value);
    console.log("Apellido: " + apellidoValue.current.value);
    console.log("Email: " + emailValue.current.value);
  };

  return (
    <div>
      <h1>Formulario</h1>

      <form onSubmit={mostrar}>
        <input type="text" placeholder="Nombre" ref={nombreValue} /><br
/>
        <input type="text" placeholder="Apellidos" ref={apellidoValue}
/><br />
        <input type="email" placeholder="Correo electrónico"
ref={emailValue} /><br />
        <input type="submit" placeholder="Enviar" />
      </form>
    </div>
  );
};

export default Formulario;
```

Y por consola obtendremos:



4.1.2.- Conclusión

Esta puede ser una manera de acceder a los datos de un formulario para luego procesarlos o enviarlos a una base de datos.

4.2.- Ejemplo con etiqueta

¿Qué ocurre si uso la referencia “ref” en una etiqueta?

Veamos qué ocurre. Para ello usemos una etiqueta :

```
<div ref={miCaja}>
        <h2>Pruebas con useRef</h2>
</div>
```

Creamos la referencia:

```
const miCaja = useRef();
```

Voy a darle un poco de estilo CSS que añadiremos al final del fichero index.css:

```
..miCaja{
   border: 3px solid white;
   padding: 10px;
   margin: 15px;
}
```

Y aplicamos al div:

```
<div ref={miCaja}>
```

Cuando yo envíe el formulario yo tendré acceso al useForm del div y podré cambiar cualquier propiedad, como el color de fondo, la línea o que se añade otra clase de estilos CSS. Por ejemplo, vamos a añadir una clase CSS para cambiar el fondo del div a verde:

4.2.1.- Cambiar color de fondo del div al enviar formulario

A.- Añadimos la clase al objeto miCaja:

```
miCaja.current.classList.add("fondoVerde");
```

B.- Creamos la clase CSS:

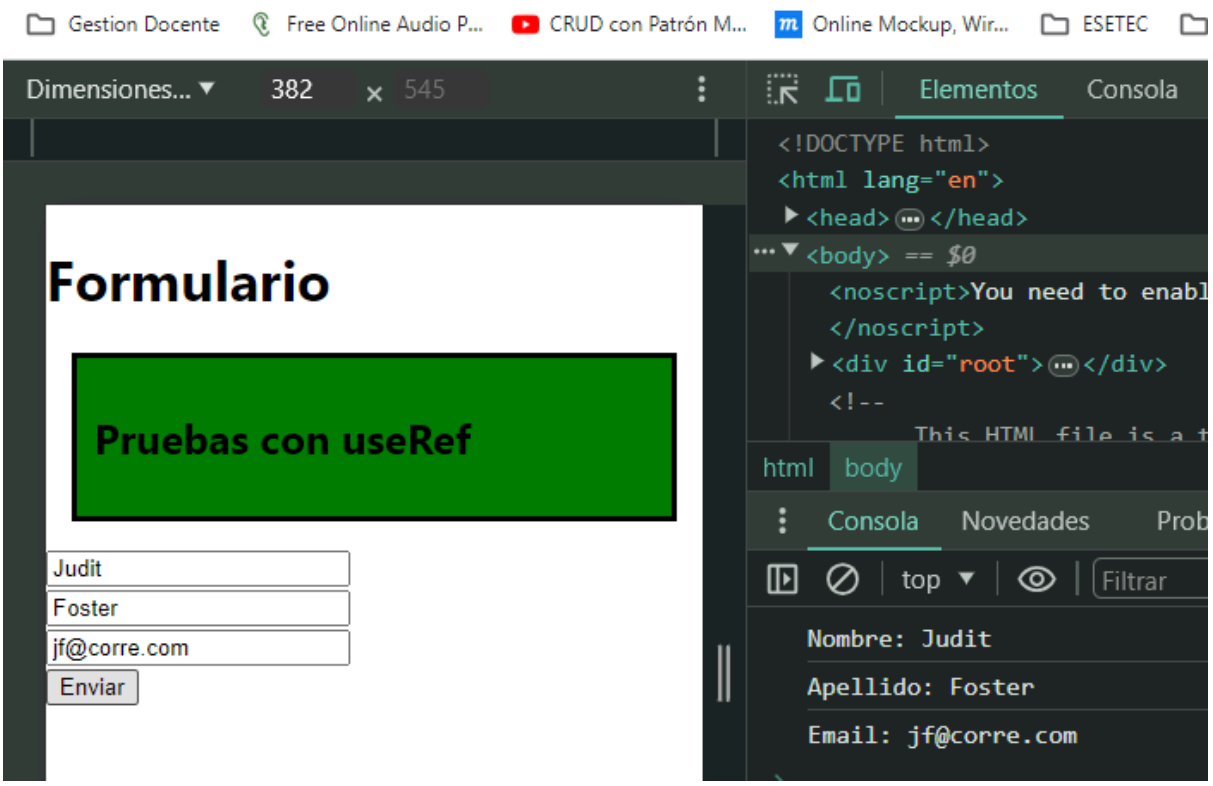
```
.fondoVerde{
   background-color: green;
}
```

El código queda:

```
import React, { useRef } from "react";

const Formulario = () => {
  const nombreValue = useRef();
  const apellidoValue = useRef();
  const emailValue = useRef();
  const miCaja = useRef();


  const mostrar = (e) => {
    e.preventDefault();
    console.log("Nombre: " + nombreValue.current.value);
```

```
    console.log("Apellido: " + apellidoValue.current.value);
    console.log("Email: " + emailValue.current.value);

    // miCaja
    miCaja.current.classList.add("fondoVerde");

  };

  return (
    <div>
      <h1>Formulario</h1>

      <div ref={miCaja} className="miCaja">
        <h2>Pruebas con useRef</h2>
      </div>

      <form onSubmit={mostrar}>
        <input type="text" placeholder="Nombre" ref={nombreValue} /><br
/>
        <input type="text" placeholder="Apellidos" ref={apellidoValue}
/><br />
        <input type="email" placeholder="Correo electrónico"
ref={emailValue} /><br />
        <input type="submit" placeholder="Enviar" />
      </form>
    </div>
  );
};

export default Formulario;
```

La salida por consola, tras enviar formulario sale:

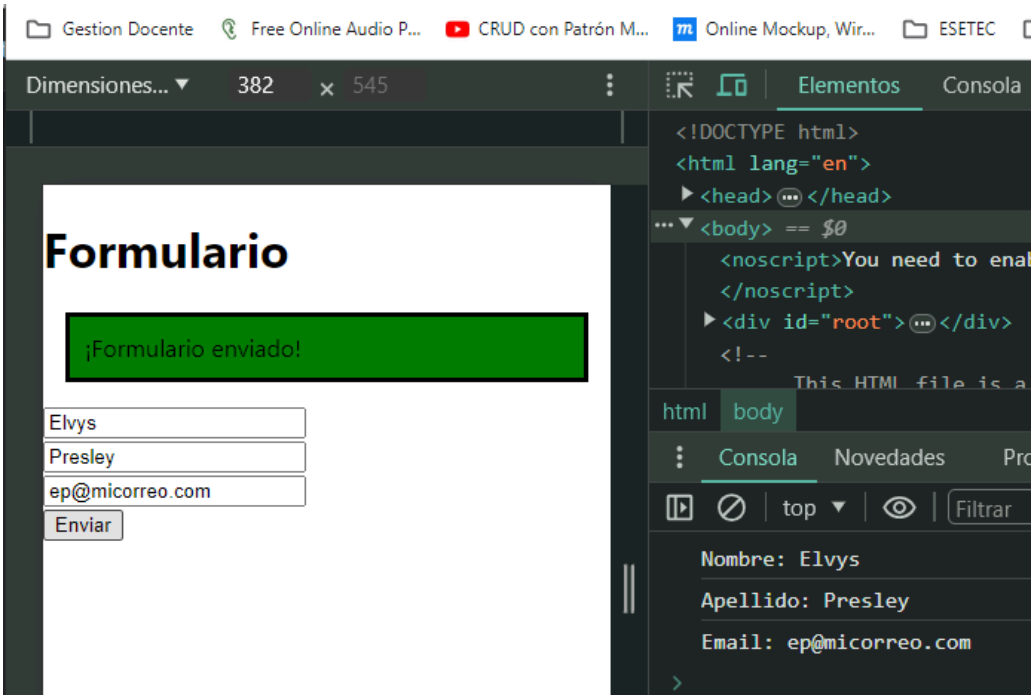


4.2.2.- Cambiar texto del div al enviar formulario

Y si quiero cambiar el texto del div al enviar el formulario y que aparezca: “¡Formulario enviado!”. Tan solo tendría que añadir:

```
miCaja.current.innerHTML = "¡Formulario enviado!";
```

La salida será:



4.2.3.- Conclusión

Hemos hecho el ejemplo con un div pero se puede hacer con cualquier objeto y acceder a sus propiedades para modificarlo.

5.- useMemo

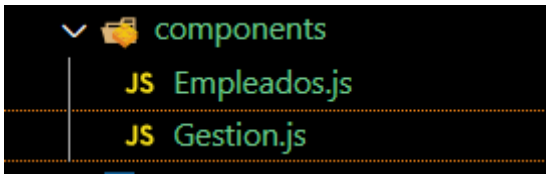
La función useMemo que vamos a ver a continuación nos va a permitir memorizar un componente, es decir, memorizar el resultado que imprime por pantalla un componente. Y solamente se va a volver a ejecutar el componente memorizado cuando realmente sus propiedades se modifiquen o cuando realmente sea necesario y no se ejecute.

5.1.- Ejemplo de gestión de empleados

Trataremos de traer elementos de una base de datos alojada en un servidor externo (en verdad vamos a usar una web que ofrece datos para pruebas de aplicaciones, es https://reqres.in/) y los mostraremos en nuestra web. Pero sólo los mostraremos la primera vez que cargue la web o cuando nosotros queramos y no cada vez que se renderice la página.

Comprobemos primeramente que cada vez que renderizamos una página lo harán también los componentes que en él se encuentren:

Creamos dos componentes:



En Gestion.js escribimos el siguiente código:

```
import React, { useRef, useState } from "react";
import Empleados from "./Empleados";

const Gestion = () => {
  const [nombre, setNombre] = useState("");
  const gestorInput = useRef();

  const asignarGestor = (e) => {
    setNombre(gestorInput.current.value);
  };

  return (
    <div>
      <h1>Nombre del gestor: {nombre}</h1>

      <input
        type="text"
        ref={gestorInput}
        onChange={asignarGestor}
        placeholder="Introduce tu nombre de gestor"
      />

      <h2>Listado de empleados</h2>
      <p>Los usuarios son gestionados por {nombre} vienen de
jasonplaceholder</p>
      <Empleados />
    </div>
  );
};
```

Explicación:

a.- Vemos que en h1 y h2 tenemos la variable “nombre” para que se actualice con cada modificación del componente. Para ello usamos la declaración de un useState(“”) que en este caso lo inicializamos a vacío.

const [nombre, setNombre] = useState("");

b.- En el input hemos puesto un ref (gestorInpu) para recuperar el contenido que se introduzca. Con lo cual definimos la variable:

const gestorInput = useRef();

c.- El mismo input tiene una llamada a la función javascript “asignarGestor” en el onChange. Este método recogerá el ref y lo asignará a la variable nombre a través del setNombre:

```
const asignarGestor = (e) => {
  setNombre(gestorInput.current.value);
};
```

Estos cambios harán que el texto introducción en el input aparezcan en los h1 y h2.

NOTA: Yo podría no usar el ref. Debdo a que la función asignarGestion es llamada por el input. El evento capturado por “e”, hace referencia al elemento input. Y yo puedo recuperar el valor del input a través del target.value. Es decir, dejarlo como:

```
import React, { useRef, useState } from "react";
import Empleados from "./Empleados";

const Gestion = () => {
  const [nombre, setNombre] = useState("");
  const asignarGestor = (e) => {
    setNombre(e.target.value);
  };
  return (
    <div>
      <h1>Nombre del gestor: {nombre}</h1>

      <input
        type="text"
```

```
        onChange={asignarGestor}
        placeholder="Introduce tu nombre de gestor"
      />
      <h2>Listado de empleados</h2>
      <p>
        Los usuarios son gestionados por {nombre} vienen de
jasonplaceholder
      </p>
      <Empleados />
    </div>
  );
};
export default Gestion;
```

Ahora escribamos en Gestion.js:

```
console.log("Se ha renderizado Gestion");
```

Y en empleados

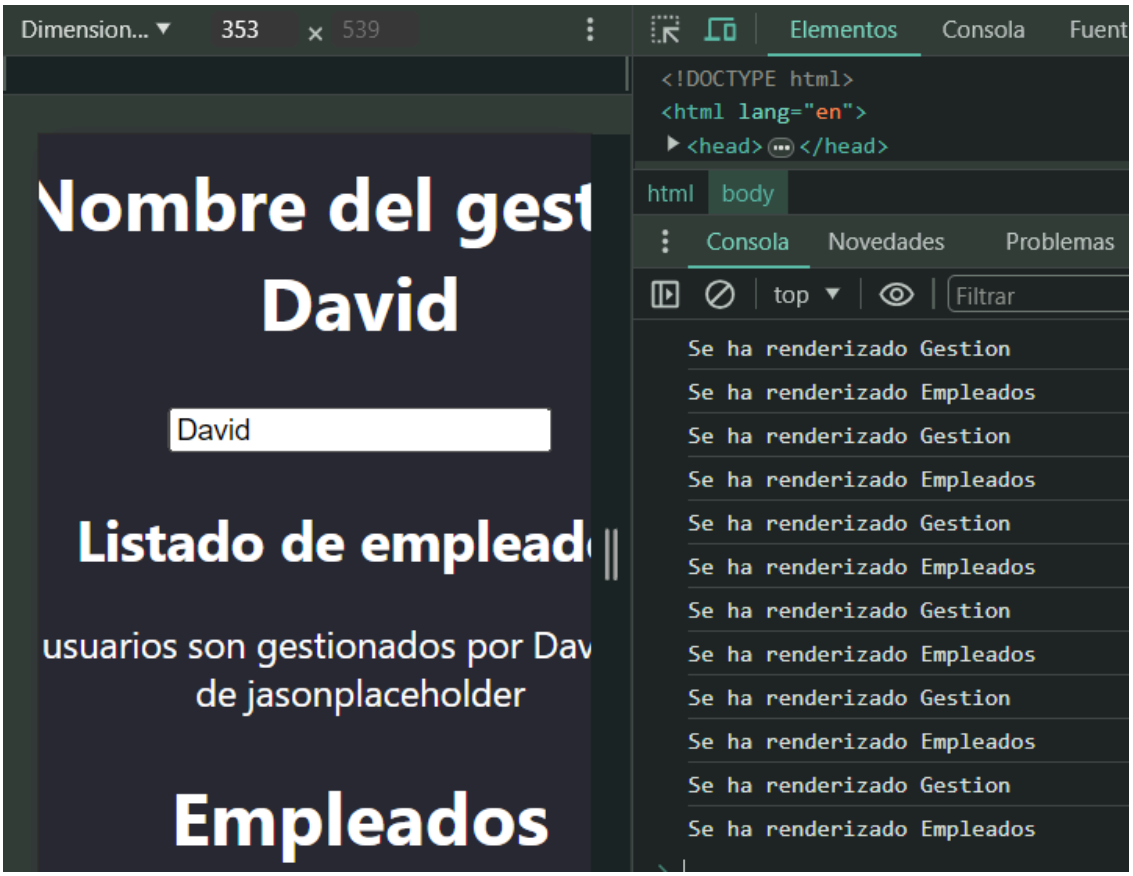
```
console.log("Se ha renderizado Empleados");
```

Y ejecutemos la aplicación.

Notemos que a cada cambio se renderiza Gestion y Empleados.

Inicialmente tenemos:

Tras introducir un nombre: David



Vemos que “Empleados” también se renderiza cuando nada más que hay un texto que no haría falta actualizarlo, solamente al inicio.

Para evitar esto haremos uso de useMemo en el componente Empleados:

```
import React from "react"
const Empleados = React.memo(
    () => {
        console.log("Se ha renderizado Empleados");
        return (
            <div>
            <h1>Empleados</h1>
            </div>
        );
    }
);
export default Empleados;
```

Ahora si introducimos un nombre en el input para renderizar la página a cada cambio vemos que Empleados sólo se carga en la primera renderización:

Comprobado que, efectivamente, el uso de React.memo hace que los componentes "hijos" (en este caso Empleados) sólo se cargan al inicio ahora vamos a traernos esos datos de la web que nos proporciona datos de ejemplo.

```
import React, { useEffect } from "react";
const Empleados = React.memo(
    () => {
        console.log("Se ha renderizado Empleados");
        const conseguirempleados = async() => {
            const url ="https://reqres.in/api/users?page=1";
            const peticion = await fetch(url);
            const {data : empleados} = await peticion.json();
            console.log(empleados);
        }
        useEffect( () => {
            conseguirempleados();
        }, []);
        return (
            <div>
            <h1>Empleados</h1>
            </div>
        );
    }
);
export default Empleados;
```

Explicación del código:

a.- Cuando cargue por primera vez, useEffect pasandole el array vacío "[]" se llamará a la función conseguirempleados():

```
useEffect( () => {
    conseguirempleados();
}, []);
```

b.- La función conseguirempleados() recoje los datos de la url y los datos del json obtenido (data) se lo pasamos a la variable "empleados" y los mostramos por consola:

```
const conseguirempleados = async() => {
    const url ="https://reqres.in/api/users?page=1";
    const peticion = await fetch(url);
    const {data : empleados} = await peticion.json();
    console.log(empleados);
}
```

Esto es lo que obtenemos:

Bien, ahora vamos a procesar los datos para mostrarlo en la web:

```
import React, { useEffect, useState } from "react";

const Empleados = React.memo(() => {
  const [empleados, setEmpleados] = useState([]);

  useEffect(() => {
    conseguirempleados();
  }, []);

  const conseguirempleados = async () => {
    const url = "https://reqres.in/api/users?page=1";
    const peticion = await fetch(url);
    const { data: empleados } = await peticion.json();
    setEmpleados(empleados);
  };

  return (
    <div>
      <h1>Empleados</h1>
      <div className="empleados">
        <ul className="empleados">
          {empleados.map((empleado) => {
            return (
              <li key={empleado.id}>
                {empleado.first_name + " " + empleado.last_name}
              </li>
            );
          })}
        </ul>
      </div>
    </div>
  );
});

export default Empleados;
```

Explicación del código:

En la variable de estado “empleados” están los datos que recorremos con un map, retornando (return) un <li> por cada iteración que tendrá el campo del jason first_name y last_name.

NOTA: Puede ser que la url se demore en devolvernos los datos y la página se renderize y no muestre datos. Para evitarlo podemos esperar a que el array sea mayor que 1:

```
return (
  <div>
    <h1>Empleados</h1>
    <div className="empleados">
      <ul className="empleados">
        {empleados.length >= 1 &&  empleados.map((empleado) => {
          return (
            <li key={empleado.id}>
              {empleado.first_name + " " + empleado.last_name}
            </li>
          );
        })}
      </ul>
    </div>
  </div>
);
```

Así nos aseguramos de que hay datos.

Como extra podemos traernos la segunda hoja de datos de la url.

Para ello crea en Gestion.js dos botones para traernos cada una de las páginas.

```
<button onClick={()=>setPage(1)}>Página 1</button>
<button onClick={()=>setPage(2)}>Página 2</button>
```

Que actualizará el estado de una variable “page”, que por defecto estará a “1”:

```
const [page, setPage] = useState(1);
```

Usaremos los props para pasar al componente Empleados la variable “page”.

```
<Empleados pagina={page}/>
```

Gestion.js queda:

```
import React, { useState } from "react";
import Empleados from "./Empleados";

const Gestion = () => {
  const [nombre, setNombre] = useState("");
  const [page, setPage] = useState(1);

  const asignarGestor = (e) => {
    setNombre(e.target.value);
  };

  return (
    <div>
      <h1>Nombre del gestor: {nombre}</h1>

      <input
        type="text"
        onChange={asignarGestor}
        placeholder="Introduce tu nombre de gestor"
      />

      <h2>Listado de empleados</h2>
      <p>
               Los usuarios son gestionados por {nombre} vienen de
jasonplaceholder
      </p>
      <button onClick={()=>setPage(1)}>Página 1</button>
      <button onClick={()=>setPage(2)}>Página 2</button>
      <Empleados pagina={page}/>
    </div>
  );
};

export default Gestion;
```

Ahora en Empleados.js

Recuperamos el props en la variable “pagina”, recordando hacer uso de React.memo:

```
const Empleados = React.memo(
        ({pagina}) => {
```

La ponemos en el userEffect para que cada vez que se recargue la página la variable se actualice:

```
useEffect(() => {
        conseguirempleados(pagina);
}, [pagina]);
```

Dentro del método de javascript conseguirempleados(pagina) recuperamos “pagina”, pero renombrada como “p” y la concatenamos a la url ya que es el sufijo que indica qué páginas solicitamos:

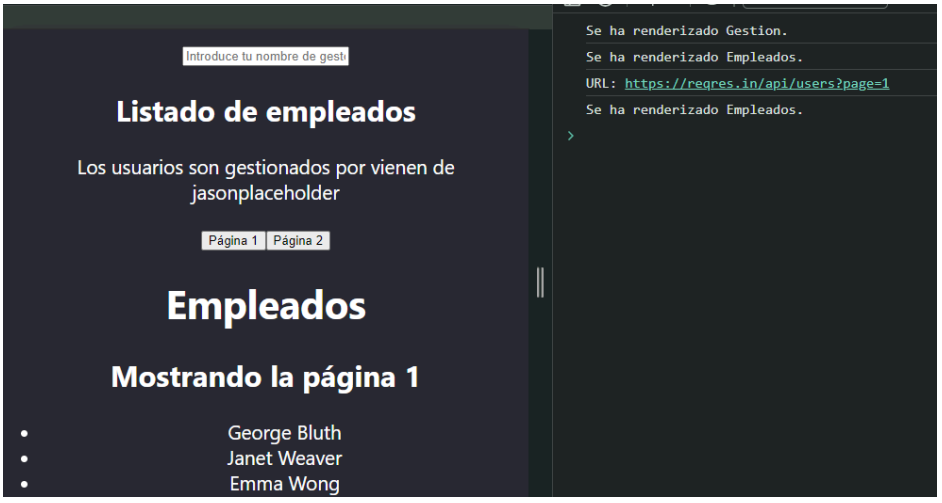
```
const conseguirempleados = async (p) => {
  const url = "https://reqres.in/api/users?page="+(p);
  console.log("URL: " + url);
  const peticion = await fetch(url);
  const { data: empleados } = await peticion.json();
  setEmpleados(empleados);
};
```

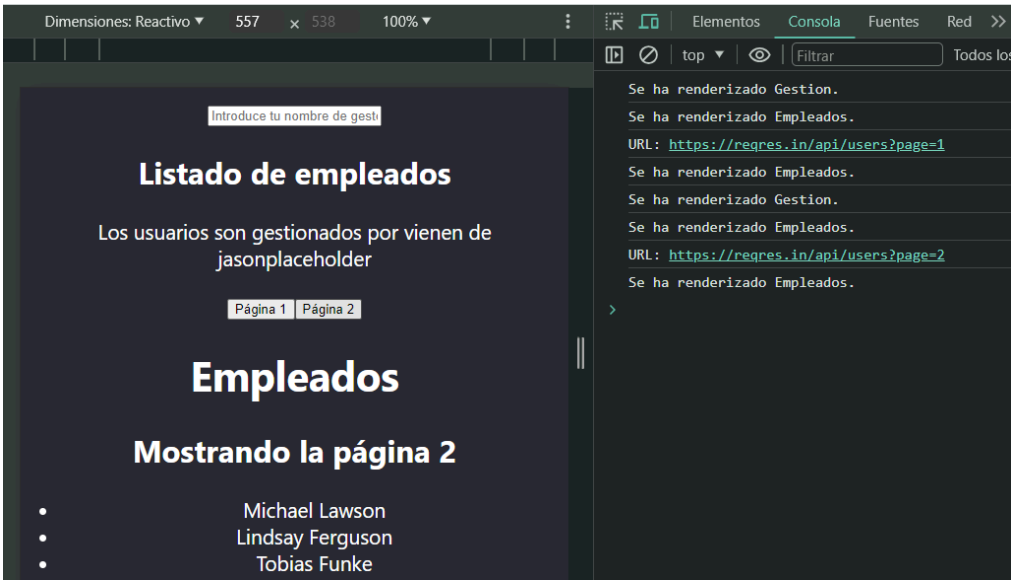
Finalmente Empleados.js queda como:

```
import React, { useEffect, useState } from "react";

const Empleados = React.memo(
({pagina}) => {
  const [empleados, setEmpleados] = useState([]);

  useEffect(() => {
    conseguirempleados(pagina);
  }, [pagina]);

  const conseguirempleados = async (p) => {
    const url = "https://reqres.in/api/users?page="+(p);
    console.log("URL: " + url);
    const peticion = await fetch(url);
```

```
    const { data: empleados } = await peticion.json();
    setEmpleados(empleados);
  };

  return (
    <div>
      <h1>Empleados</h1>
      <div className="empleados">
        <h2>Mostrando la página {pagina}</h2>
        <ul className="empleados">
          {empleados.length >= 1 &&
            empleados.map((empleado) => {
              return (
                <li key={empleado.id}>
                  {empleado.first_name + " " + empleado.last_name}
                </li>
              );
            })}
        </ul>
      </div>
    </div>
  );
});

export default Empleados;
```

En la salida se observa:





NOTA: el console.log hace que renderice la página. Si lo queremos evitar tendríamos que meterlo dentro de un userEffect con el componente que se actualice realmente.

Utilidades

https://reqres.in/ → Una REST-API alojada lista para responder a sus solicitudes AJAX.

Bibliografía

Hooks

- Hooks en React - Marta González. https://martagonzalez.dev/blog/hooks-en-react/.
- React Hooks: ¿qué son y cómo usarlos?. https://community.listopro.com/react-hooks-que-son-y-como-usarlos/.
- Un vistazo a los Hooks – React - reactjs.org. https://es.legacy.reactjs.org/docs/hooks-overview.html.
- Reglas de los hooks de React | KeepCoding Bootcamps. https://keepcoding.io/blog/reglas-de-los-hooks-de-react/.
- Hooks: la adición de React que sustituye a los Class Components. https://rootstack.com/es/blog/hooks-react/.

useState

- React. https://es.react.dev/reference/react/useState.
- React - GitHub Pages. https://react.dev/reference/react/useState.
- Entendiendo los Hooks de React, ¿Cómo usar useState y useEffect en .... https://desarrollofront.medium.com/entendiendo-los-hooks-de-react-c%C3%B3mo-usar-usestate-y-useeffect-en-nuestros-componentes-611b9e826dfa.
- useState in React: A complete guide - LogRocket Blog. https://blog.logrocket.com/guide-usestate-react/.

userEffect

- React. https://es.react.dev/reference/react/useEffect.
- React. https://react.dev/reference/react/useEffect.
- useEffect, usando el Hook de efecto en React - Itequia. https://itequia.com/es/useeffect-usando-el-hook-de-efecto-en-react/.
- Entendiendo los Hooks de React, ¿Cómo usar useState y useEffect en .... https://desarrollofront.medium.com/entendiendo-los-hooks-de-react-c%C3%B3mo-usar-usestate-y-useeffect-en-nuestros-componentes-611b9e826dfa.
- useEffect en React | KeepCoding Bootcamps. https://keepcoding.io/blog/useeffect-en-react/.

useRef

- React. https://es.react.dev/reference/react/useRef.
- Comprender el Hook useRef en React - Kinsta®. https://kinsta.com/es/base-de-conocimiento/react-useref/.

- ¿Cómo rayos funciona el hook useRef en React? - DEV Community. https://dev.to/duxtech/como-rayos-funciona-el-hook-useref-en-react-2lah
- Hook useRef en React - Pablo Monteserín. https://pablomonteserin.com/curso/react/useref/.
- useRef en React | KeepCoding Bootcamps. https://keepcoding.io/blog/useref-en-react/.