FullStack Developer

React – Formularios y onSubmit, estilos con Bootstrap y css puro

Formularios

Formularios Controlados
En un formulario controlado, cada campo del formulario tiene su valor gestionado por el estado de React. Esto significa que el valor del input es gestionado por React y actualizado cada vez que el usuario escribe, lo cual permite un control preciso y en tiempo real del valor de cada input.
Características:
•**Estado en React**: Usamos useState para almacenar y manejar el valor de cada campo del formulario.
•**Eventos onChange**: Cada input está ligado a un evento onChange que actualiza el estado con el valor actual.
•**Control en tiempo real**: Cada vez que el usuario interactúa con el formulario, los valores se actualizan en el estado.

Ventajas:
•Control completo sobre el valor de los inputs en todo momento.
•Posibilidad de hacer validaciones en tiempo real.
•Facilita la modificación de valores o la implementación de lógicas complejas.

Cuando Usar:

•Cuando necesitas validaciones complejas, interactividad en tiempo real o feedback inmediato para el usuario.

•Formularios largos o con múltiples interacciones.

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

function FormularioControlado() {
  const [nombre, setNombre] = useState("");
  const [email, setEmail] = useState("");

  const manejarEnvio = (e) => {
    e.preventDefault();
    console.log(`Nombre: ${nombre}, Email: ${email}`);
  };

  return (
    <form onSubmit={manejarEnvio}>
      <label>
        Nombre:
        <input
          type="text"
          value={nombre}
          onChange={(e) => setNombre(e.target.value)}
        />
      </label>
      <label>
        Email:
        <input
          type="email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
        />
      </label>
      <button type="submit">Enviar</button>
    </form>
  );
}

export default FormularioControlado;
```

Ejemplo

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

function FormularioCompleto() {
  const [formulario, setFormulario] = useState({
    nombre: '',
    email: '',
    password: ''
  });

  const [errores, setErrores] = useState({});
  const [esEnviado, setEsEnviado] = useState(false);

  // Manejar cambios en los campos
  const manejarCambio = (e) => {
    const { name, value } = e.target;
    setFormulario({
      ...formulario,
      [name]: value
    });
  };

  // Validaciones simples
  const validar = () => {
    const errores = {};

    if (!formulario.nombre.trim()) {
      errores.nombre = "El nombre es obligatorio";
    }

    if (!formulario.email) {
      errores.email = "El email es obligatorio";
    }

    if (formulario.password.length < 6) {
      errores.password = "La contraseña debe tener al menos 6 caracteres";
    }

    return errores;
  };
```

```
// Manejar envío del formulario
const manejarEnvio = (e) => {
  e.preventDefault();
  const erroresValidacion = validar();
  setErrores(erroresValidacion);

  if (Object.keys(erroresValidacion).length === 0) {
    setEsEnviado(true);
    console.log("Formulario enviado correctamente:", formulario);
  } else {
    setEsEnviado(false);
  }
};
```

```jsx
return (
    <div>
      <h2>Formulario Completo</h2>
      <form onSubmit={manejarEnvio}>
        <div>
          <label>Nombre:</label>
          <input
            type="text"
            name="nombre"
            value={formulario.nombre}
            onChange={manejarCambio}
          />
          {errores.nombre && <p style={{color: "red"}}>{errores.nombre}</p>}
        </div>

        <div>
          <label>Email:</label>
          <input
            type="email"
            name="email"
            value={formulario.email}
            onChange={manejarCambio}
          />
          {errores.email && <p style={{color: "red"}}>{errores.email}</p>}
        </div>

        <div>
          <label>Contraseña:</label>
          <input
            type="password"
            name="password"
            value={formulario.password}
            onChange={manejarCambio}
          />
          {errores.password && <p style={{color: "red"}}>{errores.password}</p>}
        </div>

        <button type="submit" disabled={esEnviado}>
          {esEnviado ? "Enviado" : "Enviar"}
        </button>
      </form>

      {esEnviado && <p style={{color: "green"}}>Formulario enviado exitosamente</p>}
    </div>
  );
}

export default FormularioCompleto;
```

Explicación del Código:

1. **Estado del formulario**:
   - El estado del formulario se maneja como un objeto que incluye los campos nombre, email y password.
   - Cada campo del formulario está vinculado al estado mediante el atributo value.
2. **onChange en inputs**:
   - El evento onChange en cada campo se utiliza para actualizar el estado del formulario.
   - Cada vez que el usuario escribe, la función manejarCambio actualiza el valor correspondiente en el estado.
3. **Validación de formularios**:
   - La función validar se ejecuta cuando el formulario se envía y revisa los valores de los campos.
   - Si algún campo tiene un error (como el nombre vacío o una contraseña demasiado corta), los mensajes de error se almacenan en el estado errores.
4. **Mostrar errores**:
   - Si el formulario contiene errores, se muestran mensajes debajo de los campos correspondientes en rojo.
   - El estado errores gestiona estos mensajes y solo se muestran si hay un error presente.
5. **Envío del formulario**:
   - En el método manejarEnvio, primero se validan los campos.
   - Si no hay errores, se cambia el estado esEnviado a true para simular el envío exitoso.
   - El botón de envío se deshabilita cuando el formulario ha sido enviado correctamente.

6.Botón dinámico:

- El botón de envío cambia su texto a "Enviado" y se deshabilita cuando el formulario se envía con éxito para evitar múltiples envíos.

7.Mensajes de éxito:

- Si el formulario es enviado correctamente, se muestra un mensaje verde indicando el éxito.

Ventajas de Este Enfoque:

- **Validaciones en tiempo real**: Puedes implementar validaciones dinámicas mientras el usuario interactúa con el formulario.
- **Manejo de errores**: Puedes gestionar mensajes de error personalizados por cada campo y mostrarlos solo cuando son necesarios.
- **Control total**: Tienes control total sobre los valores del formulario y las interacciones, lo que permite una experiencia de usuario más robusta.
- **Estado del envío**: El formulario puede manejar el estado del envío para mostrar el progreso o deshabilitar el botón de envío en caso de éxito.

Extensiones:

•**Validación avanzada**: Puedes integrar bibliotecas de validación como Yup o Formik para manejar validaciones más complejas.

•**Estilos personalizados**: Puedes aplicar más estilos para mejorar la experiencia visual del formulario.

•**Acciones al enviar**: En un entorno real, después de un envío exitoso, podrías hacer un fetch para enviar los datos a una API o backend.

Este enfoque te da un **control completo** sobre el formulario, ideal para aplicaciones que requieren una interacción rica y validaciones exhaustivas.

Formularios No Controlados

también podemos capturar los datos directamente sin manejar el estado ni eventos como onChange. Una forma alternativa y sencilla de obtener los valores del formulario al momento del envío es utilizando **Object.fromEntries** junto con **FormData**.

Enfoque con Object.fromEntries:

•**Sin estado**: No es necesario usar useState para almacenar los valores de los inputs.
•**Captura de valores**: Utilizamos FormData para extraer todos los valores al momento del envío del formulario.
•**Uso de Object.fromEntries**: Esto convierte los valores de FormData en un objeto JavaScript de forma sencilla.
•**Simplicidad**: Ideal para formularios que no requieren seguimiento o validación constante de los campos mientras el usuario escribe.

Ventajas:

•**Menos código**: No es necesario gestionar el estado de cada campo con useState ni actualizarlo con onChange.
•**Captura eficiente**: Convierte fácilmente los datos de FormData en un objeto con Object.fromEntries.
•**Directo al envío**: Solo necesitas capturar los datos cuando el usuario envía el formulario.

```jsx
function FormularioNoControlado() {
  const manejarEnvio = (e) => {
    e.preventDefault();

    // Crear un objeto FormData a partir del formulario
    const datosFormulario = new FormData(e.target);

    // Convertir los datos de FormData en un objeto usando Object.fromEntries
    const datos = Object.fromEntries(datosFormulario.entries());

    // Mostrar los datos capturados
    console.log("Datos del formulario:", datos);
  };

  return (
    <form onSubmit={manejarEnvio}>
      <div>
        <label>Nombre:</label>
        <input type="text" name="nombre" />
      </div>

      <div>
        <label>Email:</label>
        <input type="email" name="email" />
      </div>

      <div>
        <label>Contraseña:</label>
        <input type="password" name="password" />
      </div>

      <button type="submit">Enviar</button>
    </form>
  );
}

export default FormularioNoControlado;
```

Explicación:

1.FormData:

- •Al enviar el formulario, creamos un objeto FormData a partir del formulario mismo (e.target).
- •FormData recoge todos los campos del formulario que tienen un atributo name definido, y los convierte en pares clave-valor.

2.Object.fromEntries:

- •Object.fromEntries convierte un iterable (como el que genera FormData.entries()) en un objeto.
- •FormData.entries() devuelve un iterador con los pares [nombre_campo, valor], que Object.fromEntries convierte fácilmente en un objeto JavaScript.
- •Esto nos permite acceder a los valores del formulario como un objeto plano, con cada campo del formulario representado como una clave en el objeto.

3.Captura de datos en el envío:

- •Solo cuando el formulario es enviado (onSubmit), los datos del formulario se capturan y se convierten en un objeto.
- •Se imprime en la consola el objeto datos, que contiene los valores ingresados por el usuario.

Resultado:

•**Menos gestión de estado**: No necesitas gestionar el estado de cada input ni usar useState o onChange.

•**Acceso directo a los valores**: Los datos se capturan al momento de enviar el formulario y se convierten fácilmente en un objeto JavaScript que puedes usar para enviar a un backend o manipular según necesites.

•**Simplicidad**: Este enfoque es más directo y compacto, ideal para formularios simples donde no es necesario controlar cada cambio de los inputs en tiempo real.
Este enfoque es perfecto si no necesitas realizar validaciones o manejar los valores mientras el usuario interactúa con el formulario, y solo quieres acceder a los datos al final, cuando el usuario envía el formulario.

Diferencias Entre Ambos Enfoques:

Conclusión:

•**Formularios controlados** son ideales cuando necesitas control absoluto sobre los inputs, interacciones en tiempo real o validaciones avanzadas.

•**Formularios no controlados** son más simples y útiles cuando solo necesitas capturar los datos al momento de envío, sin necesidad de manejar estados.
Ambos enfoques son válidos y se pueden usar dependiendo de los requisitos de la aplicación. Si tu objetivo es simplicidad, el enfoque de **FormData** puede ser más que suficiente. Sin embargo, si tu formulario necesita validaciones o interactividad más compleja, entonces los **formularios controlados** son el camino a seguir.

Característica	Formularios Controlados	Formularios No Controlados (FormData)
Gestión de Estado	React controla el valor con useState	Los valores no se gestionan en React
Eventos	onChange para actualizar el estado	No necesita onChange
Validación	Validaciones en tiempo real	Validaciones después del envío
Complejidad	Ideal para formularios complejos	Ideal para formularios simples
Acceso a Datos	Estado gestionado por React	Captura con FormData
Control del Usuario	Control total del input en todo momento	Solo al enviar el formulario

Estilos

Instalar Bootstrap y sus dependencias

```
npm install bootstrap
```

En App.jsx importamos

```
import 'bootstrap/dist/css/bootstrap.min.css'
import 'bootstrap/dist/js/bootstrap.bundle.min.js'
```

Ya con esto podremos usar Bootstrap en nuestro proyecto

Y para usar css muy similar simplemente importamos nuestro css que depende la arquitectura que usemos
Importaremos en App.jsx o en cada componente correspondiente

```
import './App.css'
```