📝 Guía Completa: Formularios en React 19

Autor: Isaías Fernández Lozano **Módulo**: Desarrollo Web Entorno Cliente (DWEC) **Curso**: 2025-2026

🎯 ¿Qué cambia en React 19 con los formularios?

React 19 introduce **nuevas APIs** que hacen los formularios **más simples y potentes**. Antes (React 18) necesitabas mucho código repetitivo. Ahora es más declarativo y automático.

Comparación React 18 vs React 19

✖ ANTES (React 18) - Mucho código repetitivo:

```
function FormularioAntiguo() {
  // 1. Estado para cada campo
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');

  // 2. Estado para loading
  const [loading, setLoading] = useState(false);

  // 3. Estado para errores
  const [error, setError] = useState('');

  // 4. Handler del submit
  const handleSubmit = async (e) => {
    e.preventDefault(); // ← Obligatorio
    setLoading(true);
    setError('');

    try {
      await api.crear({ name, email });
      setName(''); // ← Resetear manual
      setEmail('');
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        value={name}
        onChange={(e) => setName(e.target.value)}
```

```
        />
        <button disabled={loading}>
          {loading ? 'Enviando...' : 'Enviar'}
        </button>
      </form>
    );
}
```

Problemas:

- 😩 Mucho estado manual (`useState` para cada cosa)
- 😩 `onChange` en cada input
- 😩 `e.preventDefault()` obligatorio
- 😩 Loading y errores manuales
- 😩 Resetear el form manualmente

✓ AHORA (React 19) - Mucho más simple:

```
function FormularioModerno() {
  // 1. UNA SOLA función para manejar todo
  async function formAction(prevState, formData) {
    const name = formData.get('name');
    const email = formData.get('email');

    await api.crear({ name, email });
    return { success: true, message: 'Creado!' };
  }

  // 2. UN SOLO hook para estado + loading + errores
  const [state, submitAction, isPending] = useActionState(formAction, {
    success: false,
    message: ''
  });

  return (
    <form action={submitAction}> {/* ← Sin onSubmit, sin preventDefault */}
      <input name="name" defaultValue="" /> {/* ← Sin onChange */}
      <SubmitButton isPending={isPending} />
      {state.message && <p>{state.message}</p>}
    </form>
  );
}

function SubmitButton({ isPending }) {
  const { pending } = useFormStatus(); // ← Hook para saber si está
enviando
  return (
    <button disabled={pending}>
```

```
        {pending ? 'Enviando...' : 'Enviar'}
      </button>
    );
  }
```

Ventajas:

- ✓ Menos código
- ✓ Sin `onChange` en cada input
- ✓ Sin `e.preventDefault()`
- ✓ Loading automático con `useFormStatus`
- ✓ Estado del form en un solo lugar

🔧 Los 2 Nuevos Hooks de React 19

1️⃣ `useActionState` - El hook principal

¿Qué hace? Maneja todo el estado del formulario: datos, loading, errores, mensajes.

Sintaxis:

```
const [state, submitAction, isPending] = useActionState(
  formAction,      // Función que se ejecuta al enviar
  initialState     // Estado inicial
);
```

Parámetros:

Parámetro	Tipo	Descripción
`formAction`	`(prevState, formData) => Promise<newState>`	Función async que procesa el form
`initialState`	`Object`	Estado inicial (ej: `{ message: '', success: false }`)

Retorna:

Valor	Tipo	Descripción
`state`	`Object`	Estado actual del formulario
`submitAction`	`Function`	Función para pasar a `action=` del form
`isPending`	`boolean`	`true` si está enviando, `false` si no

Ejemplo completo:

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

function MiFormulario() {
  // 1. Definir la acción (qué pasa cuando se envía)
  async function crearUsuario(prevState, formData) {
    // formData.get('nombre_del_input') obtiene el valor
    const nombre = formData.get('nombre');
    const email = formData.get('email');

    // Validaciones
    if (!nombre) {
      return { success: false, message: 'Nombre es requerido' };
    }

    // Llamar API
    try {
      await api.crear({ nombre, email });
      return { success: true, message: 'Usuario creado!' };
    } catch (error) {
      return { success: false, message: error.message };
    }
  }

  // 2. Usar useActionState
  const [state, submitAction, isPending] = useActionState(crearUsuario, {
    success: false,
    message: ''
  });

  // 3. Usar en el JSX
  return (
    <form action={submitAction}>
      <input name="nombre" /> {/* ← El name es importante */}
      <input name="email" />

      <button disabled={isPending}>
        {isPending ? 'Enviando...' : 'Crear'}
      </button>

      {state.message && <p>{state.message}</p>}
    </form>
  );
}
```

Cosas importantes:

1. `formData.get('nombre')` - Obtiene el valor del input con `name="nombre"`

2. **La función debe retornar el nuevo estado** - No uses `setState`, solo retorna un objeto
3. **`action={submitAction}`** - NO uses `onSubmit`, usa `action=`
4. **Los inputs DEBEN tener `name`** - Así formData los encuentra

---

2️⃣ `useFormStatus` - Para saber si está enviando

¿Qué hace? Te dice si el formulario está enviándose (pending) o no.

IMPORTANTE: Solo funciona **dentro de un componente hijo** del `<form>`.

Sintaxis:

```
import { useFormStatus } from 'react-dom'; // ← Fíjate: react-dom, no react

function SubmitButton() {
  const { pending } = useFormStatus();

  return (
    <button disabled={pending}>
      {pending ? 'Enviando...' : 'Enviar'}
    </button>
  );
}

function MiForm() {
  return (
    <form action={submitAction}>
      <input name="nombre" />
      <SubmitButton /> {/* ← Componente hijo del form */}
    </form>
  );
}
```

¿Por qué en un componente separado? Porque React necesita saber qué formulario está monitoreando. Si lo pones en el mismo componente que el `<form>`, no funciona.

Propiedades de useFormStatus:

```
const status = useFormStatus();
// status = {
//   pending: true/false,     // ¿Está enviando?
//   data: FormData,          // Datos del form
//   method: 'POST',          // Método HTTP
//   action: '/api/crear'     // URL de la acción
// }
```

Normalmente solo usas `pending`:

```
const { pending } = useFormStatus();
```

📋 Ejemplo Completo Paso a Paso

Voy a crear un formulario completo desde cero, explicando cada línea.

Paso 1: Importar los hooks

```
import { useActionState } from 'react';           // ← Para manejar el form
import { useFormStatus } from 'react-dom';        // ← Para el botón de
submit
import { useCompanies } from '../../hooks/useCompanies'; // ← Nuestro hook
custom
```

Paso 2: Crear el componente del botón

```
// IMPORTANTE: Componente separado para usar useFormStatus
function SubmitButton({ isEditing }: { isEditing: boolean }) {
  const { pending } = useFormStatus(); // ← Sabe si el form está enviando

  return (
    <button
      type="submit"
      disabled={pending}  // ← Se deshabilita automáticamente
      className="btn btn-primary"
    >
      {/* Cambia el texto según el estado */}
      {pending ? 'Guardando...' : isEditing ? 'Actualizar' : 'Crear'}
    </button>
  );
}
```

Explicación:

- `pending` es `true` cuando el form se está enviando
- El botón se deshabilita automáticamente
- El texto cambia automáticamente

Paso 3: Definir el tipo del estado

```
// Define qué estructura tendrá el estado del formulario
type FormState = {
  success: boolean;        // ¿Fue exitoso?
  message: string;         // Mensaje de éxito/error
  errors?: Record<string, string>; // Errores por campo (opcional)
};
```

Paso 4: Crear el componente principal

```
export default function CompanyForm({ companyToEdit, onEditComplete }) {
  // 1. Obtener funciones del contexto
  const { createCompany, updateCompany } = useCompanies();

  // 2. Saber si estamos editando o creando
  const isEditing = !!companyToEdit; // true si hay empresa a editar

  // 3. Definir la acción del formulario
  async function formAction(prevState, formData) {
    // 3.1 Obtener valores del formulario
    const name = formData.get('name') as string;
    const industry = formData.get('industry') as string;
    const website = formData.get('website') as string;

    // 3.2 Validar
    if (!name.trim()) {
      return {
        success: false,
        message: 'El nombre es requerido',
        errors: { name: 'Este campo es obligatorio' }
      };
    }

    // 3.3 Preparar datos
    const data = {
      name: name.trim(),
      industry: industry.trim() || undefined,
      website: website.trim() || undefined
    };

    // 3.4 Crear o actualizar
    let result;
    if (isEditing) {
      result = await updateCompany(companyToEdit.id, data);
    } else {
      result = await createCompany(data);
    }
```

```
    // 3.5 Retornar nuevo estado
    if (result) {
      if (isEditing) {
        onEditComplete?.(); // Cerrar modo edición
      }
      return {
        success: true,
        message: isEditing ? 'Empresa actualizada' : 'Empresa creada'
      };
    }

    return {
      success: false,
      message: 'Error al guardar'
    };
  }

  // 4. Usar useActionState
  const [state, submitAction, isPending] = useActionState(formAction, {
    success: false,
    message: ''
  });

  // 5. Renderizar el formulario
  return (
    <form
      action={submitAction}  // ← Aquí va submitAction
      key={companyToEdit?.id || 'new'} // ← Resetea form al cambiar empresa
    >
      {/* Inputs */}
      <input
        name="name"  // ← IMPORTANTE: name para formData.get()
        defaultValue={companyToEdit?.name || ''}  // ← defaultValue, no
value
        required
      />

      {/* Botón (componente separado) */}
      <SubmitButton isEditing={isEditing} />

      {/* Mostrar mensajes */}
      {state.message && !isPending && (
        <p className={state.success ? 'text-green-600' : 'text-red-600'}>
          {state.message}
        </p>
      )}
    </form>
  );
}
```

🔑 Conceptos Clave

1. `formData.get('nombre')`

FormData es un objeto especial de JavaScript que contiene los valores del formulario.

```
// Si tienes esto:
<input name="email" />
<input name="password" />

// En formAction puedes hacer:
const email = formData.get('email');
const password = formData.get('password');
```

Importante:

- Los inputs DEBEN tener atributo `name`
- `formData.get()` siempre retorna `string | null`
- Por eso hacemos `as string` para TypeScript

2. `defaultValue` vs `value`

En React 19 forms, usa **`defaultValue`** en lugar de `value`:

```
// ✖ MAL (React 18):
<input
  value={name}
  onChange={(e) => setName(e.target.value)}
/>

// ✓ BIEN (React 19):
<input
  name="name"
  defaultValue={name}
  // Sin onChange, React lo maneja automáticamente
/>
```

¿Por qué `defaultValue`?

- React 19 maneja el estado internamente
- `defaultValue` es el valor inicial, luego React lo gestiona
- No necesitas `onChange` ni `useState`

3. `action` vs `onSubmit`

```
// ✗ MAL (React 18):
<form onSubmit={handleSubmit}>

// ✓ BIEN (React 19):
<form action={submitAction}>
```

Diferencias:

- `onSubmit`: Necesitas `e.preventDefault()` manualmente
- `action`: React lo maneja automáticamente

4. El truco del `key` para resetear

```
<form
  action={submitAction}
  key={companyToEdit?.id || 'new'} // ← Cambia cuando cambia la empresa
>
```

¿Qué hace?

- Cuando cambia el `key`, React **desmonta y remonta** el componente
- Esto resetea todos los inputs automáticamente
- No necesitas `.reset()` ni `useRef`

Ejemplo:

1. Estás creando (key = 'new')
2. Le das editar a Google (key = 1)
3. React remonta el form con los datos de Google
4. Cancelas (key = 'new')
5. React remonta el form vacío

VS Comparación Completa

Característica	React 18	React 19
Hook principal	`useState`	`useActionState`
Loading state	Manual (`useState`)	Automático (`useFormStatus`)
Submit handler	`onSubmit` + `e.preventDefault()`	`action={submitAction}`
Inputs	`value` + `onChange`	`defaultValue` (sin onChange)
Resetear form	Manual (`.reset()` o `setState`)	Automático (con `key`)

Característica	React 18	React 19
Errores	Manual (`useState`)	Incluido en el estado
Código	~50 líneas	~30 líneas
Complejidad	Alta	Media

💡 Consejos y Trucos

1. Validaciones

```
async function formAction(prevState, formData) {
  const name = formData.get('name') as string;

  // Validar
  if (!name || name.trim().length < 3) {
    return {
      success: false,
      message: 'El nombre debe tener al menos 3 caracteres',
      errors: { name: 'Mínimo 3 caracteres' }
    };
  }

  // Continuar...
}
```

2. Mostrar errores por campo

```
<input name="name" defaultValue="" />
{state.errors?.name && (
  <p className="text-red-500 text-sm">
    {state.errors.name}
  </p>
)}
```

3. No mostrar mensaje mientras está enviando

```
{state.message && !isPending && (
  <p>{state.message}</p>
)}
```

¿Por qué? Porque cuando estás enviando no quieres que se vea el mensaje anterior.

4. Deshabilitar el form mientras envía

```
<input
  name="name"
  disabled={isPending} // ← Deshabilita mientras envía
/>
```

🚨 Errores Comunes

Error 1: useFormStatus no funciona

```
// ✖ MAL
function MiForm() {
  const { pending } = useFormStatus(); // No funciona aquí

  return (
    <form action={submitAction}>
      <button disabled={pending}>Enviar</button>
    </form>
  );
}

// ✓ BIEN
function SubmitButton() {
  const { pending } = useFormStatus(); // ← En componente hijo
  return <button disabled={pending}>Enviar</button>;
}

function MiForm() {
  return (
    <form action={submitAction}>
      <SubmitButton />
    </form>
  );
}
```

Error 2: Olvidar el `name` en inputs

```
// ✖ MAL
<input defaultValue="" />

// ✓ BIEN
<input name="nombre" defaultValue="" />
```

Sin `name`, formData.get() retorna `null`.

Error 3: Usar `value` en lugar de `defaultValue`

```
// ✖ MAL
<input value={name} onChange={...} />

// ✓ BIEN
<input name="name" defaultValue={name} />
```

Error 4: No retornar el nuevo estado

```
// ✖ MAL
async function formAction(prevState, formData) {
  await api.crear(data);
  // ¡Falta el return!
}

// ✓ BIEN
async function formAction(prevState, formData) {
  await api.crear(data);
  return { success: true, message: 'Creado!' };
}
```

📚 Recursos

- React 19 Docs - useActionState
- React 19 Docs - useFormStatus
- React 19 Blog - What's New

🎓 Ejercicios Propuestos

Ejercicio 1: Formulario de Login

Crea un formulario de login usando `useActionState` y `useFormStatus`:

- Email (obligatorio)
- Password (obligatorio, mínimo 6 caracteres)
- Botón con estado "Iniciando sesión..."
- Mostrar errores de validación

Ejercicio 2: Formulario con múltiples pasos

Crea un formulario con 2 pasos:

- Paso 1: Nombre y email
- Paso 2: Teléfono y dirección
- Usa `useActionState` para mantener el estado entre pasos

Ejercicio 3: Formulario con archivos

Crea un formulario que suba un archivo:

- Usa `formData.get('archivo')` para obtener el File
- Muestra preview de la imagen antes de subir
- Barra de progreso con `useFormStatus`

¡Ahora ya sabes usar formularios en React 19! 🚀