Guía Completa: useState y Formularios en React

📋 Índice

1. ¿Qué es useState?
2. Implementación paso a paso
3. Flujo de datos completo
4. Código final
5. Conceptos clave
6. Errores comunes y soluciones
7. Próximos pasos

¿Qué es useState?

🎯 Definición

`useState` es un **Hook de React** que permite a los componentes funcionales manejar estado local.

🔍 ¿Qué es el estado?

- **Estado** = datos que pueden cambiar durante la vida del componente
- **Ejemplo:** El texto que escribes en un input
- **Sin estado:** Los inputs no "recuerdan" lo que escribes
- **Con estado:** React sabe qué hay en cada campo y puede reaccionar

📦 Sintaxis básica

```
const [valor, setValor] = useState(valorInicial);
```

Componentes:

- `valor` = el valor actual del estado
- `setValor` = función para cambiar el valor
- `valorInicial` = valor con el que empieza

Implementación paso a paso

Paso 1: Importar useState

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

¿Por qué?

- `useState` no está disponible por defecto
- Necesitas importarlo de React para usarlo

Paso 2: Declarar estados para cada campo

```
const [formDataName, setNombre] = useState('') //estado para el nombre
const [email, setEmail] = useState('') //estado para el email
const [mensaje, setMensaje] = useState('') //estado para el mensaje
```

Explicación línea por línea:

- `formDataName` = variable que guarda el valor del campo nombre
- `setNombre` = función para cambiar el valor de `formDataName`
- `useState('')` = inicializa con string vacío
- **Importante:** Evitamos usar `name` para no conflictuar con `e.target.name`

Paso 3: Crear función handleInputChange

```
const handleInputChange = (e) => {
    const { name, value } = e.target;

    switch (name) {
        case 'nombre':
            setNombre(value);
            break;
        case 'email':
            setEmail(value);
            break;
        case 'mensaje':
            setMensaje(value);
            break;
        default:
            break;
    }
};
```

Explicación detallada:

Línea 1: `const handleInputChange = (e) => {`

- `handleInputChange` = nombre de la función (puede ser cualquier nombre)
- `(e)` = parámetro que recibe el evento
- `=>` = arrow function (función flecha)

Línea 2: `const { name, value } = e.target;`

- **Desestructuración de objetos** en JavaScript

- `e.target` = el elemento HTML que disparó el evento (el input)
- `name` = valor de la prop `name` del input (ej: 'nombre', 'email', 'mensaje')
- `value` = texto que escribió el usuario
- **¿Por qué `{ }`?** Para extraer propiedades de objetos

Líneas 4-16: Switch statement

```
switch (name) {
    case 'nombre':
        setNombre(value);
        break;
    case 'email':
        setEmail(value);
        break;
    case 'mensaje':
        setMensaje(value);
        break;
    default:
        break;
}
```

¿Cómo funciona?

- Compara `name` con cada caso
- Si `name = 'nombre'` → ejecuta `setNombre(value)`
- Si `name = 'email'` → ejecuta `setEmail(value)`
- Si `name = 'mensaje'` → ejecuta `setMensaje(value)`

Paso 4: Crear función handleSubmit

```
const handleSubmit = (e) => {
    e.preventDefault();
    const formData = {
        nombre: formDataName,
        email: email,
        mensaje: mensaje
    };
    console.log('Formulario enviado:', formData);
    setNombre('');
    setEmail('');
    setMensaje('');
};
```

Explicación línea por línea:

Línea 1: `const handleSubmit = (e) => {`

- Función que maneja el envío del formulario

- Se ejecuta cuando el usuario hace clic en "Enviar"

Línea 2: `e.preventDefault();`

- **CRUCIAL:** Evita que el formulario recargue la página
- Sin esto, la página se recargaría y perderías los datos

Líneas 3-7: Crear objeto con datos

```
const formData = {
    nombre: formDataName,
    email: email,
    mensaje: mensaje
};
```

- Recopila todos los datos del estado en un objeto
- Más fácil de manejar y enviar

Línea 8: `console.log('Formulario enviado:', formData);`

- Muestra los datos en la consola para debugging
- Más adelante aquí irá la lógica de envío real

Líneas 9-11: Limpiar formulario

```
setNombre('');
setEmail('');
setMensaje('');
```

- Vuelve todos los campos a vacío
- El formulario se limpia automáticamente

Paso 5: Conectar funciones a los elementos HTML

En el formulario:

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

En cada input:

```
<input
    type="text"
    id="name"
```

```
    name="nombre"
    value={formDataName}        // ← Muestra el valor del estado
    onChange={handleInputChange} // ← Llama a la función cuando cambia
    placeholder={t('contact.form.namePlaceholder')}
    required
/>
```

Explicación de las props:

- `value={formDataName}` = muestra el valor actual del estado
- `onChange={handleInputChange}` = ejecuta la función cuando el usuario escribe
- `name="nombre"` = identificador que usa el switch para saber qué campo es

---

Flujo de datos completo

🔄 Flujo cuando el usuario escribe:

```
Usuario escribe "Eva" en el campo nombre
    ↓
onChange se dispara
    ↓
handleInputChange(e) se ejecuta
    ↓
Extrae: name='nombre', value='Eva'
    ↓
Switch: case 'nombre' → setNombre('Eva')
    ↓
Estado formDataName cambia a 'Eva'
    ↓
React re-renderiza el componente
    ↓
Input muestra "Eva"
```

🔄 Flujo cuando el usuario envía:

```
Usuario hace clic en "Enviar mensaje"
    ↓
onSubmit se dispara
    ↓
handleSubmit(e) se ejecuta
    ↓
e.preventDefault() evita recarga
    ↓
Crea objeto formData con todos los valores
    ↓
console.log muestra los datos
    ↓
setNombre(''), setEmail(''), setMensaje('')
```

```
    ↓
React re-renderiza
    ↓
Formulario se limpia
```

Código final completo

```
import './Contact.css'
import { useTranslation } from 'react-i18next';
import { useState } from 'react';

function Contact() {
    const { t } = useTranslation();

    // Estados para cada campo del formulario
    const [formDataName, setNombre] = useState('') //estado para el nombre
    const [email, setEmail] = useState('') //estado para el email
    const [mensaje, setMensaje] = useState('') //estado para el mensaje

    // Función que maneja los cambios en los inputs
    const handleInputChange = (e) => {
        const { name, value } = e.target;

        switch (name) {
            case 'nombre':
                setNombre(value);
                break;
            case 'email':
                setEmail(value);
                break;
            case 'mensaje':
                setMensaje(value);
                break;
            default:
                break;
        }
    };

    // Función que maneja el envío del formulario
    const handleSubmit = (e) => {
        e.preventDefault();
        const formData = {
            nombre: formDataName,
            email: email,
            mensaje: mensaje
        };
        console.log('Formulario enviado:', formData);
        setNombre('');
        setEmail('');
        setMensaje('');
```

```
    };

    return (
        <section className="contact" id="contacto">
            <h2 className="section-title">{t('contact.title')}</h2>
            <div className="contact-content">
                {/* Información de contacto */}
                <div className="contact-info">
                    <h3>{t('contact.info.title')}</h3>
                    <p>{t('contact.info.description')}</p>
                    <div className="contact-links">
                        <a href="mailto:evablancomart@gmail.com">
                            📧 evablancomart@gmail.com
                        </a>
                        <a href="https://www.linkedin.com/in/eva-blanco-
mart%C3%ADnez-5a8617158/"
                           target="_blank"
                           rel="noopener noreferrer">
                            💼 LinkedIn
                        </a>
                        <a href="https://github.com/Evablan"
                           target="_blank"
                           rel="noopener noreferrer">
                            🐙 GitHub
                        </a>
                    </div>
                </div>

                {/* Formulario de contacto */}
                <div className="contact-form">
                    <h3>{t('contact.form.title')}</h3>
                    <form onSubmit={handleSubmit}>
                        <div className="form-group">
                            <label htmlFor="name">{t('contact.form.name')}</label>
                            <input
                                type="text"
                                id="name"
                                name="nombre"
                                value={formDataName}
                                onChange={handleInputChange}
                                placeholder={t('contact.form.namePlaceholder')}
                                required
                            />
                        </div>
                        <div className="form-group">
                            <label htmlFor="email">{t('contact.form.email')}
</label>
                            <input
                                type="email"
                                id="email"
                                name="email"
                                value={email}
                                onChange={handleInputChange}
                                placeholder={t('contact.form.emailPlaceholder')}
```

```
                                required
                            />
                        </div>
                        <div className="form-group">
                            <label htmlFor="message">{t('contact.form.message')}
</label>
                            <textarea
                                id="message"
                                name="mensaje"
                                value={mensaje}
                                onChange={handleInputChange}
                                placeholder={t('contact.form.messagePlaceholder')}
                                required
                            ></textarea>
                        </div>
                        <button type="submit" className="btn-primary">
                            {t('contact.form.submit')}
                        </button>
                    </form>
                </div>
            </div>
        </section>
    )
}

export default Contact;
```

Conceptos clave

🎯 Componente Controlado

Un input está **controlado** cuando:

- Su valor viene del estado de React (`value={estado}`)
- React maneja todos los cambios (`onChange={funcion}`)
- React "controla" completamente el input

🔄 Re-renderizado

- Cuando el estado cambia, React re-renderiza el componente
- Los inputs muestran automáticamente el nuevo valor
- No necesitas manipular el DOM manualmente

📦 Desestructuración

```
const { name, value } = e.target;
```

- Extrae propiedades específicas de un objeto

- Más corto que: `const name = e.target.name; const value = e.target.value;`

🎪 Eventos en React

- `onChange` = cuando cambia el valor de un input
- `onSubmit` = cuando se envía un formulario
- Los eventos reciben un objeto con información del evento

Errores comunes y soluciones

❌ Error: Conflicto de nombres de variables

```
// ❌ INCORRECTO
const [name, setNombre] = useState('')
const { name, value } = e.target; // Conflicto: dos variables 'name'
```

```
// ☑ CORRECTO
const [formDataName, setNombre] = useState('')
const { name, value } = e.target; // No hay conflicto
```

❌ Error: Switch fuera de la función

```
// ❌ INCORRECTO
const handleInputChange = (e) => {
    const { name, value } = e.target;
}
switch (name) { // ❌ Fuera de la función
    // casos...
}
```

```
// ☑ CORRECTO
const handleInputChange = (e) => {
    const { name, value } = e.target;

    switch (name) { // ☑ Dentro de la función
        // casos...
    }
}
```

❌ Error: Falta preventDefault()

```
// ❌ INCORRECTO
const handleSubmit = (e) => {
    // Sin preventDefault - la página se recargará
    console.log('Datos:', formData);
}
```

```
// ☑ CORRECTO
const handleSubmit = (e) => {
    e.preventDefault(); // ☑ Evita la recarga
    console.log('Datos:', formData);
}
```

❌ Error: Faltan value y onChange en inputs

```
// ❌ INCORRECTO
<input name="nombre" /> // Sin value ni onChange
```

```
// ☑ CORRECTO
<input
    name="nombre"
    value={formDataName}        // ☑ Muestra el valor
    onChange={handleInputChange} // ☑ Maneja cambios
/>
```

Próximos pasos

🚀 Funcionalidades avanzadas que puedes añadir:

1. Validación de formularios

```
const [errors, setErrors] = useState({});

const validateForm = () => {
    const newErrors = {};

    if (!formDataName.trim()) {
        newErrors.nombre = 'El nombre es requerido';
    }

    if (!email.includes('@')) {
        newErrors.email = 'Email inválido';
    }
```

```
    setErrors(newErrors);
    return Object.keys(newErrors).length === 0;
};
```

2. Envío real del formulario

```
const handleSubmit = async (e) => {
    e.preventDefault();

    try {
        const response = await fetch('/api/contact', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(formData)
        });

        if (response.ok) {
            alert('Mensaje enviado con éxito!');
        }
    } catch (error) {
        console.error('Error:', error);
    }
};
```

3. Estados de carga

```
const [isSubmitting, setIsSubmitting] = useState(false);

const handleSubmit = async (e) => {
    e.preventDefault();
    setIsSubmitting(true);

    try {
        // Lógica de envío
    } finally {
        setIsSubmitting(false);
    }
};
```

4. Feedback visual

```
const [submitStatus, setSubmitStatus] = useState('idle'); // 'idle' | 'success' |
'error'
```

```
// En el JSX:
{submitStatus === 'success' && <div className="success">¡Mensaje enviado!</div>}
{submitStatus === 'error' && <div className="error">Error al enviar</div>}
```

🎓 Resumen de lo aprendido

☑ Conceptos dominados:

- **useState** para manejar estado local
- **Eventos** en React (onChange, onSubmit)
- **Formularios controlados**
- **Desestructuración** de objetos
- **Manejo de formularios** completo

☑ Funcionalidades implementadas:

- Captura de datos en tiempo real
- Validación básica con required
- Envío de formulario
- Limpieza automática de campos
- Debugging con console.log

🚀 Habilidades desarrolladas:

- Pensamiento en flujo de datos
- Debugging de errores comunes
- Estructuración de código React
- Manejo de eventos complejos

Esta guía cubre todos los conceptos fundamentales de useState y formularios en React. ¡Ahora tienes una base sólida para construir formularios más complejos!