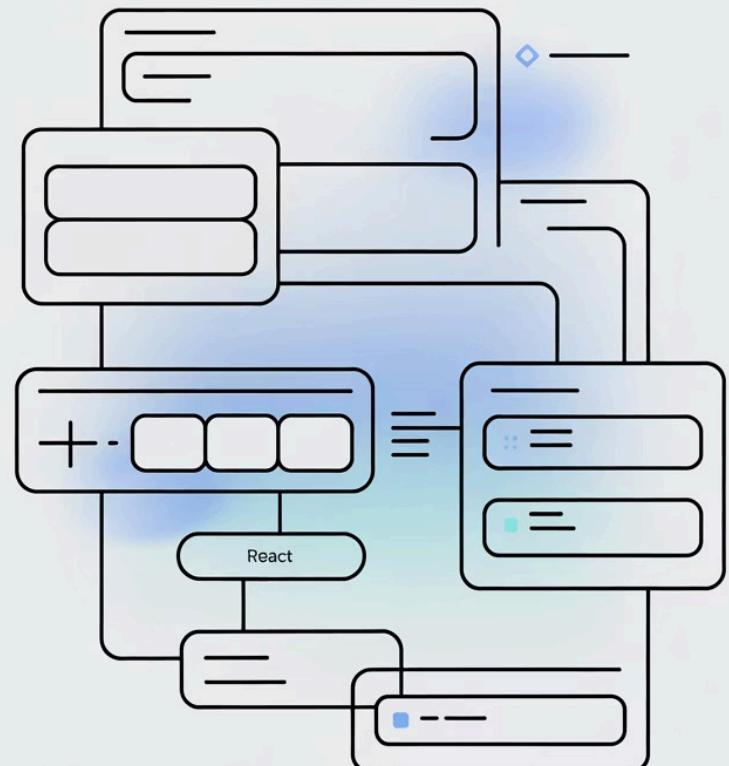


# Introducción al Desarrollo web

## Curso: Desarrollo Frontend con React

# Eventos y Formularios en React



# ¿Qué Aprenderemos Hoy?

1

## Manejo de Eventos

onClick, onChange, onSubmit y más

2

## Inputs Controlados

Control total del estado de formularios

3

## Validaciones

Técnicas de validación en tiempo real

4

## Buenas Prácticas

Accesibilidad y experiencia de usuario

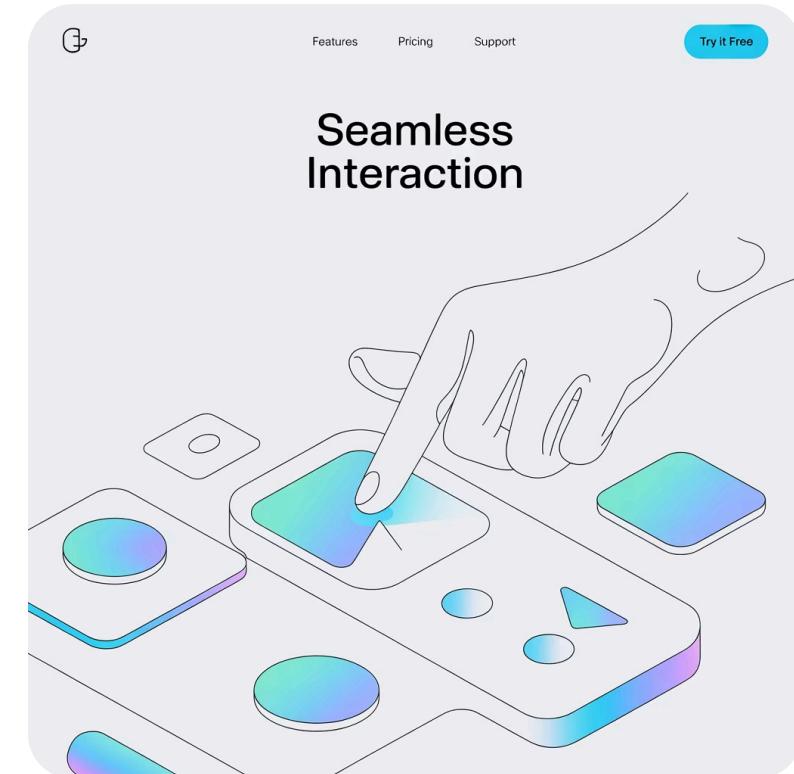
# Eventos en React

# ¿Qué son los Eventos en React?

Los eventos en React son acciones que ocurren como resultado de la interacción del usuario o del sistema.

React utiliza **SyntheticEvents**, una capa de abstracción que unifica el comportamiento de eventos entre diferentes navegadores.

Estos eventos permiten que nuestras aplicaciones respondan dinámicamente a las acciones del usuario.



# Eventos Más Comunes

## **onClick**

Se dispara cuando el usuario hace clic en un elemento

```
onClick={handleClick}
```

## **onChange**

Se activa cuando cambia el valor de un input

```
onChange={handleChange}
```

## **onSubmit**

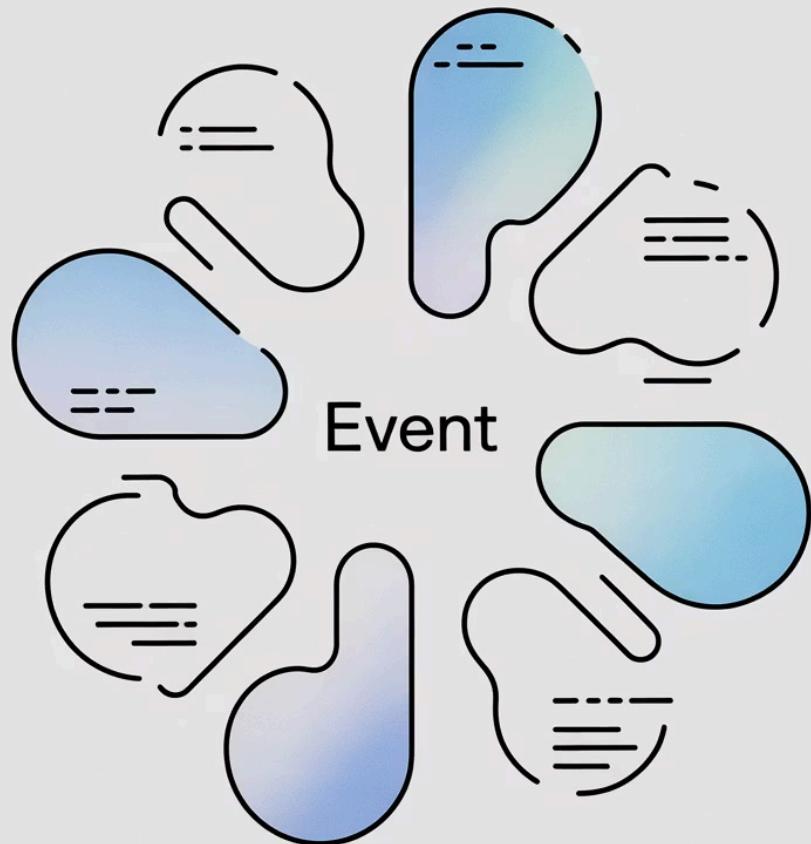
Se ejecuta al enviar un formulario

```
onSubmit={handleSubmit}
```

# Ejemplo: Manejador de Eventos

```
function Button() {  
  const handleClick = (event) => {  
    event.preventDefault();  
    console.log('¡Botón clickeado!');  
  };  
  
  return (  
    <button onClick={handleClick}>  
      Hacer clic  
    </button>  
  );  
}
```

El parámetro `event` contiene información sobre el evento que ocurrió.



# El Objeto Event

## **event.target**

Referencia al elemento que disparó el evento

## **event.preventDefault()**

Previene el comportamiento por defecto

## **event.stopPropagation()**

Detiene la propagación del evento

# Formularios en React

# Inputs Controlados vs No Controlados

## Controlados

- React controla el valor del input
- Estado sincronizado con el valor
- Mayor control y predictibilidad
- Validación en tiempo real

## No Controlados

- DOM controla el valor del input
- Se accede con refs
- Menos código para casos simples
- Similar a formularios HTML tradicionales

# Input Controlado: Ejemplo

```
function FormularioControlado() {  
  const [nombre, setNombre] = useState("");  
  
  const handleChange = (event) => {  
    setNombre(event.target.value);  
  };  
  
  return (  
    <input  
      type="text"  
      value={nombre}  
      onChange={handleChange}  
      placeholder="Ingresa tu nombre"  
    />  
  );  
}
```

El valor del input siempre refleja el estado del componente.

# ¿Por qué Usar Inputs Controlados?



## Validación Inmediata

Puedes validar y mostrar errores mientras el usuario escribe



## Estado Sincronizado

El estado siempre está actualizado con el valor del input



## Formateo Dinámico

Puedes formatear el texto mientras el usuario escribe

# Múltiples Inputs: Una Técnica

```
function Formulario() {  
  const [formData, setFormData] = useState({  
    nombre: "",  
    email: "",  
    edad: ""  
  });  
  
  const handleChange = (event) => {  
    const { name, value } = event.target;  
    setFormData(prev => ({  
      ...prev,  
      [name]: value  
    }));  
  };  
}
```

Usa un objeto para manejar múltiples campos eficientemente.

# El Formulario Completo

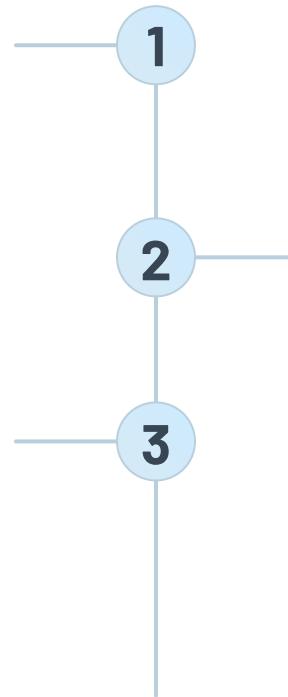
```
<form onSubmit={handleSubmit}>  
  <input  
    name="nombre"  
    value={formData.nombre}  
    onChange={handleChange}  
    placeholder="Nombre"  
  />  
  <input  
    name="email"  
    value={formData.email}  
    onChange={handleChange}  
    placeholder="Email"  
  />  
  <button type="submit">Enviar</button>  
</form>
```

# Validaciones

# Tipos de Validación

## Validación en Tiempo Real

Se ejecuta mientras el usuario escribe (onChange)



## Validación al Enviar

Se realiza antes de procesar el formulario  
(onSubmit)

## Validación al Perder Foco

Se activa cuando el usuario sale del campo  
(onBlur)

# Ejemplo: Validación de Email

```
const validateEmail = (email) => {
  const emailRegex = /^[^@\s]+@[^\s@]+\.\[^@\s]+$/;
  return emailRegex.test(email);
};

const handleEmailChange = (event) => {
  const email = event.target.value;
  setFormData(prev => ({ ...prev, email }));

  if (email && !validateEmail(email)) {
    setErrors(prev => ({
      ...prev,
      email: 'Email inválido'
    }));
  } else {
    setErrors(prev => ({ ...prev, email: "" }));
  }
};
```

# Mostrando Errores de Validación

```
<input  
  type="email"  
  name="email"  
  value={formData.email}  
  onChange={handleEmailChange}  
  className={errors.email ? 'error' : ''}  
/>  
  
{errors.email && (  
  <span className="error-message">  
    {errors.email}  
  </span>  
)}
```



# Validaciones Comunes

## Campo Requerido

```
if (!value.trim()) return 'Campo  
requerido';
```

## Longitud Mínima

```
if (value.length < 8) return  
'Mínimo 8 caracteres';
```

## Formato de Teléfono

```
if (/^\d{10}$.test(value)) return  
'Teléfono inválido';
```

# Ejemplo Práctico

# Estructura del Componente

```
function FormularioRegistro() {  
  const [formData, setFormData] = useState({  
    nombre: "",  
    email: "",  
    password: "",  
    confirmPassword: ""  
  });  
  
  const [errors, setErrors] = useState({});  
  const [isSubmitting, setIsSubmitting] = useState(false);  
  
  // Funciones de manejo...  
}
```

# Función de Validación Completa

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

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

  if (!validateEmail(formData.email)) {
    newErrors.email = 'Email inválido';
  }

  if (formData.password.length < 8) {
    newErrors.password = 'Mínimo 8 caracteres';
  }

  if (formData.password !== formData.confirmPassword) {
    newErrors.confirmPassword = 'Las contraseñas no coinciden';
  }

  return newErrors;
};
```

# Manejador de Envío

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

  const formErrors = validateForm();
  setErrors(formErrors);

  if (Object.keys(formErrors).length === 0) {
    setIsSubmitting(true);
    try {
      await registrarUsuario(formData);
      alert('¡Registro exitoso!');
    } catch (error) {
      alert('Error en el registro');
    } finally {
      setIsSubmitting(false);
    }
  }
};
```

# El JSX del Formulario

```
<form onSubmit={handleSubmit}>
  <div>
    <input
      type="text"
      name="nombre"
      value={formData.nombre}
      onChange={handleInputChange}
      placeholder="Nombre completo"
    />
    {errors.nombre && <span>{errors.nombre}</span>}
  </div>

  <button type="submit" disabled={isSubmitting}>
    {isSubmitting ? 'Registrando...' : 'Registrarse'}
  </button>
</form>
```

# Buenas Prácticas

# Principios de Usabilidad



## Feedback Visual

Proporciona indicadores claros de estado: éxito, error, carga. Los usuarios necesitan saber qué está pasando.



## Validación Inmediata

Valida campos mientras el usuario interactúa, no solo al enviar. Esto reduce frustración y errores.



## Mensajes Claros

Escribe mensajes de error específicos y accionables. "Campo requerido" es mejor que "Error".

# Accesibilidad en Formularios

## Labels Descriptivos

```
<label htmlFor="email">  
  Correo electrónico  
</label>  
<input id="email" type="email" />
```

## ARIA Attributes

```
<input  
  aria-describedby="error-email"  
  aria-invalid={!errors.email}>  
</input>
```

## Navegación por Teclado

```
tabIndex="0"  
onKeyDown={handleKeyDown}
```

# Patrones Avanzados



## Custom Hooks

Crea hooks reutilizables para lógica de formularios

## Esquemas de Validación

Usa librerías como Yup o Zod para validaciones complejas

## Estado Global

Considera Context API para formularios multi-paso

# Hook Personalizado: useForm

```
function useForm(initialValues, validationRules) {  
  const [values, setValues] = useState(initialValues);  
  const [errors, setErrors] = useState({});  
  
  const handleChange = (event) => {  
    const { name, value } = event.target;  
    setValues(prev => ({ ...prev, [name]: value }));  
  
    // Validar campo individual  
    if (validationRules[name]) {  
      const error = validationRules[name](value);  
      setErrors(prev => ({ ...prev, [name]: error }));  
    }  
  };  
  
  return { values, errors, handleChange };  
}
```

# Uso del Hook Personalizado

```
function MiFormulario() {
  const validationRules = {
    email: (value) => {
      if (!validateEmail(value)) return 'Email inválido';
      return '';
    },
    password: (value) => {
      if (value.length < 8) return 'Mínimo 8 caracteres';
      return '';
    }
  };

  const { values, errors, handleChange } = useForm(
    { email: "", password: "" },
    validationRules
  );

  return (
    // JSX del formulario...
  );
}
```

# Bibliotecas Populares

1

## React Hook Form

Rendimiento óptimo con validación mínima de re-renders

2

## Formik

Solución completa con validación y manejo de estado

3

## Yup

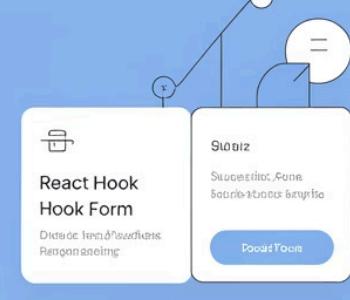
Esquemas de validación potentes y expresivos

# React Form Libraries

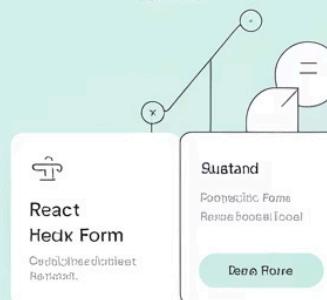
Coocnemiring



Lorementiring



Folmeres



Cococing



# Reflexión

# Ventajas de Inputs Controlados

## Predictibilidad

El estado siempre refleja lo que ve el usuario. No hay sorpresas ni inconsistencias entre el DOM y React.

## Validación en Tiempo Real

Puedes validar y dar feedback inmediatamente, mejorando la experiencia del usuario significativamente.

## Formateo Dinámico

Puedes transformar datos mientras el usuario escribe:  
mayúsculas, formatos de moneda, máscaras de  
teléfono.

## Testing Más Fácil

Al tener el estado explícito, es más sencillo escribir pruebas unitarias y de integración.

# Puntos Clave para Recordar



## **Los eventos son la base de la interactividad**

Sin eventos, React sería solo una biblioteca de renderizado estático



## **Inputs controlados dan mayor control**

Aunque requieren más código, ofrecen predictibilidad y flexibilidad



## **Validación temprana mejora UX**

Los usuarios prefieren saber inmediatamente si algo está mal

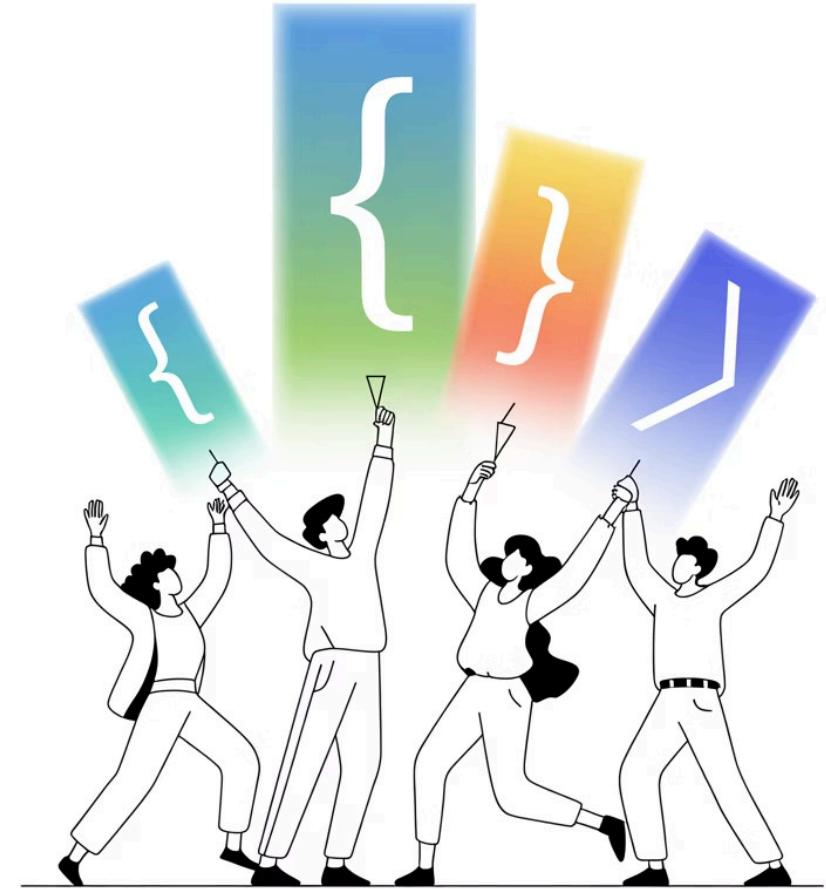


## **Accesibilidad es fundamental**

Un buen formulario debe ser usable por todos

# ¡Gracias!

¿Preguntas sobre eventos y formularios en React?



**CODE VICTORY!**

# #EDCOUNIANDES

<https://educacioncontinua.uniandes.edu.co/>

Contacto: [educacion.continua@uniandes.edu.co](mailto:educacion.continua@uniandes.edu.co)

© - Derechos Reservados: La presente obra, y en general todos sus contenidos, se encuentran protegidos por las normas internacionales y nacionales vigentes sobre propiedad Intelectual, por lo tanto su utilización parcial o total, reproducción, comunicación pública, transformación, distribución, alquiler, préstamo público e importación, total o parcial, en todo o en parte, en formato impreso o digital y en cualquier formato conocido o por conocer, se encuentran prohibidos, y solo serán lícitos en la medida en que se cuente con la autorización previa y expresa por escrito de la Universidad de los Andes.