React Forms

Sistemas y Tecnologías Web (CC3062) - 2026

Formularios en React

Semestre 01, 2026

Formularios sin librerías

Un formulario de registro básico con estado controlado.

```
function RegistroForm() {
  const [nombre, setNombre] = useState('');
  const [email, setEmail]   = useState('');
  const [edad, setEdad]     = useState('');
  const [errores, setErrores] = useState({});

  function validar() {
    const nuevosErrores = {};
    if (!nombre) nuevosErrores.nombre = 'El nombre es requerido';
    if (!email.includes('@')) nuevosErrores.email = 'Email inválido';
    if (parseInt(edad) < 18) nuevosErrores.edad = 'Debes ser mayor de edad';
    return nuevosErrores;
  }

  function handleSubmit(e) {
    e.preventDefault();
    const errs = validar();
    if (Object.keys(errs).length > 0) {
      setErrores(errs);
```

```
      return;
    }
    // enviar datos...
  }

  return (
    <form onSubmit={handleSubmit}>
      <input value={nombre} onChange={e => setNombre(e.target.value)} />
      {errores.nombre && <p>{errores.nombre}</p>}
      {/* ... más campos */}
    </form>
  );
}
```

El problema escala rápido

Con 3 campos: manejable.

Con 10 campos: un `useState` por campo, lógica de validación manual, manejo de errores disperso.

Con 20 campos y reglas complejas: código difícil de mantener, propenso a bugs, reinventando la rueda.

Lo que necesitamos que una librería resuelva

- Registrar campos sin un `useState` por cada uno.
- Validar con reglas declarativas, no código imperativo.
- Mostrar errores solo cuando corresponde (al hacer blur, al intentar enviar).
- Manejar el estado de envío (cargando, éxito, error).
- Rendimiento: no re-renderizar toda la forma en cada tecla.

React Hook Form

React Hook Form es la librería estándar para formularios en React.

- No re-renderiza el componente en cada keystroke — usa refs no controlados internamente.
- Integración nativa con esquemas de validación (Zod, Yup).
- API basada en hooks — sin componentes especiales que envuelvan el formulario.

```
npm install react-hook-form
```

El hook principal: useForm

```
import { useForm } from 'react-hook-form';

function RegistroForm() {
  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm();

  function onSubmit(data) {
    console.log(data); // { nombre: "Ana", email: "ana@uvg.edu" }
  }

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input {...register('nombre')} placeholder="Nombre" />
      {errors.nombre && <p>{errors.nombre.message}</p>}

      <input {...register('email')} placeholder="Email" />
      {errors.email && <p>{errors.email.message}</p>}

      <button type="submit">Registrar</button>
    </form>
```

```
  );
}
```

- `register('nombre')` conecta el input con React Hook Form.
- `handleSubmit(onSubmit)` valida antes de llamar a `onSubmit`.
- `errors` contiene los mensajes de error de cada campo.

Lo que devuelve register

```
const { ref, name, onChange, onBlur } = register('nombre');

// Es equivalente a pasar estas props al input:
<input
  ref={ref}
  name="nombre"
  onChange={onChange}
  onBlur={onBlur}
/>

// Por eso se usa el spread:
<input {...register('nombre')} />
```

Validación integrada en register

```
<input
  {...register('nombre', {
    required: 'El nombre es requerido',
    minLength: { value: 2, message: 'Mínimo 2 caracteres' },
    maxLength: { value: 50, message: 'Máximo 50 caracteres' },
  })}
/>

<input
  {...register('email', {
    required: 'El email es requerido',
```

```
    pattern: {
      value: /^[^\s@]+@[^\s@]+\.[^\s@]+$/,
      message: 'Formato de email inválido',
    },
  })}
/>

<input
  {...register('edad', {
    required: true,
    min: { value: 18, message: 'Debes ser mayor de edad' },
    max: { value: 120, message: 'Edad inválida' },
    valueAsNumber: true,
  })}
  type="number"
/>
```

Reglas de validación disponibles

| Regla | Descripción | Ejemplo | | --- | --- | --- | | `required` | Campo obligatorio | `required: 'Campo requerido'` | | `minLength` | Longitud mínima | `minLength: { value: 3, message: '...' }` | | `maxLength` | Longitud máxima | `maxLength: { value: 100, message: '...' }` | | `min` | Valor numérico mínimo | `min: { value: 0, message: '...' }` | | `max` | Valor numérico máximo | `max: { value: 999, message: '...' }` | | `pattern` | Expresión regular | `pattern: { value: /regex/, message: '...' }` | | `validate` | Función de validación custom | `validate: v => v !== 'admin' \|\| 'No permitido'` |

formState: más que solo errores

```
const {
  register,
  handleSubmit,
  formState: {
    errors,       // errores de validación
```

```
    isSubmitting, // true mientras onSubmit está ejecutándose (async)
    isDirty,      // true si algún campo fue modificado
    isValid,      // true si no hay errores
  },
} = useForm();

// Deshabilitar el botón mientras se envía
<button type="submit" disabled={isSubmitting}>
  {isSubmitting ? 'Enviando...' : 'Enviar'}
</button>
```

watch y setValue

```
const { register, handleSubmit, watch, setValue } = useForm();

// Observar el valor de un campo en tiempo real
const password = watch('password');

// Campos que dependen de otros
<input
  {...register('confirmar_password', {
    validate: value =>
      value === password || 'Las contraseñas no coinciden',
  })}
/>

// Asignar un valor programáticamente
<button type="button" onClick={() => setValue('nombre', 'Ana')}>
  Rellenar nombre
</button>
```

Zod

Zod es una biblioteca de validación y definición de esquemas para TypeScript y JavaScript.

Define la forma de los datos una sola vez — sirve tanto para validar como para inferir tipos.

```
npm install zod @hookform/resolvers
```

Por qué Zod en lugar de validación en register

Las reglas en `register` son para campos individuales — no pueden expresar relaciones entre campos ni lógica compleja.

Zod define el esquema completo del formulario en un solo lugar:

```
import { z } from 'zod';

const esquema = z.object({
  nombre:   z.string().min(2, 'Mínimo 2 caracteres').max(50),
  email:    z.string().email('Email inválido'),
  edad:     z.number().min(18, 'Debes ser mayor de edad'),
  password: z.string().min(8, 'Mínimo 8 caracteres'),
  confirmar: z.string(),
}).refine(
  data => data.password === data.confirmar,
  { message: 'Las contraseñas no coinciden', path: ['confirmar'] }
);
```

`refine` permite validaciones que dependen de múltiples campos.

Tipos de datos en Zod

```
z.string()           // texto
z.number()           // número
z.boolean()          // true/false
z.date()             // fecha
z.enum(['A', 'B'])   // valor fijo de un conjunto
z.array(z.string())  // arreglo de strings
```

```
z.object({ ... })    // objeto con campos definidos

// Modificadores
z.string().optional()          // puede ser undefined
z.string().nullable()          // puede ser null
z.string().default('valor')    // valor por defecto
```

Validaciones comunes de Zod

```
// Strings
z.string().min(3).max(100)
z.string().email()
z.string().url()
z.string().regex(/^\d{4}-\d{2}-\d{2}$/, 'Formato YYYY-MM-DD')
z.string().trim().min(1, 'No puede estar vacío')

// Números
z.number().int('Debe ser entero').positive('Debe ser positivo')
z.number().min(0).max(100)

// Coerción (convertir string de input a número)
z.coerce.number().min(0)
z.coerce.date()
```

Integrar Zod con React Hook Form

```
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { z } from 'zod';

const esquema = z.object({
  nombre: z.string().min(2, 'Mínimo 2 caracteres'),
  email:  z.string().email('Email inválido'),
  edad:   z.coerce.number().min(18, 'Debes ser mayor de edad'),
});
```

```
function RegistroForm() {
  const {
    register,
    handleSubmit,
    formState: { errors, isSubmitting },
  } = useForm({
    resolver: zodResolver(esquema),
  });

  async function onSubmit(data) {
    // data ya está validado y tiene los tipos correctos
    await crearUsuario(data);
  }

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <div>
        <input {...register('nombre')} placeholder="Nombre" />
        {errors.nombre && <p className="error">{errors.nombre.message}</p>}
      </div>

      <div>
        <input {...register('email')} placeholder="Email" />
        {errors.email && <p className="error">{errors.email.message}</p>}
      </div>

      <div>
        <input {...register('edad')} type="number" placeholder="Edad" />
        {errors.edad && <p className="error">{errors.edad.message}</p>}
      </div>

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

Ejemplo completo: Formulario de login

```
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { z } from 'zod';

const esquemaLogin = z.object({
  email:    z.string().email('Email inválido'),
  password: z.string().min(8, 'Mínimo 8 caracteres'),
});

function LoginForm() {
  const {
    register,
    handleSubmit,
    formState: { errors, isSubmitting },
    setError,
  } = useForm({ resolver: zodResolver(esquemaLogin) });

  async function onSubmit(data) {
    try {
      await iniciarSesion(data.email, data.password);
      // redirigir al dashboard
    } catch (error) {
      // Error del servidor → error en un campo específico
      setError('root', { message: 'Email o contraseña incorrectos' });
    }
  }

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      {errors.root && (
        <div className="error-global">{errors.root.message}</div>
      )}

      <label>
        Email
        <input
```

```
          {...register('email')}
          type="email"
          autoComplete="email"
        />
        {errors.email && <span>{errors.email.message}</span>}
      </label>

      <label>
        Contraseña
        <input
          {...register('password')}
          type="password"
          autoComplete="current-password"
        />
        {errors.password && <span>{errors.password.message}</span>}
      </label>

      <button type="submit" disabled={isSubmitting}>
        {isSubmitting ? 'Entrando...' : 'Iniciar sesión'}
      </button>
    </form>
  );
}
```

`setError('root', ...)` — para errores que no pertenecen a un campo específico.

Ejemplo completo: Formulario de registro con confirmación

```
const esquemaRegistro = z.object({
  nombre:   z.string().min(2, 'Mínimo 2 caracteres'),
  email:    z.string().email('Email inválido'),
  password: z.string()
    .min(8, 'Mínimo 8 caracteres')
    .regex(/[A-Z]/, 'Debe contener al menos una mayúscula')
```

```
    .regex(/[0-9]/, 'Debe contener al menos un número'),
  confirmar: z.string(),
  rol: z.enum(['estudiante', 'docente'], {
    errorMap: () => ({ message: 'Selecciona un rol' }),
  }),
}).refine(
  data => data.password === data.confirmar,
  { message: 'Las contraseñas no coinciden', path: ['confirmar'] }
);

function RegistroForm() {
  const { register, handleSubmit, formState: { errors } } = useForm({
    resolver: zodResolver(esquemaRegistro),
  });

  return (
    <form onSubmit={handleSubmit(console.log)}>

      <input {...register('nombre')} placeholder="Nombre completo" />
      {errors.nombre && <p>{errors.nombre.message}</p>}

      <input {...register('email')} type="email" placeholder="Email" />
      {errors.email && <p>{errors.email.message}</p>}

      <input {...register('password')} type="password" placeholder="Contraseña" />
      {errors.password && <p>{errors.password.message}</p>}

      <input {...register('confirmar')} type="password" placeholder="Confirmar contraseña
      {errors.confirmar && <p>{errors.confirmar.message}</p>}

      <select {...register('rol')}>
        <option value="">Selecciona un rol</option>
        <option value="estudiante">Estudiante</option>
        <option value="docente">Docente</option>
      </select>
      {errors.rol && <p>{errors.rol.message}</p>}

      <button type="submit">Crear cuenta</button>
```

```
    </form>
  );
}
```

Cuándo mostrar errores

Por defecto, React Hook Form muestra los errores al intentar enviar el formulario.

Se puede cambiar con el modo de validación:

```
const { register, handleSubmit, formState: { errors } } = useForm({
  resolver: zodResolver(esquema),
  mode: 'onBlur',       // validar al salir del campo
  // mode: 'onChange',  // validar en cada tecla (agresivo)
  // mode: 'onSubmit',  // solo al enviar (por defecto)
  // mode: 'all',       // onChange + onBlur
});
```

| Modo | Cuándo se muestra el error | UX | | --- | --- | --- | | `onSubmit` | Solo al intentar enviar | Menos disruptivo | | `onBlur` | Al salir del campo | Balance recomendado | | `onChange` | En cada keystroke | Muy agresivo para el usuario |

Comparación: sin librería vs con React Hook Form + Zod

| Aspecto | Sin librería | React Hook Form + Zod | | --- | --- | --- | | Estado | Un `useState` por campo | Ninguno — RHF lo maneja | | Validación | Código imperativo manual | Esquema declarativo en Zod | | Errores | Manejar manualmente | `errors.campo.message` | | Relación entre campos | Código complejo | `refine` en el esquema | | Rendimiento | Re-render en cada tecla | Sin re-renders innecesarios | | Envío async | Manejar `loading` manualmente | `isSubmitting`

automático | | Tipos (TypeScript) | Sin inferencia | Tipos inferidos del esquema Zod |

Principios clave

- `useForm` + `register` + `handleSubmit` es la base de React Hook Form
- Zod define el esquema completo del formulario — validaciones, tipos y relaciones entre campos
- `zodResolver` conecta el esquema Zod con React Hook Form
- `errors.campo.message` muestra el mensaje de error en el JSX
- `isSubmitting` deshabilita el botón mientras se procesa el envío
- `setError('root', ...)` para errores del servidor que no pertenecen a un campo
- Validar **siempre en el servidor** también — la validación del cliente es para UX, no para seguridad