1. Validação manual (sem bibliotecas)

É a forma mais simples, usando `useState` para controlar o formulário e verificar os dados antes de enviar.

Exemplo básico

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

function Formulario() {
  const [nome, setNome] = useState("");
  const [email, setEmail] = useState("");
  const [erros, setErros] = useState({});

  function validar() {
    const novosErros = {};

    if (!nome.trim()) {
      novosErros.nome = "O nome é obrigatório.";
    }

    if (!email.trim()) {
      novosErros.email = "O email é obrigatório.";
    } else if (!email.includes("@")) {
      novosErros.email = "Formato de email inválido.";
    }

    setErros(novosErros);
    return Object.keys(novosErros).length === 0;
  }

  function handleSubmit(e) {
    e.preventDefault();
    if (!validar()) return;

    console.log("Formulário válido:", { nome, email });
  }

  return (
    <form onSubmit={handleSubmit}>
      <div>
```

```
        <label>Nome</label>
        <input
          value={nome}
          onChange={(e) => setNome(e.target.value)}
        />
        {erros.nome && <p style={{ color: "red" }}>{erros.nome}</p>}
      </div>

      <div>
        <label>Email</label>
        <input
          value={email}
          onChange={(e) => setEmail(e.target.value)}
        />
        {erros.email && <p style={{ color: "red" }}>{erros.email}</p>}
      </div>

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

export default Formulario;
```

✔️ Bom para aplicações pequenas
✔️ Total controle sobre as regras
❌ Muito código repetido quando a aplicação cresce

2. Validação com HTML5 (opcional)

Alguns casos dá para usar atributos nativos:

```
<input
  type="email"
  required
  minLength={5}
/>
```

Mas **não substitui uma validação de verdade** — apenas auxilia.

✅ 3. Validação com Yup + React Hook Form (mais profissional)

Essa é a abordagem mais moderna e utilizada no mercado.

Instalar:

```
npm install react-hook-form yup @hookform/resolvers
```

Exemplo completo:

```
import { useForm } from "react-hook-form";
import { yupResolver } from "@hookform/resolvers/yup";
import * as yup from "yup";

const schema = yup.object({
  nome: yup.string().required("O nome é obrigatório"),
  email: yup.string()
    .email("Email inválido")
    .required("O email é obrigatório"),
  idade: yup.number()
    .positive("Idade deve ser positiva")
    .required("Idade é obrigatória")
});

function Formulario() {
  const {
    register,
    handleSubmit,
    formState: { errors }
  } = useForm({
    resolver: yupResolver(schema)
  });

  function onSubmit(data) {
    console.log("Dados válidos:", data);
  }
```

```
  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <div>
        <label>Nome</label>
        <input {...register("nome")} />
        <p style={{ color: "red" }}>{errors.nome?.message}</p>
      </div>

      <div>
        <label>Email</label>
        <input {...register("email")} />
        <p style={{ color: "red" }}>{errors.email?.message}</p>
      </div>

      <div>
        <label>Idade</label>
        <input type="number" {...register("idade")} />
        <p style={{ color: "red" }}>{errors.idade?.message}</p>
      </div>

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

export default Formulario;
```

✔️ Validação robusta
✔️ Muito menos código
✔️ Suporta formulários grandes e complexos
✔️ Possibilita exibir erros automaticamente

✅ 4. Validação em tempo real (onChange)

Para feedback imediato ao usuário:

```
function handleNomeChange(e) {
  const valor = e.target.value;
  setNome(valor);
```

```
  if (!valor.trim()) {
    setErros((prev) => ({ ...prev, nome: "Nome obrigatório" }));
  } else {
    setErros((prev) => ({ ...prev, nome: null }));
  }
}
```

📌 Em resumo

Método	Quando usar
Validação manual	Formulários pequenos, aprendizado.
HTML5	Regras simples.
React Hook Form + Yup	Projetos reais, formulários grandes, validação profissional.