Forms in React

Forms that listen: controlled inputs, one state object for the whole form, the [e.target.name] trick, checkboxes and selects, live validation and submit.

Introduction

An HTML form keeps its own values inside the DOM, and you dig them out at the end. React flips the ownership: **the state holds the values, and every input shows what state says.** These are **controlled inputs** — you met the pattern in useState; here it grows into complete, validated, real forms. *(Roman Urdu: React form mein malik state hota hai — input wahi dikhata hai jo state mein hai.)*

Key points

- **Controlled vs uncontrolled:** a controlled input has `value` from state + `onChange` saving every keystroke. An uncontrolled input is left to the DOM (read later with a ref) — fine for a quick file field, wrong for forms you must validate live.
- **One object for the whole form:** instead of five `useState` calls, keep one object: `const [form, setForm] = useState({ name: "", email: "", course: "React" })`.
- **The `[e.target.name]` trick:** give every input a `name` equal to its key, and one handler updates them all: `setForm({ ...form, [e.target.name]: e.target.value })` — the brackets mean "the key whose name is in this variable" (computed property, from the JavaScript series).

  (Roman Urdu: Har input ka name usi key jaisa rakho — phir aik hi handler saare fields sambhal lega.)
- **Checkboxes speak boolean:** a checkbox's value lives in `e.target.checked` (true/false), not `e.target.value` — handle it in the same handler by checking the input's `type`.
- **Selects are inputs too:** a `<select>` with `value` + `onChange` behaves like any controlled field; its options carry the values.
- **Submit is a React event:** `onSubmit` on the `<form>`, and `e.preventDefault()` stops the browser's page reload so your code can validate and decide.
- **Live validation:** because state knows every keystroke, errors can appear *while typing* (empty name, email without `@`) and the submit button can stay disabled until the form is clean.

Code examples

Example 1 — One state object, one handler, whole form

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

function SignupForm() {
  const [form, setForm] = useState({ name: "", email: "" });

  function handleChange(e) {
    setForm({ ...form, [e.target.name]: e.target.value });
  }

  return (
    <form>
      <input name="name" value={form.name} onChange={handleChange} placeholder="Name" />
      <input name="email" value={form.email} onChange={handleChange} placeholder="Email" />
      <p>Joining as: {form.name} ({form.email})</p>
    </form>
  );
}
```

What it does: Each input's `name` matches its key in the state object. `handleChange` spreads the old form and overwrites only the field that changed — one handler serves every text field you will ever add to this form.

What you will see: as you type in either box, the line below updates live — "Joining as: Ayesha (ayesha@mail.com)".

Example 2 — Checkbox and select join the same form

```
function CourseForm() {
  const [form, setForm] = useState({ course: "React", agree: false });

  function handleChange(e) {
    const value = e.target.type === "checkbox" ? e.target.checked : e.target.value;
    setForm({ ...form, [e.target.name]: value });
  }

  return (
    <form>
      <select name="course" value={form.course} onChange={handleChange}>
        <option value="React">React</option>
        <option value="JavaScript">JavaScript</option>
        <option value="CSS">CSS</option>
      </select>
      <label>
        <input type="checkbox" name="agree" checked={form.agree} onChange={handleChange} />
        I agree to the class rules
      </label>
      <button disabled={!form.agree}>Join {form.course}</button>
    </form>
  );
}
```

What it does: The handler now asks one question — is this a checkbox? — and reads `checked` instead of `value`. The select is an ordinary controlled field. The button reads state too: it stays disabled until the agreement box is ticked, and its label names the chosen course.

What you will see: the button is greyed out until you tick "I agree", then it lights up as "Join React" — or "Join CSS" if you change the select.

Example 3 — Validation and submit, no reload

```
function ContactForm() {
  const [email, setEmail] = useState("");
  const [sent, setSent] = useState(false);
  const error = email !== "" && !email.includes("@") ? "Email must contain @" : "";

  function handleSubmit(e) {
    e.preventDefault();
    if (error || email === "") return;
    setSent(true);
  }

  if (sent) return <p>Message sent. Thank you!</p>;

  return (
    <form onSubmit={handleSubmit}>
      <input value={email} onChange={(e) => setEmail(e.target.value)} placeholder="Email" />
      {error && <p className="error">{error}</p>}
      <button type="submit">Send</button>
    </form>
  );
}
```

What it does: The error is *computed from state on every render*, so it appears the moment the email looks wrong, with no extra code. Submit first stops the browser reload with `preventDefault`, refuses to send while invalid, and then swaps the whole form for a thank-you message.

What you will see: type "ayesha" and "Email must contain @" appears; fix it to "ayesha@mail.com" and the warning vanishes; press Send and the form is replaced by "Message sent. Thank you!" — the page never reloads.

Quick recap

- React forms are **controlled**: state owns the values, inputs display state, `onChange` saves every keystroke.
- One **state object** + the `[e.target.name]` handler runs a whole form; checkboxes read `checked`, selects behave like inputs.
- Submit means `onSubmit` + `e.preventDefault()`; validation can be computed live from state on every render.

Common mistakes

1. **`value` without `onChange`** — the input becomes read-only and typing does nothing, because React keeps forcing it back to the state value. The fix: a controlled input always carries both, or drop `value` and go honestly uncontrolled.
2. **Mutating the form object** — `form.name = "Ali"; setForm(form)` hands React the same object, and the render may not happen. The fix: always a fresh object — `setForm({ ...form, name: "Ali" })`.
3. **Forgetting `preventDefault`** — the form submits the browser way, the page reloads, and all state is lost. The fix: first line of every submit handler, always.

Practice

1. Build a `StudentForm` with one state object: name, city (text inputs), course (select with React / JavaScript / CSS), and hostel (checkbox). Use the single `[e.target.name]` handler for all of them, and show a live summary card under the form.
2. Add validation to it: name must be at least 3 characters, and show the error under the name field only while it is wrong. Disable the submit button until the form is fully valid, and on submit replace the form with "Registered: {name} — {course}".

Practice quiz

1. What makes an input controlled in React?

- It has a name attribute
- It has value from state plus onChange saving every keystroke
- It is inside a form
- It has a ref

Answer: It has value from state plus onChange saving every keystroke

2. How does one handler update every field of a whole form?

- It cannot, each field needs its own handler
- Give every input a name equal to its key, and use setForm({ ...form, [e.target.name]: e.target.value })
- By reading the DOM after submit
- By using one useState per field

Answer: Give every input a name equal to its key, and use setForm({ ...form, [e.target.name]: e.target.value })

3. What does a checkbox's value live in?

- e.target.value, like a text input
- e.target.checked, true or false, not e.target.value
- e.target.name
- The form's state object directly

Answer: e.target.checked, true or false, not e.target.value

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/react-forms · More free tools and notes at https://scopetreasure.com