Portals

Rendering components to a different DOM node with react-dom's createPortal -- modals, event bubbling behavior, and setting up a portal target, with simple examples.

Advanced 12 min

Last updated: August 18, 2026
Read online: https://www.learnforgex.com/en/topics/portals

Portals

The last topic in the Advanced React category -- a way to render a component to a different DOM node than its position in the React tree: **Portals**.

What Is a Portal? Getting Started with createPortal

`react-dom`'s `createPortal` function lets us render a component to a different place in the DOM:

```
import { createPortal } from "react-dom";

function BasicPortalExample() {
  // createPortal(child, container) renders `child` into a different DOM
  // node (`container`) INSTEAD OF its place in the normal React tree --
  // here, `document.body` itself. In the component tree (in React
  // DevTools) it still appears INSIDE BasicPortalExample, but in the
  // actual DOM it's in a completely different location.
  return createPortal(<p className="tooltip">I'm rendered directly on body!</p>, document.
body);
}
```

`createPortal(child, container)` renders `child` into the `container` DOM node, instead of its normal position in the React tree. It still appears in its expected place in the component tree (in React DevTools), but its actual DOM position is completely different.

Using Portals for Modals

The most common use case for Portals is modals:

```
import { useState } from "react";
import { createPortal } from "react-dom";

function Modal({ onClose, children }) {
  // A modal is the most common use case for a Portal -- the modal's CSS
  // (position: fixed, z-index) makes it appear ON TOP of the rest of the
  // page, but its actual DOM position (e.g. "inside a card") can
  // sometimes block that (e.g. overflow: hidden). The Portal ELIMINATES
  // this problem by rendering the modal directly into `document.body`.
  return createPortal(
    <div className="modal-overlay" onClick={onClose}>
      <div className="modal-content" onClick={(event) => event.stopPropagation()}>
        {children}
        <button onClick={onClose}>Close</button>
      </div>
    </div>,
    document.body,
  );
}

function ModalWithPortalExample() {
  const [isOpen, setIsOpen] = useState(false);

  return (
    <div>
      <button onClick={() => setIsOpen(true)}>Open Modal</button>
      {isOpen && (
        <Modal onClose={() => setIsOpen(false)}>
          <p>This is a modal, rendered outside the normal DOM tree.</p>
        </Modal>
      )}
    </div>
  );
}
```

A modal's CSS (`position: fixed` , a high `z-index`) needs to make it appear ABOVE the rest of the page -- but the modal's actual DOM position (say, inside a card with `overflow: hidden`) can sometimes prevent that. A Portal ELIMINATES this problem by rendering the modal directly into `document.body` .

Event Bubbling: A Portal's Surprising Behavior

The most important (and most surprising) property of Portals is how events behave:

```
import { useState } from "react";
import { createPortal } from "react-dom";

function Popup() {
  return createPortal(<button>Click me (rendered in document.body)</button>, document.body);
}

function EventBubblingThroughPortalExample() {
  const [clicks, setClicks] = useState(0);

  return (
    // IMPORTANT: Popup is rendered in the DOM OUTSIDE this <div> (in
    // document.body). But when its <button> is clicked, onClick still
    // fires HERE (at its real position in the React tree) -- React
    // "bubbles" events according to its OWN component tree, not the
    // actual DOM tree. This is the most surprising yet most useful
    // feature of Portals.
    <div onClick={() => setClicks(clicks + 1)}>
      <p>Clicks: {clicks}</p>
      <Popup />
    </div>
  );
}
```

`Popup` renders OUTSIDE the outer `<div>` in the DOM (into `document.body`). But clicking the button inside it still causes `onClick` to bubble up to the outer `<div>` -- React propagates events according to its OWN component tree, not the actual DOM tree. This is the most important behavior to know when using Portals.

Setting Up a Portal Target

Instead of `document.body`, a dedicated target is usually used:

```
import { createPortal } from "react-dom";

function Tooltip({ text }) {
  // Instead of document.body, it's more common to use a target set aside
  // specifically in index.html -- for example <div id="tooltip-root"></div>,
  // added as a SIBLING to the app's #root. This makes it easier for the
  // portal content to manage its own styles/position.
  const target = document.getElementById("tooltip-root");

  if (!target) {
    return null;
  }

  return createPortal(<span className="tooltip">{text}</span>, target);
}

function PortalTargetSetupExample() {
  return (
    <div>
      <p>Hover for more info</p>
      <Tooltip text="This tooltip lives in its own DOM node." />
    </div>
  );
}
```

Adding something like `<div id="tooltip-root"></div>` as a SIBLING to the app's `#root` in `index.html` is common practice -- it makes it easier for the portal's content to manage its own styles and positioning.

Summary and Glossary

`createPortal(child, container)` renders a component to a different DOM node while KEEPING its position in the React tree -- the most common uses are modals, tooltips, and dropdowns (to avoid CSS properties like `overflow: hidden` on ancestor elements). Events bubble according to React's component tree, not the actual DOM position -- this lets us keep using Portals like normal components.

Glossary

Portal — A mechanism for rendering a component to a different DOM node while preserving its position in the React tree.

Event Bubbling — An event propagating upward from the element it was triggered on, through its ancestor elements.

Practical Project

There's a real, runnable example project that brings together the concepts from this category (React Performance, Error Boundaries, Lazy Loading & Code Splitting, Suspense, Portals): **Advanced React Demo** -- an application showing a course list optimized with

`React.memo`, an Error Boundary, a detail panel code-split with `React.lazy` + `Suspense`, and a Portal modal, all working together.

You can download it and run it yourself, and read through the code line by line:

```
git clone https://github.com/cdurgun/react-course-projects.git
cd react-course-projects
npm install
cd projects/advanced-react
npm run dev
```

The `react-course-projects` repo uses **npm workspaces** -- `npm install` only needs to run once, at the repo root, and every project folder shares the same dependencies (no separate `node_modules` per folder). If you've already run `npm install` at the root, you can just `cd react-course-projects/projects/advanced-react` and run `npm run dev`.