**🧠 1️⃣ What are React Portals?**

Normally, when you render components in React, they stay **inside their parent’s DOM hierarchy**.

But sometimes…  
you want a component to **visually appear outside** its parent — like a **modal** or **tooltip** that floats on top of everything.

👉 **React Portal** lets you render a component **outside** of its normal DOM tree,  
**while still keeping it inside the same React tree (for logic/state).**

**🏠 Example Without Portal**

function App() {

return (

<div className="app">

<h1>Main App</h1>

<Modal /> {/\* rendered inside .app \*/}

</div>

);

}

Your HTML structure looks like:

<div class="app">

<h1>Main App</h1>

<div class="modal">Modal content...</div>

</div>

⚠️ Problem:  
If .app has CSS like overflow: hidden or z-index: 1,  
your modal might **get cut off or hidden** behind other elements.

**✨ 2️⃣ Portal Fixes That**

With **React Portals**, you can render that modal **outside of .app**,  
for example directly inside <body> — so it appears above everything else.

**🔧 Example with React Portal**

**Step 1️⃣ — Create a modal root in your HTML:**

In your index.html file, add:

<body>

<div id="root"></div>

<div id="modal-root"></div>

</body>

**Step 2️⃣ — Create a Modal component using a portal**

import ReactDOM from "react-dom";

function Modal({ children }) {

const modalRoot = document.getElementById("modal-root");

return ReactDOM.createPortal(

<div className="modal">{children}</div>,

modalRoot

);

}

**Step 3️⃣ — Use it anywhere in your React tree**

function App() {

return (

<div className="app">

<h1>Main App</h1>

<Modal>

<h2>I’m a modal!</h2>

<p>This is rendered outside the app div but works like normal React!</p>

</Modal>

</div>

);

}

✅ Output (in HTML):

<div class="app">

<h1>Main App</h1>

</div>

<div id="modal-root">

<div class="modal">

<h2>I’m a modal!</h2>

<p>This is rendered outside the app div but works like normal React!</p>

</div>

</div>

**🧩 3️⃣ Why Portals Are Useful**

✅ **Use cases:**

* Modals / Dialog boxes
* Tooltips / Popovers
* Dropdown menus
* Floating notifications

✅ **Why needed:**

* Avoid CSS issues like z-index, overflow: hidden
* Keep logic in React while rendering outside visually

**⚙️ 4️⃣ How It Still Works**

Even though the modal is outside the main #root in HTML,  
it’s still part of the **React component tree**.

So:

* State and props still flow normally
* Events like onClick, onSubmit still work as usual
* React lifecycle hooks (useEffect, etc.) behave the same

**🧠 5️⃣ Simple Tooltip Example**

function Tooltip({ text }) {

const tooltipRoot = document.getElementById("modal-root");

return ReactDOM.createPortal(

<div className="tooltip">{text}</div>,

tooltipRoot

);

}

function ButtonWithTooltip() {

return (

<button>

Hover me

<Tooltip text="I’m a tooltip rendered via portal!" />

</button>

);

}

**🧾 Summary**

| **Concept** | **Explanation** |
| --- | --- |
| **Portal** | Allows rendering of React components outside the parent DOM hierarchy |
| **Created using** | ReactDOM.createPortal(child, container) |
| **Best for** | Modals, tooltips, dropdowns, popovers |
| **Keeps logic** | Inside same React tree (so props/state still work) |

**🧩 In one line:**

**Portals let you place a React component outside its parent in the DOM for layout or layering reasons — while still keeping it logically inside the React app.**

**🧱 1️⃣ What does a normal React app look like in HTML?**

When you create a React app (like using Vite or Create React App), your **index.html** looks like this 👇

<!DOCTYPE html>

<html>

<head>

<title>My React App</title>

</head>

<body>

<!-- This empty div is where ALL of React will go -->

<div id="root"></div>

</body>

</html>

👉 #root is like the **main container** —  
React takes over this empty box and **renders everything inside it**.

**🌳 2️⃣ When React runs…**

In your main.jsx (or index.js) you write:

import ReactDOM from "react-dom/client";

import App from "./App";

ReactDOM.createRoot(document.getElementById("root")).render(<App />);

That means:

“Hey React, please render <App /> inside the <div id="root"></div>.”

**💡 So inside the browser DOM, it becomes like:**

<div id="root">

<div class="App">

<h1>Hello</h1>

<button>Click</button>

</div>

</div>

✅ Everything React renders is **inside #root**.  
That’s React’s **normal DOM hierarchy**.

**🧩 3️⃣ So what’s the "DOM hierarchy"?**

Let’s picture it visually 🧠:

<html>

└── <body>

└── <div id="root"> ← React app starts here

└── <App> ← Your main React component

├── <Header />

├── <Sidebar />

└── <MainContent />

Every child stays *inside its parent* → like Russian nesting dolls 🪆.

**🚪 4️⃣ Problem — What if you want something ABOVE everything?**

Example:  
You want a **Modal**, **Tooltip**, or **Dropdown** that floats **on top of the entire app**,  
not trapped inside <App> or <Sidebar> (which might have overflow: hidden or z-index: 1).

But your structure looks like this 👇

<div id="root">

<div class="App">

<div class="sidebar"></div>

<div class="content">

<div class="modal">❌ Stuck inside content</div>

</div>

</div>

</div>

Now your modal might get **cut off** or **not visible properly**,  
because it’s still *inside* the app’s hierarchy.

**🧙‍♂️ 5️⃣ Solution — React Portal 🌀**

You can create a **new div** anywhere outside #root, for example:

<body>

<div id="root"></div>

<div id="modal-root"></div> <!-- 🟢 separate box just for modals -->

</body>

Then tell React:

“Hey React, even though I’m inside <App>,  
please render this component into #modal-root instead.”

That’s what Portals do!

**⚙️ Example**

import ReactDOM from "react-dom";

function Modal({ children }) {

const modalRoot = document.getElementById("modal-root");

return ReactDOM.createPortal(children, modalRoot);

}

Now in App.jsx:

function App() {

return (

<div className="App">

<h1>Main Page</h1>

<Modal>

<h2>I’m rendered OUTSIDE the #root DOM!</h2>

</Modal>

</div>

);

}

**🧩 Real DOM now looks like this:**

<div id="root">

<div class="App">

<h1>Main Page</h1>

</div>

</div>

<div id="modal-root">

<h2>I’m rendered OUTSIDE the #root DOM!</h2>

</div>

💥 Boom — your modal is **outside** of the main app’s DOM,  
but React still **controls** it, because it’s part of the same React tree logically.

**💡 6️⃣ Visual analogy (baby-friendly 🍼)**

Imagine your React app is a **house** 🏠.

* The main app (#root) is your **house interior**.
* The portal (#modal-root) is like a **balloon floating outside** the house —  
  but it’s still tied with a string 🎈 (React keeps control).

You can still talk to it, change its state, and control it,  
even though it’s outside visually.

**🧾 Summary**

| **Term** | **Meaning** |
| --- | --- |
| **DOM** | The HTML structure the browser renders |
| **React DOM tree** | React’s virtual representation of components |
| **Normal React render** | Everything stays inside #root |
| **Portal render** | You tell React to render somewhere else (like #modal-root) |
| **Why** | To avoid CSS/z-index issues and show floating UI cleanly |
|  |  |
| **⚙️ 3️⃣ When to use custom (portal-based) modals** |  |

| **❌ Reason to skip MUI/Bootstrap** | **✅ Use Portal-based custom modal when…** |
| --- | --- |
| You want **total freedom** of design | e.g. an animated floating modal with custom transitions |
| You’re building a **design system** (your own component library) | You can’t depend on MUI/Bootstrap |
| You need **lightweight bundle** | MUI adds ~100KB+; portals are small |
| You’re building **non-standard UIs** | Like custom tooltips, contextual popups, or nested modals |

Example:

* WhatsApp web → uses custom lightweight modals
* Admin dashboards → use MUI modals
* Design systems → use **Portal + custom modal component**

**🧮 4️⃣ In baby terms 👶**

Imagine you’re making a pizza 🍕

* Using **MUI Modal** → you order from Domino’s. Quick, reliable, consistent taste.
* Using **Portal Modal** → you bake it yourself. Takes longer but total control on toppings.

Both work — just depends if you want **speed and standard look** or **custom control**.

**🧠 5️⃣ Rule of thumb (for interviews + real projects)**

| **Project Type** | **Recommended** |
| --- | --- |
| **Internal tools / dashboards / admin apps** | ✅ MUI / Bootstrap Modal |
| **Public-facing website with brand design** | ✅ Custom Portal Modal |
| **Learning React fundamentals** | ✅ Build Portal Modal once manually |
| **Team project with UI library already used** | ✅ Stick with the same library Modal |

So in short 👉

✅ Use MUI / Bootstrap modals in most cases (saves time + built-in best practices)  
💡 Build your own with Portals only when you need custom animation, layout, or are building your own library.