**What is react?**

React is a Javascript library.

Used to make sigle page applications

Easy to learn (good developer experience)

Learn once use anywhere

React is an open-source **JavaScript library** used for building **user interfaces (UIs)**, especially for **single-page applications (SPAs)**. It was developed by **Facebook (now Meta)** and is maintained by both Meta and a community of developers.

**Component-Based Architecture**

* React follows a modular approach, allowing developers to build reusable UI components.
* This makes code **easier to manage, scale, and maintain**.

**Virtual DOM (Faster Performance)**

* React uses a **Virtual DOM**, which updates only the necessary parts of the UI instead of reloading the entire page.
* This makes applications **faster and more efficient**.

**Declarative Syntax (Easy to Read and Write)**

* React allows developers to **describe what the UI should look like**, and it handles updating the DOM efficiently.
* We don’t need to update the element on the DOM manually after updating the value
* JSX (JavaScript XML) makes writing UI components easier.

**Unidirectional Data Flow**

* React follows a **one-way data flow**, meaning data moves in a **single direction**, making debugging easier.

**Large Ecosystem & Community Support**

* React has a **huge developer community**, which means more libraries, resources, and job opportunities.

**Cross-Platform Development**

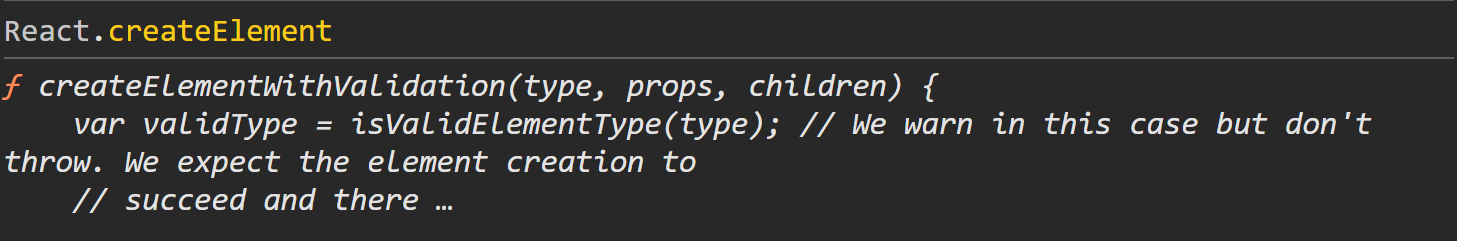
* With **React Native**, you can build mobile applications for **iOS and Android** using the same React principles.

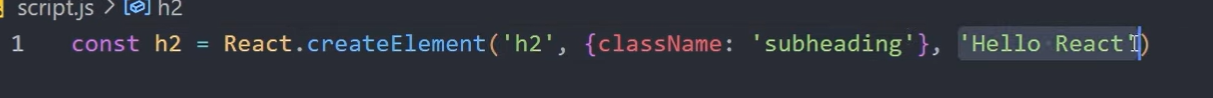
Basic structure of react app

You can use react cdn links to directly add react to your html

Creating element from react ->

**React.createElement**





How to add this element to Browser?

Using React DOM

Start by creating a root div on your html page(can use anything instead of root but root is conventional).

Then use ReactDOM.createRoot(document.querySelector(‘#root’)); to specify which element should we use as root element



Then use root.render method to render the element

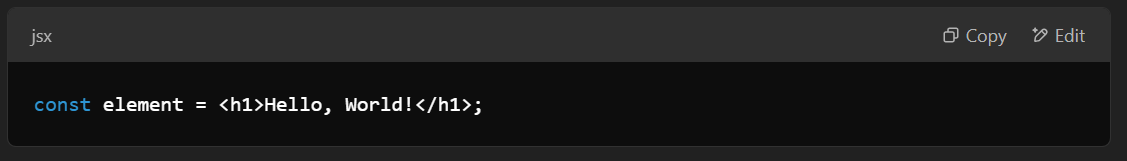


**What is JSX?**

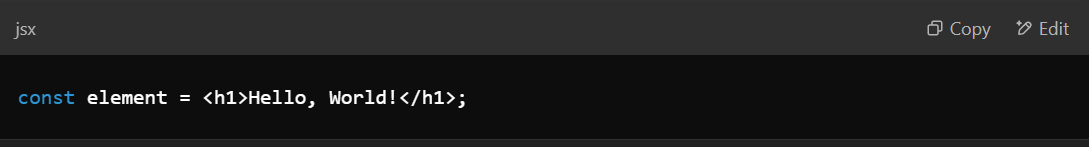
JSX (JavaScript XML) is a syntax extension for JavaScript that allows us to write HTML-like code inside JavaScript files. It makes it easier to create React components by allowing developers to write UI elements in a way that closely resembles HTML.

XML, or **Extensible Markup Language**, is a markup language designed for storing and transporting data, allowing users to define their own tags to structure and describe data in a way that is both human and machine-readable.

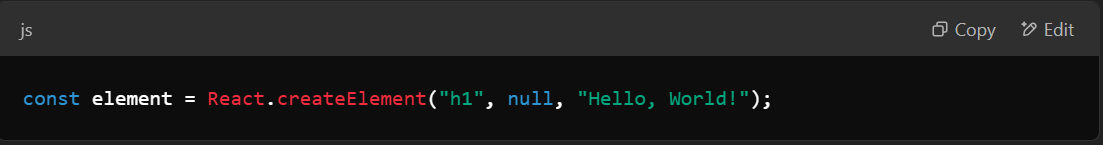
For example, instead of using React.createElement, we can write:



JSX is not directly understood by the browser, so it gets compiled by Babel into standard JavaScript before execution.

Under the hood, JSX: 

compiles to:

****

Key Features of JSX:

* Makes UI code more readable and maintainable.
* Can embed JavaScript expressions using {} (e.g., <h1>{name}</h1>).
* Prevents injection attacks by escaping values automatically.
* Must return a single parent element (use <React.Fragment> or <> if needed).

**What is Babel?**

Babel is a JavaScript compiler that converts modern JavaScript (ES6+) code into backward-compatible JavaScript that can run in older browsers.

Why is Babel used in React?

React uses JSX, which is not natively supported by browsers. Babel compiles JSX into standard JavaScript so that it can be interpreted correctly.

Key Features of Babel:

1. JSX to JavaScript conversion – Converts JSX into React.createElement().
2. ES6+ to ES5 transpilation – Converts modern JavaScript features (like arrow functions and classes) to ES5 for better browser support.
3. Plugin-based architecture – Allows customization with plugins like @babel/preset-react for JSX and @babel/preset-env for modern JavaScript.
4. Polyfills support – Can add missing JavaScript features using libraries like core-js.

Common Babel Presets for React:

* @babel/preset-env – Converts ES6+ syntax to ES5.
* **@babel/preset-react – Transforms JSX into JavaScript.**

**What are Source maps in React?**

A **source map** in React (or any JavaScript application) is a file that maps the **minified and bundled** code back to the **original source code**. This helps developers debug their applications by seeing the **original file names and line numbers** instead of the minified code.

**Why are Source Maps Needed?**

When React apps are built for production (npm run build), the JavaScript files are:  
✅ **Minified** (removing spaces and shortening variable names)  
✅ **Bundled** (merging multiple files into one or a few files)

This improves performance but makes debugging difficult. **Source maps solve this by allowing browsers to reconstruct the original code.**

**How Do Source Maps Work?**

* When a source map is enabled, an extra \*.map file is generated for each minified file.
* This file contains mappings between the **minified** and **original** source code.
* The browser reads the source map and allows debugging using the original source files in **DevTools**.

Example in a minified file:

js

CopyEdit

//# sourceMappingURL=main.js.map

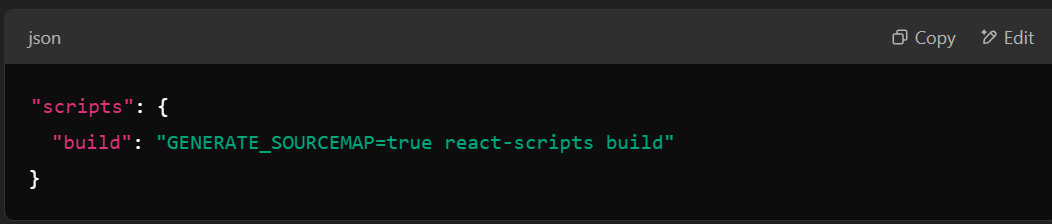
**How to Enable Source Maps in React?**

1️⃣ **For Development** (enabled by default in npm start):

* CRA (create-react-app) automatically generates source maps in **development mode** for easy debugging.

2️⃣ **For Production** (disabled by default for security reasons):

* You can enable source maps in production by modifying the package.json:



* But **be cautious!** Exposing source maps in production can reveal your original code to attackers.

**How to Use Source Maps in Chrome DevTools?**

1️⃣ Open your React app in **Google Chrome**  
2️⃣ Right-click → **Inspect** (or press F12)  
3️⃣ Go to **Sources** → Find your original .js or .tsx files  
4️⃣ Set breakpoints and debug just like in development mode 🎯

**What are bundlers in js?**

A bundler in JavaScript is a tool that combines multiple files (JavaScript, CSS, images, etc.) into a single optimized file (or a few smaller files) for efficient browser loading.

Why Do We Need Bundlers?

🔹 Modular Code: In modern JavaScript, we split code into multiple files (ES Modules, CommonJS). Browsers don’t support module imports efficiently in production, so bundlers merge them into a single file.

🔹 Performance Optimization:  
✅ Minification – Removes unnecessary spaces, comments, and renames variables to reduce file size.  
✅ Tree Shaking – Removes unused code to optimize performance.  
✅ Code Splitting – Breaks the app into smaller chunks for faster loading.

🔹 Cross-Browser Compatibility: Bundlers integrate Babel to convert modern JavaScript into versions supported by older browsers.

Popular JavaScript Bundlers

1️⃣ Webpack – Most commonly used, powerful with plugins and loaders.  
2️⃣ Vite – Faster alternative to Webpack, great for React, Vue, and modern frameworks.  
3️⃣ Parcel – Zero-config bundler, fast and beginner-friendly.  
4️⃣ Rollup – Focuses on tree-shaking and smaller bundle sizes, often used for libraries.  
5️⃣ ESBuild – Extremely fast bundler, written in Go, used in tools like Vite.

**How a Bundler Works**

1️⃣ **Entry Point**: The main file (e.g., index.js) where bundling starts.  
2️⃣ **Dependency Graph**: It analyzes imports (import or require) and builds a dependency tree.  
3️⃣ **Processing**:

* **Transpilation** (via Babel)
* **Minification** (removes whitespace, renames variables)
* **Tree Shaking** (removes unused code)  
  4️⃣ **Output**: A single or multiple optimized .js files for production.

**why is vite faster then webpack?**

Vite is faster than Webpack mainly because of its modern architecture and efficient build process. Here’s why:

1️⃣ Instant Server Start (No Bundling in Dev Mode)

Webpack bundles the entire project before serving, which can be slow for large projects.

Vite leverages native ES Modules (ESM), serving unbundled JavaScript files on demand.

This means instant startup, even for large projects.

2️⃣ On-Demand Compilation (Hot Module Replacement - HMR)

Webpack recompiles the entire bundle when you change a file, making hot reload slow.

Vite compiles only the modified file using ESBuild, leading to near-instant updates.

3️⃣ ESBuild for Pre-Bundling

Vite uses ESBuild, a Go-based tool, which is 10-100x faster than Webpack (JavaScript-based).

It quickly pre-bundles dependencies (like React, Lodash) for faster execution.

4️⃣ Optimized Production Build with Rollup

While Webpack bundles everything in dev mode and production, Vite only bundles for production using Rollup, which generates optimized tree-shaken output.

5️⃣ Efficient Caching & Code Splitting

Vite intelligently caches modules, so unchanged files don’t need to be reprocessed on reload.

Webpack rebuilds many files even if they haven’t changed, making it slower.

Conclusion

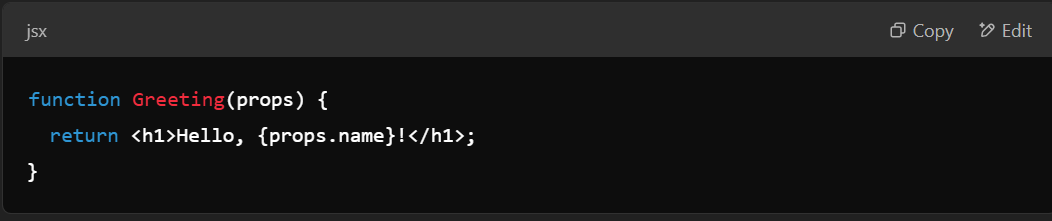
Vite is designed for modern development, focusing on speed, efficiency, and better DX (Developer Experience). For new React projects, Vite is often the better choice over CRA/Webpack due to its faster dev server and better HMR performance. 🚀

**What is component in React?**

A component in React is a reusable, self-contained piece of UI that defines how a part of the interface should appear and behave. Components allow us to break the UI into smaller, manageable pieces.

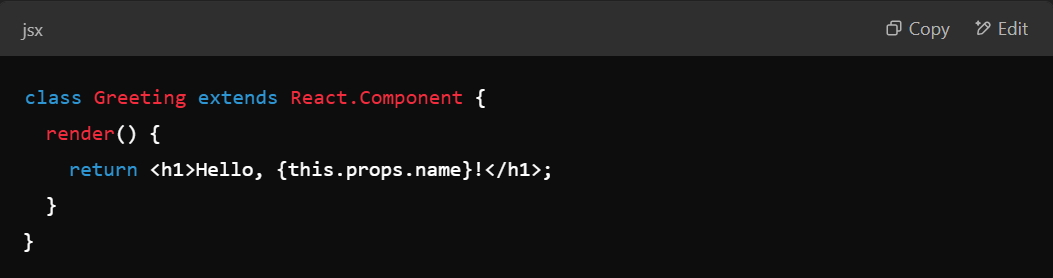
Types of Components in React:

1. Functional Components (Preferred in Modern React)
   * These are simple JavaScript functions that return JSX.
   * They can use React Hooks for state and lifecycle management.



Class Components (Older Approach)

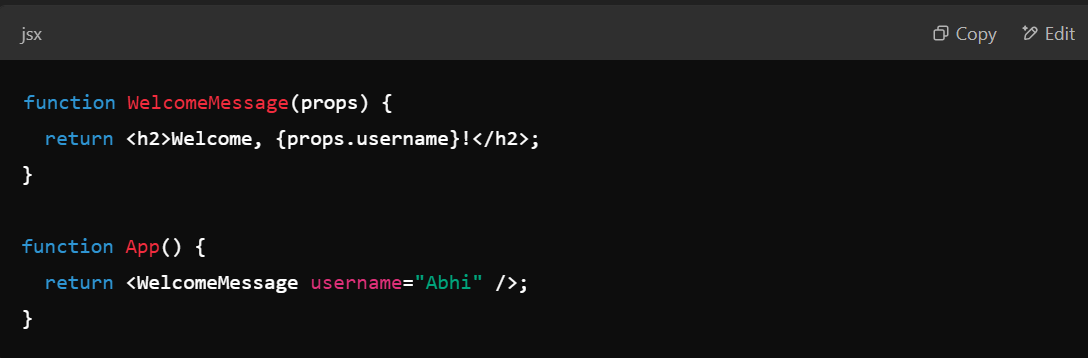
* These are ES6 classes that extend React.Component.
* They have a render() method and support lifecycle methods.
* Example:

****

Key Features of Components:

* Reusable – Components can be reused across the application.
* Composability – Components can be nested inside other components.
* Stateful or Stateless – Components can manage their own state (useState in functional components or this.state in class components).
* Props (Properties) – Components receive data via props to make them dynamic.

Example of a Parent-Child Component Relationship:

****

**How to use Image in react?**

**To use images in react conventionally you can create a folder named assets inside that images.**

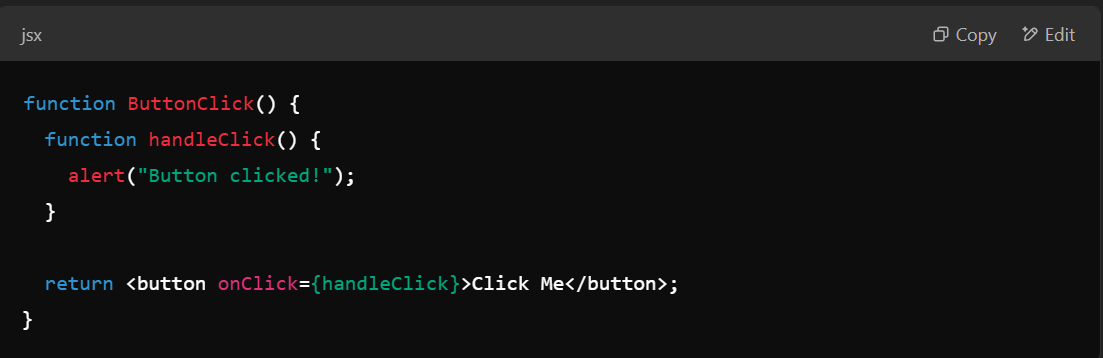
**And then you need to import the file into the component in which you are using it using this syntax –**

**Import image1 from ../assets/images/image1**

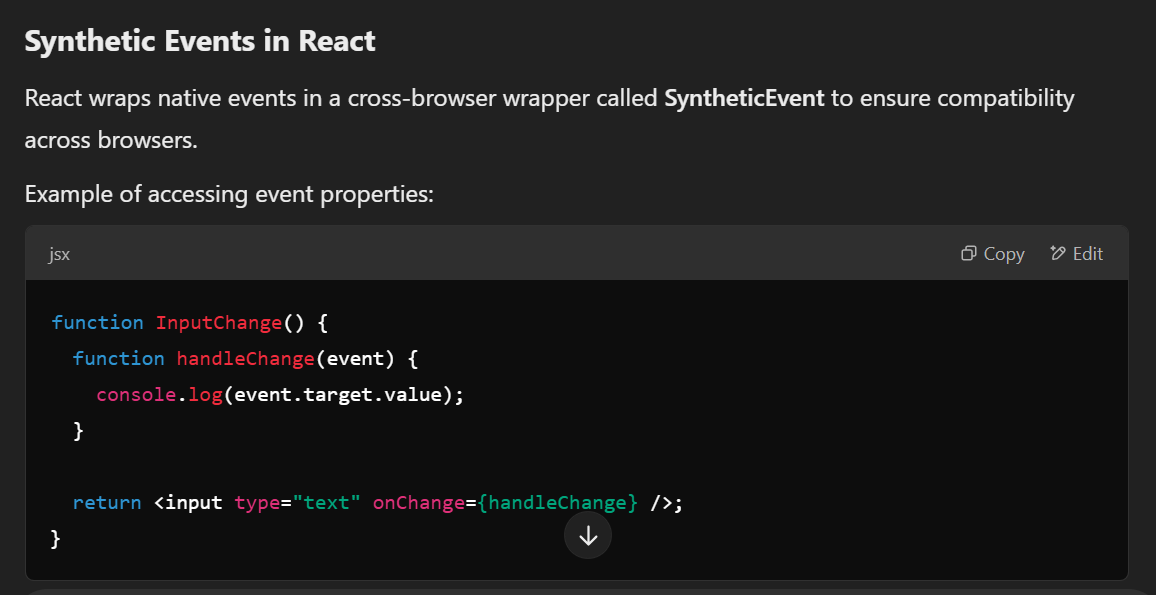
**Note – you cannot give direct path in src attribute you need to import it.**

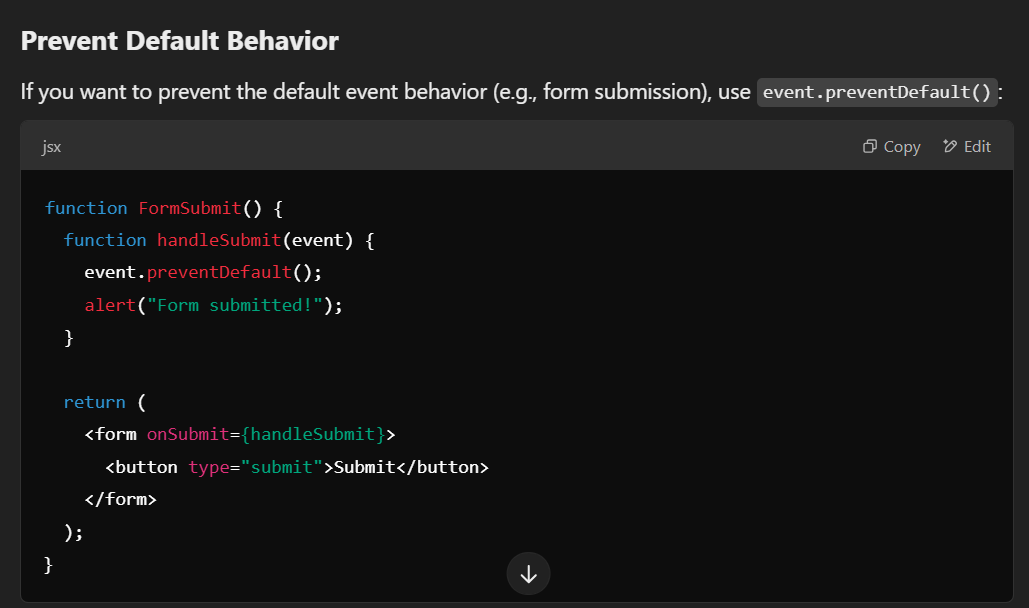
**How to handle events in React?**

**To handle events in react you can directly use attributes like onClick, onHover, onMouseDown, etc. into the jsx elements and provide a function their.**

****

****

****

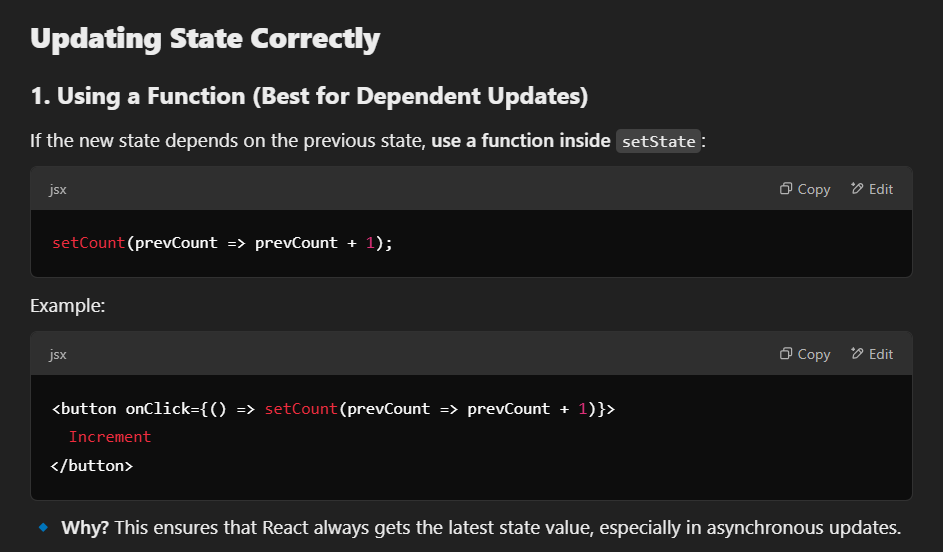
****

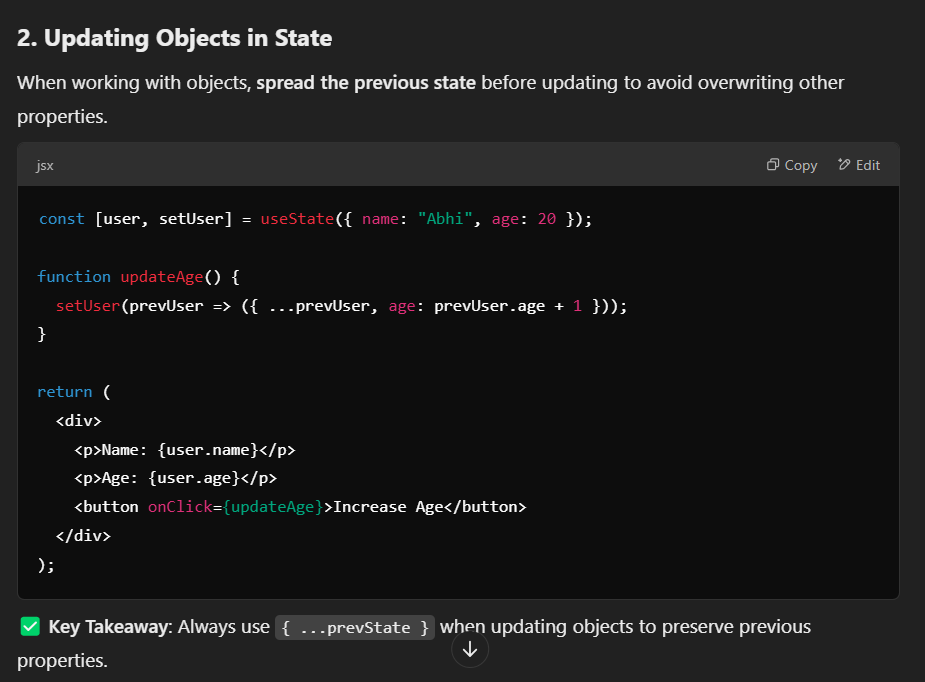
**useState hook in React –**

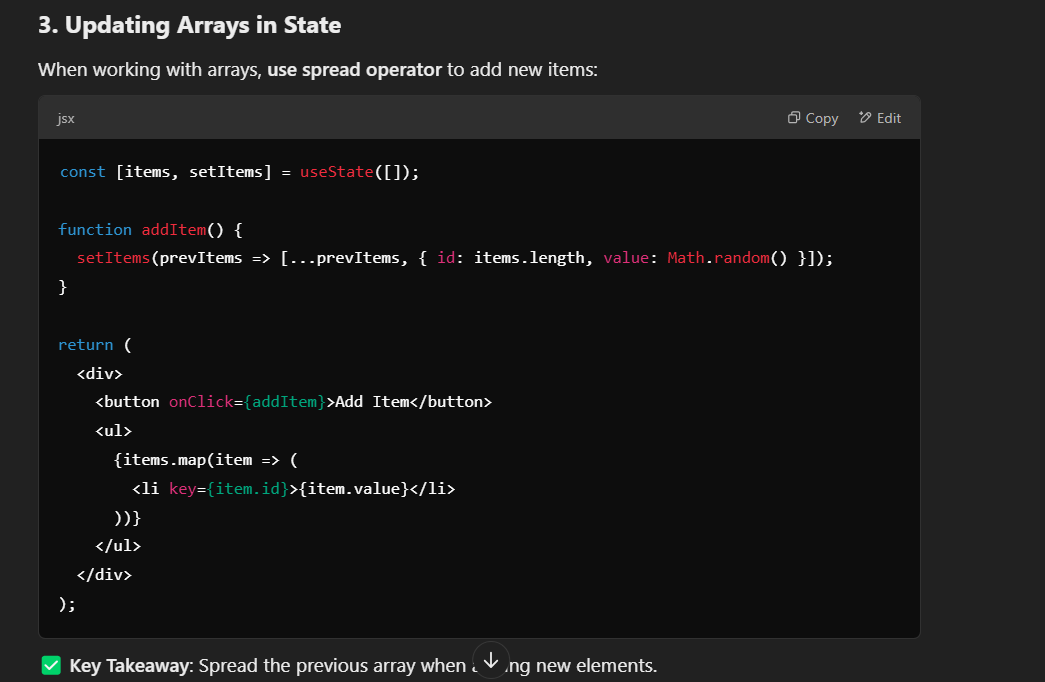
The useState hook is a built-in React Hook that allows functional components to manage state without using class components. It returns an array with two values:

Current state value

Function to update the state









**State in React -**

State in React is an object that holds **dynamic data** and determines how a component behaves and renders. Unlike props, which are **read-only**, state **can be updated** using the useState hook in functional components or this.setState in class components.

**Key Features of State**

✅ **Mutable:** Can be changed using setState or useState.  
✅ **Component-specific:** Each component manages its own state.  
✅ **Triggers Re-rendering:** When state updates, React re-renders the component.  
✅ **Asynchronous:** State updates may not happen immediately.

**State vs Props in React –**

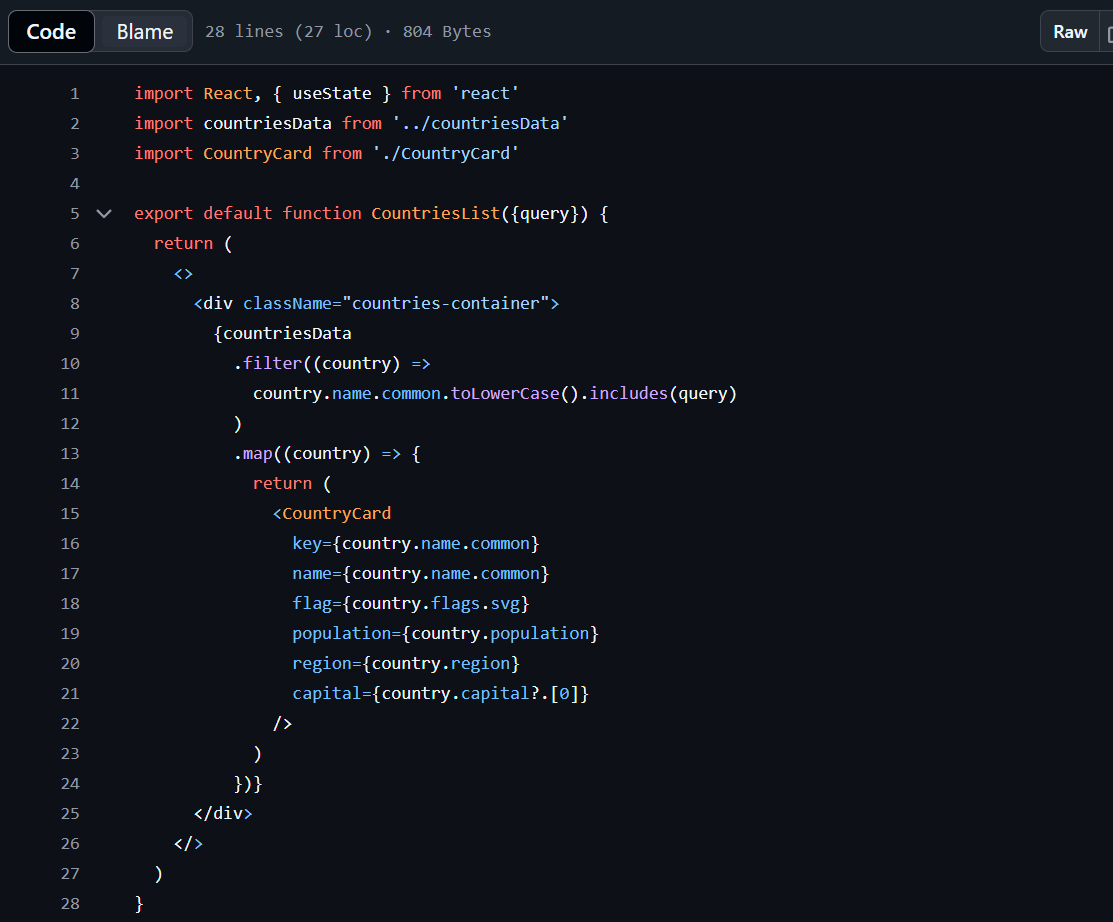
**Difference Between Props and State in React**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Props** | **State** |
| **Definition** | **Props (short for "properties") are used to pass data from a parent component to a child component.** | **State is a local data storage that a component can manage and update itself.** |
| **Mutability** | **Immutable (cannot be changed by the child component).** | **Mutable (can be updated using setState or useState).** |
| **Where It’s Used** | **Passed from parent to child component.** | **Managed within a single component.** |
| **Who Controls It?** | **Controlled by the parent component.** | **Controlled by the component itself.** |
| **Reusability** | **Allows components to be reusable by passing different props.** | **State is specific to the component and doesn't affect others.** |
| **Updates Cause Re-render?** | **Yes, when props change, the component re-renders.** | **Yes, updating state triggers a re-render.** |
| **Example Use Cases** | **Passing data, functions, event handlers to child components.** | **Managing component-level data like form inputs, UI toggles, etc.** |

**How to implement search functionality in react?**

****

****

****

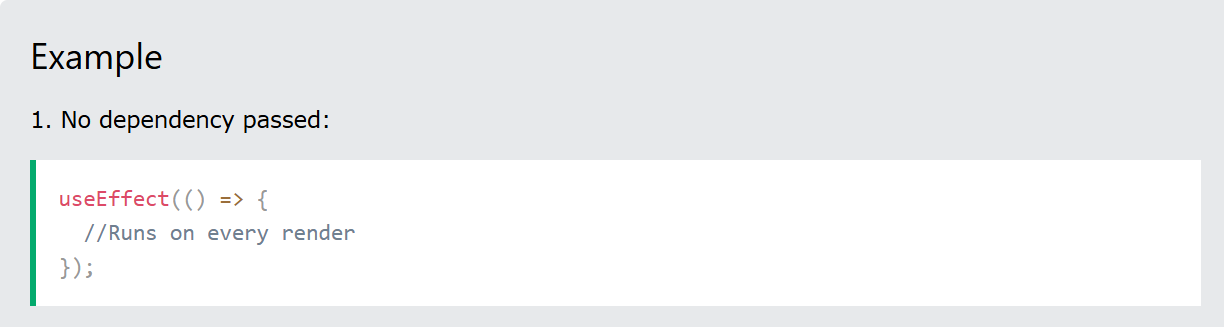
**useEffect Hook in React**

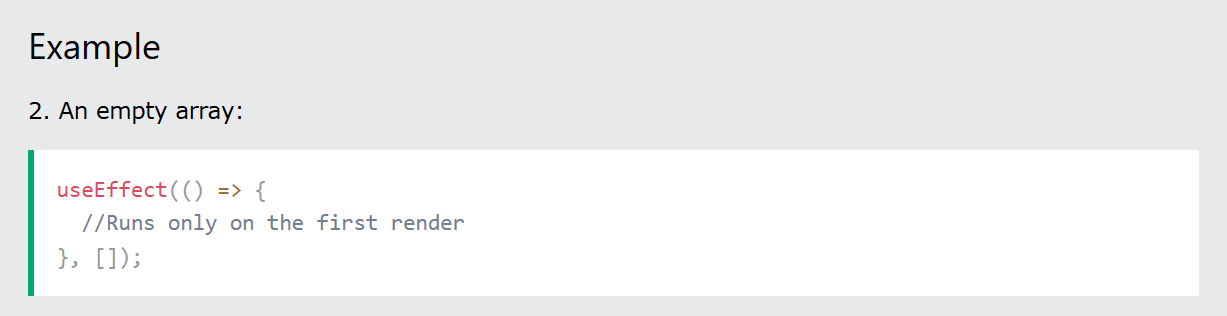
**It gets executed after full jsx of the component is loaded.**

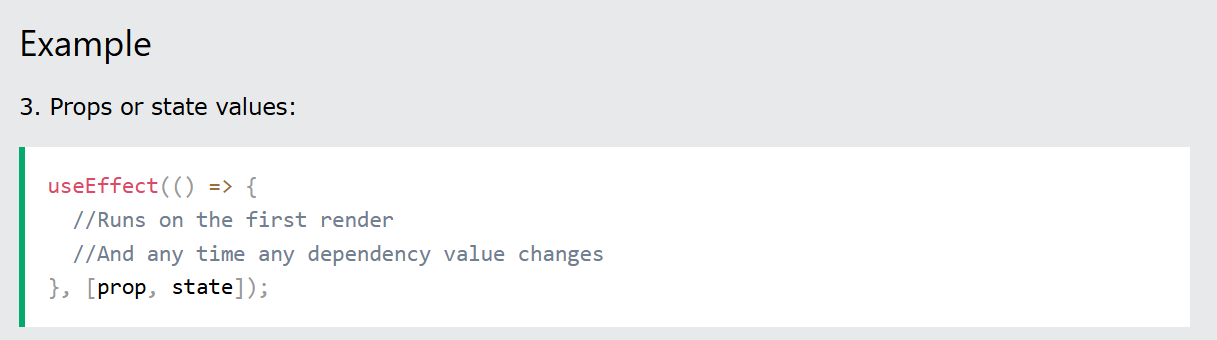
The useEffect Hook allows you to perform side effects in your components.

Some examples of side effects are: fetching data, directly updating the DOM, and timers.

useEffect accepts two arguments. The second argument is optional.







**Effect Cleanup**

**Some effects require cleanup to reduce memory leaks.**

**Timeouts, subscriptions, event listeners, and other effects that are no longer needed should be disposed.**

**We do this by including a return function at the end of the useEffect Hook.**

****

**useLayoutEffect Hook -**

**It is synchronous useEffect means it is going to block the rendering of this component until it executes the code inside it.**

**useRef hook (persist values without re-rendering) –**

Used with forms usually

Returns an object with **current** **property**.

It doesn’t cause re-render on updating the value

Why don’t we use a normal variable instead of useRef?

So basically difference between normal variable and useRef is that on re-rendering the component useRef stores(remembers) the value that it had while a normal variable looses it values and gets back to the initial value.

For eg. – count variable that has been created using useRef will remember its count even when the component re-renders while normal variable doesn’t.

**Syntax –**

const myRef = useRef(0) //it just returns 1 object.

useRef is commonly used for referencing directly dom node for html elements

Syntax –

const myRef = useRef();

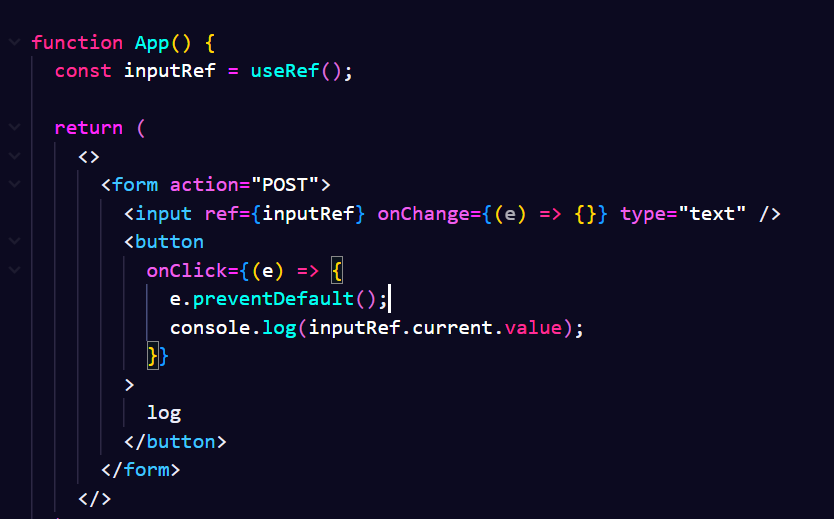
<button ref={myRef} >Hello</button>

This will make mrRef.current to reference to this button directly and then you can do –

myRef.current.style.backgrounColor = ‘red’;

but it commonly used to access the value property of input fields.

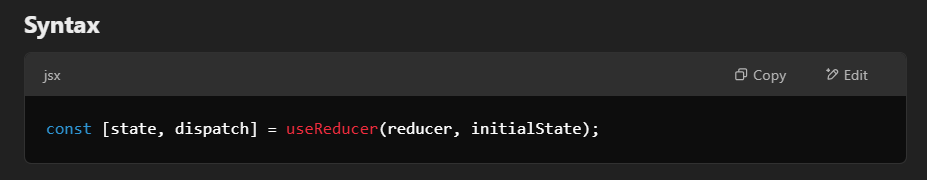
It saves re-renders while updating values in input tags.

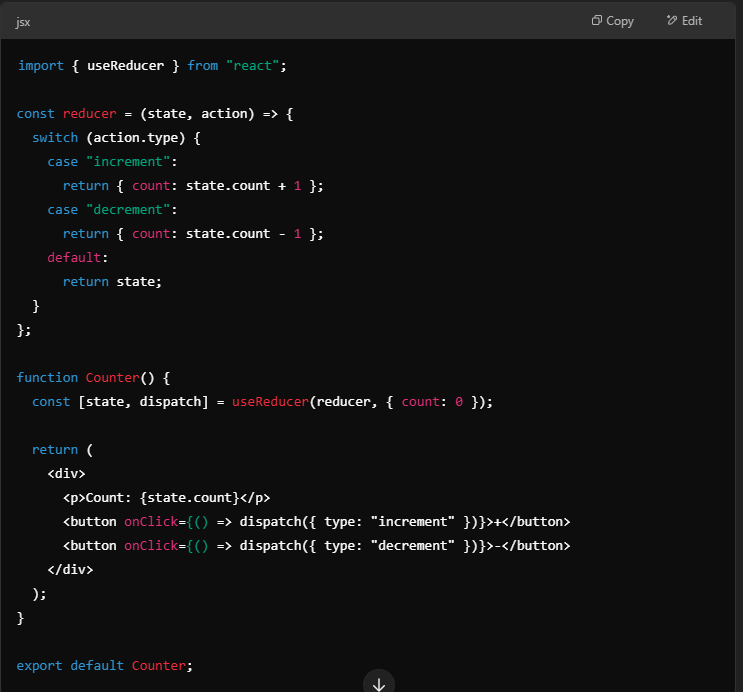


**commonly we pass null inside the useRef(null) in typescript.**

**useReducer hook in React –**

The useReducer hook is an alternative to useState, suitable for complex state logic.

****

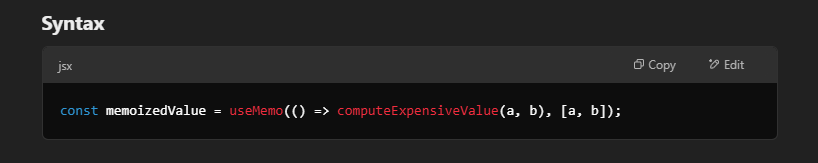
****

**Explanation**

* **useReducer manages state updates via actions instead of direct updates.**
* **It’s useful when state logic is complex (like multiple dependent variables).**

**useMemo hook –**

**The useMemo hook optimizes performance by memoizing (caching) expensive calculations.**

****

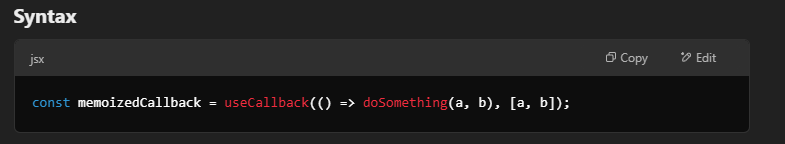
****

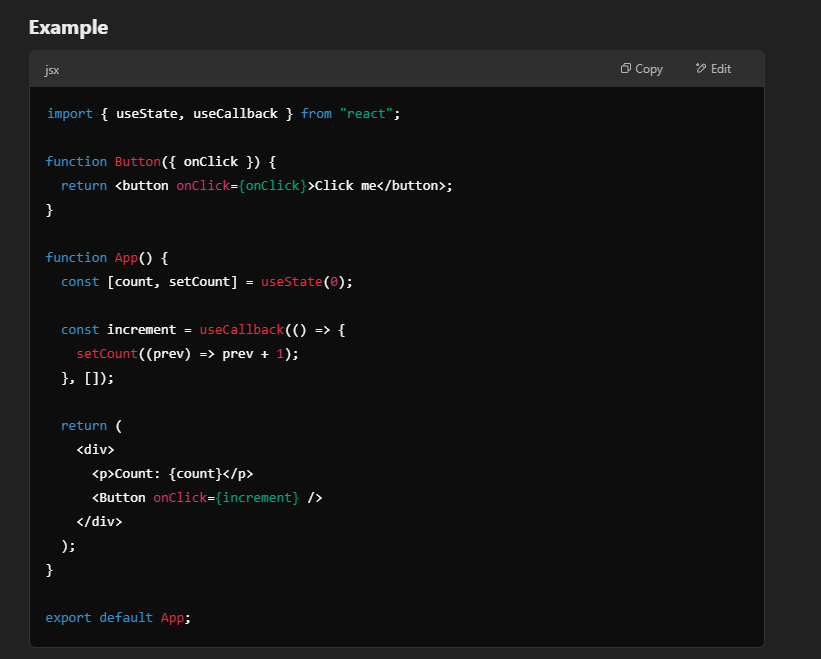
**Explanation**

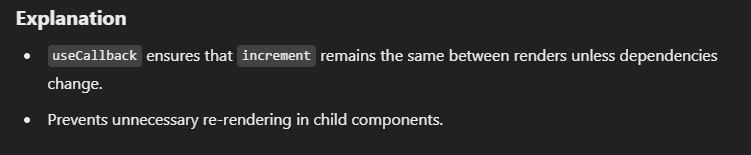
* **useMemo caches the result of number \* 2 and recalculates it only when number changes.**
* **Prevents unnecessary recalculations and improves performance.**

**useCallback Hook –**

The useCallback hook memoizes a function so that it doesn't get recreated on every render.







**For eg.** –

You are using a function that is passed to the child component and on re-render the functions gets recreated so the child component will also get re-rendered coz we have passed the function to child. To save this re-render you can pass the function to useCallback so the function doesn’t gets created on every re-render.

**useContext and createContext hook in react –**

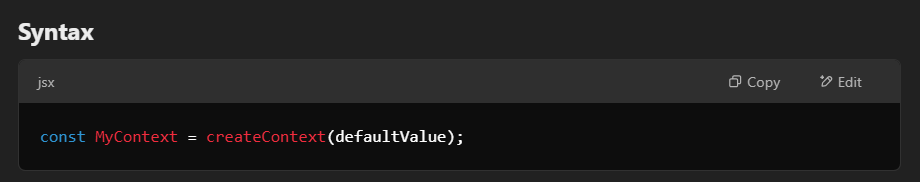
React provides **Context API** to manage and share global state without prop drilling. The two main hooks used in this are:

1. createContext(): Creates a context object to store values globally.
2. useContext(): Allows consuming the context value in any component.

1. createContext() Hook

The createContext() hook is used to create a new context object that holds shared data.

Syntax

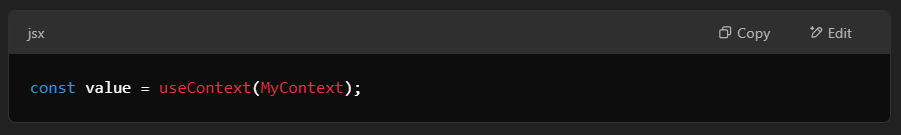


defaultValue (optional) is used when a component tries to access the context without a provider.

**2. useContext() Hook**

The useContext() hook allows **child components** to access the context **without manually passing props**.

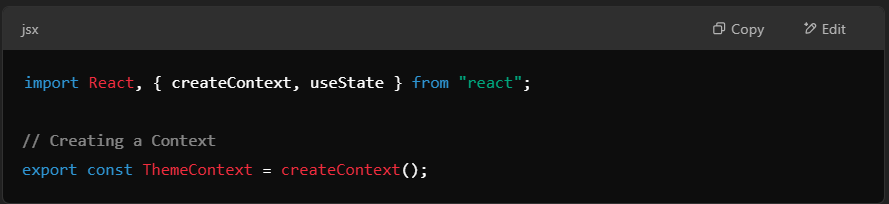
**Syntax**



This hook takes a **context object** (created using createContext) and returns its **current value**.

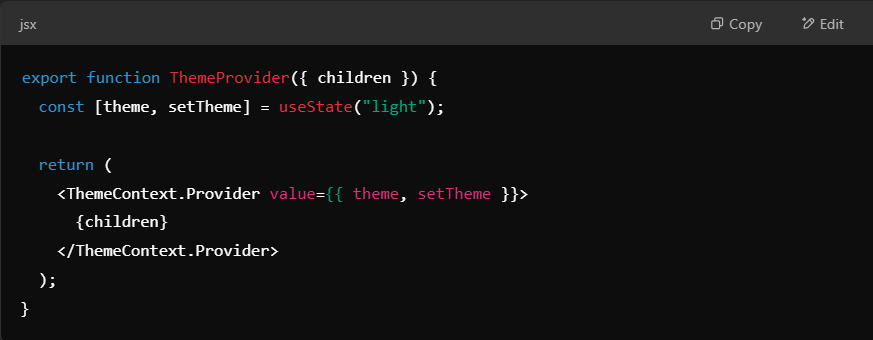
**Complete Example of useContext & createContext**

**Step 1: Create a Context**



**Step 2: Create a Provider Component**

The provider component **wraps child components** and provides them with a shared state.



 ThemeContext.Provider wraps children and provides { theme, setTheme } as **context value**.

 theme stores the current theme (light or dark).

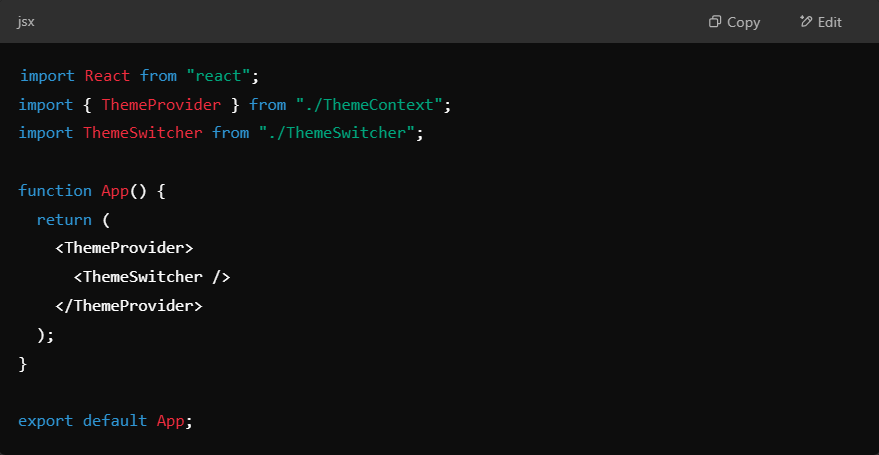
 setTheme updates the theme.

**Step 3: Use useContext() in a Child Component**



* useContext(ThemeContext) gets the **current theme** and setTheme function.
* The button toggles between **light** and **dark** mode.

Step 4: Wrap Your App with the Provider



**ThemeProvider wraps the entire app**, so all components inside can access the theme.

**Summary**

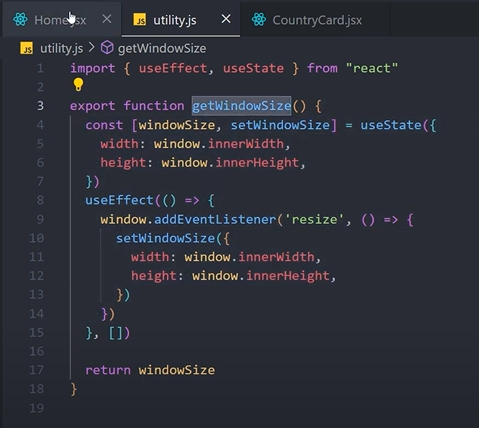
✅ createContext() – Creates a **context object**.  
✅ useContext() – Fetches the **current value** from the context.  
✅ Context Provider – Wraps components to **share state** globally.  
✅ **Avoids prop drilling** – No need to pass props manually through multiple components.

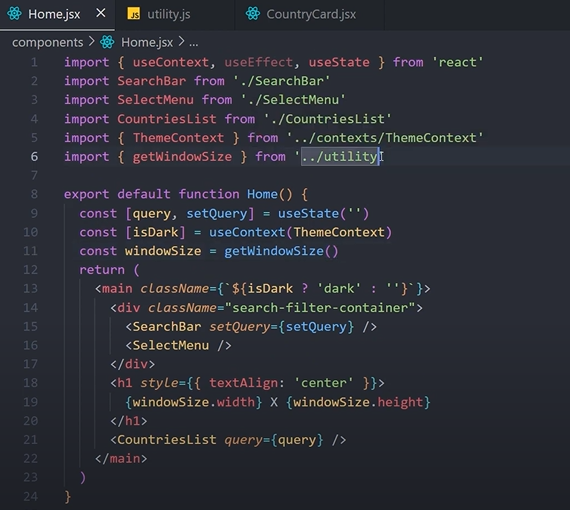
🔹 **Best Use Cases**

* Theme Switching
* Authentication (User Login State)
* Language Localization (Multi-language Support)
* Global State Management (Alternative to Redux)

**CUSTOM REACT HOOKS**

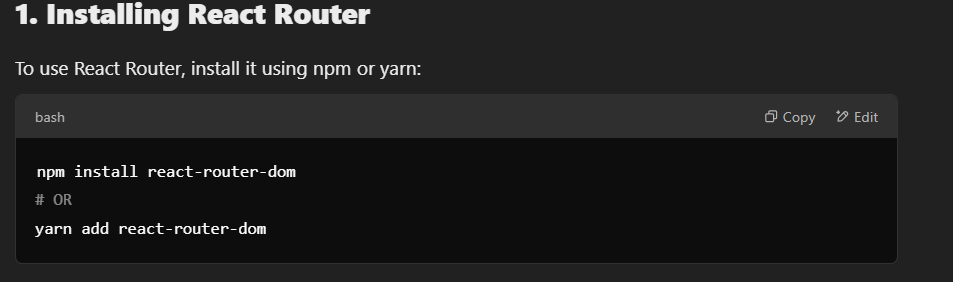
To create a custom hook in React you need to create a js file with the hook name inside a hook folder (conventional way) and make a function inside it and can use it as a custom hook anywhere.





**React Router**

**React Router is a client-side routing library that allows navigation between different pages without reloading the page. It helps create single-page applications (SPA) by changing the URL and rendering components dynamically.**

****

**2. Basic Setup of React Router**

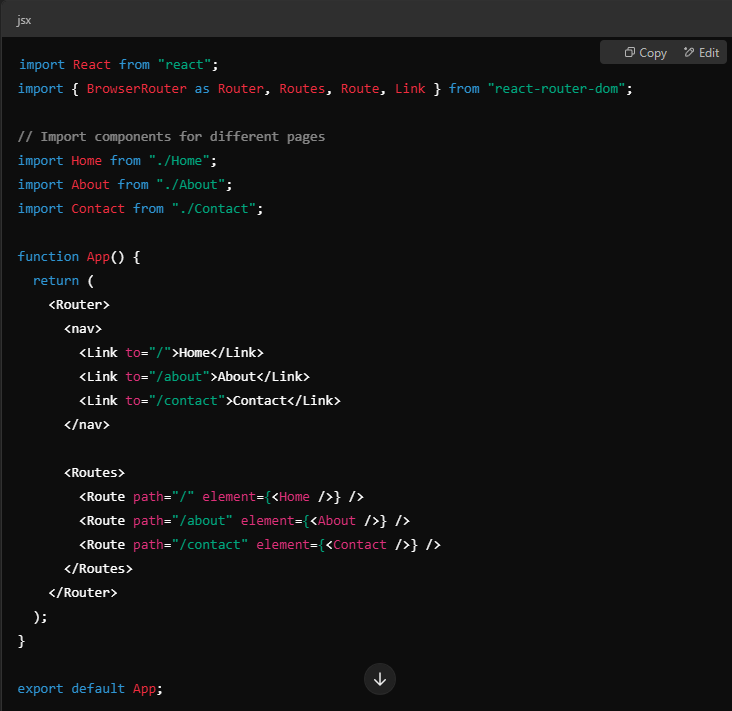
**Step 1: Import Required Components**

**React Router provides the following components:**

* **BrowserRouter: Wraps the application to enable routing.**
* **Routes: Groups all route definitions.**
* **Route: Defines the mapping between a path (path) and a component (element).**
* **Link & NavLink: Used for navigation without refreshing the page.**

**Step 2: Setting Up Routes**

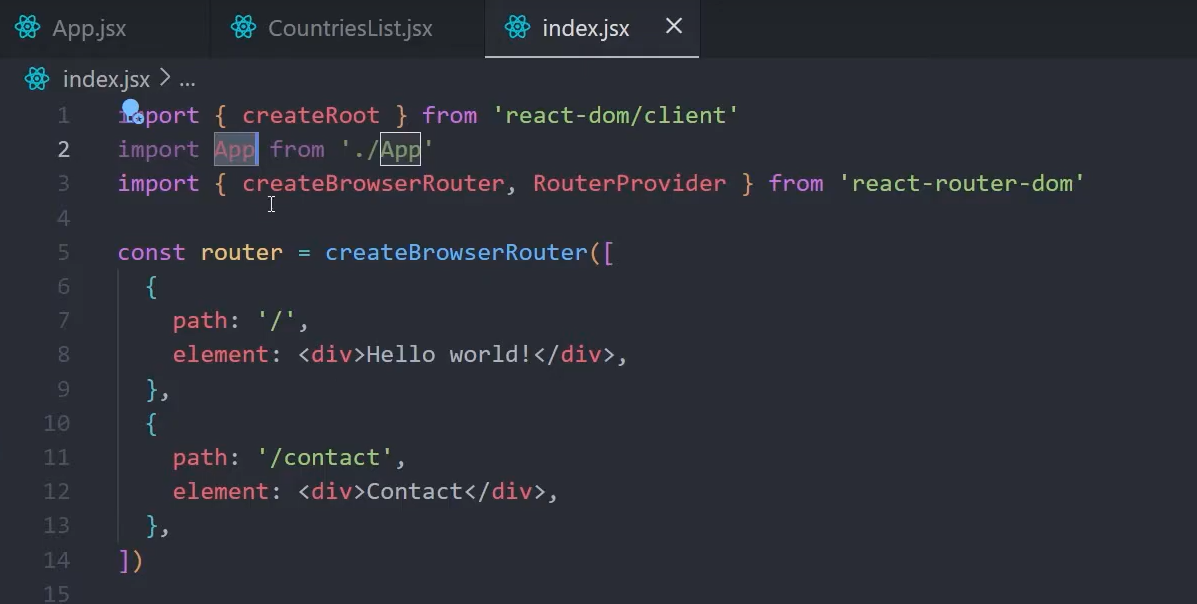
**Create a file App.js and define routes inside the <Routes> component:**

****

**Explanation:**

* <BrowserRouter> enables client-side routing.
* <Routes> is used to wrap all <Route> components.
* <Route path="/" element={<Home />} /> means when the URL is /, it renders <Home />.
* <Link to="/about">About</Link> creates a clickable link to navigate without refreshing.

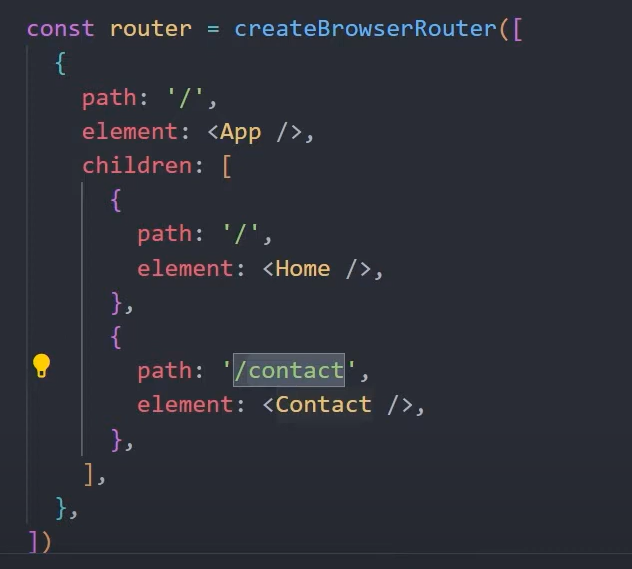
**Other way of using router**

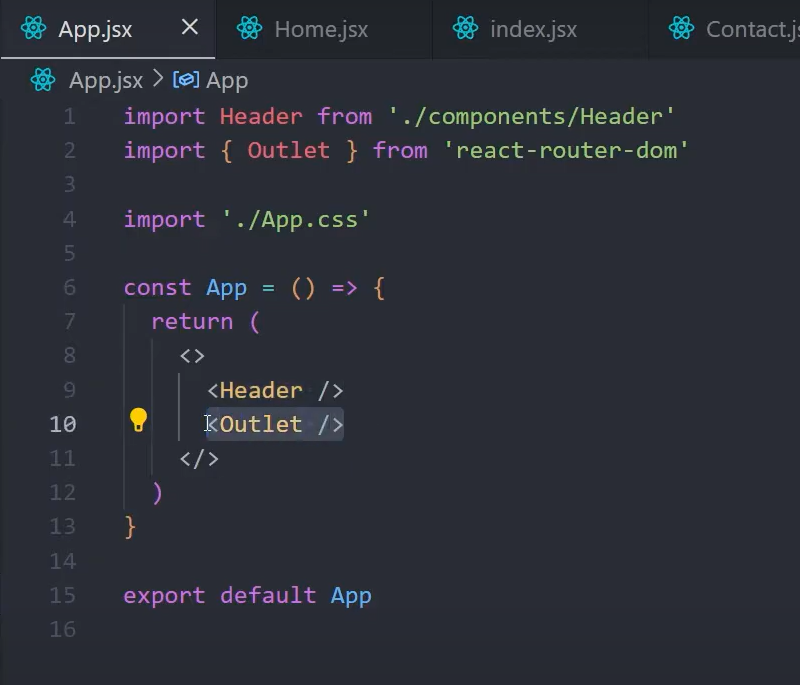
****

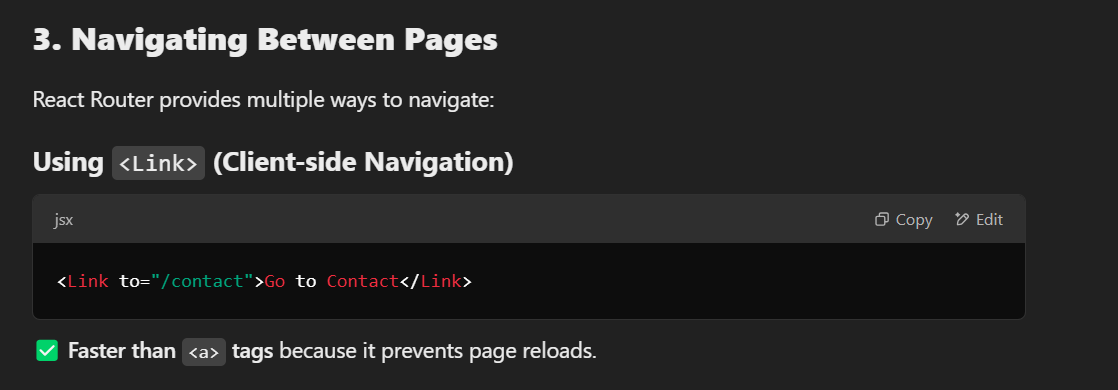
****

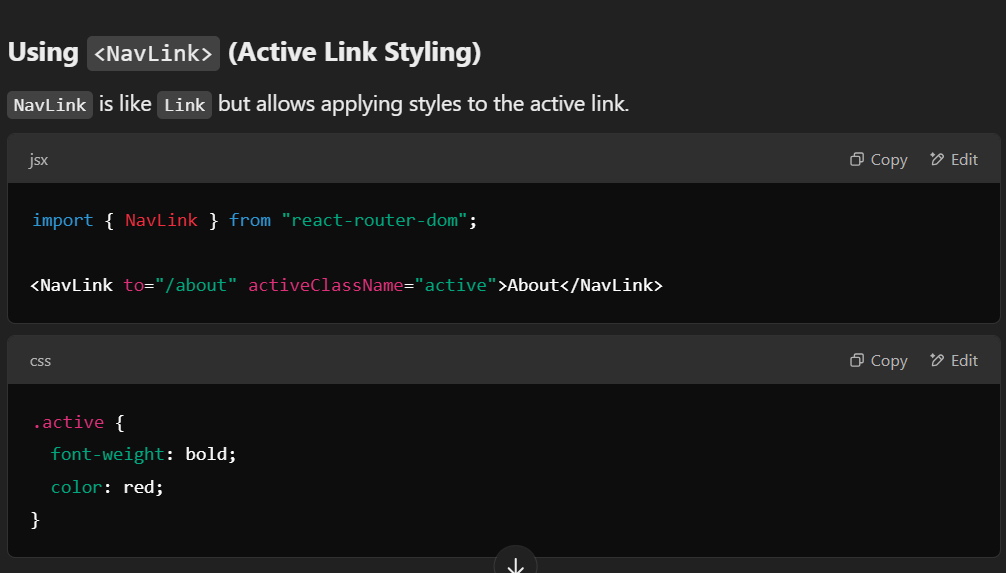
**Using outlet –**

**Main.jsx**

****

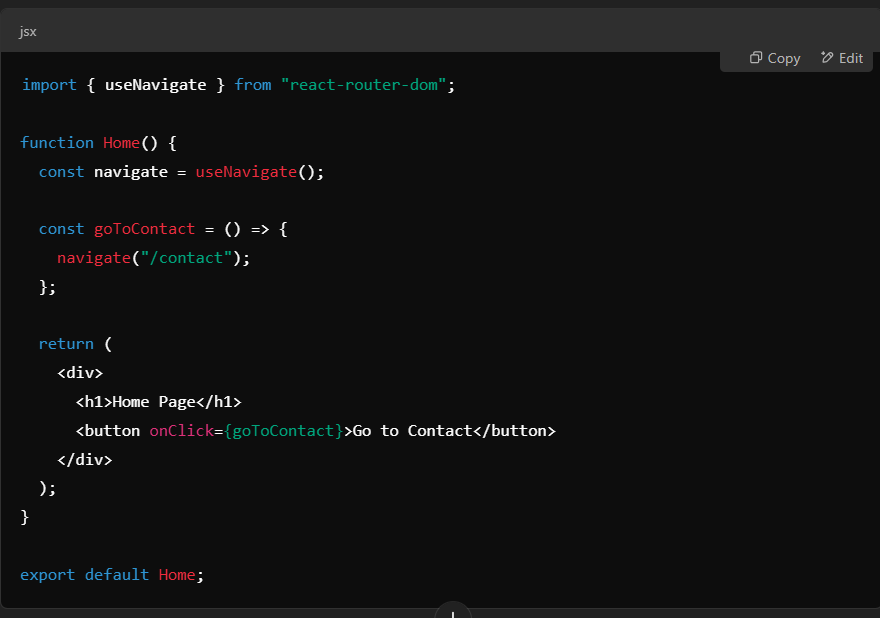
****

****

****

**Using useNavigate() Hook (Programmatic Navigation)**

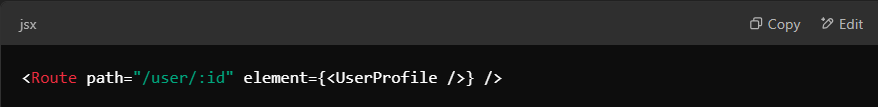
**If you need to navigate on a button click or after an event, use useNavigate().**

****

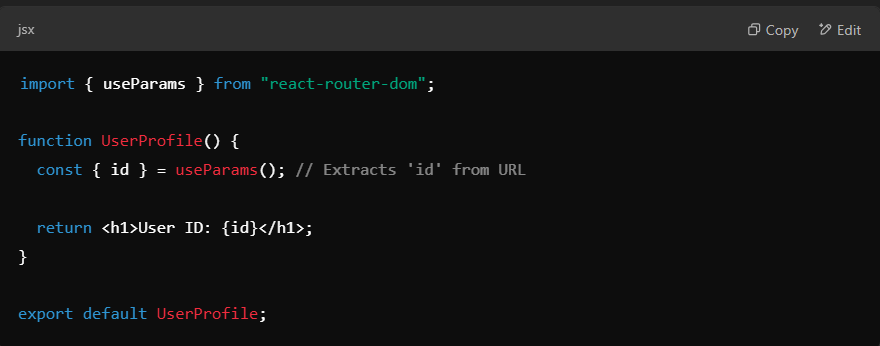
**4. Dynamic Routing with URL Parameters**

**To create dynamic routes (e.g., /users/:id), use URL parameters.**

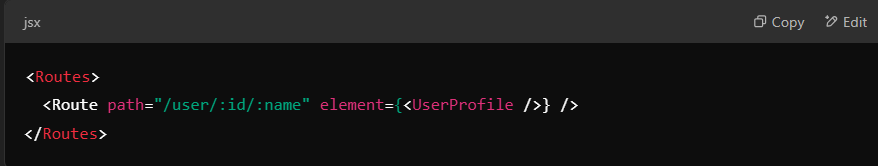
**Step 1: Define a Dynamic Route**

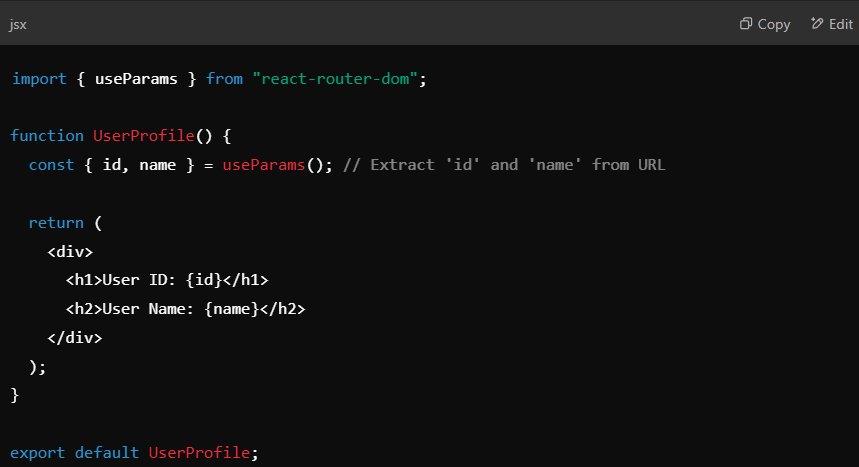
****

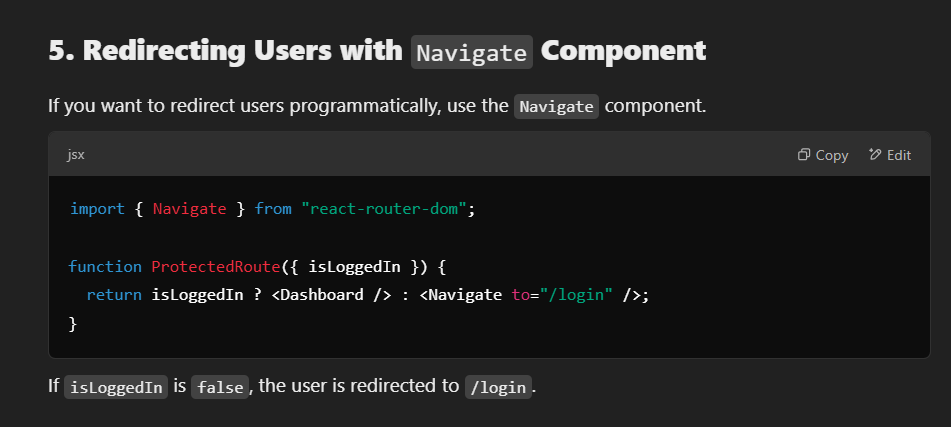
**Step 2: Access Parameters Using useParams()**

****

**🛠️ Example URL: http://localhost:3000/user/42 → Displays User ID: 42**

****

****

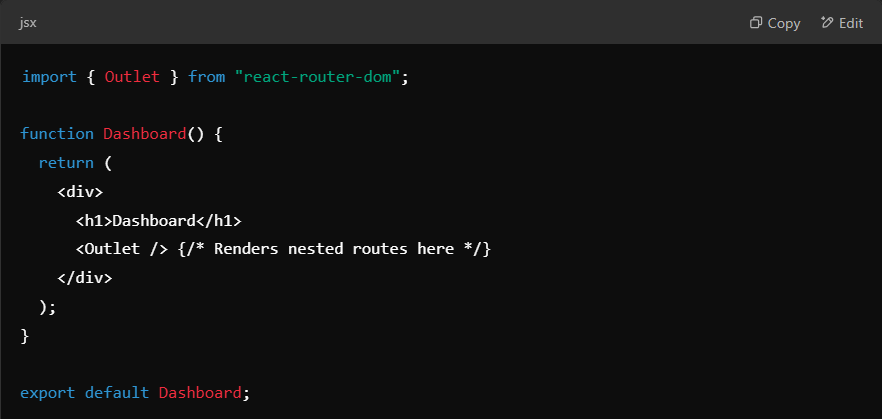
****

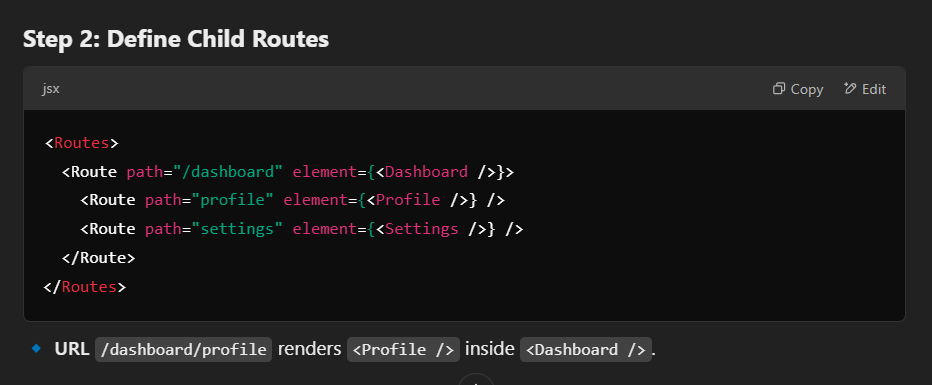
****

**7. Nested Routes (Child Routes)**

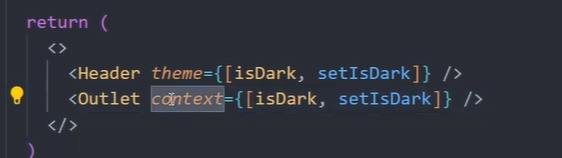
**For routes inside another component (e.g., /dashboard/profile), define nested routes.**

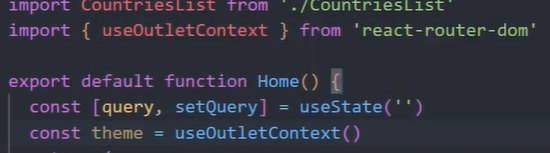
**Step 1: Parent Route (Dashboard)**

****

****

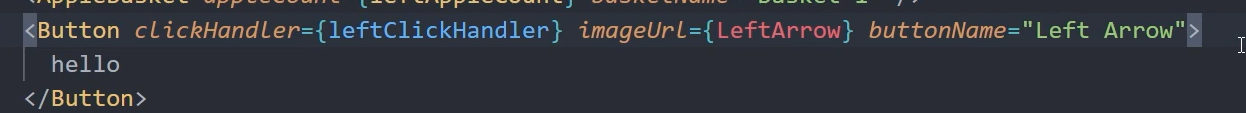
**Context and useOutletContext in React router –**

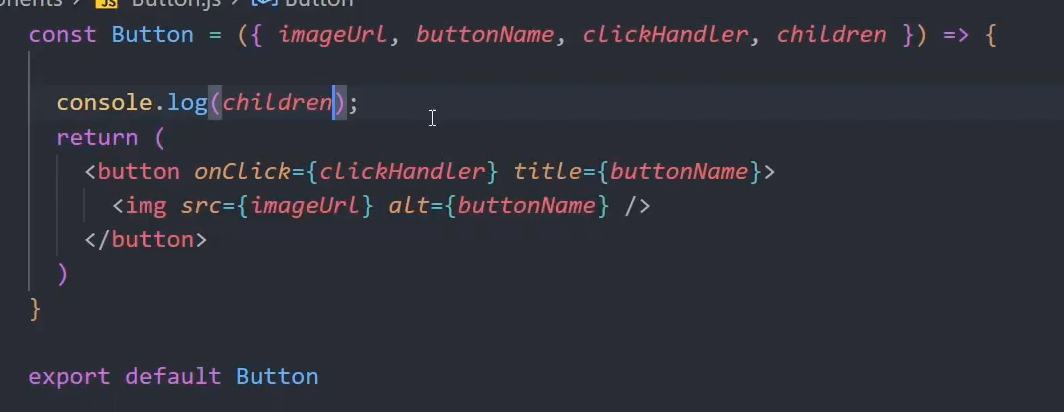
****

****

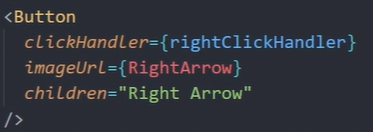
**Children props in ReactJs**

**If you use a component with a closing tag and opening tag instead of single tag then you get all the things inside it as children and can receive it in that component as props.children or directly {children}.**

****

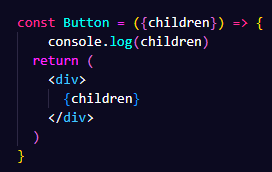
****

**Children can also be passed like this –**

****

**If you pass multiple items then it will give you a array of the items passed.**

**Note – children props don’t render automatically you need to render them inside the component by yourself –**

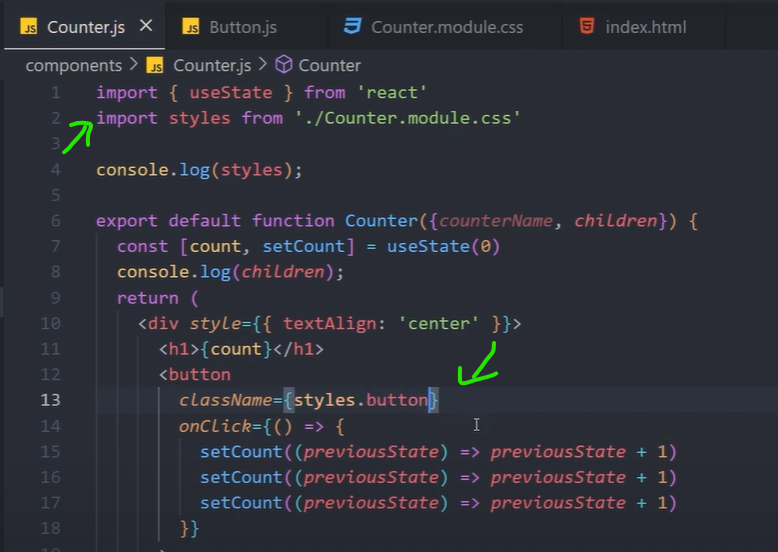
****

**Modular Css in React –**

**If you want to scope your css for a component then you can use modular css. For doing so you just need to –**

**Rename the css file name with filename.module.css instead of filename.css**

**Then you need to import and use css using this Syntax –**

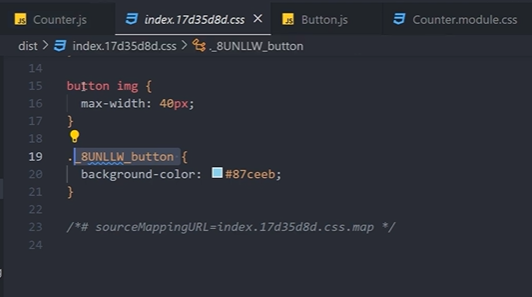
****

**To give multiple classes –**

****

**How it works?**

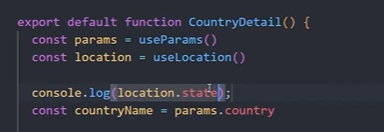
**Behind the scenes the compiler changes the class name with some modifier ahead it –**

****

**Passing data to different pages in ReactJs –**

**You can add the data you want to send as an attribute to the link tag and then on the page you can use it using useLocation hook from react router. –**

****

****

**Local Storage in ReactJs –**

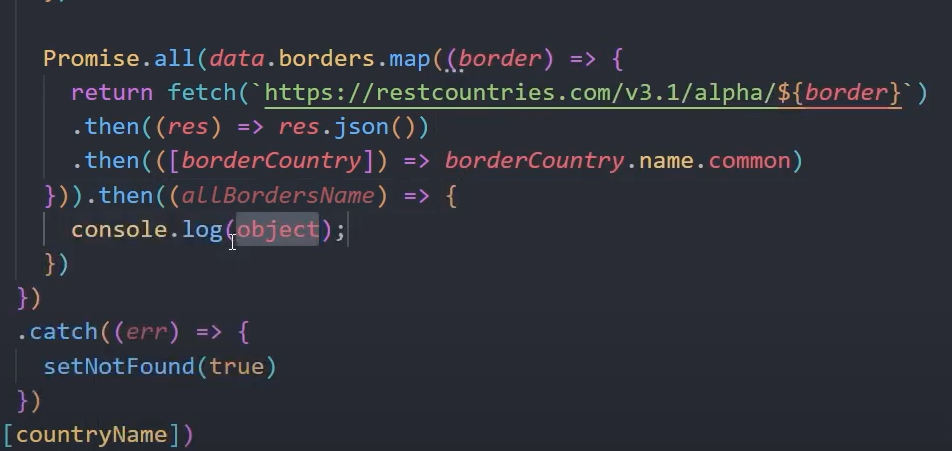
****

**Lifting up state in ReactJS**

**Tips**

**You can use history.back() function to implement back button directly**

**You can use promise.all to resolve many promises at same time.**

****

**Working with forms in React**

To extract data from forms you can use a class called FormData which needs reference to the from (can be provided by e.target) and then we can loop the objects .entries property to get the form data (it returns data in array form so you can directly destructure it and use) pairs or you can use the .value property to just get the value.

Note - The input fields in form must have name attribute.

****