React.js

Date - 15/03/2025

ABOUT

- Upcoming Developer at Western Union

- Technology Intern at Driptech
- Former Intern at Sunsoft Technologies

- Worked on 10+ Freelance Projects
- Received Multiple Funding and internship Offers from Companies for Personal Projects
- 160+ Stars on Github

- Computer Vision Developer at Trident Labs
- Mentor, Former Vice-President, and Former Finance Secretary at CSI VIT Pune
- Former Website and Broadcasting Secretary at V-EDC

AGENDA

TOPIC
INTRODUCTION TO REACT.JS
REACT ARCHITECTURE, JSX & COMPONENTS
PROPS, STATE, EVENTS & LIFECYCLE
🚀 BREAK (10 MIN)
RENDERING, LISTS & FORMS
REACT ROUTER & NAVIGATION
🚀 BREAK (10 MIN)
ADVANCED CONCEPTS (REDUX, CONTEXT API, OPTIMIZATION)
COMMON INTERVIEW QUESTIONS AND ANSWERS
Q&A

INTRODUCTION TO REACT.JS

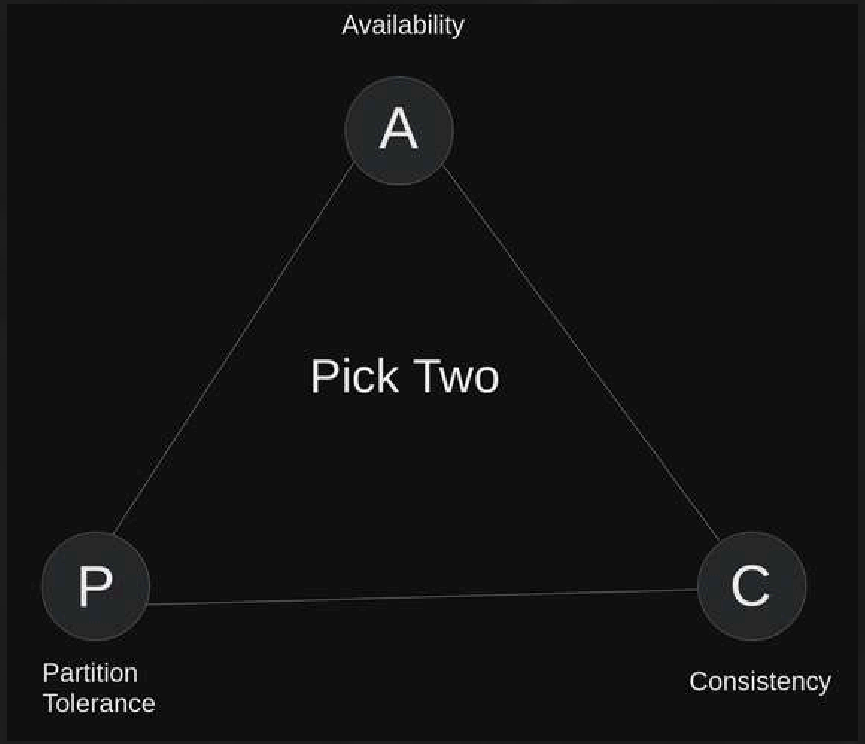
TOPIC
INTRODUCTION TO REACT.JS
REACT ARCHITECTURE, JSX & COMPONENTS
PROPS, STATE, EVENTS & LIFECYCLE
🚀 BREAK (10 MIN)
RENDERING, LISTS & FORMS
REACT ROUTER & NAVIGATION
🚀 BREAK (10 MIN)
ADVANCED CONCEPTS (REDUX, CONTEXT API, OPTIMIZATION)
COMMON INTERVIEW QUESTIONS AND ANSWERS
Q&A

WHAT IS REACT.JS?

- A **JavaScript library** for building user interfaces
- Component-based, declarative, and efficient
- Maintained by Meta (Facebook)
- Fast rendering with **Virtual DOM**
- Reusable components for efficient development
- Great for Single Page Applications (SPAs)

INDUSTRY DEMAND

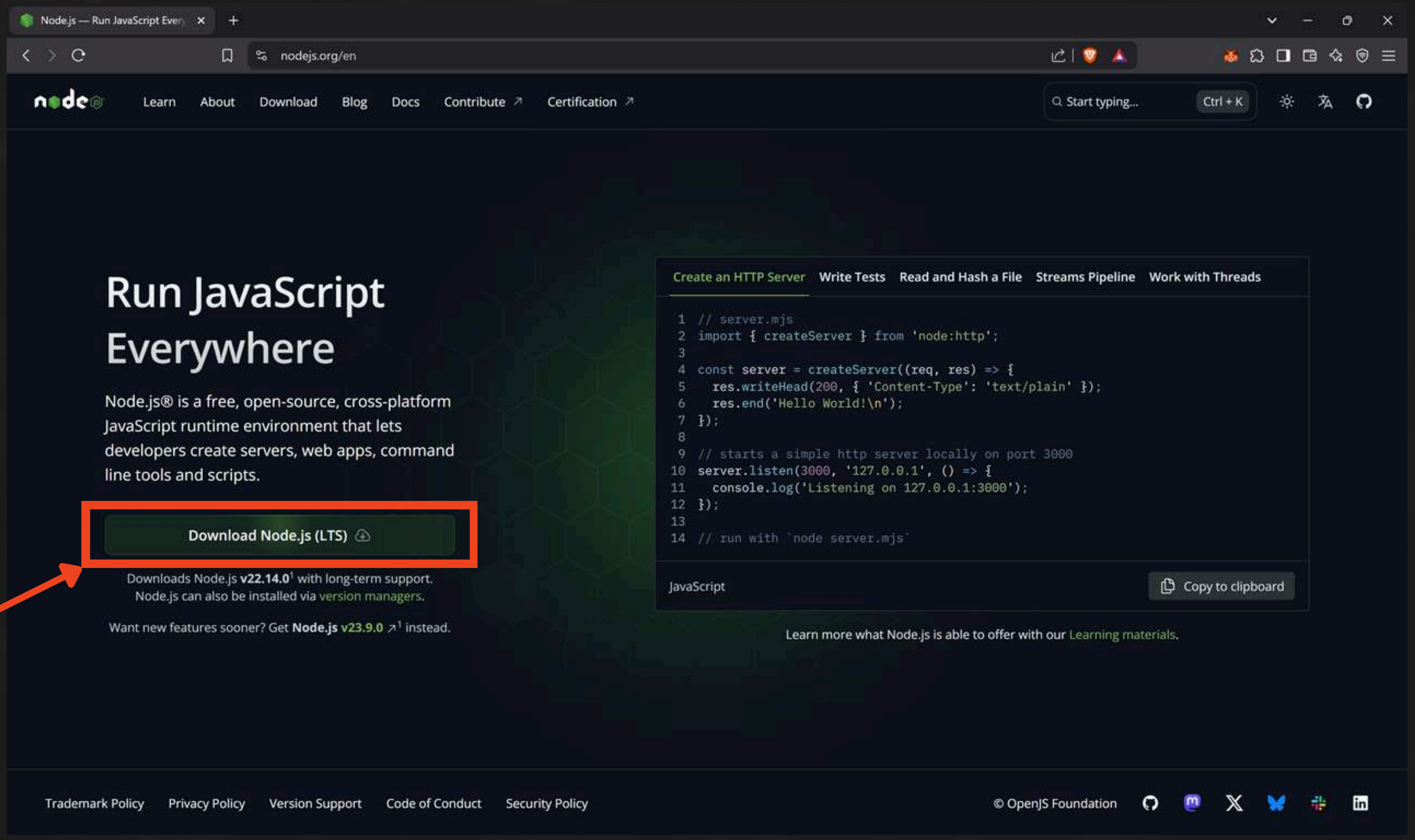
Feature	React.js 🚀	Angular 🏢
Usage	Startups, fintech, product-based companies (Meta, Netflix, PayPal, OpenAI, Amazon, JP Morgan)	Enterprise apps, banking (Microsoft, IBM, Accenture, HDFC)
Learning Curve	Easier, flexible, lightweight	Steeper, opinionated
Performance	Faster due to Virtual DOM	Heavy due to real DOM
Community & Jobs	Huge, largest NPM package ecosystem	Strong but smaller compared to React



SETTING UP REACT.JS

Step 1: Install Node.js & npm

Download from: https://nodejs.org/



Run JavaScript Everywhere

Node.js® is a free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts.

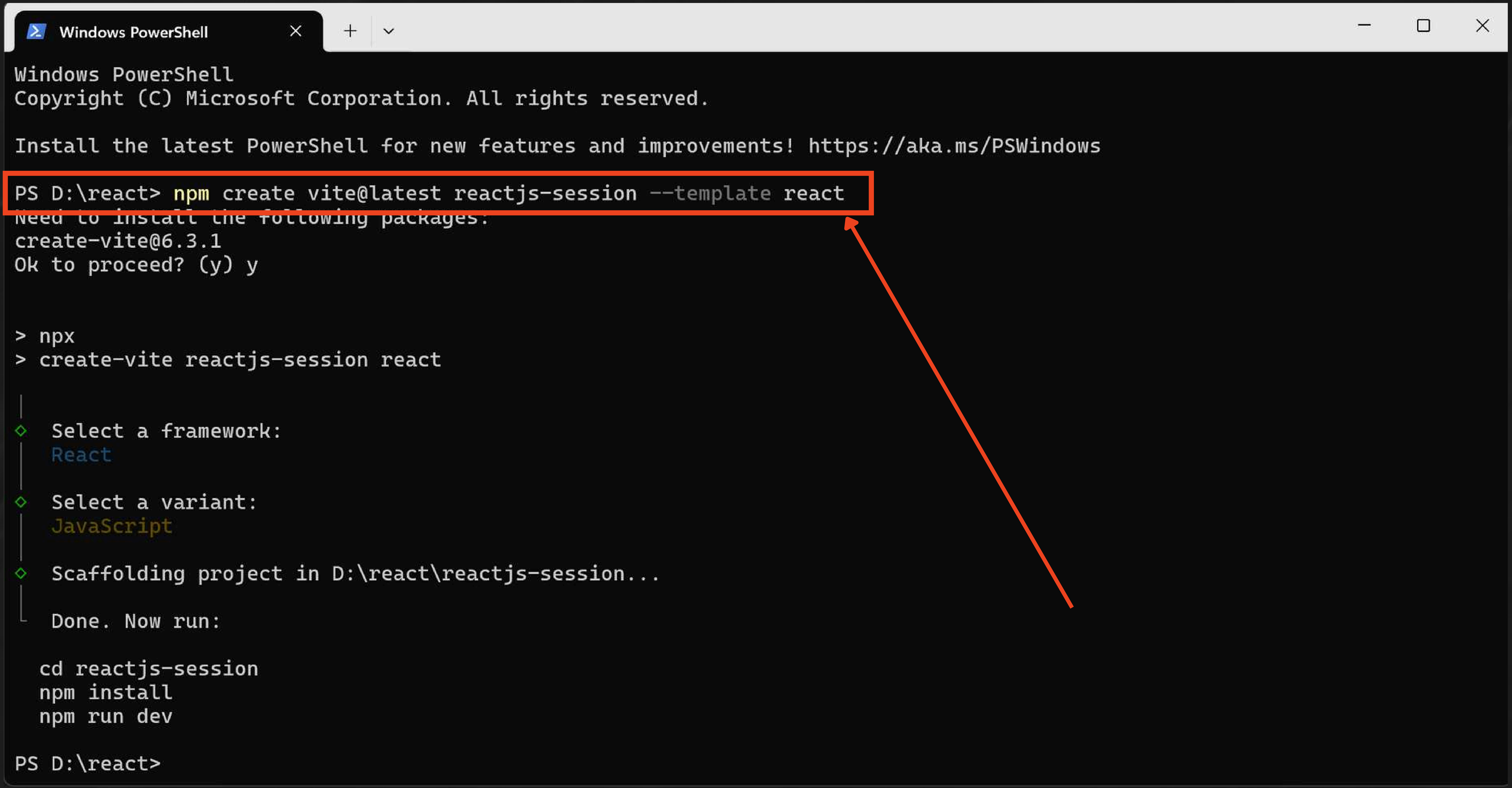
Downloads Node.js **v22.14.0**[1] with long-term support.
Node.js can also be installed via version managers.

Want new features sooner? Get **Node.js** v23.9.0 ↗[1] instead.

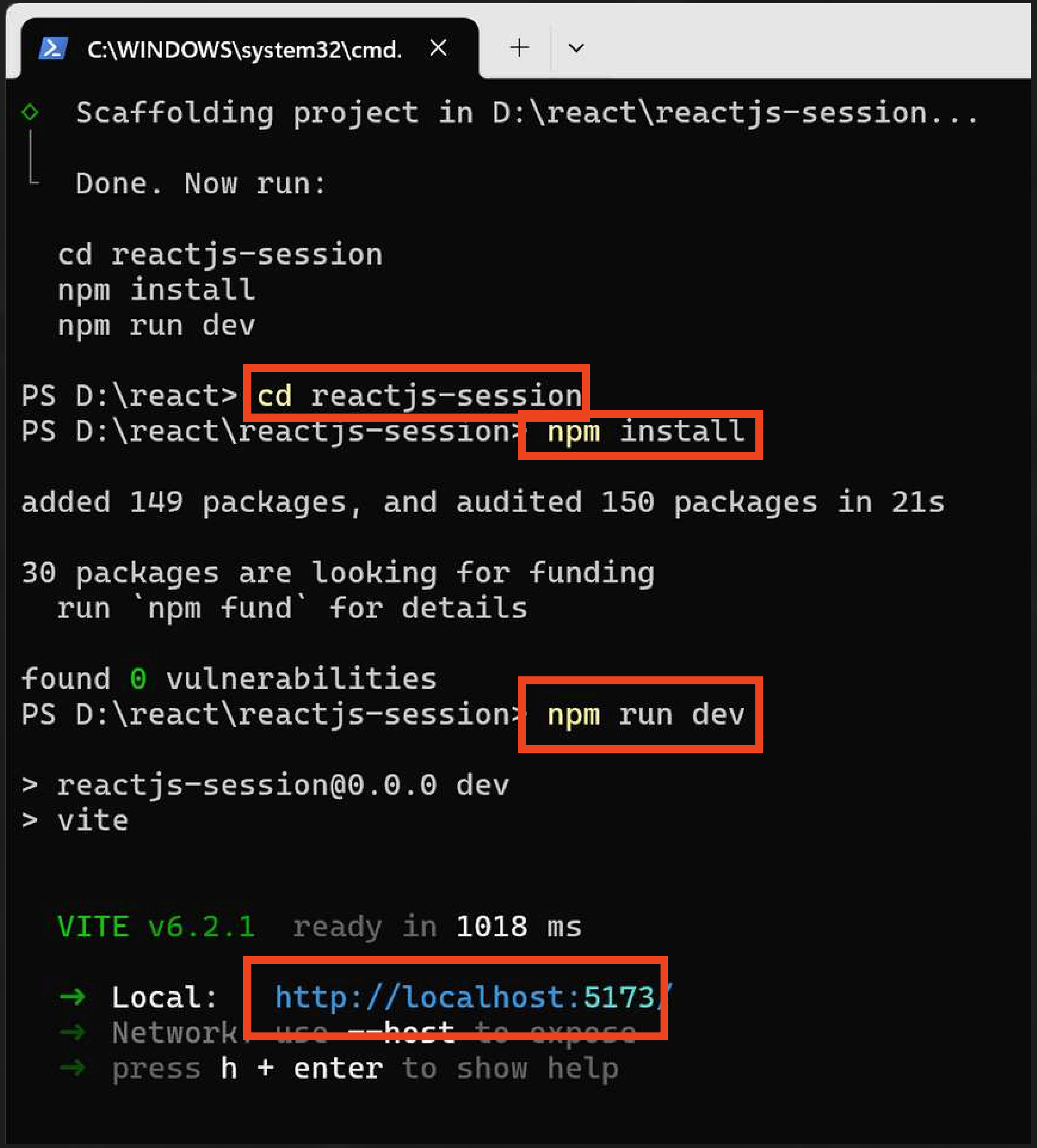
```javascript
// server.mjs
import { createServer } from 'node:http';

const server = createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello World!\n');
});

// starts a simple http server locally on port 3000
server.listen(3000, '127.0.0.1', () => {
  console.log('Listening on 127.0.0.1:3000');
});

// run with `node server.mjs`
```

Learn more what Node.js is able to offer with our Learning materials.

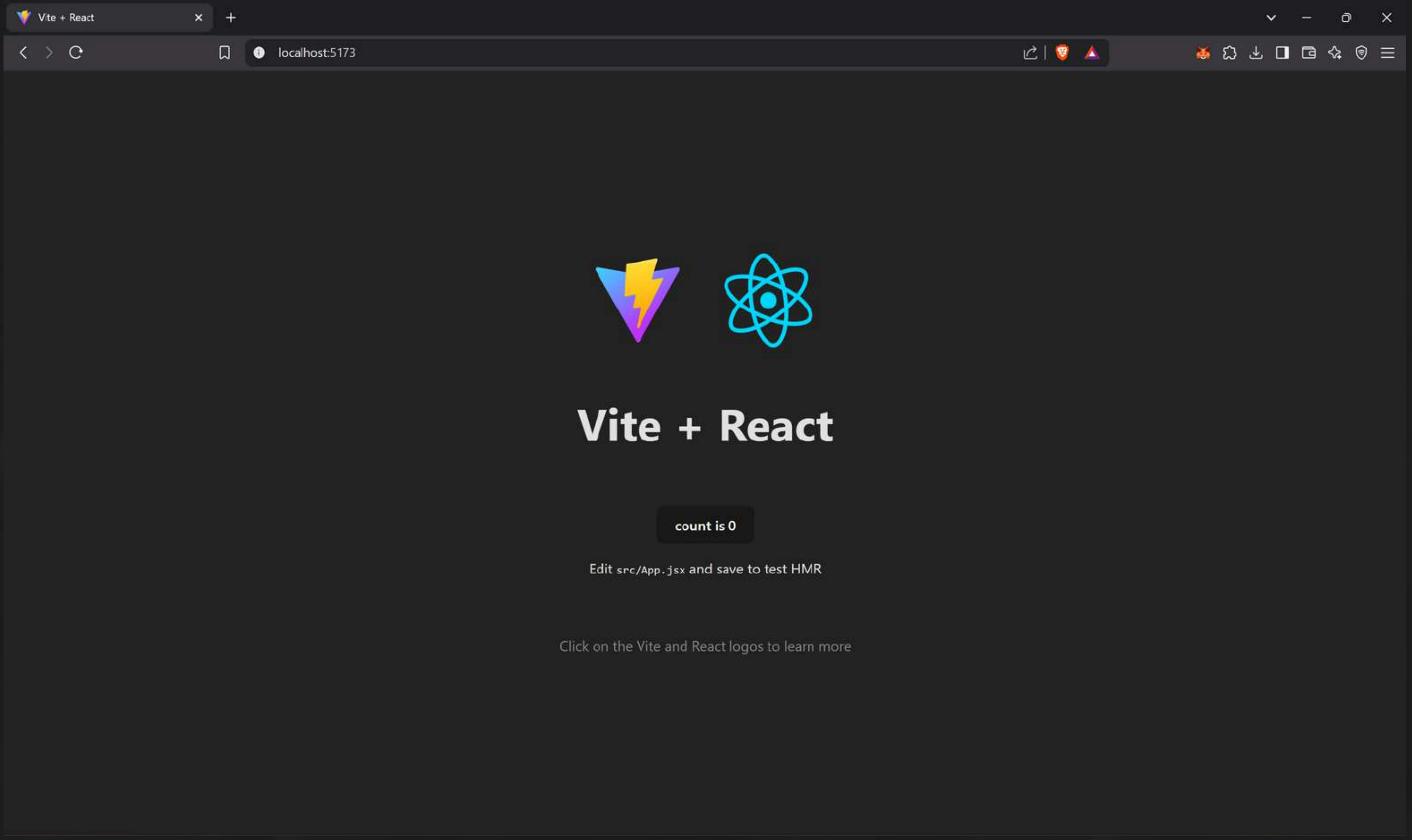
Step 2: Create a React App with vite



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\react> npm create vite@latest reactjs-session --template react
Need to install the following packages:
create-vite@6.3.1
Ok to proceed? (y) y


> npx
> create-vite reactjs-session react

│
◇  Select a framework:
│  React
│
◇  Select a variant:
│  JavaScript
│
◇  Scaffolding project in D:\react\reactjs-session...
│
└  Done. Now run:

  cd reactjs-session
  npm install
  npm run dev

PS D:\react>
```



```
C:\WINDOWS\system32\cmd.

◇  Scaffolding project in D:\react\reactjs-session...
│
└  Done. Now run:

  cd reactjs-session
  npm install
  npm run dev

PS D:\react> cd reactjs-session
PS D:\react\reactjs-session> npm install

added 149 packages, and audited 150 packages in 21s

30 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\react\reactjs-session> npm run dev

> reactjs-session@0.0.0 dev
> vite


  VITE v6.2.1  ready in 1018 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

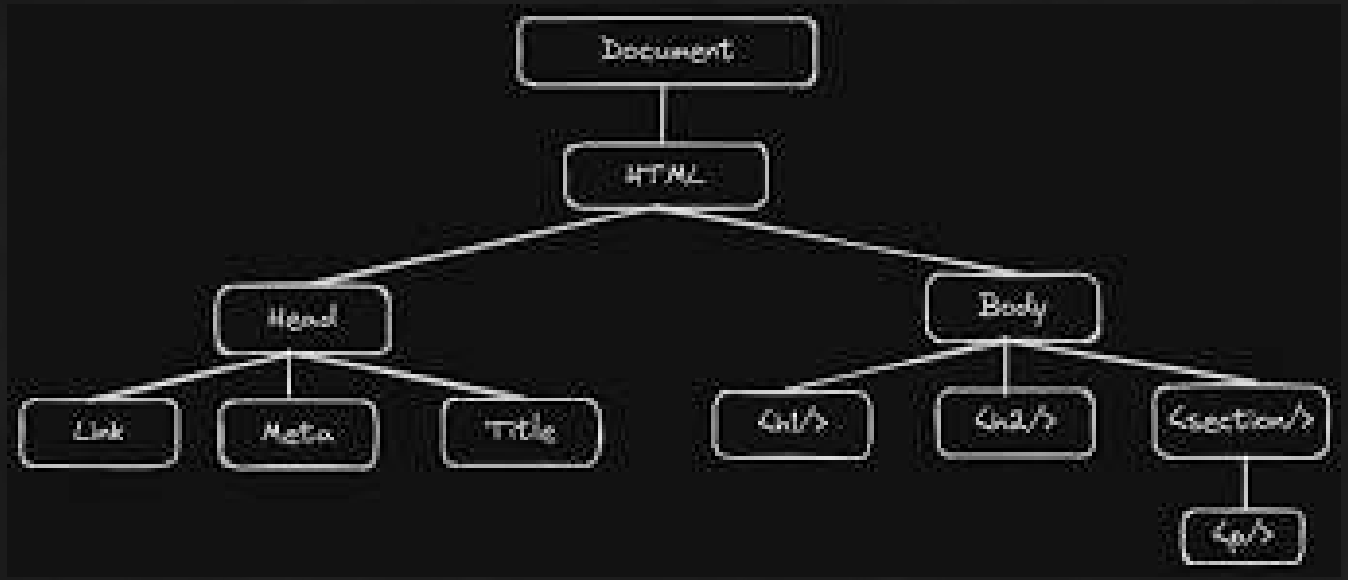

Vite + React
localhost:5173
Vite + React
count is 0
Edit src/App.jsx and save to test HMR
Click on the Vite and React logos to learn more

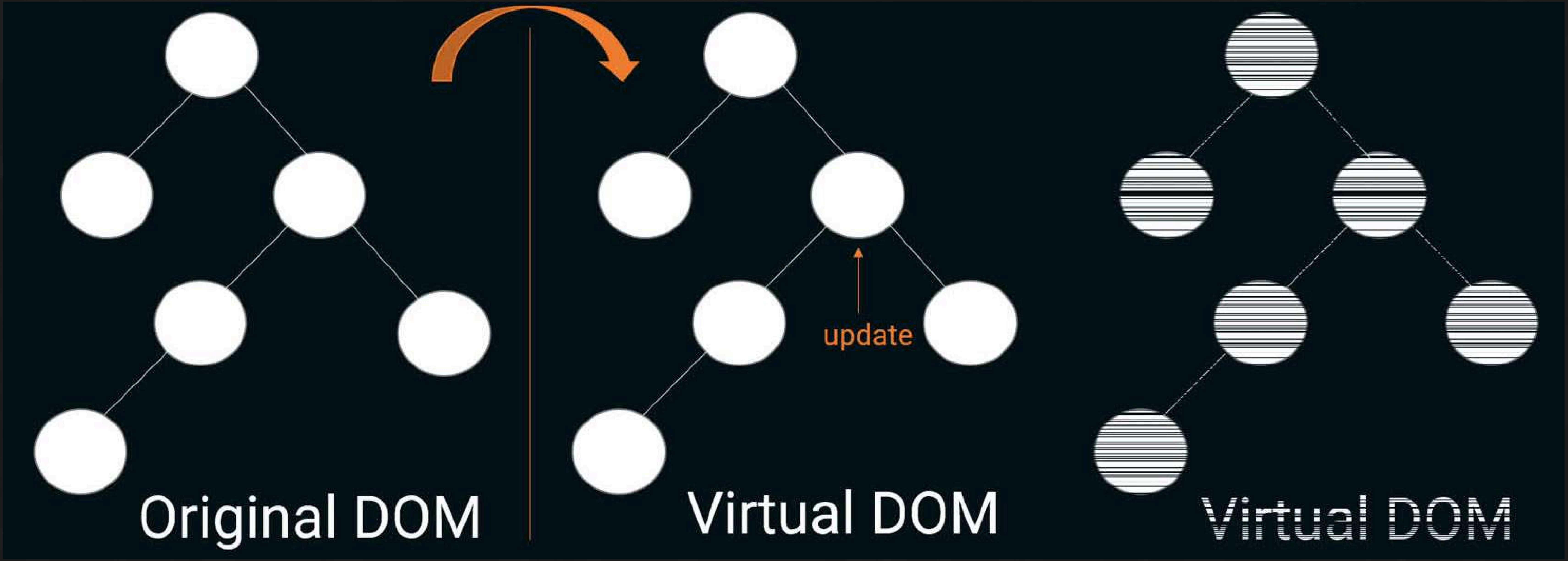
REACT ARCHITECTURE, JSX & COMPONENTS

TOPIC
INTRODUCTION TO REACT.JS
REACT ARCHITECTURE, JSX & COMPONENTS
PROPS, STATE, EVENTS & LIFECYCLE
🚀 BREAK (10 MIN)
RENDERING, LISTS & FORMS
REACT ROUTER & NAVIGATION
🚀 BREAK (10 MIN)
ADVANCED CONCEPTS (REDUX, CONTEXT API, OPTIMIZATION)
COMMON INTERVIEW QUESTIONS AND ANSWERS
Q&A

VIRTUAL DOM

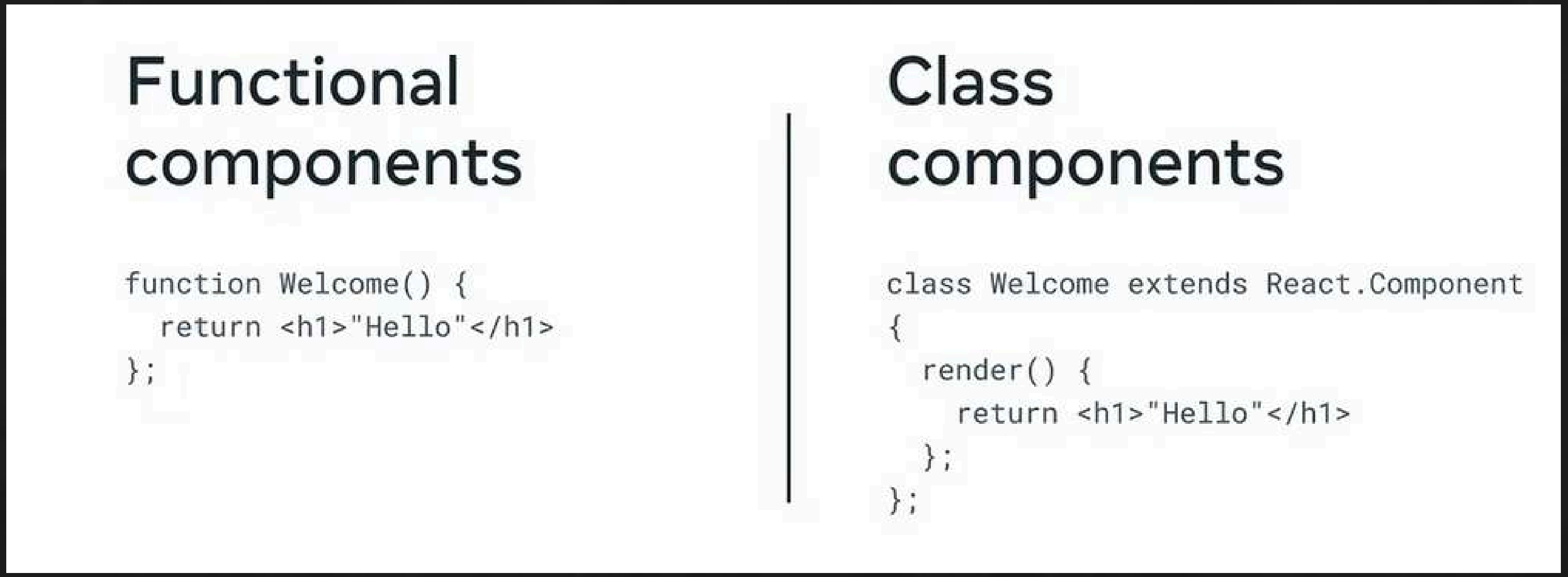
- A **lightweight copy** of the real DOM (Document Object Model).
- React updates the Virtual DOM first, then syncs only necessary changes with the real DOM.
- Avoids direct manipulation of the actual DOM.
- Uses a Diffing Algorithm to efficiently update UI.
- Reduces unnecessary re-renders.




update
Original DOM
Virtual DOM
Virtual DOM

COMPONENT-BASED ARCHITECTURE

- React apps are built using **reusable** components.
- Every UI element (button, navbar, form) is a component.
- Components can be nested inside each other.



Functional components

```
function Welcome() {
  return <h1>"Hello"</h1>
};
```

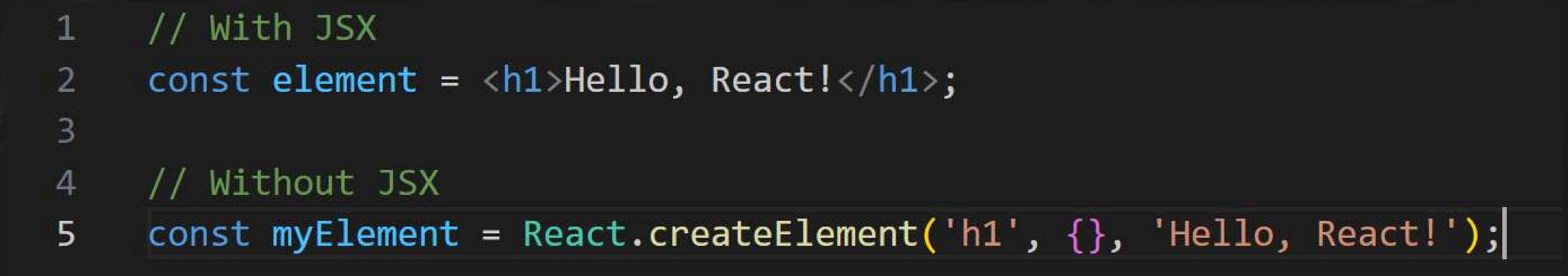
Class components

```
class Welcome extends React.Component
{
  render() {
    return <h1>"Hello"</h1>
  };
};
```

- Functional components are preferred today due to **React Hooks** (better performance and readability)

JSX (JAVASCRIPT XML)

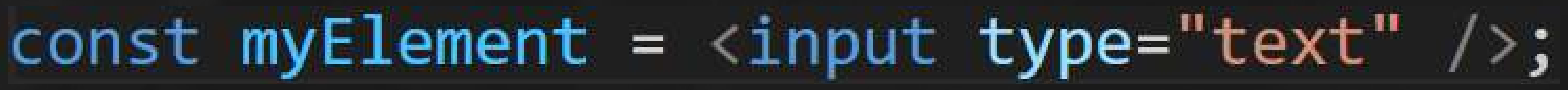
- JSX allows writing HTML-like syntax inside JavaScript.
- JSX is not HTML → It compiles to React.createElement().



```
// With JSX
const element = <h1>Hello, React!</h1>;

// Without JSX
const myElement = React.createElement('h1', {}, 'Hello, React!');
```

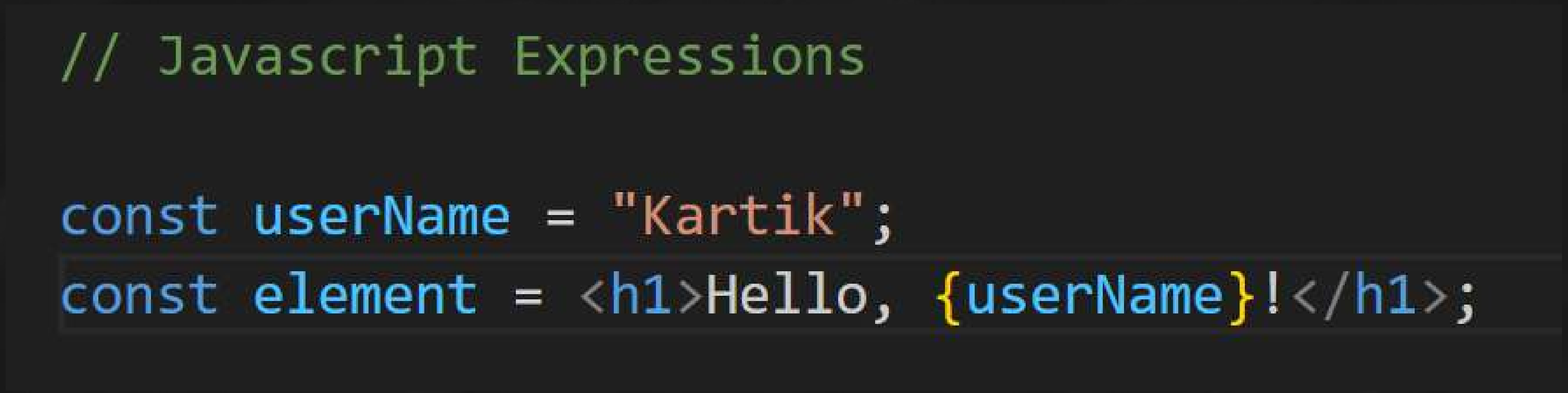
- JSX is converted to JavaScript behind the scenes.

- JSX follows XML rules, and therefore HTML elements must be properly closed.



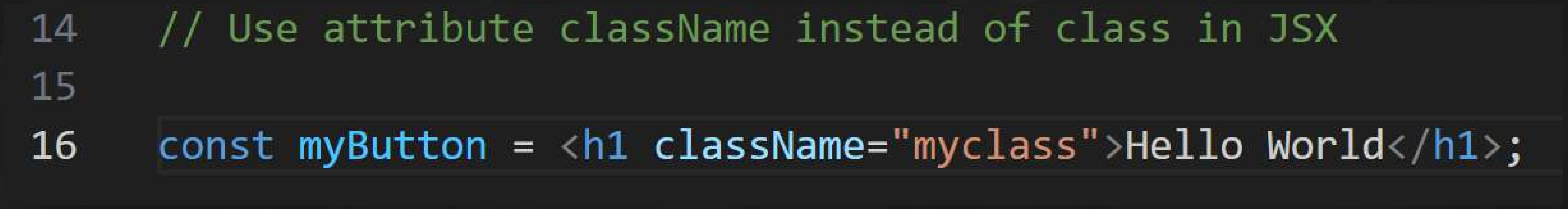
```
const myElement = <input type="text" />;
```

- Allows embedding JavaScript expressions inside {}.



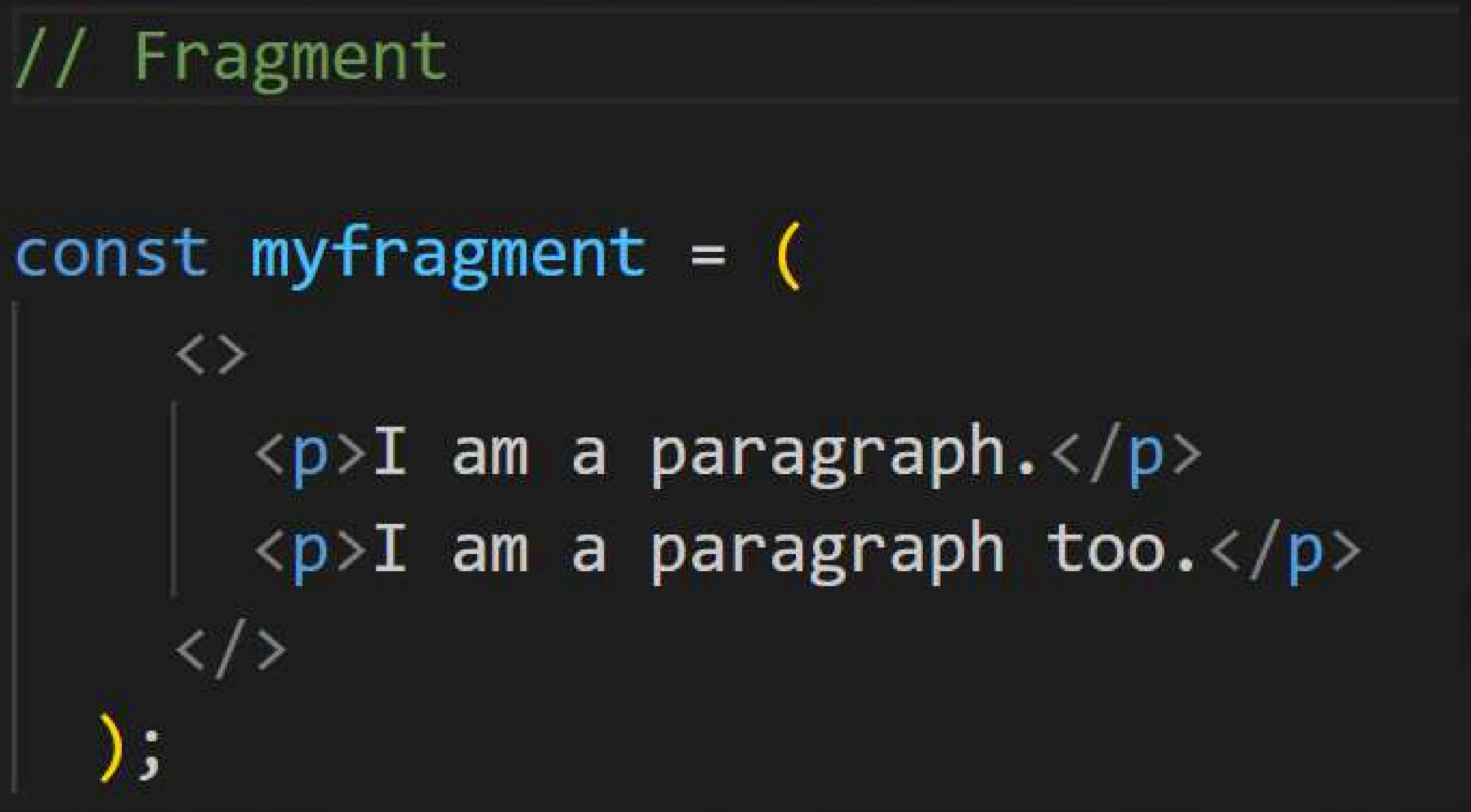
```
// Javascript Expressions

const userName = "Kartik";
const element = <h1>Hello, {userName}!</h1>;
```

- Attribute class = className



```
// Use attribute className instead of class in JSX

const myButton = <h1 className="myclass">Hello World</h1>;
```

- JSX will throw an error if the HTML is not correct, or if the HTML misses a parent element.
- You can use a "fragment" to wrap multiple lines which will prevent unnecessarily adding extra nodes to the DOM. (<></>)



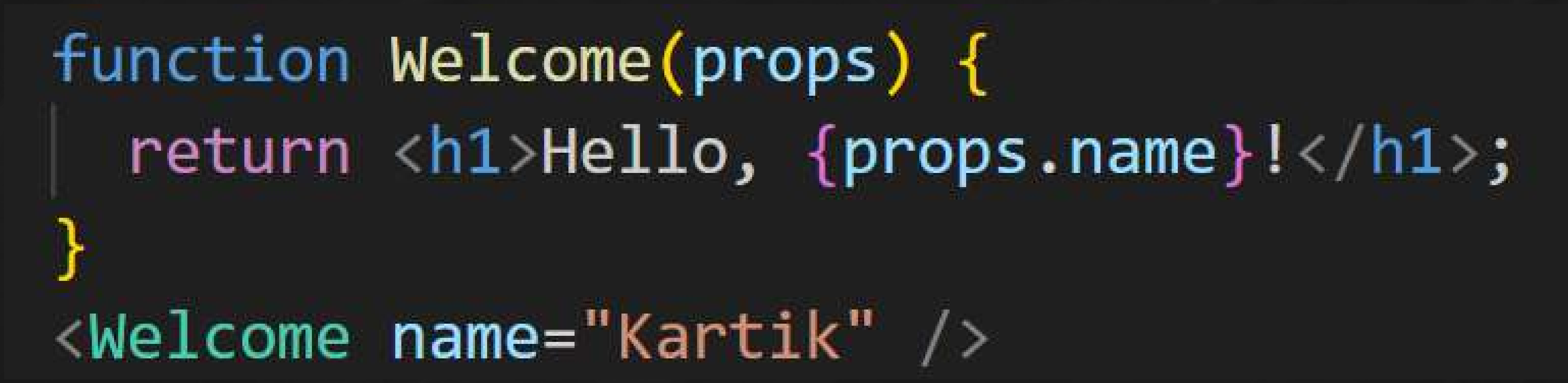
```jsx
// Fragment

const myfragment = (
    <>
      <p>I am a paragraph.</p>
      <p>I am a paragraph too.</p>
    </>
  );
```

PROPS, STATE, EVENTS & LIFECYCLE

TOPIC
INTRODUCTION TO REACT.JS
REACT ARCHITECTURE, JSX & COMPONENTS
PROPS, STATE, EVENTS & LIFECYCLE
🚀 BREAK (10 MIN)
RENDERING, LISTS & FORMS
REACT ROUTER & NAVIGATION
🚀 BREAK (10 MIN)
ADVANCED CONCEPTS (REDUX, CONTEXT API, OPTIMIZATION)
COMMON INTERVIEW QUESTIONS AND ANSWERS
Q&A

PROPS

- Props (short for "Properties") allow components to communicate by passing data.
- Props are read-only and passed from parent to child.
- React Props are like function arguments in JavaScript.



```
function Welcome(props) {
  return <h1>Hello, {props.name}!</h1>;
}
<Welcome name="Kartik" />
```

PROPS VS STATE

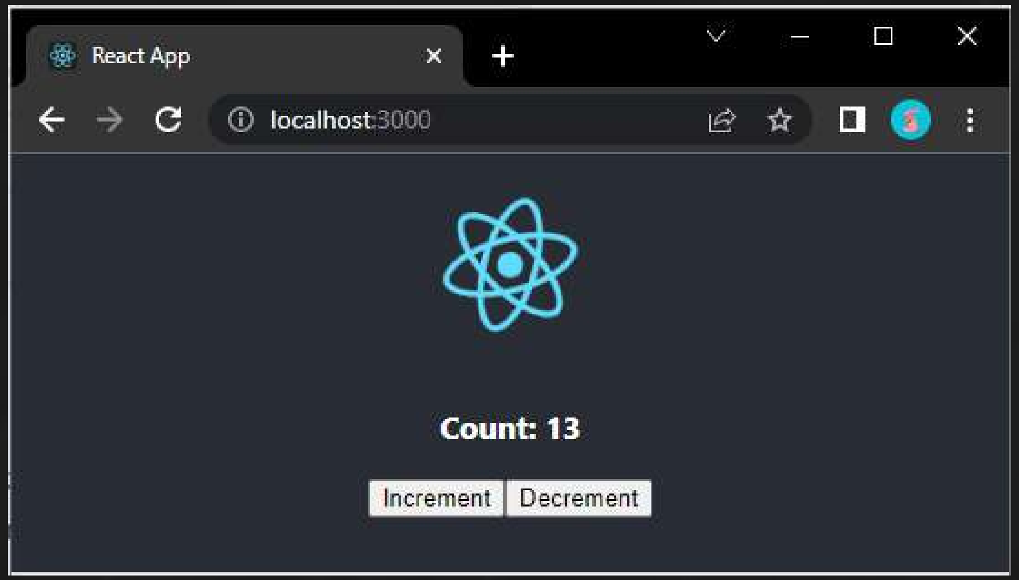
Feature	Props	State
Can be modified?	❌ No (read-only)	✅ Yes (mutable)
Passed from parent?	✅ Yes	❌ No
Triggers re-render?	✅ Yes	✅ Yes

STATE MANAGEMENT

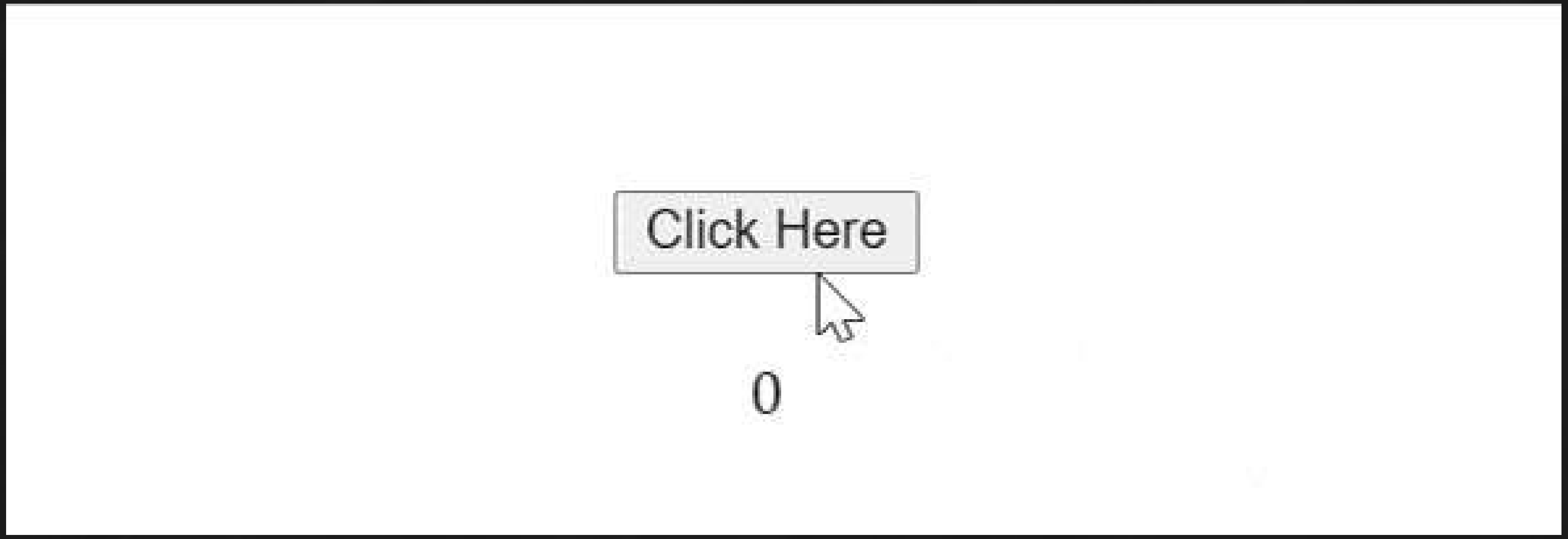
- State is mutable and controls component behavior.
- Managed using the useState Hook.

Why use useState instead of variables?

- React re-renders the component whenever state updates.
- This ensures the UI is always up-to-date with the state.
- Variables do not trigger a re-render.

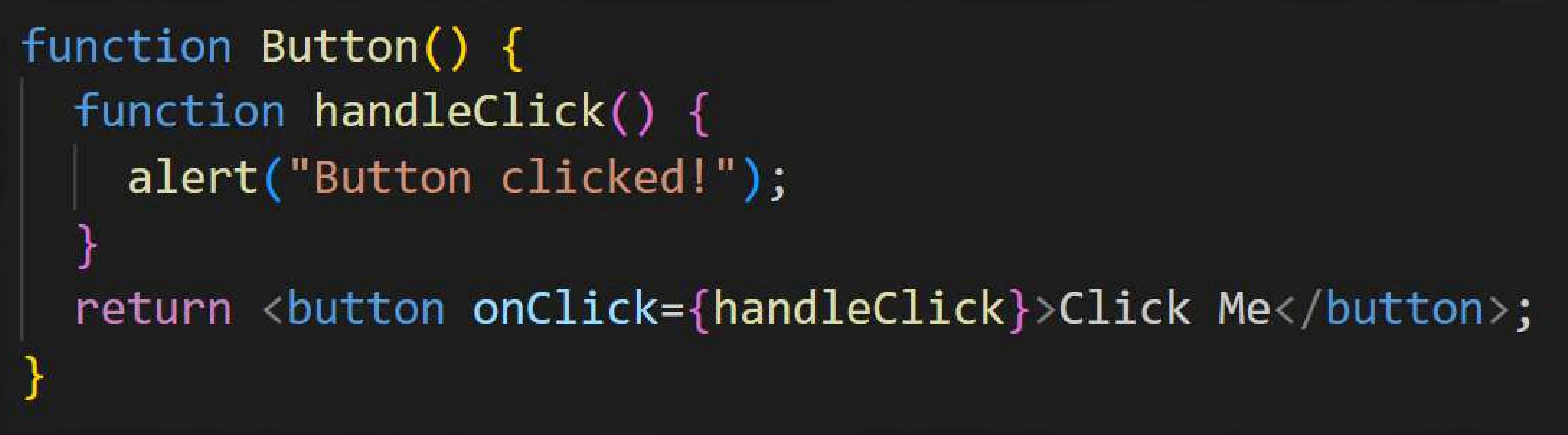


```
import { useState } from "react";
function Counter() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```



EVENT HANDLING

- Event handling is similar to JavaScript but uses camelCase.
- Event handlers must be functions, not function calls (onClick={handleClick} ✅, onClick={handleClick()} ❌).



```
function Button() {
  function handleClick() {
    alert("Button clicked!");
  }
  return <button onClick={handleClick}>Click Me</button>;
}
```

- Arrow functions help pass parameters inside event handlers.

```
function Football() {
  const shoot = (a) => {
    alert(a);
  }

  return (
    <button onClick={() => shoot("Goal!")}>Take the shot!</button>
  );
}
```

LIFECYCLE METHODS

React components go through different phases in their lifecycle:

- Mounting (When the component appears on the screen)
- Updating (When the component’s data changes and it re-renders)
- Unmounting (When the component is removed from the screen)

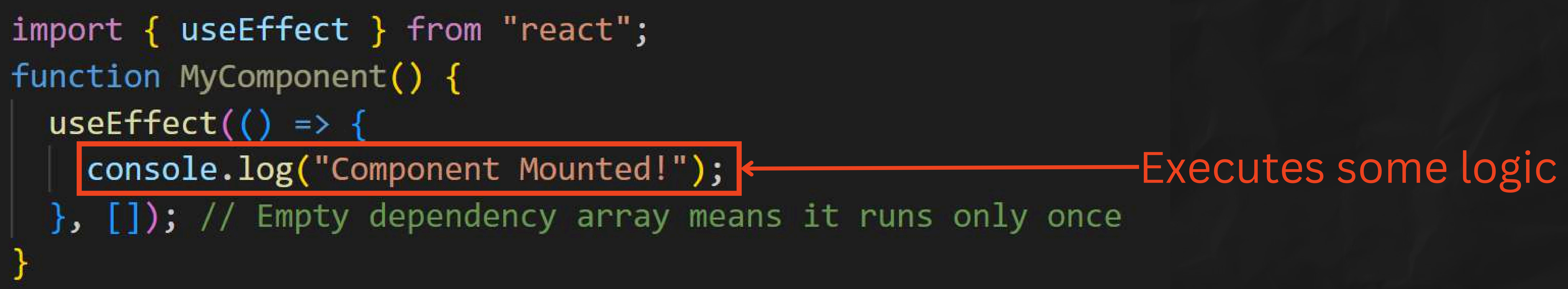
In class components, we had lifecycle methods like:

- componentDidMount() → Runs after the component is first added to the DOM.
- componentDidUpdate() → Runs after the component updates due to state/prop changes.
- componentWillUnmount() → Runs before the component is removed to clean up side effects.

However, function components don't have lifecycle methods, so we use the useEffect Hook instead.

useEffect allows function components to handle side effects, such as:

- Fetching API data when the component loads
- Updating data when state change
- Cleaning up resources (like event listeners, intervals) when the component is removed



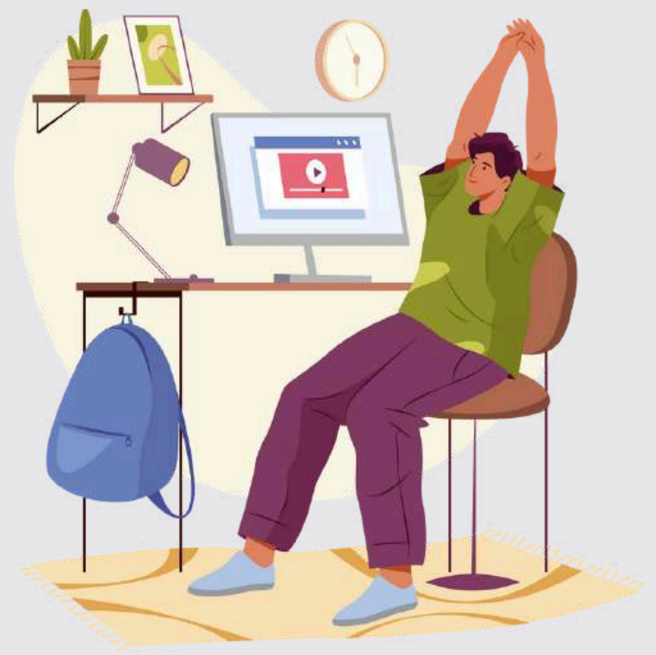
```
import { useEffect } from "react";
function MyComponent() {
  useEffect(() => {
    console.log("Component Mounted!");
  }, []); // Empty dependency array means it runs only once
}
```

Lifecycle Phases & Equivalent useEffect Hooks

Lifecycle Phase	Class Component Method	Equivalent useEffect Hook
Mounting (when the component appears)	componentDidMount()	useEffect(() => {...}, [])
Updating (when state/props change)	componentDidUpdate()	useEffect(() => {...}, [dependency])
Unmounting (when the component is removed)	componentWillUnmount()	Cleanup function inside useEffect

BREAK TIME

Let’s take a short break to stretch and hydrate.
See you back soon!



RENDERING, LISTS & FORMS

TOPIC
INTRODUCTION TO REACT.JS
REACT ARCHITECTURE, JSX & COMPONENTS
PROPS, STATE, EVENTS & LIFECYCLE
🚀 BREAK (10 MIN)
RENDERING, LISTS & FORMS
REACT ROUTER & NAVIGATION
🚀 BREAK (10 MIN)
ADVANCED CONCEPTS (REDUX, CONTEXT API, OPTIMIZATION)
COMMON INTERVIEW QUESTIONS AND ANSWERS
Q&A

CONDITIONAL RENDERING

- Dynamically show/hide elements based on conditions.

Methods for Conditional Rendering:

- && (Logical AND) operator
- Ternary (condition ? true : false) operator
- if statements (inside function body)

Using && Operator

```jsx
function Message({ isLoggedIn }) {
  return <p>{isLoggedIn && "Welcome Back!"}</p>;
}
```

Using Ternary Operator

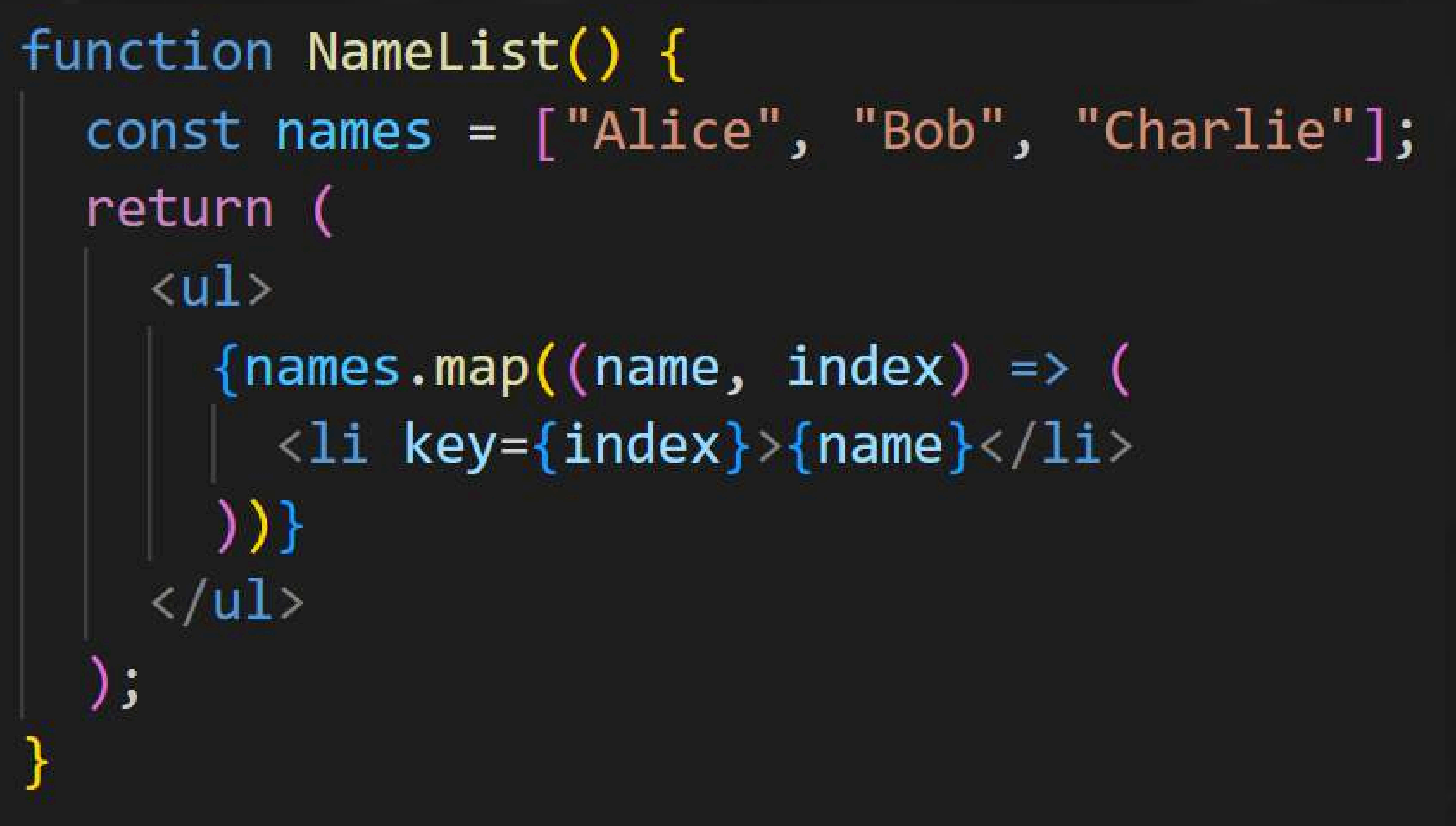
```jsx
function Greeting({ isLoggedIn }) {
  return <p>{isLoggedIn ? "Welcome Back!" : "Please Log In"}</p>;
}
```

Using If inside function

```jsx
function Greeting({ isLoggedIn }) {
  if (isLoggedIn) {
    return <p>Welcome Back!</p>;
  } else {
    return <p>Please Log In</p>;
  }
}
```

LIST RENDERING

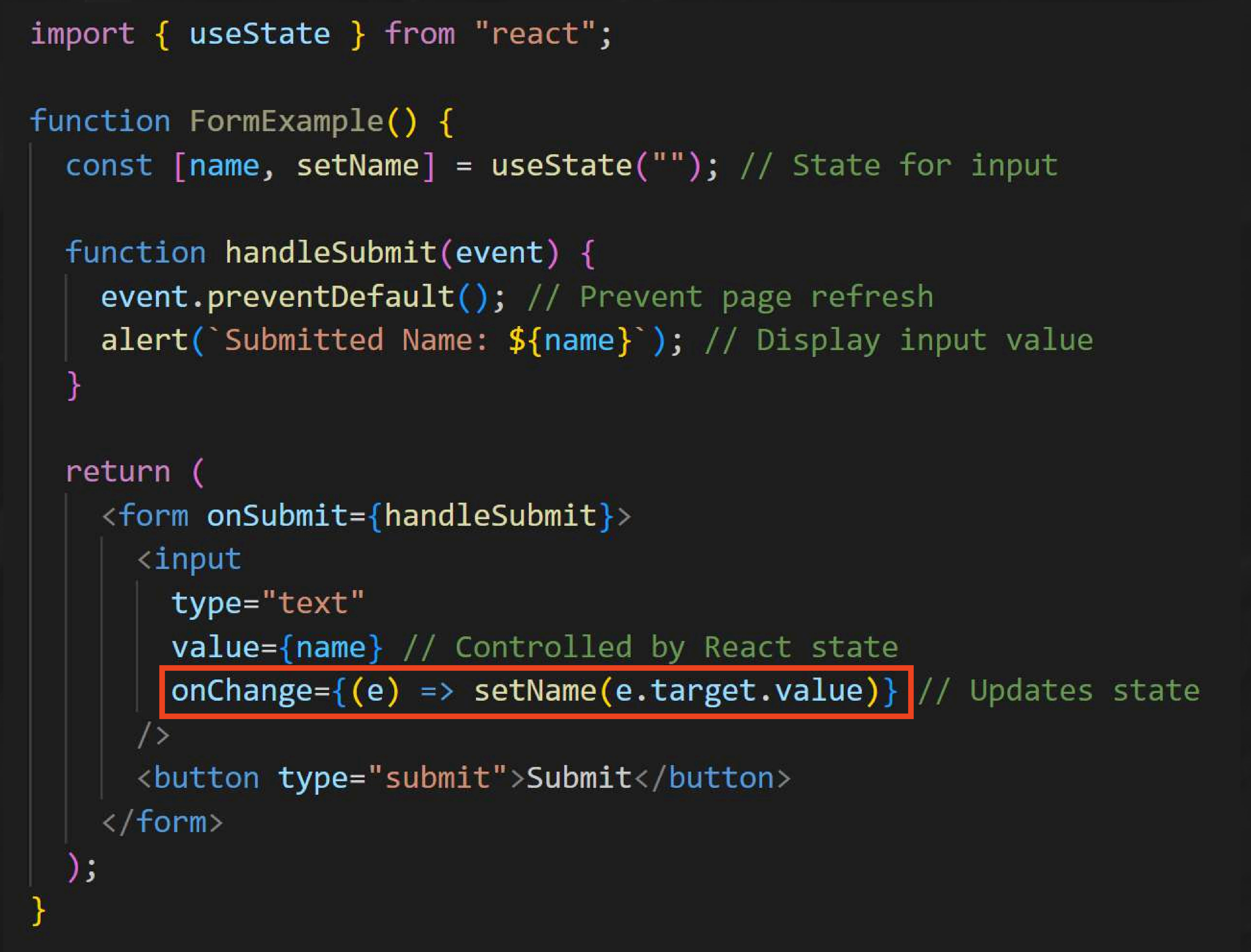
- Dynamically rendering a list of items using .map().
- Key Rule : Each list item must have a **unique key** to help React track changes efficiently.



```
function NameList() {
  const names = ["Alice", "Bob", "Charlie"];
  return (
    <ul>
      {names.map((name, index) => (
        <li key={index}>{name}</li>
      ))}
    </ul>
  );
}
```

HANDLING FORMS IN REACT

- Uncontrolled: The browser (DOM) manages the input (not recommended).
- Controlled: React **controls the input** via state.
- The input field’s value is stored in **state** (useState).
- Whenever the user types, React updates the value in state using **onChange**.
- Better performance when handling multiple inputs.



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

function FormExample() {
  const [name, setName] = useState(""); // State for input

  function handleSubmit(event) {
    event.preventDefault(); // Prevent page refresh
    alert(`Submitted Name: ${name}`); // Display input value
  }

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        value={name} // Controlled by React state
        onChange={(e) => setName(e.target.value)} // Updates state
      />
      <button type="submit">Submit</button>
    </form>
  );
}
```

REACT ROUTER & NAVIGATION

TOPIC
INTRODUCTION TO REACT.JS
REACT ARCHITECTURE, JSX & COMPONENTS
PROPS, STATE, EVENTS & LIFECYCLE
🚀 BREAK (10 MIN)
RENDERING, LISTS & FORMS
REACT ROUTER & NAVIGATION
🚀 BREAK (10 MIN)
ADVANCED CONCEPTS (REDUX, CONTEXT API, OPTIMIZATION)
COMMON INTERVIEW QUESTIONS AND ANSWERS
Q&A

SINGLE PAGE APPLICATIONS (SPA)

- A web app that loads a single HTML page and updates content dynamically without full-page reloads.
- Uses JavaScript to update the UI dynamically instead of reloading the page.
- React handles navigation using the **React Router library**.
- Benefits of SPAs: Faster navigation (no full reload), Better user experience, Reduced server load

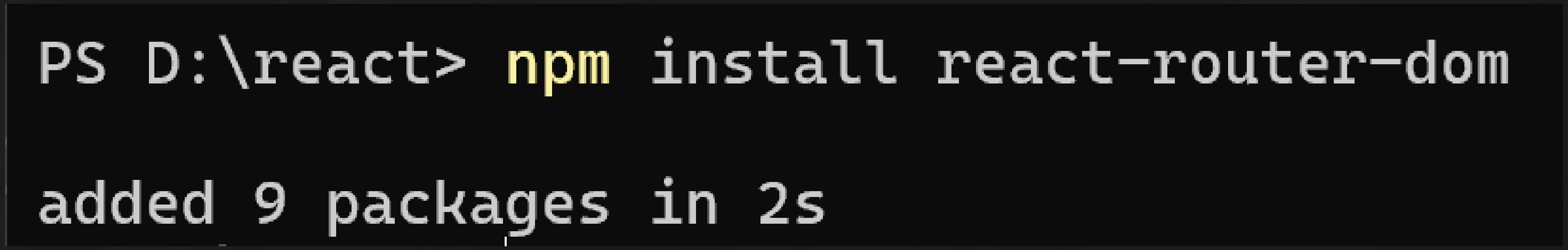
REACT ROUTER

- A library for handling navigation in React apps.
- Allows switching between pages without refreshing the browser.

Core Components:

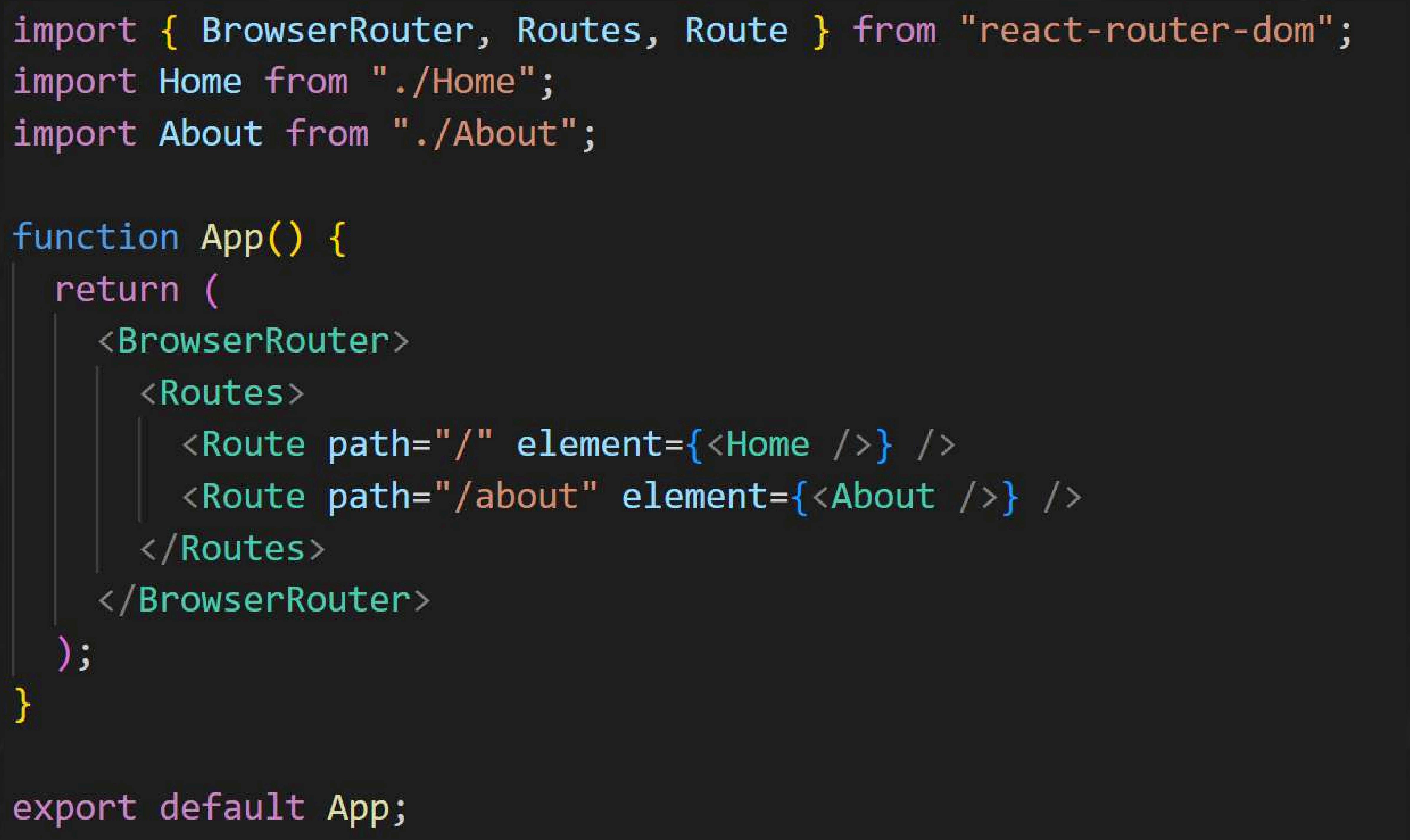
- <BrowserRouter> → Wraps the app to enable routing.
- <Routes> → Contains different <Route> components.
- <Route> → Defines a specific path and its corresponding component.
- <Link> → Used for navigation instead of <a> tags (prevents full reload).

- Step 1: Install React Router



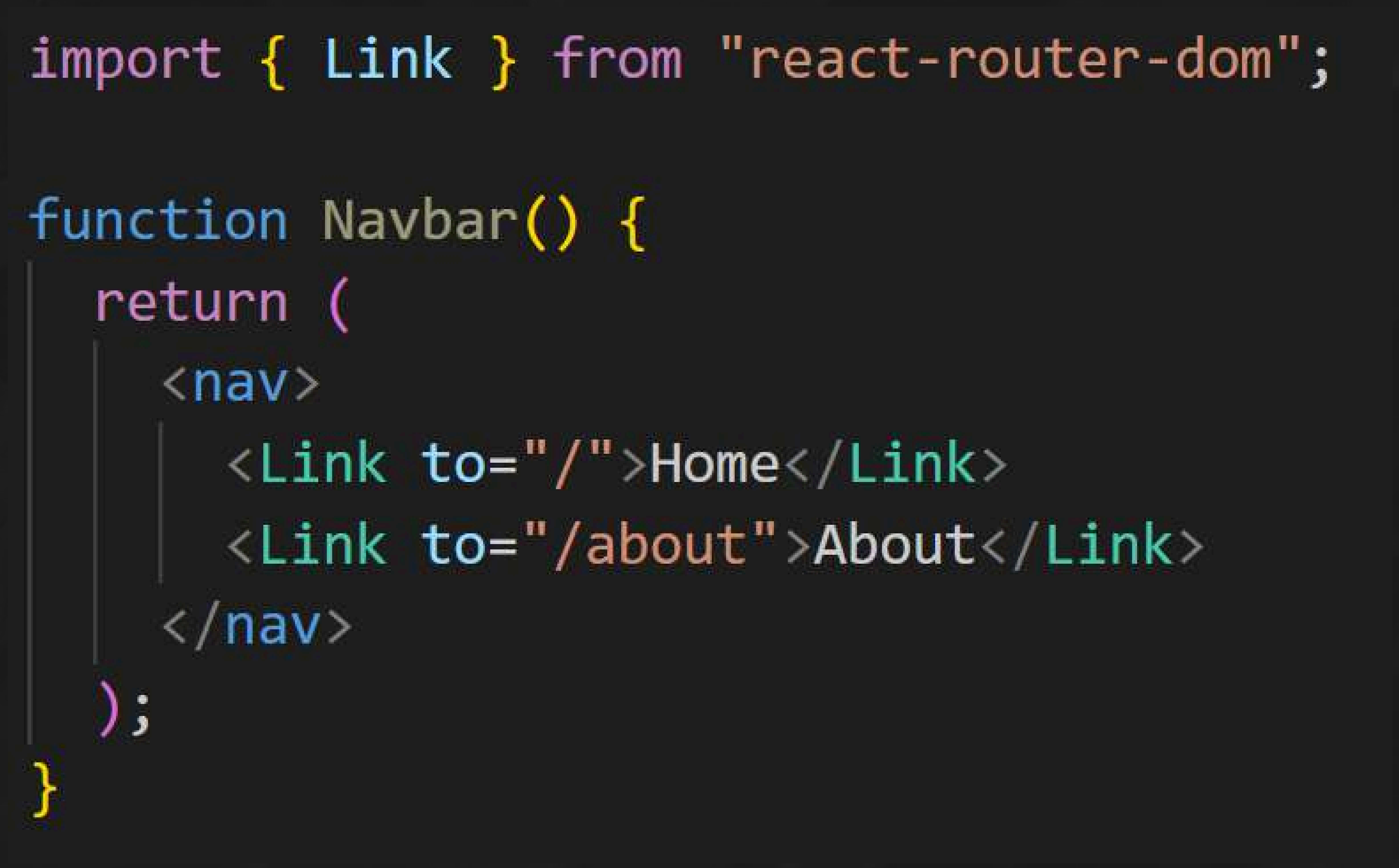
```
PS D:\react> npm install react-router-dom

added 9 packages in 2s
```

- Step 2: Set up Routing

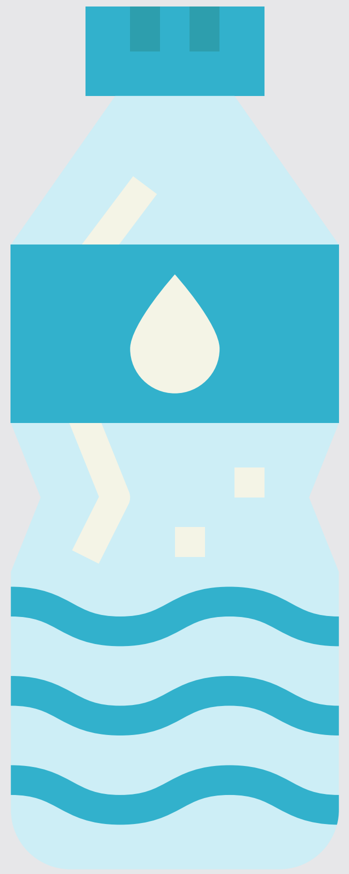


```
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Home from "./Home";
import About from "./About";

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </BrowserRouter>
  );
}

export default App;
```

- Step 3: Use <Link> for Navigation



```jsx
import { Link } from "react-router-dom";

function Navbar() {
  return (
    <nav>
      <Link to="/">Home</Link>
      <Link to="/about">About</Link>
    </nav>
  );
}
```



BREAK TIME

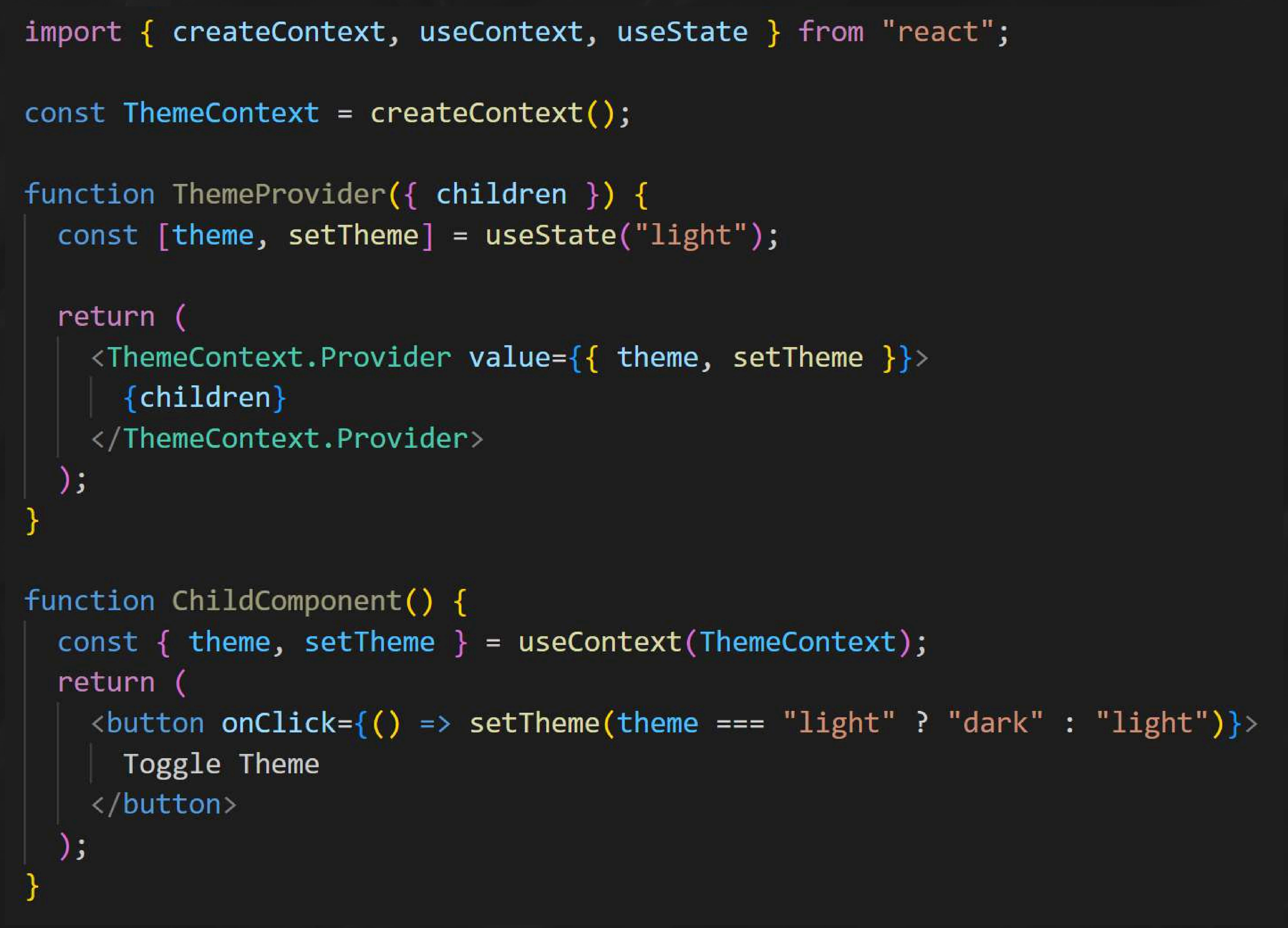
Let’s take a short break to stretch and hydrate.
See you back soon!

ADVANCED CONCEPTS (REDUX, CONTEXT API, OPTIMIZATION)

TOPIC
INTRODUCTION TO REACT.JS
REACT ARCHITECTURE, JSX & COMPONENTS
PROPS, STATE, EVENTS & LIFECYCLE
🚀 BREAK (10 MIN)
RENDERING, LISTS & FORMS
REACT ROUTER & NAVIGATION
🚀 BREAK (10 MIN)
ADVANCED CONCEPTS (REDUX, CONTEXT API, OPTIMIZATION)
COMMON INTERVIEW QUESTIONS AND ANSWERS
Q&A

FLUX & STATE MANAGEMENT

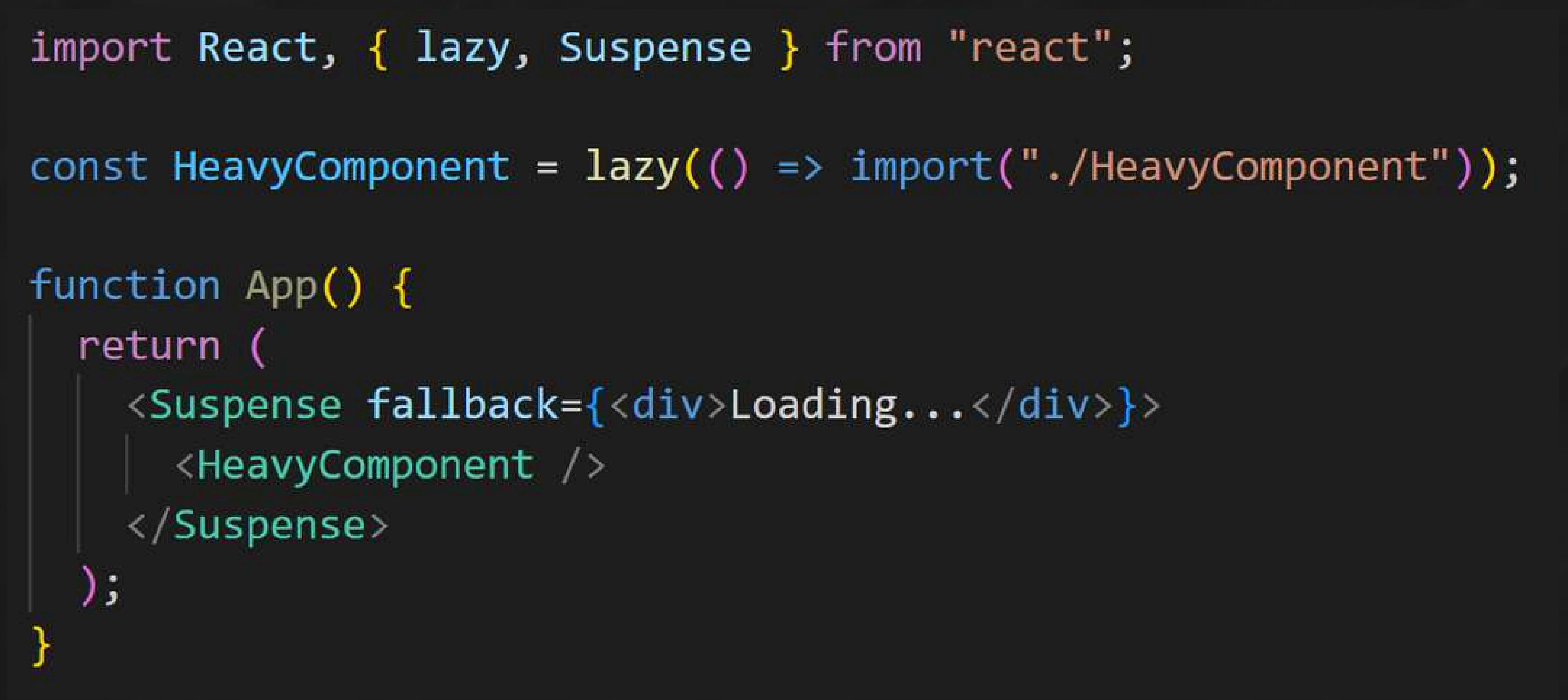
- Managing state across multiple components can be complex.
- Flux Architecture (Unidirectional Data Flow)
- Redux: A Predictable State Container uses a single store
- Actions → Reducers → Updated State
- Context API: Lightweight alternative to Redux
- Used for prop drilling elimination



```jsx
import { createContext, useContext, useState } from "react";

const ThemeContext = createContext();

function ThemeProvider({ children }) {
  const [theme, setTheme] = useState("light");

  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      {children}
    </ThemeContext.Provider>
  );
}

function ChildComponent() {
  const { theme, setTheme } = useContext(ThemeContext);
  return (
    <button onClick={() => setTheme(theme === "light" ? "dark" : "light")}>
      Toggle Theme
    </button>
  );
}
```

PERFORMANCE OPTIMIZATIONS

- Lazy Loading: Load components only when needed
- Code Splitting: Reduce initial bundle size
- React.memo(): Avoid unnecessary re-renders



```
import React, { lazy, Suspense } from "react";

const HeavyComponent = lazy(() => import("./HeavyComponent"));

function App() {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <HeavyComponent />
    </Suspense>
  );
}
```

COMMON INTERVIEW QUESTIONS AND ANSWERS

TOPIC
INTRODUCTION TO REACT.JS
REACT ARCHITECTURE, JSX & COMPONENTS
PROPS, STATE, EVENTS & LIFECYCLE
🚀 BREAK (10 MIN)
RENDERING, LISTS & FORMS
REACT ROUTER & NAVIGATION
🚀 BREAK (10 MIN)
ADVANCED CONCEPTS (REDUX, CONTEXT API, OPTIMIZATION)
COMMON INTERVIEW QUESTIONS AND ANSWERS
Q&A

1. What is the Virtual DOM, and how does React use it?

- The Virtual DOM (VDOM) is a lightweight copy of the actual DOM.
- React uses the VDOM to improve performance by minimizing direct DOM updates.
- When state changes, React updates the VDOM first, compares it with the previous version (diffing algorithm), and efficiently updates only the changed parts in the real DOM.

2. What are Controlled and Uncontrolled Components in React Forms?

- Controlled Components: Form inputs are managed by React state (useState).
- Uncontrolled Components: The input state is managed by the browser

3. What is the difference between functional and class components?

Feature	Functional Component	Class Component
Definition	Uses functions	Uses ES6 classes
State	Uses useState()	Uses this.state
Lifecycle Methods	Uses useEffect()	Uses lifecycle methods (componentDidMount)
Performance	Faster (no this binding)	Slower (requires this binding)

4. What is React Router, and why is it used?

- React Router is a library used for navigation in Single Page Applications (SPAs).
- It allows switching between different views without reloading the page.

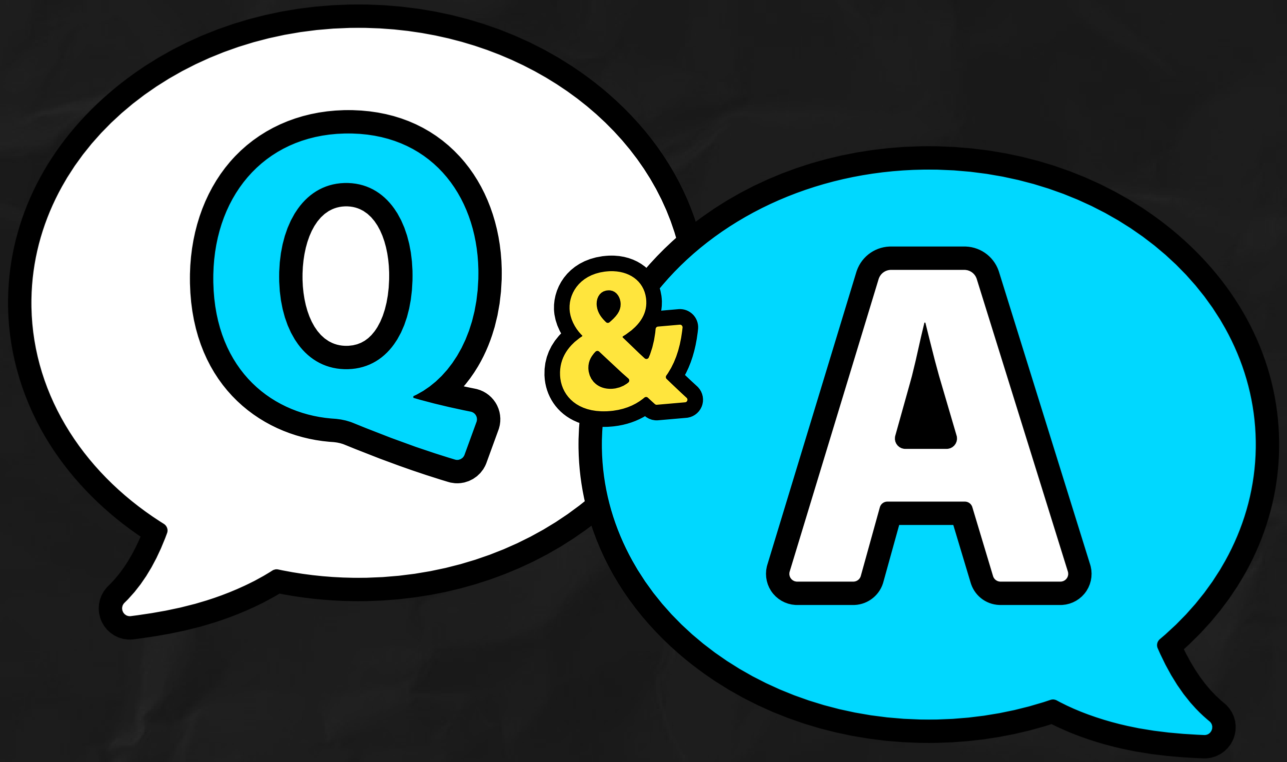
5. What are React Props and Hooks?

- Props are used to pass data from a parent component to a child component.

- Props cannot be modified by the child (they are read-only).

- Hooks allow functional components to use state and lifecycle features (which were previously only available in class components).

- Common Hooks in React:
- useState → Manages component state.
- useEffect → Handles side effects like API calls.



TOPIC
INTRODUCTION TO REACT.JS
REACT ARCHITECTURE, JSX & COMPONENTS
PROPS, STATE, EVENTS & LIFECYCLE
🚀 BREAK (10 MIN)
RENDERING, LISTS & FORMS
REACT ROUTER & NAVIGATION
🚀 BREAK (10 MIN)
ADVANCED CONCEPTS (REDUX, CONTEXT API, OPTIMIZATION)
COMMON INTERVIEW QUESTIONS AND ANSWERS
Q&A

Thank you !

Kartik-Katkar

Kartik Katkar

Resources : **https://github.com/Kartik-Katkar/ReactJS-Session**