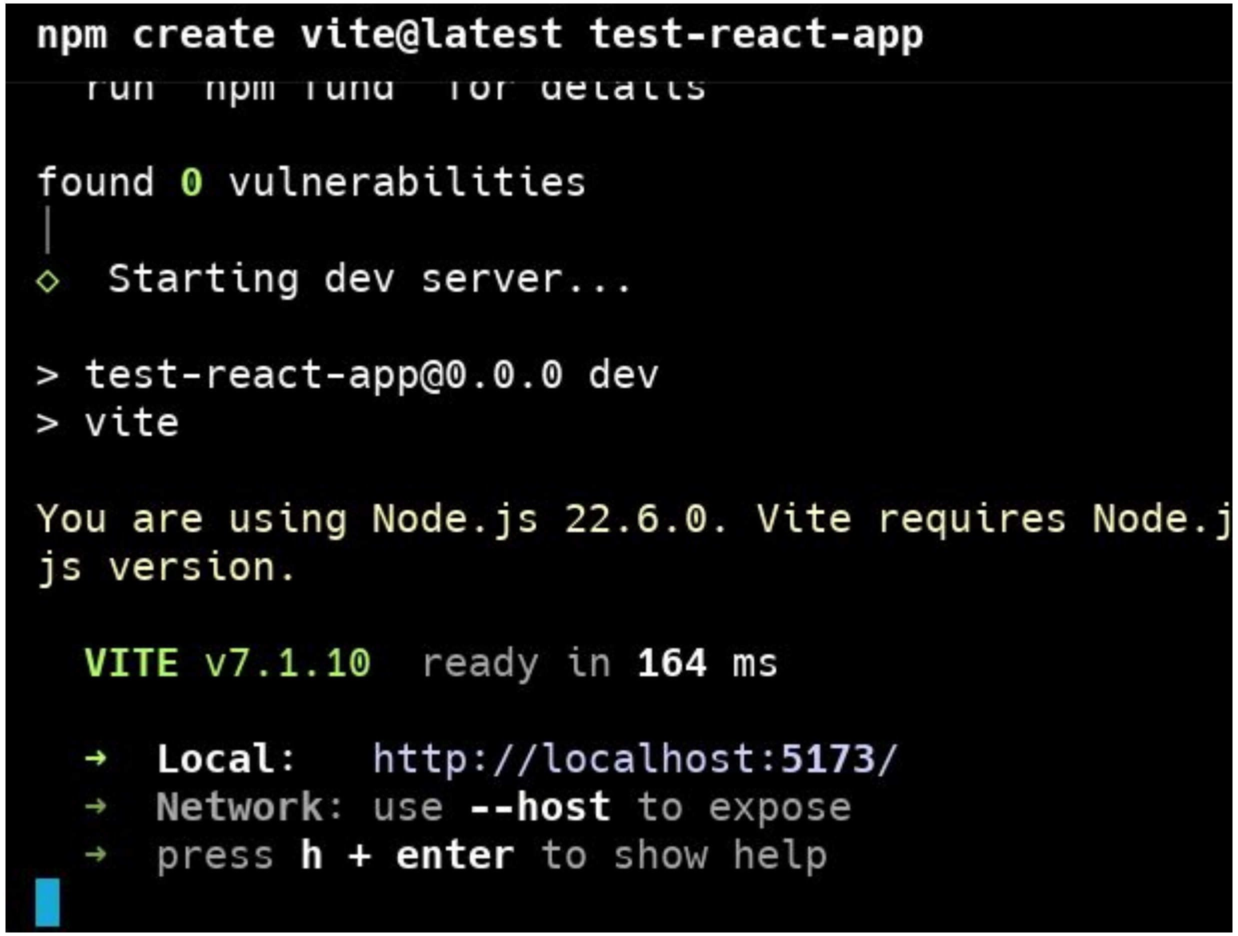
React

By: Puskar Adhikari

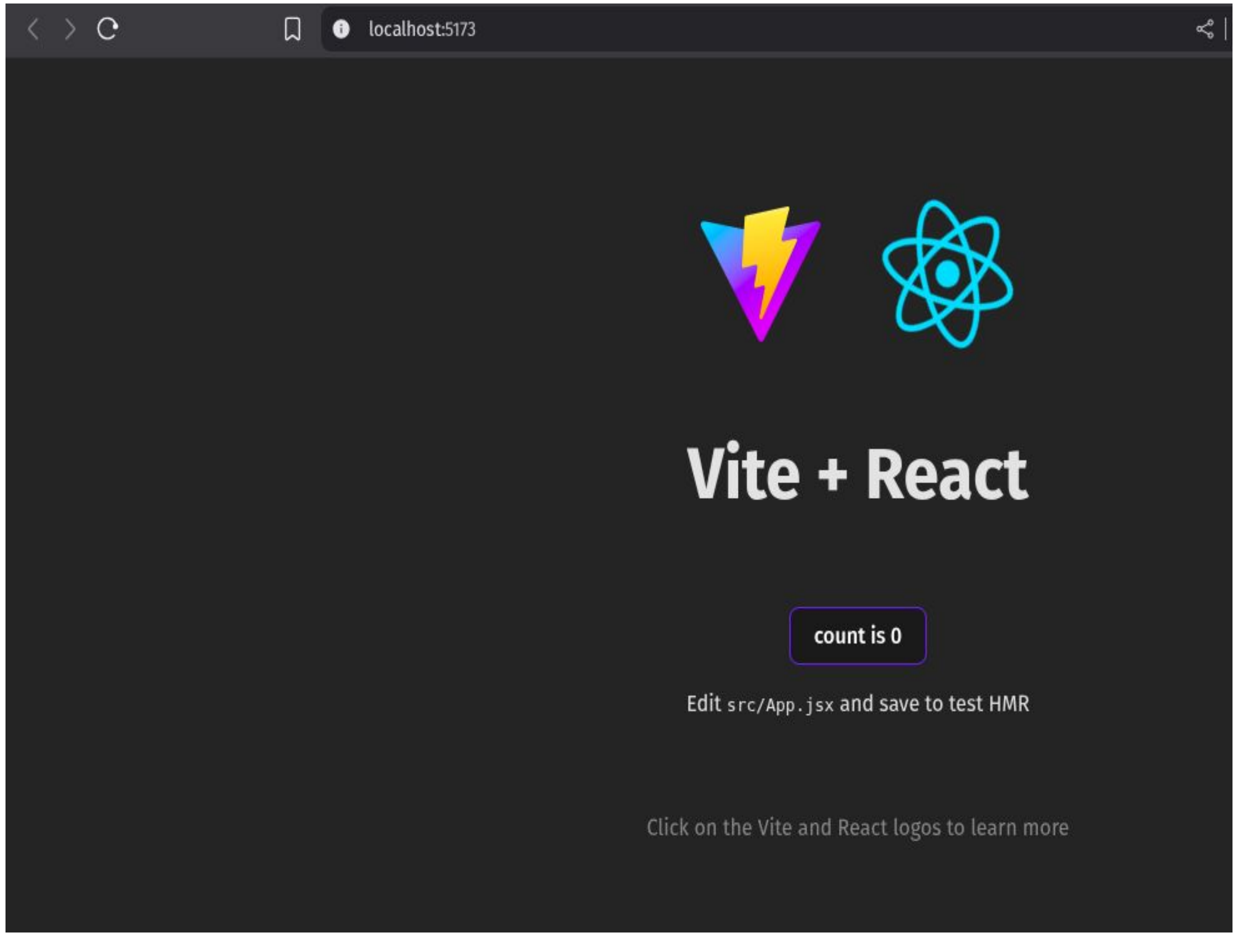
Installation

- [React Js]

React App

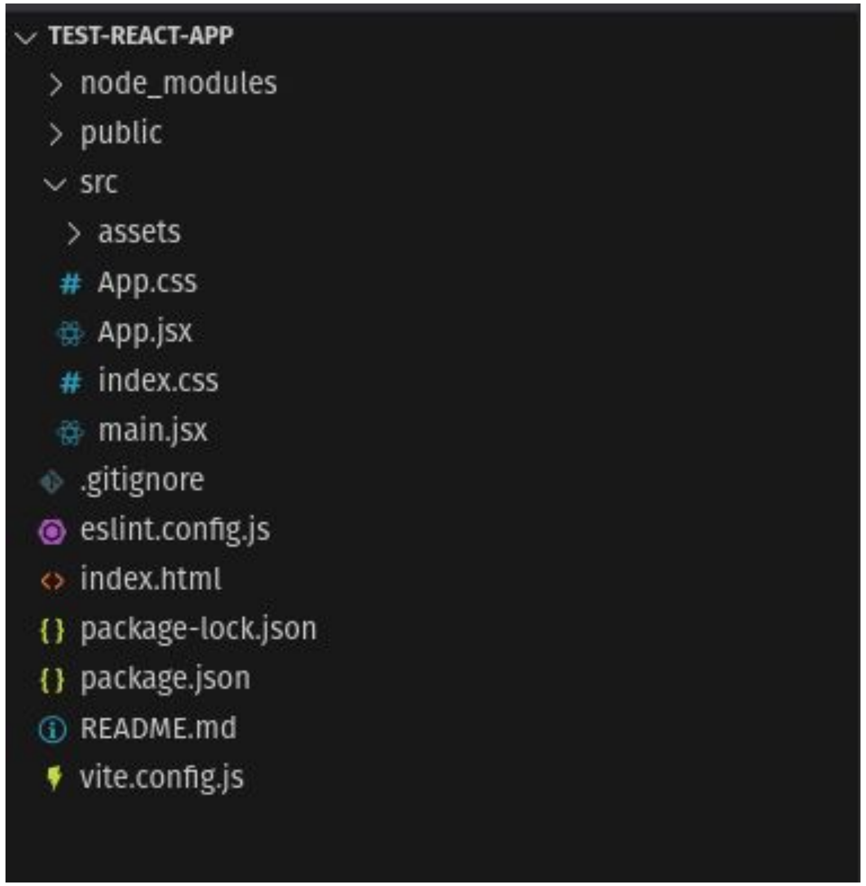


```
npm create vite@latest test-react-app
  run  npm fund  for details

found 0 vulnerabilities
│
◇  Starting dev server...

> test-react-app@0.0.0 dev
> vite

You are using Node.js 22.6.0. Vite requires Node.j
js version.

  VITE v7.1.10  ready in 164 ms

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


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

Project Structure

```
my-app/
│
├── node_modules/        # 📦 All installed dependencies (auto-created)
├── public/              # 🌐 Static files like images, icons, favicon
├── src/                 # 💡 All your React code lives here
│   ├── assets/          # → Optional: images, logos, etc.
│   ├── App.jsx          # → Main component of your app
│   ├── main.jsx         # → Entry file (connects React to the DOM)
│   └── index.css        # → Global CSS styles
│
├── .gitignore           # 🚫 Files Git should ignore
├── package.json         # 📦 Lists dependencies & scripts
├── vite.config.js       # ⚙️ Vite configuration
└── README.md            # 📝 Basic info about the project
```

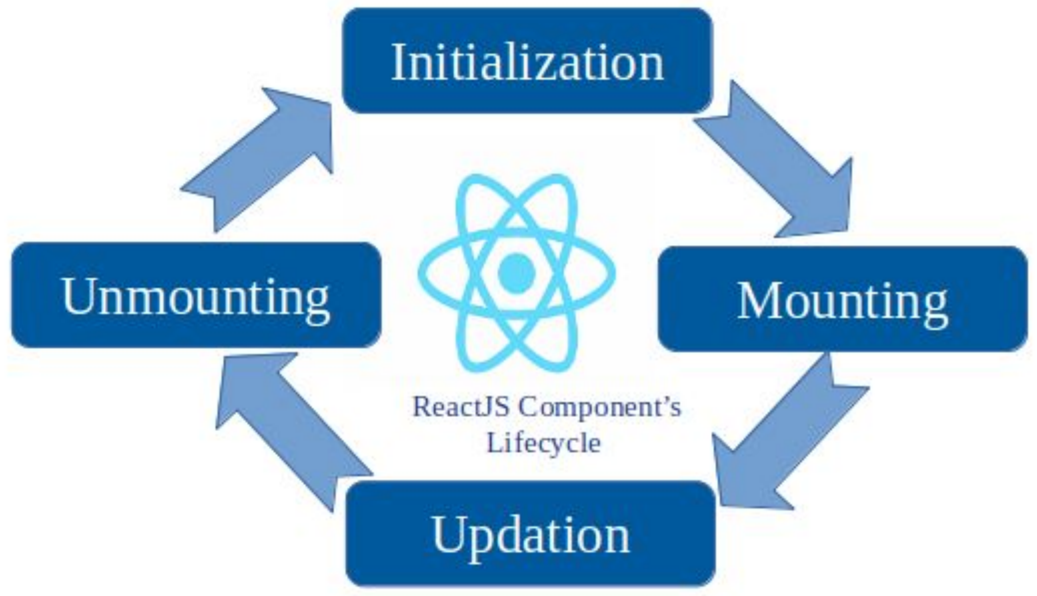


```
TEST-REACT-APP
  > node_modules
  > public
  v src
    > assets
    App.css
    App.jsx
    index.css
    main.jsx
  .gitignore
  eslint.config.js
  index.html
  package-lock.json
  package.json
  README.md
  vite.config.js
```

React Component Lifecycle

Each component in React has a lifecycle which you can monitor and manipulate during its three main phases.

1. **Mounting** → when the component is **created and added** to the DOM
2. **Updating** → when the component **re-renders** (because props or state changed)
3. **Unmounting** → when the component is **removed** from the DOM



React Hooks

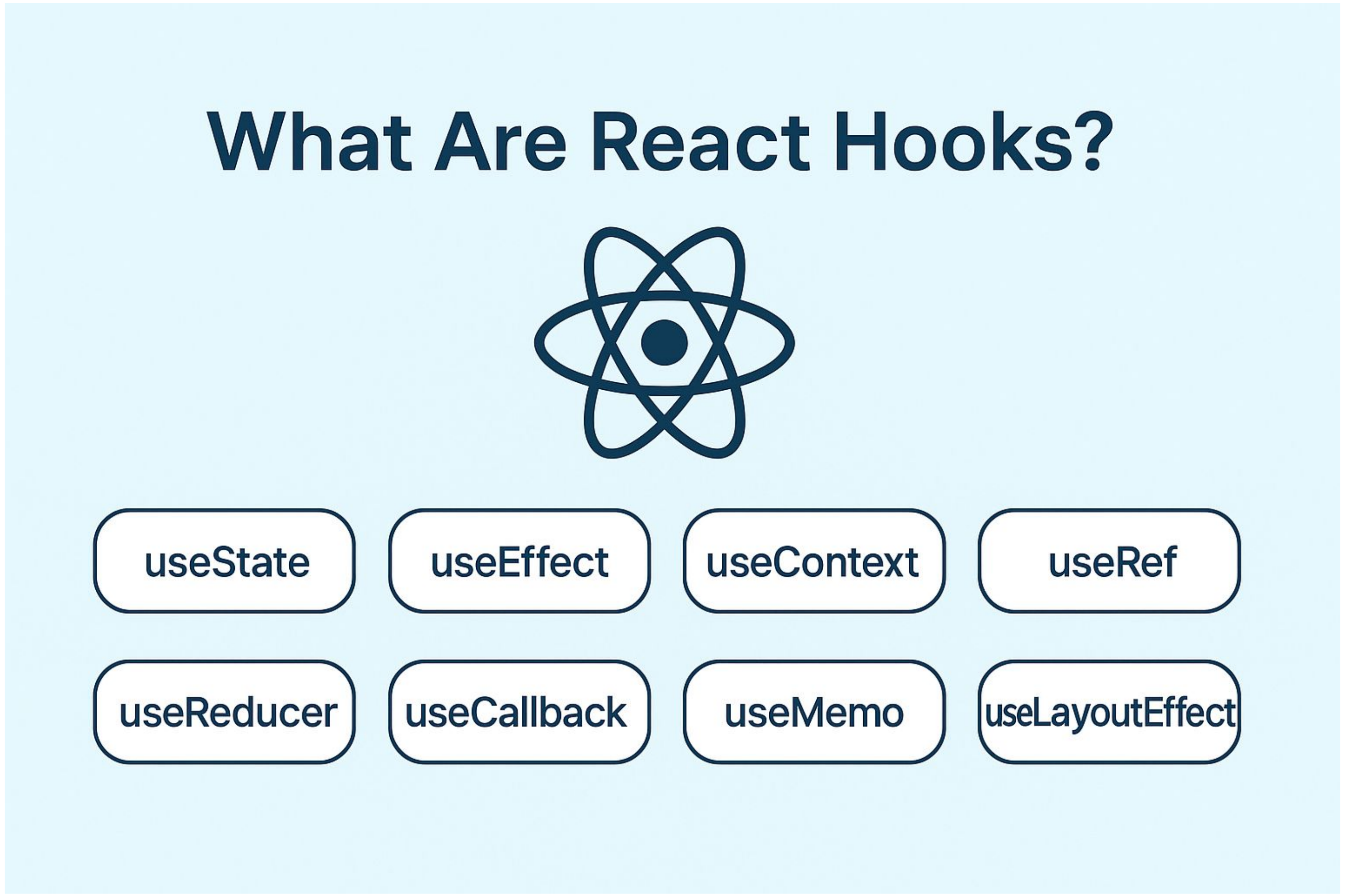
- ❏ Hooks make components shorter, cleaner, and easier to reuse.
- ❏ There are only **two main rules**, but they are very important:

  - ➢ **Only call Hooks at the top level**
    - ○ Not inside loops, conditions, or nested functions.
    - ○ React relies on the order of Hooks.

  - ➢ **Only call Hooks inside React functions**
    - ○ Either in a React component, or in your own custom Hook.

React Hooks

❑ Built in React Hooks



Arrow Functions: () => {}

Normal function:

```
function greet(name) {
    return "Hello " + name;
}
```

Arrow function:

```
const greet = (name) => {
  return "Hello " + name;
};
```

Or:

```
const greet = (name) => "Hello " + name;
```

Usage:

a. Defining components

```
const Welcome = () => {
  return <h1>Hello React!</h1>;
};
```

b. Event handlers

```
<button onClick={() =>
alert("Clicked!")}>Click Me</button>
```

c. State updates

```
setCount((prev) => prev + 1);
```

React Hook: useState

- ❑ useState lets your component **remember values between renders**, it gives your component its **own memory**.
- ❑ Every time your component re-renders, React will keep track of this value.

Syntax

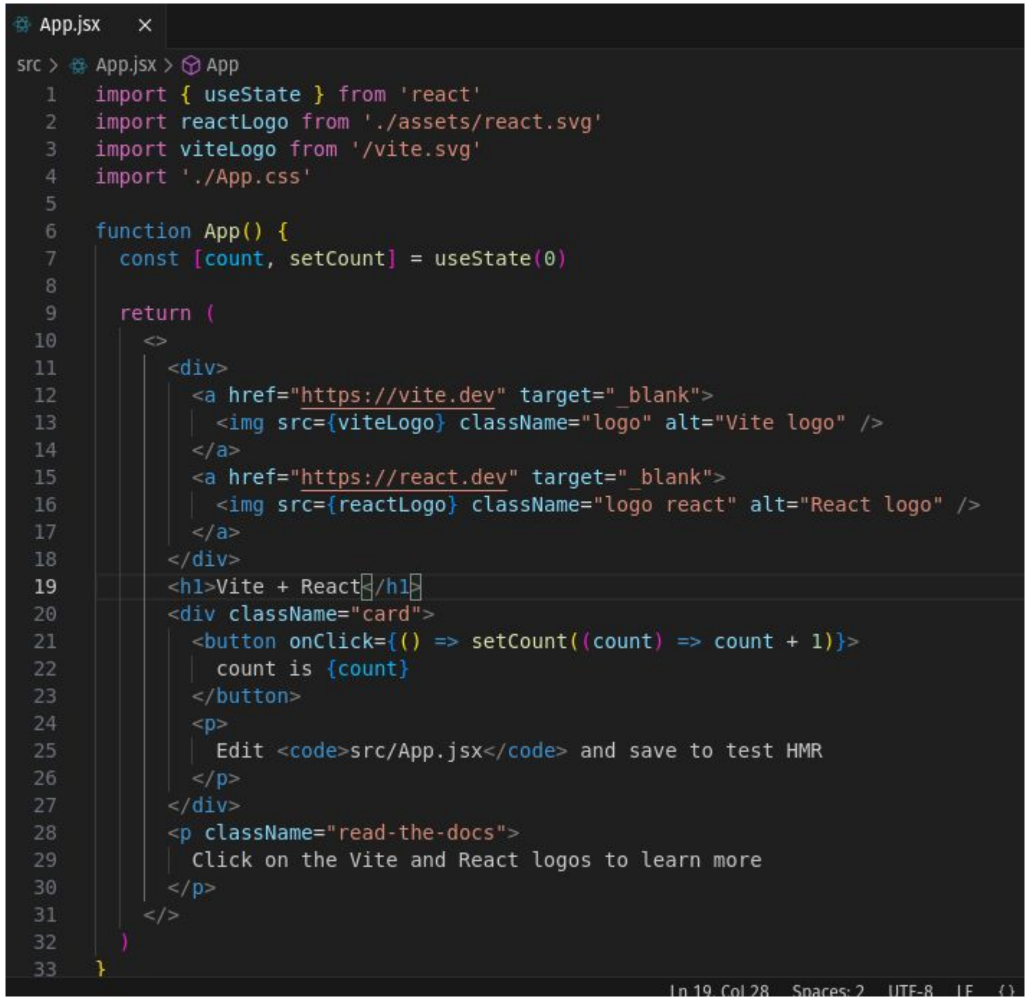
```
const [stateVariable, setStateFunction] = useState(initialValue);
```

Example

```
const [count, setCount] = useState(0);
```

React Hook: useState

Find the example in src/App.jsx



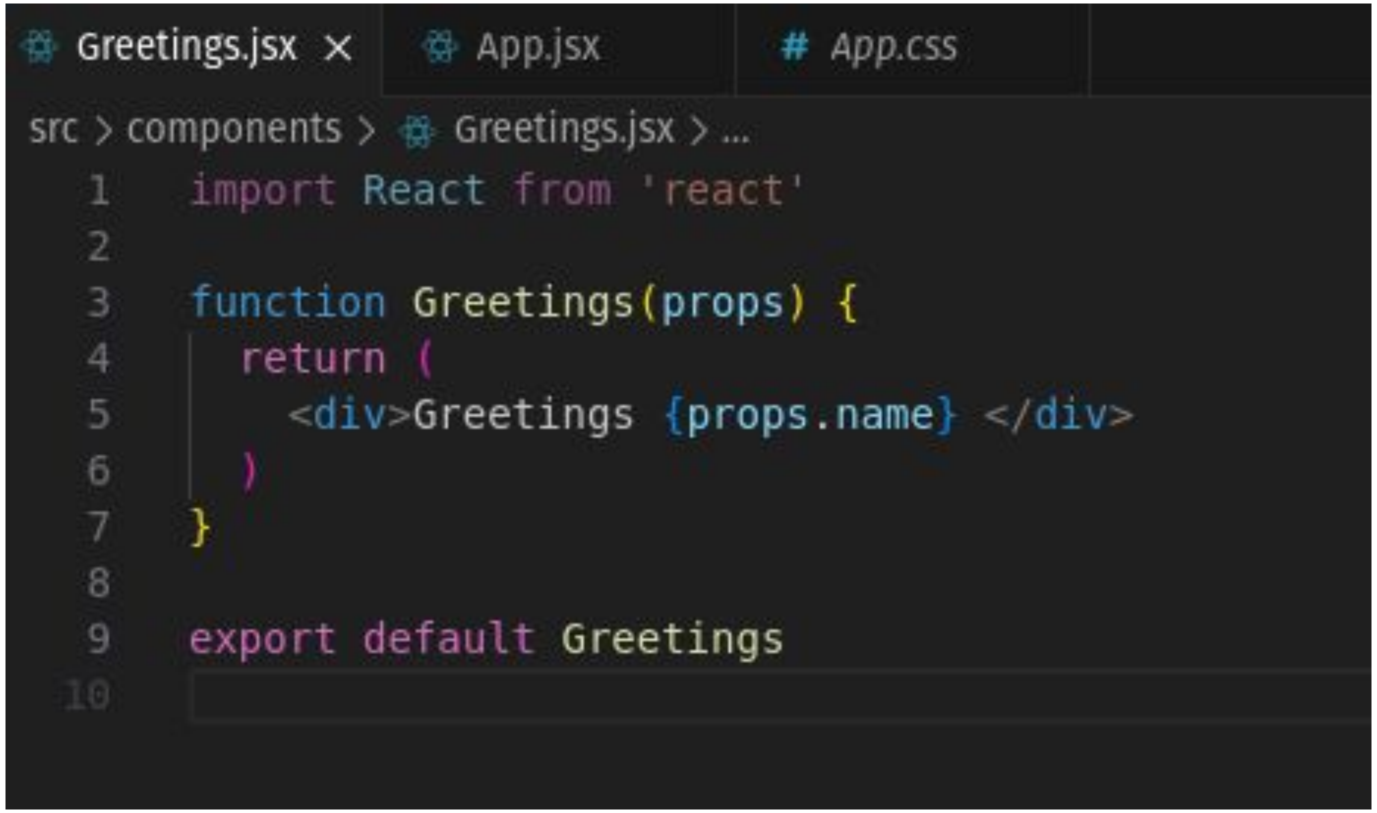
```jsx
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'

function App() {
  const [count, setCount] = useState(0)

  return (
    <>
      <div>
        <a href="https://vite.dev" target="_blank">
          <img src={viteLogo} className="logo" alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank">
          <img src={reactLogo} className="logo react" alt="React logo" />
        </a>
      </div>
      <h1>Vite + React</h1>
      <div className="card">
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <p>
          Edit <code>src/App.jsx</code> and save to test HMR
        </p>
      </div>
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
    </>
  )
}
```

Components

Make a new folder components inside src/, then make a new file Greetings.jsx

Recommended:

- ❑ React Extension
- ❑ Type rfcp
- ❑ Check rfc…



```
import React from 'react'

function Greetings(props) {
  return (
    <div>Greetings {props.name} </div>
  )
}

export default Greetings
```

Props

What is a Prop in React?

- ❖ Prop stands for “property”.
- ❖ It’s a way to pass data from a parent component to a child component.
- ❖ Props are read-only — a child component cannot modify them directly.

Why Props?

- ❖ Props allow components to be dynamic and reusable.
- ❖ Instead of hardcoding values inside a component, you can pass different values from the parent.

Using a Custom Component with Props

Call the Greetings component just like an HTML tag:

<Greetings />

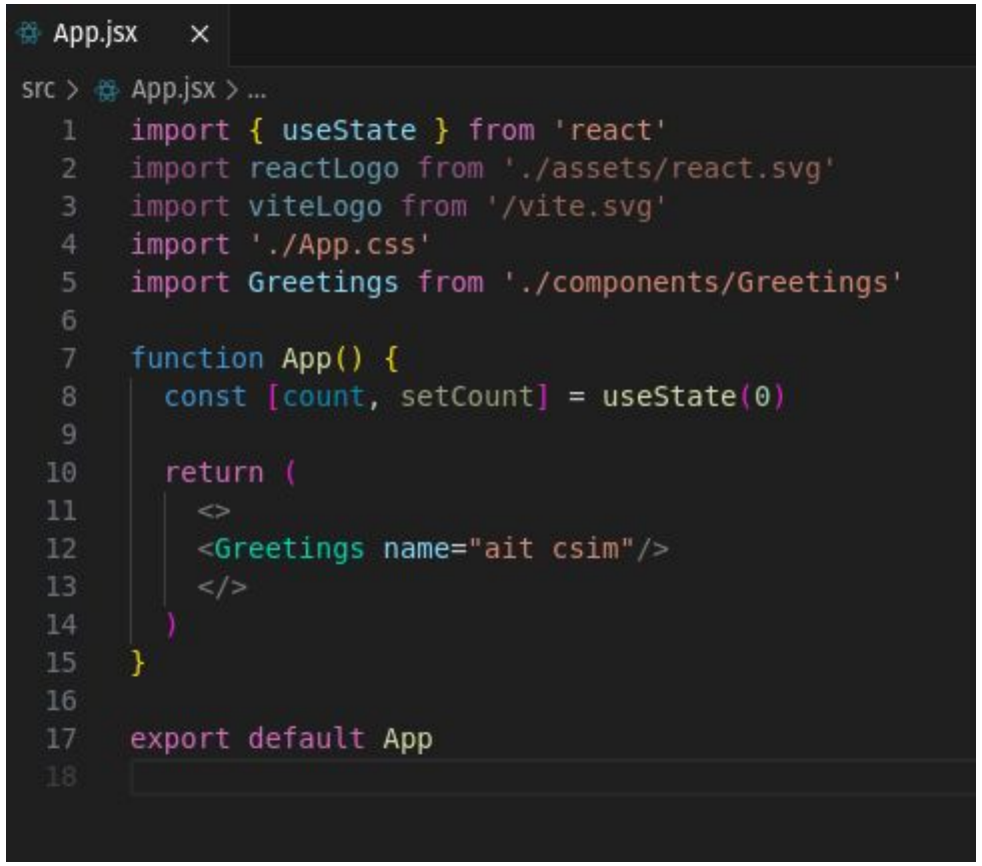
Add it inside the App function’s return statement.

Pass data to it using **props**, just like HTML attributes:

<Greetings name="Puskar" message="Welcome to React!" />

Inside Greetings, access the props using **props.variableName**

props.name // "Puskar"
props.message // "Welcome to React!"



```jsx
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'
import Greetings from './components/Greetings'

function App() {
  const [count, setCount] = useState(0)

  return (
    <>
    <Greetings name="ait csim"/>
    </>
  )
}

export default App
```

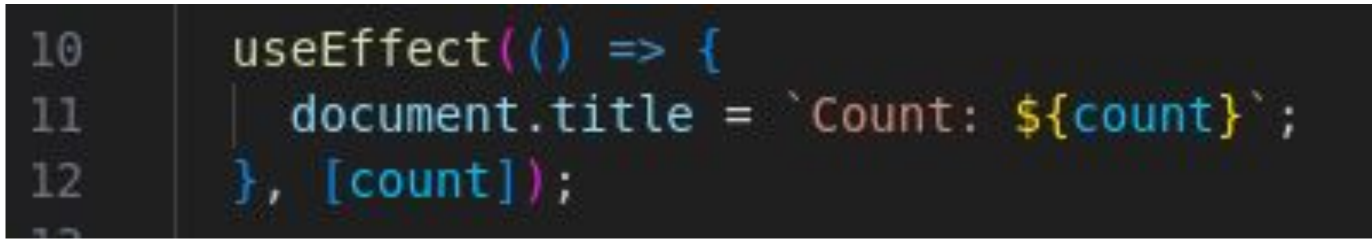
React Hook: useEffect

What Does useEffect Do?

- useEffect lets you **run side effects** in your component.
- Side effects are actions that happen **outside** the normal React rendering flow.

Examples of Side Effects

- Fetching data from an API
- Updating the page title
- Setting up a timer or interval
- Working with browser storage (localStorage, sessionStorage)
- Subscribing to events (and cleaning them up)

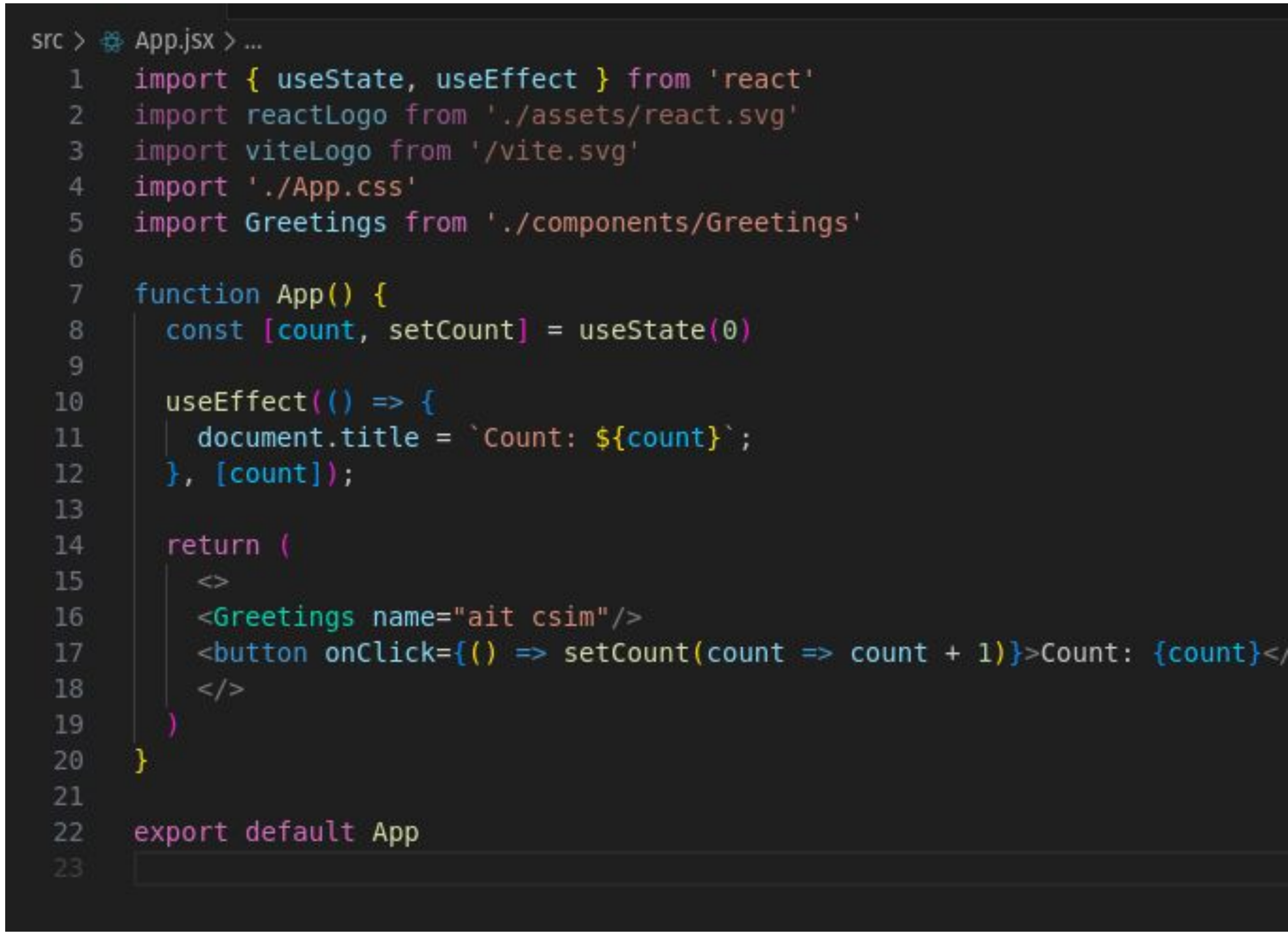


```
useEffect(() => {
  document.title = `Count: ${count}`;
}, [count]);
```

Syntax: useEffect(() => {
  // Your side effect code here
  // This code runs after every render by default
}, [*/* dependencies */*]); *// Optional dependency array*

React Hook: useEffect

```
import { useState, useEffect } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'
import Greetings from './components/Greetings'

function App() {
 const [count, setCount] = useState(0)

 useEffect(() => {
   document.title = `Count: ${count}`;
 }, [count]);

 return (
   <>
   <Greetings name="ait csim"/>
```

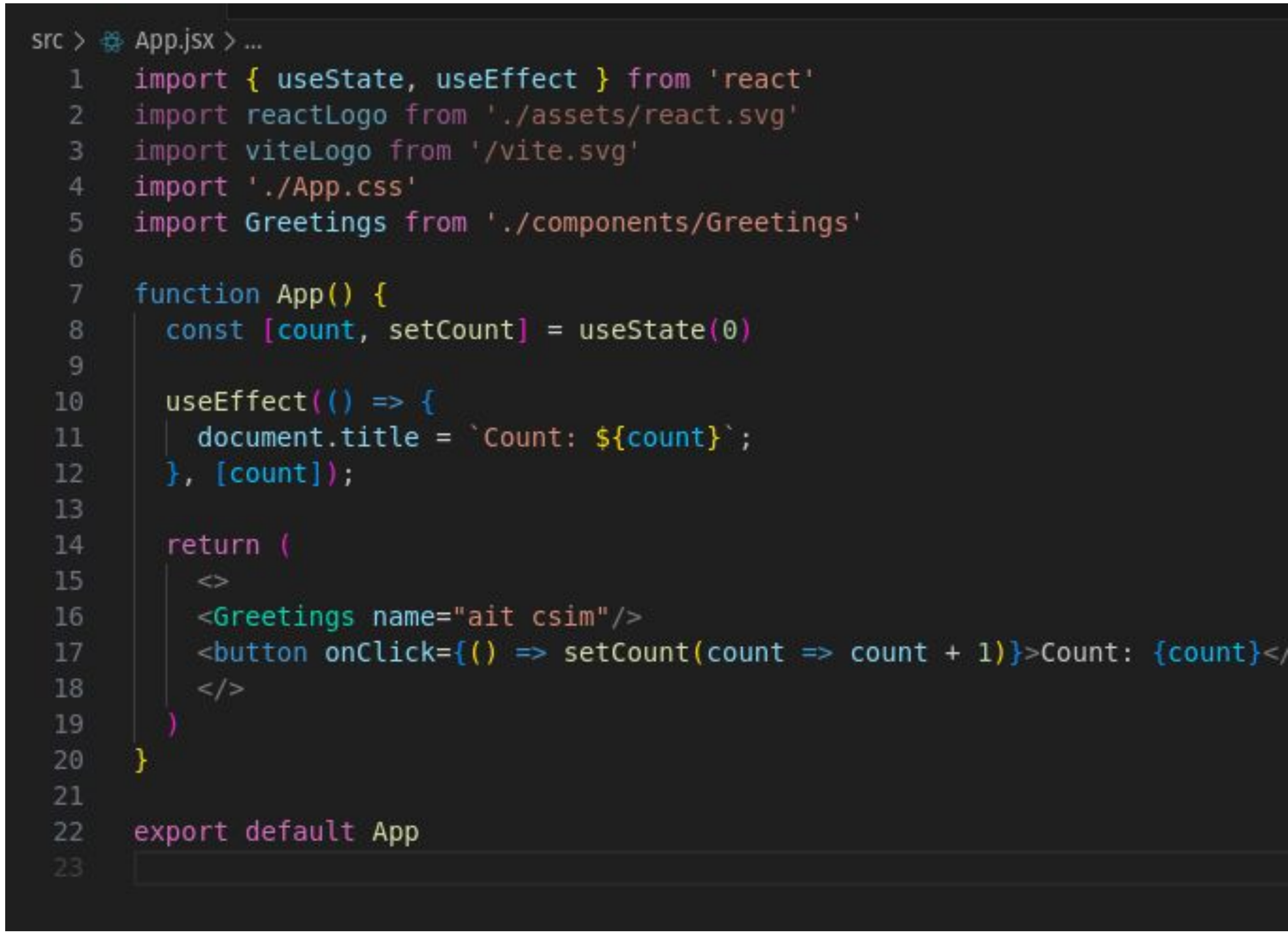


```
import { useState, useEffect } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'
import Greetings from './components/Greetings'

function App() {
  const [count, setCount] = useState(0)

  useEffect(() => {
    document.title = `Count: ${count}`;
  }, [count]);

  return (
    <>
    <Greetings name="ait csim"/>
    <button onClick={() => setCount(count => count + 1)}>Count: {count}</
    </>
  )
}

export default App

```

React Hook: useEffect

Everytime the variable count changes, the react hook useEffect is triggered.

Lets test by removing the dependency count from useEffect and check again. You will see it runs twice, on development mode only due to react being on strict mode.



```jsx
import { useState, useEffect } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'
import Greetings from './components/Greetings'

function App() {
  const [count, setCount] = useState(0)

  useEffect(() => {
    document.title = `Count: ${count}`;
  }, [count]);

  return (
    <>
      <Greetings name="ait csim"/>
      <button onClick={() => setCount(count => count + 1)}>Count: {count}</
    </>
  )
}

export default App
```

Promise

A **Promise** is a JavaScript object that represents **the eventual result of an asynchronous operation**.

- It can be in **one of three states**:
  - **Pending** – The async operation hasn't finished yet
  - **Fulfilled** – The operation completed successfully.
  - **Rejected** – The operation failed
- Use a simple fetch example:
  - Start request → Pending
  - Response arrives → Fulfilled → Render data
  - Error → Rejected → Show error message

async/await is really just syntax sugar for Promises.

Asynchronous Function (async… await)

An async function is a function that:

1. Runs asynchronously (doesn’t block the rest of your code).
2. Always **returns a promise**.
3. Can use the **await** keyword to “pause” execution until a promise resolves.

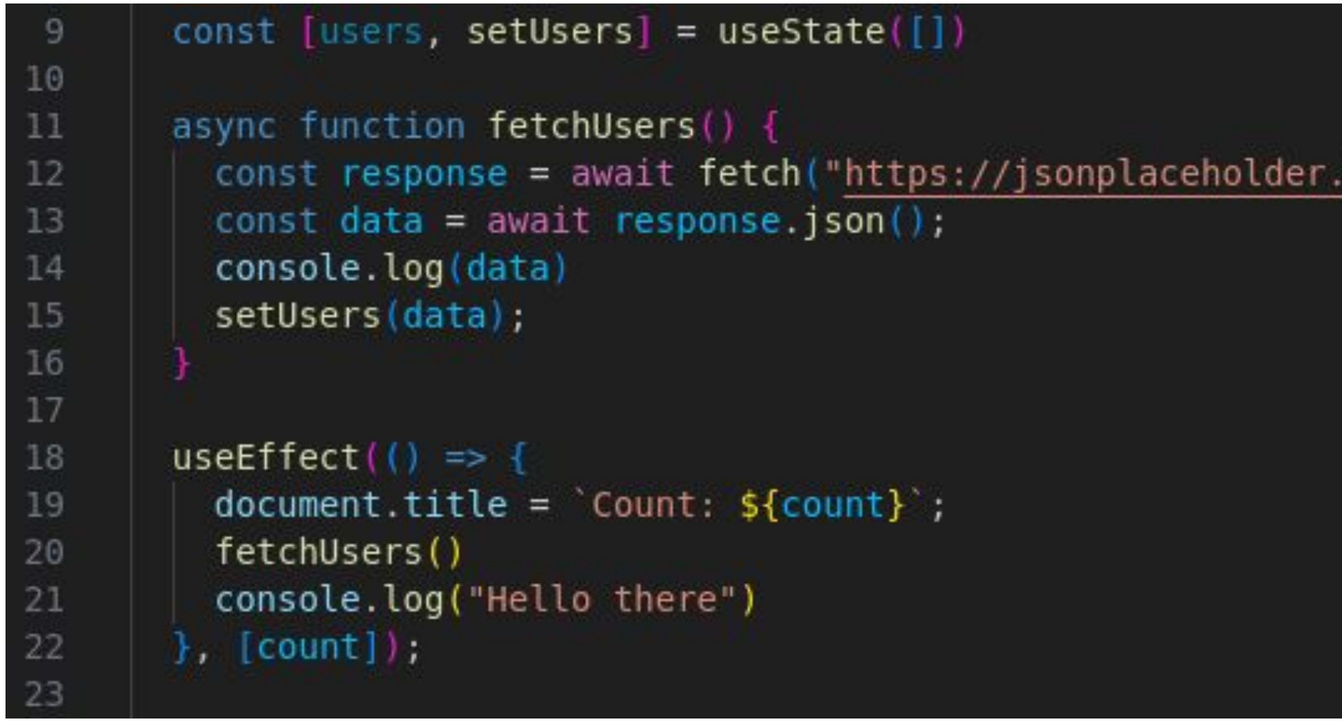
React is all about **rendering UI fast**.
If you do **long-running tasks** (like API requests) **synchronously**, the UI would freeze.

Async functions allow you to:

- Fetch data **without freezing the UI**.
- Handle API requests **cleanly** with await instead of chaining .then() calls.
- Integrate with useEffect to **load data when a component mounts**.

Asynchronous Function (async… await)

```
import { useState, useEffect } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'
import Greetings from './components/Greetings'

function App() {
 const [count, setCount] = useState(0)
 const [users, setUsers] = useState([])

 async function fetchUsers() {
   const response = await
fetch("https://jsonplaceholder.typicode.com/users");
   const data = await response.json();
   console.log(data)
   setUsers(data);
 }

 useEffect(() => {
   document.title = `Count: ${count}`;
   fetchUsers()
   console.log("Hello there")
 }, [count]);
```



```
const [users, setUsers] = useState([])

async function fetchUsers() {
  const response = await fetch("https://jsonplaceholder.
  const data = await response.json();
  console.log(data)
  setUsers(data);
}

useEffect(() => {
  document.title = `Count: ${count}`;
  fetchUsers()
  console.log("Hello there")
}, [count]);
```

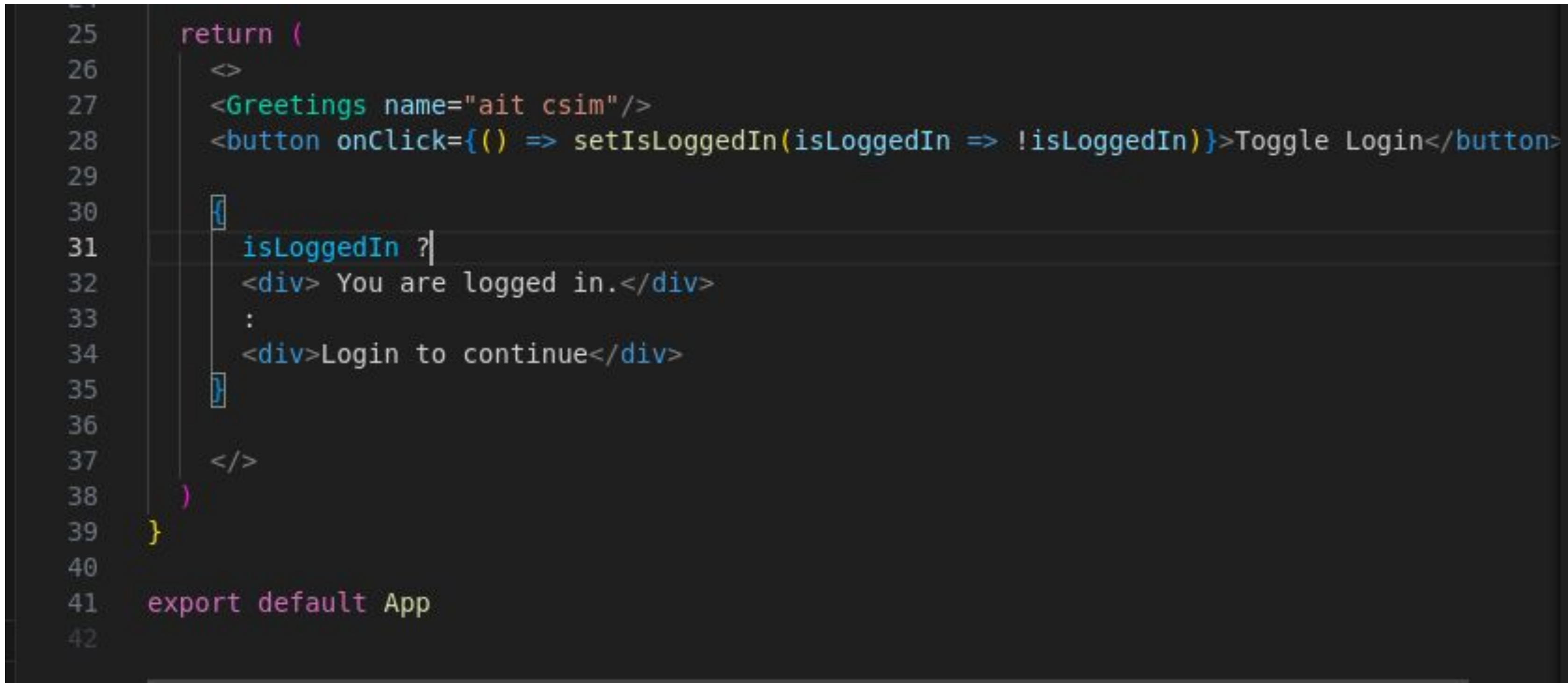
Check the console and see why you see “hello there” before the data. This is async.

Ternary Operator: condition ? ifTrue : ifFalse

Example 1: **Simple Show/Hide**

Add a useState hook for isLoggedIn. Toggle the state when button is clicked.

```
{
  isLoggedIn ?
  <div> You are logged in.</div>
  :
  <div>Login to continue</div>
}
```

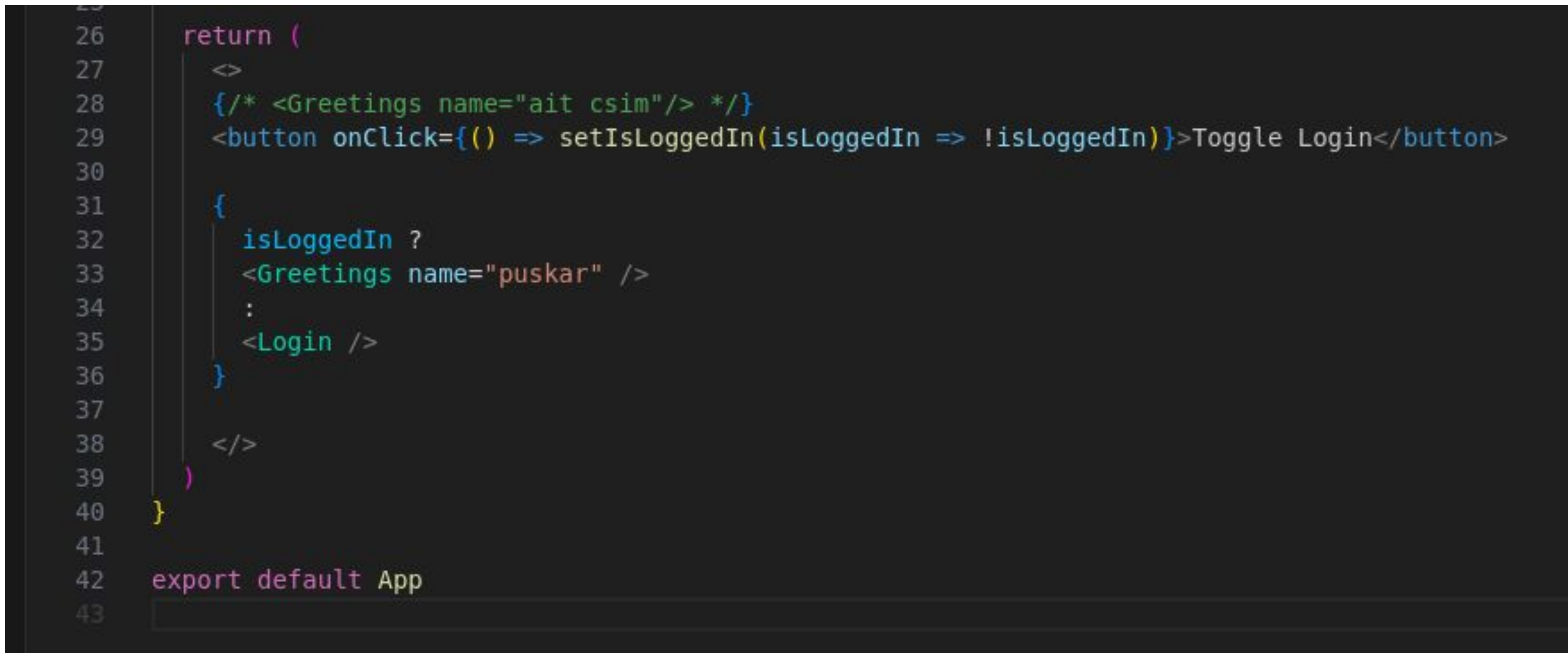


```
  return (
    <>
    <Greetings name="ait csim"/>
    <button onClick={() => setIsLoggedIn(isLoggedIn => !isLoggedIn)}>Toggle Login</button>

    {
      isLoggedIn ?
      <div> You are logged in.</div>
      :
      <div>Login to continue</div>
    }

    </>
  )
}

export default App
```

Ternary Operator: condition ? ifTrue : ifFalse

Example 2: **Conditional Component Rendering**

Similar to the first example, instead of div, use component. This is useful when handling the overall state of the application.



```
  return (
    <>
    {/* <Greetings name="ait csim"/> */}
    <button onClick={() => setIsLoggedIn(isLoggedIn => !isLoggedIn)}>Toggle Login</button>

    {
      isLoggedIn ?
      <Greetings name="puskar" />
      :
      <Login />
    }

    </>
  )
}

export default App
```

React Router Dom

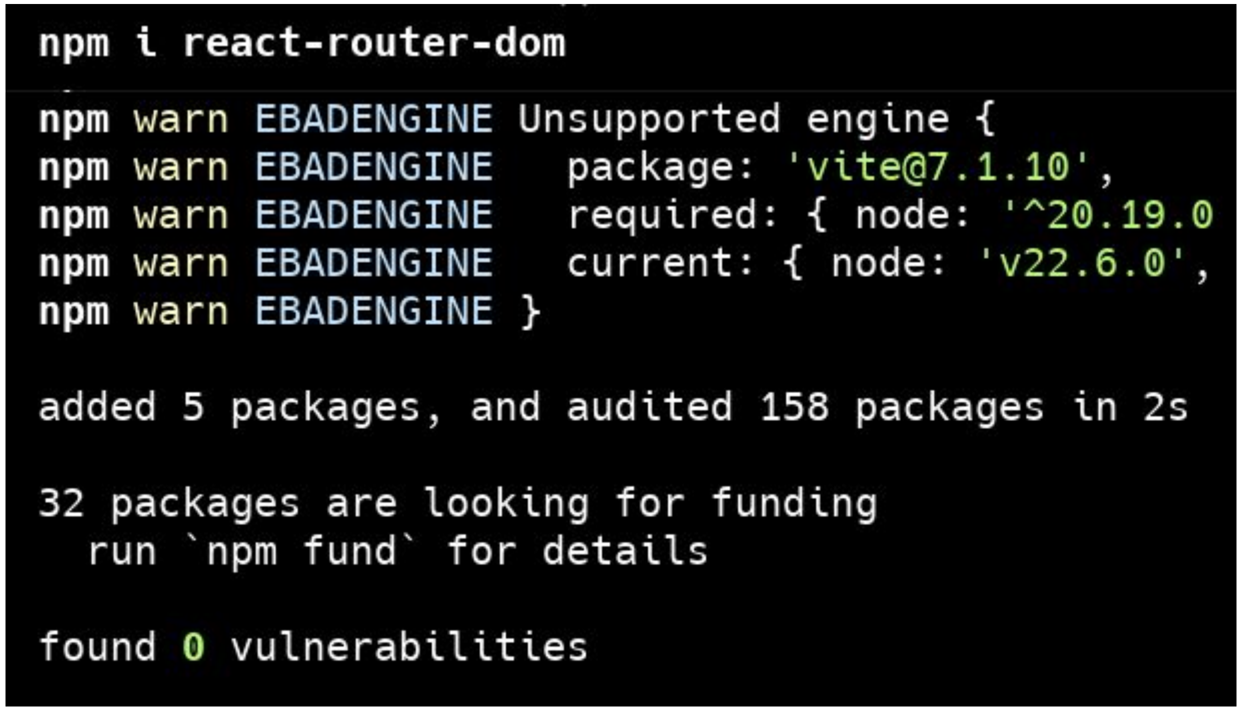
React does not support routing itself.

So, we use this third party package:

react-router-dom

Install using:

- npm install react-router-dom

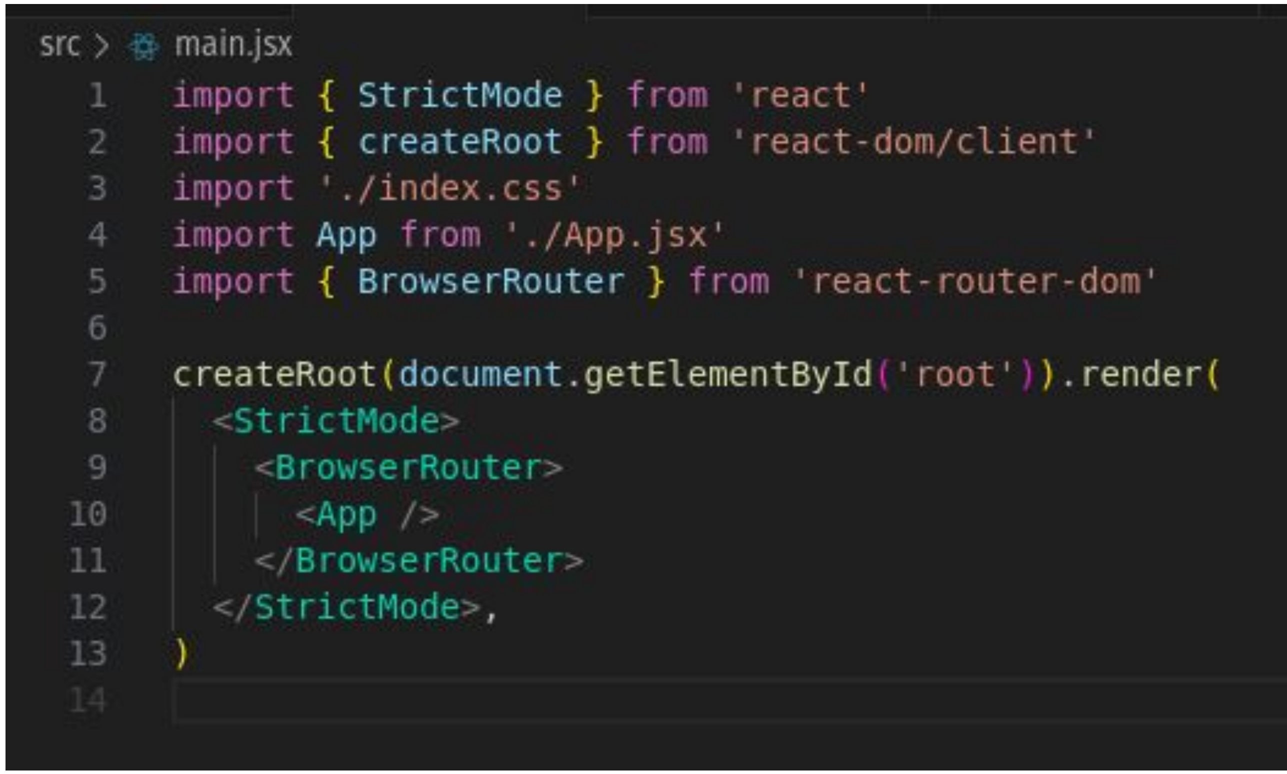


```
npm i react-router-dom

npm warn EBADENGINE Unsupported engine {
npm warn EBADENGINE   package: 'vite@7.1.10',
npm warn EBADENGINE   required: { node: '^20.19.0
npm warn EBADENGINE   current: { node: 'v22.6.0',
npm warn EBADENGINE }

added 5 packages, and audited 158 packages in 2s

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

found 0 vulnerabilities
```

React Router Dom

In **main.jsx** (or index.jsx)

❖ Wrap the entire react app with the BrowserRouter

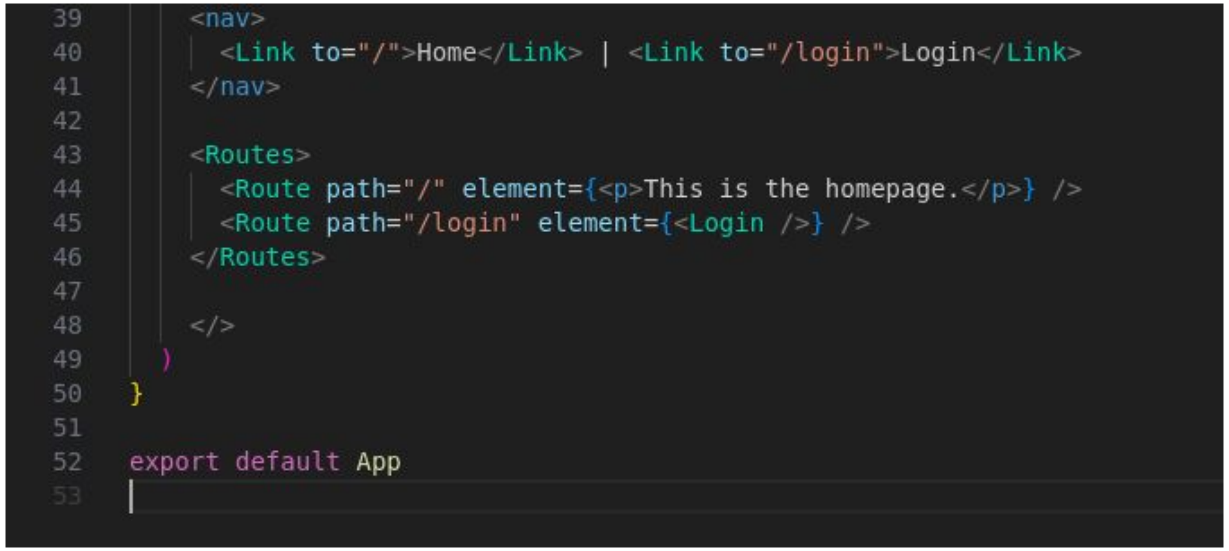


src > main.jsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'
import { BrowserRouter } from 'react-router-dom'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </StrictMode>,
)
```

React Router Dom

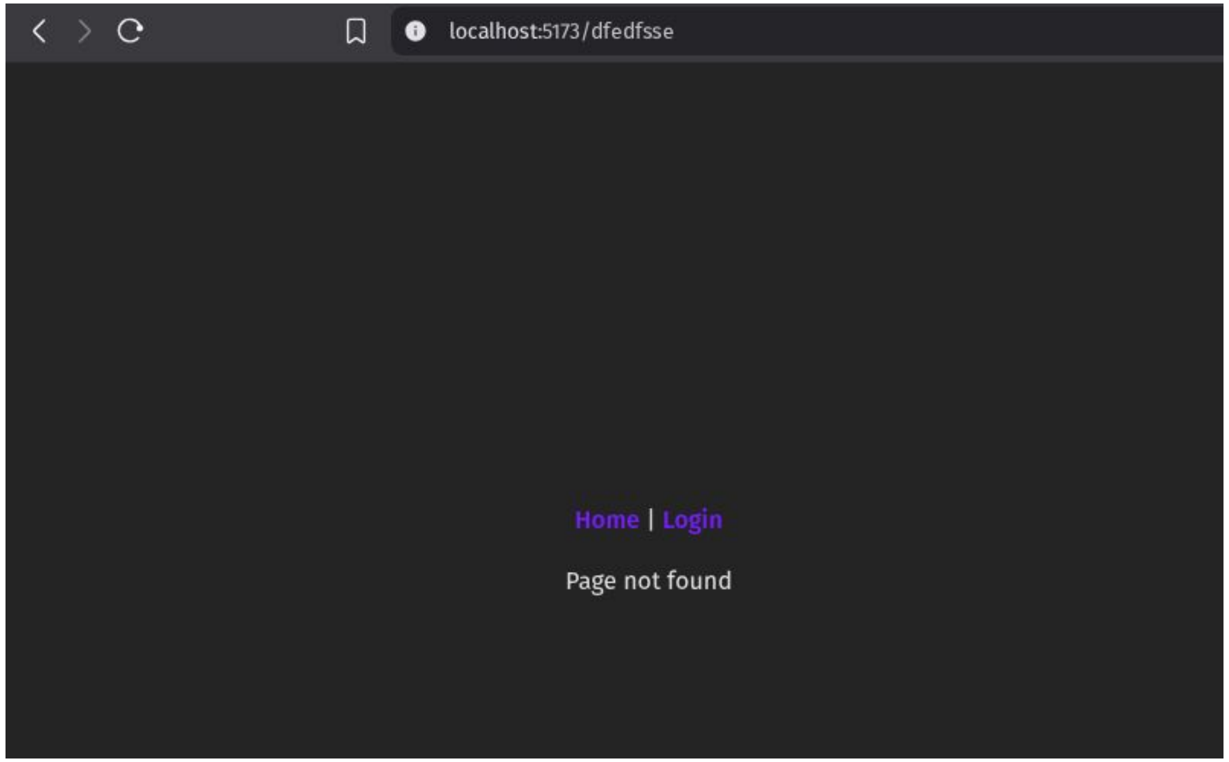
- Import Routes, Link and Route from react-router-dom.
- <BrowserRouter> wraps the app - enables routing.
- <Routes> holds all route definitions.
- <Route path="..." element={...} /> defines a URL and which component to render.
- <Link to="..."> replaces <a> for SPA navigation without page reload.



```
    <nav>
      <Link to="/">Home</Link> | <Link to="/login">Login</Link>
    </nav>

    <Routes>
      <Route path="/" element={<p>This is the homepage.</p>} />
      <Route path="/login" element={<Login />} />
    </Routes>

    </>
  )
}

export default App
```

React Router Dom

```
import { useState, useEffect } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'
import Greetings from './components/Greetings'
import Login from './components/Login'
import { Routes, Route, Link } from
'react-router-dom'

function App() {
 const [count, setCount] = useState(0)
 const [users, setUsers] = useState([])
```


localhost:5173/dfedfsse

Home | Login

Page not found

Topics Covered

- ❖ Components
- ❖ Props
- ❖ React Hook - useState
- ❖ React Hook - useEffect
- ❖ Promises & Async functions
- ❖ Ternary Operator
- ❖ Routing