5.1 | React Deep dive

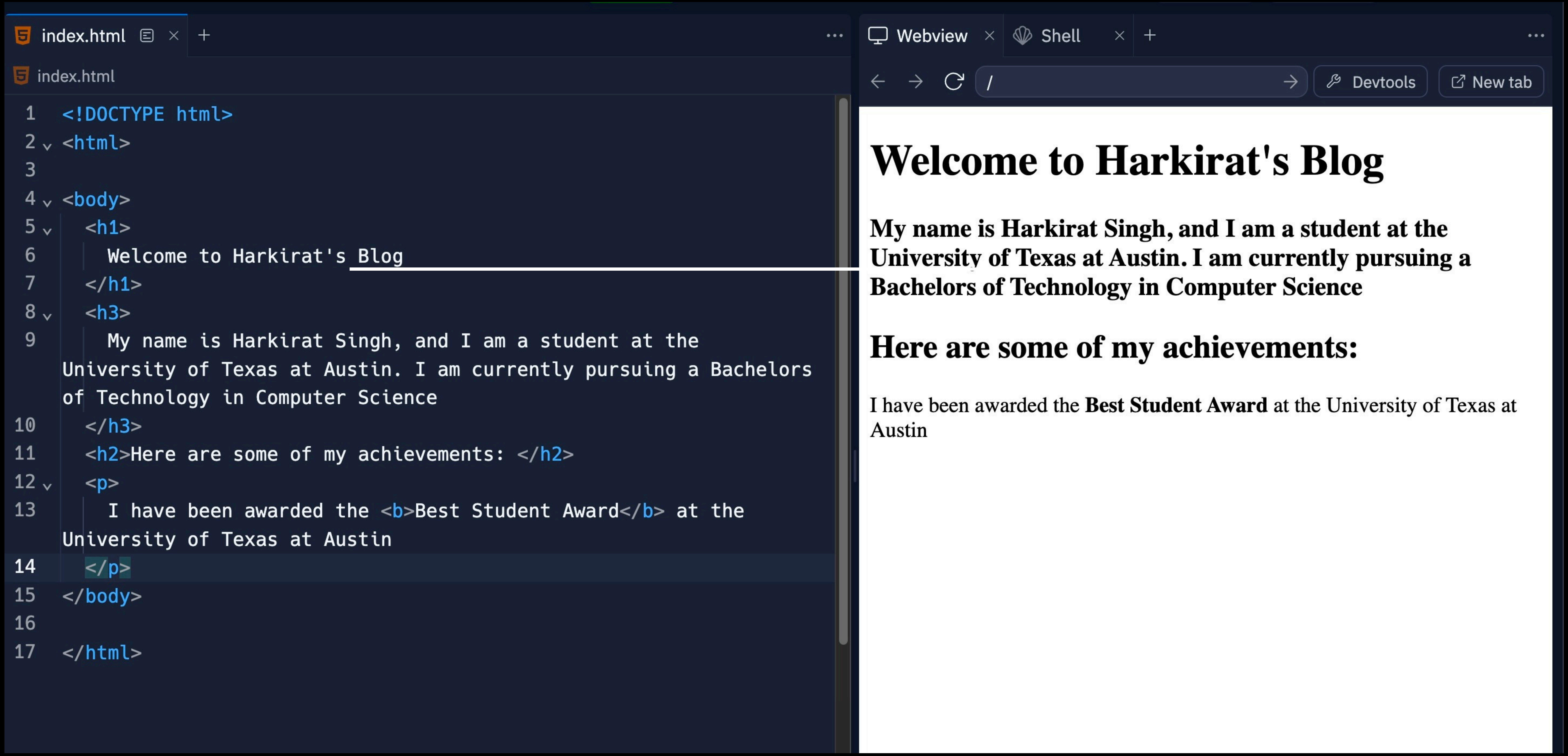
Understanding React from examples

Jargon we’ll learn today

Jsx, class vs className, static vs dynamic websites,
State, components, re-rendering

Why do you need React?

For static websites, you don't!



```html
<!DOCTYPE html>
<html>

<body>
  <h1>
    Welcome to Harkirat's Blog
  </h1>
  <h3>
    My name is Harkirat Singh, and I am a student at the University of Texas at Austin. I am currently pursuing a Bachelors of Technology in Computer Science
  </h3>
  <h2>Here are some of my achievements: </h2>
  <p>
    I have been awarded the <b>Best Student Award</b> at the University of Texas at Austin
  </p>
</body>

</html>
```

Welcome to Harkirat's Blog

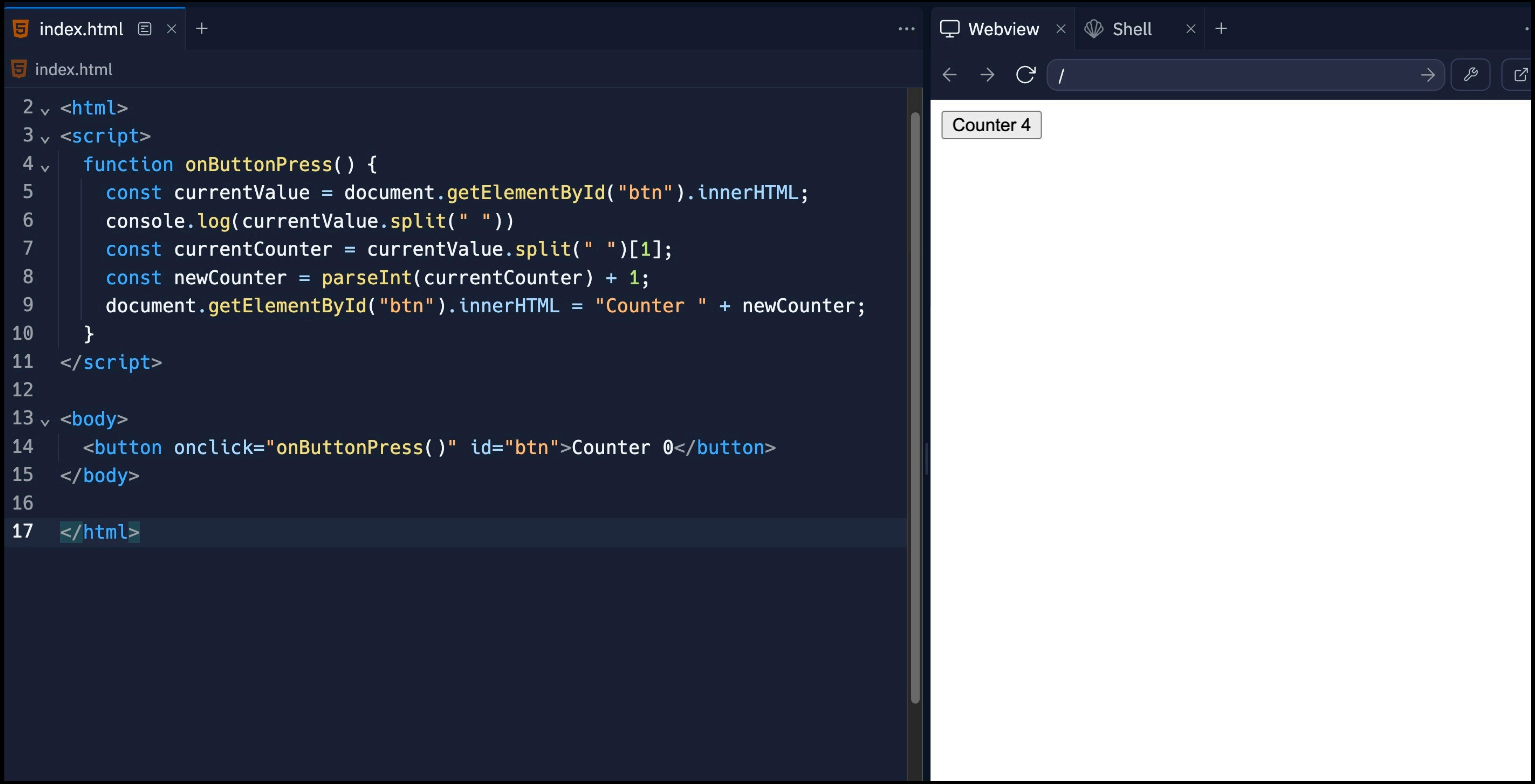
My name is Harkirat Singh, and I am a student at the University of Texas at Austin. I am currently pursuing a Bachelors of Technology in Computer Science

Here are some of my achievements:

I have been awarded the **Best Student Award** at the University of Texas at Austin

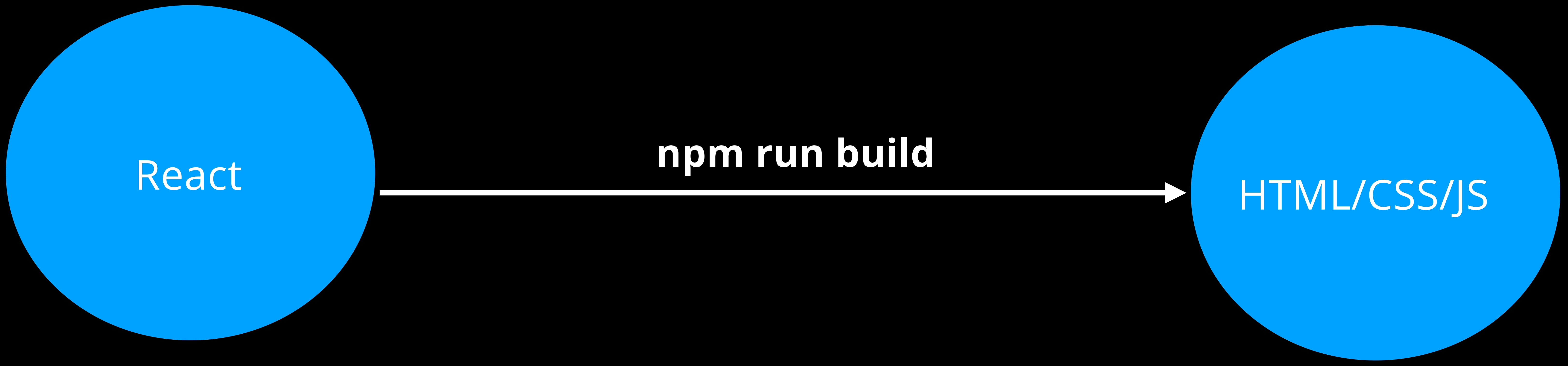
Why do you need React?

For dynamic websites, these libraries make it easier to do DOM manipulation

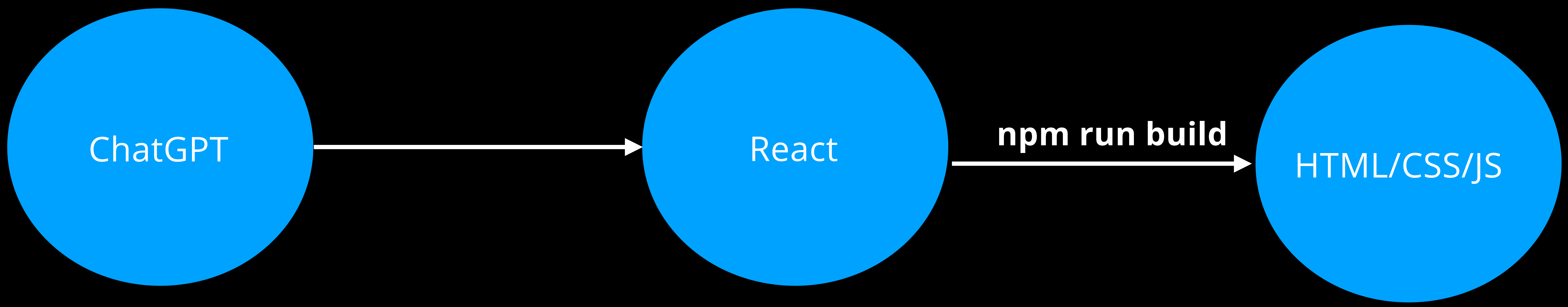


```html
<html>
<script>
  function onButtonPress() {
    const currentValue = document.getElementById("btn").innerHTML;
    console.log(currentValue.split(" "))
    const currentCounter = currentValue.split(" ")[1];
    const newCounter = parseInt(currentCounter) + 1;
    document.getElementById("btn").innerHTML = "Counter " + newCounter;
  }
</script>

<body>
  <button onclick="onButtonPress()" id="btn">Counter 0</button>
</body>

</html>
```

React is just an easier way to write normal HTML/CSS/JS
It’s a new syntax, that under the hood gets converted to HTML/CSS/JS



Just how ChatGPT is an easier way to write code, React is an easier way to write HTML/CSS

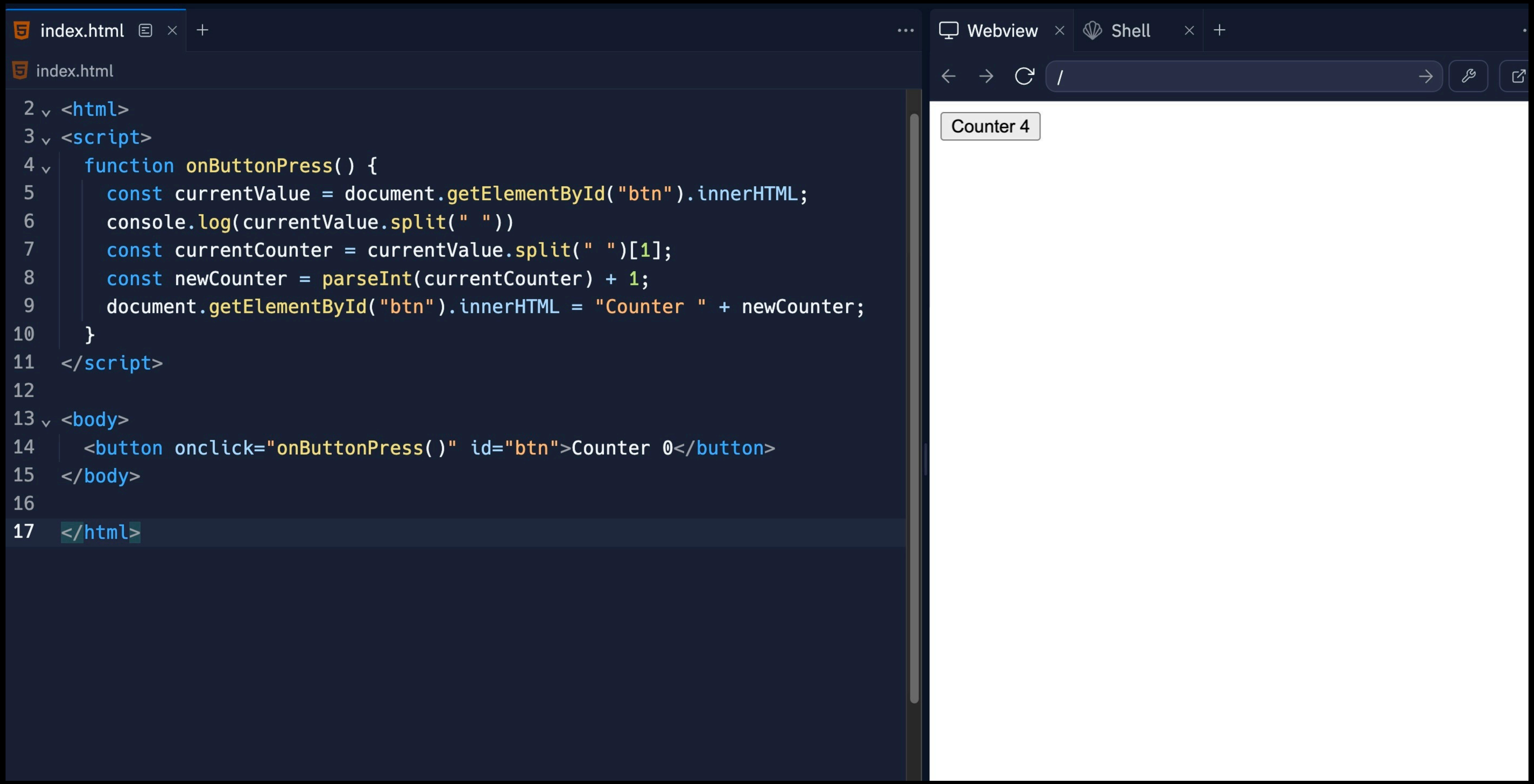


Why React?

People realised it’s harder to do DOM manipulation the conventional way
There were libraries that came into the picture that made it slightly easy, but still for a very big app it’s very hard (JQuery)
Eventually, VueJS/React created a new syntax to do frontends
Under the hood, the react compiler convert your code to HTML/CSS/JS

Let’s look at a simple example

Problem with this approach
1. **Too much code you have to write as the developer**
2. **As your app scales (todo app for eg), this gets harder and harder.**



```html
<html>
<script>
  function onButtonPress() {
    const currentValue = document.getElementById("btn").innerHTML;
    console.log(currentValue.split(" "))
    const currentCounter = currentValue.split(" ")[1];
    const newCounter = parseInt(currentCounter) + 1;
    document.getElementById("btn").innerHTML = "Counter " + newCounter;
  }
</script>

<body>
  <button onclick="onButtonPress()" id="btn">Counter 0</button>
</body>

</html>
```

https://gist.github.com/hkirat/0c22122a9485d4d592b92677570e6be8

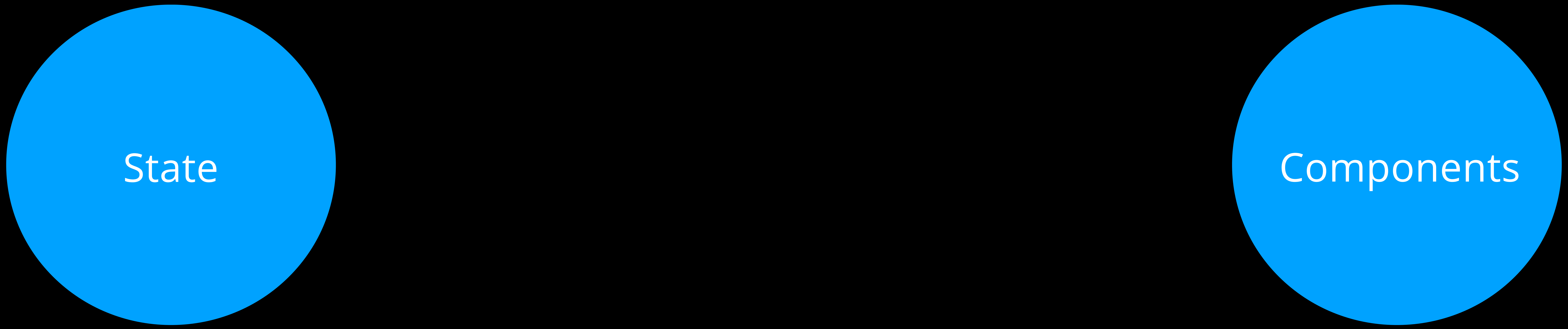
Some react jargon

Some react jargon

To create a react app, you usually need to worry about two things

Some react jargon

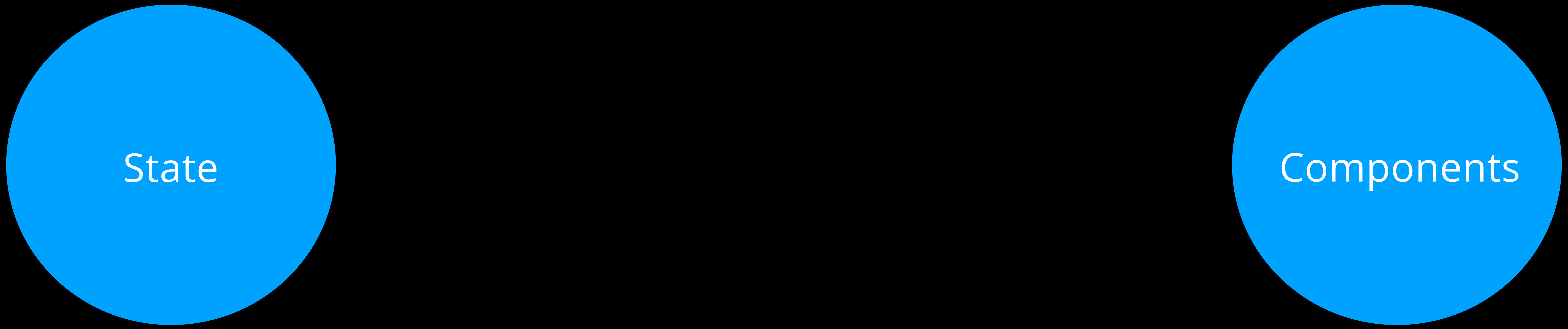
To create a react app, you usually need to worry about two things



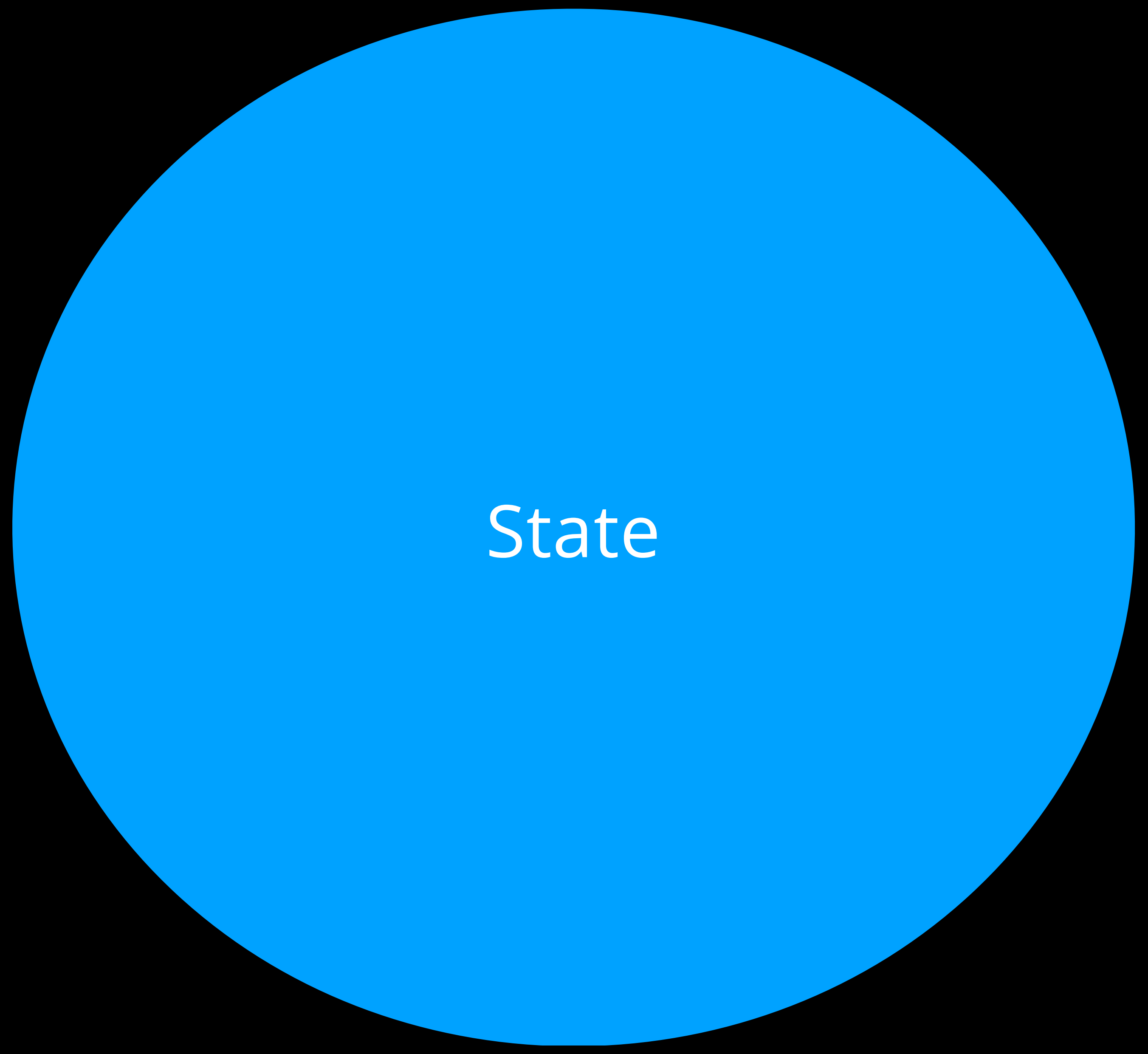
Some react jargon

To create a react app, you usually need to worry about two things

Creators of frontend frameworks realised that all websites can effectively be divided into two parts



State/Components/Re-rendering

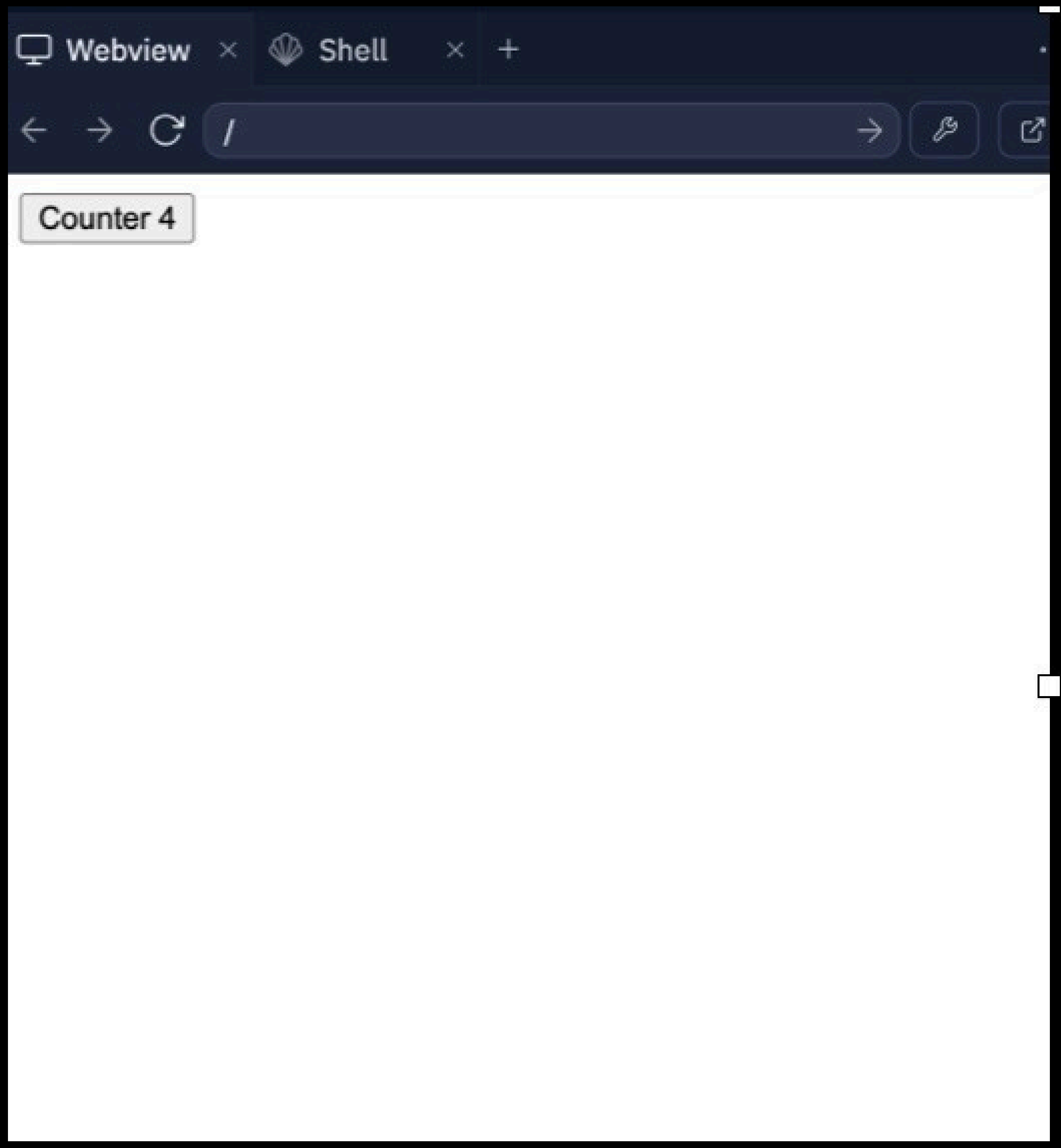


An object that represents the current state of the app

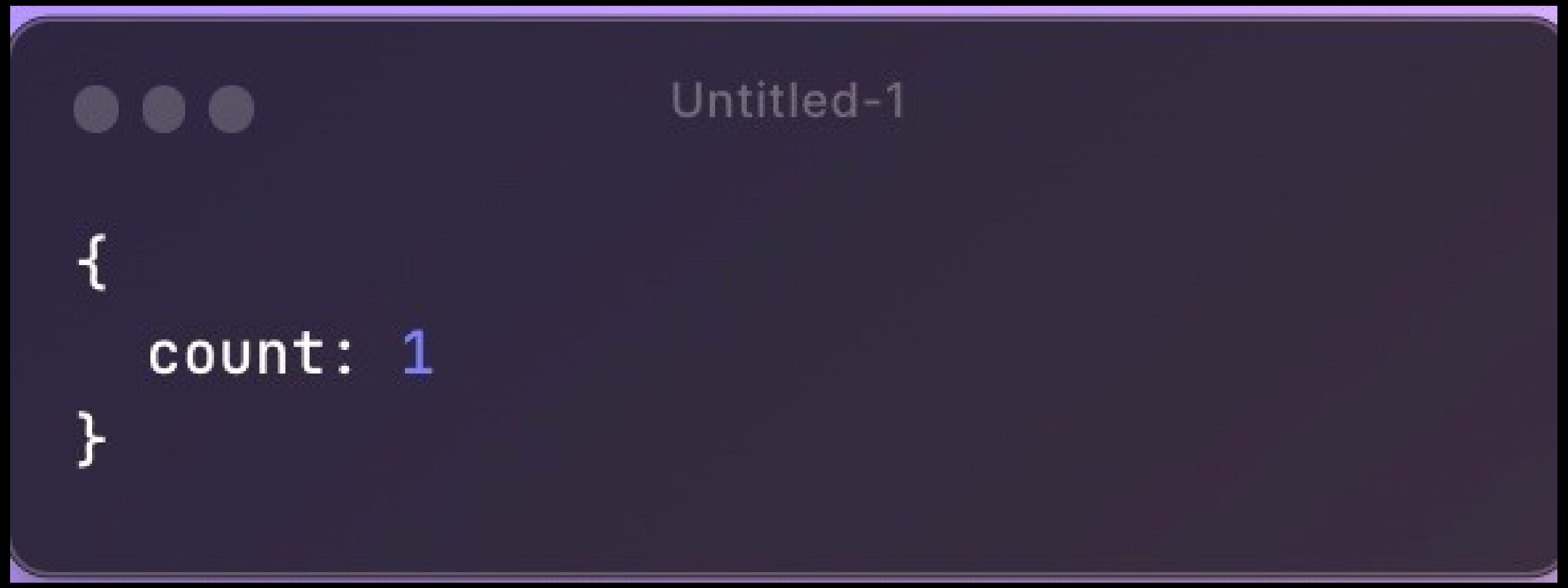
It represents the dynamic things in your app (things that change)

For example, the value of the counter

State/Components/Re-rendering



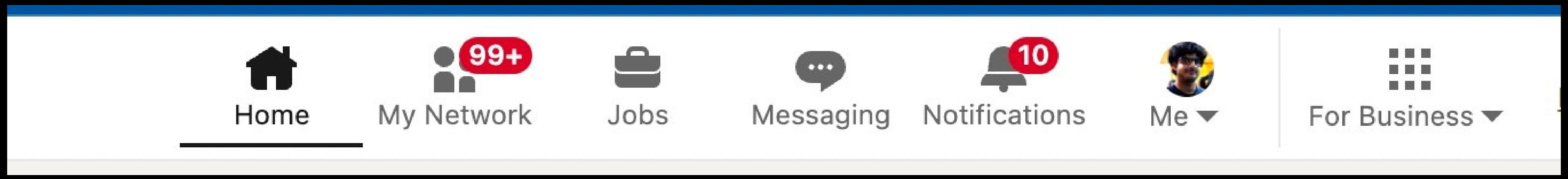
For the counter app, it could look something like this -

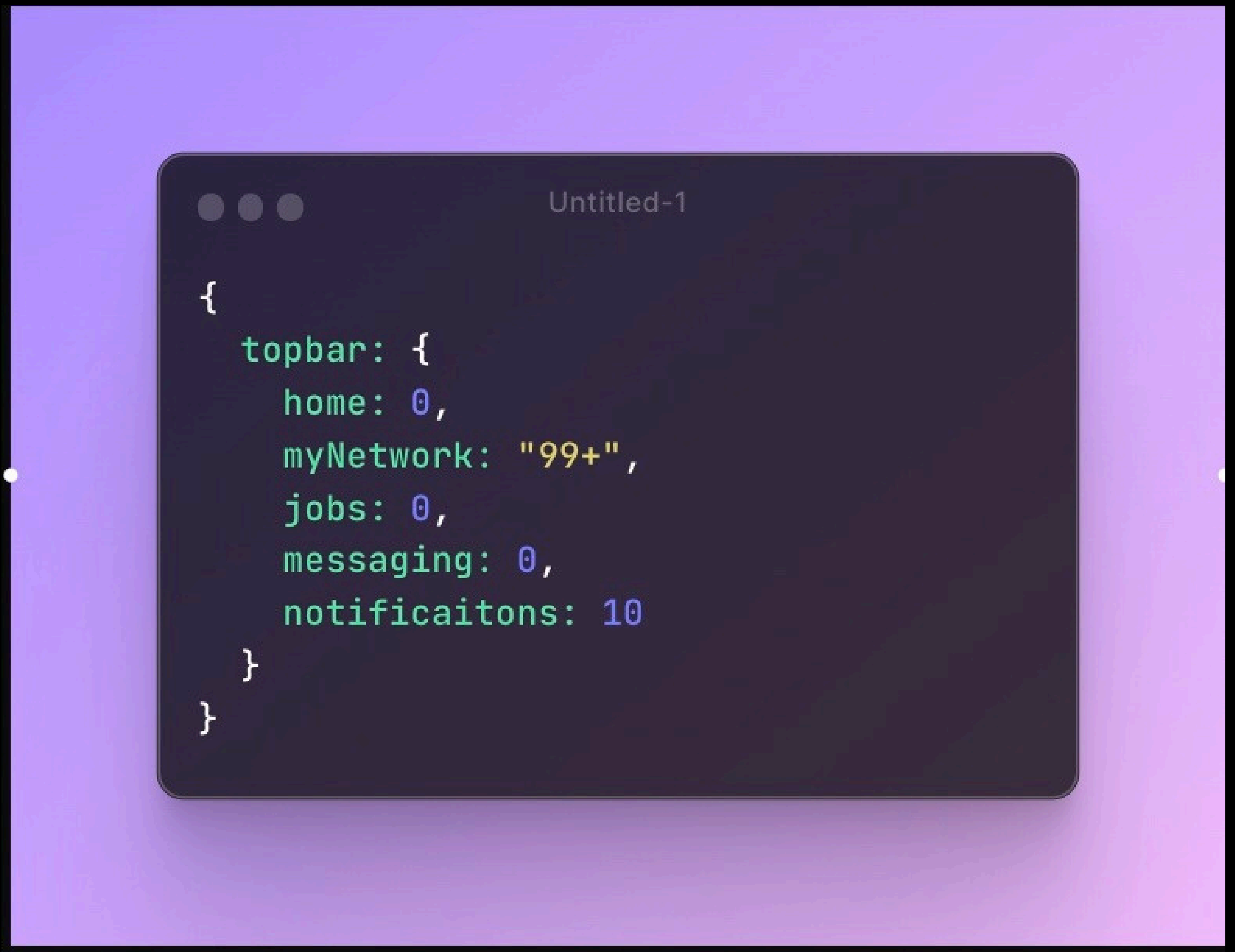


```
{
  count: 1
}
```

State/Components/Re-rendering

For the LinkedIn Topbar, it could be something like this -





```
{
  topbar: {
    home: 0,
    myNetwork: "99+",
    jobs: 0,
    messaging: 0,
    notificaitons: 10
  }
}
```

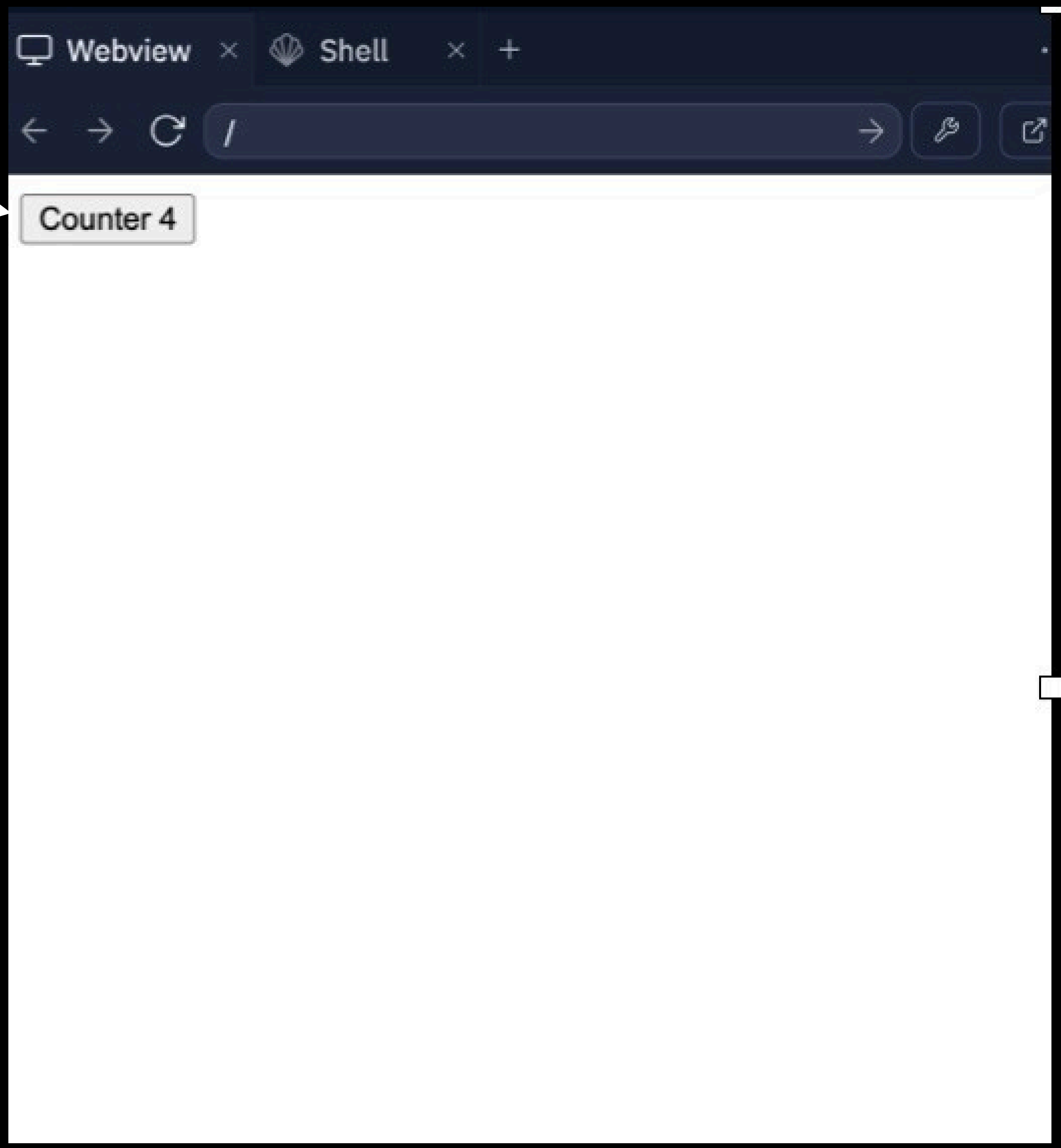
State/Components/Re-rendering

Components

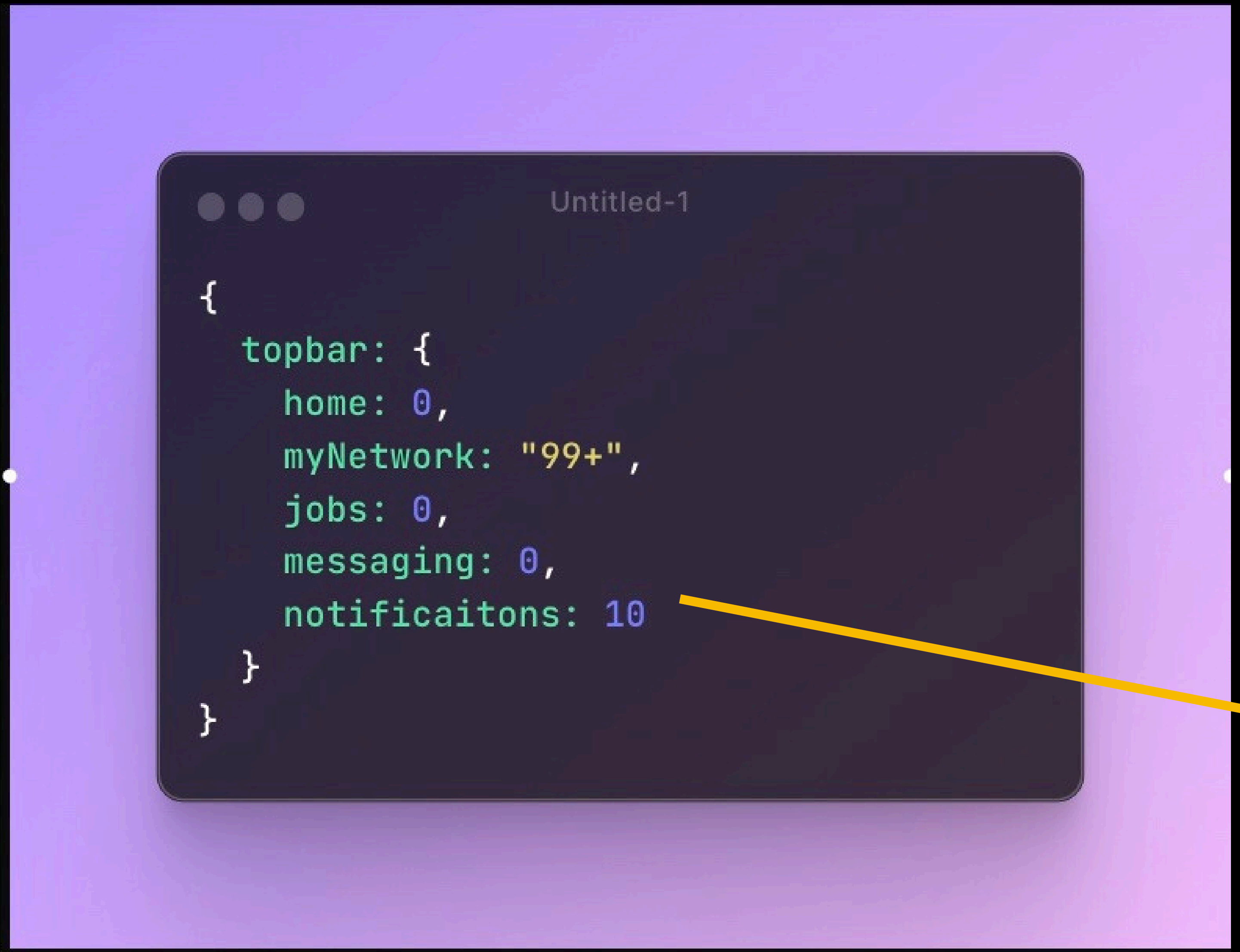
How a DOM element should render, given a state
It is a re-usable, dynamic, HTML snippet that changes given the state

State/Components/Re-rendering

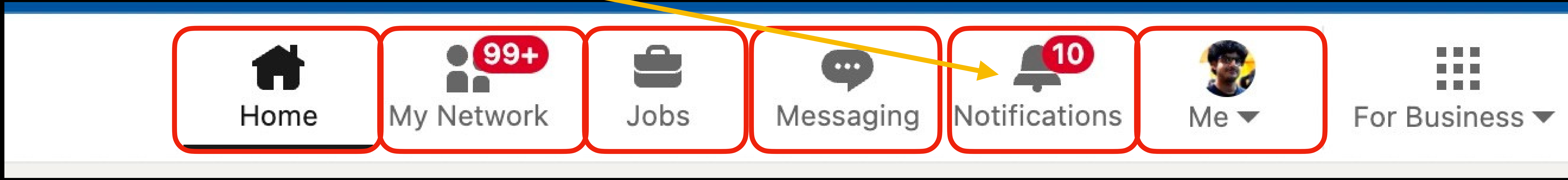
This button is a component
It takes the state (currentCount) as an input
And is supposed to render accordingly



State/Components/Re-rendering

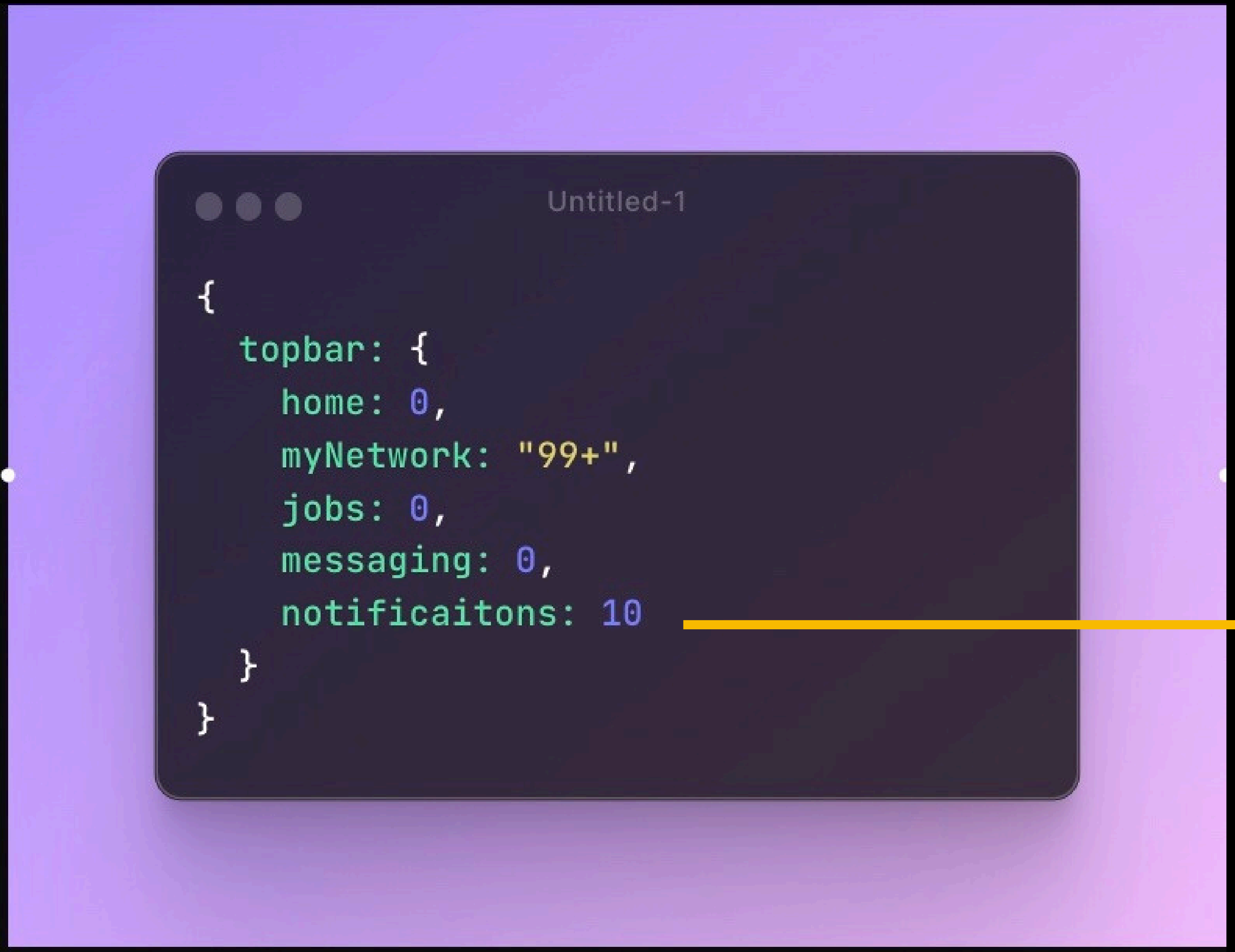


```
{
  topbar: {
    home: 0,
    myNetwork: "99+",
    jobs: 0,
    messaging: 0,
    notificaitons: 10
  }
}
```



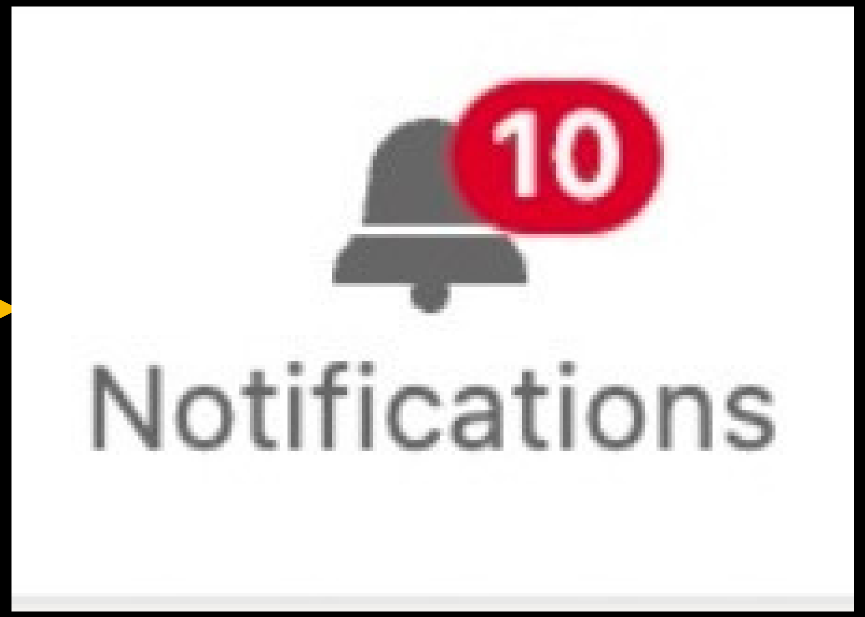
State/Components/Re-rendering

State



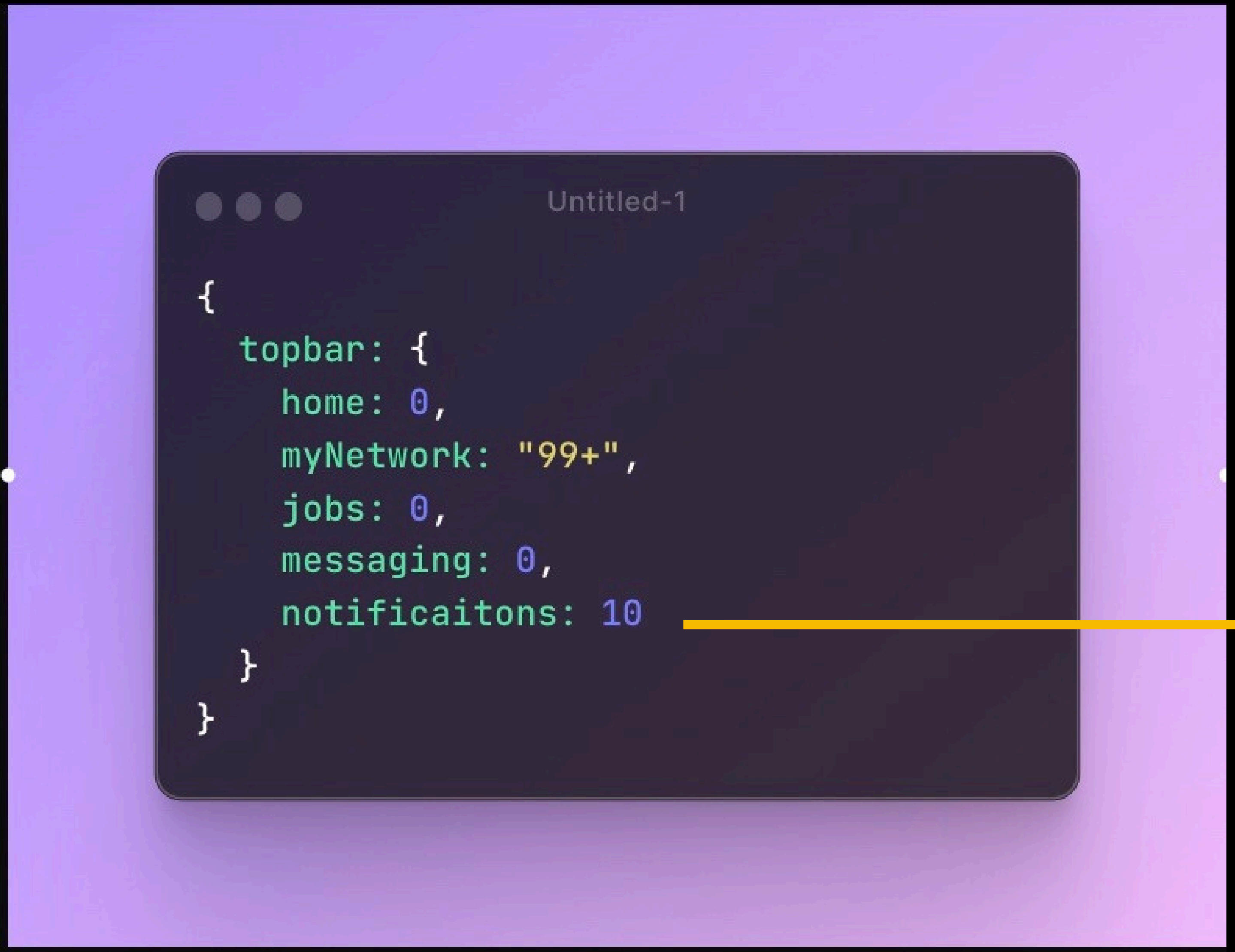
```
Untitled-1
{
  topbar: {
    home: 0,
    myNetwork: "99+",
    jobs: 0,
    messaging: 0,
    notificaitons: 10
  }
}
```

Component



State/Components/Re-rendering

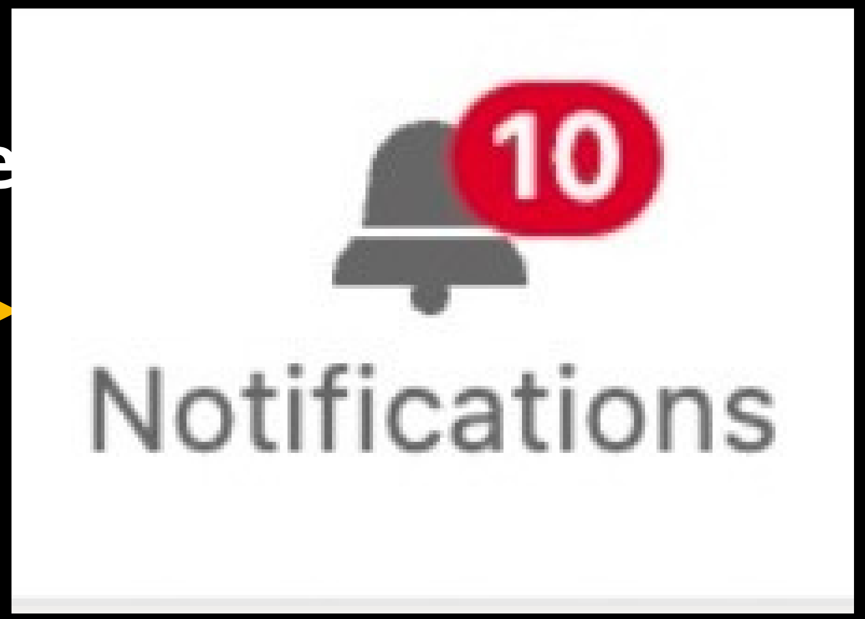
State



```
Untitled-1
{
  topbar: {
    home: 0,
    myNetwork: "99+",
    jobs: 0,
    messaging: 0,
    notificaitons: 10
  }
}
```

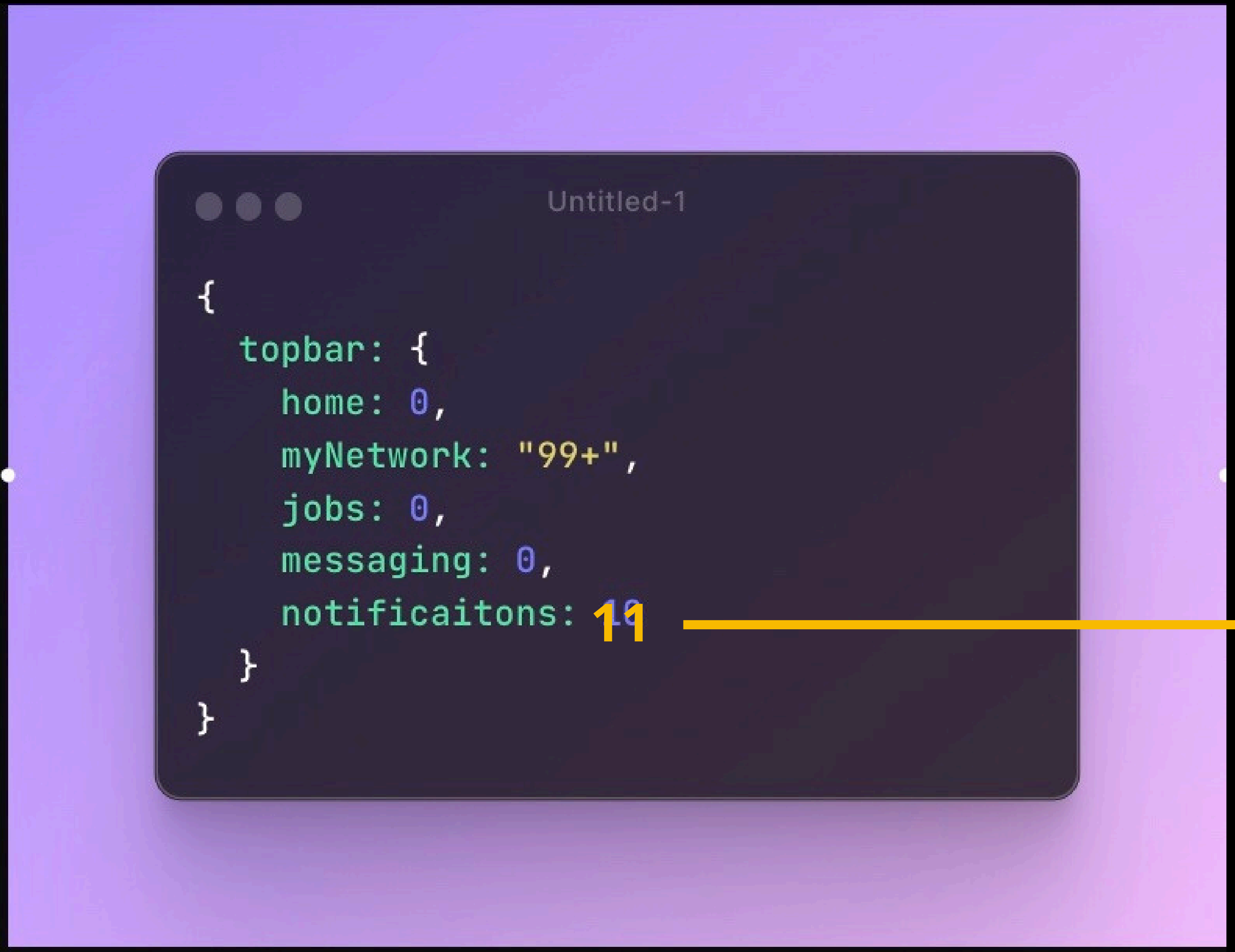
A state change triggers a re-render
A re-render represents the actual DOM being manipulate
when the state changes

Component



State/Components/Re-rendering

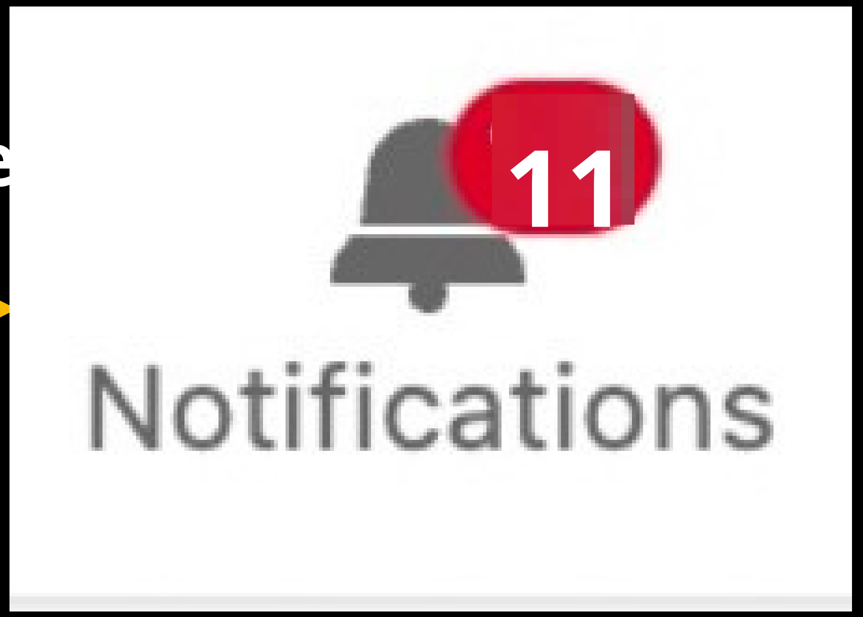
State



```
{
  topbar: {
    home: 0,
    myNetwork: "99+",
    jobs: 0,
    messaging: 0,
    notificaitons: 11
  }
}
```

Component

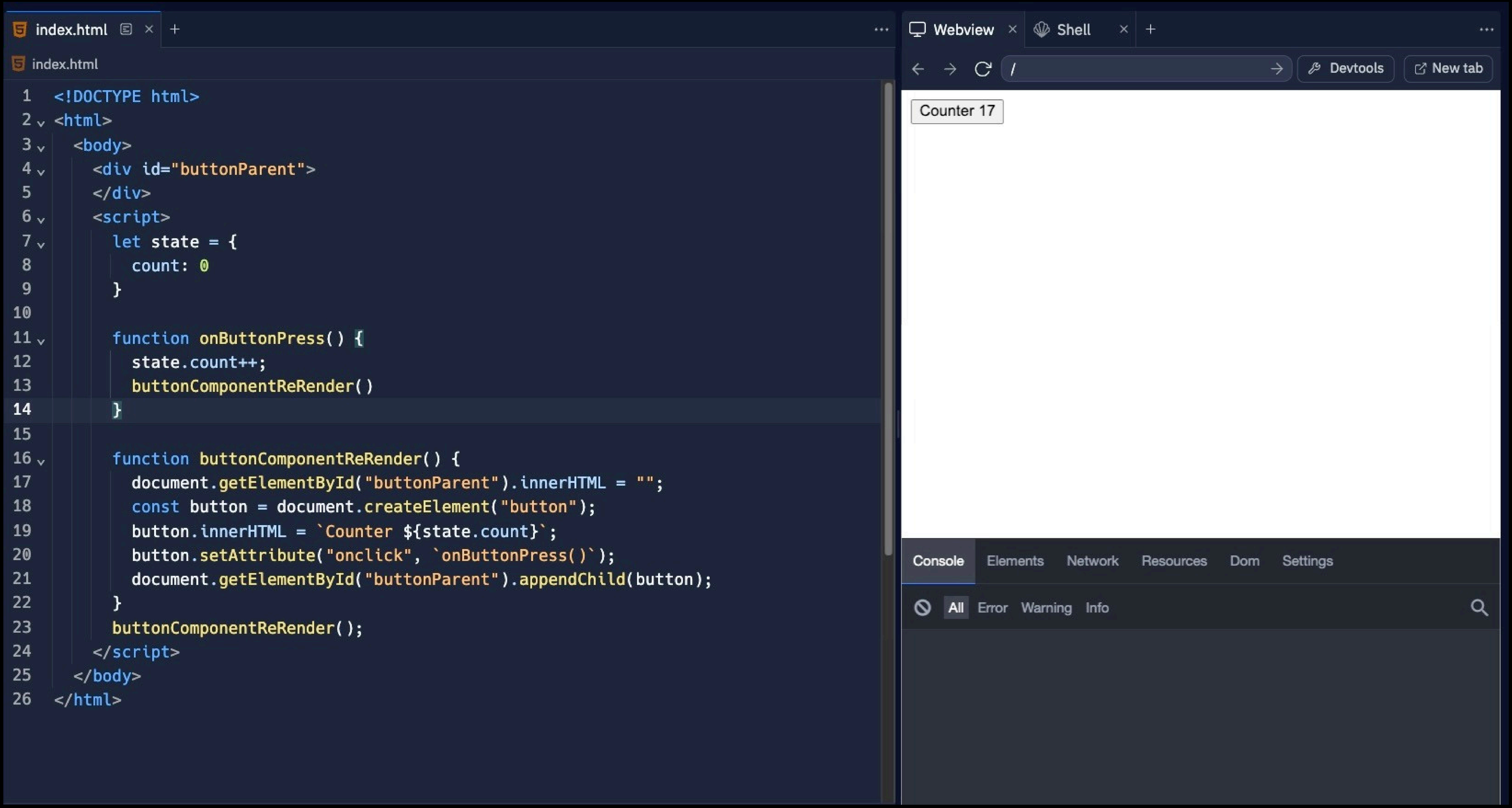
A state change triggers a re-render
A re-render represents the actual DOM being manipulate
when the state changes



State/Components/Re-rendering

You usually have to define all your components once
And then all you have to do is update the state of your app, React takes care of re-rendering your app

Let’s create a counter app using state/components

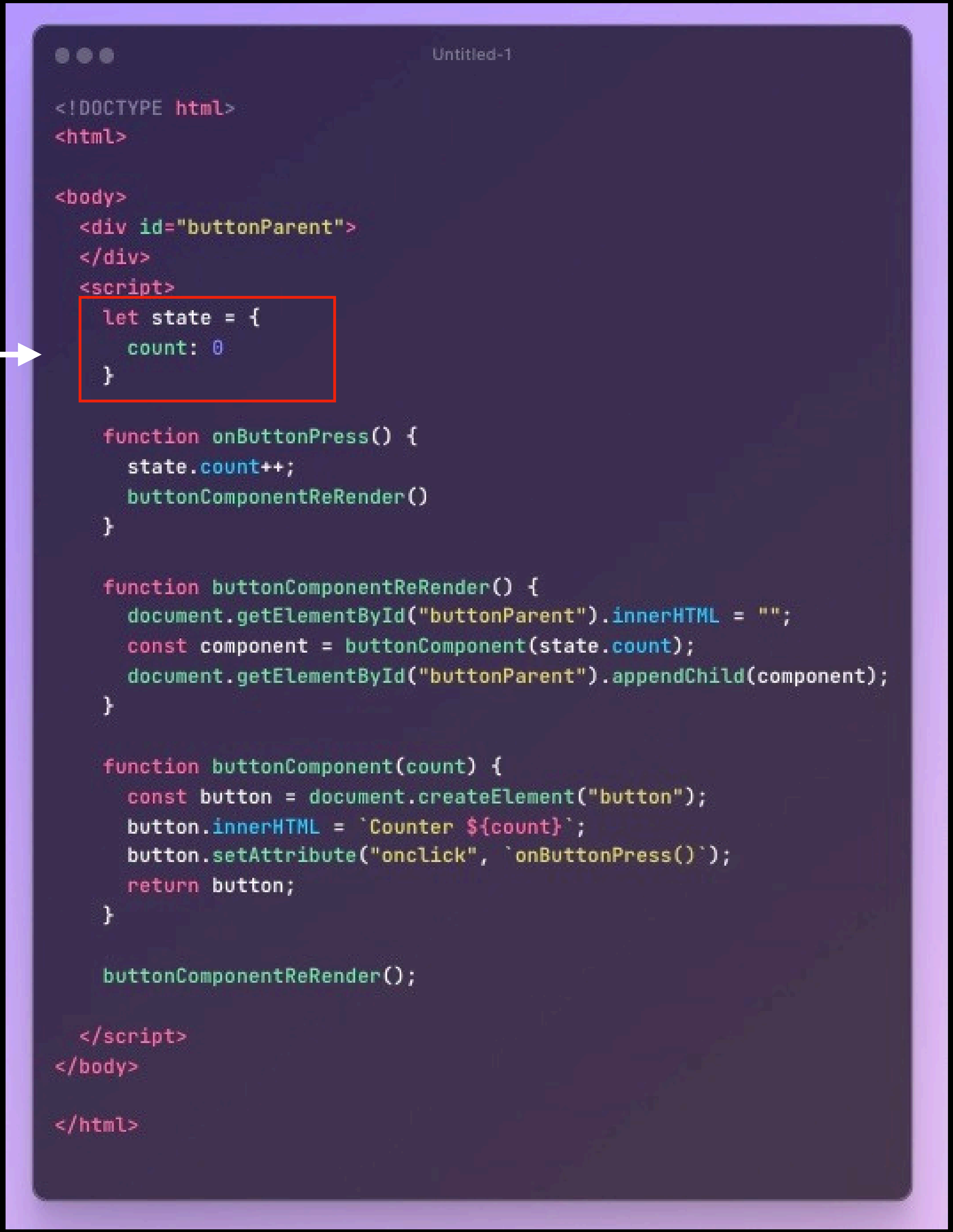


```html
<!DOCTYPE html>
<html>
  <body>
    <div id="buttonParent">
    </div>
    <script>
      let state = {
        count: 0
      }

      function onButtonPress() {
        state.count++;
        buttonComponentReRender()
      }

      function buttonComponentReRender() {
        document.getElementById("buttonParent").innerHTML = "";
        const button = document.createElement("button");
        button.innerHTML = `Counter ${state.count}`;
        button.setAttribute("onclick", `onButtonPress()`);
        document.getElementById("buttonParent").appendChild(button);
      }
      buttonComponentReRender();
    </script>
  </body>
</html>
```

https://gist.github.com/hkirat/c3d98735cec445e718b08f972dda7101

Let’s create a counter app using state/components

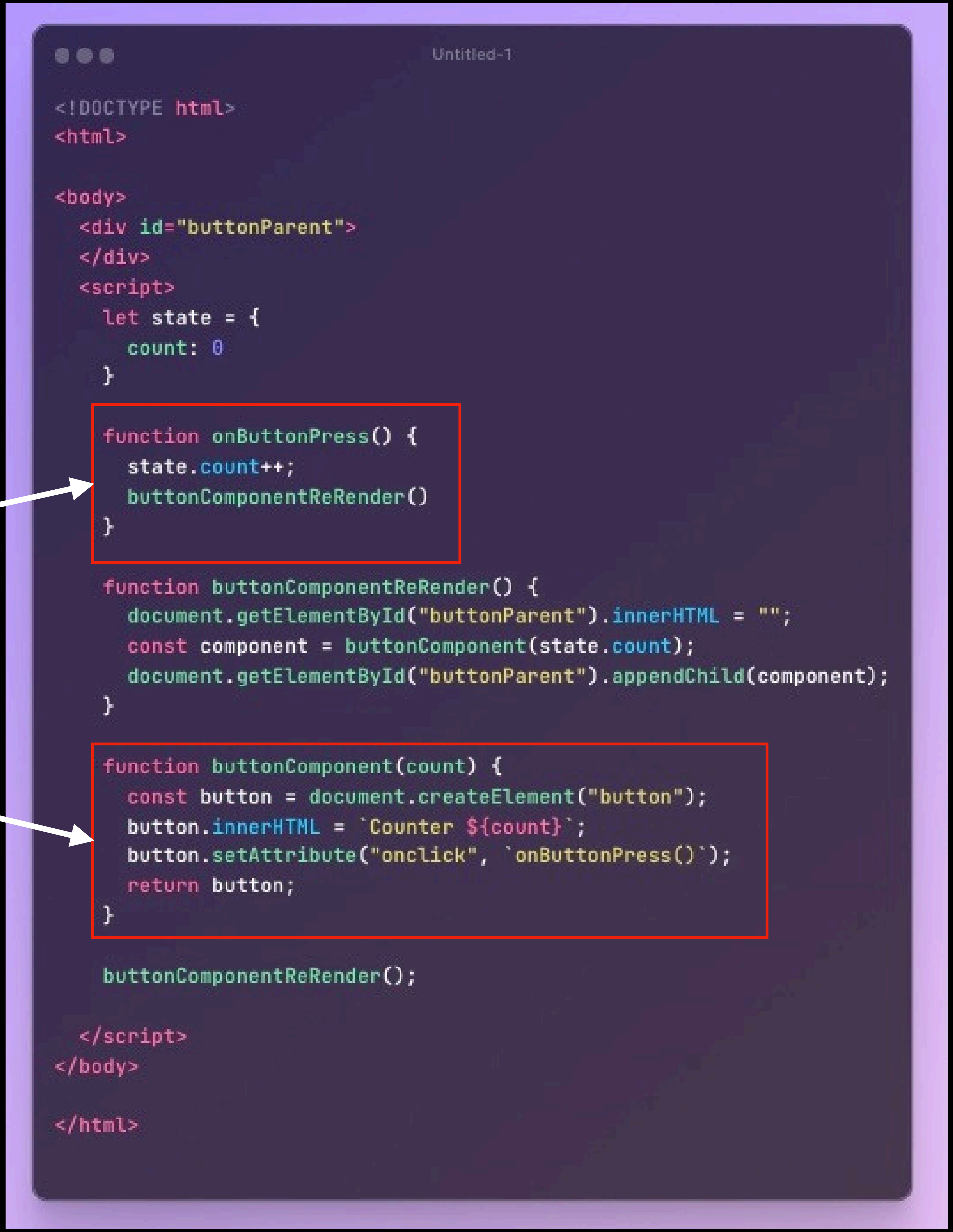
1. State initialisation



```
<!DOCTYPE html>
<html>

<body>
  <div id="buttonParent">
  </div>
  <script>
    let state = {
      count: 0
    }

    function onButtonPress() {
      state.count++;
      buttonComponentReRender()
    }

    function buttonComponentReRender() {
      document.getElementById("buttonParent").innerHTML = "";
      const component = buttonComponent(state.count);
      document.getElementById("buttonParent").appendChild(component);
    }

    function buttonComponent(count) {
      const button = document.createElement("button");
      button.innerHTML = `Counter ${count}`;
      button.setAttribute("onclick", `onButtonPress()`);
      return button;
    }

    buttonComponentReRender();

  </script>
</body>

</html>
```

Let’s create a counter app using state/components

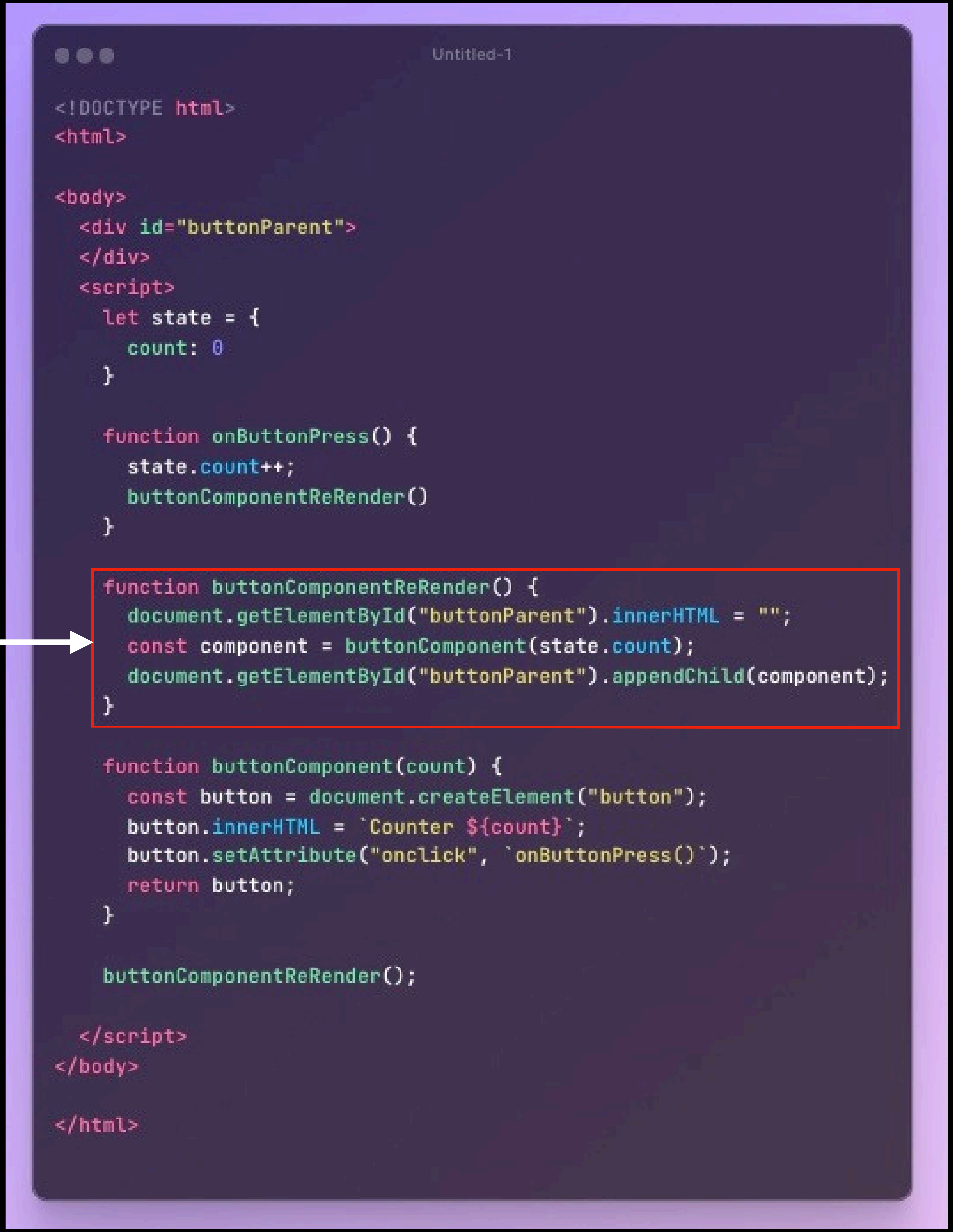
2. Defining the button component



```
<!DOCTYPE html>
<html>

<body>
  <div id="buttonParent">
  </div>
  <script>
    let state = {
      count: 0
    }

    function onButtonPress() {
      state.count++;
      buttonComponentReRender()
    }

    function buttonComponentReRender() {
      document.getElementById("buttonParent").innerHTML = "";
      const component = buttonComponent(state.count);
      document.getElementById("buttonParent").appendChild(component);
    }

    function buttonComponent(count) {
      const button = document.createElement("button");
      button.innerHTML = `Counter ${count}`;
      button.setAttribute("onclick", `onButtonPress()`);
      return button;
    }

    buttonComponentReRender();

  </script>
</body>

</html>
```

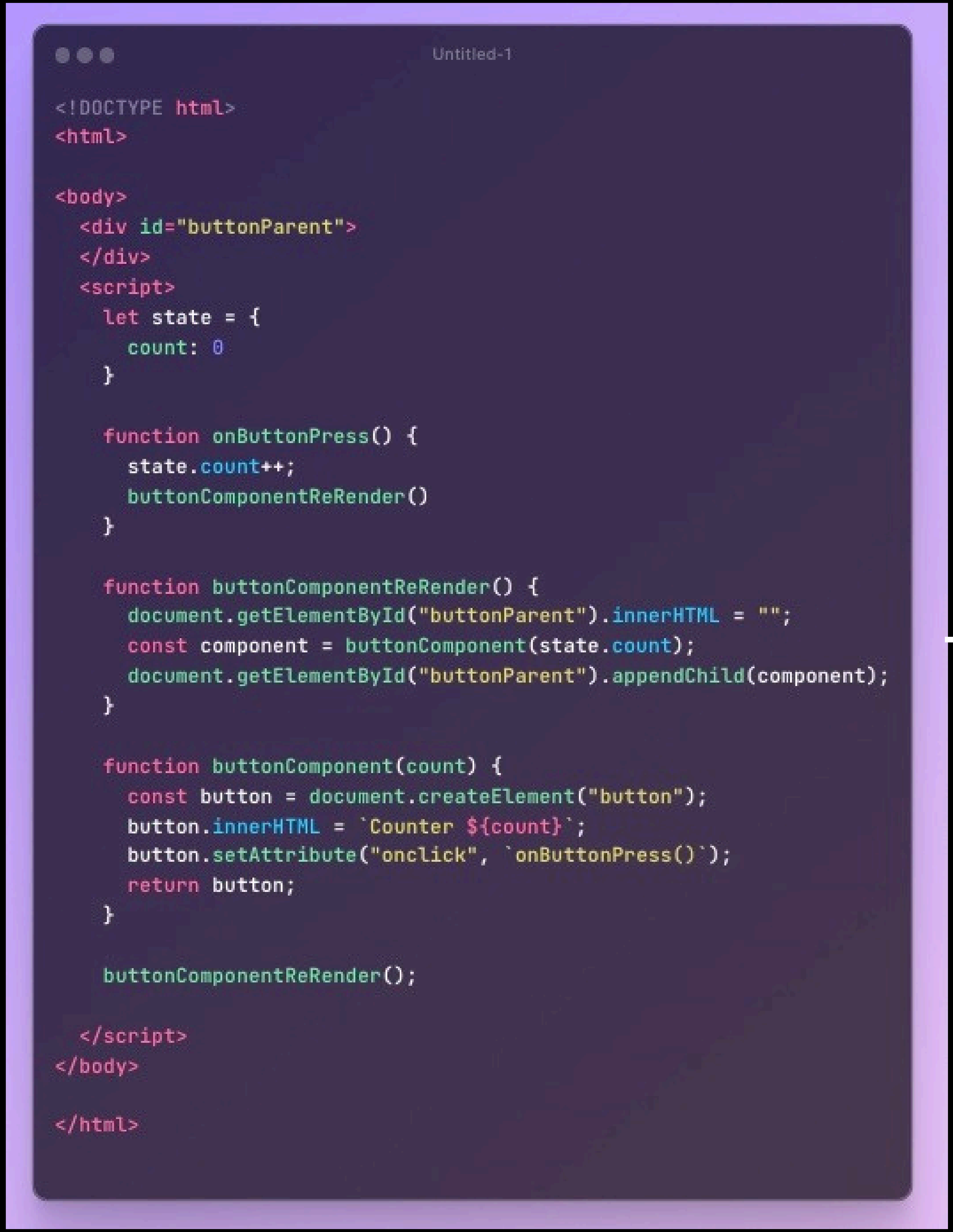
Let's create a counter app using state/components

The react library

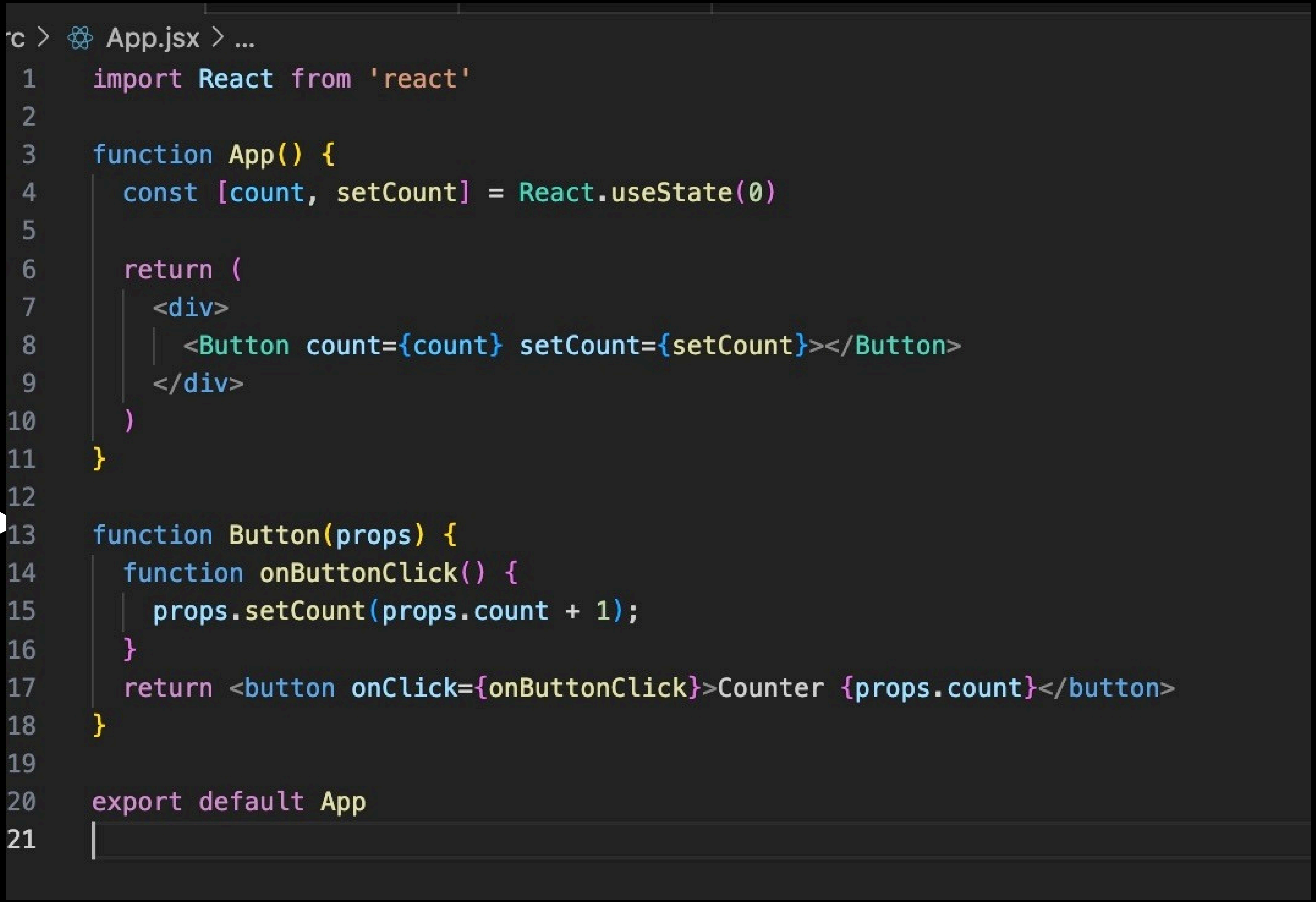


```html
<!DOCTYPE html>
<html>

<body>
  <div id="buttonParent">
  </div>
  <script>
    let state = {
      count: 0
    }

    function onButtonPress() {
      state.count++;
      buttonComponentReRender()
    }

    function buttonComponentReRender() {
      document.getElementById("buttonParent").innerHTML = "";
      const component = buttonComponent(state.count);
      document.getElementById("buttonParent").appendChild(component);
    }

    function buttonComponent(count) {
      const button = document.createElement("button");
      button.innerHTML = `Counter ${count}`;
      button.setAttribute("onclick", `onButtonPress()`);
      return button;
    }

    buttonComponentReRender();

  </script>
</body>

</html>
```

The equivalent code in React looks like this



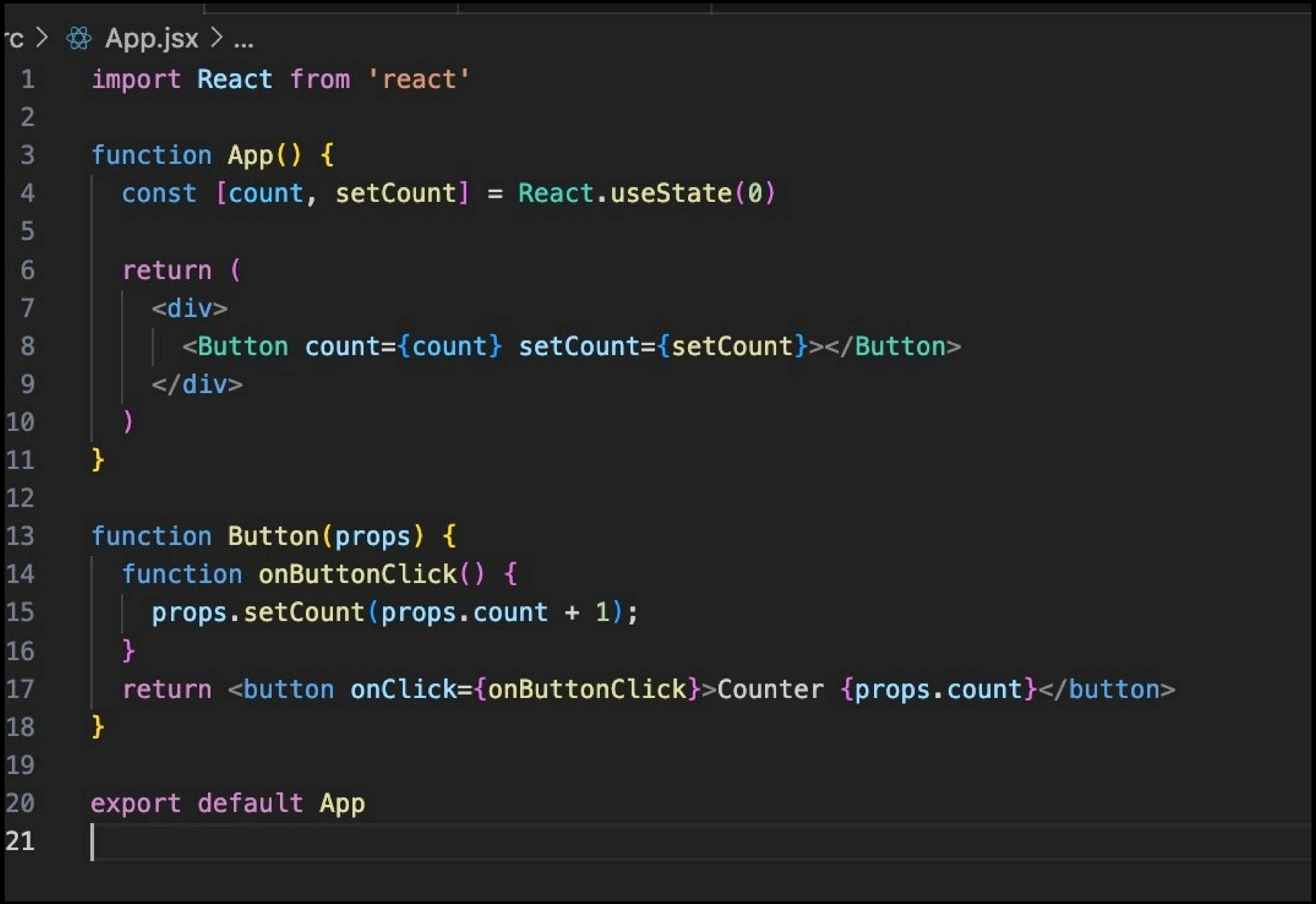
```html
<!DOCTYPE html>
<html>

<body>
  <div id="buttonParent">
  </div>
  <script>
    let state = {
      count: 0
    }

    function onButtonPress() {
      state.count++;
      buttonComponentReRender()
    }

    function buttonComponentReRender() {
      document.getElementById("buttonParent").innerHTML = "";
      const component = buttonComponent(state.count);
      document.getElementById("buttonParent").appendChild(component);
    }

    function buttonComponent(count) {
      const button = document.createElement("button");
      button.innerHTML = `Counter ${count}`;
      button.setAttribute("onclick", `onButtonPress()`);
      return button;
    }

    buttonComponentReRender();

  </script>
</body>

</html>
```



```jsx
import React from 'react'

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

  return (
    <div>
      <Button count={count} setCount={setCount}></Button>
    </div>
  )
}

function Button(props) {
  function onButtonClick() {
    props.setCount(props.count + 1);
  }
  return <button onClick={onButtonClick}>Counter {props.count}</button>
}

export default App
```

The equivalent code in React looks like this

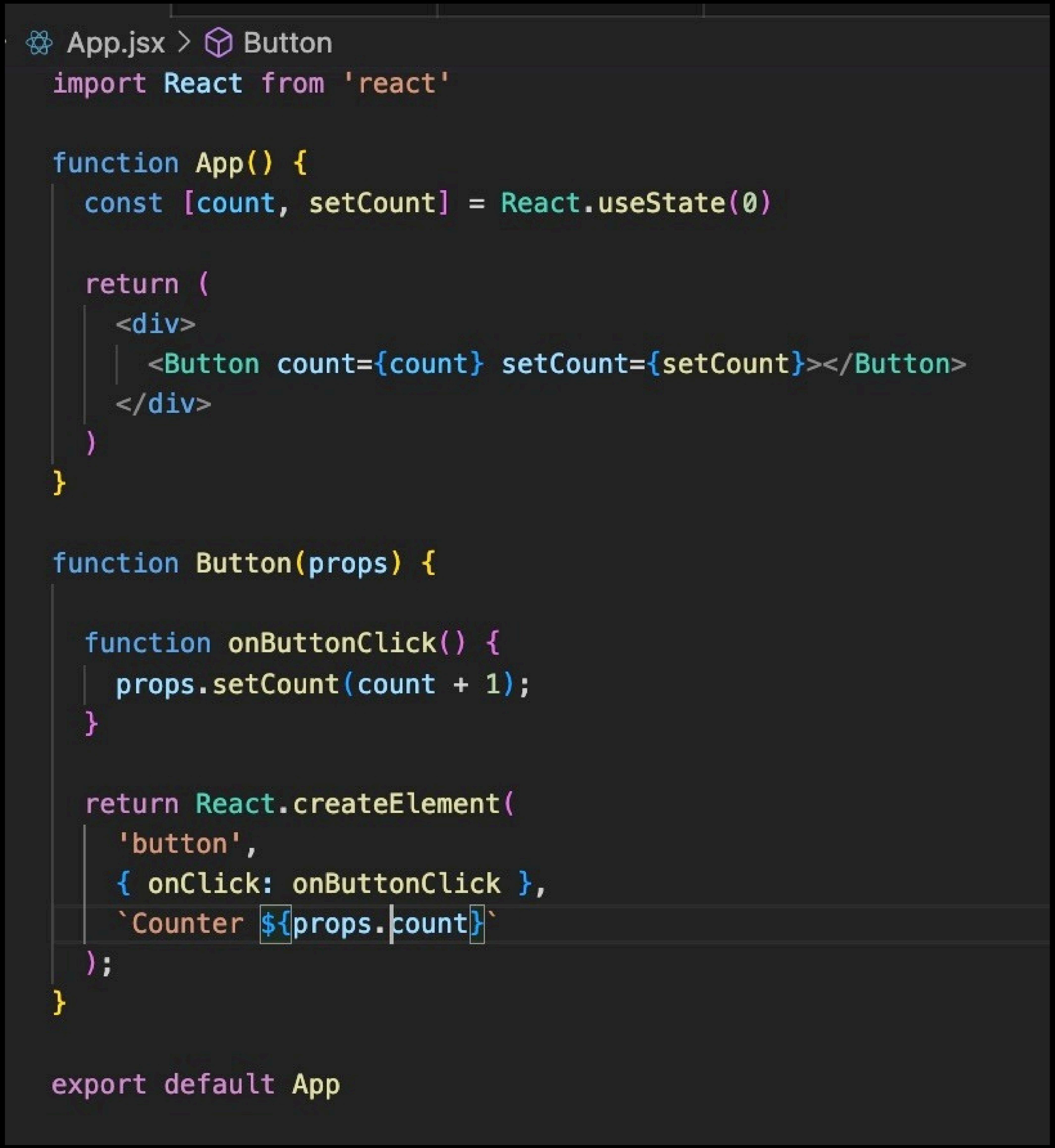
Lets start small, and then build up to this app



```jsx
import React from 'react'

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

  return (
    <div>
      <Button count={count} setCount={setCount}></Button>
    </div>
  )
}

function Button(props) {
  function onButtonClick() {
    props.setCount(props.count + 1);
  }
  return <button onClick={onButtonClick}>Counter {props.count}</button>
}

export default App
```

The equivalent code in React looks like this

Lets start with a simple button component

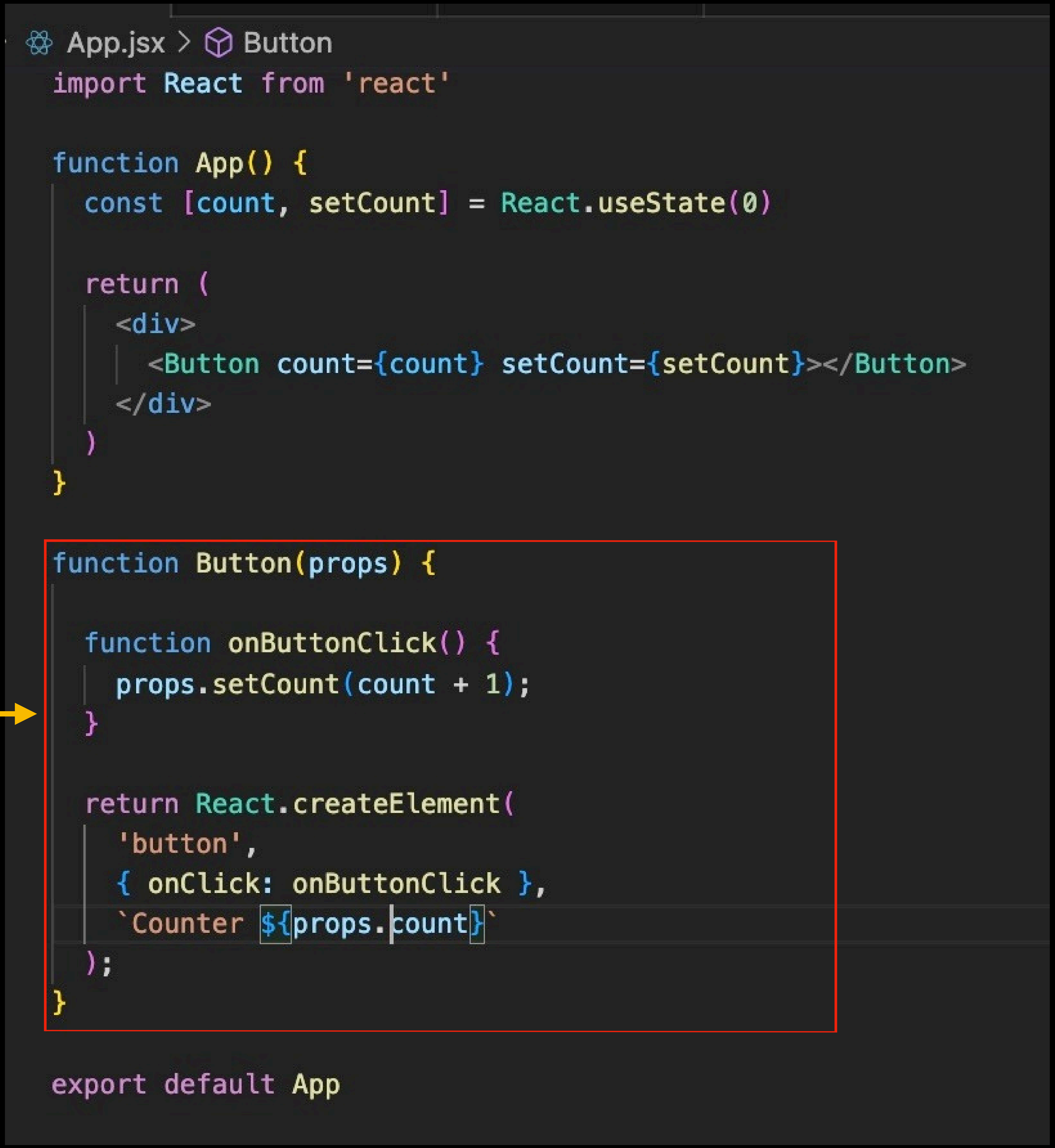


App.jsx > Button

```jsx
import React from 'react'

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

  return (
    <div>
      <Button count={count} setCount={setCount}></Button>
    </div>
  )
}

function Button(props) {

  function onButtonClick() {
    props.setCount(count + 1);
  }

  return React.createElement(
    'button',
    { onClick: onButtonClick },
    `Counter ${props.count}`
  );
}

export default App
```

https://gist.github.com/hkirat/db15d13b42c906269b3114efb96d820f

The equivalent code in React looks like this



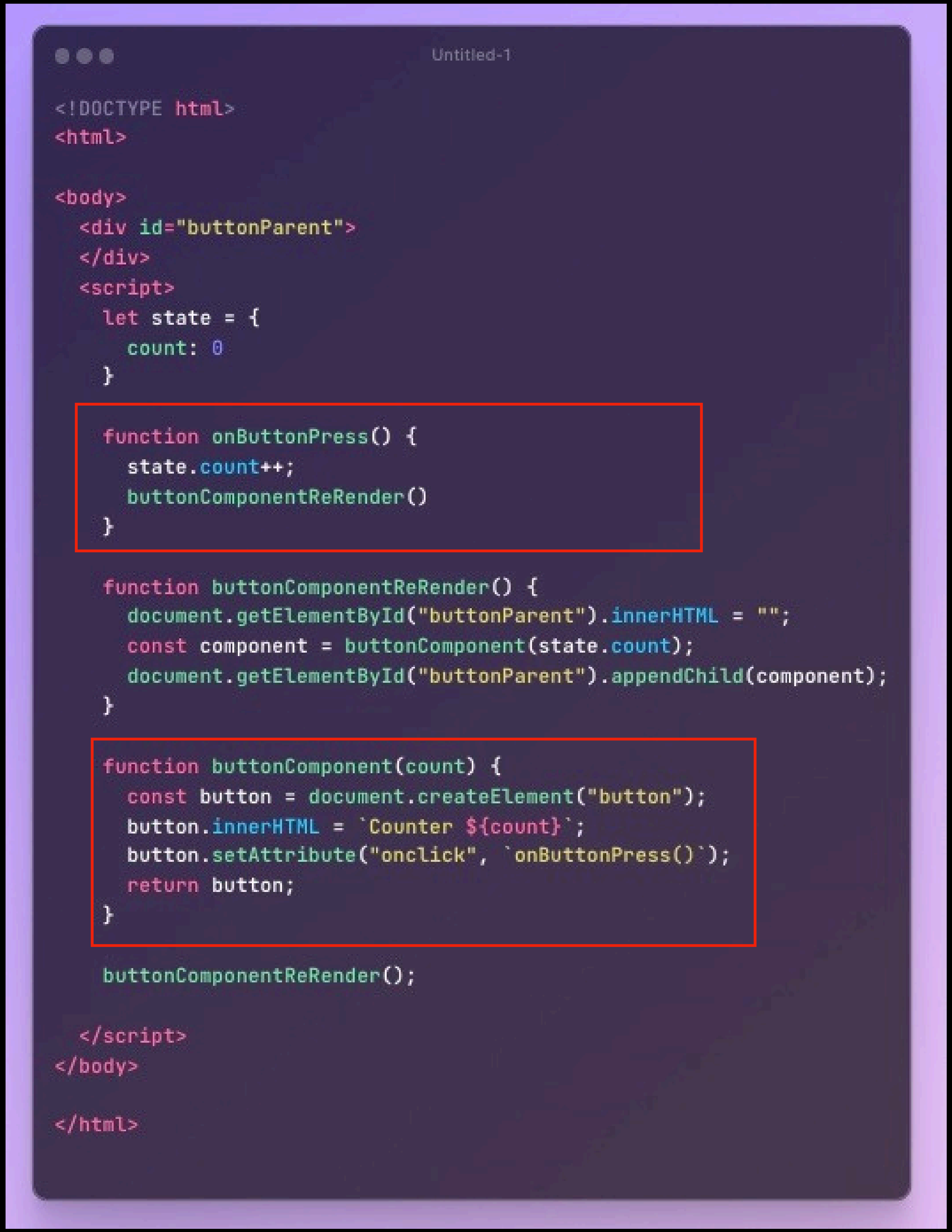
```
import React from 'react'

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

  return (
    <div>
      <Button count={count} setCount={setCount}></Button>
    </div>
  )
}

function Button(props) {

  function onButtonClick() {
    props.setCount(count + 1);
  }

  return React.createElement(
    'button',
    { onClick: onButtonClick },
    `Counter ${props.count}`
  );
}

export default App
```

Defining Button component

https://gist.github.com/hkirat/8801c2cfad70853decd0ad1759d4e63c

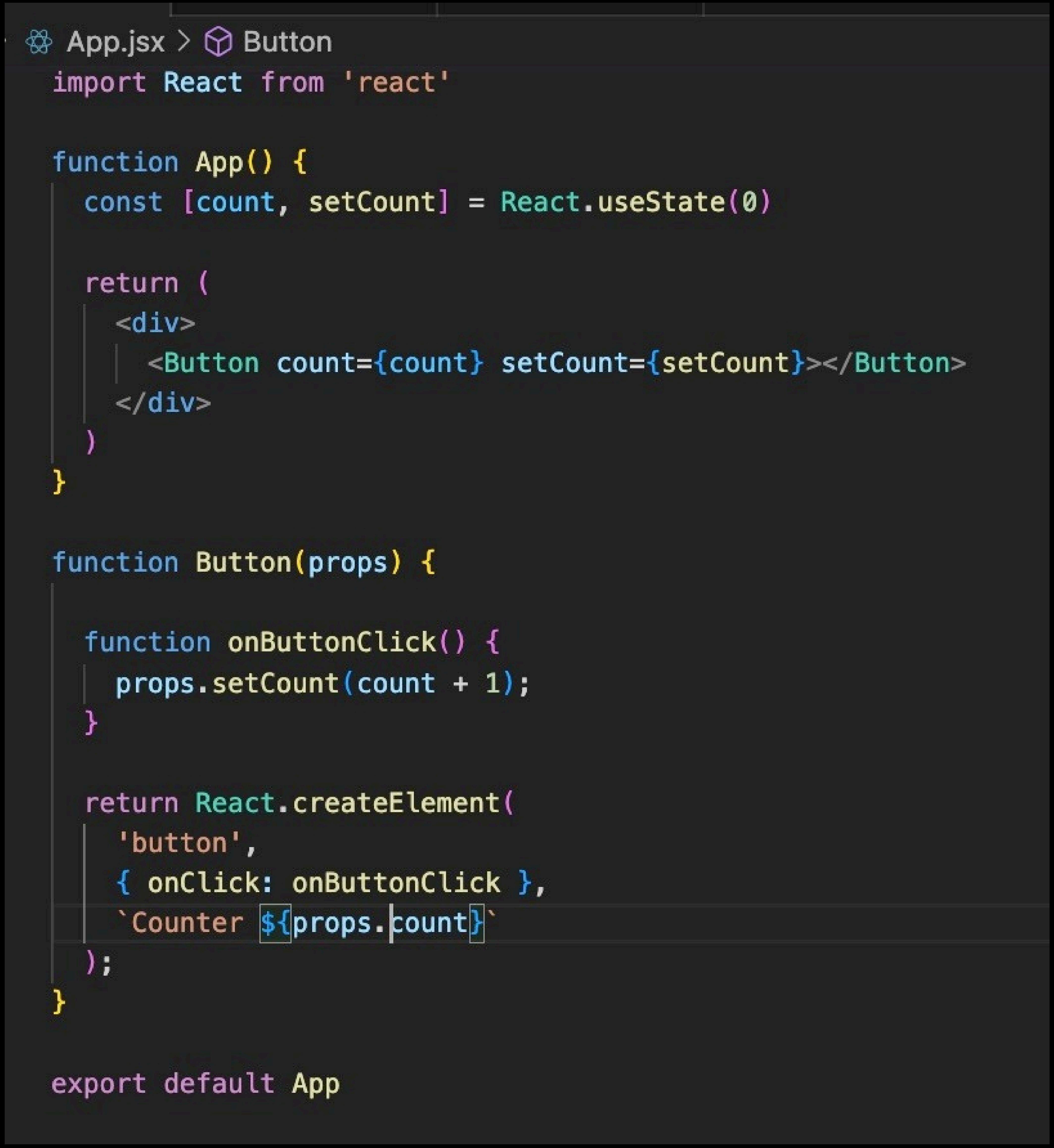
The equivalent code in React looks like this

Defining Button component



Untitled-1

```
<!DOCTYPE html>
<html>

<body>
  <div id="buttonParent">
  </div>
  <script>
    let state = {
      count: 0
    }

    function onButtonPress() {
      state.count++;
      buttonComponentReRender()
    }

    function buttonComponentReRender() {
      document.getElementById("buttonParent").innerHTML = "";
      const component = buttonComponent(state.count);
      document.getElementById("buttonParent").appendChild(component);
    }

    function buttonComponent(count) {
      const button = document.createElement("button");
      button.innerHTML = `Counter ${count}`;
      button.setAttribute("onclick", `onButtonPress()`);
      return button;
    }

    buttonComponentReRender();

  </script>
</body>

</html>
```



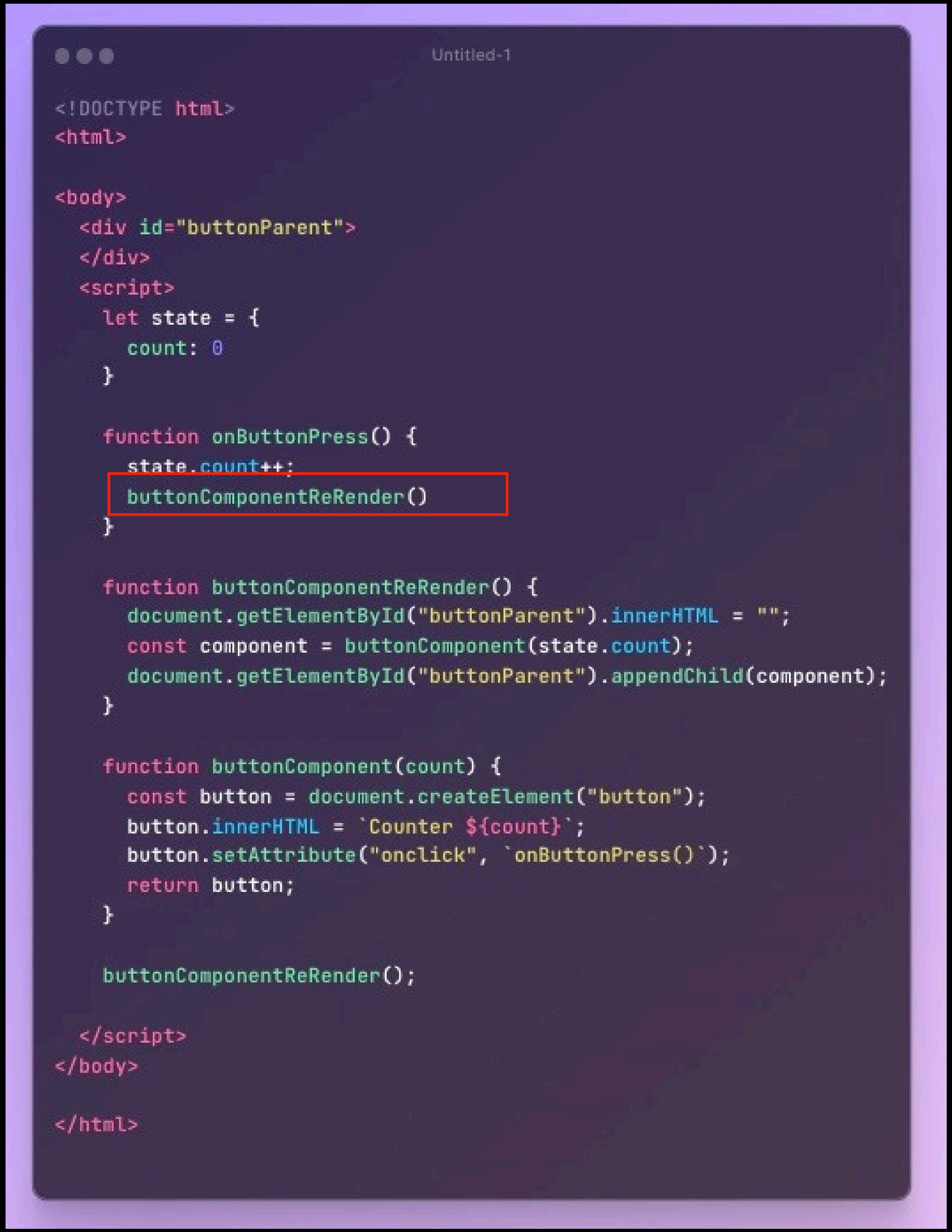
App.jsx > Button

```
import React from 'react'

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

  return (
    <div>
      <Button count={count} setCount={setCount}></Button>
    </div>
  )
}

function Button(props) {

  function onButtonClick() {
    props.setCount(count + 1);
  }

  return React.createElement(
    'button',
    { onClick: onButtonClick },
    `Counter ${props.count}`
  );
}

export default App
```

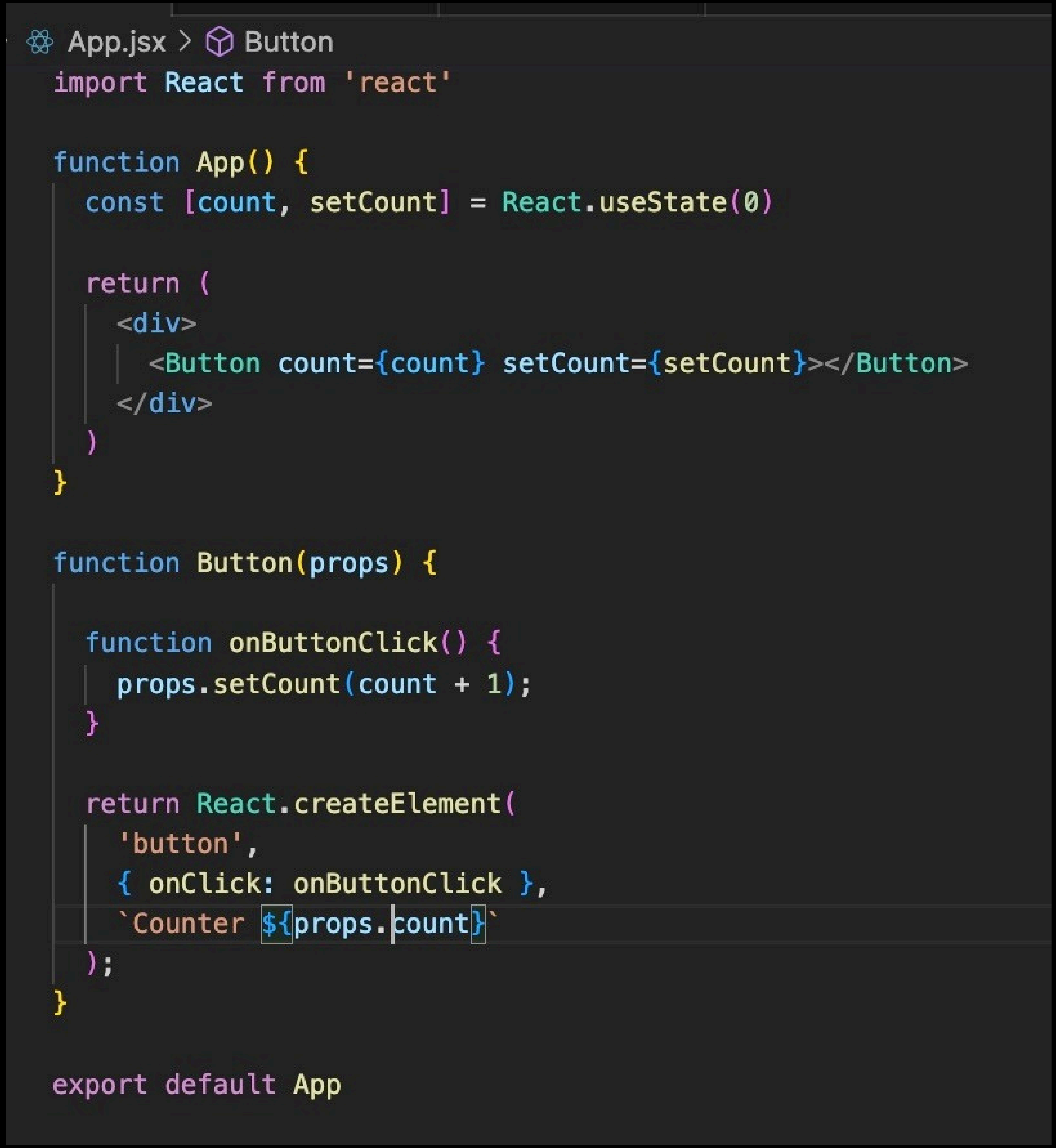
https://gist.github.com/hkirat/8801c2cfad70853decd0ad1759d4e63c

The equivalent code in React looks like this

Triggering re-render



```
<!DOCTYPE html>
<html>

<body>
  <div id="buttonParent">
  </div>
  <script>
    let state = {
      count: 0
    }

    function onButtonPress() {
      state.count++;
      buttonComponentReRender()
    }

    function buttonComponentReRender() {
      document.getElementById("buttonParent").innerHTML = "";
      const component = buttonComponent(state.count);
      document.getElementById("buttonParent").appendChild(component);
    }

    function buttonComponent(count) {
      const button = document.createElement("button");
      button.innerHTML = `Counter ${count}`;
      button.setAttribute("onclick", `onButtonPress()`);
      return button;
    }

    buttonComponentReRender();

  </script>
</body>

</html>
```

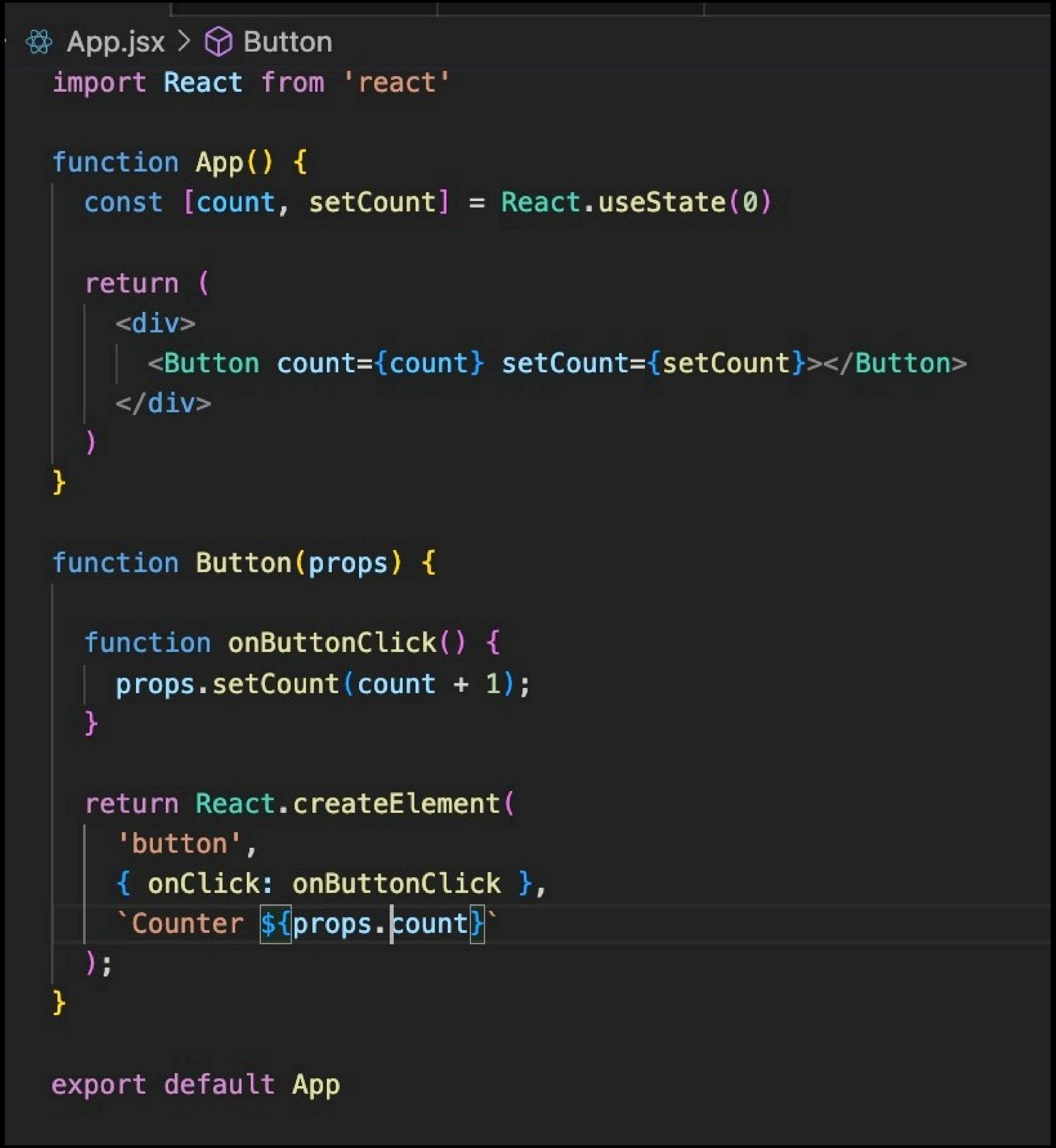


```
import React from 'react'

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

  return (
    <div>
      <Button count={count} setCount={setCount}></Button>
    </div>
  )
}

function Button(props) {

  function onButtonClick() {
    props.setCount(count + 1);
  }

  return React.createElement(
    'button',
    { onClick: onButtonClick },
    `Counter ${props.count}`
  );
}

export default App
```

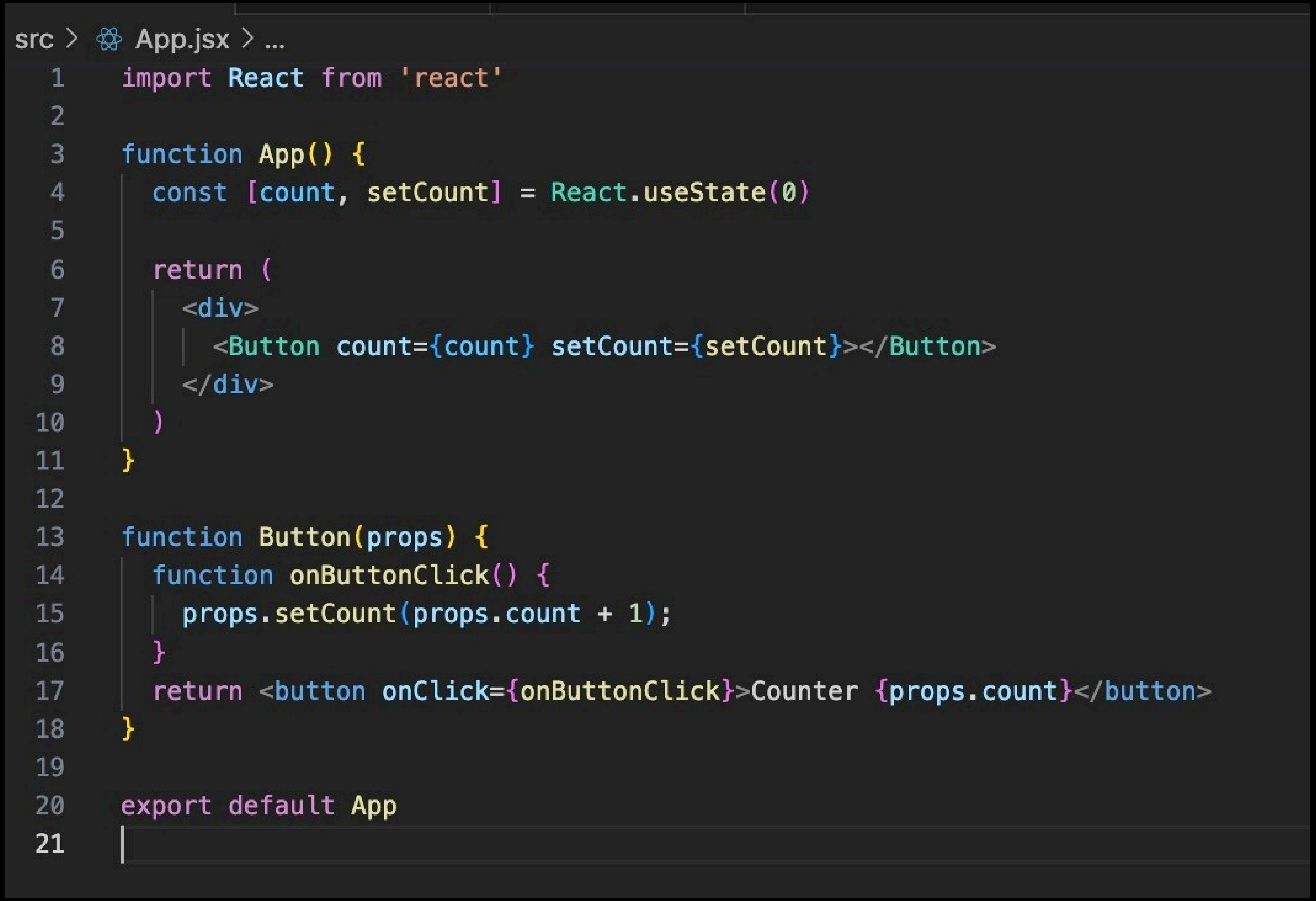
https://gist.github.com/hkirat/8801c2cfad70853decd0ad1759d4e63c

The equivalent code in React looks like this

Jsx syntax is a cleaner way to wrote components



```jsx
import React from 'react'

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

  return (
    <div>
      <Button count={count} setCount={setCount}></Button>
    </div>
  )
}

function Button(props) {

  function onButtonClick() {
    props.setCount(count + 1);
  }

  return React.createElement(
    'button',
    { onClick: onButtonClick },
    `Counter ${props.count}`
  );
}

export default App
```

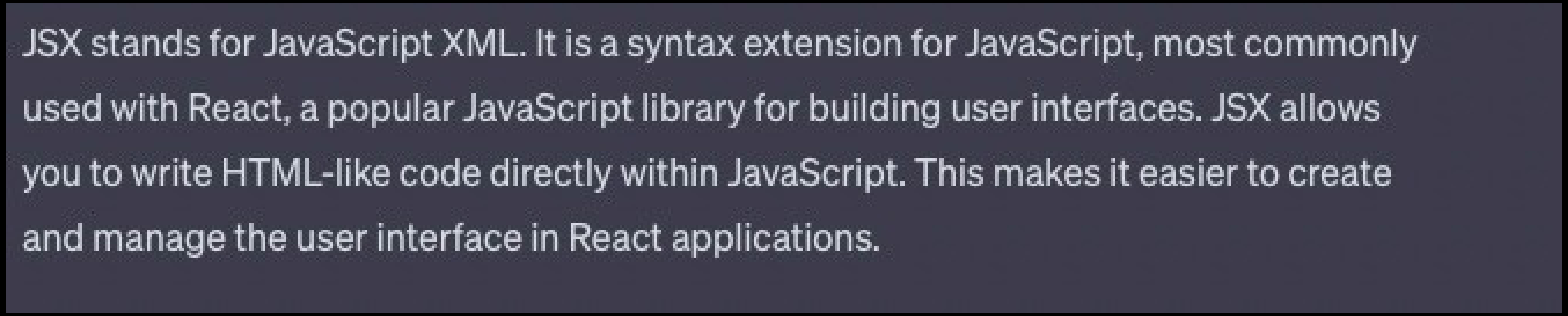


```jsx
import React from 'react'

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

  return (
    <div>
      <Button count={count} setCount={setCount}></Button>
    </div>
  )
}

function Button(props) {
  function onButtonClick() {
    props.setCount(props.count + 1);
  }
  return <button onClick={onButtonClick}>Counter {props.count}</button>
}

export default App

```

https://gist.github.com/hkirat/8801c2cfad70853decd0ad1759d4e63c

The equivalent code in React looks like this

What Is jsx



JSX stands for JavaScript XML. It is a syntax extension for JavaScript, most commonly used with React, a popular JavaScript library for building user interfaces. JSX allows you to write HTML-like code directly within JavaScript. This makes it easier to create and manage the user interface in React applications.

https://gist.github.com/hkirat/dcc85803a20639826bf8f64c6be24a31