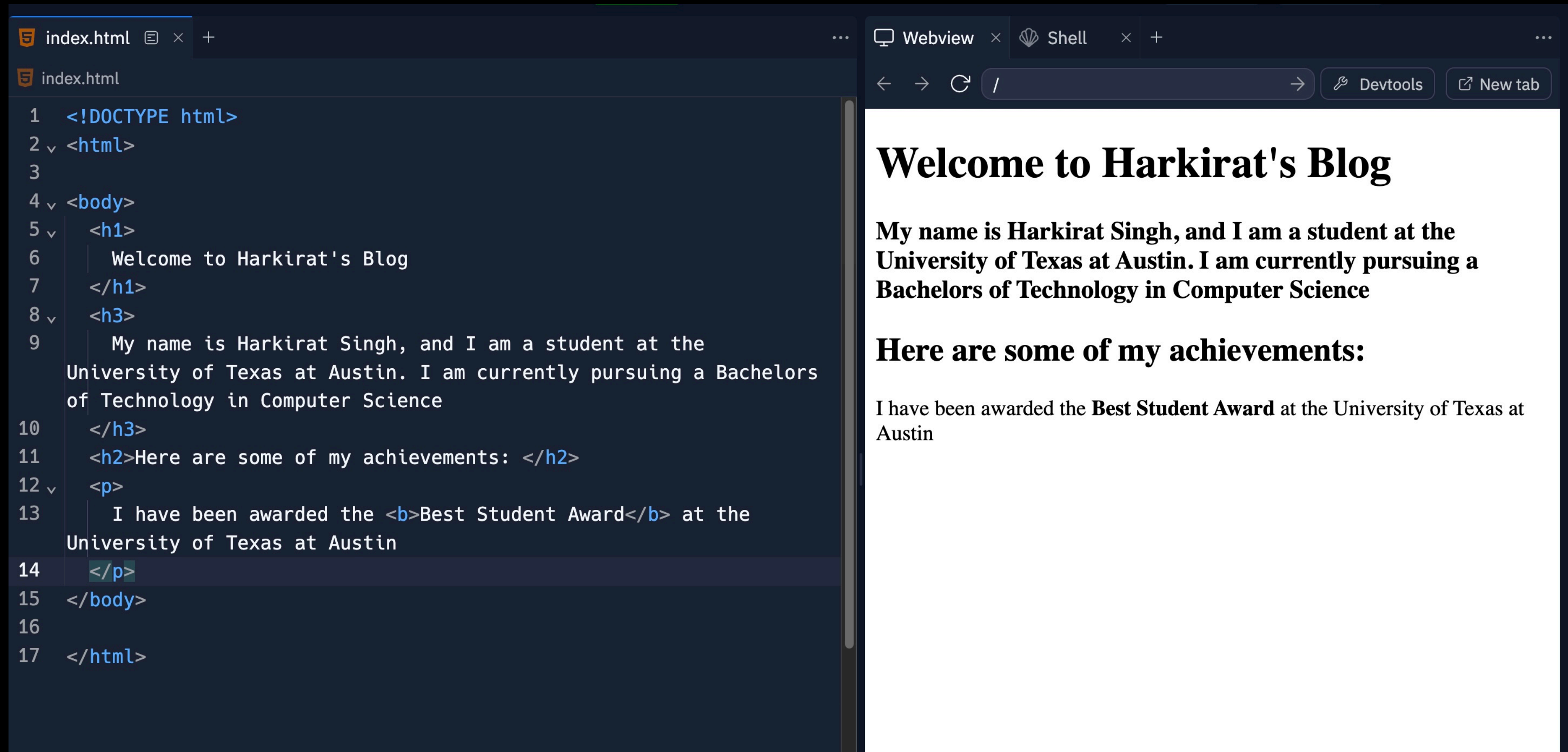# 5.1 | React Deep dive

Understanding React from examples

Jargon we'll learn today

Jsx, class vs className, static vs dynamic websites, State, components, re-rendering
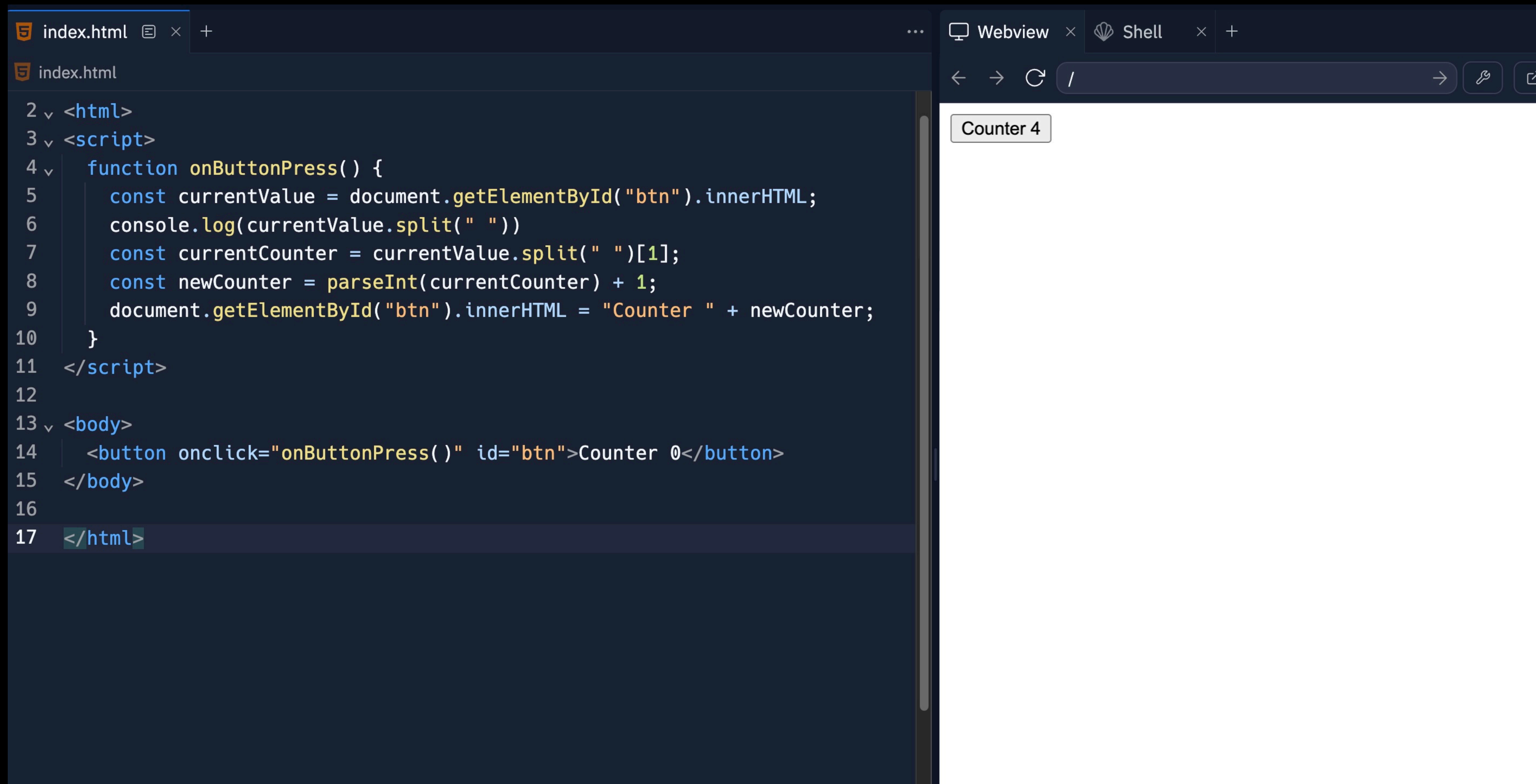
Why do you need React?

For static websites, you don't!

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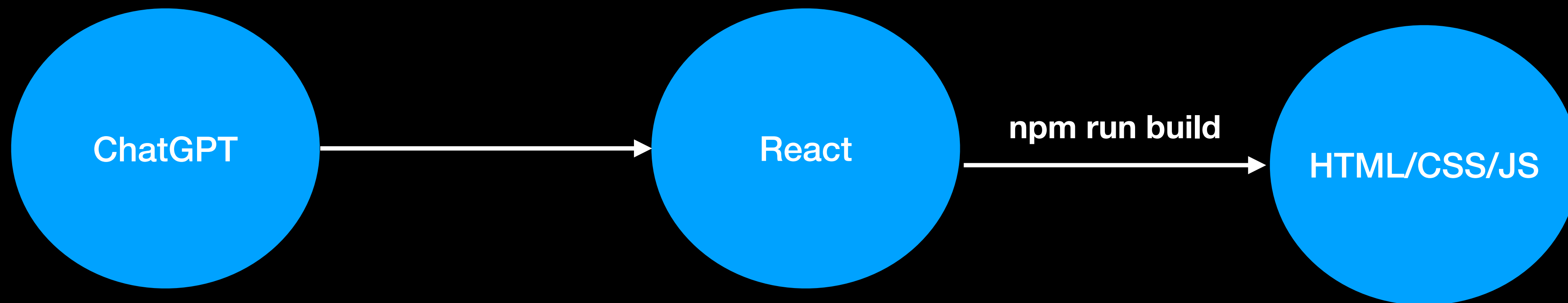
React is just an easier way to write normal HTML/CSS/JS
It's a new syntax, that under the hood gets converted to HTML/CSS/JS

React → **npm run build** → HTML/CSS/JS
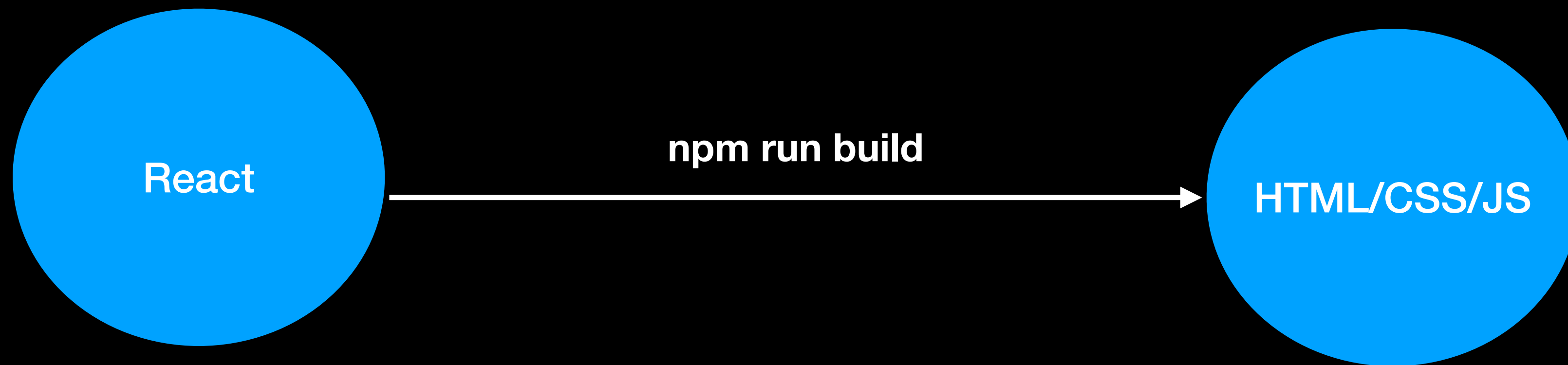
Just how ChatGPT is an easier way to write code, React is an easier way to write HTML/CSS

ChatGPT → React → **npm run build** → HTML/CSS/JS

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)
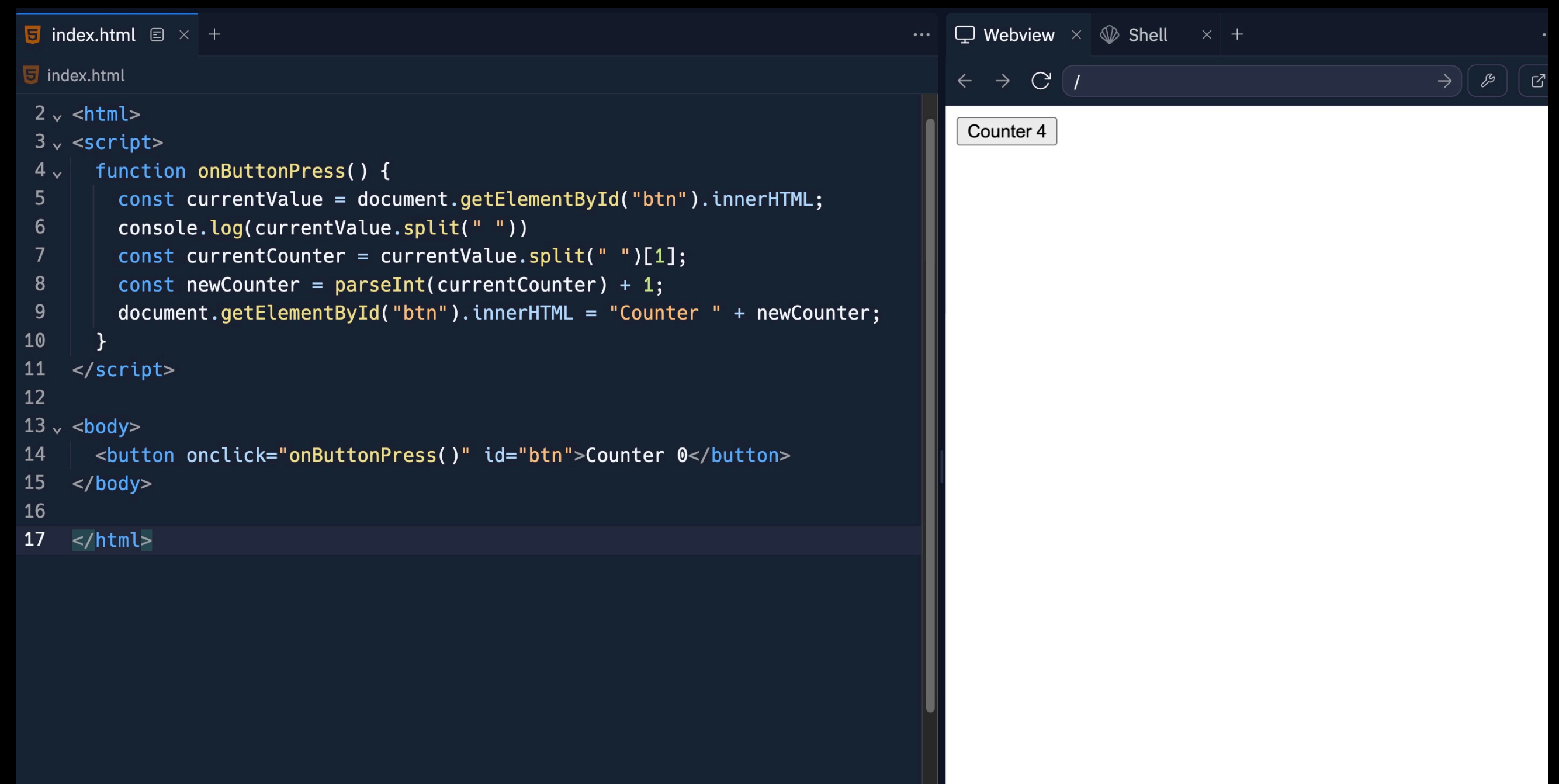Eventually, VueJS/React created a new syntax to do frontends
Under the hood, the react compiler convert your code to HTML/CSS/JS

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.**

```
2   <html>
3   <script>
4     function onButtonPress() {
5       const currentValue = document.getElementById("btn").innerHTML;
6       console.log(currentValue.split(" "))
7       const currentCounter = currentValue.split(" ")[1];
8       const newCounter = parseInt(currentCounter) + 1;
9       document.getElementById("btn").innerHTML = "Counter " + newCounter;
10    }
11  </script>
12
13  <body>
14    <button onclick="onButtonPress()" id="btn">Counter 0</button>
15  </body>
16
17  </html>
```

index.html

Webview   Shell

Counter 4

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

State

Components

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

State/Components/Re-rendering
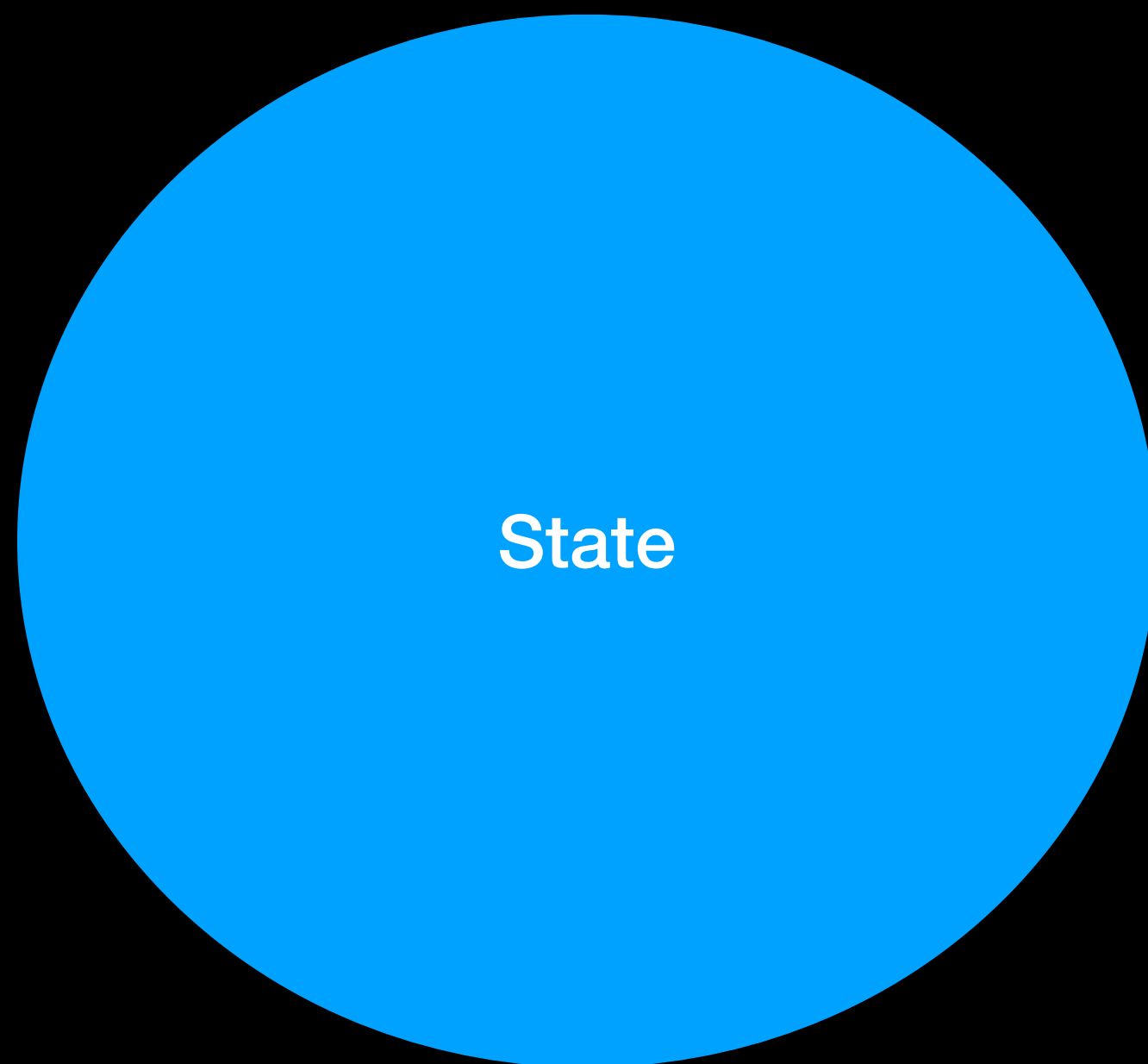
State

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
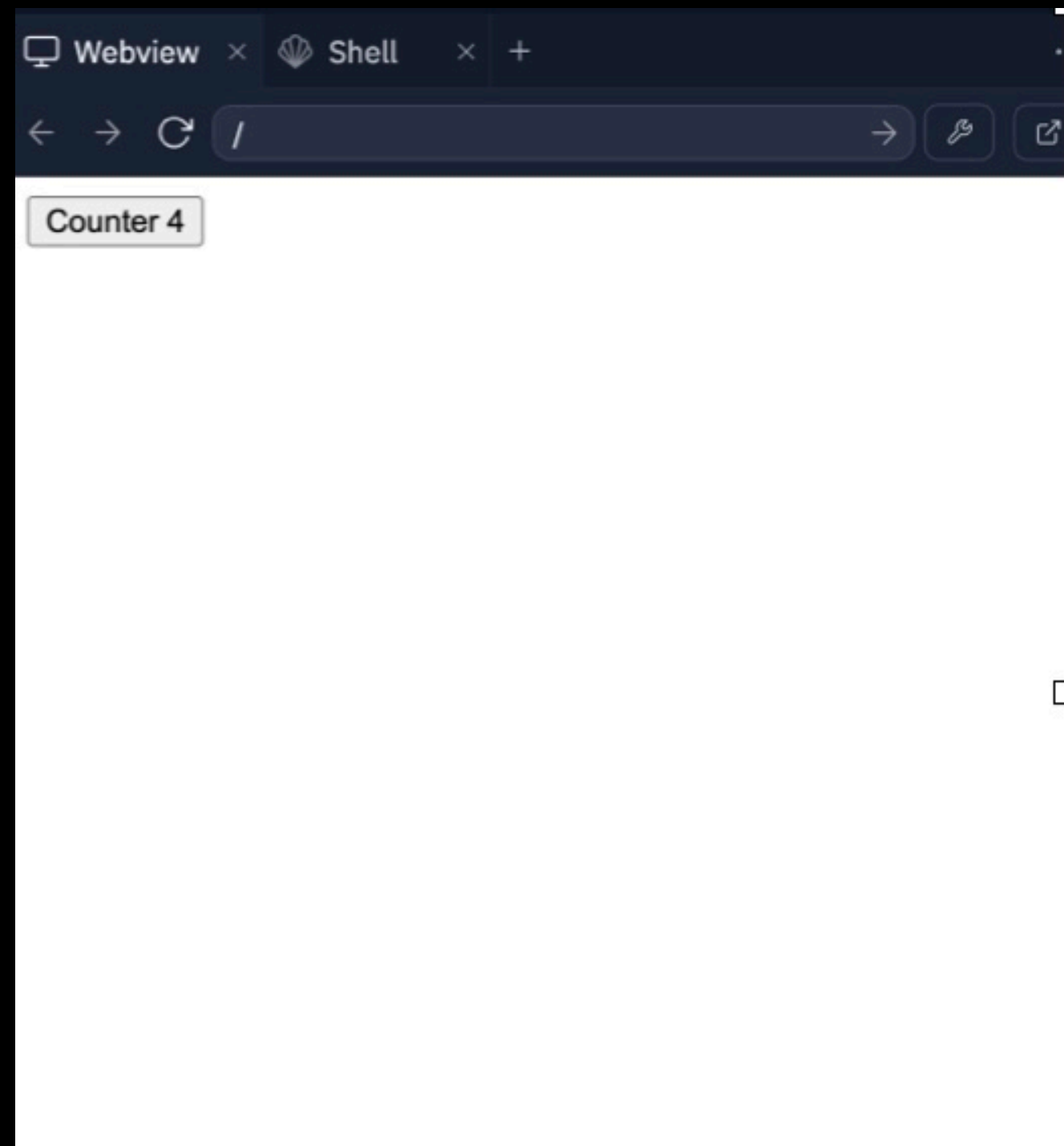
Counter 4

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

```
Untitled-1
{
  count: 1
}
```
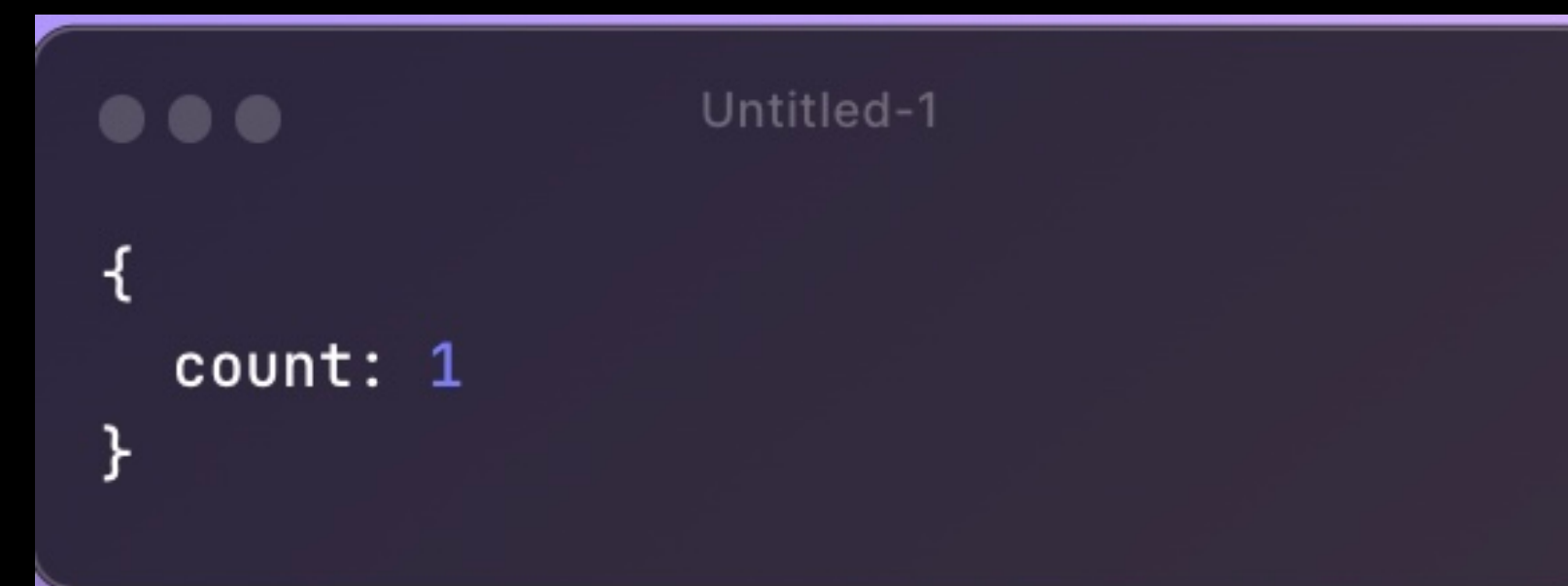
State/Components/Re-rendering

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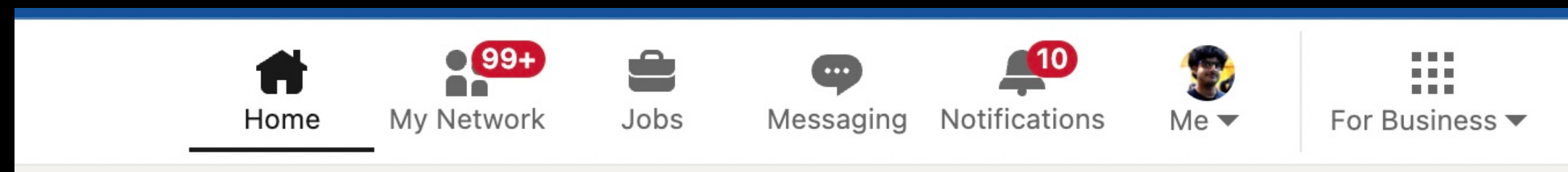

```
{
  topbar: {
    home: 0,
    myNetwork: "99+",
    jobs: 0,
    messaging: 0,
    notificaitons: 10
  }
}
```

State/Components/Re-rendering

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

Components

State/Components/Re-rendering

This button is a component
It takes the state (currentCount) as an input
And is supposed to render accordingly
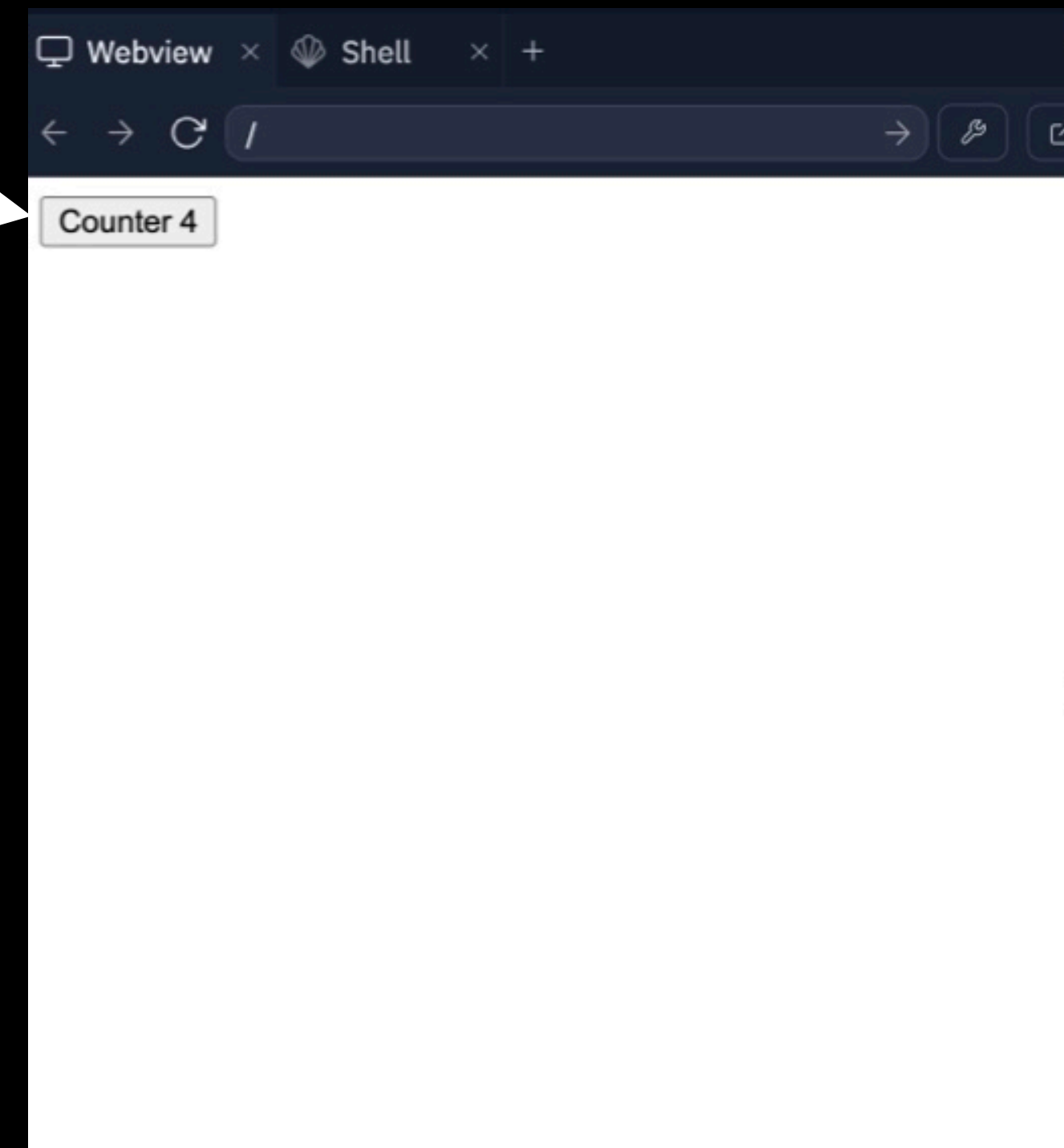
State/Components/Re-rendering


```
Untitled-1

{
  topbar: {
    home: 0,
    myNetwork: "99+",
    jobs: 0,
    messaging: 0,
    notificaitons: 10
  }
}
```
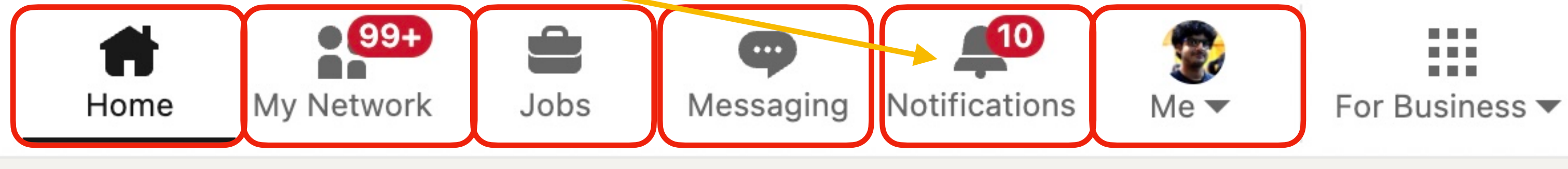
State/Components/Re-rendering

State

Component

```
 ● ● ●              Untitled-1

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

State/Components/Re-rendering

State

Component

```
        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 manipulated
when the state changes

State/Components/Re-rendering

State

Component

```
●  ●  ●              Untitled-1

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

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

11

Notifications
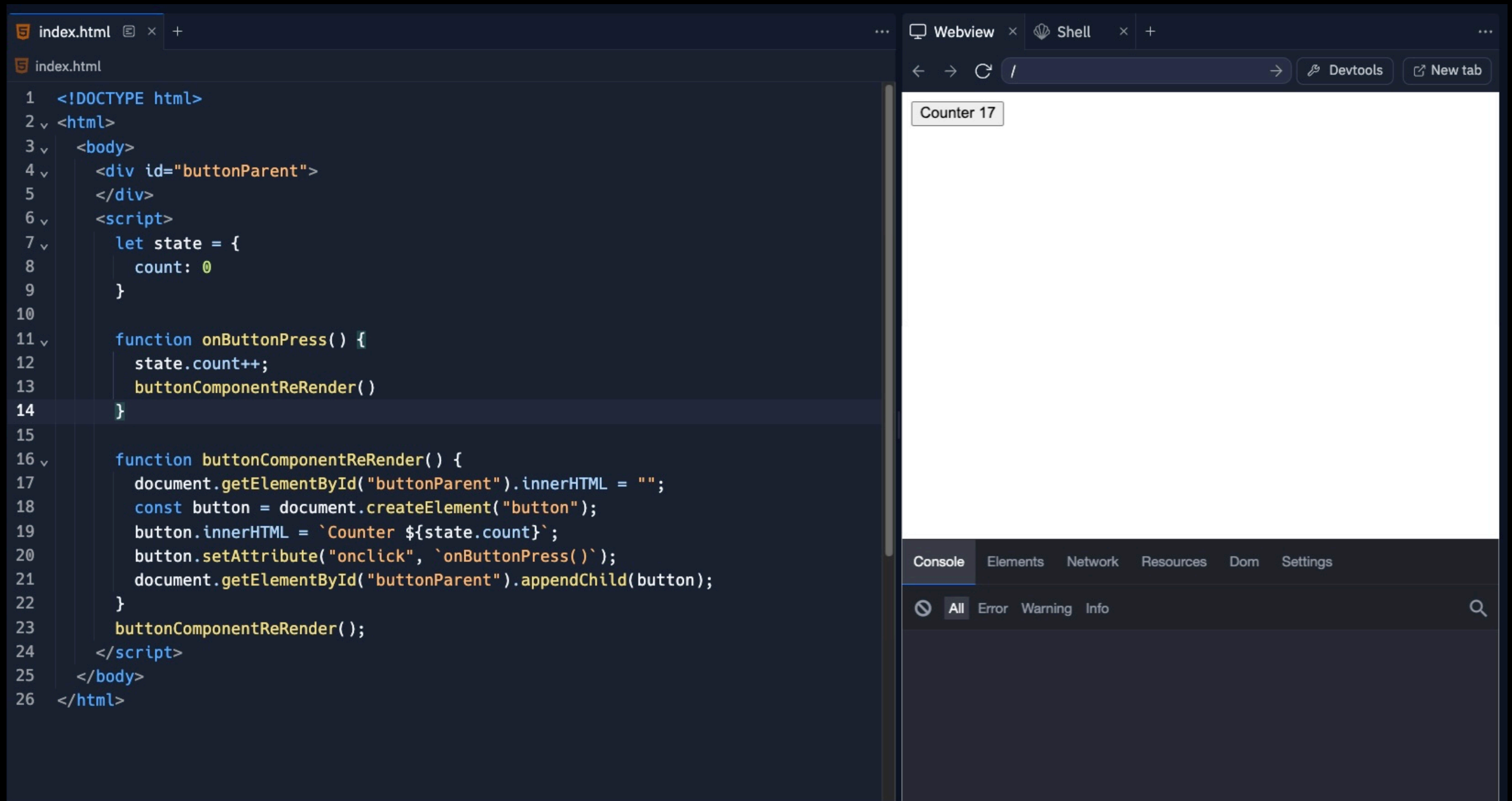
State/Components/Re-rendering

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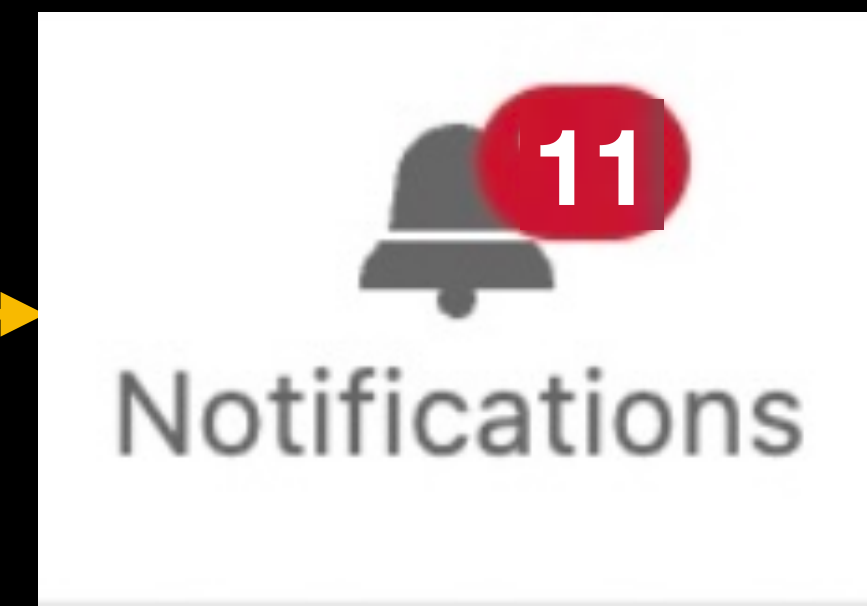And then all you have to do is update the state of your app, React takes care of re-rendering your app

Let's create a counter app using state/components


```html
1  <!DOCTYPE html>
2  <html>
3    <body>
4      <div id="buttonParent">
5      </div>
6      <script>
7        let state = {
8          count: 0
9        }
10
11       function onButtonPress() {
12         state.count++;
13         buttonComponentReRender()
14       }
15
16       function buttonComponentReRender() {
17         document.getElementById("buttonParent").innerHTML = "";
18         const button = document.createElement("button");
19         button.innerHTML = `Counter ${state.count}`;
20         button.setAttribute("onclick", `onButtonPress()`);
21         document.getElementById("buttonParent").appendChild(button);
22       }
23       buttonComponentReRender();
24     </script>
25   </body>
26 </html>
```

Webview | Shell

Devtools   New tab

Counter 17

Console   Elements   Network   Resources   Dom   Settings

All Error Warning Info

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

Let's create a counter app using state/components

1. State initialisation ⟶

```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>
```

Let's create a counter app using state/components

2. Defining the button component

```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>
```

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
rc > ⚛ App.jsx > ...
1   import React from 'react'
2
3   function App() {
4     const [count, setCount] = React.useState(0)
5
6     return (
7       <div>
8         <Button count={count} setCount={setCount}></Button>
9       </div>
10    )
11  }
12
13  function Button(props) {
14    function onButtonClick() {
15      props.setCount(props.count + 1);
16    }
17    return <button onClick={onButtonClick}>Counter {props.count}</button>
18  }
19
20  export default App
21  |
```

The equivalent code in React looks like this

Lets start with a simple button component

```jsx
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/db15d13b42c906269b3114efb96d820f

The equivalent code in React looks like this

```jsx
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
```

Defining Button component →

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

The equivalent code in React looks like this

Defining Button component

```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
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

```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
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

Jsx syntax is a cleaner way to wrote components

```jsx
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
```

```jsx
src > App.jsx > ...
1  import React from 'react'
2
3  function App() {
4    const [count, setCount] = React.useState(0)
5
6    return (
7      <div>
8        <Button count={count} setCount={setCount}></Button>
9      </div>
10   )
11 }
12
13 function Button(props) {
14   function onButtonClick() {
15     props.setCount(props.count + 1);
16   }
17   return <button onClick={onButtonClick}>Counter {props.count}</button>
18 }
19
20 export default App
21
```

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.