

# 5.1 | React Deep dive

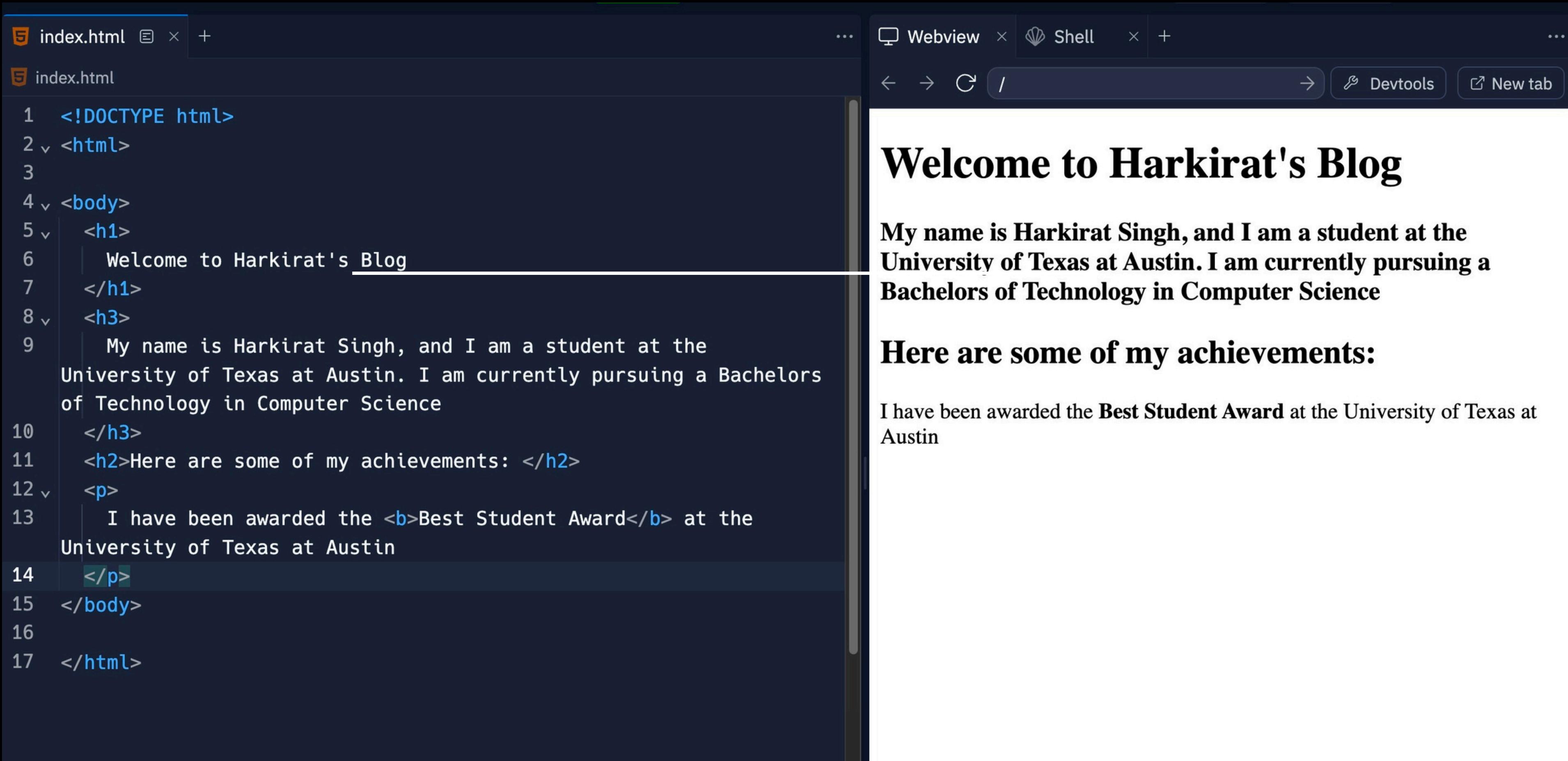
Understanding React from examples

# 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!



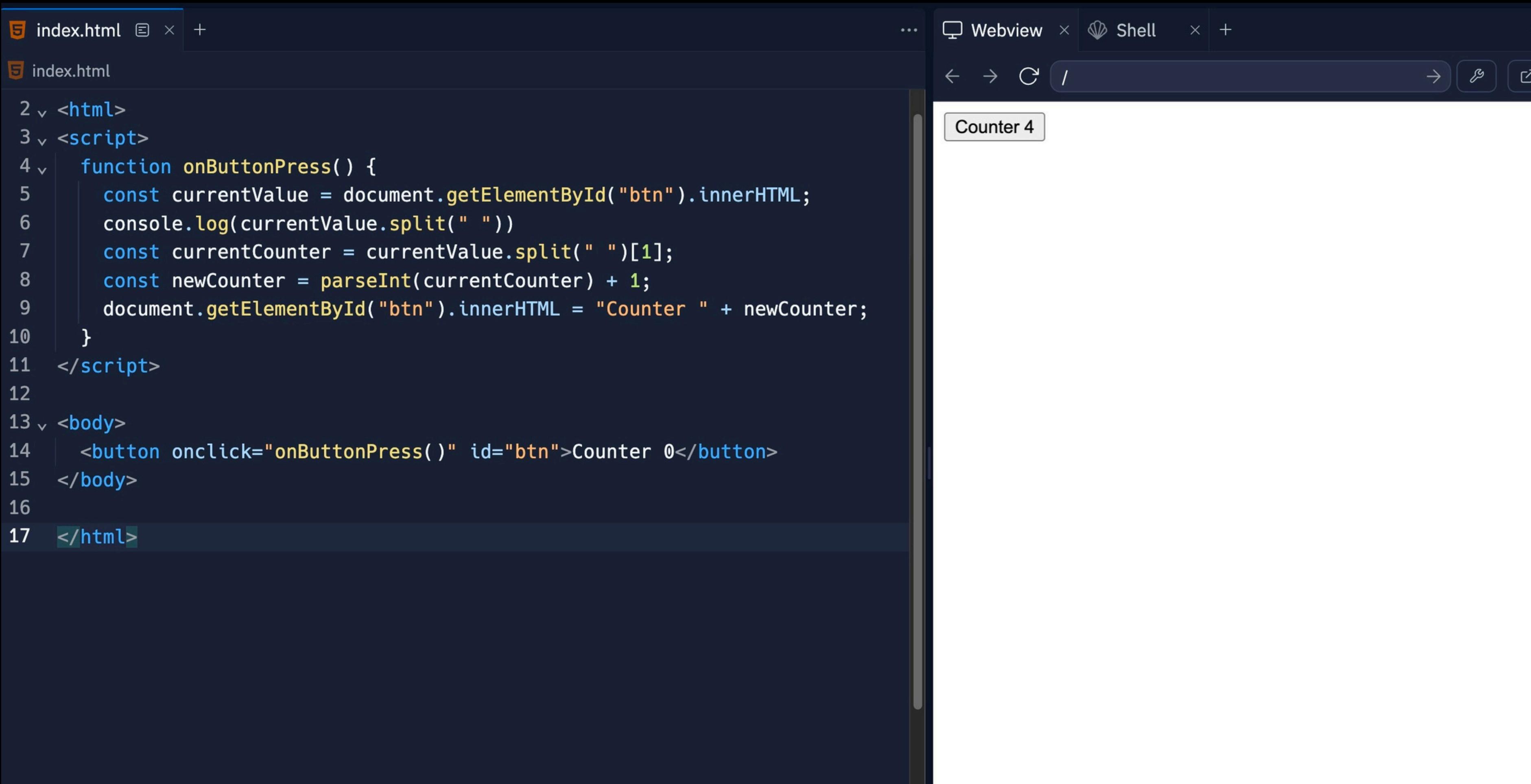
The image shows a split-screen view. On the left is a code editor window titled "index.html" containing the following HTML code:

```
1  <!DOCTYPE html>
2  <html>
3
4  <body>
5    <h1>
6      Welcome to Harkirat's Blog
7    </h1>
8    <h3>
9      My name is Harkirat Singh, and I am a student at the
10     University of Texas at Austin. I am currently pursuing a Bachelors
11     of Technology in Computer Science
12   </h3>
13   <h2>Here are some of my achievements: </h2>
14   <p>
15     I have been awarded the <b>Best Student Award</b> at the
16     University of Texas at Austin
17   </p>
18 </body>
19 </html>
```

On the right is a browser window titled "Webview" showing the rendered HTML. The page title is "Welcome to Harkirat's Blog". The content includes a heading "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" and a section "Here are some of my achievements:" followed by a paragraph about the Best Student Award.

# Why do you need React?

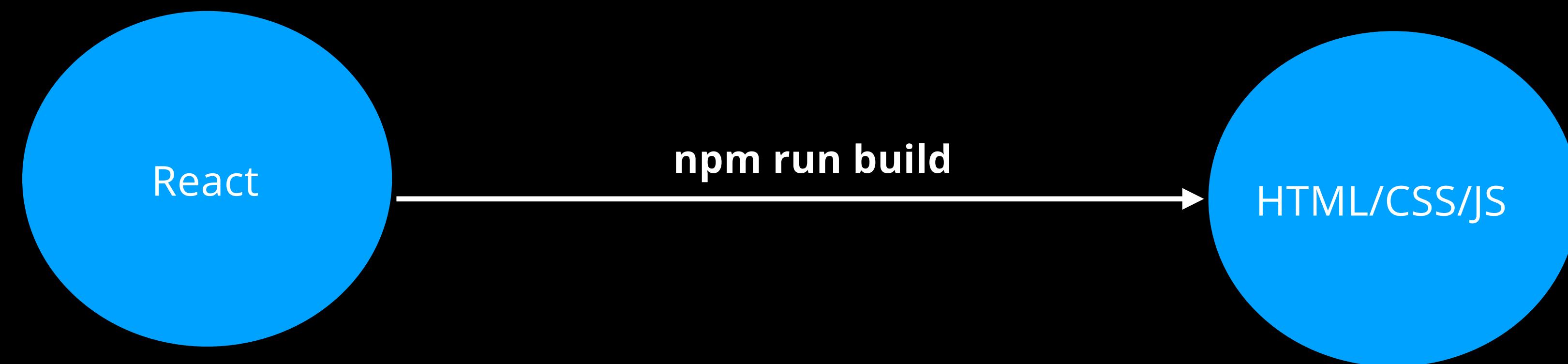
For dynamic websites, these libraries make it easier to do DOM manipulation



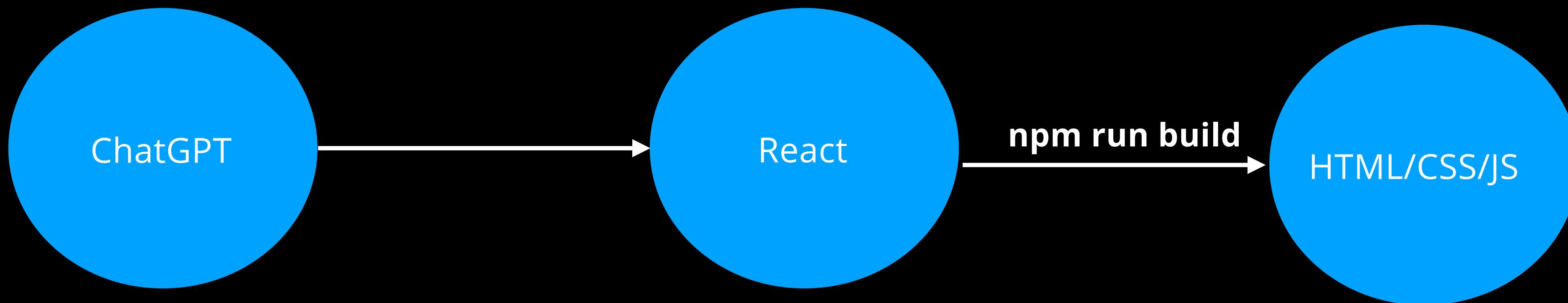
The image shows a split-screen interface. On the left, a code editor displays the file `index.html`. The code contains a script block that defines a function `onButtonPress` which reads the current value from a button's innerHTML, increments it by 1, and then updates the button's innerHTML to reflect the new value. The browser window on the right shows the result of running this code, displaying a button labeled "Counter 4".

```
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>
```

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)**

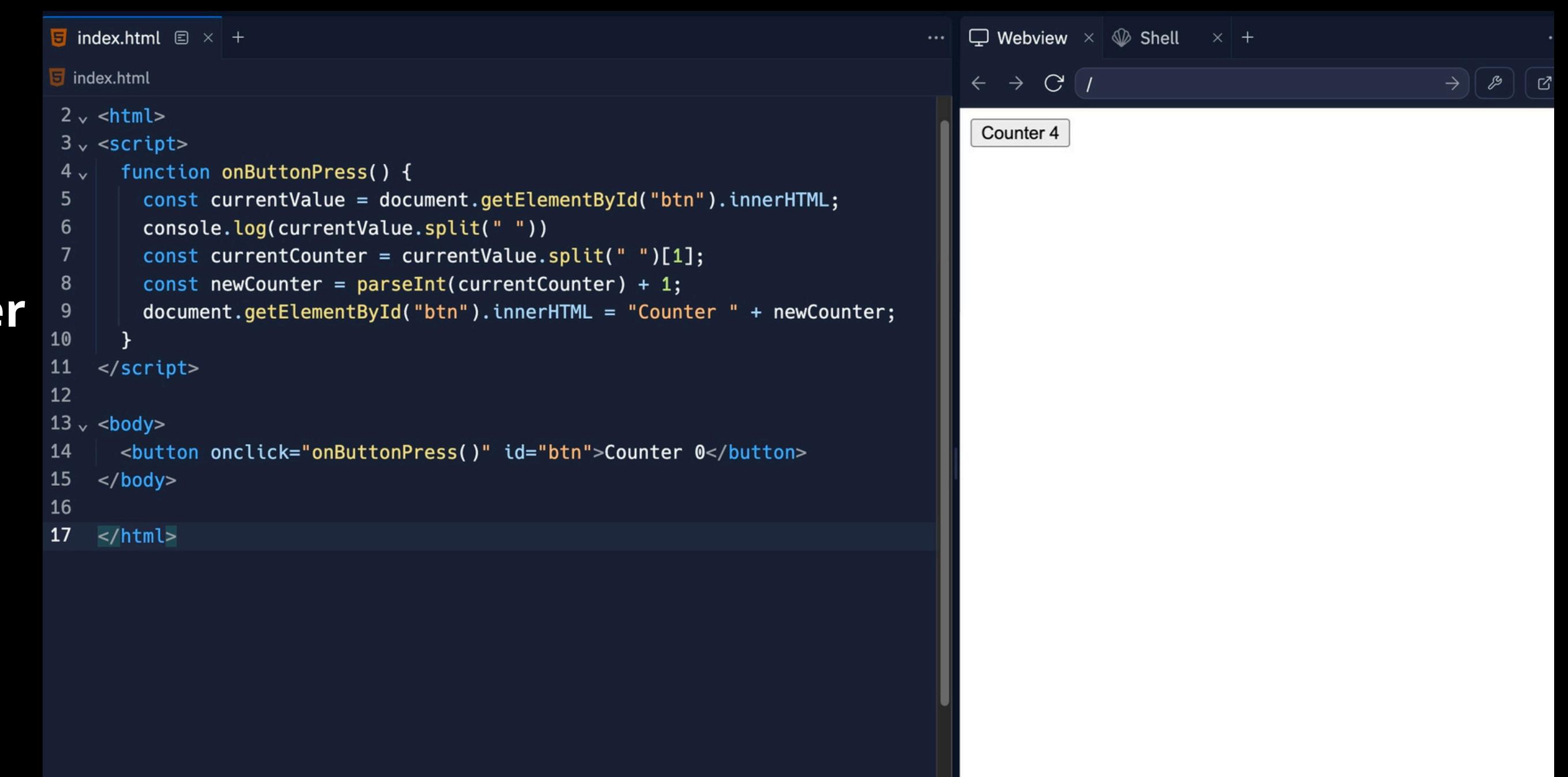
**Eventually, VueJS/React created a new syntax to do frontends**

**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.



The image shows a code editor and a browser window side-by-side. The code editor on the left has a dark theme and displays the following HTML and JavaScript code:

```
index.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>
```

The browser window on the right shows a simple web page with a single button. The button's text is "Counter 0". Above the button, there is some developer tool output: "Counter 4". This demonstrates that each time the button is clicked, the counter value is logged to the console and then updated on the page.

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

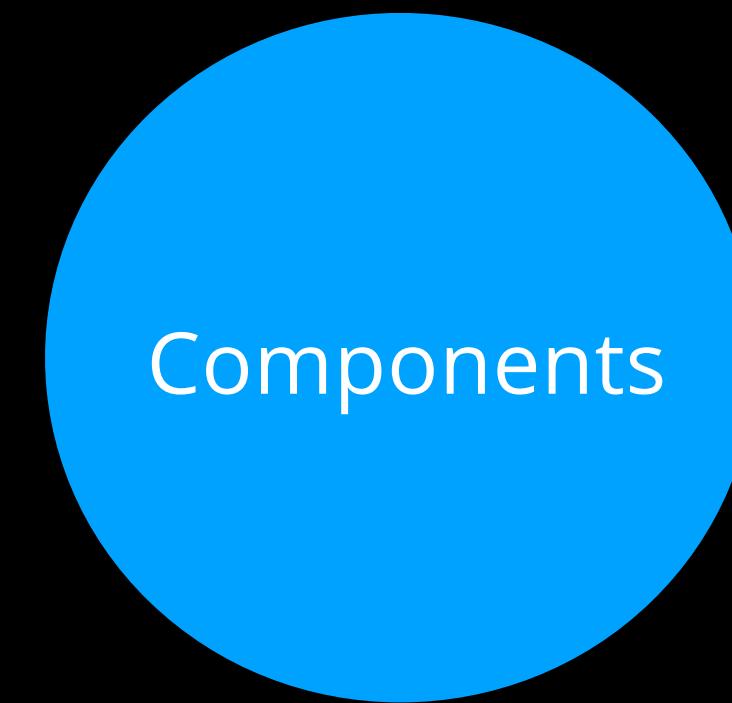
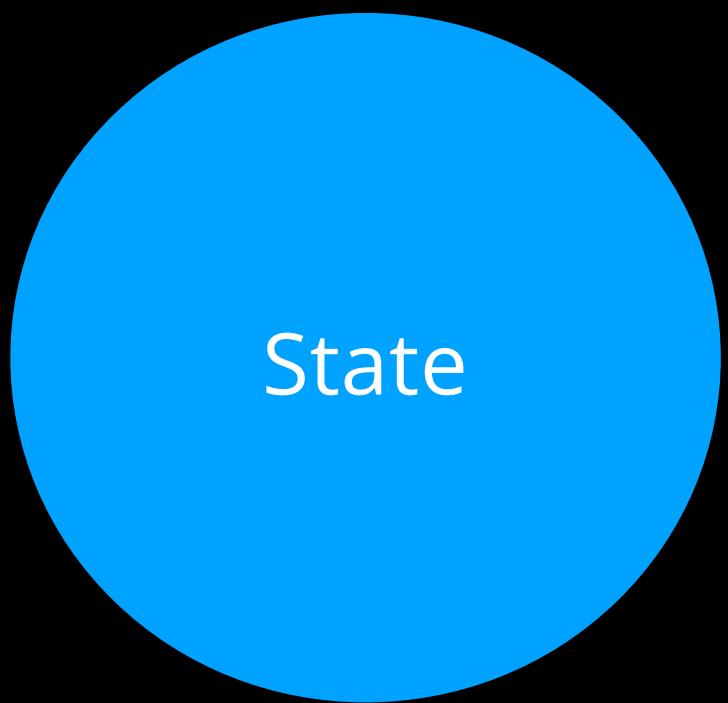
# Some react jargon

# Some react jargon

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

# Some react jargon

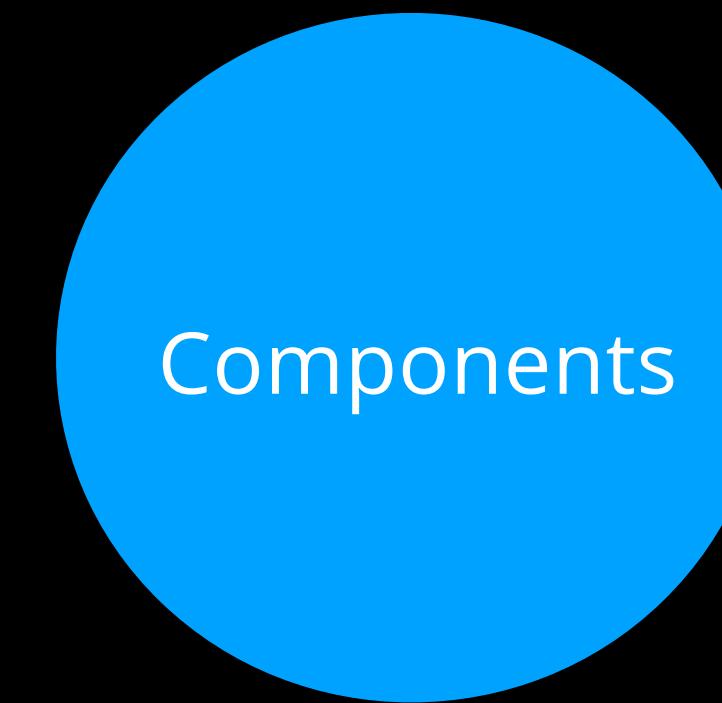
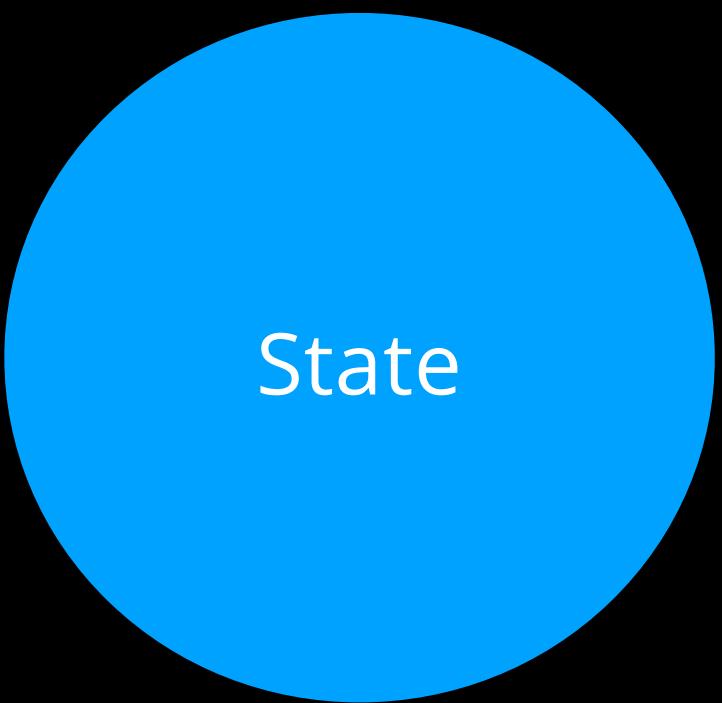
To create a react app, you usually need to worry about two things



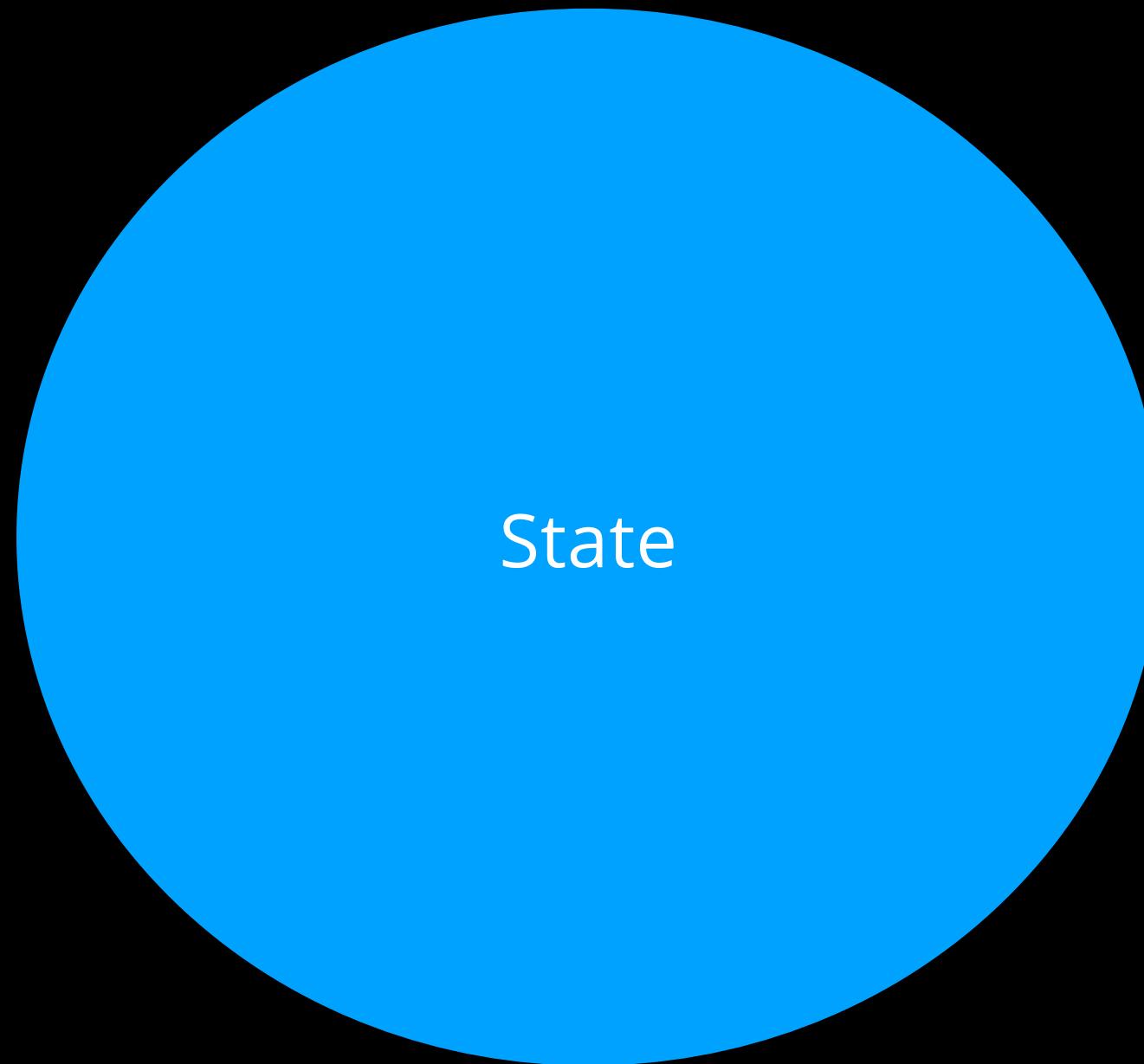
# Some react jargon

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

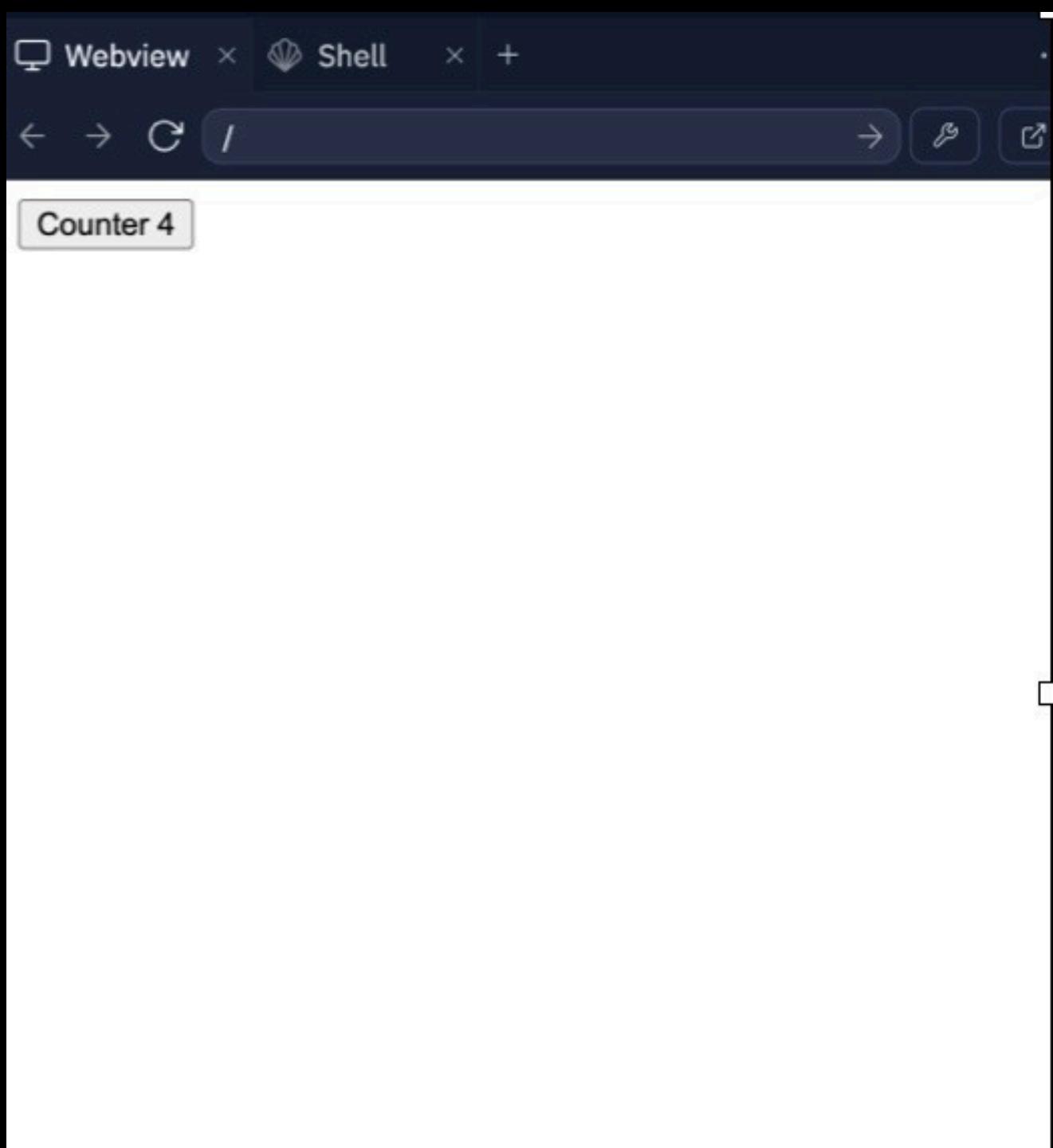


An object that represents the current **state** of the app

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

For example, the value of the counter

# State/Components/Re-rendering



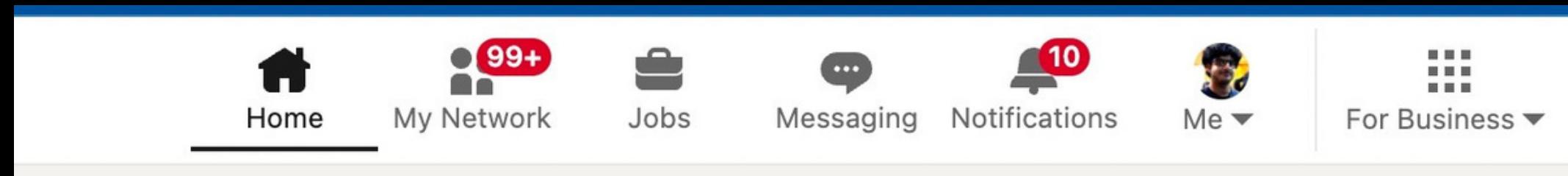
For the counter app, it could look something like this -

A screenshot of a code editor window titled "Untitled-1". The code is as follows:

```
{\n  count: 1\n}
```

# State/Components/Re-rendering

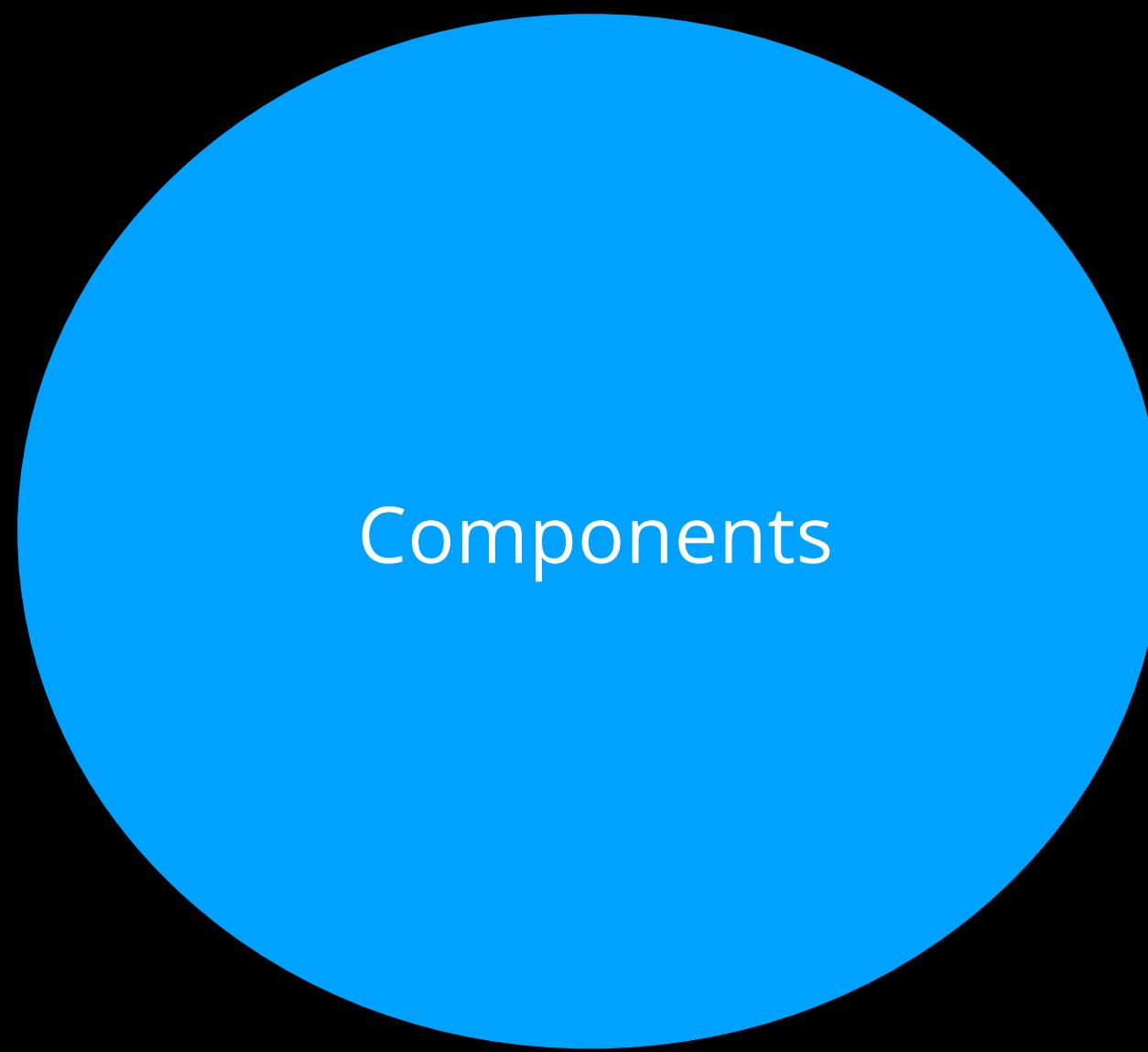
For the LinkedIn Topbar, it could be something like this -



```
Untitled-1

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

# State/Components/Re-rendering

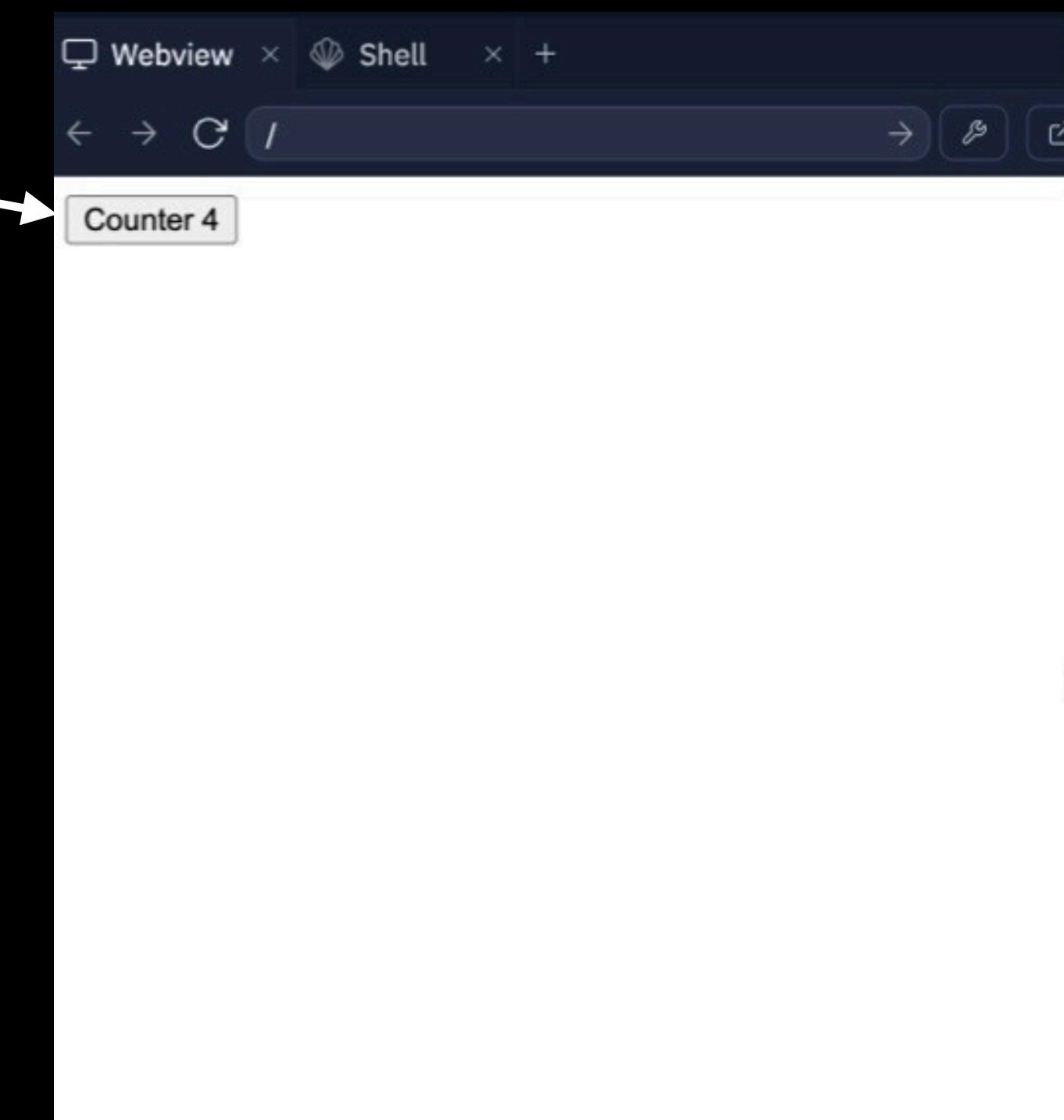


**How a DOM element should render, given a state  
It is a re-usable, dynamic, HTML snippet that changes given the state**

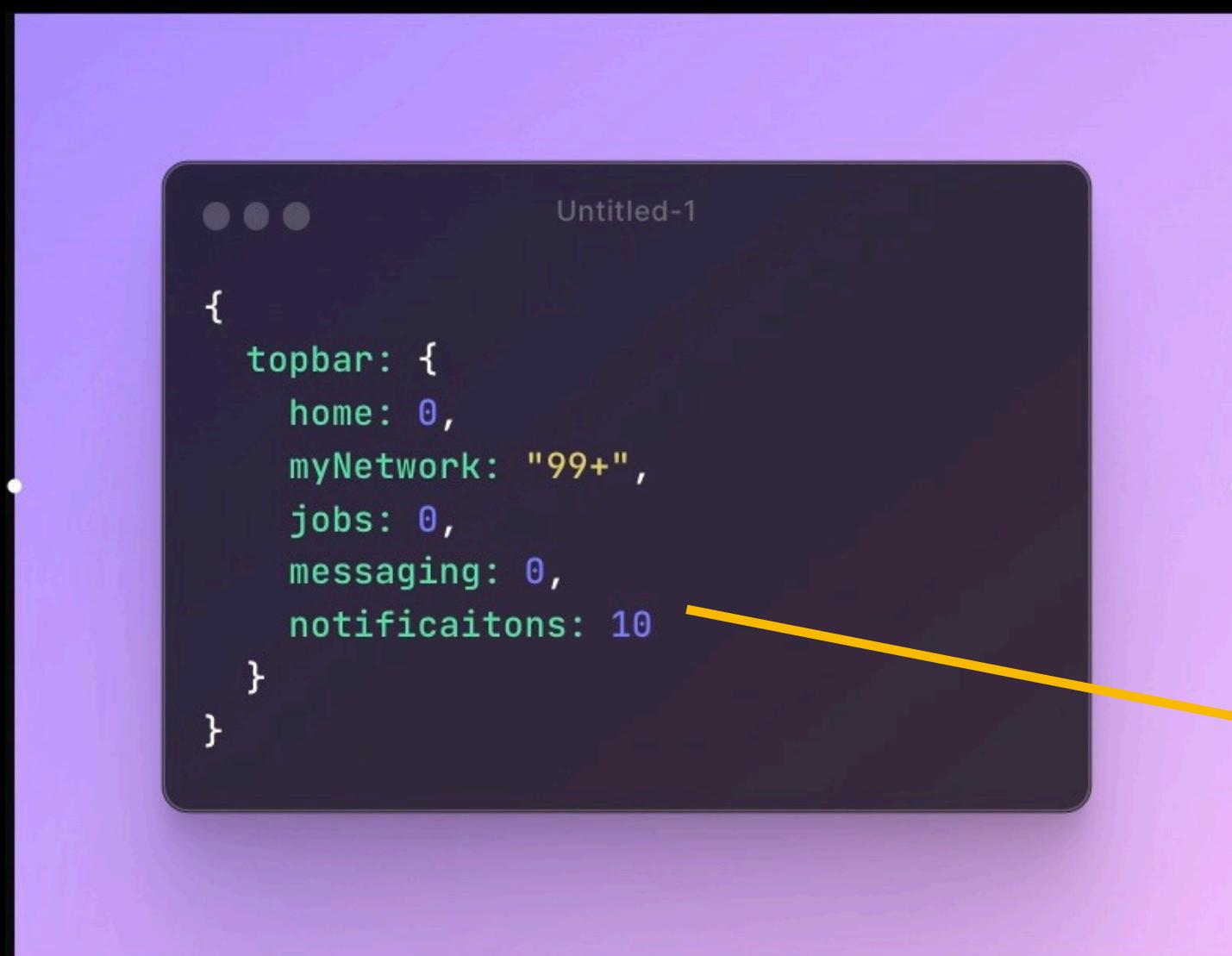
# State/Components/Re-rendering

This button is a component

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

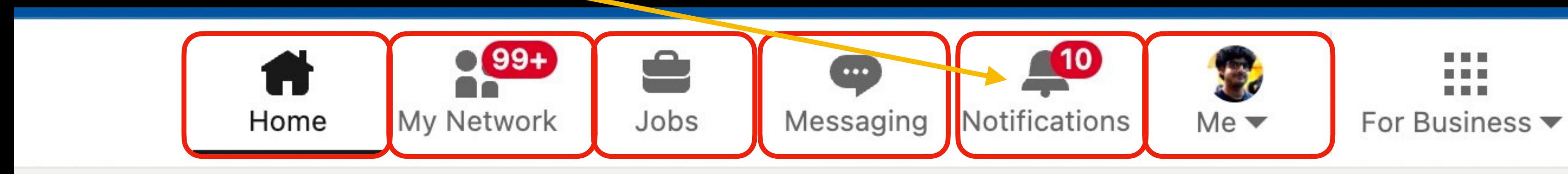


# State/Components/Re-rendering

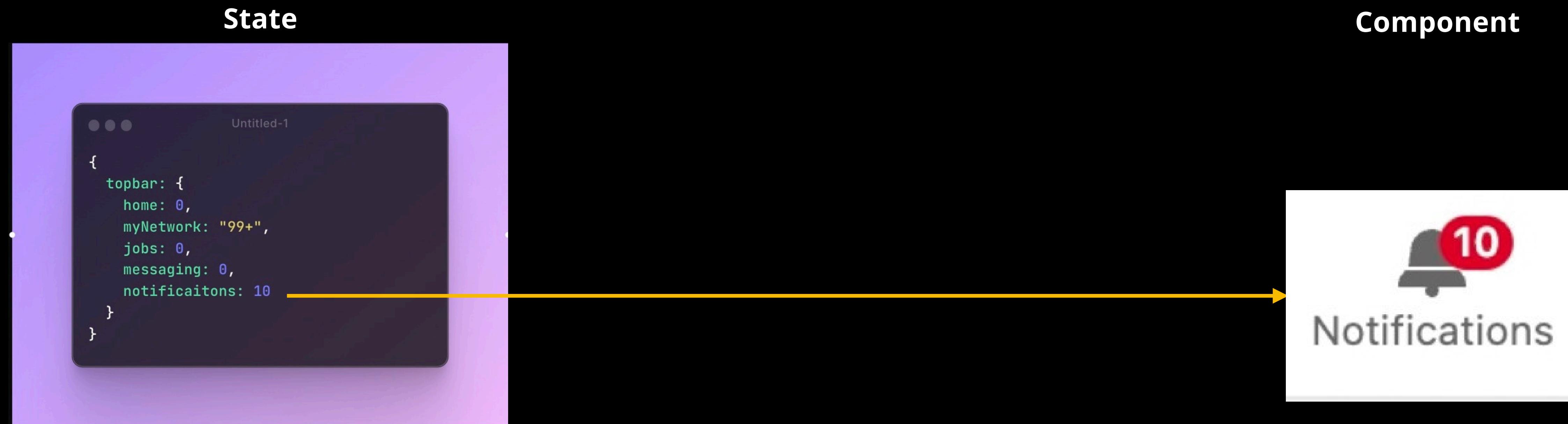


A screenshot of a code editor window titled "Untitled-1". The code is a JavaScript object representing a component's state:

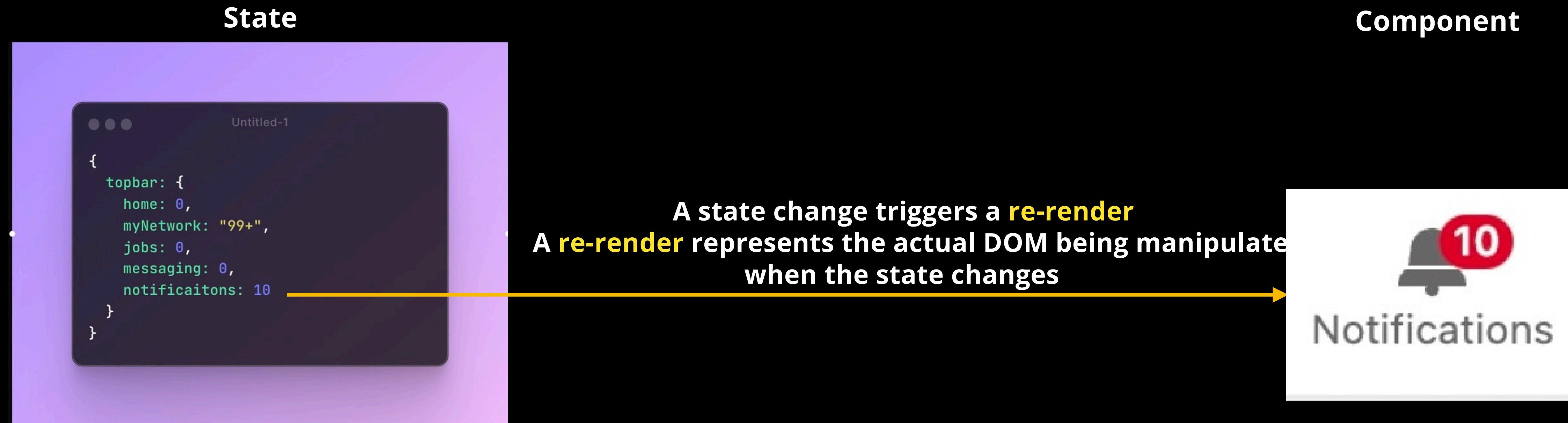
```
{  
  topbar: {  
    home: 0,  
    myNetwork: "99+",  
    jobs: 0,  
    messaging: 0,  
    notifications: 10  
  }  
}
```



# State/Components/Re-rendering

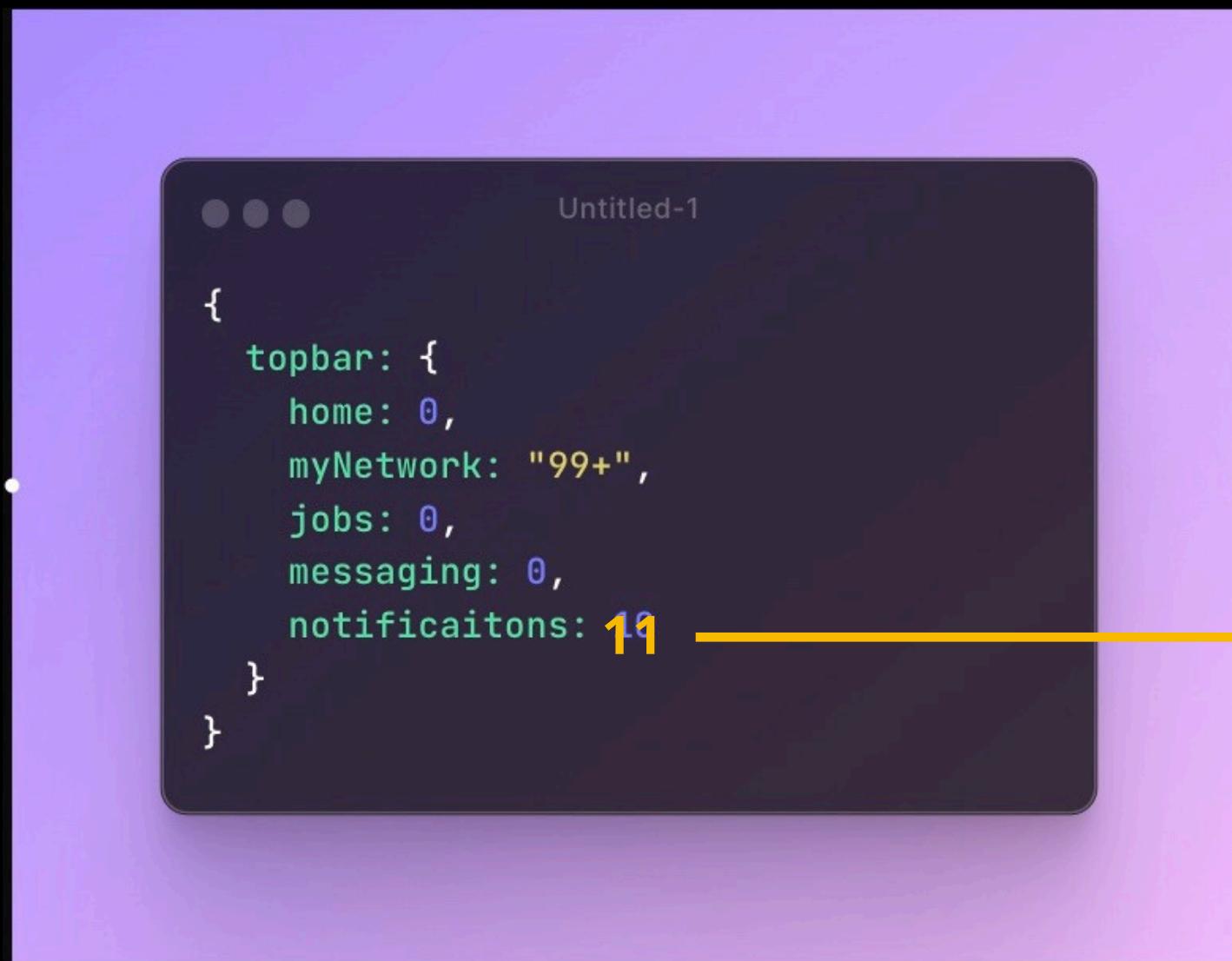


# State/Components/Re-rendering



# State/Components/Re-rendering

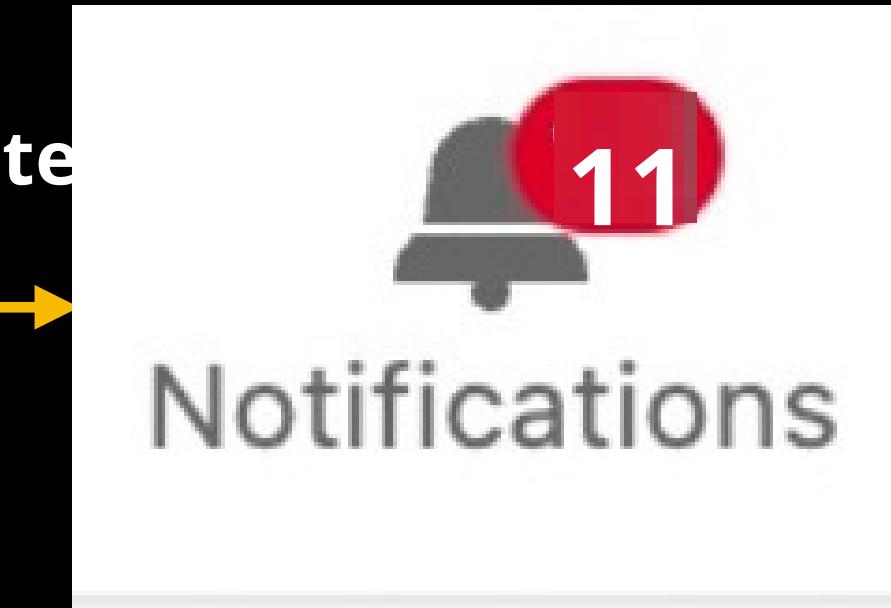
State



```
Untitled-1

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

Component



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

The screenshot shows a code editor on the left and a browser window on the right. The code editor displays the file `index.html` with the following content:

```
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>
```

The browser window shows a single button labeled "Counter 17". The browser's developer tools are open, showing the "Console" tab which is currently selected, the "Elements" tab, and the "Network", "Resources", "Dom", and "Settings" tabs. The console output is empty.

<https://gist.github.com/hkirat/c3d98735cec445e718b08f972dda7>

# Let's create a counter app using state/components

## 1. State initialisation

```
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>
```

# Let's create a counter app using state/components

## 2. Defining the 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>
```

# Let's create a counter app using state/components

The react library

```
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>
```

# The equivalent code in React looks like this

```
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.js
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

```
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 |
```

Lets start small, and then build up to this app

# The equivalent code in React looks like this

```
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
```

Lets start with a simple button component

# The equivalent code in React looks like this

Defining Button component

```
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
```

# 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
```

# The equivalent code in React looks like this

## Triggering re-render

```
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
```

# The equivalent code in React looks like this

Jsx syntax is a cleaner way to write components

```
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
```

```
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.

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