COS30049 – Computing Technology Innovation Project

Week 8 - Web Development Basics: Front-End with React

(Lecture – 02)

Ningran Li (Icey)
ningranli@swin.edu.au

Acknowledgement of Country

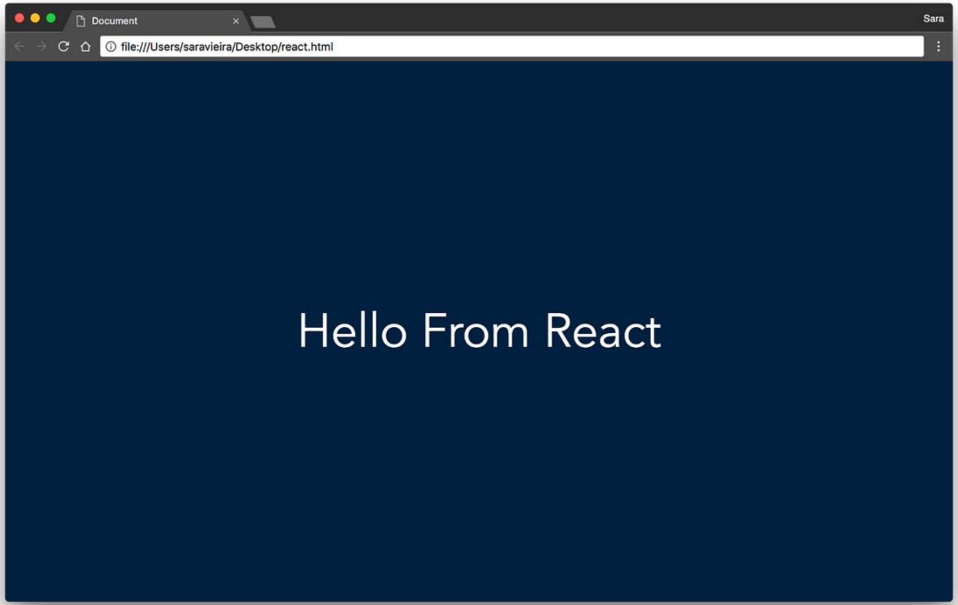
We respectfully acknowledge the Wurundjeri People of the Kulin Nation, who are the Traditional Owners of the land on which Swinburne's Australian campuses are located in Melbourne's east and outer-east, and pay our respect to their Elders past, present and emerging.

We are honoured to recognise our connection to Wurundjeri Country, history, culture, and spirituality through these locations, and strive to ensure that we operate in a manner that respects and honours the Elders and Ancestors of these lands.

We also respectfully acknowledge Swinburne's Aboriginal and Torres Strait Islander staff, students, alumni, partners and visitors.

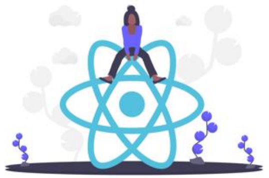
We also acknowledge and respect the Traditional Owners of lands across Australia, their Elders, Ancestors, cultures, and heritage, and recognise the continuing sovereignties of all Aboriginal and Torres Strait Islander Nations.

Key Usage of React



Hello From React

Enriching the UI



1. Component-based UI
   - Allows developer to break down the UI into reusable and modular components
   - Easily compose them to build complex and dynamic UI
2. Styling and CSS
   - Style components using CSS
   - Either through inline styles or by importing external CSS files
3. UI libraries and frameworks
   - By leveraging these libraries, we can quickly build modern and professional-looking UI without starting from scratch
4. Responsive Design
   - Adapt to different screen sizes and devices (e.g. web app)
5. Animations and transitions

Building Interactive Pages

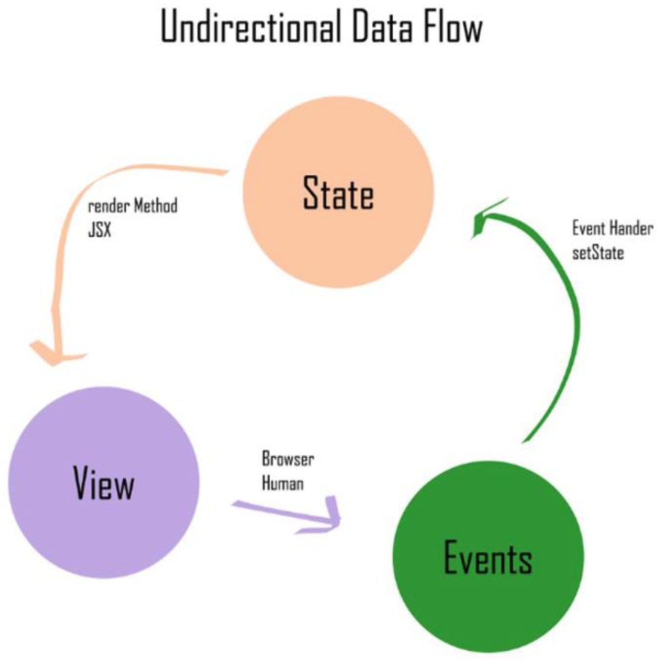
1. Event handling
   - Attach event handlers to components, enabling interactivity and user input
   - Define functions that respond to user events like clicks, key presses, form submissions, and more
   - Trigger actions, update component stat or initiate data fetching based on user interactions
2. State management
   - Reacts uses state to manage and update the data within components
   - Each component can have is own local state
3. Conditional rendering
4. Real-time updates

Understanding React’s Unidirectional Data Flow

In React, data flows in one direction, ensuring predictability and consistency. Here’s how it works:

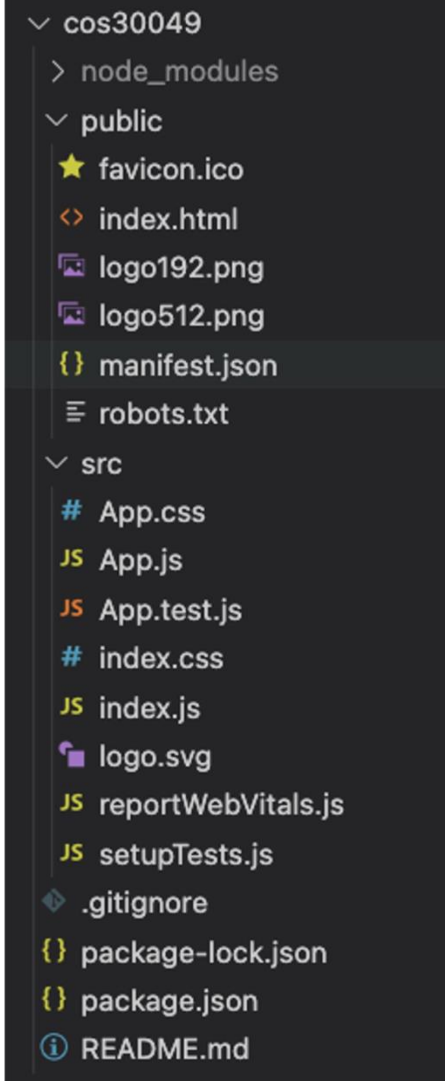
1. **State**: Stores data and determines the UI.
2. **View**: Displays the UI based on the state.
3. **Events**: User actions trigger events, which update the state.

The cycle continues as updated state triggers a re-render of the view. This flow simplifies debugging and ensures that changes are easily traceable from state to view and back through events.



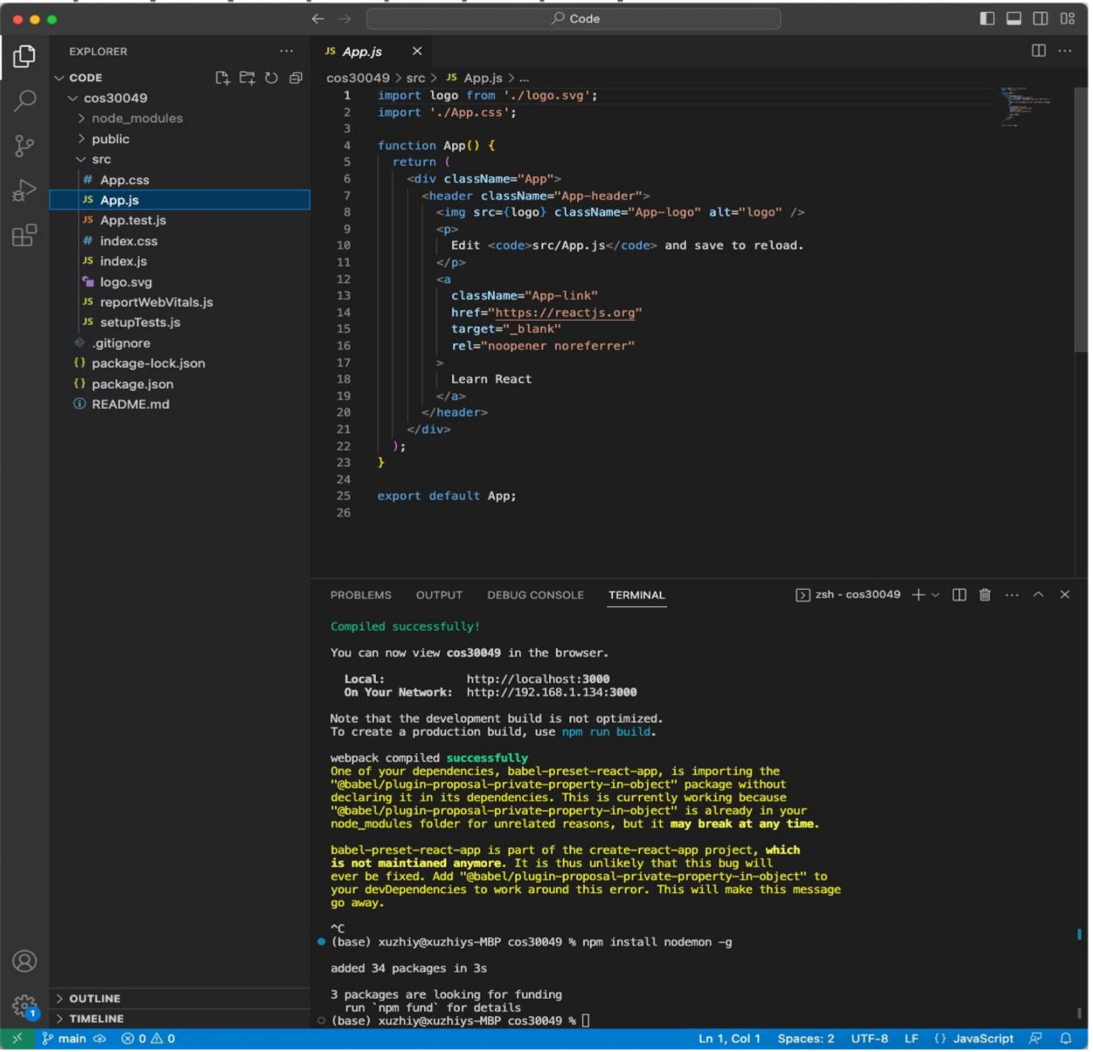
Structure of React

- **node_modules**: dependencies
- **public folder**: store static file (html, img, video …..)
- **src folder**: React source code, most of your codes should be here



```
cos30049
  node_modules
  public
    favicon.ico
    index.html
    logo192.png
    logo512.png
    manifest.json
    robots.txt
  src
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
    reportWebVitals.js
    setupTests.js
  .gitignore
  package-lock.json
  package.json
  README.md
```

Structure of React



```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

```
Compiled successfully!

You can now view cos30049 in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.1.134:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
One of your dependencies, babel-preset-react-app, is importing the
"@babel/plugin-proposal-private-property-in-object" package without
declaring it in its dependencies. This is currently working because
"@babel/plugin-proposal-private-property-in-object" is already in your
node_modules folder for unrelated reasons, but it may break at any time.

babel-preset-react-app is part of the create-react-app project, which
is not maintianed anymore. It is thus unlikely that this bug will
ever be fixed. Add "@babel/plugin-proposal-private-property-in-object" to
your devDependencies to work around this error. This will make this message
go away.

^C
(base) xuzhiy@xuzhiys-MBP cos30049 % npm install nodemon -g

added 34 packages in 3s

3 packages are looking for funding
  run `npm fund` for details
(base) xuzhiy@xuzhiys-MBP cos30049 %
```

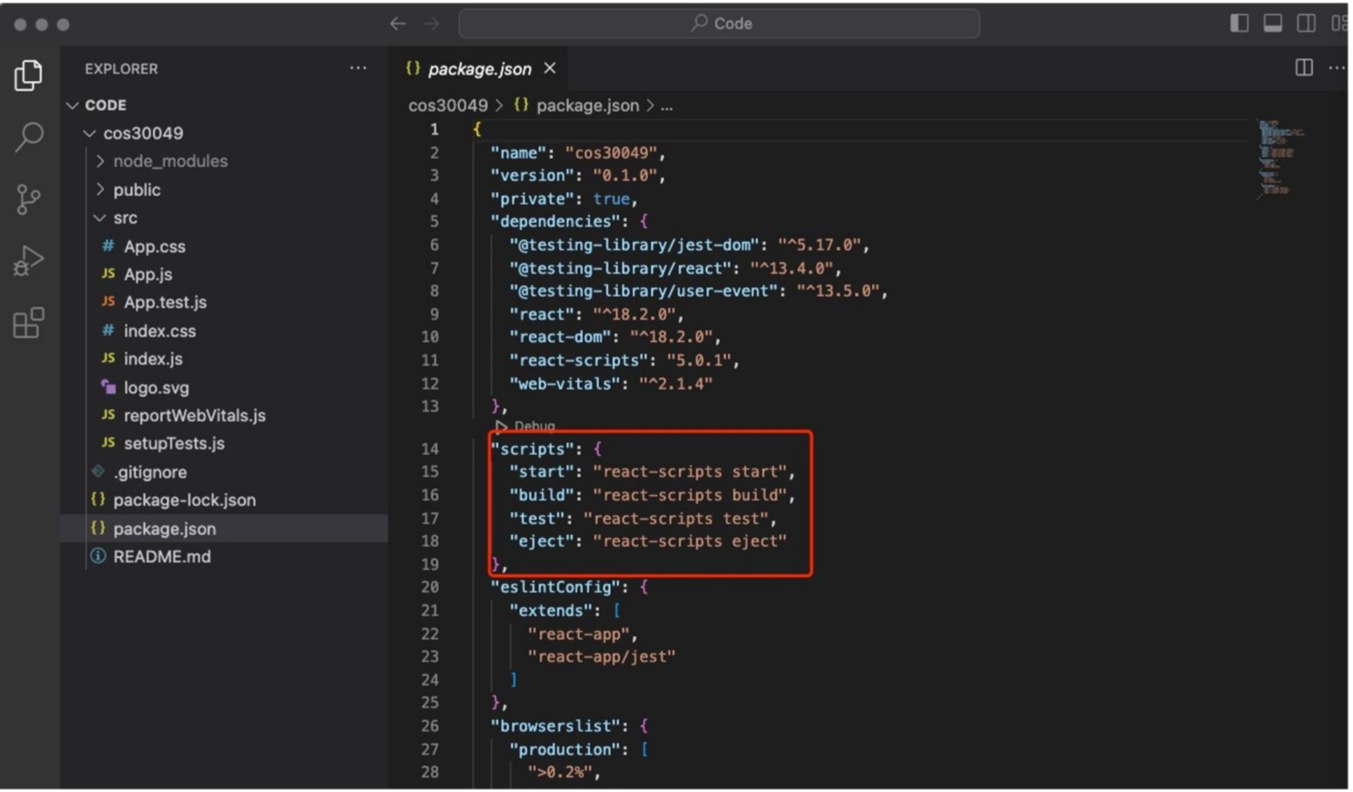
App.js: The **main root component** where you build your app's structure and logic, which is imported and rendered by index.j

Run the project: npm start

Image from CRVS Digitisation Guidebook

Structure of React

package.json - a key file in any React project



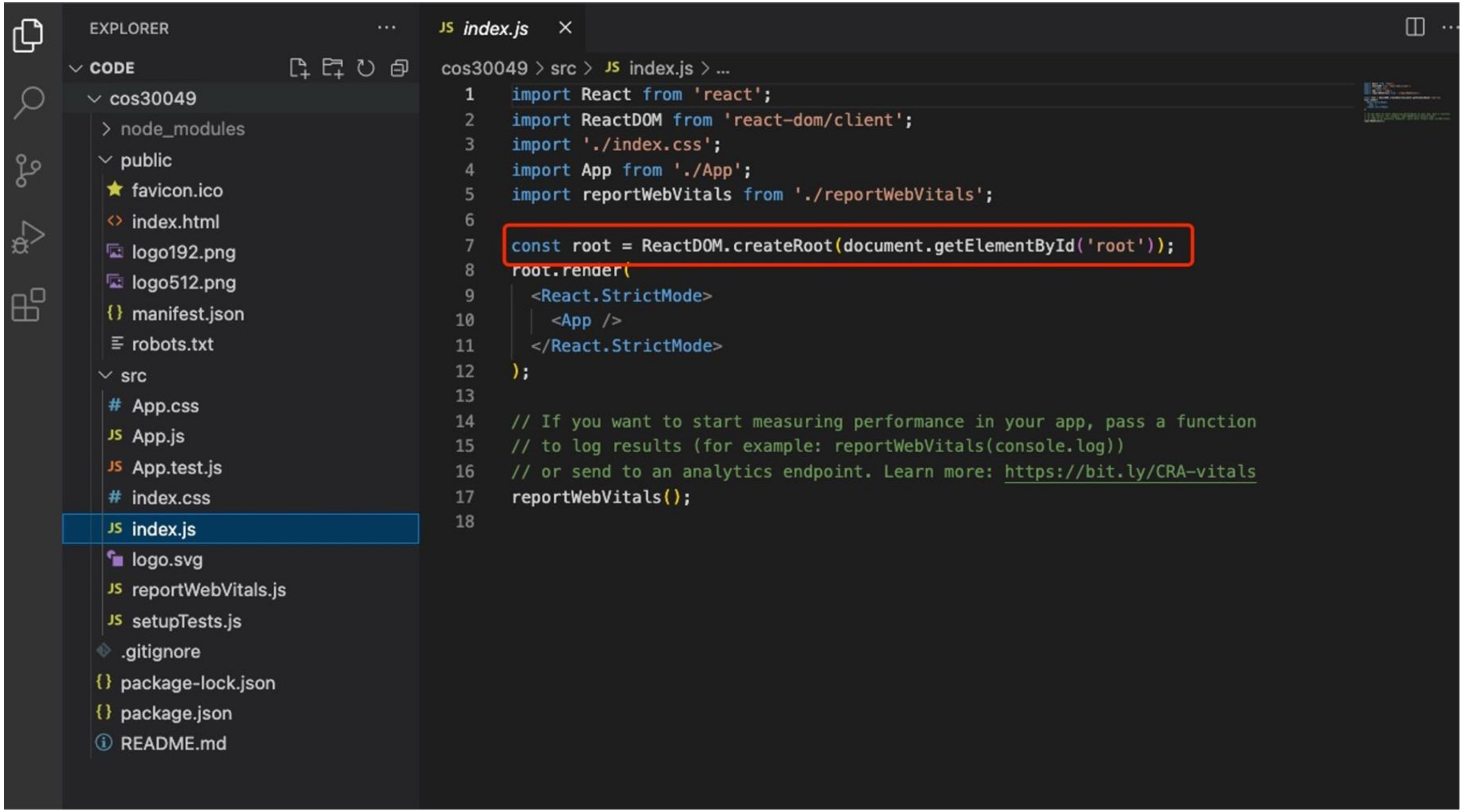
```json
{
  "name": "cos30049",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
```

More commands can be found in package.json

Image from CRVS Digitisation Guidebook

Structure of React

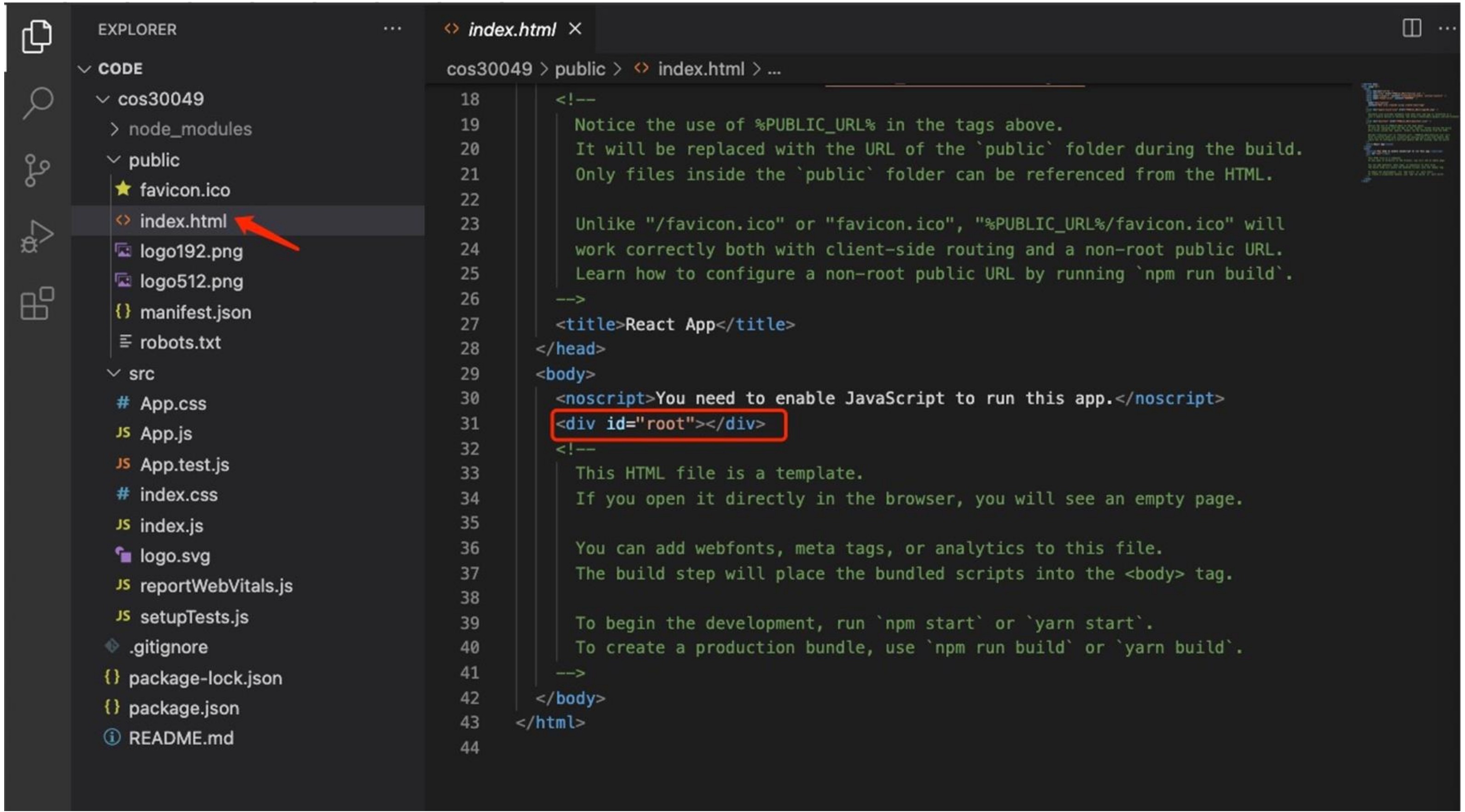
Index.js - true entry point of the React application



```javascript
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

Structure of React

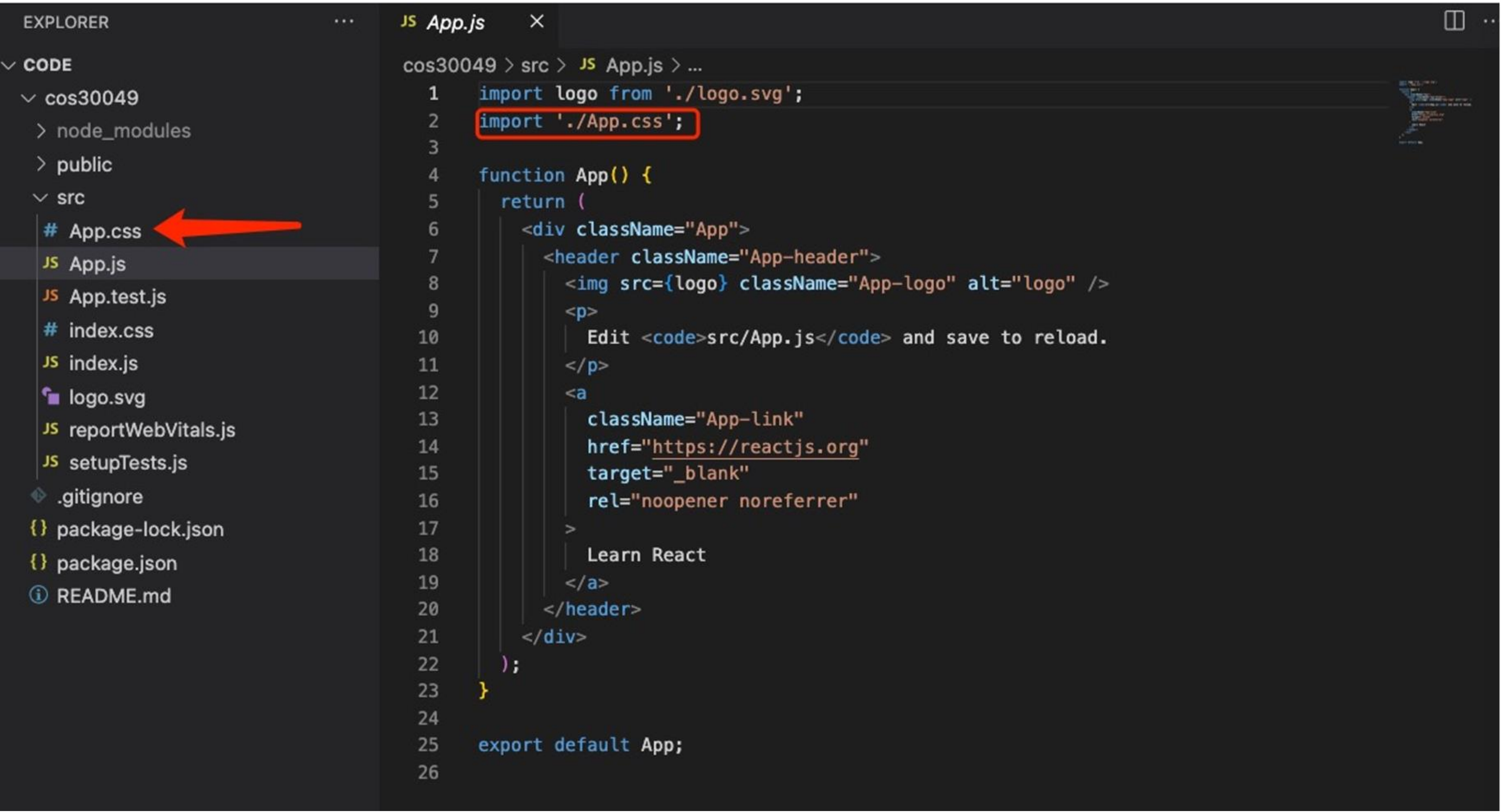
Index.html- the **base HTML file** for your entire React application



```html
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>
```

Styling in React

App.css: You can define the elements' CSS in separate files.

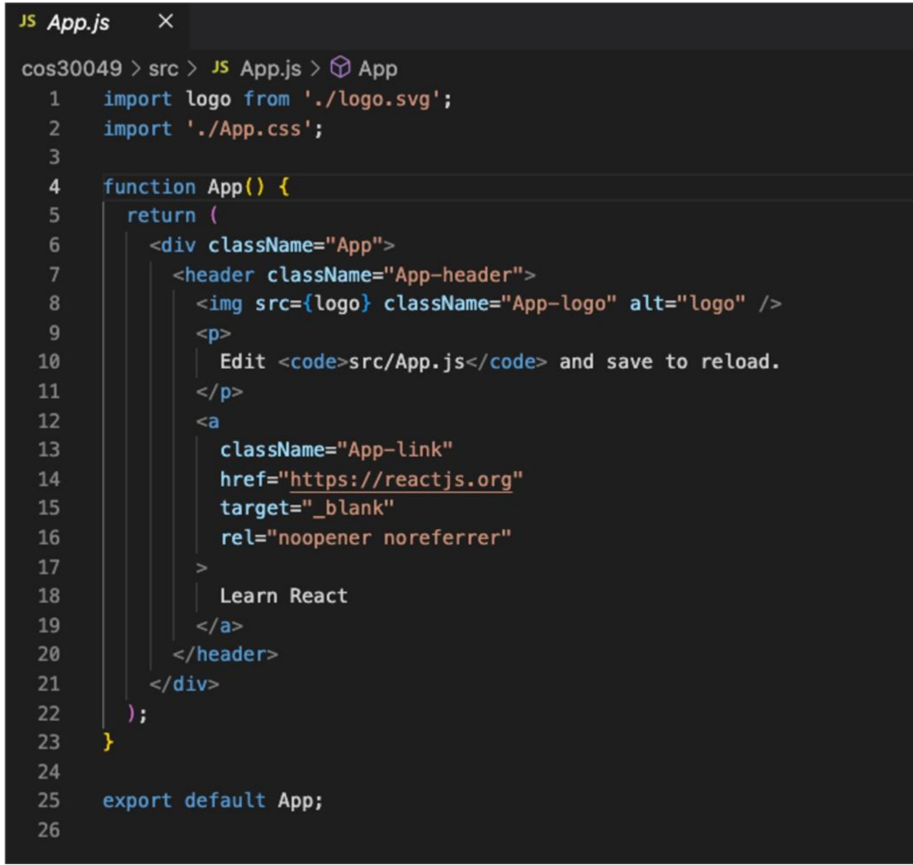


```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

Basic Concept of React

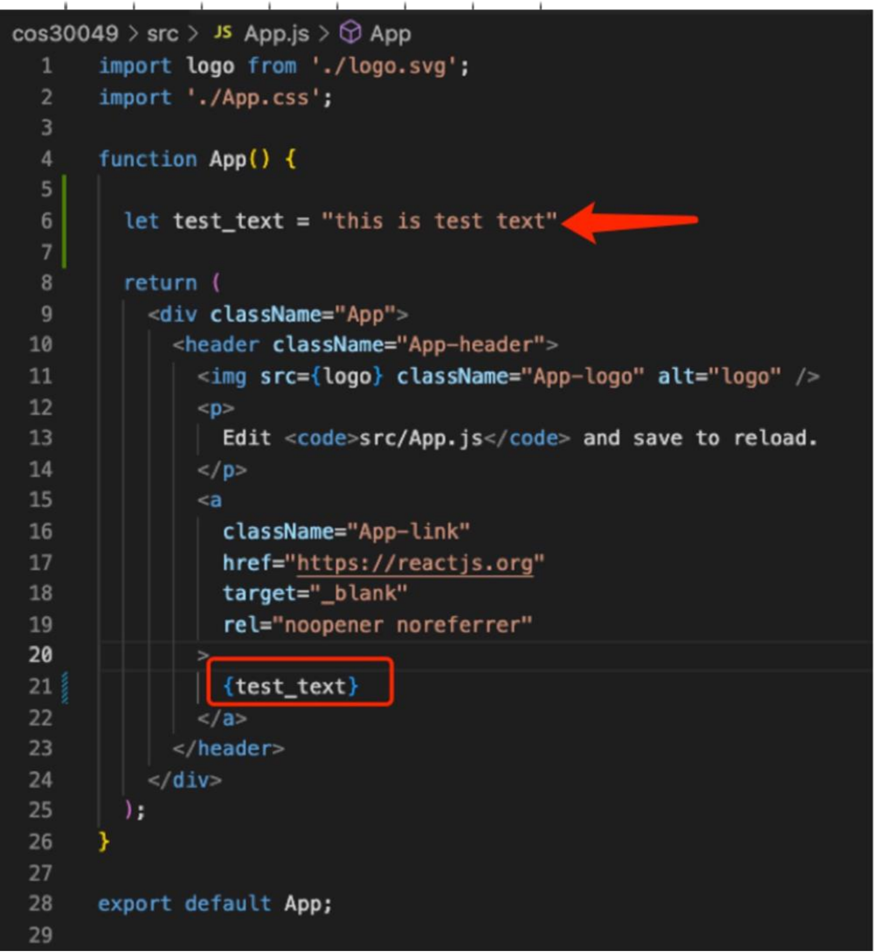
JSX (JavaScript XML) is a JavaScript syntax extension that allows you to write XML or HTML-like structures within your JavaScript code. In the context of React, JSX is widely used to describe the structure and appearance of UI components.

The purpose of JSX is to simplify the process of creating and rendering React components, making the code more readable and writable. It enables developers to write markup similar to HTML directly within JavaScript, without the need to manually construct virtual DOM elements.



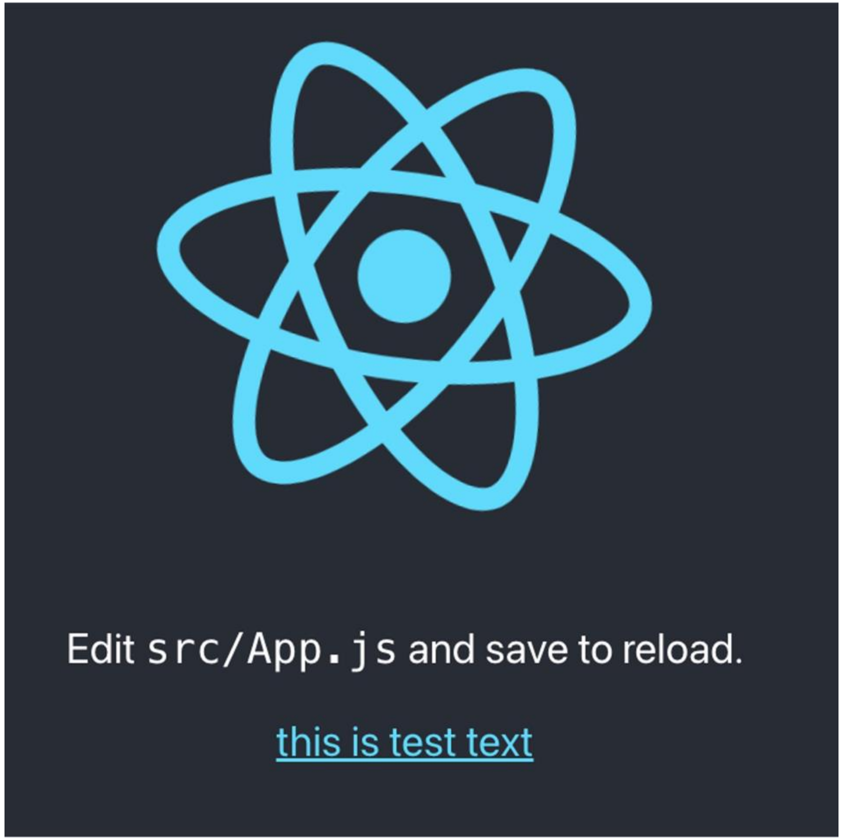
```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;

```

Basic Concept of React



cos30049 > src > JS App.js > App

```
import logo from './logo.svg';
import './App.css';

function App() {

  let test_text = "this is test text"

  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          {test_text}
        </a>
      </header>
    </div>
  );
}

export default App;

```



Edit `src/App.js` and save to reload.

this is test text

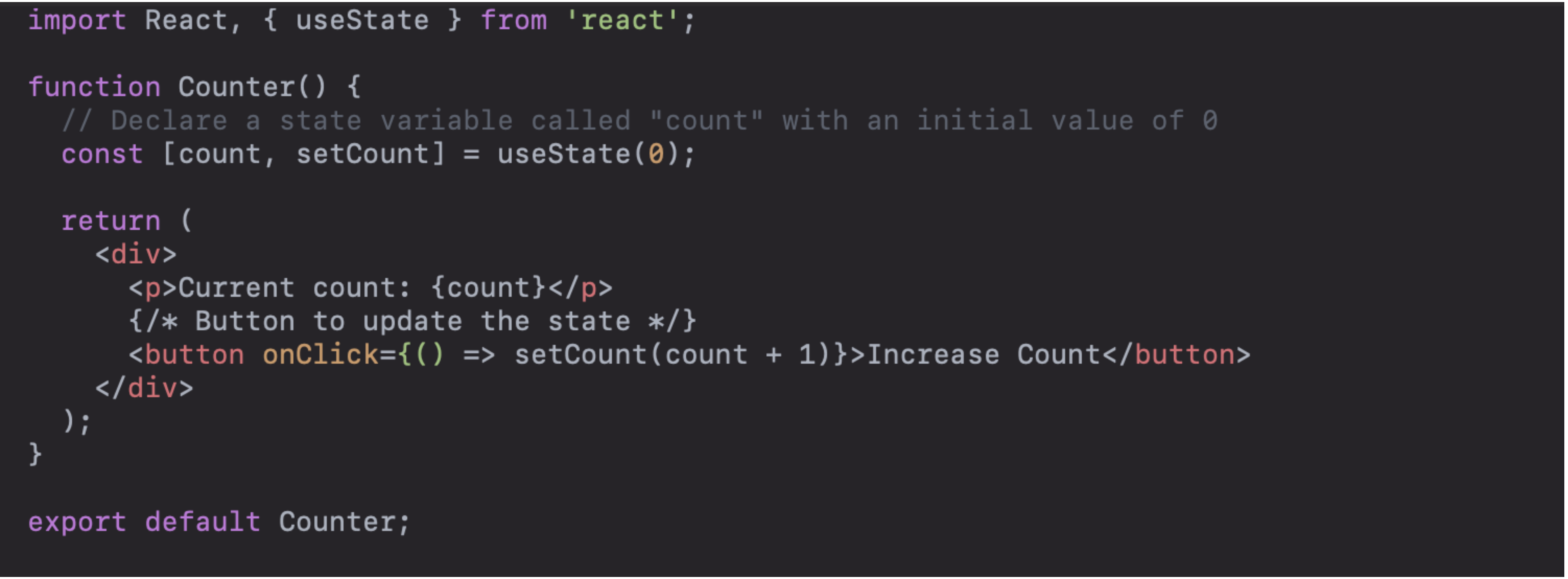
State in React.js

What is State:

State is an object in React that stores dynamic data about the component. It represents the **current status** of the component.

- State allows React components to keep track of changing information—like user input, timers, or other data that might change over time.
- When the state of a component changes, React **re-renders** the component to update the UI with the latest values

State in React.js



```
import React, { useState } from 'react';

function Counter() {
  // Declare a state variable called "count" with an initial value of 0
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Current count: {count}</p>
      {/* Button to update the state */}
      <button onClick={() => setCount(count + 1)}>Increase Count</button>
    </div>
  );
}

export default Counter;
```

Declare a **state variable called "count"** with an initial value of 0

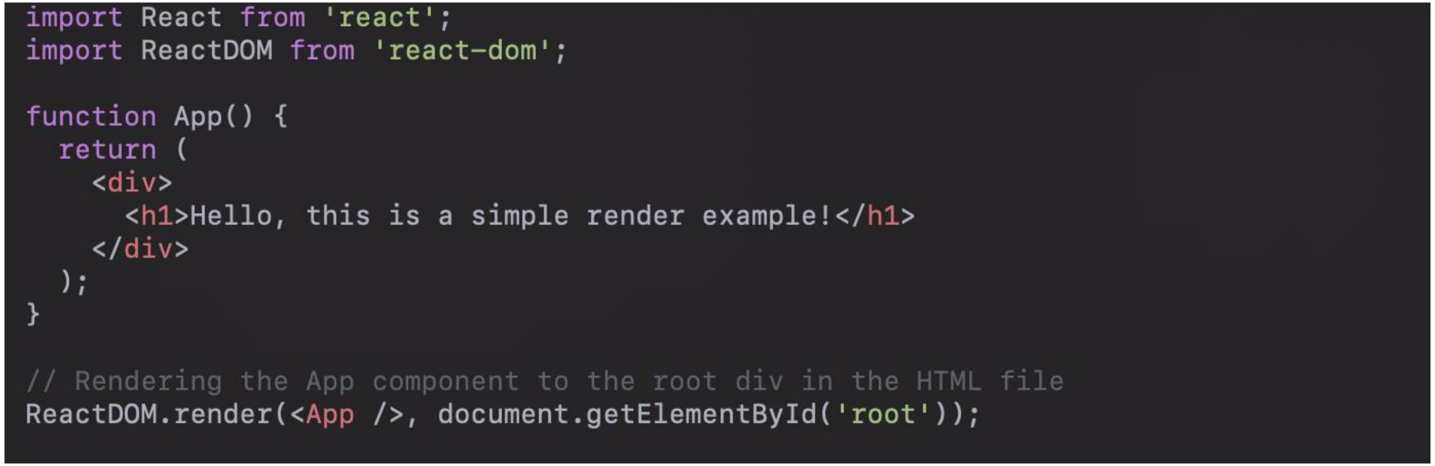
Render in React.js

- The **render** process is how React takes the JSX code (the HTML-like structure inside your JavaScript) and **turns it into actual HTML** that gets displayed in the browser.
- Whenever a React component's **state** or **props** change, React automatically triggers a **re-render** to update the UI.

Render Cycle:
- The **render** method in React describes what the UI should look like at any given time. React then uses this information to update the **DOM**.
- Every time the component's state changes, React **re-renders** the component to ensure that the DOM stays in sync with the current state.

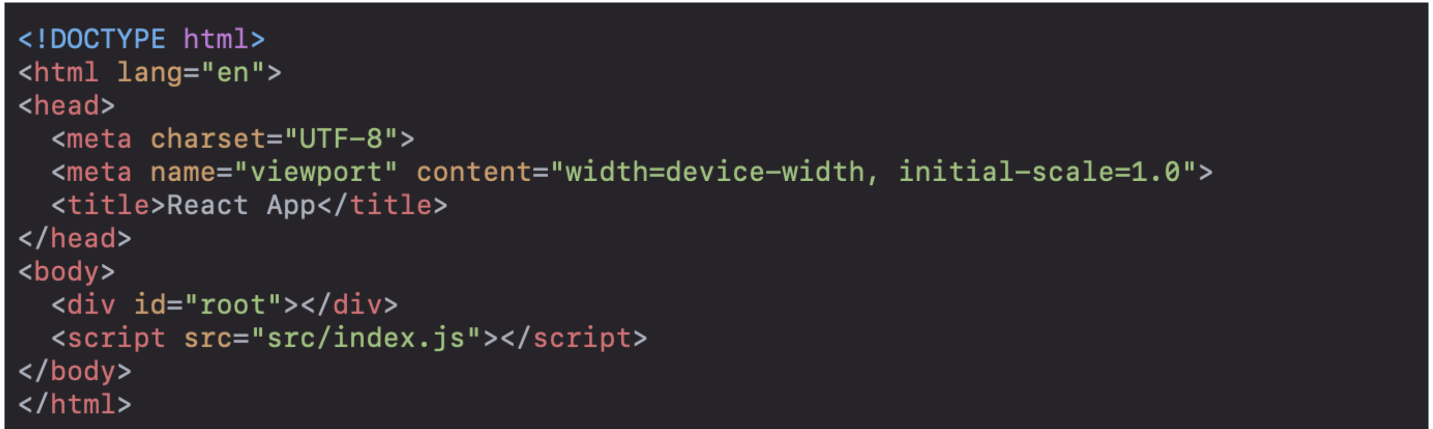
Example of Re-render:
- In the previous example, when the count state is updated, React **re-renders** the component to show the new count on the screen.
- The re-render happens because React **detects the state change** (in this case, when setCount is called).

Render in React.js



```
import React from 'react';
import ReactDOM from 'react-dom';

function App() {
  return (
    <div>
      <h1>Hello, this is a simple render example!</h1>
    </div>
  );
}

// Rendering the App component to the root div in the HTML file
ReactDOM.render(<App />, document.getElementById('root'));
```

HTML file should have a div with an id of root, like this:



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>React App</title>
</head>
<body>
  <div id="root"></div>
  <script src="src/index.js"></script>
</body>
</html>
```

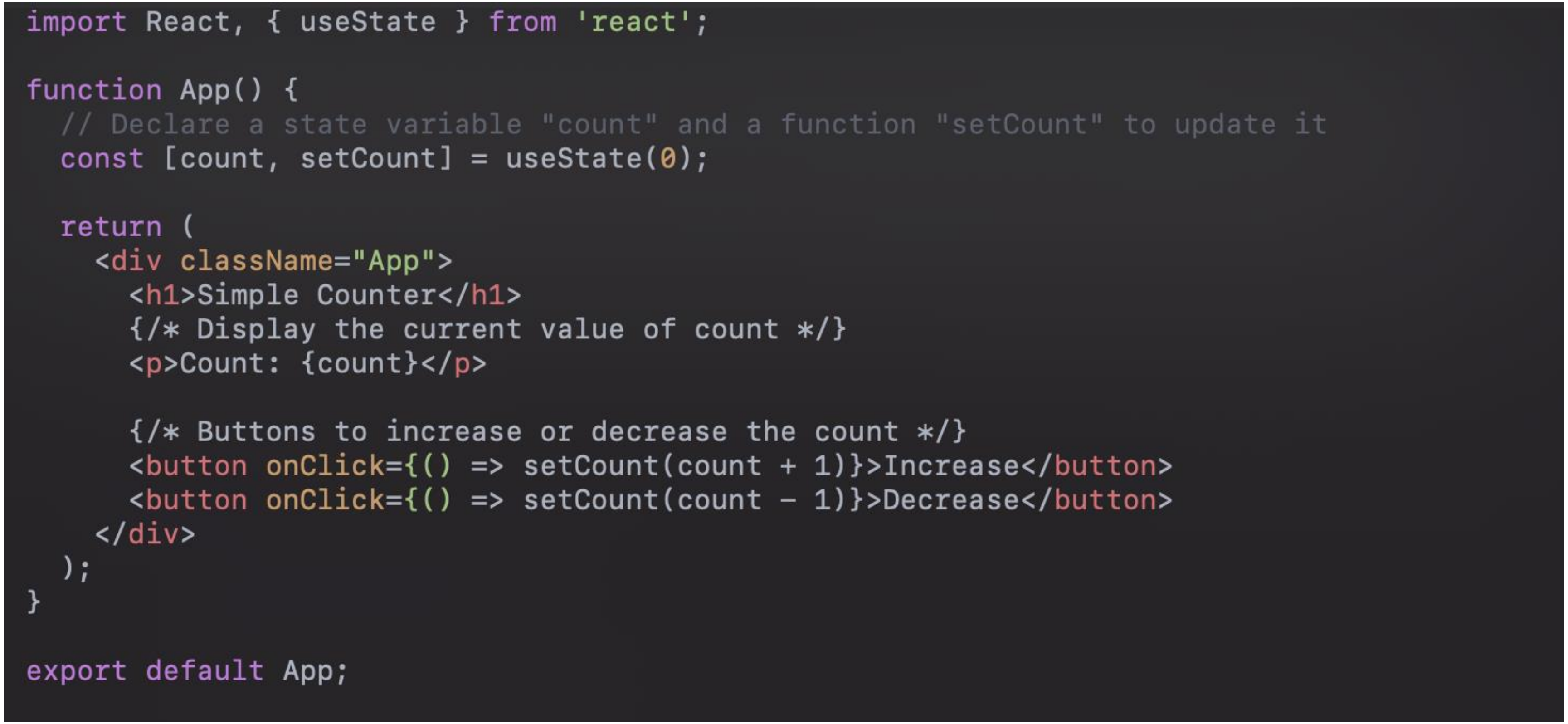
How State and Render Work Together?

• React uses **state** to store and manage dynamic data. When the state changes, React **re-renders** the component to update the UI with the new state.
• This automatic re-rendering is what makes React apps **interactive** and **responsive** to user input or other dynamic events.

Summary:

• **State**: Holds dynamic data that can change over time (e.g., user inputs, API responses).
• **Render**: The process React uses to convert JSX into actual HTML elements and update the browser when the state or props change.

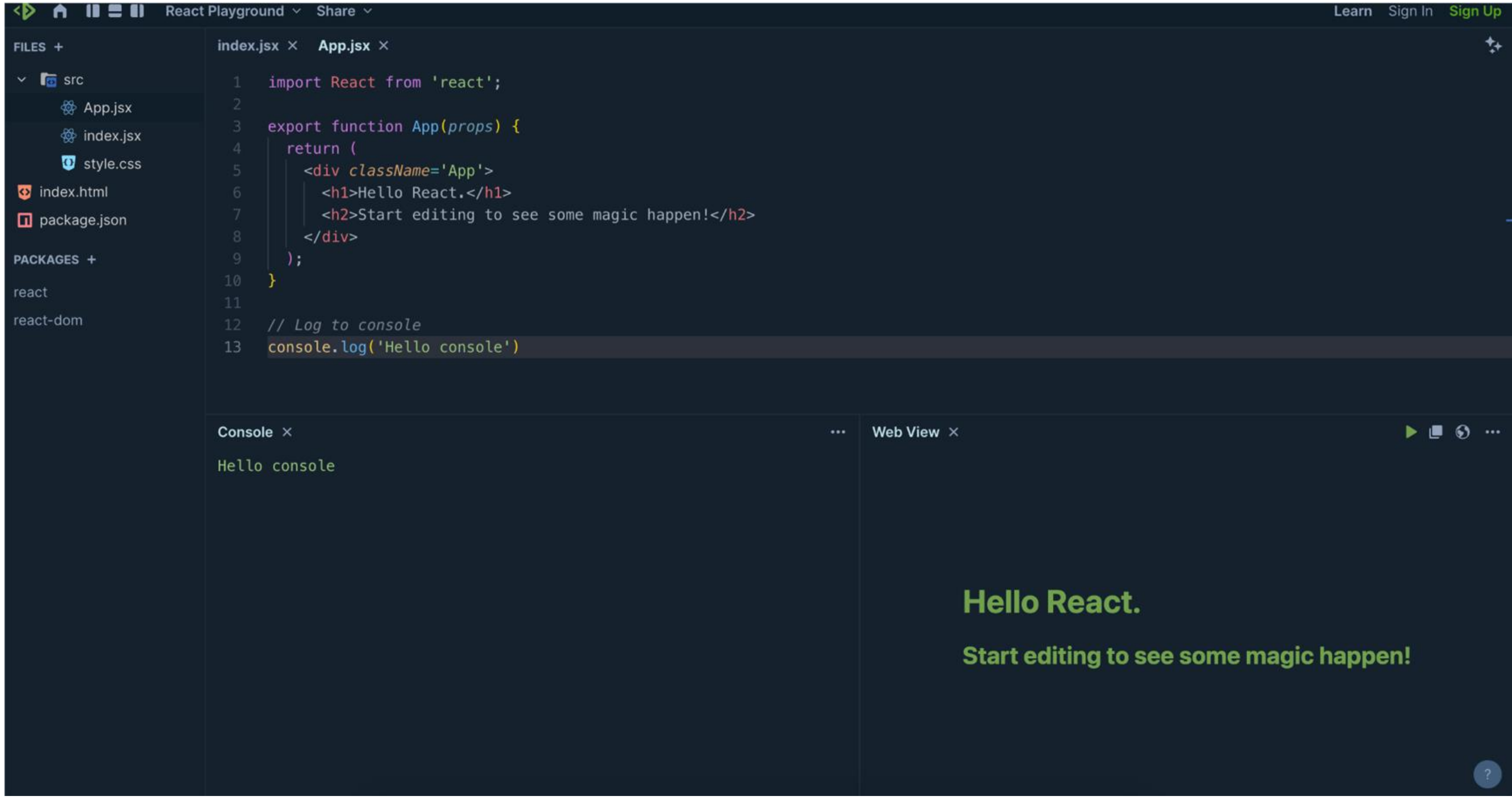
How State and Render Work Together?



```
import React, { useState } from 'react';

function App() {
  // Declare a state variable "count" and a function "setCount" to update it
  const [count, setCount] = useState(0);

  return (
    <div className="App">
      <h1>Simple Counter</h1>
      {/* Display the current value of count */}
      <p>Count: {count}</p>

      {/* Buttons to increase or decrease the count */}
      <button onClick={() => setCount(count + 1)}>Increase</button>
      <button onClick={() => setCount(count - 1)}>Decrease</button>
    </div>
  );
}

export default App;
```

Web Design: Building Your Web with React.js

React Playground : playcode.io/react



```
import React from 'react';

export function App(props) {
  return (
    <div className='App'>
      <h1>Hello React.</h1>
      <h2>Start editing to see some magic happen!</h2>
    </div>
  );
}

// Log to console
console.log('Hello console')
```

```
Hello console
```

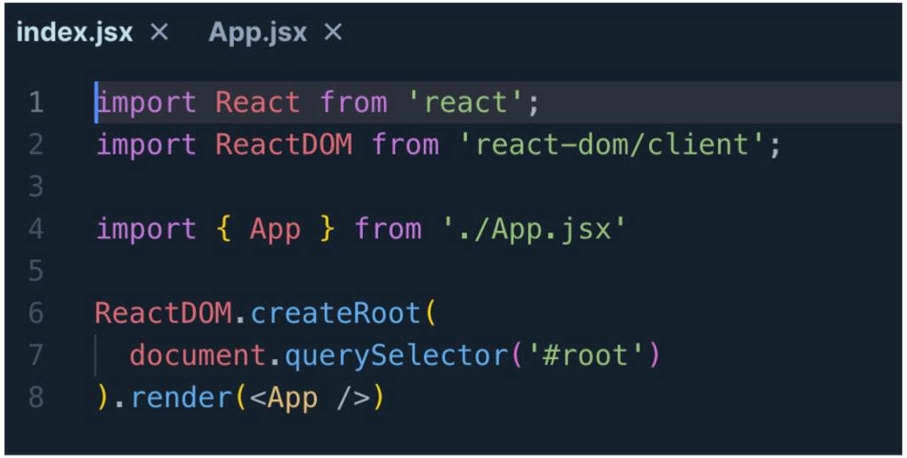
Hello React.

Start editing to see some magic happen!

Web Design: Building Your Web with React.js

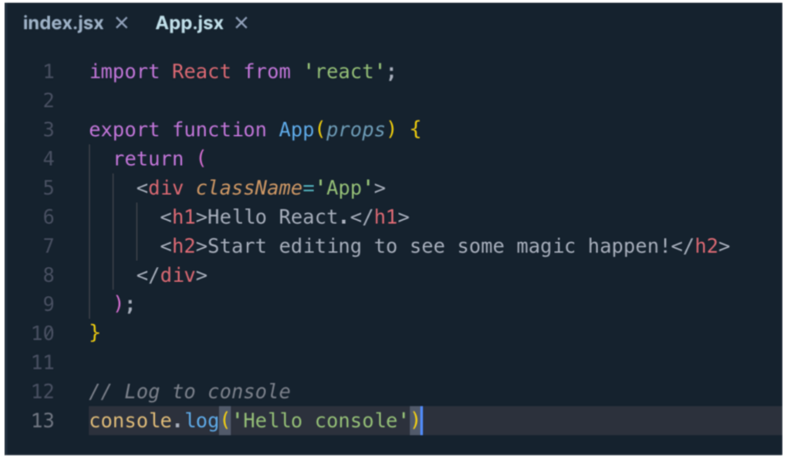
In React, the ***render*** function is a required method to define the output of a component. Every React component must contain a render function that is responsible for returning a React element or a set of React elements that describe the output of the component on the screen.

The main purpose of the render function is to generate a virtual DOM based on the component's current state (state) and properties (props) and render it into the actual DOM.



index.jsx × App.jsx ×

```
import React from 'react';
import ReactDOM from 'react-dom/client';

import { App } from './App.jsx'

ReactDOM.createRoot(
  document.querySelector('#root')
).render(<App />)
```

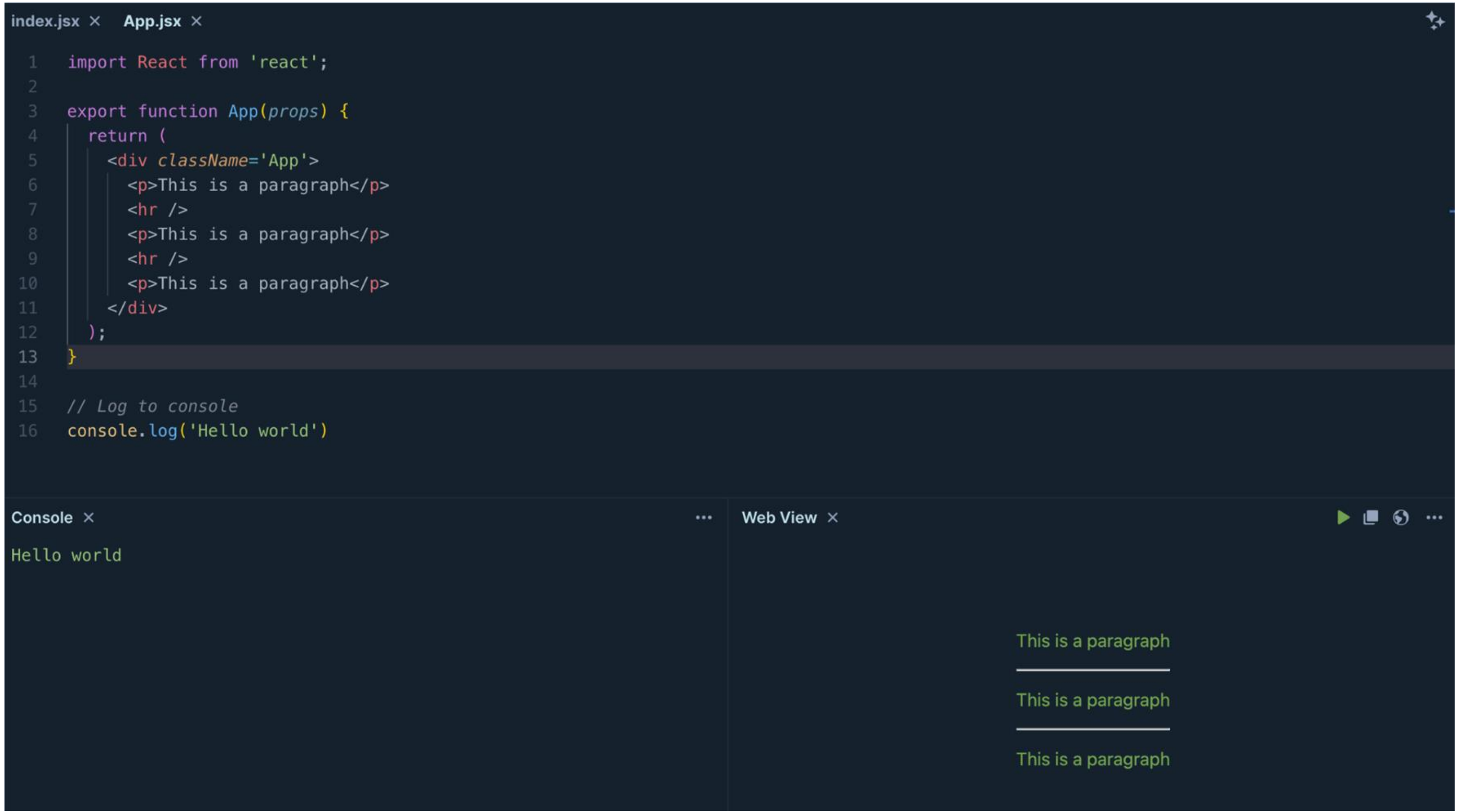


index.jsx × App.jsx ×

```
import React from 'react';

export function App(props) {
  return (
    <div className='App'>
      <h1>Hello React.</h1>
      <h2>Start editing to see some magic happen!</h2>
    </div>
  );
}

// Log to console
console.log('Hello console')
```

Web Design: Integrating HTML in React.js

In React, HTML elements are also support in the React.js



```
import React from 'react';

export function App(props) {
  return (
    <div className='App'>
      <p>This is a paragraph</p>
      <hr />
      <p>This is a paragraph</p>
      <hr />
      <p>This is a paragraph</p>
    </div>
  );
}

// Log to console
console.log('Hello world')
```

```
Hello world
```

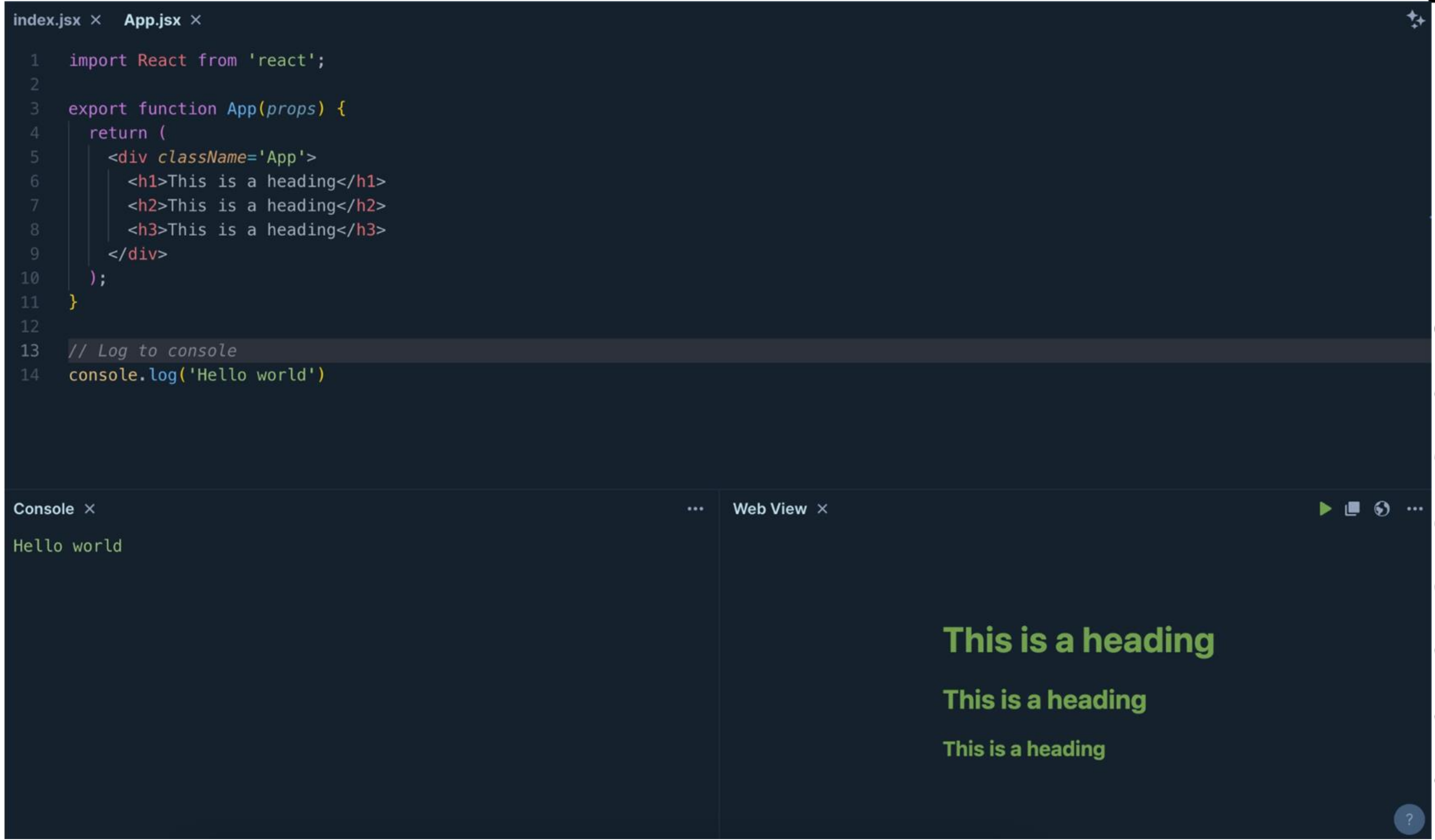
This is a paragraph

This is a paragraph

This is a paragraph

Web Design: Integrating HTML in React.js

In React, HTML elements are also support in the React.js



```
import React from 'react';

export function App(props) {
  return (
    <div className='App'>
      <h1>This is a heading</h1>
      <h2>This is a heading</h2>
      <h3>This is a heading</h3>
    </div>
  );
}

// Log to console
console.log('Hello world')
```

```
Hello world
```

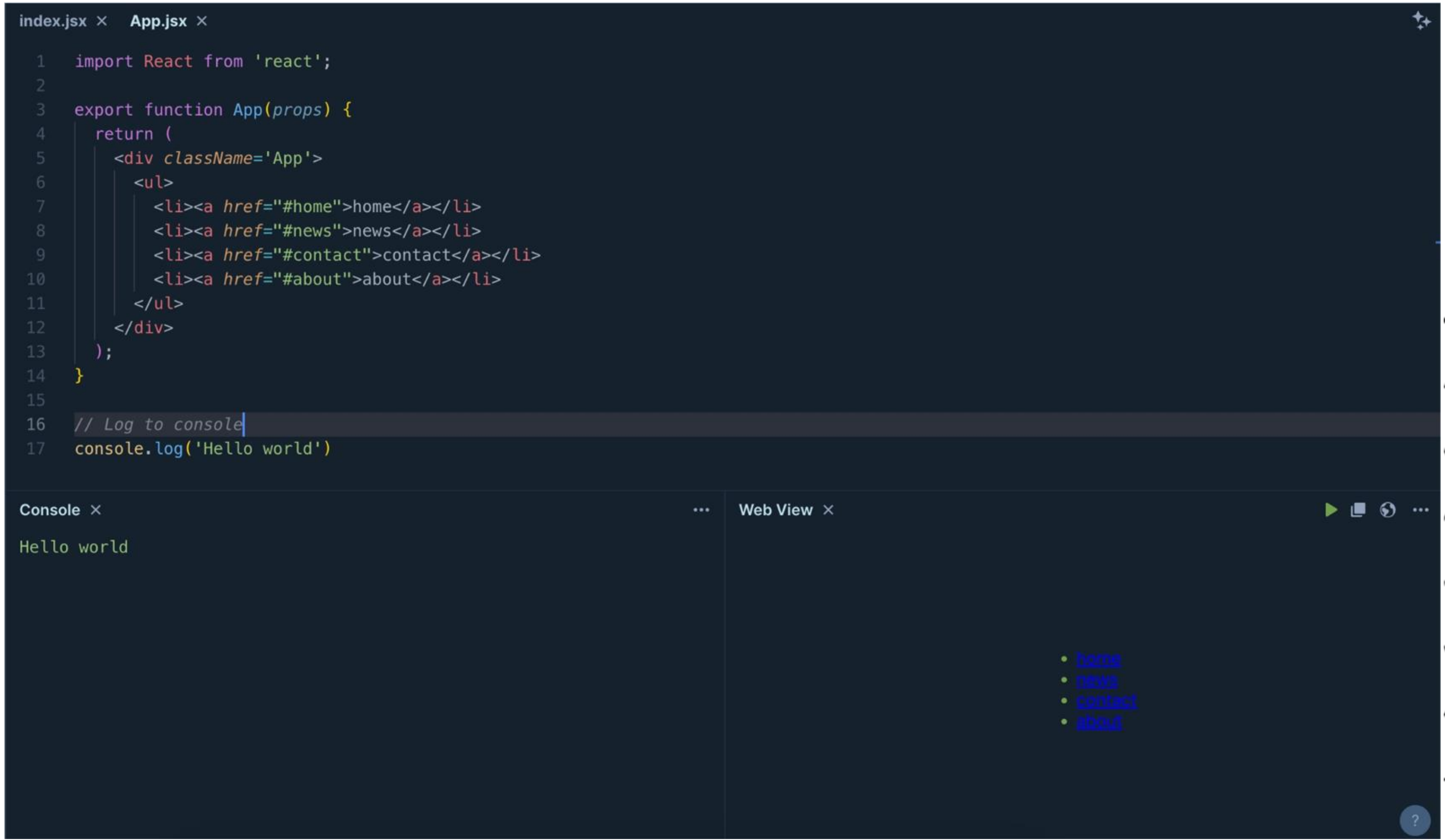
This is a heading

This is a heading

This is a heading

Web Design: Integrating HTML in React.js

In React, HTML elements are also support in the React.js

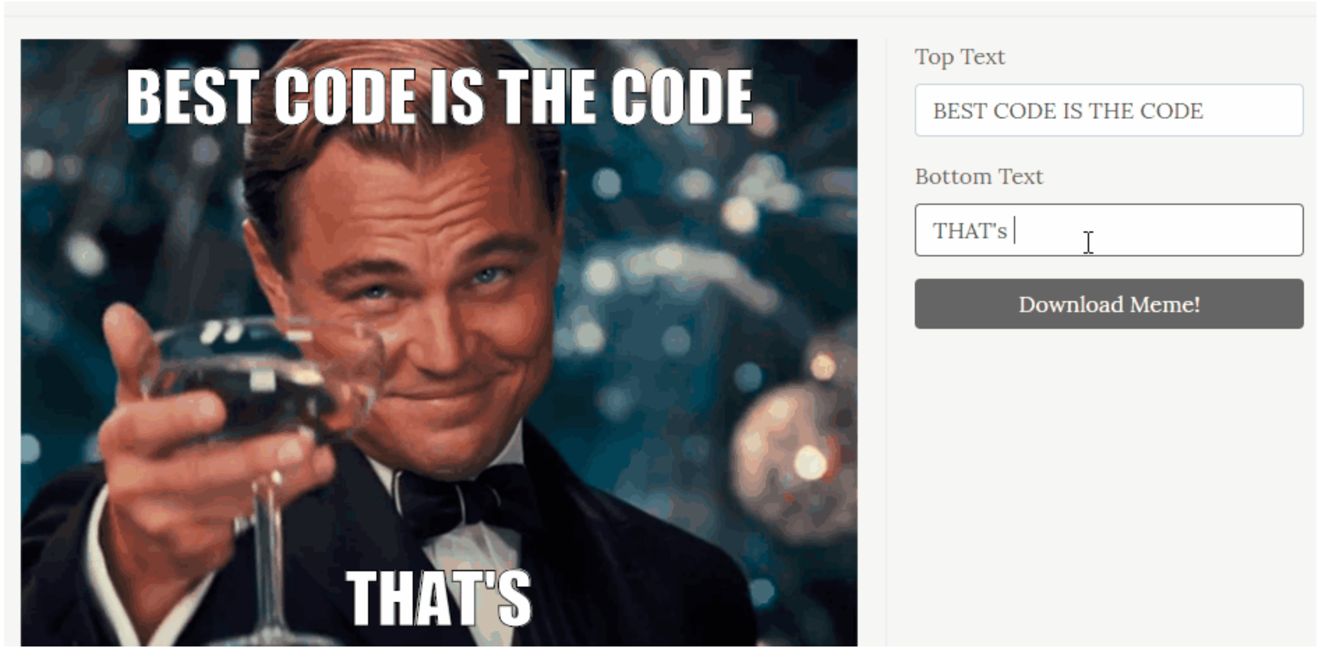


```jsx
import React from 'react';

export function App(props) {
  return (
    <div className='App'>
      <ul>
        <li><a href="#home">home</a></li>
        <li><a href="#news">news</a></li>
        <li><a href="#contact">contact</a></li>
        <li><a href="#about">about</a></li>
      </ul>
    </div>
  );
}

// Log to console
console.log('Hello world')
```

```
Hello world
```

- home
- news
- contact
- about

Any Questions?



Top Text: BEST CODE IS THE CODE

Bottom Text: THAT's

Image Source:[React for Beginners: Building a Meme Maker with React. (2023). *FreeCodeCamp*]

Learning Resources

MUI Documentation
https://mui.com/material-ui/getting-started/

What is Flexbox ?
https://css-tricks.com/snippets/css/a-guide-to-flexbox/

What is the Grid system ?
https://mui.com/material-ui/react-grid/

Breakpoints in MUI
https://mui.com/material-ui/customization/breakpoints/

Link in MUI
https://mui.com/material-ui/react-link/

Image from CRVS Digitisation Guidebook