PrepBytes

React JS

What is React ?

- Developed by Facebook in 2013. (Jordan Walke)
- A JavaScript library for building User Interfaces.

Why React ?

PrepBytes

Angular vs React vs Vue

<u>ANGULAR</u>

- Framework
- Developed by Google
- Typescript
- Develop Native, Hybrid & Web apps
- MVC architecture

<u>REACT</u>

- Library
- Developed by Facebook
- JSX
- Develop SPA & Mobile Apps
- Virtual DOM

<u>VUE</u>

- Library
- Open-Source Project
- HTML & JavaScript
- Develop SPA & Native Apps
- Virtual DOM

PrepBytes

Features of React JS

- Based on component structure
- Uses JSX (Extension of JavaScript)
- Best used for SPA (Single Page Applications)
- Utilises both Virtual DOM and Real DOM

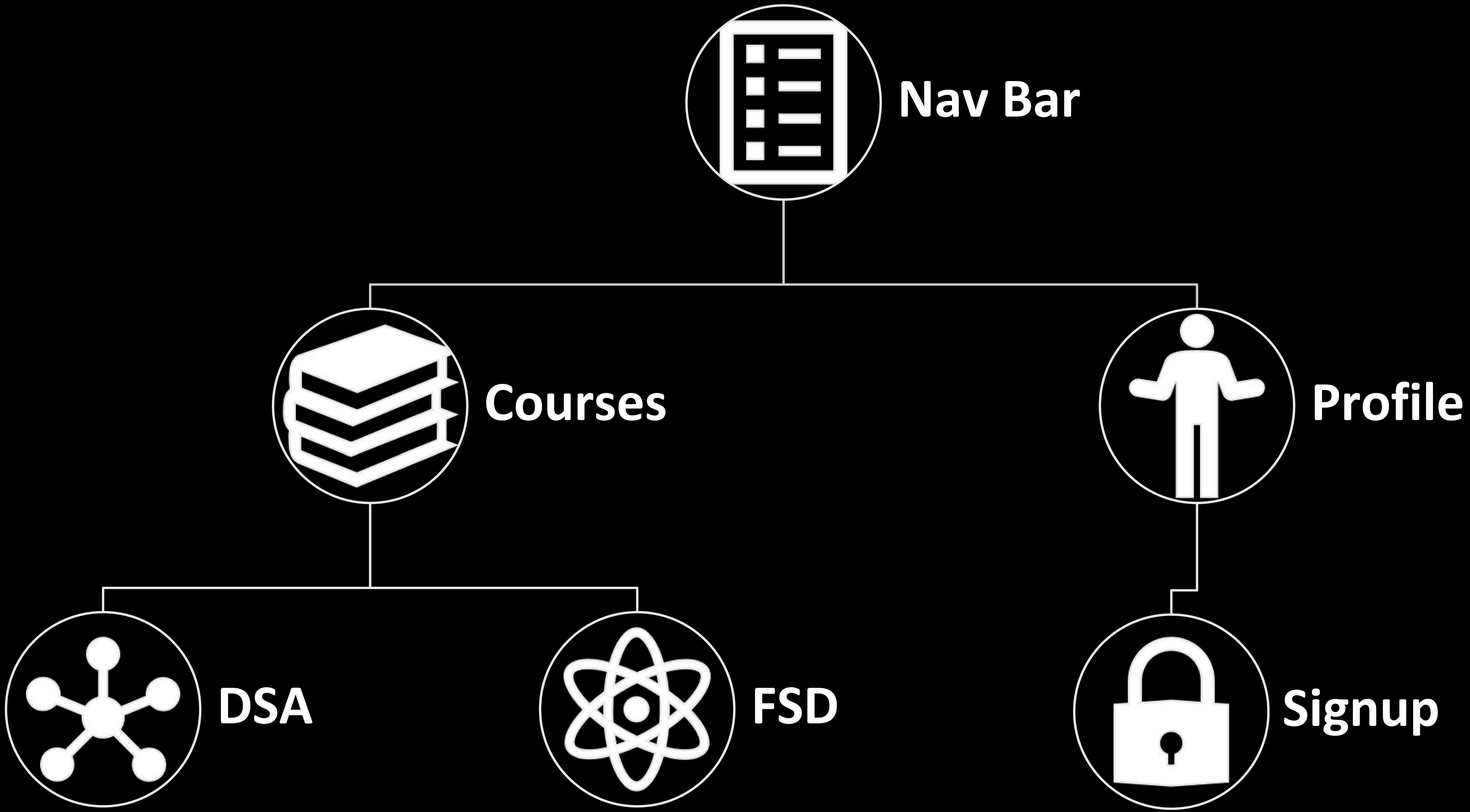
PrepBytes

Environment Setup

- **Step 1 :** Install Node.js
- **Step 2 :** npm install -g create-react-app
- **Step 3 :** npx create-react-app <app_name>

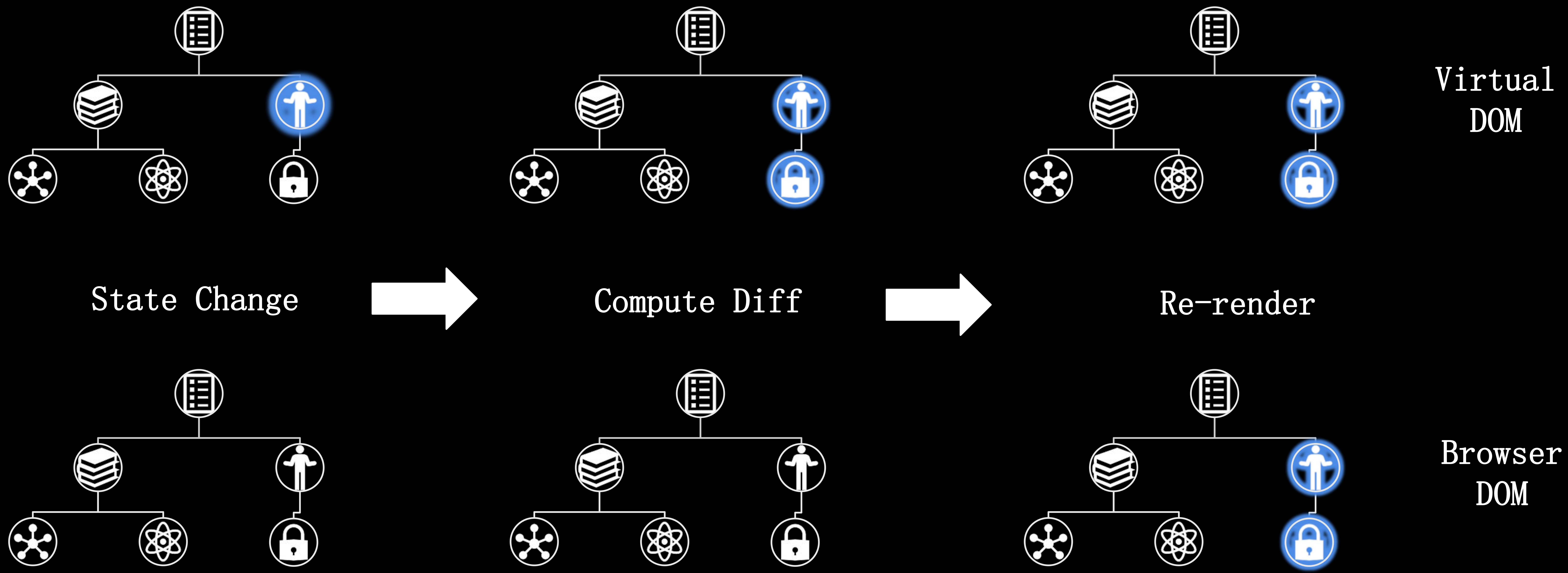
PrepBytes

Document Object Model



PrepBytes

Virtual DOM



JavaScript Executable (JSX)

- Syntax Extension to JavaScript.
- Produces React “elements”.

```
const element = <h1>Hello, world!</h1>;
```

PrepBytes

React without JSX

```
import React from 'react'
const element = React.createElement(
  'h1',
  {className : 'message'},
  'PrepBytes'
);
```

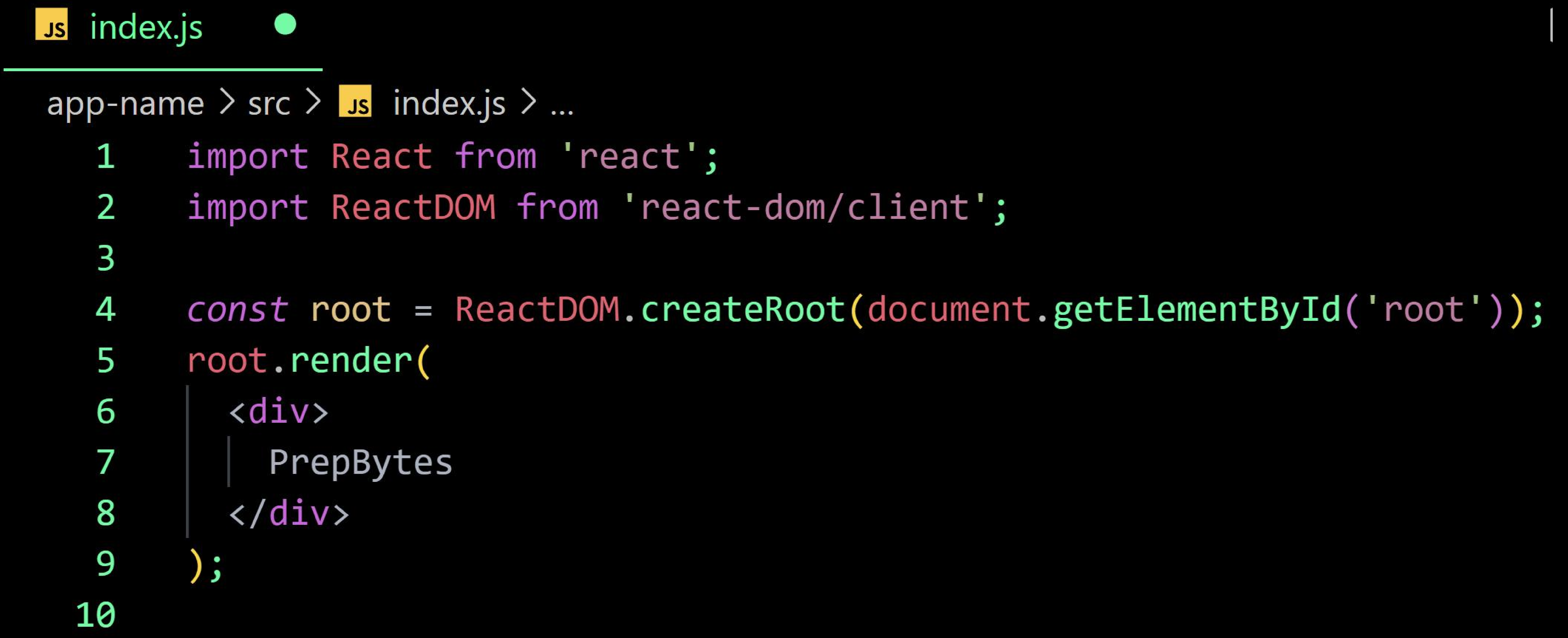
<u>createElement() definition</u>

```
const element = {
  type : 'h1',
  prop : {
    className : 'message',
    children : 'PrepBytes'
  }
}
```

<u>Representation of Object Created</u>

PrepBytes

React with JSX



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

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <div>
    PrepBytes
  </div>
);

```

JavaScript Executable (JSX)

- JSX can be used for
  - Embedding Expression

```
const myVariable = 'PrepBytes';
const element = <h1>Hello, {myVariable}</h1>;
```

  - Specifying Attributes

```
const element = <a href="https://www.prepbytes.com/"> link </a>;
```

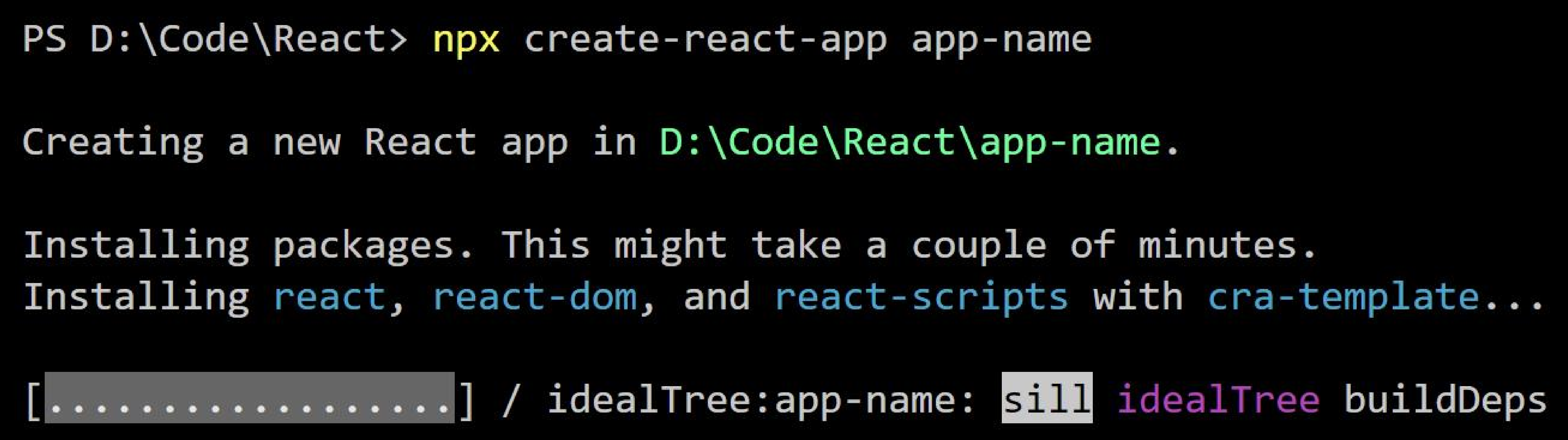
  - Represent Objects

```
React.createElement();
```

And many more tasks.....

create-react-app

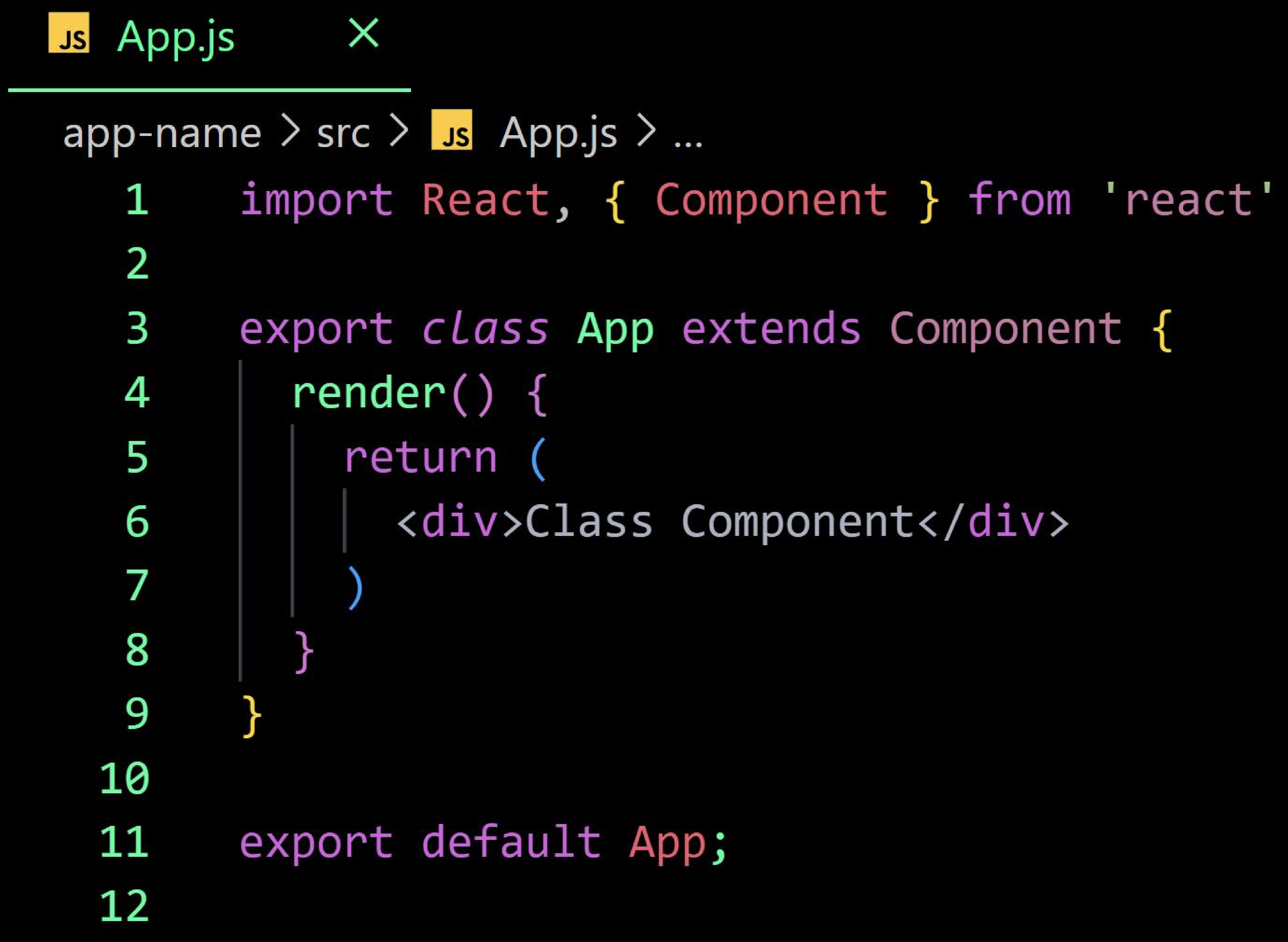
- Less to Learn
- Only one dependency
- No lock in



```
PS D:\Code\React> npx create-react-app app-name

Creating a new React app in D:\Code\React\app-name.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

[..................] / idealTree:app-name: sill idealTree buildDeps
```

Ways to Create Components

- **Class Components**



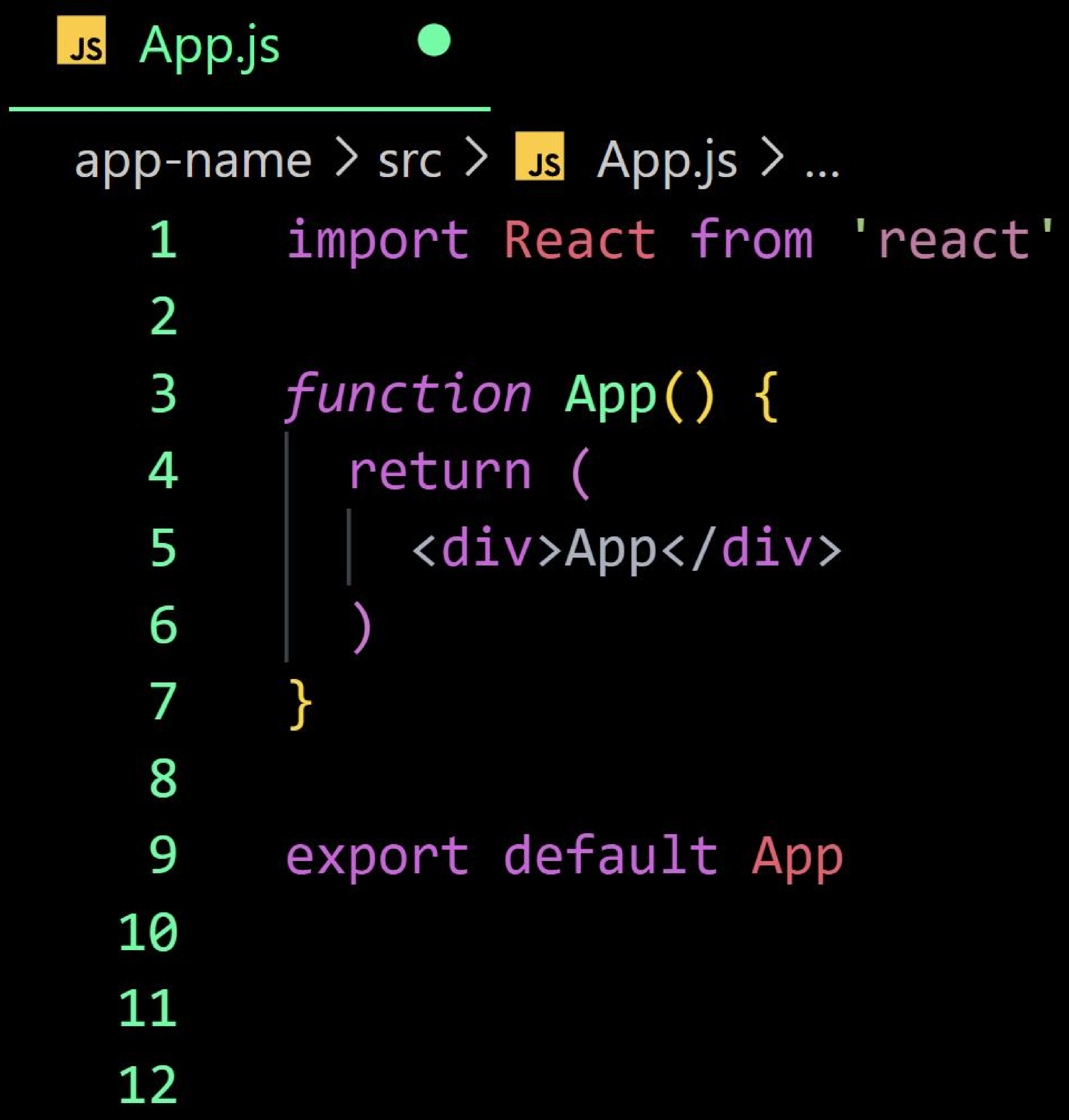
App.js

app-name > src > App.js > ...

```
import React, { Component } from 'react'

export class App extends Component {
  render() {
    return (
      <div>Class Component</div>
    )
  }
}

export default App;

```

Ways to Create Components

- **Function Components**



App.js

app-name > src > App.js > ...

```
import React from 'react'

function App() {
  return (
    <div>App</div>
  )
}

export default App
```

Functional vs Class Components

Functional Components	Class Components
Pure JavaScript function	Class that extends properties from React.Component
No render Method	render() method is mandatory
Stateless Components	Stateful Component
React Lifecycle methods cannot be used	React lifecycle methods can be used

Functional vs Class Components

Functional Components

- Hooks can be easily used
- No State Variables

Class Components

- Hooks can be used with different syntax
- Contains State Variables

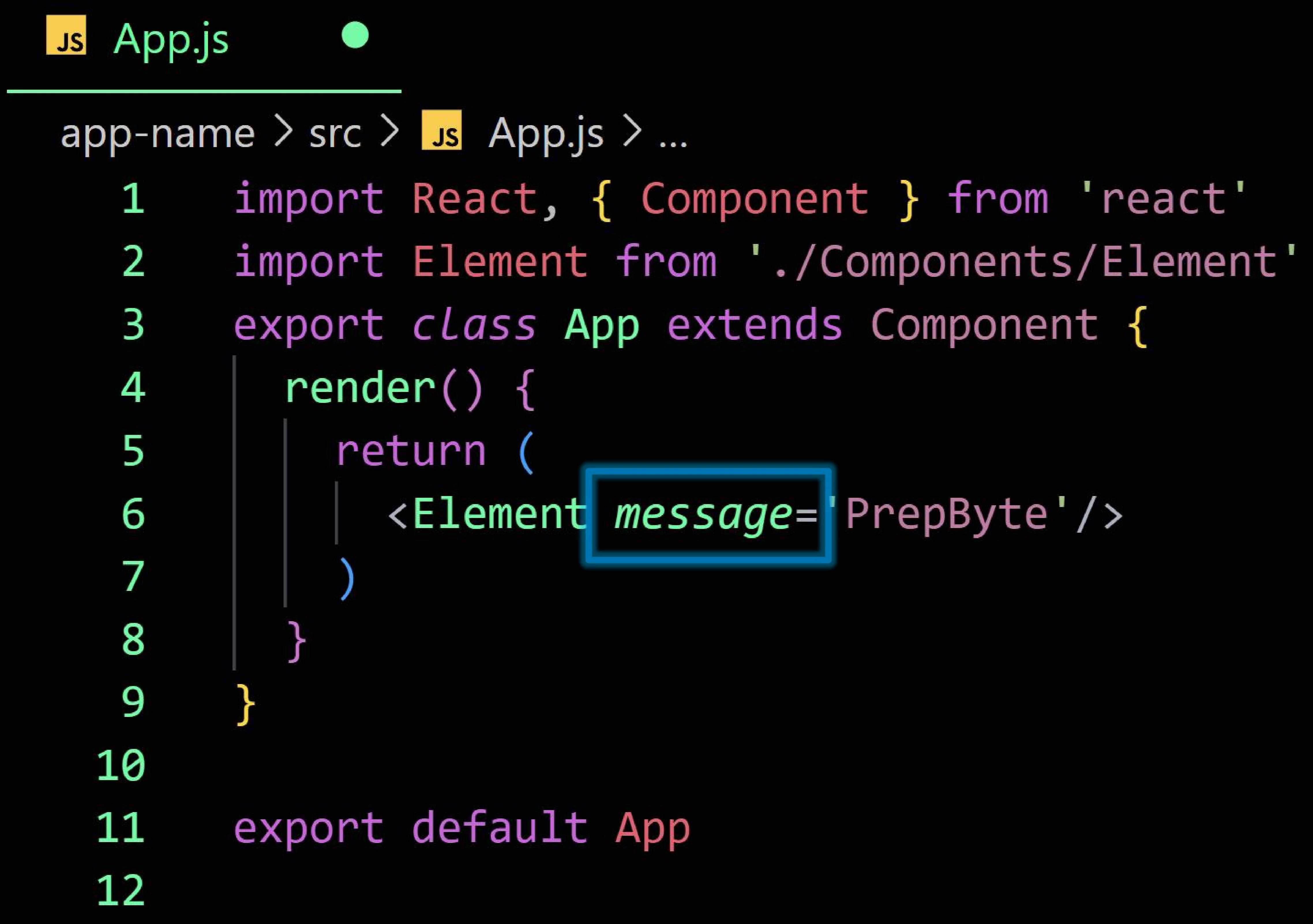
Props

- Arguments passed into React Components
- React is pretty flexible but it has a single strict rule:

  “All React components must act like pure functions with respect to their props.”

- **Pure Functions don’t attempt to change their inputs , i.e. they are immutable**

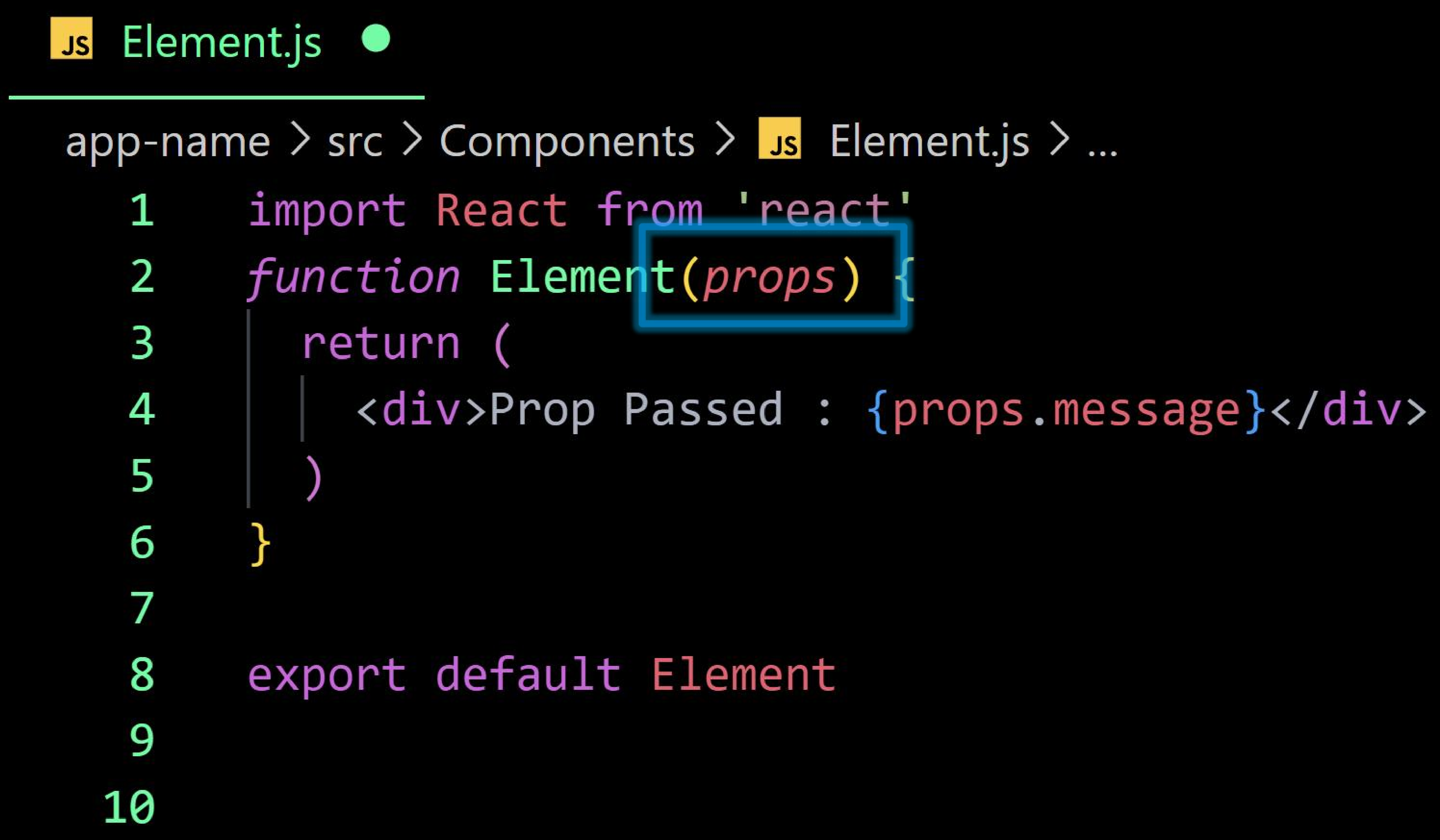
Props



App.js

app-name > src > App.js > ...

```
import React, { Component } from 'react'
import Element from './Components/Element'
export class App extends Component {
  render() {
    return (
      <Element message='PrepByte'/>
    )
  }
}

export default App

```



Element.js

app-name > src > Components > Element.js > ...

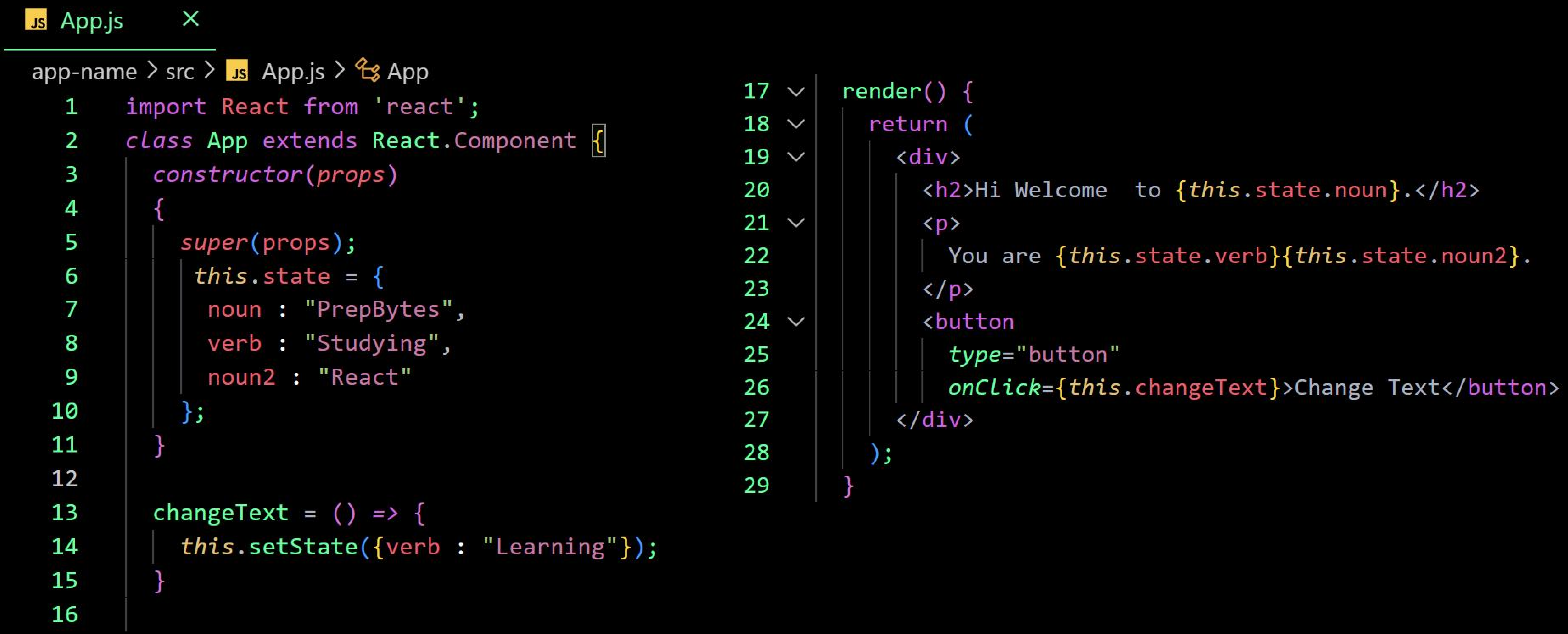
```
import React from 'react'
function Element(props) {
  return (
    <div>Prop Passed : {props.message}</div>
  )
}

export default Element

```

State

- Build-in React Object
- Used to contain data or information about component
- State can change over time
- On change of state , the component re-renders
- A state can be modified based on user action or network changes
- this.setState() is used to change the value of the state object

PrepBytes

setState()



App.js

app-name > src > App.js > App

```
import React from 'react';
class App extends React.Component {
  constructor(props)
  {
    super(props);
     this.state = {
      noun : "PrepBytes",
      verb : "Studying",
      noun2 : "React"
    };
  }

  changeText = () => {
    this.setState({verb : "Learning"});
  }

```

```
  render() {
    return (
      <div>
        <h2>Hi Welcome  to {this.state.noun}.</h2>
        <p>
          You are {this.state.verb}{this.state.noun2}.
        </p>
        <button
          type="button"
          onClick={this.changeText}>Change Text</button>
      </div>
    );
  }
```

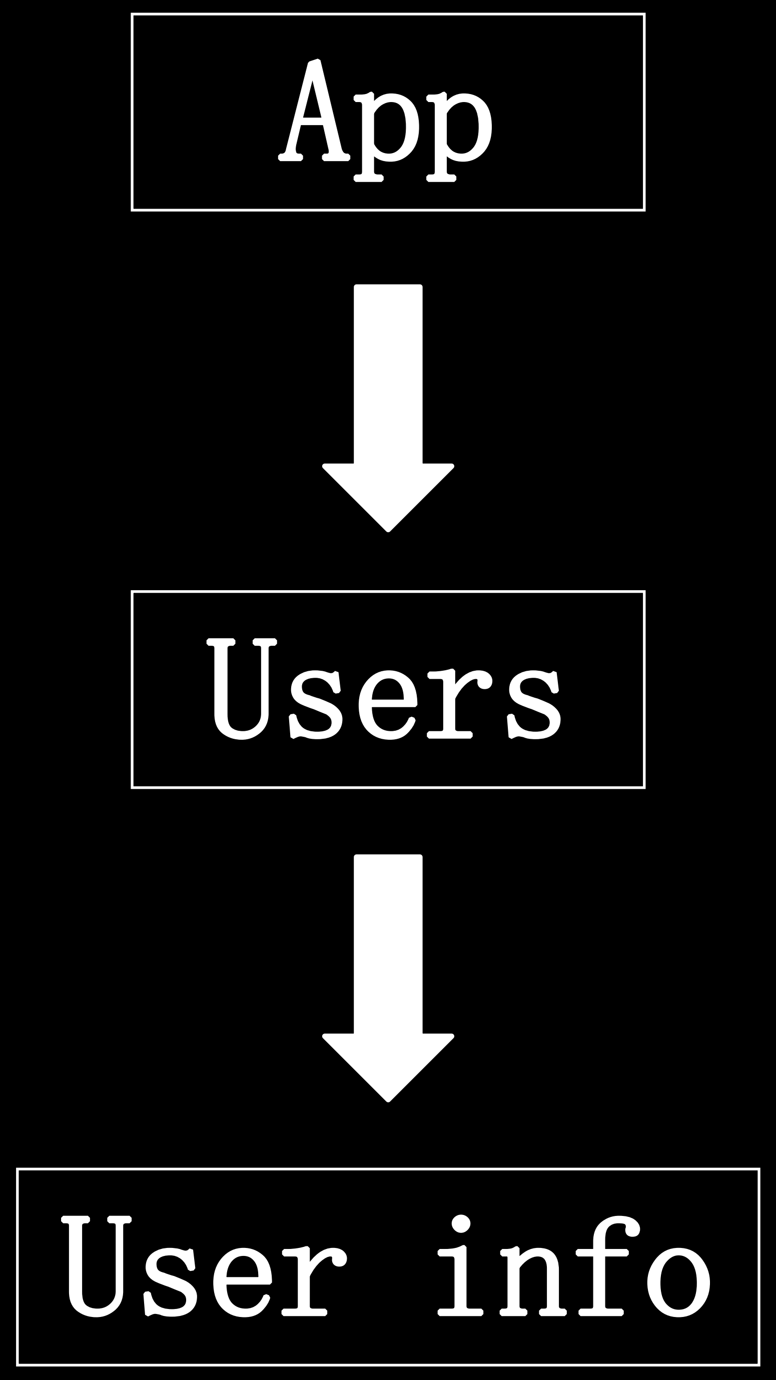
Props Vs State

	State	Props
Use Case	• State is used to store the data of the components that have to be rendered to the view	• Props are used to pass data and event handlers to the children components
Mutability	• State holds the data and can change over time	• Props are immutable—once set, props cannot be changed
Component	• State can only be used in class components	• Props can be used in both functional and class components
Updation	• Event handlers generally update state	• The parent component sets props for the children components

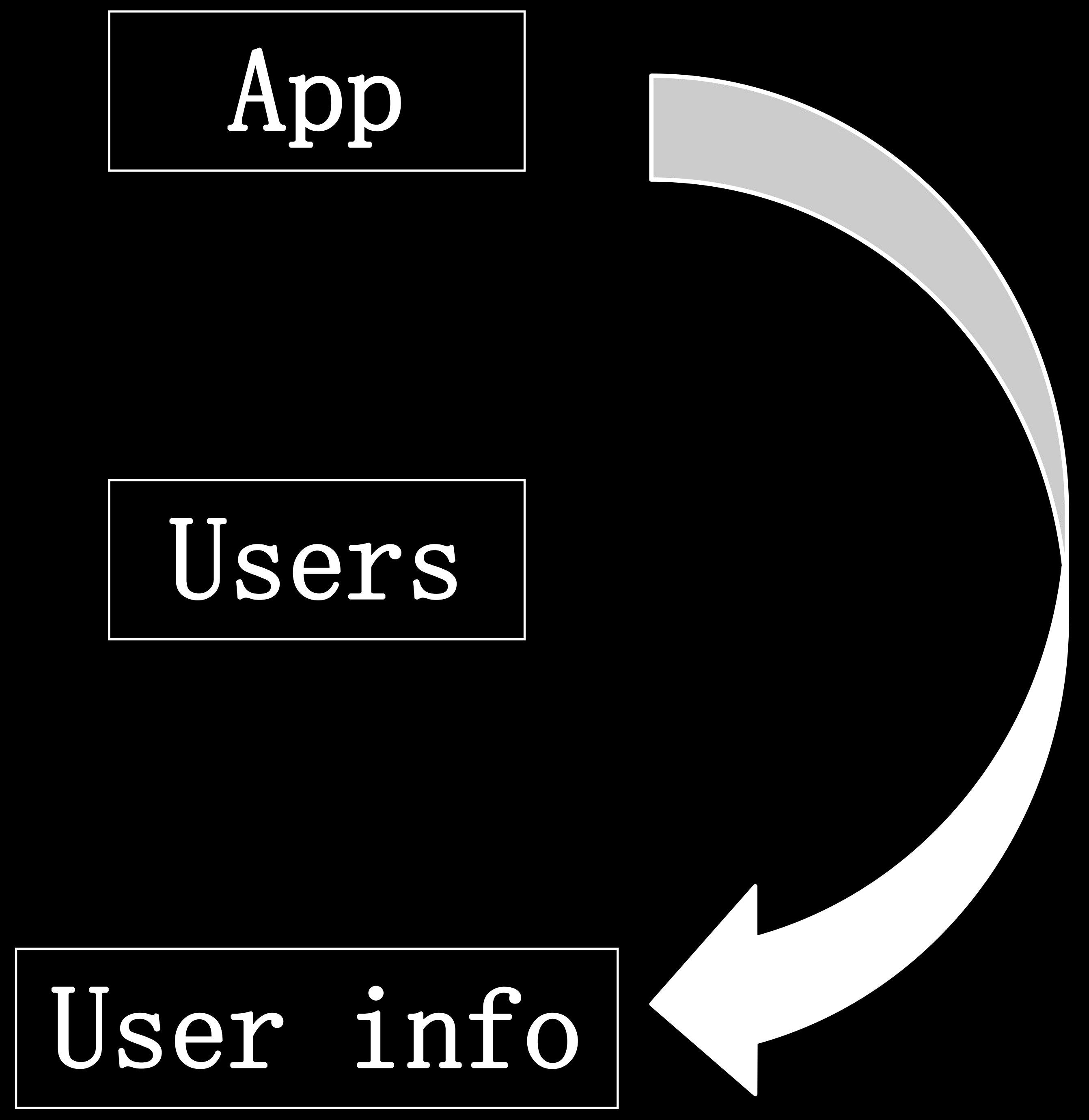
Event Listeners

- Functions that listen for some events happening and execute when that event happens.
- React events are usually written in camelCase
- React event handlers are written inside of curly braces.
- Arguments are passed to event handlers using an arrow function

React Context API



Without Context API

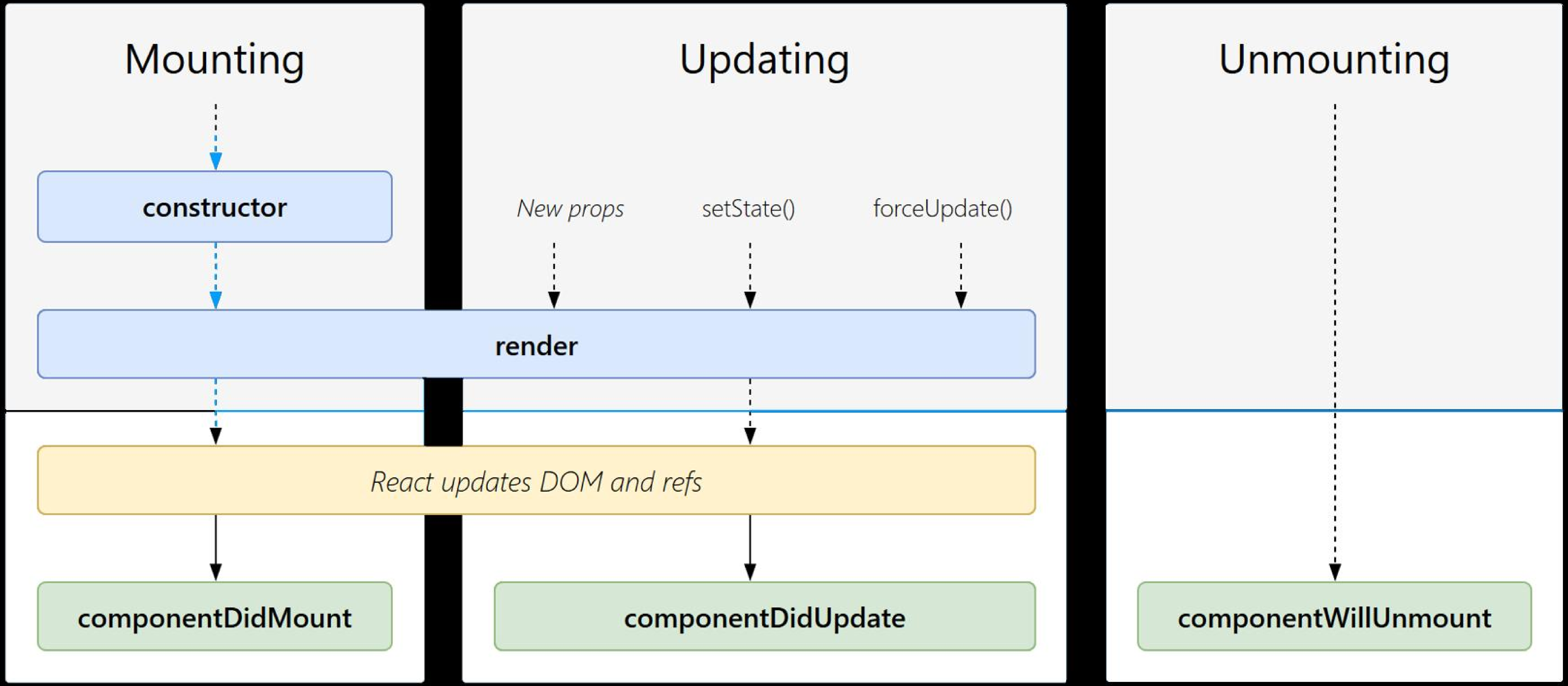


With Context API

React Context API

- React.createContext()
- Context.Provider
- Class.contextType
- Context.Consumer
- Context.displayName

React Life Cycle



Conditional Rendering

- Rendering the components in react can be conditional.
- There are many ways to achieve this as :
    - Logical && Operator
    - If-else
    - Ternary Operator
    - Switch-case

High Order Components

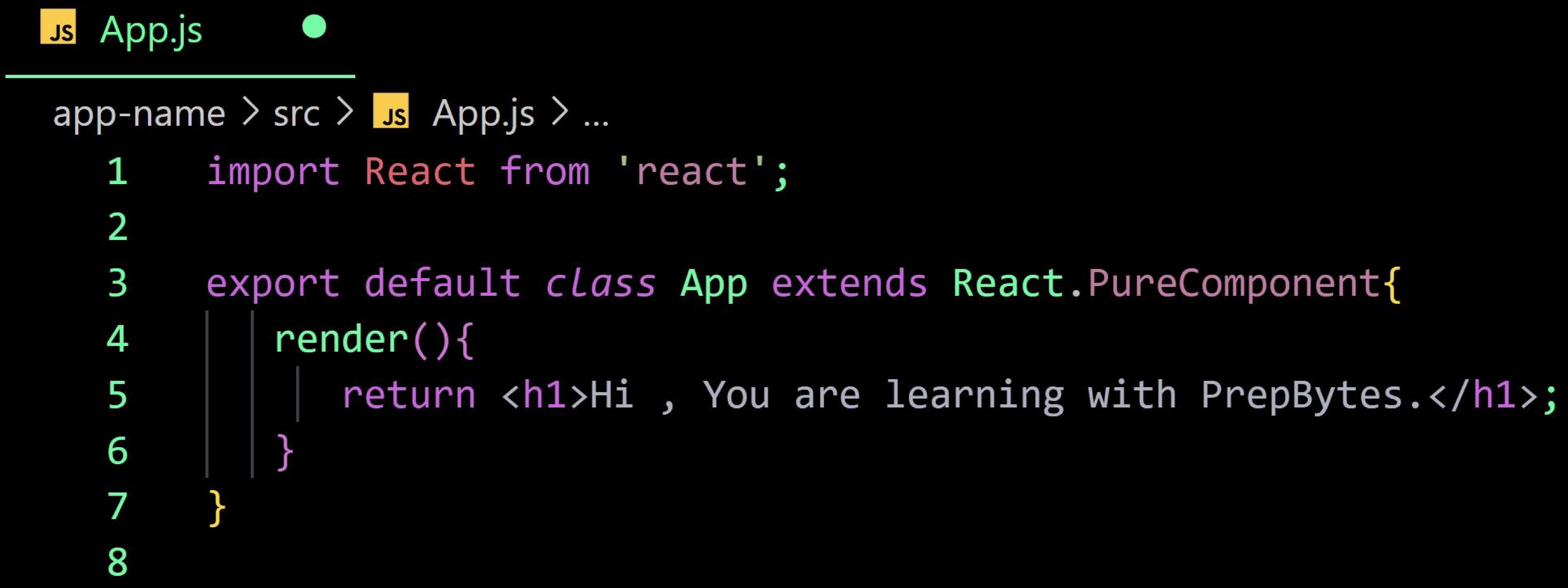
- HOC is a function which takes a Wrapped component as input argument and returns a new Enhanced component
- It should always be a pure function.
- It should never modify the Wrapped Component.

Pure Components

- Component is Pure if
  - Return value is only determined by it’s input values
  - It’s return value is always the same for the same input values
  - Class components that extend the React.PureComponent class are treated as pure components

PrepBytes

Pure Components


App.js ●

app-name > src > App.js > ...

```
import React from 'react';

export default class App extends React.PureComponent{
    render(){
        return <h1>Hi , You are learning with PrepBytes.</h1>;
    }
}

```

Pure Components

- Pure Components prevents components being re-rendered if the values of state and props has not changed.
- These components will be rendered only in 3 conditions:
  - this.setState({ })
  - Change in props
  - this.forceUpdate( )
- Exception : shouldComponentUpdate

React Forms

- Controlled Components
- Uncontrolled Components

Form Elements

- The input Tag
- The textarea Tag
- The select Tag
- The file input Tag

Handling Multiple Inputs

- **Multiple inputs are handled by using *name* Attribute**
- **To access the fields in the event handler use the *event.target.name* and *event.target.value* syntax.**

```
const handleChange = (event) => {
  const name = event.target.name;
  const value = event.target.value;
  setInputs(values => ({...values, [name]: value}))
}
```

React Router

Multi page Application

Single page Application

React Router

PrepBytes

React Router

react-router-dom

react-router-native

React Router

Types of Routers

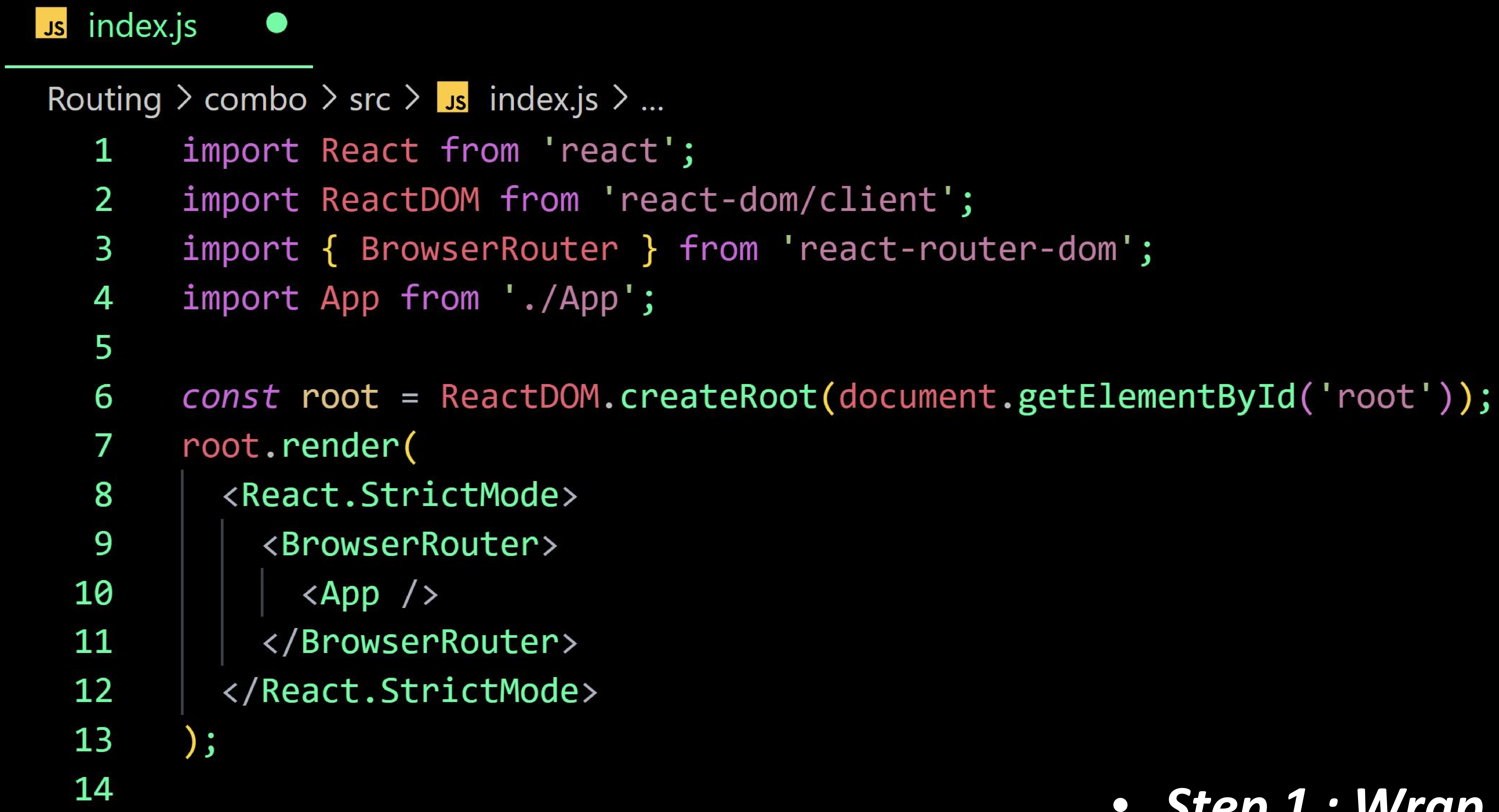
- BrowserRouter
- HashRouter
- MemoryRouter
- NativeRouter
- StaticRouter

React Router

Data Routers

- createBrowserRouter
- createMemoryRouter
- createHashRouter

Implementing Routing



index.js

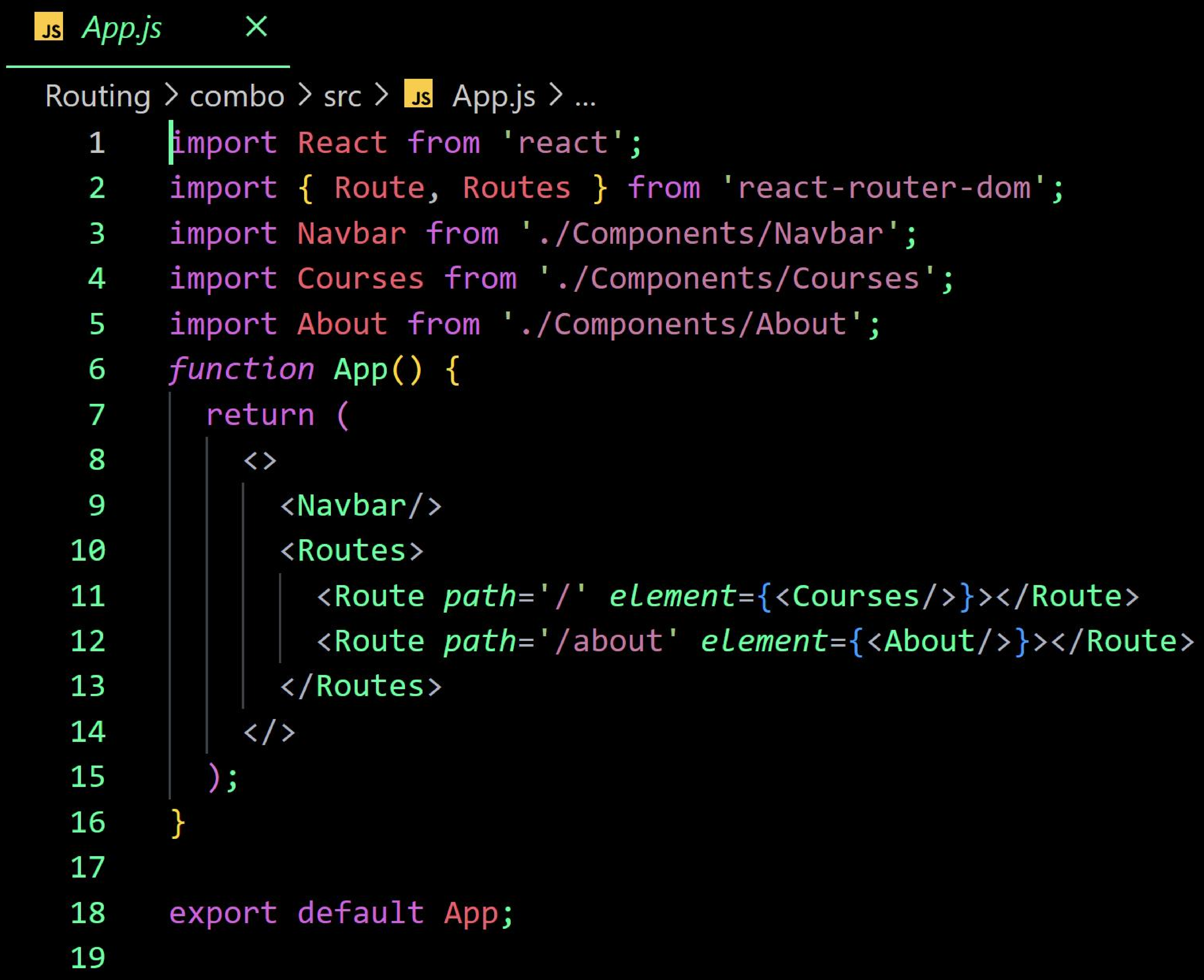
Routing > combo > src > index.js > ...

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter } from 'react-router-dom';
import App from './App';

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

```

- ***Step 1 : Wrap Component with a Router***

Implementing Routing

PrepBytes



App.js ×

Routing > combo > src > App.js > ...

```
import React from 'react';
import { Route, Routes } from 'react-router-dom';
import Navbar from './Components/Navbar';
import Courses from './Components/Courses';
import About from './Components/About';
function App() {
  return (
    <>
      <Navbar/>
      <Routes>
        <Route path='/' element={<Courses/>}></Route>
        <Route path='/about' element={<About/>}></Route>
      </Routes>
    </>
  );
}

export default App;

```

- ***Step 2 : Set-up Routes and Route.***

React Router

Components

- Link
- NavLink
- Navigate
- Outlet

React Router

More on Routing

- Dynamic Segment
- Splat / MatchAll
- Routing Priority
- Nesting Routes

React Router

Hooks

- useRoutes
- useParams
- useHistory
- useLocation
- useOutlet
- useOutletContext

React Hooks

- Allows use of state and other features in Functional Components
- Hooks are functions that let you “hook into” React state and lifecycle features from function components
- Types of Hooks

  - State Hook

  - Effect Hook

  - Context Hook

State Hook

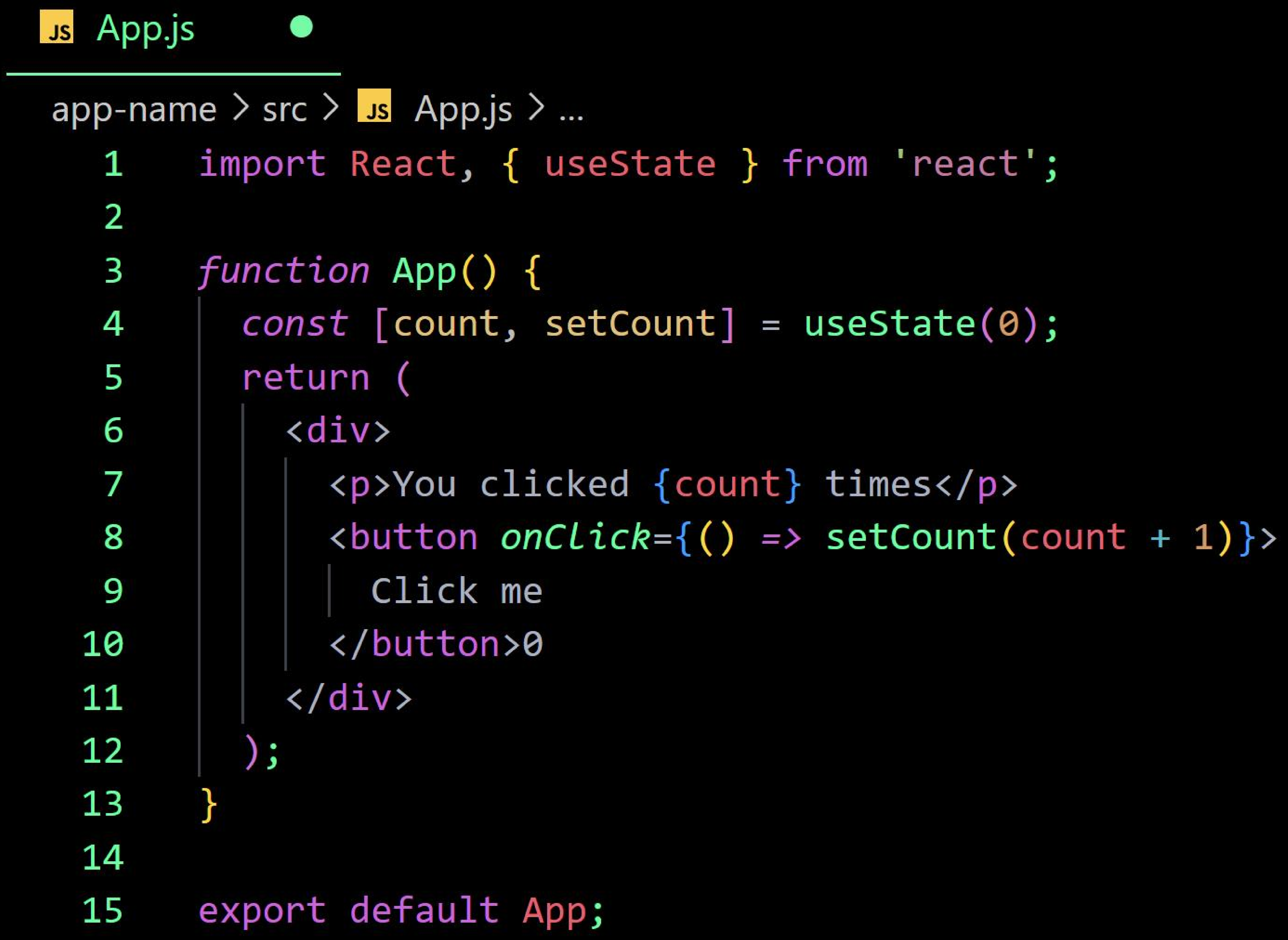
useState()

↳ [currStateValue,updatefunction]

State Hook

Problem Statement : Create A Counter App using useState()

useState() Hook



App.js ●

app-name > src > App.js > ...

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

function App() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>0
    </div>
  );
}

export default App;
```

Effect Hook

- Adds the ability to perform ***side effects*** from a function component
- These Side effects can be

  - Data Fetching

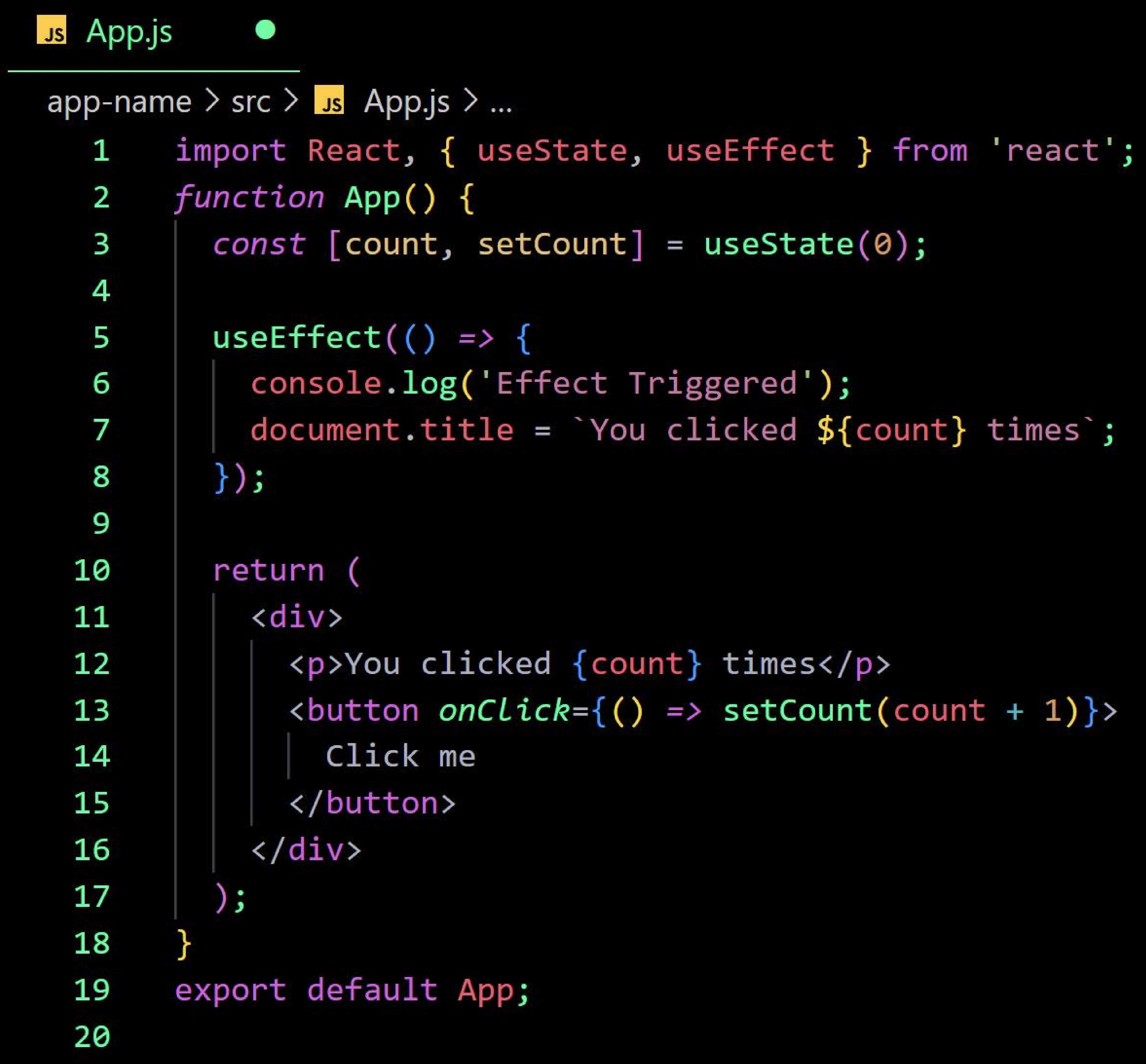
  - Subscriptions

  - Manually Changing DOM , etc.
- Serves the same purpose as componentDidMount, componentDidUpdate and componentWillUnMount.

Effect Hook

Problem Statement : Update Document title using useEffect() on the Counter App.

Effect Hook



App.js

app-name > src > App.js > ...

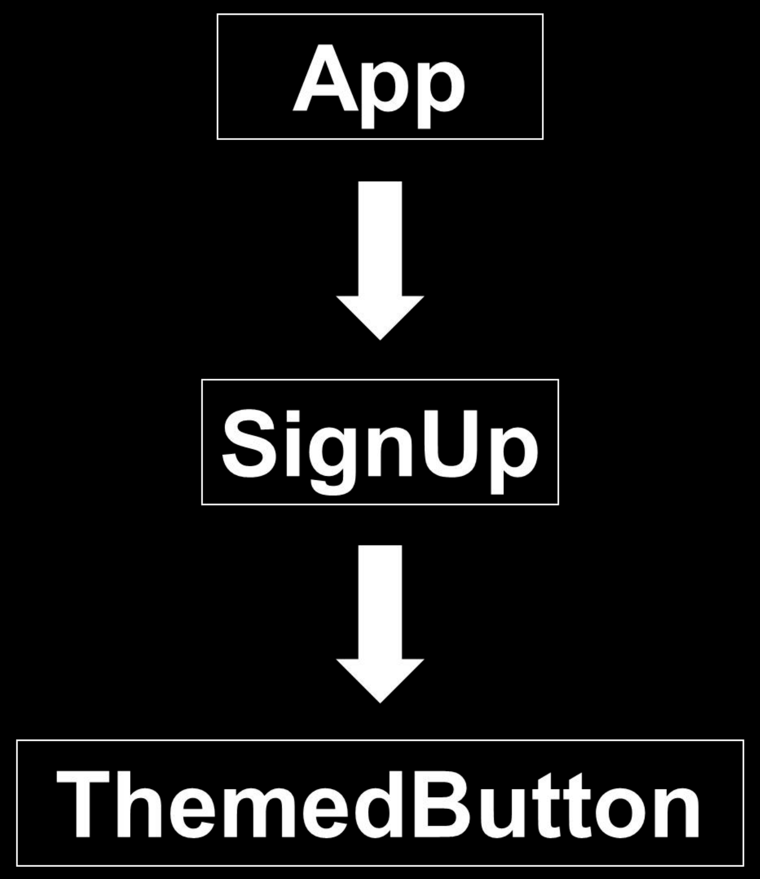
```jsx
import React, { useState, useEffect } from 'react';
function App() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log('Effect Triggered');
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
export default App;

```

Context Hook

- Accepts a context object and returns the current context value for that context
- When the nearest <MyContext.Provider> above the component updates, this Hook will trigger a re-render with the latest context value passed to that MyContext provider.
- A component calling useContext will always re-render when the context value changes.

Context Hook

Problem Statement : Change Theme of a Button element called ThemedButton on change of context value. The button is a Child component of a component called SignUp and SignUp is a Child component of App Component.



Redux

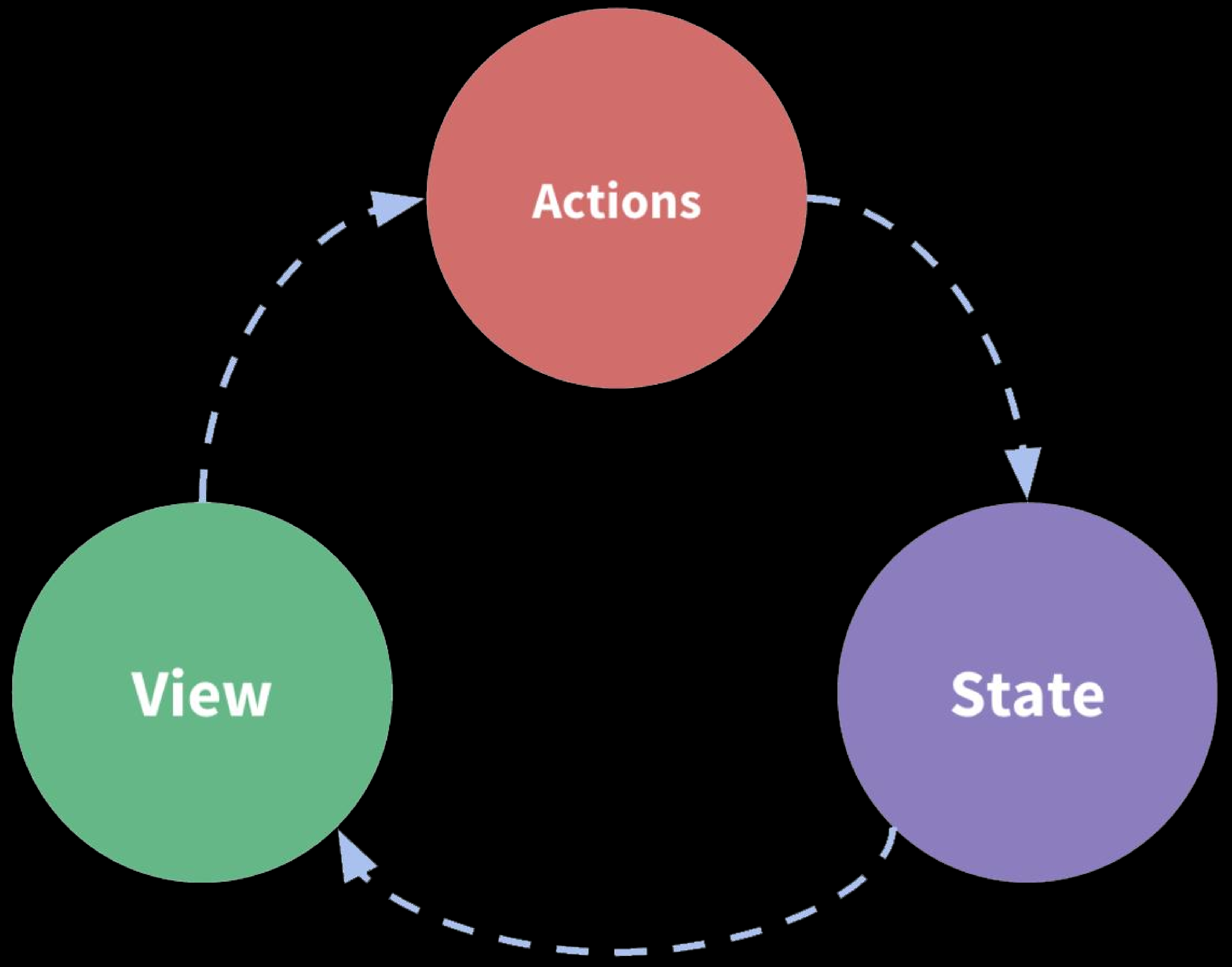
- Pattern/Library for managing and updating application state, using events called ‘***actions***’ .
- Helps manage ‘**GLOBAL**’ state

PrepBytes

When to use Redux

- You have large amounts of application state that are needed in many places in the app
- The app state is updated frequently over time
- The logic to update that state may be complex
- The app has a medium or large-sized codebase, and might be worked on by many people

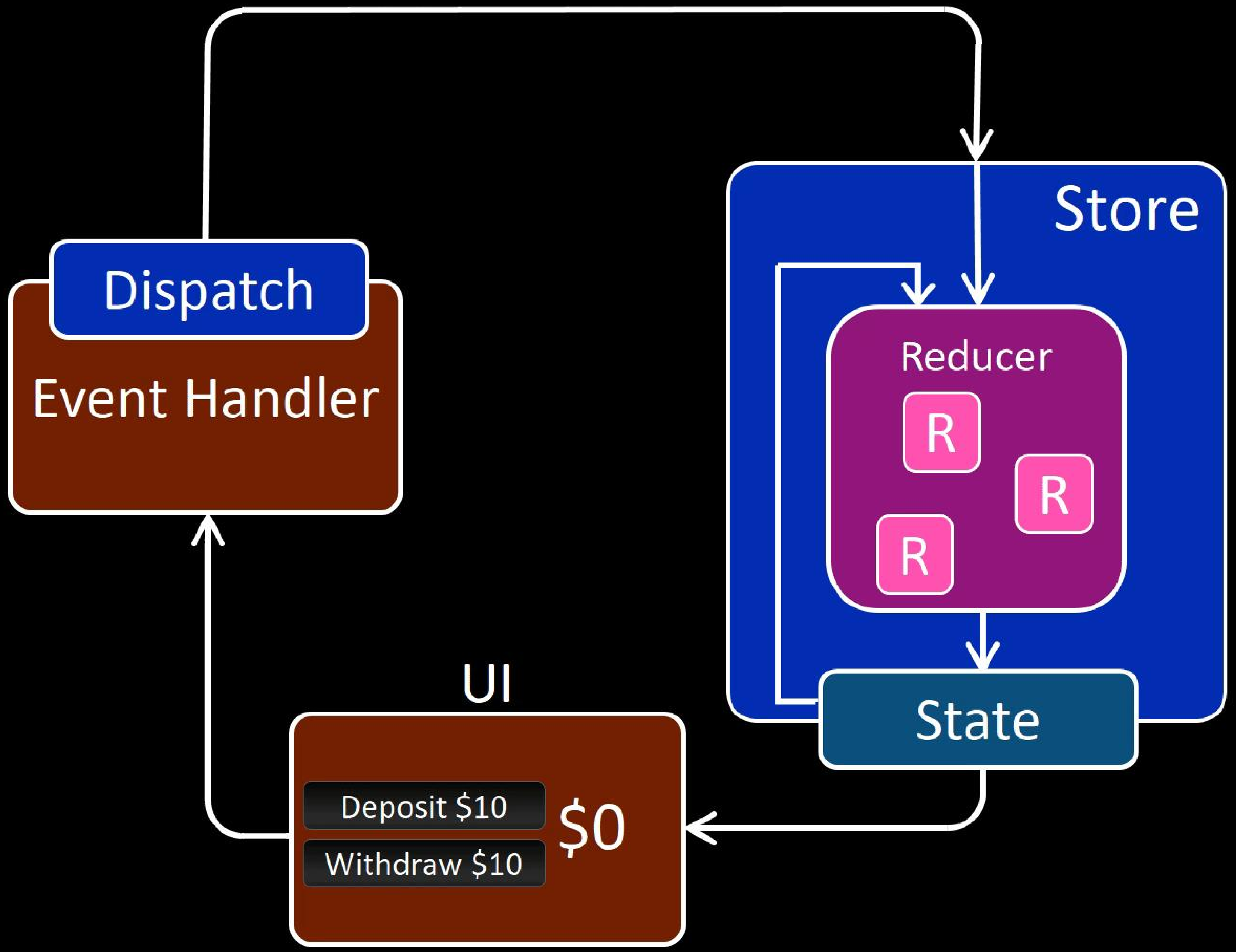
State Management



Redux Terminologies

- Action
- Action Creators
- Reducers
- Store
- Dispatch
- Selectors

Redux Application Data Flow



Redux Application Data Flow

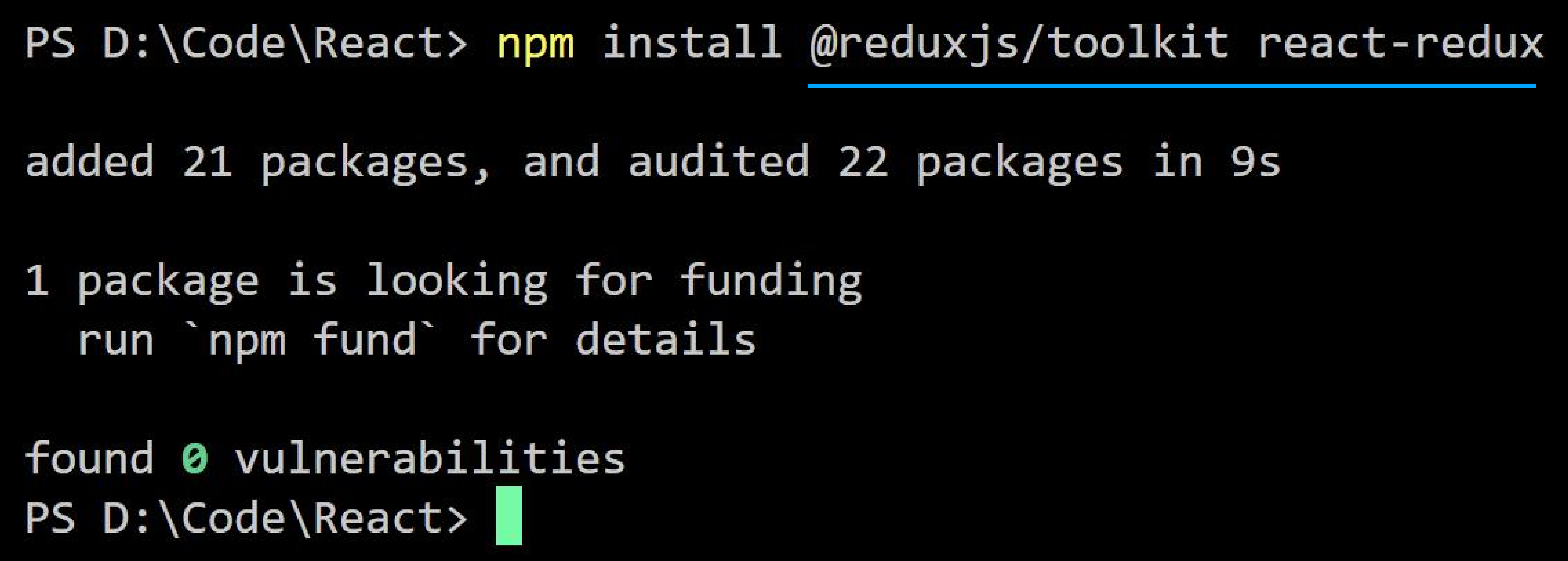
- State describes the condition of the app at a point in time, and UI renders based on that state
- When something happens in the app:
  - The UI dispatches an action
  - The store runs the reducers, and the state is updated based on what occurred
  - The store notifies the UI that the state has changed
- The UI re-renders based on the new state

Redux

Problem Statement : Create a Custom Increment App using Redux .

Redux Toolkit

- Redux Toolkit is a set of tools that helps simplify Redux development.
- Includes utilities for creating and managing Redux stores, as well as for writing Redux actions and reducers.
- Initialization



```
PS D:\Code\React> npm install @reduxjs/toolkit react-redux

added 21 packages, and audited 22 packages in 9s

1 package is looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\Code\React>
```

Redux Toolkit

- Create a Redux store with configureStore

  - configureStore accepts a reducer function as a named argument

  - configureStore automatically sets up the store with good default settings

- Provide the Redux store to the React application components

  - Put a React-Redux <Provider> component around your <App />

  - Pass the Redux store as <Provider store={store}>

Redux Toolkit

- Create a Redux "slice" reducer with createSlice

  - Call createSlice with a string name, an initial state, and named reducer functions

  - Reducer functions may "mutate" the state using Immer

  - Export the generated slice reducer and action creators

- Use the React-Redux useSelector/useDispatch hooks in React components

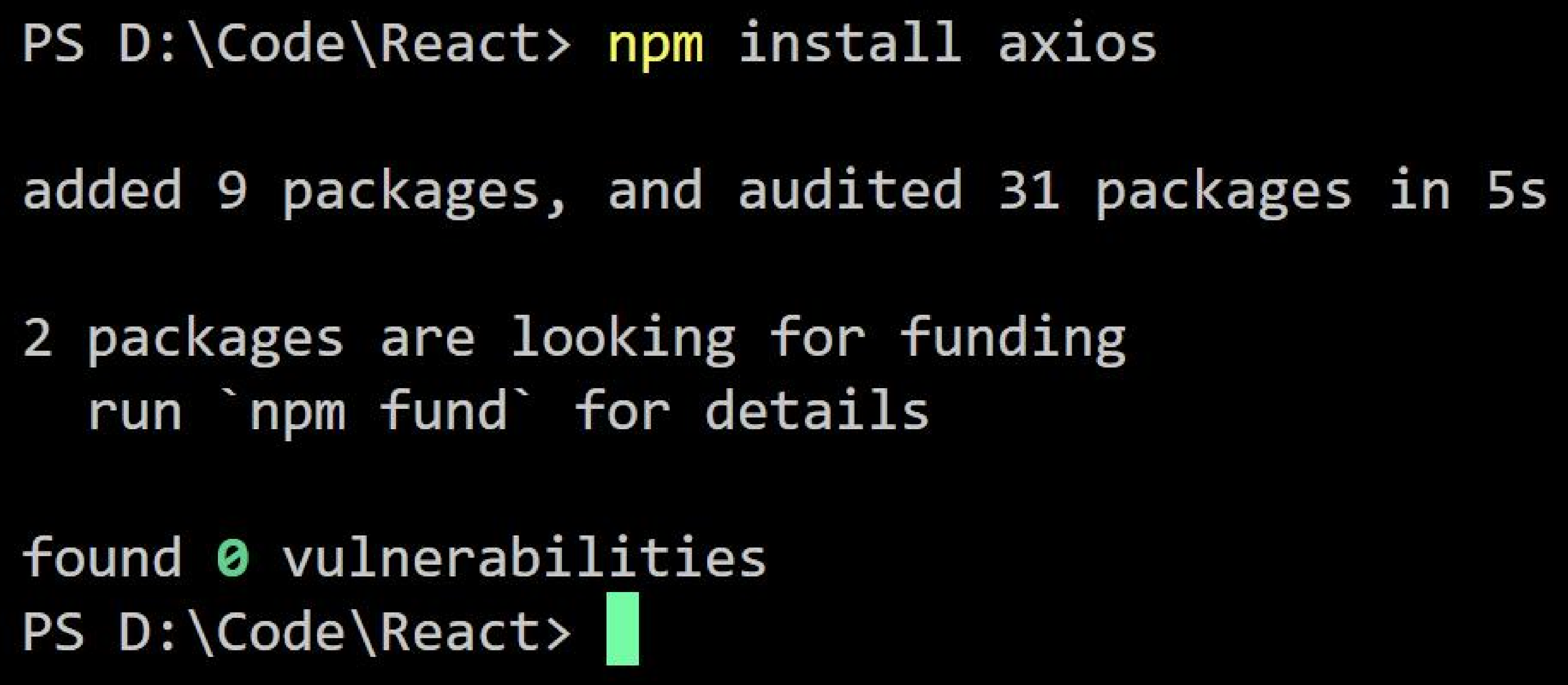
  - Read data from the store with the useSelector hook

  - Get the dispatch function with the useDispatch hook, and dispatch actions as neede

Axios

- HTTP Client Library
- Promise Based
- Allows to make requests to a given endpoint.

- Installation :



```
PS D:\Code\React> npm install axios

added 9 packages, and audited 31 packages in 5s

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

found 0 vulnerabilities
PS D:\Code\React> 
```

Sending HTTP Request

- axios() Function
- Configuration Options

- **method**: The HTTP method through which the request should be sent in

- **url**: The server's URL to which the request must be sent to

- **data**: The data specified with this option is sent in the body of the HTTP request in Axios POST requests, PUT, and PATCH.

```
axios({
  method: "post",
  url: "/user_login",
  data:{
    username: "PrepBytes",
    firstname: "Prep",
    lastname: "Bytes"
  }
});
```

Axios Request Methods

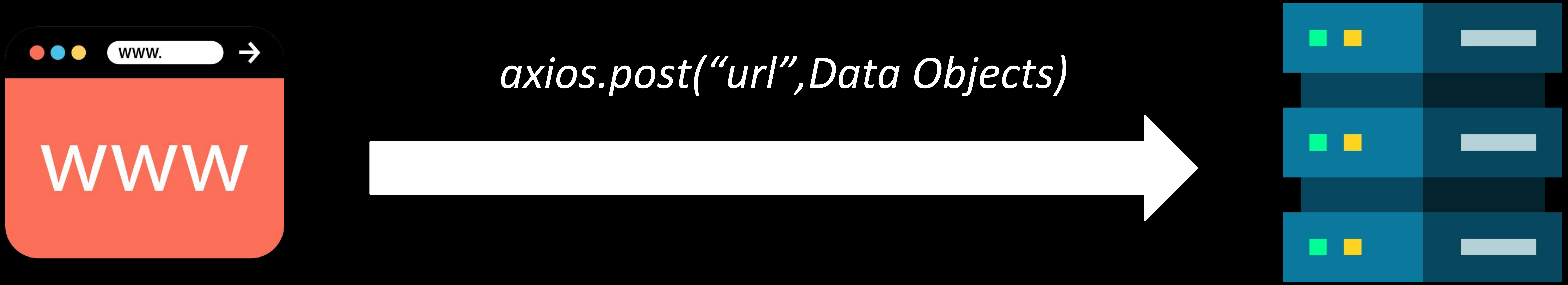
- axios.request(config)
- axios.get(url[, config])
- axios.delete(url[, config])
- axios.head(url[, config])
- axios.options(url[, config])
- axios.post(url[, data[, config]])
- axios.put(url[, data[, config]])
- axios.patch(url[, data[, config]])

Axios Response Objects

- data - the payload returned from the server
- status - the HTTP code returned from the server
- statusText - the HTTP status message returned by the server
- headers - headers sent by the server
- config - the original request configuration
- request - the request object

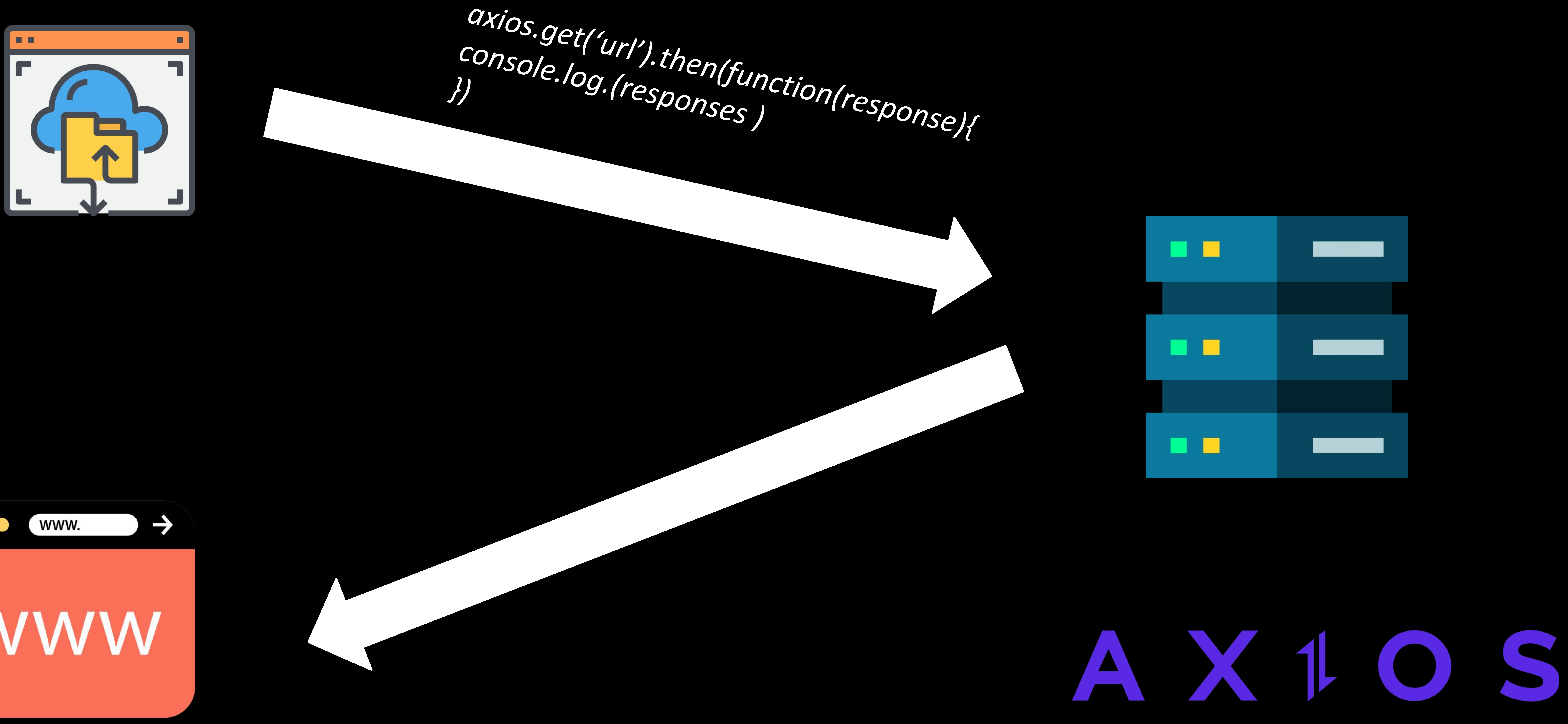
PrepBytes

Axios Post Request



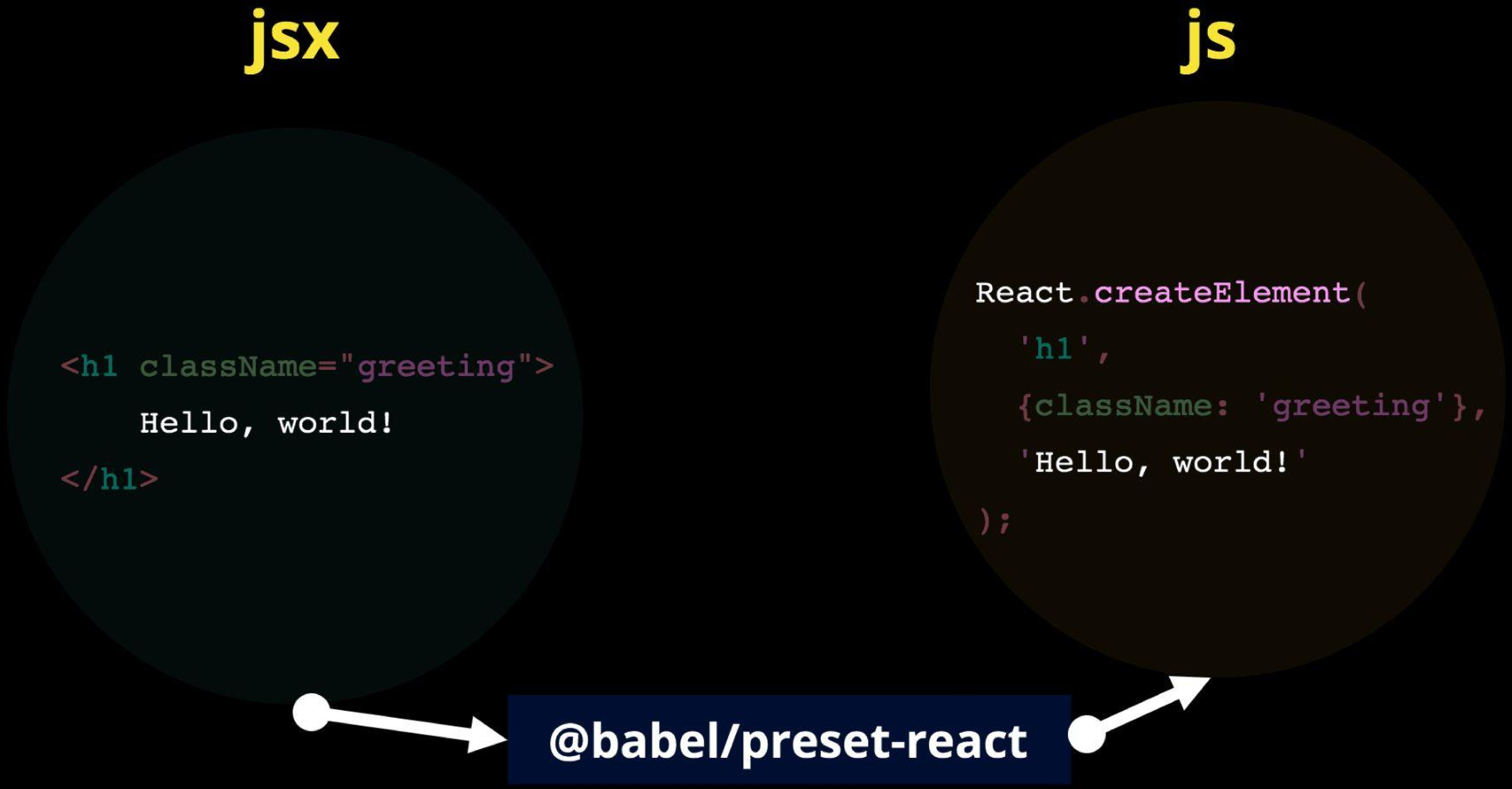
AXIOS

Axios Get Request



```
axios.get('url').then(function(response){
console.log.(responses )
})
```

Babel



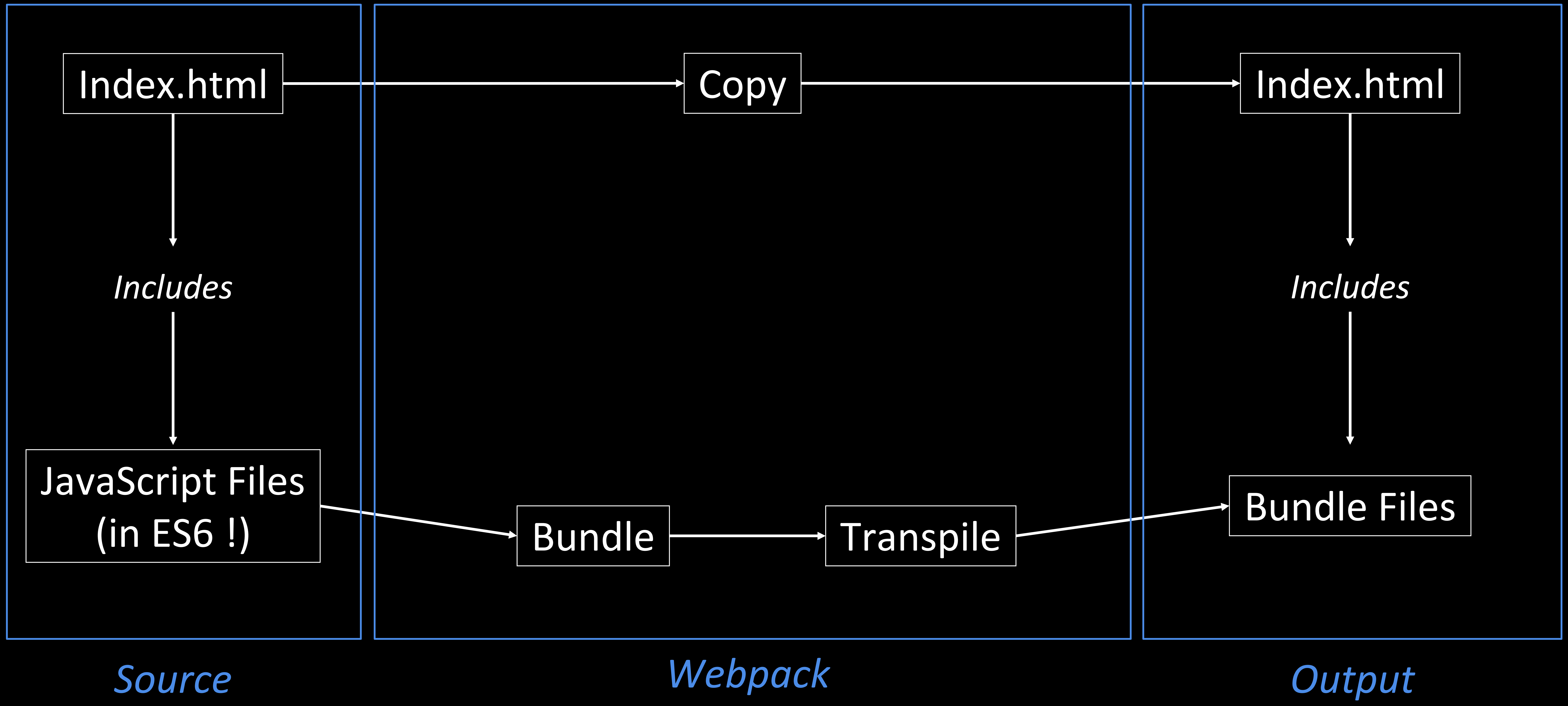
```
<h1 className="greeting">
    Hello, world!
</h1>
```

```
React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

Babel

- @babel/core
- @babel/preset-react
- @babel/preset-env
- Babel-loader

Webpack


Index.html
Includes
JavaScript Files (in ES6 !)
Copy
Bundle
Transpile
Index.html
Includes
Bundle Files
Source
Webpack
Output

Webpack

- webpack
- webpack-dev-server
- html-webpack-plugin