

# OneHead Programming Series

Simplified React Programming

(A step-by-step guide to mastering Reactjs)

By

Adeleke Bright

MAY, 2021

## Table of Contents

### Section One – Understanding React Ecosystem

Chapter	Page
1.0 Understanding React	
2.0 Hello, World	
3.0 JSX	
4.0 Components	
5.0 State and Lifecycle Methods	
6.0 Event Handling	
7.0 Lists	
8.0 Forms	
9.0 Lifting state up	
10.0 React Philosophy	
11.0 Demos	
a) Todo List	
b) Calculator	
c) Tic-Tac Toe	
d) Snake Game	
e) Brick Wall	

### Section Two – Advanced Concepts

12.0 React Hooks	
13.0	

### Section Three – React Router

18. Understanding Routing	
19. React Routing Components	
20. A complete Example	

### Section Four – React and API's

21. API communication	
-----------------------	--

### Section Five – Plugin and Components Creation

22. Creating a custom dropdown component	
22. Creating a custom carousel component	
22. Creating a custom animation component	

23. Creating a custom mobile navigation

Section Six – Authentication and Authorization

Section seven – State Management and other concepts

# Understanding React

What is React?

React is a JavaScript library for building user interface.

A user interface is the part of an application that the user interacts with.

React makes the following easier:

## a) **Rendering**

You can build your application either as a standalone or as a mix of other parts known as components.

When we build user interfaces using HTML, we run into code duplication and unnecessary redundancy.

See the code example below:

```
<h2>List of fruits</h2>
```

```
<ul>
```

```
  <li>Mango</li>
```

```
  <li>Orange</li>
```

```
  <li>Cashew</li>
```

```
  <li>Pawpaw</li>
```

```
  <li>Guava</li>
```

```
</ul>
```

With react we can leverage the programming power of JS to create components as modules and share them across our project.

The code above can be simplified with react as:

```
const fruits = ["mango", "Orange", "Cashew", "Pawpaw", "Guava"]
```

```
const fruitList = ({ fruits }) => {
```

```
  const items = fruits.map ((fruit, index) => <li key={index}>{ fruit}</li>
```

```
  }
```

```
  return (
```

```
    <>
```

```
<h2>List of fruits</h2>
```

```
  <ul>
```

```
{items}
```

```
</ul>
```

```
</>
```

## b) **Interaction Management**

React makes it easier to handle what happens when users interact with your application, or when a state changes.

You can bind actions to a part of your app, localize that action and handle them efficiently.

## c) **Write once, run anywhere**

Your knowledge of react will enable you to build UI for any platform ranging from mobile to desktop.

You can use React Native to build cross platform mobile App.

React is designed, published, and maintained by Facebook. You can go ahead to join in using it for your next project.

The guide in this book will help you in your next project.

# HELLO, WORLD

Before we dive into our first hello world project using react, let us get some basic things setup first.

We need to install react and setup a local development environment

## Installation:

We will be using react with create-react-app, a toolset that allows for easy react tooling with little to no configuration.

Let us start by creating a project directory that we will be storing all our react codes within. Move into your terminal and to any location of choice and run the command below

```
$ mkdir bigjara
```

```
$ cd Bigjara-react
```

To install create react app, use this command below:

```
$ npm install create-react-app
```

```
$ npx create-react-app bigjara-react
```

After installing the toolset, and creating your app; a directory (Bigjara-react) will be created inside bigjara-react.

This is going to be our first project.

Within this directory, you will have this file structure as shown below:

Name	Date modified	Type	Size
.git	05/11/2020 07:45	File folder	
node_modules	05/11/2020 09:19	File folder	
public	05/11/2020 07:45	File folder	
src	05/11/2020 07:45	File folder	
.env	12/05/2021 04:01	ENV File	1 KB
.gitignore	05/11/2020 07:26	Git Ignore Source ...	1 KB
Outline	07/11/2020 09:20	File	4 KB
package	12/05/2021 04:01	JSON Source File	1 KB
package-lock	05/11/2020 08:21	JSON Source File	631 KB
README	05/11/2020 07:45	Markdown Source...	4 KB
yarn.lock	05/11/2020 07:44	LOCK File	511 KB
yarn-error	05/11/2020 07:46	Text Document	537 KB

```
> public
> src
> npm_modules
> package.json
> package_lock.json
>
```

Most of our code will reside inside src directory.

Some common commands for react :

```
$ npm start
```

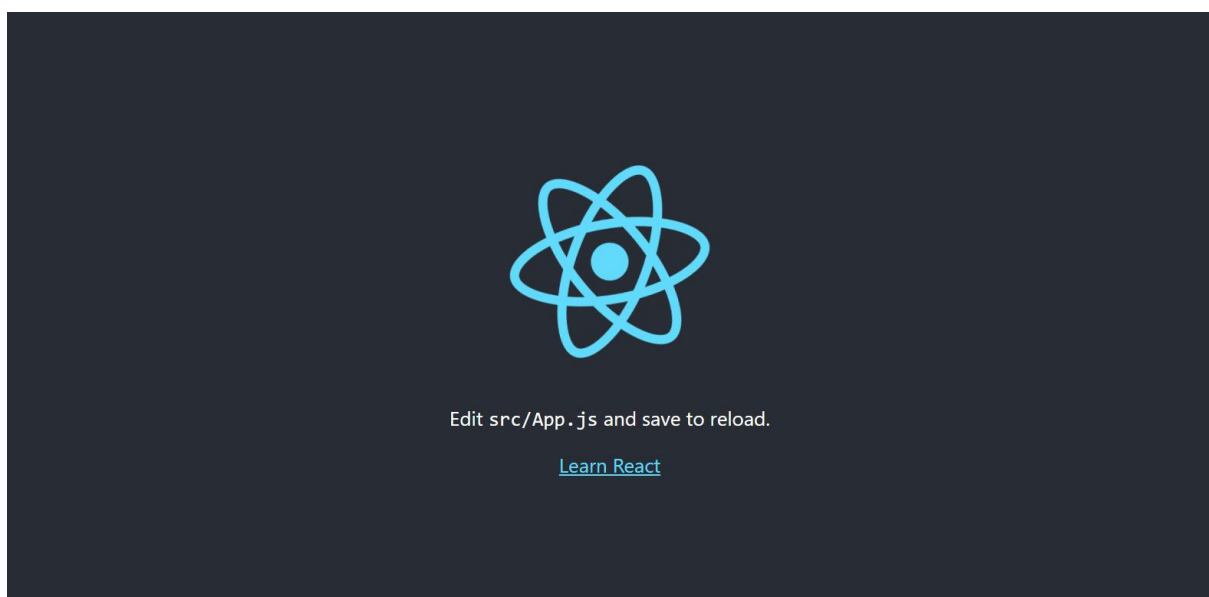
This command will start your local Development server on port 3000.

If you have a project running on port 3000, react will prompt you to run the app on another port.

If you want to set a port to run your app; use a .env file to set the port configuration as:

```
PORT=<yourPortNumber>
```

After running npm start , this default page will display in your browser



```
$ npm build
```

This will build a production ready bundle of your project.

The static files will be compressed

`$ npm eject`

This is a one-way command that cannot be undone.

It puts you in charge of the entire react tooling.

You will have to manually configure webpack , babel and other toolsets



JavaScript Concepts to understand before starting react:

This is not a guide on vanilla JS but core knowledge of vanilla JS is necessary for understanding and enjoying the use of react.

Some things worthy of understanding are:

I. Lexical Scopes

II. Ternary operators

III. Pure and higher order functions

IV. Arrays

V. ES 6 Classes

VI. Destructuring

VII. Asynchronous Programs

VIII. Modules

IX. HTTP, API, and usage

Development Tools:

IDE: Visual Studio Code

Browser: Google Chrome

React Jargons:

Elements

Elements in react are synonymous to elements in HTML

JSX

JavaScript as XML is a declarative language that enables the insertion of dynamic data to HTML templates

Components:

Piece of codes that take in optional properties known as props and returns elements via its render method

Props

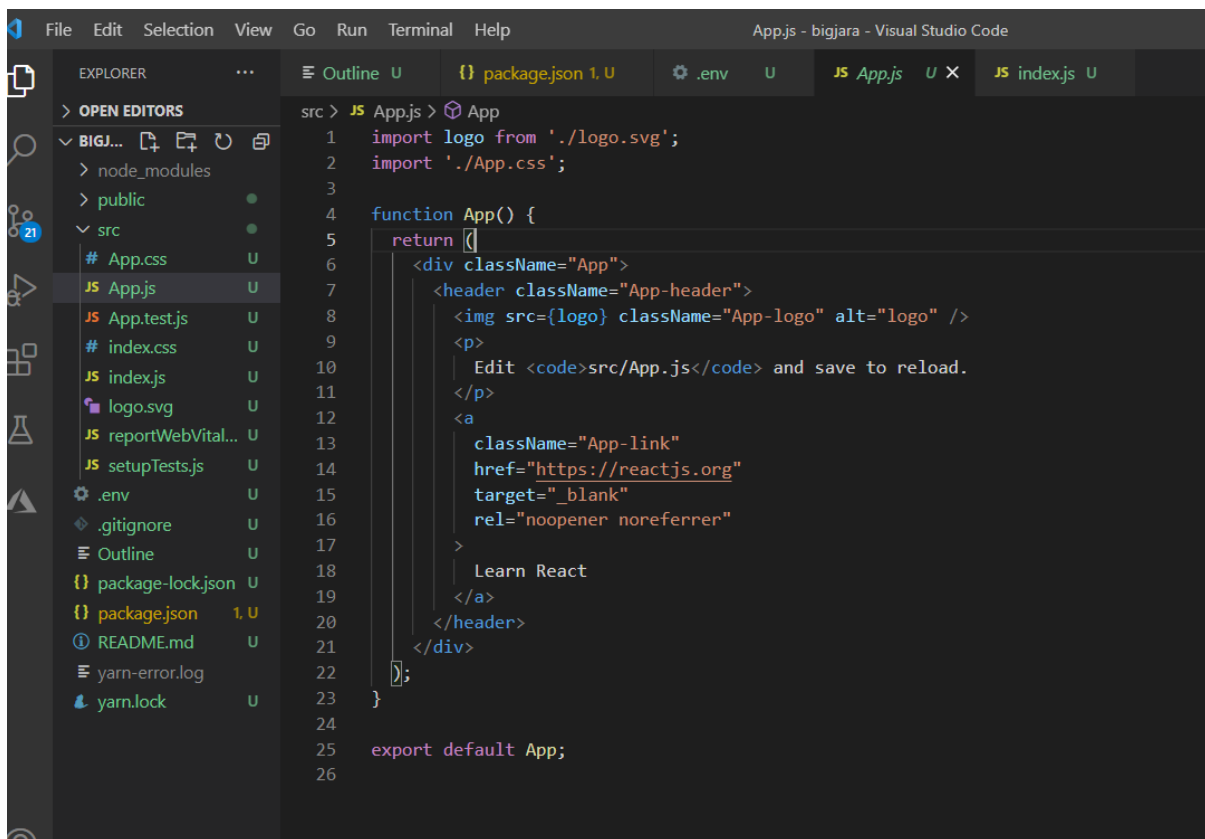
State

A state refers to a change affecting a particular part or the entirety of your UI.

Lifecycle

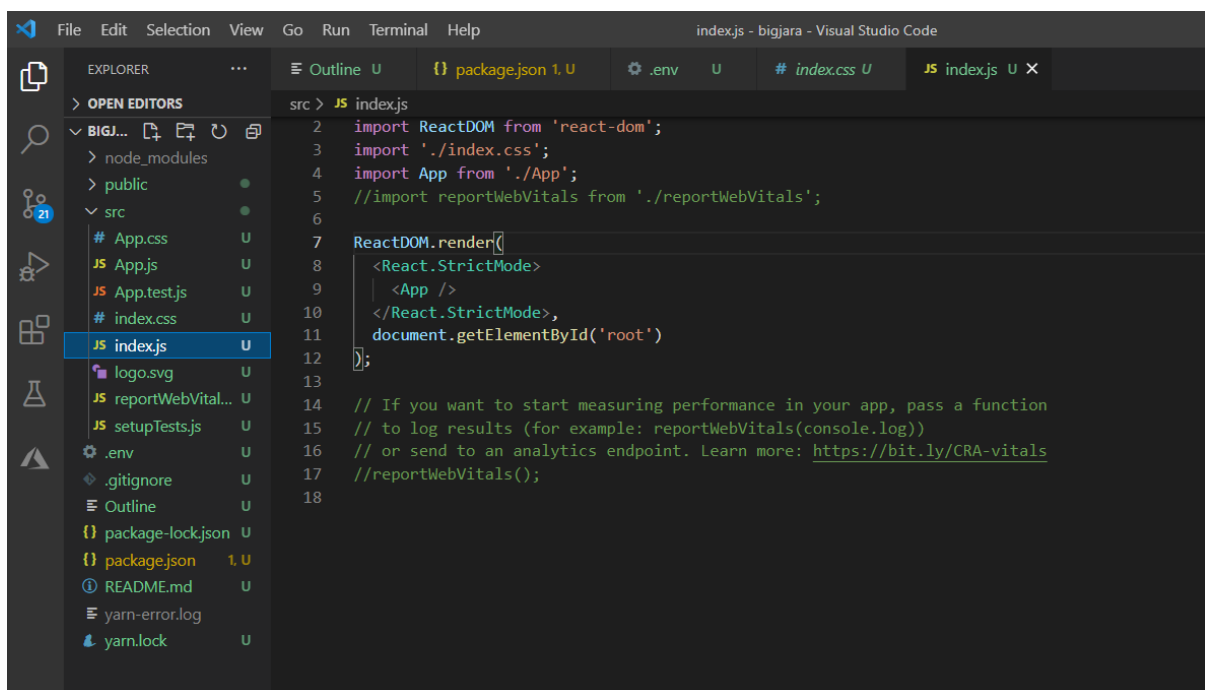
Lifecycle are the timelines of progression that a component makes.

## Understanding the default app, we just opened



The screenshot shows the Visual Studio Code editor with the 'App.js' file open. The Explorer sidebar on the left shows the project structure, including 'node\_modules', 'public', and 'src'. The 'App.js' file is selected in the Explorer. The main editor area displays the code for 'App.js', which is a function component named 'App'. The code imports 'logo' from './logo.svg' and './App.css'. It returns a JSX element with a 'div' containing a 'header' with a 'logo' and a 'p' with the text 'Edit <code>src/App.js</code> and save to reload.' Below the 'p' is a link to 'https://reactjs.org' with the text 'Learn React'. The 'App' function is exported as the default export.

```
1 import logo from './logo.svg';
2 import './App.css';
3
4 function App() {
5   return (
6     <div className="App">
7       <header className="App-header">
8         <img src={logo} className="App-logo" alt="logo" />
9         <p>
10           Edit <code>src/App.js</code> and save to reload.
11         </p>
12         <a
13           className="App-link"
14           href="https://reactjs.org"
15           target="_blank"
16           rel="noopener noreferrer"
17         >
18           Learn React
19         </a>
20       </header>
21     </div>
22   );
23 }
24
25 export default App;
```



The screenshot shows the Visual Studio Code editor with the 'index.js' file open. The Explorer sidebar on the left shows the project structure, including 'node\_modules', 'public', and 'src'. The 'index.js' file is selected in the Explorer. The main editor area displays the code for 'index.js', which imports 'ReactDOM' from 'react-dom', './index.css', and 'App' from './App'. It also imports 'reportWebVitals' from './reportWebVitals'. The code uses 'ReactDOM.render' to render the 'App' component into the 'root' element. Below the render call, there are comments about performance measurement and analytics endpoints.

```
1 import ReactDOM from 'react-dom';
2 import './index.css';
3 import App from './App';
4 //import reportWebVitals from './reportWebVitals';
5
6 ReactDOM.render(
7   <React.StrictMode>
8     <App />
9   </React.StrictMode>,
10   document.getElementById('root')
11 );
12
13 // If you want to start measuring performance in your app, pass a function
14 // to log results (for example: reportWebVitals(console.log))
15 // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
16 //reportWebVitals();
```

Line 1 and 2 uses the import statement. The import statement is used for fetching functionality contained in a file known as a module into another module in this case our App.js

To create a functionality that should be available to other code, we use the export statement.

We import React from the react library.

We also import App.css so as to be able to use the styling available in the file

There are three types of modules and they include:

- System Modules
- File System Modules
- Third party modules

System modules include classes and functions that are exposed by Nodejs.

They include Path, process, http, etc

File System Modules are modules created as part of our project.

Third party modules like react are community developed modules that we can use to build our project.

In React, you use className to denote a class since class a keyword in JavaScript.

The function App returns group of elements known as components.

We will see in a later part of this book how to create react components

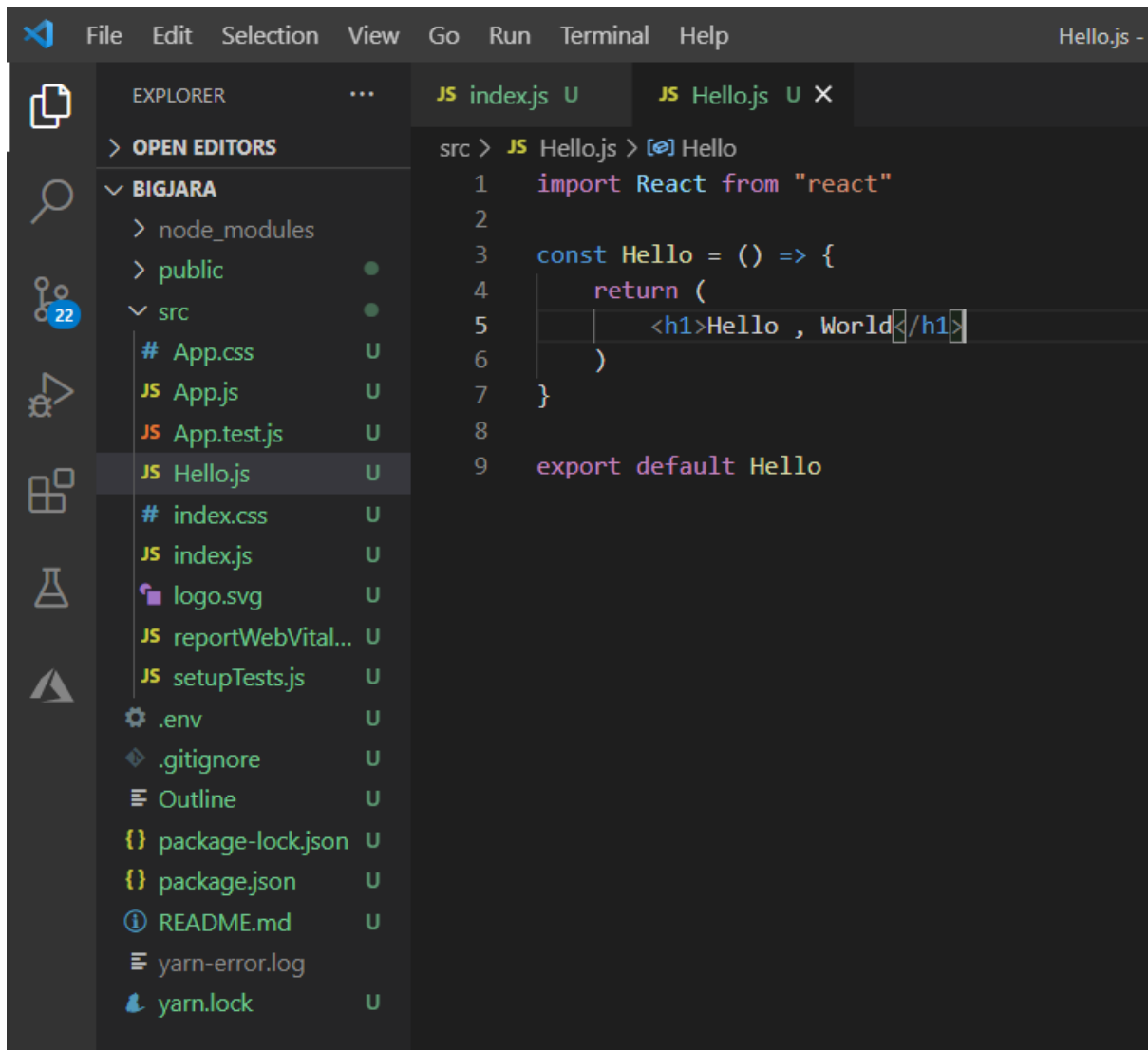
Rendering element within part of the DOM is done via react-dom

The main file for our project is index.js.

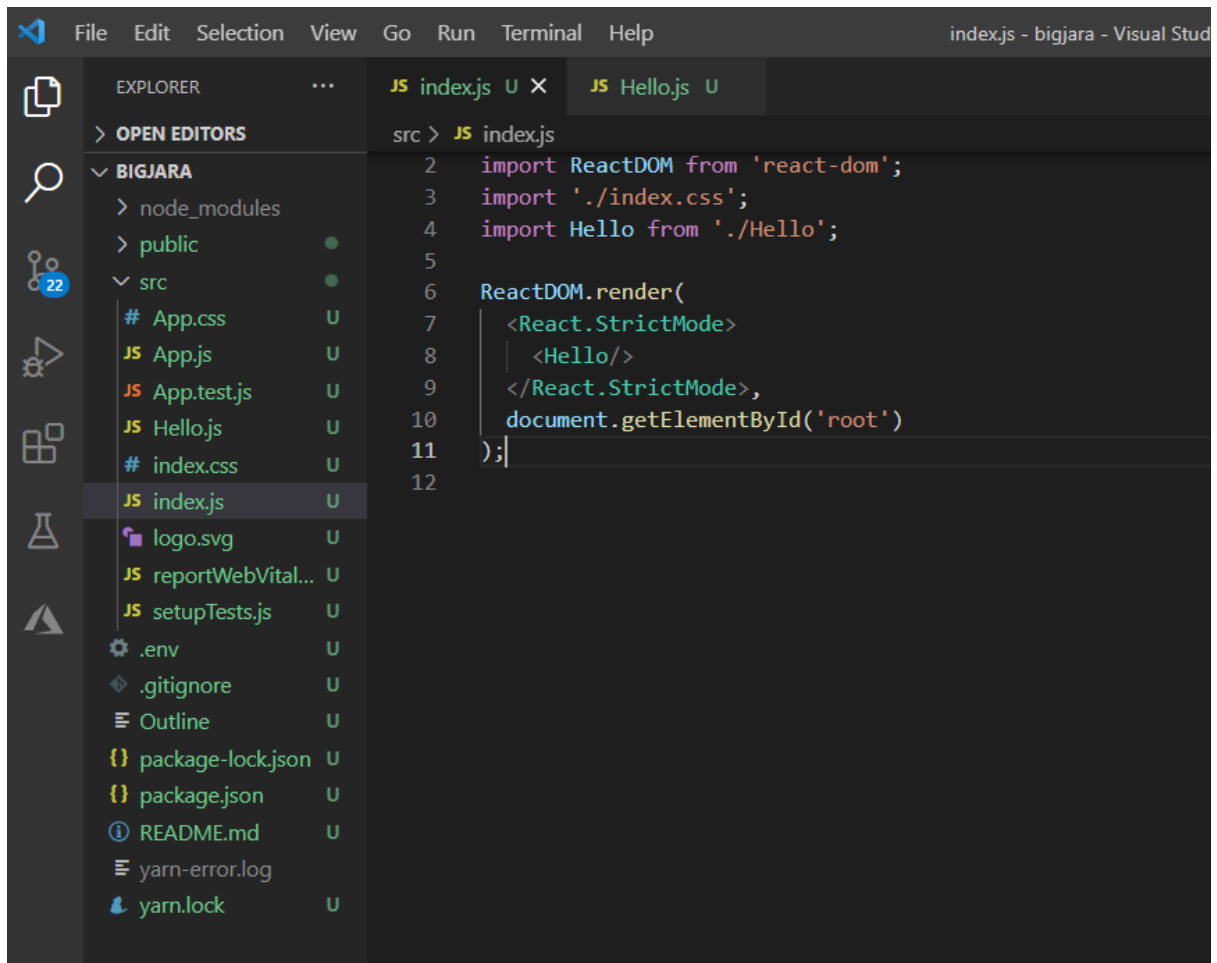
In it we have the part from line 6 that renders our App within the root node whose id is root.

## 2. Creating a Hello, World App

Open src, and create a new file call Hello.js with same content as below

A screenshot of the Visual Studio Code editor interface. The Explorer sidebar on the left shows a project named 'BIGJARA' with a 'src' folder containing several files. 'Hello.js' is highlighted in the Explorer. The main editor area shows the content of 'Hello.js', which is a React component. The code includes an import for 'React' from 'react', a function 'Hello' that returns a JSX element, and a default export. The JSX element is a heading 'Hello , World' inside an 'h1' tag. The file explorer shows files like 'App.css', 'App.js', 'App.test.js', 'index.css', 'index.js', 'logo.svg', 'reportWebVital...', 'setupTests.js', '.env', '.gitignore', 'Outline', 'package-lock.json', 'package.json', 'README.md', 'yarn-error.log', and 'yarn.lock'. The top menu bar includes 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help'. The title bar shows 'Hello.js -'.

Hello.js



## Index.js

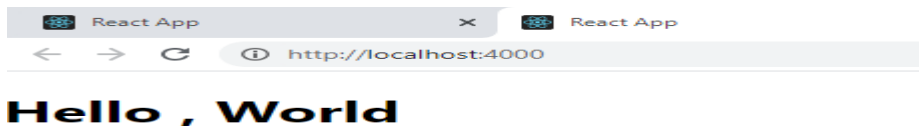


Figure 1.3 Output of our Hello World App

What is going on here:

Line 1 gets the react functionality into our component.

To import functionality from one module into another, we use the import statement.

To export functionality, we use export.

You can export more than one functionality from a module.

You can export the main functionality using a default keyword as shown in the file above.

This code above is asking react to select an element with id root and then render your component within.

If you've manually handled DOM manipulation you will be used to the following problems:

Rendering and Attaching event listeners

Run the app using:

\$ npm start

This command will open the app on port 3000

# JSX

JSX is a syntax extension to JavaScript.

It is used with react to describe what the UI will look like.

JSX produce react elements that are renderable to the DOM

This code listing below aims to highlight the difference therein between HTML, JS, and JSX

## HTML

```
<h1> Welcome to Bigjara </h1>
```

## JS

```
const appName = " Bigjara "  
const output = `Welcome to ${appName}`
```

## JSX

```
const appName = " Bigjara "  
const output = <h1> Welcome to {appName} </h1>
```

JavaScript expression of any kind is wrapped within { }

React elements are description of what you want to see on the screen.

JSX helps to prevent cross site scripting as malicious codes are converted to string before rendering

## Why JSX?

When using vanilla JS to manipulate the DOM, we usually separate technologies.

We write our HTML sperate from JS.

Over time, we come to realize that how we render a new element in the DOM is tightly coupled to how we build the UI logic for that element.



React embraces the fact that rendering logic is inherently coupled with other UI logic: how events are handled, how the state changes over time, and how the data is prepared for display.

Instead of artificially separating technologies by putting markup and logic in separate files, react separates concerns with loosely coupled units called “components” that contain both. ( <https://react.org> Official React Documentation)

### **Demonstrating JSX 1.0**

```
/**
 *@description randomNumber generates a number between 0
 and m
 */
const randomNumber = (m) => {
  try {
    if (Object.is (typeof m, "number")) {
      return Math.floor(Math.random()*m + 1)
    }
    Throw new Error("Provide valid number")
  }catch(error){
    return error
  }
}
```

```
// A component using JSX
const RenderNumber = props => {
  return (
    <p>
      A random number between 0 and {props.number}
      : {RandomNumber (props.number)}
    </p>
  )
}

ReactDOM.render(
  <RenderNumber /> ,
  document.querySelector("#root")
)
```

## Demonstration Two

```
const user = {  
  name: "Adeleke Bright",  
  avatar : "Bigjara.com/profiles/adeleke-bright/avatar"  
}
```

```
const ShowUser = props => {  
  return (  
    <>  
      <p> My name is {props.user.name}</p>  
      <img src={props.user.avatar}/>  
    </>  
  )  
}
```

```
ReactDOM.render(  
  <ShowUser /> ,  
  Document.getElementById("root")  
)
```

### Demonstration 3 - JSX can be made up of elements

```
const article = (  
  <article>  
    <header>  
      <h2> Learning ReactReact </h2>  
    </header>  
    <section>  
      <p>  
        React is a Javascript library for  
        building UI  
      </p>  
    </section>  
  </article>  
)  
ReactDOM.render(  
  {article} ,  
  Document.getElementById("root")  
)
```

## Rendering Elements

An element is the smallest building block of a react application.

It is synonymous to one created by using HTML tags.

It describes what you want to see.

React DOM has the responsibility of matching the Dom to suit the element that was created.

Elements are pure objects. They can be composed of JSX

```
const element = <h1>This is an element</h1>
```

Assuming you have this code somewhere in your html :

```
<div id="root"></div>
```

This div is known as the root node because everything in it will be managed by React DOM.

To render we use:

```
ReactDOM.Render(element , rootNode)
```

open index.html in public directory. The content of the file should look like what we have below:

```
<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>
```

Image illustrating Root node in a react application

“React Elements are immutable “. You will get to understand why we say elements in react are immutable

# Components and Props

With component, we can split our UI into independent, reusable units that can be built in isolation.

Components accepts optional argument known as props properties.

Props can be of any data type, and with it we can manipulate how we render our UI logic.

Components can be created using Function or class statement.

The examples below illustrate how to create functional and class-based components

```
src > JS Button.js > [e] button
1  import React , {Fragment , Component} from "react"
2  import "./Button.css"
3
4  const button = {
5    styleGuide : "button no-border no-input pad-10 bg-blue white-text radius-5" ,
6    textContent : "Learn more"
7  }
8
9  const ButtonA = props => {
10    return (
11      <Fragment>
12        <button
13          className={props.styleGuide}
14          role="button"
15          aria-label="button"
16        >
17          {props.textContent}
18        </button>
19      </Fragment>
20    )
21  }
22
```

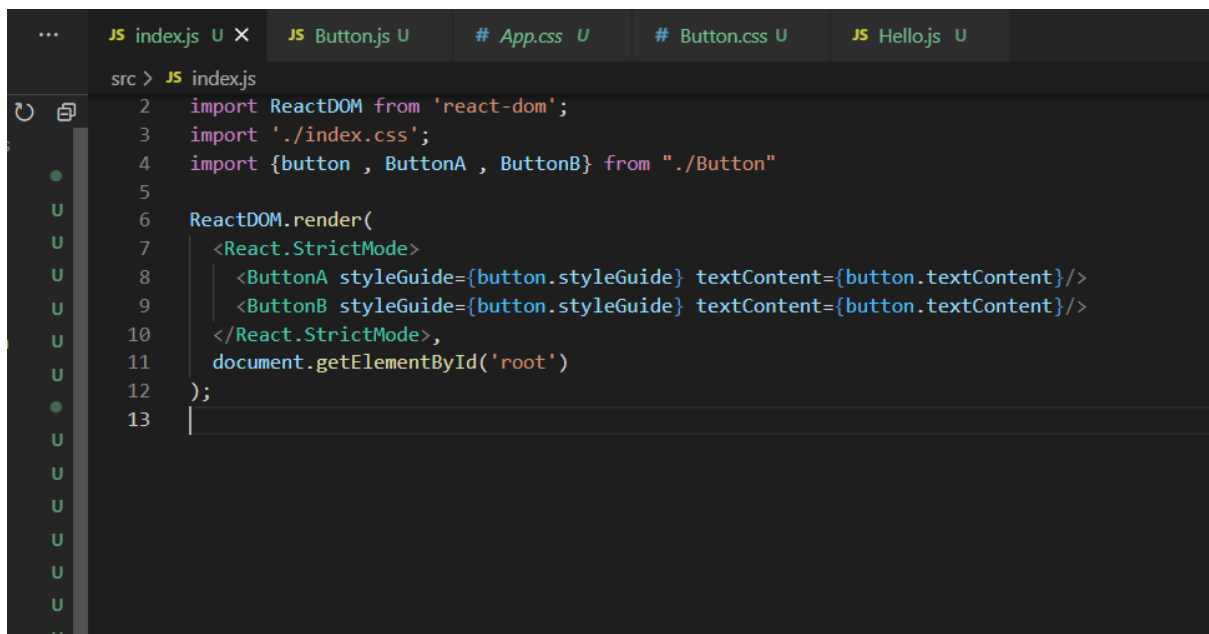
Functional Component

```

22
23 ✓ class ButtonB extends Component {
24 ✓   render(){
25     return (
26 ✓       <Fragment>
27 ✓       <button
28 ✓         className={this.props.styleGuide}
29         role="button"
30         aria-label="button"
31       >
32         {this.props.textContent}
33       </button>
34     </Fragment>
35   )
36 }
37 }
38
39 export {ButtonA , ButtonB , button}

```

## Class Component



```

src > JS index.js
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import {button , ButtonA , ButtonB} from './Button'
5
6 ReactDOM.render(
7   <React.StrictMode>
8     <ButtonA styleGuide={button.styleGuide} textContent={button.textContent}/>
9     <ButtonB styleGuide={button.styleGuide} textContent={button.textContent}/>
10  </React.StrictMode>,
11  document.getElementById('root')
12 );
13

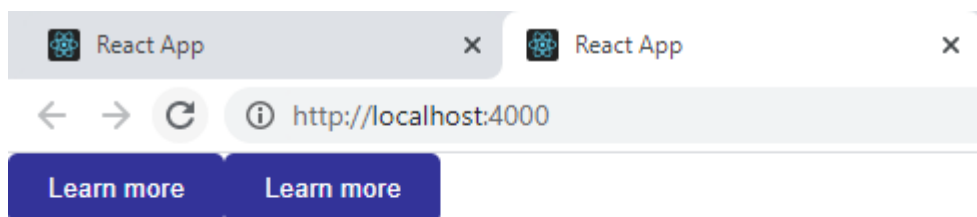
```

## Rendering the elements



```
index.js U JS Button.js U # App.css U # Button.css U X JS Hello.js U
> # Button.css > .bg-blue
1 .button {
2   padding : 10px 20px;
3 }
4
5 .no-border {
6   border : none ;
7 }
8
9 .no-outline {
0   outline: 0;
1 }
2
3 .radius-5 {
4   border-radius: 5px;
5 }
6
7 .bg-blue {
8   background : #339 ;
9 }
0
1 .white-text {
2   color : #fff;
3 }
```

Button.css as our stylesheet



Output

Elements can also instead of DOM tags be composed of Components.

When react sees elements been made up of Components, it passes JSX attributes and children in form of props to this component as a single object via props

Always start components name with capital letter.

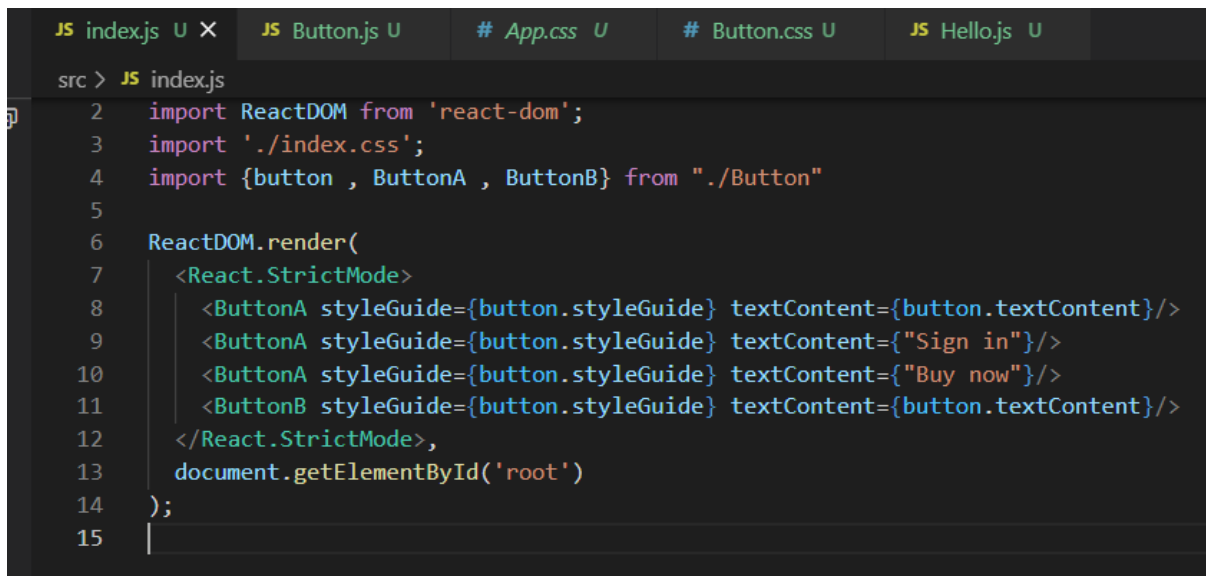
Lowercase Components are treated as DOM tags.

## Composing Components

You can reuse components in another Components.

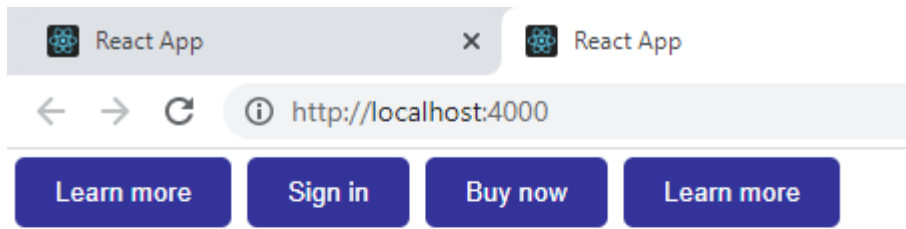
You can call Component more than once with either the same properties or different props.

See the modification to index.js and the output in the browser:



```
src > JS index.js
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import {button, ButtonA, ButtonB} from './Button'
5
6 ReactDOM.render(
7   <React.StrictMode>
8     <ButtonA styleGuide={button.styleGuide} textContent={button.textContent}/>
9     <ButtonA styleGuide={button.styleGuide} textContent={"Sign in"}/>
10    <ButtonA styleGuide={button.styleGuide} textContent={"Buy now"}/>
11    <ButtonB styleGuide={button.styleGuide} textContent={button.textContent}/>
12  </React.StrictMode>,
13  document.getElementById('root')
14 );
15 |
```

Composing of Components



## Output of Component Composition

### Extracting Component

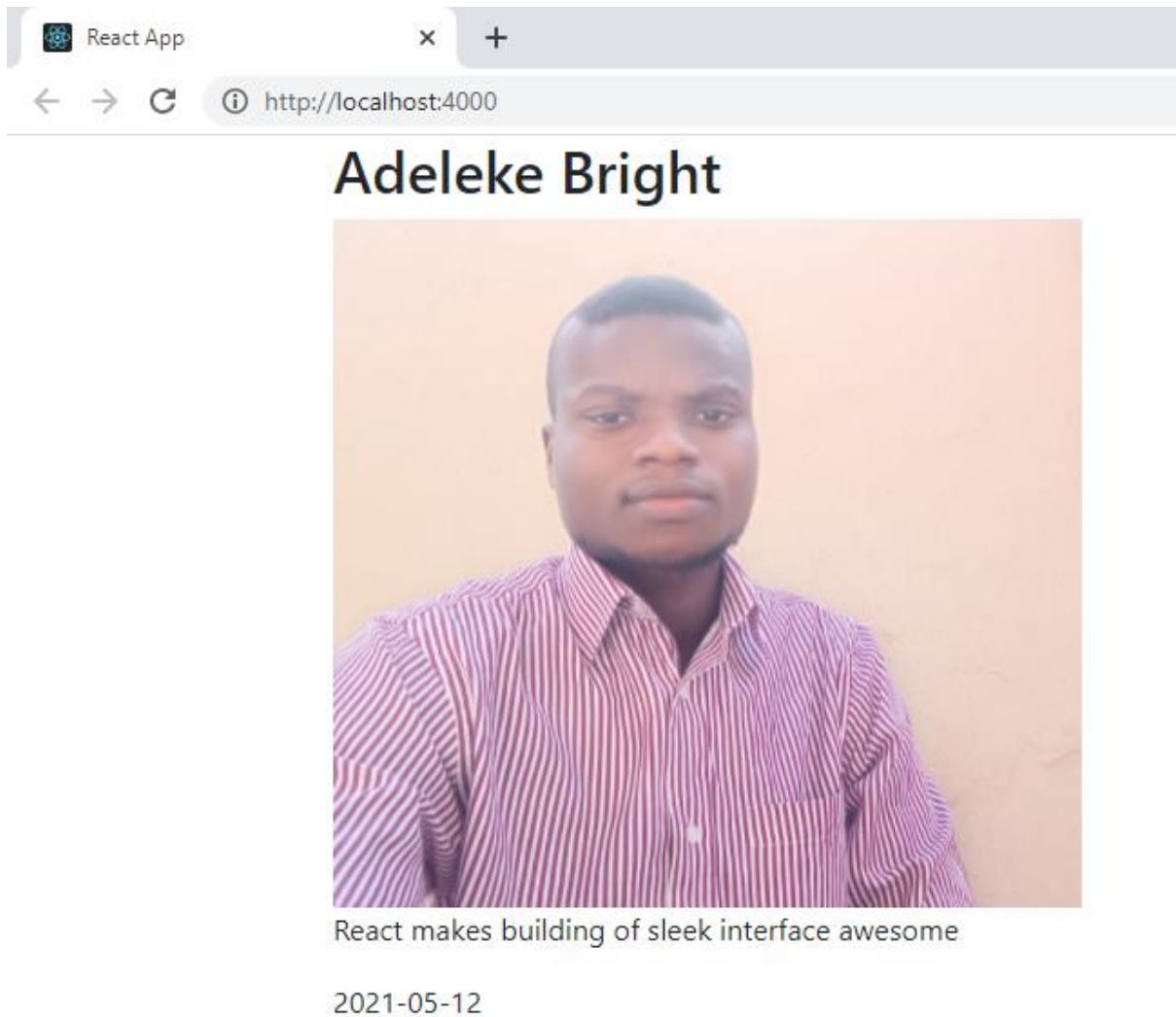
Always try to split a component into smaller components.

We can extract components from a component and reuse those components in other areas of our application.

Let us use the comment example below for illustration

```
JS index.js U # easyframer.css U JS Comment.js U # style.css U JS Button.js U
src > JS Comment.js > Comment
1 import React from "react"
2 import Bright from "../images/bright.jpg"
3 import "../css/easyframer.css"
4
5 function Comment(props){
6   return (
7     <div className="framer">
8       <div className="frame">
9         <div className="fr-md-4">
10          <div className="comment">
11            <div className="user-information">
12              <h2>{props.userInfo}</h2>
13              <img src={props.avatar} className="w-100"></img>
14            </div>
15            <p>{props.comment}</p>
16            <p>{props.commentDate}</p>
17          </div>
18        </div>
19      </div>
20    </div>
21  )
22 }
23 export default function App(pros){
24   return (
25     <Comment
26       userInfo={"Adeleke Bright"}
27       avatar={Bright}
28       comment={"React makes building of sleek interface awesome"}
29       commentDate={"2021-05-12"}
30     />
31   )
32 }
```

A compact component



### Output for comment

We can extract components from our Comment component so as to reuse those components in other areas of our application.

We can reuse the user avatar for as a login icon, we can use the Avatar for notification pop-ups, we can reuse the name in other places.

So, tightly coupling these components to Comment is one pain point in our UI logic that we need to address.

See our example below in extracting components

```
JS index.js U # easyframer.css U JS Comment.js U # style.css U JS Button.js U
src > JS Comment.js > App
1 import React from "react"
2 import Bright from "../images/bright.jpg"
3 import "../css/easyframer.css"
4
5 const Avatar =(props) =>{
6   return (
7     <figure>
8       <img
9         src={props.user.avatarUrl}
10        alt={props.user.name}
11        style={{height : "auto" , width:"100%}}
12      />
13     </figure>
14   )
15 }
16
17 const UserInfo = (props) =>{
18   return (
19     <div className="user">
20       <h2>{props.user.name}</h2>
21       <Avatar user={props.user} />
22     </div>
23   )
24 }
25
```

```
25
26 const Comment = (props) => {
27   return (
28     <div className="fr-md-4">
29       <div className="comment">
30         <UserInfo user={props.author} />
31         <p>{props.author.comment}</p>
32         <p>{props.author.commentDate}</p>
33       </div>
34     </div>
35   )
36 }
37
38 const author = {
39   name : "Adeleke Bright" ,
40   comment : "React makes building of sleek user interface awesome" ,
41   commentDate : "2021-05-12" ,
42   avatarUrl : Bright
43 }
44 export default function App(){
45   return (
46     <div className="framer">
47       <div className="frame">
48         <Comment author={author}/>
49       </div>
50     </div>
51   )
52 }
```

# Adeleke Bright



React makes building of sleek user interface awesome

2021-05-12

## State and Lifecycle Method

How and when do we update the UI?

How do we rewrite data already attached to a rendered element?

These questions are solved using state management principles.

React does not re-render the entire DOM when part of the UI changes.

In React, presentation and interaction are closely knit. This makes it easy to truly reuse components.

A component should handle both rendering and any UI related logic.

This makes a component to be truly reusable.

An application state although similarly like a props is private to a component

A state is a transition unlike a props that is static, a state is dynamic as it reacts to changes.

A stateful Component that is a component with implementation details for state management was primarily implemented using class Component before the introduction of react hooks.

Now, with the help of react hooks, we can build stateful Component using functions.

Lifecycle methods are functions that automatically runs as our App is loaded, when an update occurs, when an error happens, or when a part of the UI is entirely removed.

When a new element is created, it is called "mounting".

When you are removing the element that is unmounting.

Lifecycle methods include but not limited to the following:

`componentDidMount` : Use for rendering new elements

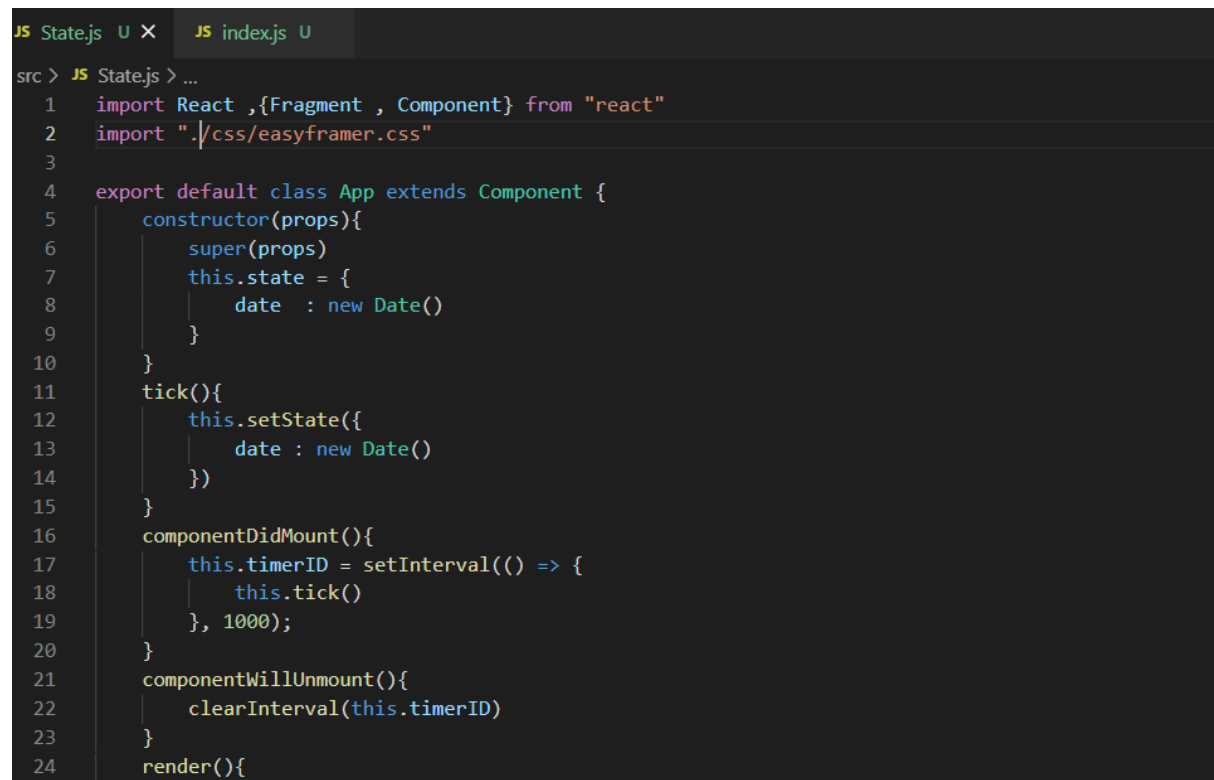
`componentWillUnmount` : When a part of the UI is destroyed , this lifecycle method helps in cleaning and freeing resources for other parts of our UI



**componentWillUpdate** : Use this method when part of the UI needs to be updated with dynamic data

**componentWillCatch** : This lifecycle method is used for error handling when part of a UI returns error instead of rendering an element

See the code below for explanation on how state management operates

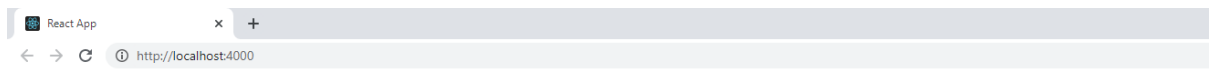


```
JS State.js U X JS index.js U
src > JS State.js > ...
1 import React,{Fragment , Component} from "react"
2 import "../css/easyframer.css"
3
4 export default class App extends Component {
5   constructor(props){
6     super(props)
7     this.state = {
8       date : new Date()
9     }
10  }
11  tick(){
12    this.setState({
13      date : new Date()
14    })
15  }
16  componentDidMount(){
17    this.timerID = setInterval(() => {
18      this.tick()
19    }, 1000);
20  }
21  componentWillUnmount(){
22    clearInterval(this.timerID)
23  }
24  render(){
```

```

22     clearInterval(this.timerID)
23   }
24   render(){
25     return (
26       <Fragment>
27         <div className="framer">
28           <div
29             className="frame"
30             style={{
31               flexDirection : "column" ,
32               alignItems : "center" ,
33               paddingTop:"30px"
34             }}
35           >
36             <h1>
37               <span>The time is </span>
38               <span>{this.state.date.toLocaleTimeString()}</span>
39             </h1>
40           </div>
41         </div>
42       </Fragment>
43     )
44   }
45 }

```



The time is 8:43:03 PM

# Handling Event

Event Handling in react is almost same like event handling in vanillaJs only with slight difference.

There is no use of `addEventListener` as event are named using camelCase notation within the elements and handlers are passed in using JSX functions instead of strings.

In HTML, you will have:

```
<button onclick="handleClick">Click</button>
```

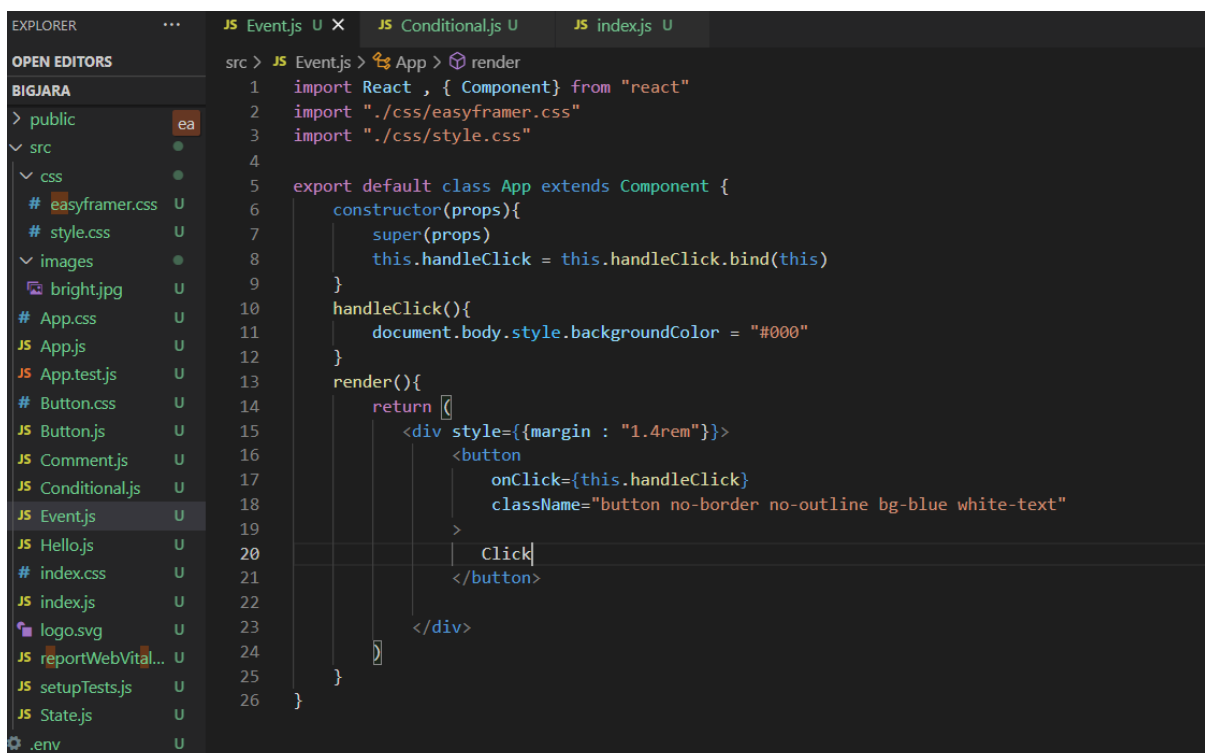
In React, it is:

```
<button onClick={handleClick}> click</button>
```

React events are named using camelCase notation.

The handlers are passed in as functions using JSX instead of string.

See example below:

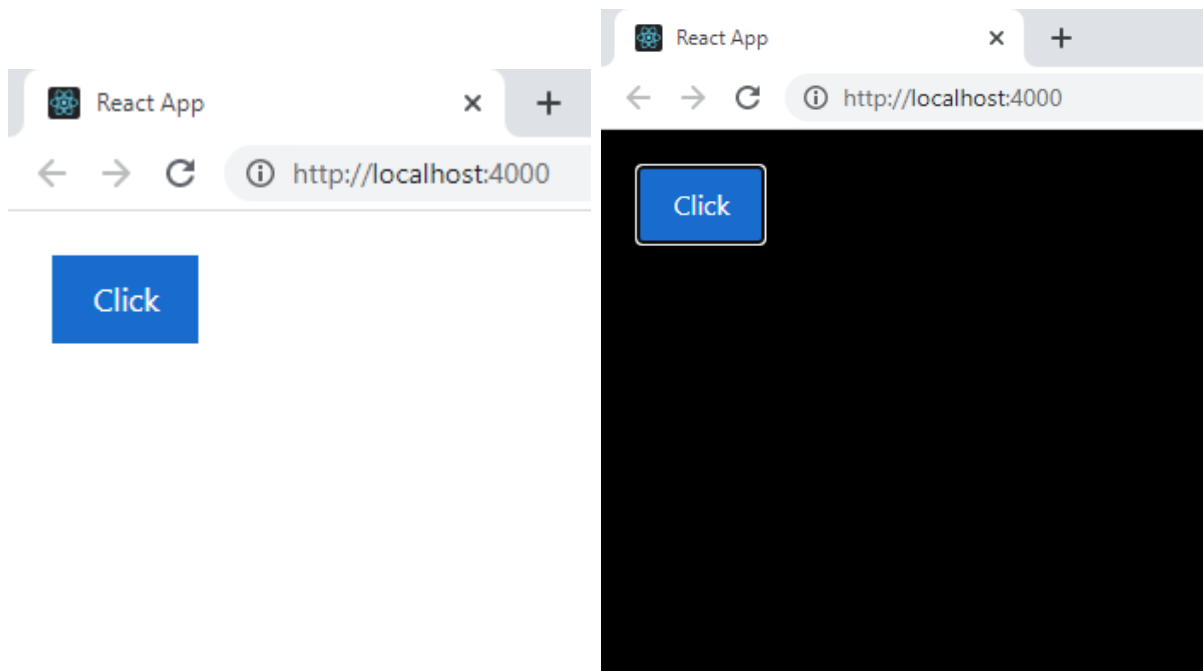
A screenshot of a code editor with a dark theme. On the left, the 'EXPLORER' sidebar shows a file tree with folders like 'public' and 'src', and various files including 'App.js', 'Button.js', 'Event.js', and 'index.js'. The 'Event.js' file is selected. The main editor area shows the code for 'App.js'. It imports 'React' and 'Component' from 'react', and two CSS files. It defines a class 'App' that extends 'Component'. The 'constructor' calls 'super(props)' and binds 'this.handleClick'. The 'handleClick' method sets 'document.body.style.backgroundColor = "#000"'. The 'render' method returns a JSX element: a 'div' with a 'button'. The 'button' has an 'onClick' prop set to 'this.handleClick' and a 'className' of 'button no-border no-outline bg-blue white-text'. The button's text is 'Click'.

```
src > JS Eventjs U X JS Conditional.js U JS index.js U
OPEN EDITORS
BIGJARA
> public
  src
    css
      # easyframer.css U
      # style.css U
    images
      # bright.jpg U
      # App.css U
      JS App.js U
      JS App.test.js U
      # Button.css U
      JS Button.js U
      JS Comment.js U
      JS Conditional.js U
      JS Event.js U
      JS Hello.js U
      # index.css U
      JS index.js U
      # logo.svg U
      JS reportWebVital... U
      JS setupTests.js U
      JS State.js U
      .env U

1  import React , { Component} from "react"
2  import "../css/easyframer.css"
3  import "../css/style.css"
4
5  export default class App extends Component {
6    constructor(props){
7      super(props)
8      this.handleClick = this.handleClick.bind(this)
9    }
10   handleClick(){
11     document.body.style.backgroundColor = "#000"
12   }
13   render(){
14     return (
15       <div style={{margin : "1.4rem"}}>
16         <button
17           onClick={this.handleClick}
18           className="button no-border no-outline bg-blue white-text"
19         >
20           Click
21         </button>
22       </div>
23     )
24   }
25 }
26
```

## Event Handling

When we click on the button, the background of the page turns dark because we attached an event listener on the button



The common method is to write your event listeners as methods inside your class Component.

In JS, “this” is not bounded. It gets lost within a newer scope, in order to preserve “this”, we bind it to the calling method by using the bind statement

## Conditional Rendering

Conditional rendering refers to the process of displaying an element only if a certain condition is met.

Assuming we want to build an application that allows users to switch between dark mode and light mode, we will need to conditionally render our UI to meet with users' expectations.

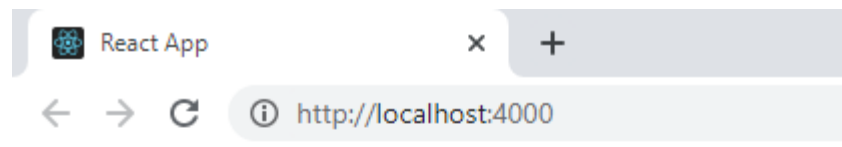
Let us consider the example below for more detail:

```
JS Event.js U • JS Conditional.js U JS index.js U
src > JS Event.js > ...
1 import React, { Component } from "react"
2 import "../css/easyframer.css"
3 import "../css/style.css" |
4
5 export default class App extends Component {
6   constructor(props){
7     super(props)
8     this.state = {
9       mode : "Light" ,
10      bgColor : "#000"
11    }
12    this.handleClick = this.handleClick.bind(this)
13  }
14  handleClick(){
15    this.setState({
16      mode : this.state.mode.startsWith("L") ? "Dark" : "Light" ,
17      bgColor: this.state.bgColor === "#000" ? "#fff" : "#000"
18    })
19    document.body.style.backgroundColor = this.state.bgColor
20  }
21  render(){
22    return (
23      <div style={{margin : "1.4rem"}}>
24        <button
25          onClick={this.handleClick}
26          className="button no-border no-outline bg-blue white-text"
27        >
28          {this.state.mode}
29        </button>
30      </div>
31    )
32  }
33 }
```

We conditionally render a dark background or light background depending on the user's choice on interaction

# List and Keys

```
JS List.js U JS Conditional.js U JS index.js U
src > JS List.js > [x] StudentList
  1 import React from "react"
  2
  3 let users = ["Femi" , "Fuad" , "Mohammed" , "Promise" , "Ayo"]
  4
  5 v const StudentList = props => {
  6 v   const students = props.students.map((student , i) =>
  7 v     <li key={i}>
  8     |   {student}
  9     | </li>
 10   )
 11   return(
 12 v     <ol>
 13     |   {students}
 14     | </ol>
 15   )
 16 }
 17
 18 v export default function App(){
 19   return (
 20 v     <>
 21     |   <h1>Working with List</h1>
 22     |   <StudentList students={users} />
 23     | </>
 24   )
 25 }
```



## Working with List

1. Femi
2. Fuad
3. Mohammed
4. Promise
5. Ayo

# Forms

```
5 export default class App extends Component {
6   constructor(props){
7     super(props)
8     this.state = {
9       message: ""
10    }
11    this.handleBlur = this.handleBlur.bind(this)
12  }
13  handleBlur(e){
14    const {value} = e.target
15    this.setState({
16      message : `Your name is ${value}`
17    })
18  }
19  render(){
20    return (
21      <div className="frame">
22        <div className="fr-4">
23          <form className="shadow bg-white pad-20 radius-5 m-b-1">
24            <label htmlFor="userName">Username</label>
25            <input
26              type="text"
27              placeholder="Enter your name"
28              className="input input-border-faint pad-10 m-b-1"
29              onBlur = {this.handleBlur}
30            />
31            <p>{this.state.message}</p>
32          </form>
33        </div>
34      </div>
35    )
36  }
```

React App

⌵

⌵

⏪ ⏩ ↺ ⓘ http://localhost:4000

Username

Your name is Adebaba