# REACT

Developed by Alabian Solutions Ltd

# CHAPTER 1

## Introduction to React

React.JS is a simple, feature rich, component based JavaScript UI library. It can be used to develop small applications as well as big, complex applications. React.JS provides minimal and solid feature set to kick-start a web application. React community compliments React library by providing large set of ready-made components to develop web application in a record time. React community also provides advanced concept like state management, routing, etc., on top of the React library.

## React versions

The initial version, 0.3.0 of React is released on May, 2013 and the latest version, *17.0.1* is released on October, 2020. The major version introduces breaking changes and the minor version introduces new feature without breaking the existing functionality. Bug fixes are released as and when necessary. React follows the *Sematic Versioning (semver)* principle.

## Features

The salient features of *React library* are as follows –

- Solid base architecture
- Extensible architecture
- Component based library
- JSX based design architecture
- Declarative UI library

## Benefits

Few benefits of using *React library* are as follows –

- Easy to learn
- Easy to adept in modern as well as legacy application
- Faster way to code a functionality
- Availability of large number of ready-made component
- Large and active community

## Applications

Few popular websites powered by *React library* are listed below –

- *Facebook*, popular social media application
- *Instagram*, popular photo sharing application
- *Netflix*, popular media streaming application
- *Code Academy*, popular online training application
- *Reddit*, popular content sharing application

As you see, most popular application in every field is being developed by *React Library*.

## What is React?

React is a JavaScript library for building fast and interactive user interface for the web as well as mobile application. React was developed by Facebook in 2001 and currently by far the most popular JavaScript library to build user interface.

## How does React work?

React work in a way that it helps to update or refresh a page without the page loading. For example you can easily get Facebook notification even when the Facebook page or browser is not loaded or refreshed.

## Why is React so popular?

- ❑ Easy creation of dynamic web applications
- ❑ Re-usable components
- ❑ Can be used for mobile applications
- ❑ Performance enhancement
- ❑ Easy to learn
- ❑ Large and active community

# CHAPTER 2

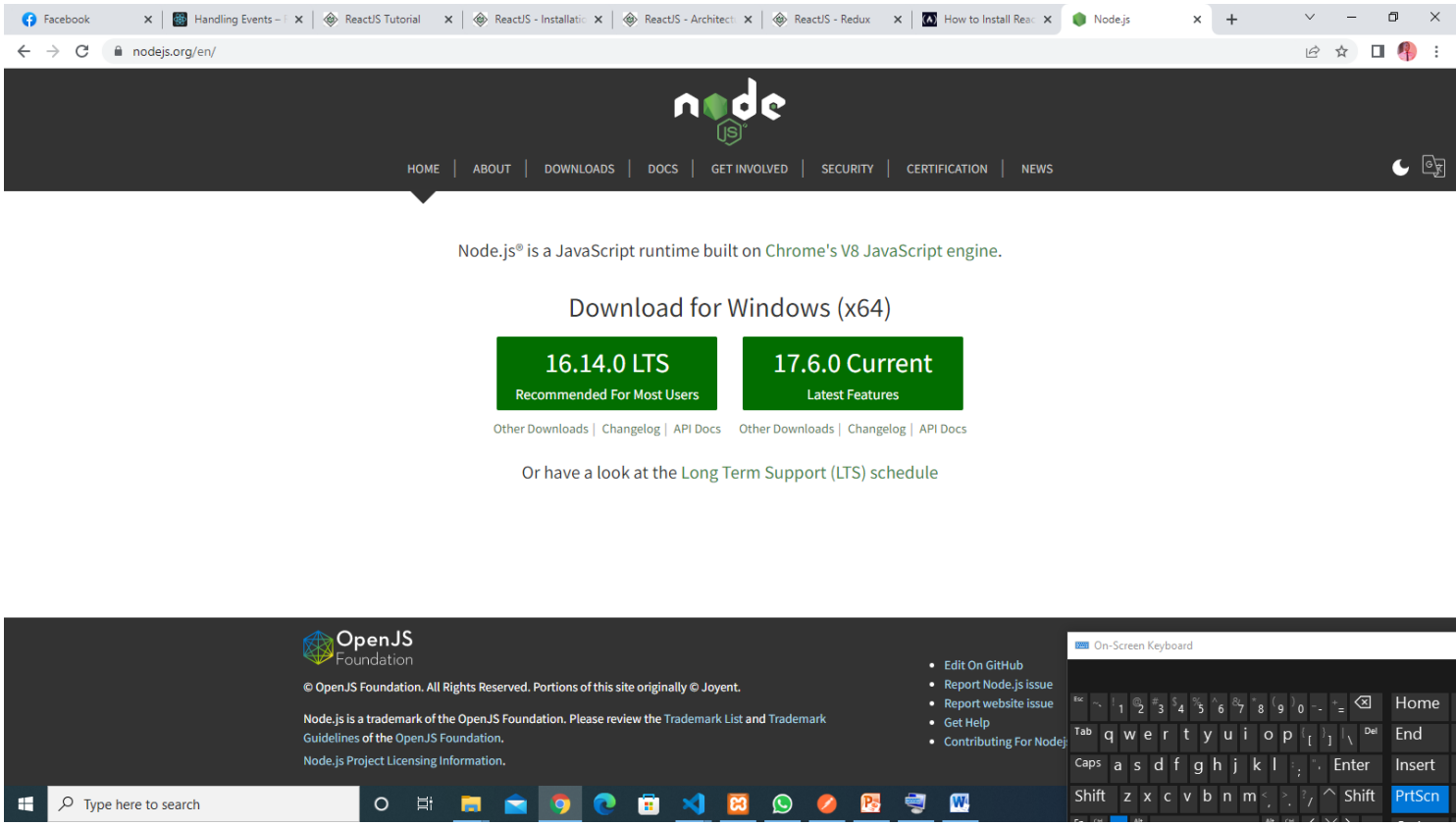
## Installation of React

To install react you must first download and install node.

## How to Download & Install Node.js

First of all, you are going to need NPM (or Yarn, alternatively). Let's use NPM for this example.

If you don't have it installed on your system, then you need to head to the official Node.js website to download and install Node, which also includes NPM (Node Package Manager).




Select the "Recommended For Most Users" button and download the current version for your operating system.

After you download and install Node, start your terminal/command prompt and run node -v and npm -v to see which versions you have.

## What is create-react-app?

Since it is complicated and takes a lot of time, we don't want to configure React manually. create-react-app is a much easier way which does all the configuration and necessary package installations for us automatically and starts a new React app locally, ready for development.

Another advantage of using create-react-app is that you don't have to deal with Babel or Webpack configurations. All of the necessary configurations will be made by create-react-app for you.

According to the React documentation, create-react-app is one of the officially supported ways to create single-page applications in React. You can find other ways here.

## How to Install Create-React-App

In order to install your app, first go to your workspace (desktop or a folder) and run the following command:

```
npx create-react-app my-app
```

The installation process may take a few minutes. After it is done, you should see a folder that appears in your workspace with the name you gave to your app.

Note: If you're on Mac and receiving permission errors, don't forget to be a super user first with the sudo command.

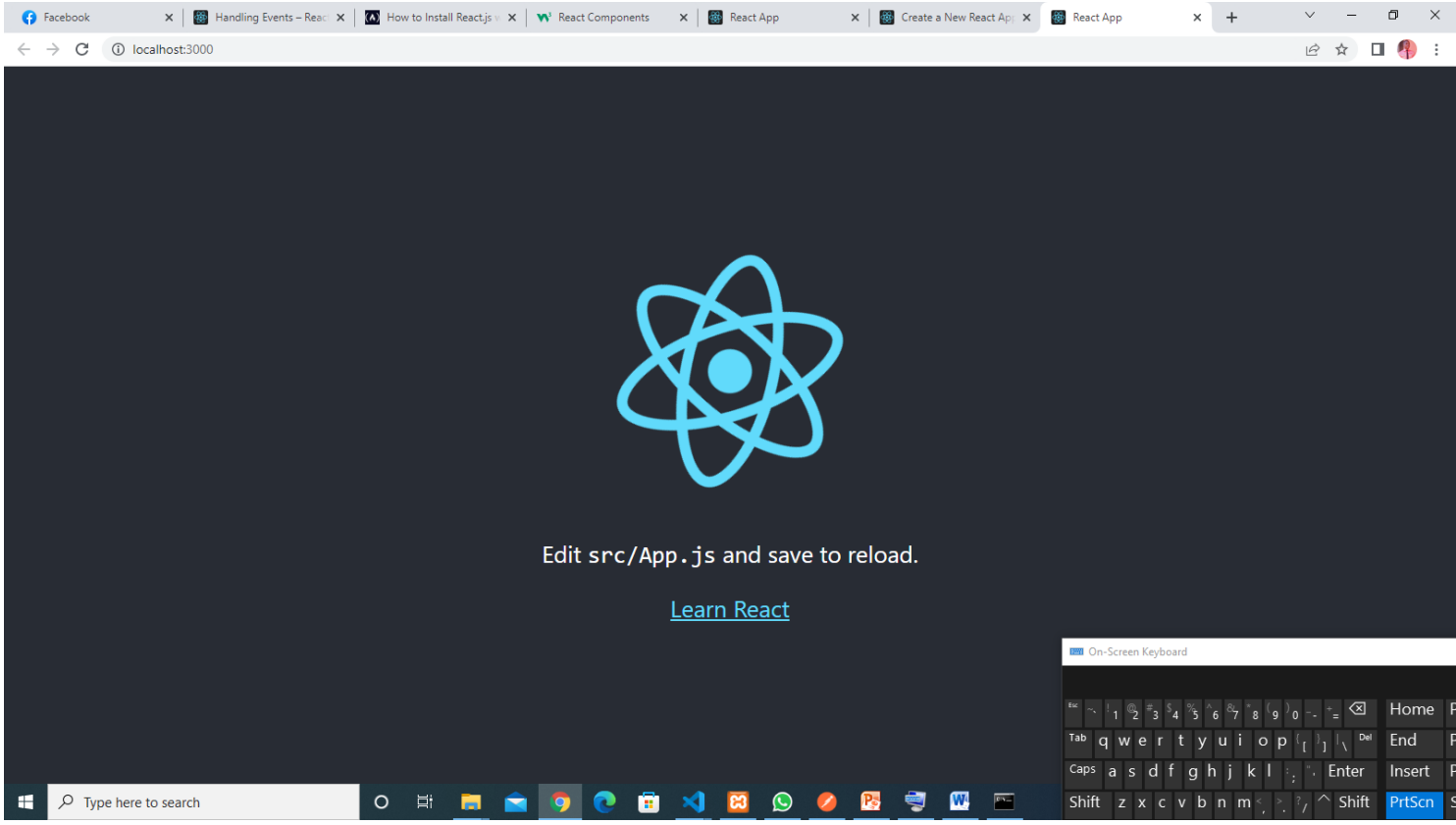
## How to Run the App You Created with Create-React-App

After the installation is completed, change to the directory where your app was installed:

```
cd my-app
```

and finally run npm start to see your app live on localhost:

```
npm start
```




If you see something like this in your browser, you are ready to work with React. Congratulations! :)

# CHAPTER 3

## What are Components?

Components are the building blocks of any React applications. Components are Re-usable.

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.

Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen.

## Types of Components

There are two types of components in React. They are:

1. Functional component
2. Class component

## Functional component

Functional Components are JavaScript functions. They return HTML which describes the User interface.

**Syntax:**

```
function ternary() {
  return (
    <div>ternary</div>
  )
}
```

**Example of Functional component:**

```
import React from 'react'

function App() {
  return (
    <div>
      <h1>Hello world!</h1>
    </div>
  )
}
```

```
export default App
```

## Class component

Class components are regular ES6 class that extends the component class. They must contain a render method which returns HTML.

**Syntax:**

```
class ternary extends Component {
  render() {
    return (
      <div>ternary</div>
    )
  }
}
```

**Example of class component:**

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

class App extends Component {
  render() {
    return (
      <div>
        <h1>Welcome</h1>
        <p>Mr. John Doe</p>
      </div>
    )
  }
}

export default App
```

## Differences between Functional and Class components

| FUNCTIONAL COMPONENTS | CLASS COMPONENTS |
| --- | --- |
| 1. Absence of “this” keyword | 1. Presence of “this” keyword |
| 2. They use React Hooks as their states | 2. They maintain their own private data-state |
| 3. Simple functions | More rich features |

## CLASS ACTIVITIES

1. Create a functional component and display it on the browser
2. Create a class component and display it on the browser
3. State the difference between functional and class based components

# CHAPTER 4

## What is JSX?

JSX stands for JavaScript XML.

JSX allows us to write HTML in React.

JSX makes it easier to write and add HTML in React.

## Coding JSX

JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement() and/or appendChild() methods.

JSX converts HTML tags into react elements.

**Example:**

```
const simple = <h1> Hello world </h1>
```

As you can see in the example above, JSX allows us to write HTML directly within the JavaScript code.

const is a JavaScript code while <h1> is a HTML code. Therefore the combination of JavaScript and HTML makes up JSX

**JavaScript + HTML = JSX**

## Expressions in JSX

With JSX you can write expressions inside curly braces { }.

The expression can be a React variable, or property, or any other valid JavaScript expression. JSX will execute the expression and return the result:

## Example

Execute the expression 5 + 5:

```
const myelement = <h1>React is {5 + 5} times better with JSX</h1>;
```

```
// Output
React is 10 times better with JSX
```

## One Top Level Element

The HTML code must be wrapped in *ONE* top level element.

So if you like to write *two* paragraphs, you must put them inside a parent element, like a div element.

## Example

Wrap two paragraphs inside one DIV element:

```
const myelement = (
    <div>
      <p>I am a paragraph.</p>
      <p>I am a paragraph too.</p>
    </div>
  );
```

JSX will throw an error if the HTML is not correct, or if the HTML misses a parent element.

Alternatively, you can use a "fragment" to wrap multiple lines. This will prevent unnecessarily adding extra nodes to the DOM.

A fragment looks like an empty HTML tag: <></>.

## Example

Wrap two paragraphs inside a fragment:

```
const myelement = (
    <>
      <p>I am a paragraph.</p>
      <p>I am a paragraph too.</p>
    </>
  );
```

## Elements Must be Closed

JSX follows XML rules, and therefore HTML elements must be properly closed.

### Example

Close empty elements with />

```
const myelement = <input type="text" />;
```

JSX will throw an error if the HTML is not properly closed.

## Attribute class = className

The class attribute is a much used attribute in HTML, but since JSX is rendered as JavaScript, and the class keyword is a reserved word in JavaScript, you are not allowed to use it in JSX.

**Use attribute className instead.**

JSX solved this by using className instead. When JSX is rendered, it translates className attributes into class attributes.

### Example

Use attribute className instead of class in JSX:

```
const myelement = <h1 className="myclass">Hello World</h1>;
```

## JSX Attribute styling

```
import React from 'react'

function Car() {
  return (
    <>
        <h1>Good boy</h1>
        <h2 style={{color:'green', backgroundColor:'yellow'}}>Good car</h2>
    </>
  )
}

export default Car
```

## CLASS ACTIVITY

1. Write an expression in JSX and any attribute styling of your choice to it.

# CHAPTER 5

## Styling React Using CSS

### Inline Styling

To style an element with the inline style attribute, the value must be a JavaScript object:

### Example:

```
import React from 'react'

function Car() {
  return (
    <>
        <h1>Good boy</h1>
        <h2 style={{color:'green', backgroundColor:'yellow'}}>Good car</h2>
    </>
  )
}

export default Car
```

Note: **In JSX, JavaScript expressions are written inside curly braces, and since JavaScript objects also use curly braces, the styling in the example above is written inside two sets of curly braces {{}}.**

### camelCased Property Names

Since the inline CSS is written in a JavaScript object, properties with two names, like background-color, must be written with camel case syntax:

### Example:

Use backgroundColor instead of background-color:

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

```
class Car extends Component {
  render() {
    return (
      <div>
            <h1 style={{backgroundColor: "lightblue"}}>This is my car!</h1>
             <p>Add a little style!</p>
      </div>
    )
  }
}

export default Car
```

## JavaScript Object

You can also create an object with styling information, and refer to it in the style attribute:

## Example:

Create a style object named mystyle:

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

class Car extends Component {

  render() {
    const mystyle = {
        color: "white",
        backgroundColor: "DodgerBlue",
        padding: "10px",
        fontFamily: "Arial",
        fontSize: '50px'

      };
    return (
      <div>
            <h1 style={mystyle}>This is my car!</h1>
             <p>Add a little style!</p>
      </div>
    )
```

```
  }
}

export default Car
```

## CSS Stylesheet

You can write your CSS styling in a separate file, just save the file with the .css file extension, and import it in your application.

## style.css:

Create a new file called "style.css" and insert some CSS code in it:

```
body {
    background-color: #282c34;
    color: white;
    padding: 40px;
    font-family: Arial;
    text-align: center;
  }
```

**Note:** You can call the file whatever you like, just remember the correct file extension.

Import the stylesheet in your application:

```
import style from './style.css'
```

## CLASS ACTIVITY

1. Recreate the following code snippet in your computer.
2. Using external CSS styling change the background color of the body

3. Change the color of <h1> and <p> tag elements to be white
4. Change the font family
5. Display on the browser

```
import React from 'react'

function Texting() {
  return (
    <div>
        <h1>Alabian Solution Limited</h1>
        <p>This is the home of web programming</p>
    </div>
  )
}

export default Texting
```

# CHAPTER 6

## What are Props?

Props are arguments passed into React components.

Props are passed to components via HTML attributes.

**props stands for properties.**

## React Props

React Props are like function arguments in JavaScript *and* attributes in HTML.

To send props into a component, use the same syntax as HTML attributes:

## Example

Add a "brand" attribute to the Car element:

```
<Car brand="Ford" />;
```

The component receives the argument as a props object:

Use the brand attribute in the component:

```
import React from 'react'

function Car(props) {
  return (
    <div>
        <h2>I am a { props.brand }!</h2>;
    </div>
  )
}
```

## Pass Data

Props are also how you pass data from one component to another, as parameters.

## Example

Send the "brand" property from the Garage component to the Car component:

```
function Car(props) {
    return <h2>I am a { props.brand }!</h2>;
  }

  function Garage() {
    return (
      <>
        <h1>Who lives in my garage?</h1>
        <Car brand="Ford" />
      </>
    );
  }

  ReactDOM.render(<Garage />, document.getElementById('root'));
```

If you have a variable to send, and not a string as in the example above, you just put the variable name inside curly brackets:

## Example

Create a variable named carName and send it to the Car component:

```
function Car(props) {
    return <h2>I am a { props.brand.model }!</h2>;
  }

  function Garage() {
```

```
    const carInfo = { name: "Ford", model: "Mustang" };
    return (
      <>
        <h1>Who lives in my garage?</h1>
        <Car brand={ carInfo } />
      </>
    );
  }

  ReactDOM.render(<Garage />, document.getElementById('root'));
```

Example:

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

class Bike extends Component {
  render() {
    return (
      <div>
          <h1>The name of my motorcycle is {this.props.name}</h1>
      </div>
    )
  }
}

export default Bike
```

## Destructuring of Props

You can destructure props in both Class and Functional components.

**Syntax:**

```
const {title} = props
```

**Destructuring of Props in Functional component**

```
import React from 'react'
```

```
function Bike(props) {
   // console.log(props);
   const{name, model, year} = props

  return (
    <div>
        <h1>The name of my motorcycle is {name}</h1>
        <h1>The model of my motorcycle is {model}</h1>
        <h1>My motorcycle was bought in the year {year}</h1>
    </div>
  )
}

export default Bike
```

```
// Output

The name of my motorcycle is Power Bike
The model of my motorcycle is GrandPix
My motorcycle was bought in the year 2022
```

**Destructuring of Props in Class component**

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

class Bike extends Component {
  render() {
      const{name, model, year} = this.props
    return (
      <div>
          <h1>The name of my motorcycle is {this.name}</h1>
          <h1>The model of my motorcycle is {this.model}</h1>
          <h1>My motorcycle was bought in the year {this.year}</h1>
      </div>
    )
  }
}

export default Bike
```

**CLASSS ACTIVITY**

1. Pass a Props from a parent component to a child component using both class and functional components method
2. Destructure the Props and display on the browser

# CHAPTER 7

## What is State?

- State is an object that store the value of properties belonging to component that could be change over a period of time
- A state can be modified based on the user action or network changes.
- The state object is initialized in the constructor.
- Every time the state of an object changes, React re-renders the component to the browser
- The state object can store multiple properties
- The this.setState() is used to change the value of the state object.
- setState() method enqueues all the updates made to the component state and instructs React to re-render the component and it's children with the updated state.

## Differences between a Props and a State

| Props | State |
|---|---|
| Props are used to pass data and event handlers to its children components | State is used to store data of the components that has to be rendered to the view |
| Props are immutable – Once set, props cannot be changed | State holds the data and can change over time |
| Props can be used in both class and functional components | State can only be used in class components |
| Props are set by the parent component for the children components | State is generally updated by the event handlers |

**Example:**

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

```
class Car extends Component {

    state = {
        message: 'Subscribe for more news',
        sub: 'Subscribe'
    }
    btn = ()=>{
        this.setState({
            message: 'Thank you for subscribing',
            sub: 'Thank you!'
        })
    }
  render() {
      const mystyle={
          color: 'red',
      }
    return (
      <div>
          <h1 style={mystyle}>{this.state.message}</h1>
          <button onClick={this.btn}>{this.state.sub}</button>
      </div>
    )
  }
}

export default Car
```

## CLASS ACTIVITY

1. What are stateful components?
2. Write a react program that will change the state of “Good morning” to “Good evening”.

# CHAPTER 8

## What is Map?

A map is a data collection type where data is stored in the form of pairs. It contains a unique key. The value stored in the map must be mapped to the key. We cannot store a duplicate pair in the map(). It is because of the uniqueness of each stored key. It is mainly used for fast searching and looking up data.

### The Map() Function

The map() function is used to iterate over an array and manipulate or change data items. In React, the map() function is most commonly used for rendering a list of data to the DOM.
To use the map() function, attach it to an array you want to iterate over. The map() function expects a callback as the argument and executes it once for each element in the array. From the callback parameters, you can access the current element, the current index, and the array itself.
The map() function also takes in an optional second argument, which you can pass to use as this inside the callback. Each time the callback executes, the returned value is then added to a new array.

Example:

```
import React from 'react'

function Bike() {
    const students =['Ernest', 'Seyi', 'Michael', 'Joshua', 'David', 'Busayo', 'Janet']
  return (
    <div>
        <h1>
            {
                students.map((item, idx)=>{
                    return(
                        <div key={idx}>
                            <h4>{item}</h4>
```

```
                        </div>
                    )
                })
            }
        </h1>
    </div>
  )
}

export default Bike
```

Example:

```
const Users = () => {
    const data = [
      { id: 1, name: "John Doe" },
      { id: 2, name: "Victor Wayne" },
      { id: 3, name: "Jane Doe" },
    ];

    return (
      <div className="users">
        {data.map((user) => (
          <div className="user">{user}</div>
        ))}
      </div>
    );
  };
```

## CLASS ACTIVITY

1. Map through the following array:
   **students =['Ernest', 'Seyi', 'Michael', 'Joshua', 'David', 'Busayo', 'Janet']**

# CHAPTER 9

## What are Events in React?

An event is an action that could be triggered as a result of the user action or system generated event. For example, a mouse click, loading of a web page, pressing a key, window resizes, and other interactions are called events.

React has its own event handling system which is very similar to handling events on DOM elements. The react event handling system is known as Synthetic Events. The synthetic event is a cross-browser wrapper of the browser's native event.

Just like HTML DOM events, React can perform actions based on user events.

React has the same events as HTML: click, change, mouseover etc.

## Adding Events

React events are written in camelCase syntax:

onClick instead of onclick.

React event handlers are written inside curly braces:

onClick={shoot} instead of onClick="shoot()".

**In React:**

```
<button onClick={shoot}>Take the Shot!</button>
```

```
<button onClick={showMessage}>
      Hello JavaTpoint
```

```
</button>
```

**In HTML:**

```
<button onclick="shoot()">Take the Shot!</button>
```

```
<button onclick="showMessage()">
      Hello JavaTpoint
</button>
```

## Example:

Put the shoot function inside the Football component:

```
function Football() {
  const shoot = () => {
    alert("Great Shot!");
  }

  return (
    <button onClick={shoot}>Take the shot!</button>
  );
}

ReactDOM.render(<Football />, document.getElementById('root'));
```

## PreventDefault() in React

We must call **preventDefault** event explicitly to prevent the default behavior. For example:

```
import React from 'react'

function Functional() {
  const changeMe = (e)=>{
```

```
    e.preventDefault();
    console.log('You had clicked a Link.');
  }
  return (
    <div>
        <h1>She is pretty</h1>
        <a href = "https://www.facebook.com" onClick={changeMe}>Click Me</a>
    </div>
  )
}

export default Functional
```

## CLASS ACTIVITY

1. Create an button
2. Add an event handler to it, so that it will alert “Hello world”

# CHAPTER 10

## React Hooks?

Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class.

As you know, Functional component is a stateless component, that's why we were using the class components in some previous lesson. With the help of React hooks we can implement all the features of class components like state and life cycle method. That's why we have to use the React hook in the functional component.

### Commonly Used Hooks

1. useState – Similar to setState() in class components. If you want to declare any variable in functional components, you need to use the useState. With the help of useState hook, you can declare a variable and manipulate it.
2. useEffect – This will be executed automatically whenever the functional component is rendered

### Working with Hooks

Whenever you want to use hooks in Functional components, you have to insert the hooks in the curling braces of your React declaration. This import statement is mandatory when you want to use hooks in functional component.

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

### useState Hook

With useState hook you can create a variable and manipulate it. To create any variable in hook you have to write the following syntax:

```
const[name, setName] = useState()
```

Before initializing any useState hook you have to write the "const" keyword follow by the square bracket. In the square bracket you need to write two things, they are the variable name and the function name which is used to change or update the variable name. In the "useState()" hook you place in the values in it. It can also be empty.

In the example below, "name" is the variable name while "setName" is the function that can be used to change or update the "name". "Helen" is the value of the variable called "name".

You can used any name to declare the variable and function parameters.

Example:

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

function Texting() {
    const[name, setName] = useState('Helen')
  return (
    <div>
        <h1>React Hook</h1>
        <h1>Your name is {name}</h1>
    </div>
  )
}

export default Texting
```

You can as well manipulate the variables with the help of the useState hook.

**Example:**

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

function Texting() {
    const[name, setName] = useState('Helen')
```

```
    function changePlayer (){
        setName('Anthony');
    }

  return (
    <div>
        <h1>React Hook</h1>
        <h1>Your name is {name}</h1>
        <button onClick={changePlayer}>Change Name</button>
    </div>
  )
}

export default Texting
```

In the above example the "setName" function parameter was used to change the name from "Helen" to "Anthony" when the user clicked on the button.

**Example:**

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

function Texting() {
    const[numbers, setNumbers] = useState([1,3,5,7,9])


    function changeNumber (){
        setNumbers([2, 4, 6, 8]);
    }

  return (
    <div>
        <h1>React Hook</h1>
        <h1>These are the numbers {numbers}</h1>
        <button onClick={changeNumber}>Change Number</button>
    </div>
  )
}

export default Texting
```

The example above changed the odd numbers to even numbers when the user clicked on the button. The above example also stores the values in an array.

**Example:**

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

function Texting() {
    const[profile, setProfile] = useState({
        name: "John Doe",
        job: "Web Developer",
        company: "Microsoft"
    })


  return (
    <div>
        <h1>My Profile - React Hook</h1>
        <h3>Name: {profile.name}</h3>
        <h3>Job: {profile.job}</h3>
        <h3>Company: {profile.company}</h3>
    </div>
  )
}

export default Texting
```

The above example also stores the values in an object.

**Example:**

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

function Texting() {
    const[profile, setProfile] = useState({
        name: "John Doe",
        job: "Web Developer",
        company: "Microsoft"
    })
```

```
    const updateCompany = ()=>{
        setProfile({...profile, company: "Google"})
    }


  return (
    <div>
        <h1>My Profile - React Hook</h1>
        <h3>Name: {profile.name}</h3>
        <h3>Job: {profile.job}</h3>
        <h3>Company: {profile.company}</h3>
        <button onClick={updateCompany}>Change Company</button>
    </div>
  )
}

export default Texting
```

The above example updates the company's name whenever the user clicked on the button.

## useEffect Hook

The useEffect is the second most commonly used hook after the useState hook. The useEffect hook let you perform side effects in functional components.
There are times you want to trigger an action when your page loads or when the page re-renders or when a state or props changes.

useEffect is what we use to trigger our function (side effect) in this cases.

### Working with useEffect Hook

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

### useEffect syntax

```
    useEffect(()=>{
        console.log("useEffect triggered!");
    })
```

The useEffect hooks takes in a call back function.

**Example:**

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

function Texting() {
    const[count, setCount] = useState(0)

    useEffect(()=>{
        console.log("useEffect triggered!");
    })

  return (
    <div>
        <h1>{count}</h1>
        <button onClick={()=>setCount(count + 1)}>Increment</button>
    </div>
  )
}

export default Texting
```

On every initial render (on first loads) the useEffect hook is triggered automatically.

On every re-render (whenever the browser is refreshed) the useEffect hook is triggered automatically.

When a state or props is changed the useEffect hook is triggered automatically.

**Dependency Array [ ]**

If you want the useEffect hook to run once at initial render you have to insert a dependency array [] as a parameter. This way the useEffect will only run on the first page render.
Syntax:

```
 useEffect(()=>{
        console.log("useEffect triggered!");
    }, [])
```

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

```
function Texting() {
    const[count, setCount] = useState(0)

    useEffect(()=>{
        console.log("useEffect triggered!");
    }, [])

  return (
    <div>
        <h1>{count}</h1>
        <button onClick={()=>setCount(count + 1)}>Increment</button>
    </div>
  )
}

export default Texting
```

When a state or props is changed the useEffect hook is triggered automatically.

**Example**

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

function Texting() {
    const[count, setCount] = useState(0)

    useEffect(()=>{
        console.log("useEffect triggered!");
    }, [count])

  return (
    <div>
        <h1>{count}</h1>
        <button onClick={()=>setCount(count + 1)}>Increment</button>
    </div>
  )
}

export default Texting
```

## CLASS ACTIVITY

1. Create an app that will change the name and job title of a particular employee whenever he or she click on a button
2. Create a counter app and explore some of the useEffect hook concepts and functionalities.
3. Create an app that will update a student's matric number and department whenever he or she clicks on a particular button (note: use object as the hook's value).

# CHAPTER 11

## Manipulating CSS using React Hooks

You can manipulate CSS using React Hooks. React gives the ability to change an element's class name or style attribute.

**Example:**

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

function Texting() {
    const[color, setColor] = useState('red')
    const[bgcolor, setBgcolor] = useState('yellow')

  return (
    <div>
        <h1>Manipulating CSS using React Hooks</h1>
        <h1 style={{color:color}}>I am learning React</h1>
        <h1 style={{backgroundColor: bgcolor}}>I want to be a developer</h1>
    </div>
  )
}

export default Texting
```

**Example:**

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

function Texting() {
    const[color, setColor] = useState('red')
    const[bgcolor, setBgcolor] = useState('yellow')

    const changeStyling = ()=>{
        setColor('green')
        setBgcolor('blue')
    }
```

```
  return (
    <div>
        <h1>Manipulating CSS using React Hooks</h1>
        <h1 style={{color:color}}>I am learning React</h1>
        <h1 style={{backgroundColor: bgcolor}}>I want to be a developer</h1>
        <button onClick={changeStyling}>Change Styling</button>
    </div>
  )
}

export default Texting
```

**CLASS ACTIVITY**

1. Write a React application of your choice.
2. Manipulate the CSS styling with React Hooks

# CHAPTER 12

## Conditionals

Conditional statements are used to perform different actions based on different conditions. The following are the ways to handle conditional statement in React:

1. If else statement operator
2. Logical AND (&&) operator
3. Ternary operator
4. Conditionals with CSS

### IF ELSE STATEMENT

You can use the IF ELSE statement to conditionally render element.

**Example:**
Using the IF ELSE statement to determine if a user is logged in or not.

```
import React, {useState} from "react"

function App() {

const[isLoggedIn, setIsLoggedIn] = useState(false);
  let message;
  if(isLoggedIn){
    message = "Welcome John Doe"
  }
  else{
    message = "Welcome Guest"
  }

  return (
    <div>
```

```
      {message}
    </div>
  );
}

export default App;
```

## LOGICAL AND (&&) OPERATOR

Every statement in front of a logical AND (&&) operator will be display if the logical AND (&&) operator is set to be TRUE.

**Example:**
In the example below, "Welcome Peter" will be displayed on the browser if the logical AND (&&) operator is set to be TRUE.

```
import React, {useState} from "react"

function App() {

const[isLoggedIn, setIsLoggedIn] = useState(true);

  return (
    <div>
      {isLoggedIn && <h1>Welcome Peter</h1>}
    </div>
  );
}

export default App;
```

## TERNARY OPERATOR

The ternary operator works like the "if else" statement but written on a single line.

**Example:**

The example below will display "Welcome John Doe" on the browser if the condition is set to true.

It will display “Welcome Guest” if the condition is set to be false.

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

function App() {
  const [isLoggedIn, setIsLoggedIn] = useState(true);

  return (
    <div>
      {isLoggedIn ? <h1>Welcome John Doe</h1> : <h1>Welcome Guess</h1>}
    </div>
  );
}

export default App;
```

## ADDING CSS CLASSES CONDITIONALLY

You can conditionally add CSS classes to your react element

**Example:**

```
import React, { useState } from "react";
import "./style.css"

function App() {
  const [isLoggedIn, setIsLoggedIn] = useState(true);

  return (
    <div className={isLoggedIn ? "user" : "guess"}>
      {isLoggedIn ? <h1>Welcome John Doe</h1> : <h1>Welcome Guess</h1>}
    </div>
  );
}

export default App;
```

# CLASS ACTIVITY

1. Using a ternary operator, create a conditional statement
2. Add CSS class conditionally to the task above.

# CHAPTER 13

## Forms

Similar to HTML, React uses the form component that helps the users to interact with the web application.

**Example of form:**

```
import "./style.css";
import "../node_modules/bootstrap/dist/css/bootstrap.min.css"

function App() {
  return (
    <div>
      <form>
        <div class="mb-3">
          <label for="exampleFormControlInput1" class="form-label">
            Username
          </label>
          <input
            type="text"
            class="form-control"
            id="exampleFormControlInput1"
            placeholder="username"
          />
        </div>
        <div class="mb-3">
          <label for="exampleFormControlInput1" class="form-label">
            Email address
          </label>
          <input
            type="email"
            class="form-control"
            id="exampleFormControlInput1"
            placeholder="name@example.com"
          />
        </div>
        <button type="submit" className="btn btn-primary">Submit</button>
      </form>
```

```
    </div>
  );
}

export default App;
```

## Submitting Forms

You can control the submit action by adding an event handler in the onSubmit attribute for the <form>:

You can also add a click event to your button for submitting.

To prevent the form behavior (which is refreshing the page) we use the **e.preventDefault().**

## Forms - Control Inputs

In a controlled input, you hook your input to a state.

In a controlled input you read and set the input value through the component's state.

**Example:**

```
import React, { useState } from "react";
import "./style.css";
import "../node_modules/bootstrap/dist/css/bootstrap.min.css"

function App() {
  const[username, setUsername] = useState('')
  const[password, setPassword] = useState('')

  const handleSubmit = (e)=>{
    e.preventDefault()
    console.log(username, password);
    setUsername("")
    setPassword("")
  }
```

```
  return (
    <div>
      <h1>Controlled Inputs</h1>
      <form onSubmit={handleSubmit}>
        <div class="mb-3">
          <label for="exampleFormControlInput1" class="form-label">
            Username
          </label>
          <input
            type="text"
            class="form-control"
            id="exampleFormControlInput1"
            placeholder="username"
            name="username"
            value={username}
            onChange = {(e)=>setUsername(e.target.value)}
          />
        </div>
        <div class="mb-3">
          <label for="exampleFormControlInput1" class="form-label">
            Password
          </label>
          <input
            type="password"
            class="form-control"
            id="exampleFormControlInput1"
            placeholder="name@example.com"
            name="password"
            value={password}
            onChange = {(e)=>setPassword(e.target.value)}
          />
        </div>
        <button type="submit" className="btn btn-primary">Submit</button>
      </form>
    </div>
  );
}

export default App;
```

## Forms - Multiple Control Inputs

When you have multiple fields you can as well make use of the multiple control inputs

Example:

```
import React, { useState } from "react";
import "./style.css";
import "../node_modules/bootstrap/dist/css/bootstrap.min.css"

function App() {
  // const[username, setUsername] = useState('')
  // const[password, setPassword] = useState('')
  const[person, setPerson] =useState({username: "", password: ""})

  const handleInput = (e)=>{
    const name = e.target.name
    const value = e.target.value
    setPerson({...person, [name]:value})
  }

  const handleSubmit = (e)=>{
    e.preventDefault()
    console.log(person);
    setPerson({username: "", password:""})

  }


  return (
    <div>
      <h1>Multiple Controlled Inputs</h1>
      <form onSubmit={handleSubmit}>
        <div class="mb-3">
          <label for="exampleFormControlInput1" class="form-label">
            Username
          </label>
          <input
            type="text"
            class="form-control"
            id="exampleFormControlInput1"
```

```
            placeholder="username"
            name="username"
            value={person.username}
            onChange = {handleInput}
          />
        </div>
        <div class="mb-3">
          <label for="exampleFormControlInput1" class="form-label">
            Password
          </label>
          <input
            type="password"
            class="form-control"
            id="exampleFormControlInput1"
            placeholder="name@example.com"
            name="password"
            value={person.password}
            onChange = {handleInput}
          />
        </div>
        <button type="submit" className="btn btn-primary">Submit</button>
      </form>
    </div>
  );
}

export default App;
```

User's Authentication Form

You can verify the identity of a user attempting to gain access to a server or database by authenticating the user using a form input.

**Example:**

```
import React, { useState } from "react";
import swal from "sweetalert";
import "../node_modules/bootstrap/dist/css/bootstrap.min.css"
```

```
function App() {
  const [username, setUsername] = useState('')
  const [password, setPassword] = useState('')

  const handleSubmit = (e) => {
    e.preventDefault()

    if (username === "" || password === "") {
      swal("Empty fields", "Please fill the required fields", "error")
    }
    else if (username === "Ernest" && password === "123") {
      swal("Login successful", "Congratulations your login is successful", "success")
      setUsername("")
      setPassword("")
    }
    else {
      swal("Invalid login details", "Please check your details once more", "error")
    }

  }

  return (
    <div>
      <h1>Controlled Inputs</h1>
      <form onSubmit={handleSubmit}>
        <div class="mb-3">
          <label for="exampleFormControlInput1" class="form-label">
            Username
          </label>
          <input
            type="text"
            class="form-control"
            id="exampleFormControlInput1"
            placeholder="username"
            name="username"
            value={username}
            onChange={(e) => setUsername(e.target.value)}
          />
        </div>
        <div class="mb-3">
          <label for="exampleFormControlInput1" class="form-label">
            Password
          </label>
          <input
            type="password"
            class="form-control"
```

```
            id="exampleFormControlInput1"
            placeholder="name@example.com"
            name="password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
          />
        </div>
        <button type="submit" className="btn btn-primary">Submit</button>
      </form>
    </div>
  );
}

export default App;
```

**CLASS ACTIVITY**

1. Create a form in react that will accept user's details and display it on the browser.
2. Create a user's authentication form.

# CHAPTER 14

## HTTP METHODS

HTTP methods are used for the communication between client-side server, meaning we can send the data using HTTP methods and we can receive data from an external server using HTTP method. We will discuss about the HTTP methods.

1. **Fetch method:** The fetch () method starts the process of fetching a resource from a server. The fetch () method returns a Promise that resolves to a Response object. The fetch () method takes one mandatory argument, the path to the resource you want to fetch. Fetch () method is asynchronous.

**Example:**

```
import React from 'react'
import "../node_modules/bootstrap/dist/css/bootstrap.min.css"

function App() {

  const getData = async()=>{
    const response = await fetch('https://jsonplaceholder.typicode.com/posts')
    const data = response.json();
    console.log(data);
  }

  return (
    <div>
      <button onClick={getData}>Fetch Api</button>
    </div>
  )
}

export default App
```

2. **Axios method:** Axios makes it easy to send asynchronous HTTP requests to REST endpoints and perform CRUD operations. It can be used in plain JavaScript or with a library such as Vue or React. Axios is not a built-in library. Axios is an external library meaning you have to install before using it.

npm install axios

**Eample:**

```
import axios from 'axios'
import React from 'react'
import "../node_modules/bootstrap/dist/css/bootstrap.min.css"

function App() {

  const getAxiosData = async()=>{
    axios.get('https://jsonplaceholder.typicode.com/posts')
    .then((response)=>{
      console.log(response);
    })
    .catch((error)=>{
      console.log(error);
    })
  }

  return (
    <div>
      <button onClick={getAxiosData}>Axios Api</button>
    </div>
  )
}

export default App
```

## Return HTML Content using HTTP Methods

We are going to return HTML content using the response gotten from the REST API

**Example:**

```
import axios from 'axios'
import React, { useState } from 'react'
import "../node_modules/bootstrap/dist/css/bootstrap.min.css"

function App() {

  const[post, setPost] = useState([])
```

```
  const getAxiosData = async()=>{
    axios.get('https://jsonplaceholder.typicode.com/posts')
    .then((response)=>{
      console.log(response.data);
      setPost(response.data)
    })
    .catch((error)=>{
      console.log(error);
    })
  }

  const blog = post.map((myPost)=>{
    return (
      <div key={myPost.id}>
        <h4>{myPost.id}</h4>
        <h4>{myPost.title}</h4>
        <p>{myPost.body}</p>
      </div>

    )
  })

  return (
    <div>
      <button onClick={getAxiosData}>Axios Api</button>
      {blog}
    </div>
  )
}

export default App
```

**CLASS ACTIVITY**

1. Return HTML content using the response gotten from the REST API
2. Use both Fetch and Axios HTTP method for question 1

# CHAPTER 15

## REACT ROUTER DOM

React Router is a fully-featured client and server-side routing library for React, a JavaScript library for building user interfaces. React Router runs anywhere React runs; on the web, on the server with node.js, and on React Native.

If you're just getting started with React generally, we recommend you follow the excellent Getting Started guide in the official docs. There is plenty of information there to get you up and running. React Router is compatible with React >= 16.8.

Install React Router dependencies:

```
npm install react-router-dom
```

Finally, let's teach React Router how to render our app at different URLs by creating our first "Route Config" inside of `main.jsx` or `index.js`.

```
import { render } from "react-dom";
import {
  BrowserRouter,
  Routes,
  Route,
} from "react-router-dom";
import App from "./App";
import Expenses from "./routes/expenses";
import Invoices from "./routes/invoices";

const rootElement = document.getElementById("root");
render(
  <BrowserRouter>
    <Routes>
      <Route path="/" element={<App />} />
```

```
      <Route path="expenses" element={<Expenses />} />
      <Route path="invoices" element={<Invoices />} />
    </Routes>
  </BrowserRouter>,
  rootElement
);
```