React/ReactJS

==========

It is a declarative, efficient and flexible javascript frontend library responsible for building frontend applications or User interfaces(UI).

It is an open source, component based frontend library responsible only for view layer of the application.

It was developed by Jordan Walke who was a software engineer at Facebook.

It was initially developed and maintained by Facebook and later it is used in their own products like Whatsapp and Instagram.

React was released to the public in the month of May, 2013.

The latest version of React/Reactjs is v18.2.2.

The official website of React is http://www.reactjs.org.

React is used to create a reusable component.

A component is a building block of react application.

Advantages of React/ReactJS

==========

1) It is easy to learn and easy to use.

2) It supports one way data binding.

3) It supports Virtual DOM.

4) It supported by all major browsers.

5) It creates reusable components.

6) Good Documentation and Community support.

Q)Differences between Angular and React?

Angular	React
It was released in October,2010.	It was released in May, 2013.
Angular is a product of Google.	React is a product of Facebook.
Angular is an open source javascript framework for web and mobile development.	React is a open source frontend javascript library responsible only for view layer of the application.
Typescript language is used in angular.	JSX language is used in react.
Jasmine and Karma used as a testing frameworks.	Jest and Enzyme used as a testing frameworks.
It supports two way data binding.	It supports one way data binding.
It supports Traditional DOM.	It supports Virtual DOM.
The default port number is 4200.	The default port number is 3000.

It is used for large scale and rich featured applications.	It is used for SPA (Single Page Application).
Angular used by Google, Nike, McDonoalds, paypal,Gmail and etc.	React used by Facebook,whatsapp, instagram,airbnb and etc.

How ReactJS works internally

==========

React uses a virtual DOM that is basically a DOM tree representation in JavaScript.

So when it needs to read or write to the DOM, it will use the virtual representation of it.Then the virtual DOM will try to find the most efficient way to update the browser's DOM.

Assume we have created multiple components and consistently we are performing some changes in our application.Now we need to see ,how virtual DOM react on each change.

Diagram: react1.1

Pre-requiste to learn React/ReactJS

==========

1) Basics of HTML,CSS,Javascript and Bootstrap.

2) Strong knowledge on JSX.

3) Usage of npm commands.

4) Basics on ES6 standards.

What is JSX

==========

JSX stands for JavaScript XML.

JSX allows us to write HTML in React.

JSX tags having a tagname, attributes and children.

JSX is not a necessity to write React applications.Instead we can use Babel.

JSX makes your react code simpler and elegant.

JSX ultimately transpiles to pure JavaScript which is understood by the browsers.

JSX Elements

==========

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

ex:1

JSX code

```
<h1>IHUB Talent</h1>
```

Here h1 is a jsx element.

Babel code

```
React.createElement("h1",null,"IHUB Talent");
```

Here h1 is a tag name.

Here null is a optional property name.

Here IHUB Talent is a text.

ex:2

JSX code

```
<div>
    <h1>Hello React JS </h1>
</div>
```

babel code

```
React.createElement("div",null,
                    React.creteElement("h1",null,"Hello ReactJs"));
```

ex:3

JSX code

```
<div id="myId">
    <h1>Hello React JS </h1>
</div>
```

babel code

```
React.createElement("div",{id:'myId'},
                    React.creteElement("h1",null,"Hello ReactJs"));
```

ex:4

JSX code

```
<div class="myClass">
    <h1>Hello React JS </h1>
</div>
```

babel code

```
React.createElement("div",{class:'myClass'},
                    React.creteElement("h1",null,"Hello ReactJs"));
```

Note:

In above code , warning message will be displayed on console i.e

Invalid DOM property 'class'.

In order to remove this warning from our application we need to use "className" attribute.

In javascript, "class" is a keyword which is used to create React components.

In react , CSS class name must specify by "className" attribute.

ex:5

JSX code

```
<div id="myId" class="myClass">
    <h1>Hello React JS </h1>
</div>
```

babel code

```
React.createElement("div",{id:'myId',className:'myClass'},
                    React.creteElement("h1",null,"Hello ReactJs"));
```

JSX Expressions

==========

JSX allows us to 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.

ex:1

```
let name="Alan Morries";

<h1>My Name is {name}</h1>
```

ex:2

```
<h1>The value is = {5+5} </h1>
```

ex:3

```
<h1>{Math.random()*100}</h1>
```

ex:4

```
<h1>{Math.floor(Math.random()*100)}</h1>
```

Note:

---

we can't use conditional statements like if, while and etc inside JSX expression.

NPM

=====

NPM stands for Node Package Manager.

It is integrated tool for NodeJs.

It is used to download node modules/dependencies/libraries.

We can download any mododule as follow.

ex:

cmd> npm install -g node_module/dependency/library

All the modules will downloaded inside "node_modules" folder.

Setup for npm command

---

1) Download and Install NodeJS module.

ex:

https://nodejs.org/en/download

2) Copy the nodejs directory from "C/program files" drive.

ex:

C:\Program Files\nodejs

3) Paste nodejs directory in environment variables.

ex:

right click to my computer --> properties --> advanced system settings -->

environmental variables --> user variables --> click to new button -->

variable name : path

variable value : C:\Program Files\nodejs; -->ok -->ok -->ok.

4) Open the command prompt and check below commands.

ex:

cmd> npm -version

cmd> node --version

First application development using React/ReactJS

==========

step1:

Make sure Nodejs setup has done perfectly.

step2:

Download and Install VSC(Visual Studio Code) editor.

ex:

https://code.visualstudio.com/

step3:

Create a "Reactprojects" folder inside "E" drive.

step4:

Open the command prompt from "Reactprojects" folder.

step5:

Open the visual studio code editor from "Reactprojects" folder.

ex:

Reactprojects> code .

step6:

Install "create-react-app" module for creating react applications.

ex:

Reactprojects> npm install -g create-react-app

step7:

Create a "myapp1" react project in VSC editor.

ex:

Reactprojects> create-react-app myapp1

step8:

Switch to myapp1 project.

ex:

Reactprojects>cd myapp1

step9:

Run the myapp1 project.

ex:

Reactprojects/myapp1> npm start

step10:

Test the react application/project.

ex:

http://localhost:3000

Note

By default react application runs on a light weight development server with 3000 port number.

Explaination of React project and Work flow

==========

myapp1

|

|----node_modules

|

|----public

| |

| |---favicon.ico

| |---index.html

| |---manifest.json

| |

|

|-----src

| |

| |---App.css

| |---App.js

| |---App.test.js

| |---index.css

| |---index.js

| |---logo.svg

| |

|---package.json

|---README.md

"myapp1" is a Name of a project.

"node_modules" contains all packages and dependencies installed.

"favicon.ico" is a favourite icon for a web site.

"index.html" file holds HTML template of our application(Main template).

"manifest.json" file provides metadata used when your web app is
installed on a user's mobile device or desktop.

"App.css" is a css file related to App.js but it's global.

"App.js" is parent component of our React app.

"App.test.js" is for test environment.

"index.css" is a css file related to index.js but it's global.

"index.js" is a javascript entry point.

"logo.svg" is a React logo.

"package.json" contains all dependencies used in React app along with their versions..

Note:

To build the project, "index.js" and "index.html" must exist
with exact file name(mandatory).

Above two files are mandatory at the time of deployment
not at the time of development.

load to render to output

App.js--------->index.js---------->index.html------->browser

Second Application development using React/ReactJS

==========

myapp2

|

|---node_modules

|

|

|-----public

|

|---favicon.ico

|---index.html (3)

|---manifest.json

|

|-----src

|

|---index.js (2)

|

|---App.js (1)

|

|-----package.json

|-----README.md

step1:

Create a myapp2 react application/project.

ex:

Reactprojects> create-react-app myapp2

or

Reactprojects> npx create-react-app myapp2

step2:

Open the VSC editor from Reactprojects folder.

ex:

Reactprojects> code .

step3:

Delete App.css,App.js,App.test.js,index.js and index.css file

from src folder.

step4:

Create index.js file inside "src" folder.

ex:

index.js

```
import App from "./App";
import React from "react";
import ReactDOM from "react-dom/client";

const root=ReactDOM.createRoot(document.getElementById('root'));

root.render(
    <React.StrictMode>
<App/>
    </React.StrictMode>
    )
```

step5:

Create App.js file inside "src" folder.

ex:

App.js

```
function App()
{
    return(

        <h1>I Love ReactJS </h1>

    )
}
export default App;
```

step6:

Switch to myapp2 project.

ex:

```
Reactprojects> cd myapp2
```

step7:

Run the myapp2 project.

ex:

```
Reactprojects/myapp2> npm start
```

step8:

Test the application by using below request url.

ex:

http://localhost:3000

React Fragment

==========

Fragment is used to group of list of childrens without adding

extra nodes of the DOM.

In general, We can return

only one element at a time but we can't return more then one element directly.

To return more then one element we need to use React Fragment.

syntax

<React.Fragment>

-

-

</React.Fragment

or

<>

-

-

</>

Examples

App.js

```
function App
{
   return (
      //return react element
      return  <h1>IHUB Talent</h1>
            <h2>React Tutorial For Freshers</h2>
   );
}
//export React component
export default App
```

o/p: Filed to compile

To overcome above problem we can use tag and inside that

 tag we can declare any child tags.

ex:

App.js

```
function App
{
   return (
      //return react element
```

```
    return
                <div>
                        <h1>IHUB Talent</h1>
                <h2>React Tutorial For Freshers</h2>
                </div>
  );
}
//export React component
export default App
```

Note:

In above program " " tag is a unused tag.

To remove unused/unnecessary tags we can use React Fragment.

approach1

App.js

```
import React from "react";

function App()
{
  return (
      <React.Fragment>
      <h1>IHUB React Tutorial</h1>
      <h1>React Classes for Freshers</h1>
      </React.Fragment>
```

```
    );

}
export default App;
```

approach2

App.js

```
import React from "react";
import {Fragment} from 'react';
function App()
{
   return (
         <Fragment>
         <h1>IHUB React Tutorial</h1>
         <h1>React Classes for Freshers</h1>
         </Fragment>
   );

}
export default App;
```

approach3

App.js

```
import React from "react";

function App()
{
   return (
        <>
        <h1>IHUB React Tutorial</h1>
        <h1>React Classes for Freshers</h1>
        </>
   );

}
export default App;
```

React Components

==========

Components are Building blocks of any React app.

Component allows us to split UI into independent reusable pieces.

ex:

navbar, header, footer , body and etc.

Components are like Javascript functions.They accept arbitrary inputs called "props" and return React Element describing what should appears on the screen.

A Component name always starts with capital letter.

ex:

 represent as HTML div tag.

But <Div> represent a component in react.

we can create react component in two ways.

1)Function Component /functional component

2)Class Component

1)Function Component

==========

It is a Javascript function which accept single "props" object as argument

with data and returns a React Element.

The functional component is also known as a stateless component because they do not hold or manage state.

syntax

```
function function_name()
{

    return element;
}
```

Project Directory structure

==========

myapp3

|

```
|----node_modules
|
|----public
|          |
|          |---favicon.ico
|          |---index.html (3)
|          |---manifest.json
|          |
|
|-----src
|          |
|          |---index.js   (2)
|          |---App.js (1)
|          |
|---package.json
|---README.md
```

step1:

Develop React Application

ex:

E:/BUI-2pm/ReactProjects>create-react-app myapp3

step2:

Delete all the starting 6 files from src folder.

step3:

---

create a App.js file in src folder to develop a function component.

Student.js

```
function App()
{
  return <h1>Function Component Example</h1>
}
export default App;
```

Note:

Above code is applicable for older versions and for lastest versions like EC6 ,we use below code.

ex:

App.js

```
const App=()=>{
   return <h1>Function Component Example</h1>
}
export default App;
```

step4:

---

create "index.js" file to render the component to index.html file.

index.js

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

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

step5:

---

move to myapp3

ex:

cmd/ReactProjects>cd myapp3

step6:

Run the Application

ex:

cmd/myapp3> npm start

step7:

Test the Application.

ex:

http://localhost:3000

Function component with props

==========

In order to use props in a component We need to perform following changes in react "myapp3" project.

syntax

```
function fun_name(props)
{
        return React Element
}
```

ex:1

App.js

```
function App(props)
{
  return <h1>Hello {props.name}</h1>
}

export default App;
```

or

App.js

```
const Student=(props)=>
{
  return <h1>Hello {props.name}</h1>
```

```
}

export default App;
```

Rendering the Component

---

we can render the component in index.js file as given below.

index.js

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

const root=ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <React.StrickMode>
    <App name="Alan"/>
    </React.StrickMode>
)
```

ex2:

App.js

```
const App=(props)=>
{
  return (
        <>
        <h1>First Name : {props.firstName}</h1>
        <h1>Second Name: {props.lastName}</h1>
        </>
  )
}
export default App;
```

index.js

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

const root=ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <React.StrickMode>
          <App firstName="Alan" lastName="Morries"/>
    </React.StrickMode>
)
```

2)Class Component

==========

A class Component requires to extends from React Component.

The class must implements a render() method function which returns A react

Element to be render.This is Similar to return value of a functional

component.

In a class based component props are accessible via this.props.

The class component is also known as a stateful component because they can hold or manage local state.

syntax

```
class Class_name extends Component
{
        render()
        {
                return element.
        }
}
```

Project structure

```
myapp4
|
|------node-modules
|
|------public
|       |
|       |------favicon.ico
|       |------index.html (3)
|       |------manifest.json
```

|

|------src

|

|------index.js(2)

|------App.js (1)

|

|

|------package.json

|------README.md

step1:

Develop React Application.

ex:

cmd/ReactProjects>create-react-app myapp4

step2:

Delete all the files from src folder.

step3:

------

create a "App.js" file in "src " folder.

App.js

```
import { Component } from "react";
export default class App extends Component
{
  render()
  {
```

```
   return <h1>First Class Component</h1>
  }
}
```

step4:

---

create "index.js" file to render the output to index.html file.

index.js

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

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

step5:

---

move to myapp4

ex:

cmd/ReactProjects> cd myapp4

step6:

Run the application.

ex:

cmd/ReactProjects/myapp4>npm start

step7:

Test the React Application.

ex:

http://localhost:3000

Class component with props

==========

In order to use props in a class component we need to perform following changes.

App.js

```
import { Component } from "react";
export default class App extends Component
{
  render()
  {
    return <h1>Name : {this.props.name}</h1>
  }
}
```

index.js

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

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

)
```

Composing Components in React

==========

A component can refer to other components in their output is called composing component.

Let us use some component abstraction for any level of details.

Project structure

```
myapp4
|
|-----node_modules
|
|-----public
|          |
           |---index.html (main template)
           |---favicon.ico (favicon)
```

```
        |---manifest.json (metadata)
|
|-----src
|       |
        |---index.js (entry point)
        |
        |
        |---App.js  (parent component)
        |
        |---Student.js (custom component)

|
|-----package.json
|-----README.md
```

step1:

Create a React Application.

ex:

ReactProjects> create-react-app myapp4

step2:

Start Visual Studio Code (VSC) Editor.

ex:

ReactProjects> code .

step3:

Delete all the files from "src" folder.

step4:

Create "index.js" file inside "src" folder.

index.js

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";

const root=ReactDOM.createRoot(document.getElementById('root'));

root.render(
  <React.StrictMode>
    <App/>
  </React.StrictMode>
)
```

step5:

Create App.js file inside "src" folder.

App.js

```
import Student from './Student';

function App()
{
  return (
    <Student/>
```

```
  )
}
export default App;
```

step6:

Create Student.js file inside "src" folder.

Student.js

```
function Student()
{
  return (
    <h1>Student Component</h1>
  )
}
export default Student;
```

step7:

Move to myapp4.

ex:

ReactProjects> cd myapp4

step8:

Run the react application.

ex:

ReactProjects/myapp4> npm start

step9:

Check the output by using below url.

ex:

http://localhost:3000

composing components using props

==========

index.js

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";

const root=ReactDOM.createRoot(document.getElementById('root'));

root.render(
  <React.StrictMode>
    <App course="React"/>
  </React.StrictMode>
)
```

App.js

```
import Student from './Student';

function App(props)
{
  return (
    <Student crs={props.course}/>
```

```
    )
}
export default App;
```

Student.js

```
function Student(props)
{
    return (
        <h1>My Course Name : {props.crs}</h1>
    )
}
export default Student;
```

React CSS

==========

CSS in React is used to style the React App or Component.

There are three ways available to add styling to your React App or Component with CSS.

1) Inline Styling

2) CSS Stylesheet

3) CSS Module

1)Inline CSS

==========

Inline CSS represent by "style" attribute in React application.

The inline styles are specified with a JavaScript object in camelCase version of the style name.

ex:

App.js

```
import Student from "./Student";

function App()
{
  return <>
        <h1 style={{color:"green"}}>React Inline CSS</h1>
        <h1 style={{backgroundColor:"yellow"}}>React Inline CSS</h1>
         </>
}
export default App;
```

The inline styling also allows us to create an object with styling information and refer it in the style attribute.

App.js

```
import Student from "./Student";

function App()
{
  const mystyle = {
    color: "white",
    backgroundColor: "DodgerBlue",
```

```
   padding: "10px",
   fontFamily: "Arial"
  };
  return <>
        <h1 style={mystyle}>React Inline CSS</h1>
        <h1 style={{backgroundColor:"yellow"}}>React Inline CSS</h1>
         </>
}
export default App;
```

2) CSS Stylesheet

==========

We can write styling in a separate file for your React application, and save the file with a .css extension.

Later we can import .css file in our required application.

ex:1

App.js

```
import Student from "./Student";
import './App.css';
function App()
{

  return <>
        <h1>React CSS styles</h1>
```

```
        <h1>React CSS styles</h1>
        </>
}
export default App;
```

App.css

```
body{
  background-color: yellow;
}
h1
{
  color:blue;
}
```

ex:2

App.js

```
import Student from "./Student";
import './App.css';
function App()
{

  return <>
        <h1 id="myId">React CSS styles</h1>
        <h1 className="myClass">React CSS styles</h1>
```

```
        </>
}
export default App;
```

App.css

```
body{
  background-color: yellow;
}
#myId
{
  color:blue;
}
.myClass
{
  color:red;
}
```

3. CSS Module

==========

CSS Module is another way of adding styles to your application.

It is a CSS file where all class names and animation names are scoped locally by default.

It is available only for the component which imports it, means any styling you add can never be applied to other components without your permission, and you never need to worry about name conflicts.

We can create CSS Module with the .module.css extension like a myStyles.module.css.

ex:

App.js

```
import Student from "./Student";
import styles from './mystyles.module.css';
function App()
{

  return <>
        <h1  className={styles.mystyle}>React CSS styles</h1>
        <h1 className={styles.parastyle}>React CSS styles</h1>
         </>
}
export default App;
```

mystyles.module.css

```
.mystyle {
    background-color: #cdc0b0;
    color: Red;
    padding: 10px;
    font-family: Arial;
    text-align: center;
  }

  .parastyle{
    color: Green;
    font-family: Arial;
    font-size: 35px;
    text-align: center;
```

}

State

=======

State is similar to props but it is a private and fully controlled by the component.

we can create a state only in class component but not in functional component.

It is possible to update the state or modify the state , where as props only for read only.

There are two ways to initialize the state in React component.

1)Directly inside class

2)Inside the Constructor

1)Directly inside class

==========

```
class Student extends Component
{
        //define state
        state={
                name: "Anna Julie",
                prop1: this.props.prop1
        }
        render()
```

```
        {
                -
        }
}
```

Note:

The "state" property is refered as state.

"this" is a class instance property

example

Project structure

---

myapp5

```
|
|------node-modules
|
|------public
|       |
|       |------favicon.ico
|       |------index.html
|       |------manifest.json
|
|------src
        |
        |------index.js
        |------App.js
|
|------package.json
|------README.md
```

step1:

Develop React Application.

ex:

E:/BUI-2pm/ReactProjects>create-react-app myapp5

step2:

Delete all starting 6 files from src folder.

step3:

Install "ES7 React " Plugin/Extension from Visual Studio Code

for shortcuts to create React Applications.

ex:

imr +tab

imrc + tab

imrd + tab

imp + tab

rcc - class component

rcfe - named function component

rafce - anonymous function component

conlg+ tab

step4:

---

create a "App.js" file in "src " folder (rcc).

Student.js

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

export default class App extends Component {
                        state={
                                name:"Alan"
                        }
  render() {
            return (
              <h1>Hello {this.state.name}</h1>
            )
  }
}
```

step:5

---

create "index.js" file to render the output to index.html file.

(imr, imrd , imp)

index.js

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

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

step6:

---

move to myapp7

ex:

E:/BUI-2pm/ReactProjects> cd myapp5

step7:

Run the application.

ex:

DE:/BUI-2pm/ReactProjects/myapp5>npm start

step8:

Test the React Application.

ex:

http://localhost:3000

ex:2

App.js

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

export default class App extends Component {
        state={
                name:"Alan",
                roll:this.props.rollno
```

```
                    }
 render() {
          return (
           <div>
                    <h1>Name: {this.state.name}</h1>
                    <h1>RollNo: {this.state.roll}</h1>
            </div>
          )
 }
}
```

index.js

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

const root=ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <React.StrictMode>
        <App rollno={501} />
    </React.StrictMode>
)
```

Note:

Here props property we are storing into a state.

2)Inside the Constructor

==========

```
class App extends Component
{
	//constructor
	//props is optional
	constructor(props)
	{
		//it is required to call the parent class constructor
		super(props);

		//state
		this.state={
			name:"alan",
			prop1: this.props.prop1
		}
	}
	render()
	{
		-
	}
}
```

When the component class is created,The constructor is the first called

so it is right place to add state.

The class instance has already been created in memory .So we can use "this" to set properties on it.

When we write a constructor ,make sure to call parent class constructor by using super(props) keyword.

When we call super with props ,React will make props avaiable accross/access the component through this.props.

Project structure

```
myapp6
|
|------node-modules
|
|------public
|          |
|          |------favicon.ico
|          |------index.html
|          |------manifest.json
|
|------src
           |
           |------index.js
           |------App.js
|
|------package.json
|------README.md
```

step1:

Develop React Application.

ex:

E:/BUI-2pm/ReactProjects>create-react-app myapp6

step2:

Delete all starting 6 files from src folder.

step3:

---

create a "App.js" file in "src " folder (rcc).

Student.js

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

export default class App extends Component {

        constructor()
        {
                super();

                this.state={
                        name: "Alan",
                        roll: 101
                }
        }
  render() {
        return (
          <div>
```

```
                    <h1>Name: {this.state.name}</h1>
                    <h1>RollNo: {this.state.roll}</h1>
                </div>
            )
    }
}
```

step:4

---

create "index.js" file to render the output to index.html file.

(imr, imrd , imp)

index.js

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

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

step5:

---

move to myapp6

ex:

E:/BUI-2pm/ReactProjects> cd myapp6

step6:

Run the application.

ex:

DE:/BUI-2pm/ReactProjects/myapp6>npm start

step7:

Test the React Application.

ex:

http://localhost:3000

ex:2

App.js

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

export default class App extends Component {

    constructor(props)
    {
        super(props);

        this.state={
            name: "Alan",
            roll: this.props.rollno
```

```
                }
        }
 render() {
        return (
          <div>
                <h1>Name: {this.state.name}</h1>
                <h1>RollNo: {this.state.roll}</h1>
           </div>
        )
 }
}
```

index.js

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

const root=ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <React.StrictMode>
        <App rollno={501} />
    </React.StrictMode>


)
```

Q)Differences between ReactJS and React Native ?

ReactJS	React Native
==========	==========
The ReactJS initial release was in 2013.	The React Native initial release was in 2015.
It is used for developing web applications.	It is used for developing mobile applications.
It can be executed on all platforms.	It is not platform independent.It takes more efforts to be executed on all platforms.
It uses a JavaScript library and CSS for animations.	It comes with built-in animation libraries.
It uses React-router for navigating web pages.	It has built-in Navigator library for navigating mobile applications.
It uses HTML tags.	It does not use HTML tags.
It provides high security.	It provides low security in comparison to ReactJS.
In this, the Virtual DOM renders the browser applications.	In this, Native uses its API to render code for mobile code.

Event Handling in React

==========

Event

Action to which a javascript can respond is called event.

ex:

clicking on button

hovering of an element

and etc.

Handling events on react Elements are same like handling events on DOM elements.

ex:

Javascript

<button onclick="f1()">clickMe</button

React

<button onClick={handleClick}>clickMe</button> --> Function component

<button onClick={this.handleClick}>clickMe</button> --> Class component

Eventing Handling using Function component

==========

Project structure

---

myapp7

|

|----node_modules

|

|----public

|

|---index.html

|---favicon.ico

|---manifest.json

|

|-----src

|

|---index.js

|---index.css

|

|---App.js

|---App.css

|---App.test.js

|

|----package.json

|

step1:

-----

create a react project/application.

ex:

ReactProjects> create-react-app myapp7

step2:

Starts VSC code editor.

ex:

ReactProjects> code .

step3:

Move to the project.

ex:

ReactProjects> cd myapp7

step4:

Run the react application/project.

ex:

ReactProjects/myapp7> npm start

ex:1

App.js

```
function App()
{

  function handleClick()
  {
    console.log("Button is clicked");
  }


 return (
    <button onClick={handleClick}>clickMe</button>
```

```
 )
}
export default App;
```

ex:2

App.js

```
function App()
{

  const handleClick=()=>
  {
    console.log("Button is clicked");
  }


 return (
    <button onClick={handleClick}>clickMe</button>
 )
}
export default App;
```

ex:3

App.js

```
function App()
```

```
{

  const handleClick=()=>
  {
    console.log("Link is clicked");
  }


 return (
   <a href="http://www.google.com" onClick={handleClick}> clickMe </a>
 )
}
export default App;
```

ex:4

App.js

```
function App()
{

  const handleClick=(e)=>
  {
    e.preventDefault();
    console.log("Link is clicked");
  }


 return (
   <a href="http://www.google.com" onClick={handleClick}> clickMe </a>
```

```
 )
}
export default App;
```

Eventing Handling using class component

==========

Project structure

---

myapp8

|

|----node_modules

|

|----public

|

|---index.html

|---favicon.ico

|---manifest.json

|

|-----src

|

|---index.js

|---index.css

|

|---App.js

|---App.css

|---App.test.js

|

|----package.json

|

step1:

---

create a react project/application.

ex:

ReactProjects> create-react-app myapp8

step2:

Starts VSC code editor.

ex:

ReactProjects> code .

step3:

Move to the project.

ex:

ReactProjects> cd myapp8

step4:

Run the react application/project.

ex:

ReactProjects/myapp8> npm start

ex:1

App.js

```
import {Component} from "react";
export default class App extends Component
{


   handleClick=()=>
   {
      console.log("Button is clicked",this);
   }


   render()
   {
      return(
        <button  onClick={this.handleClick}>clickMe</button>
      )
   }
}
```

index.js

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


const root = ReactDOM.createRoot(document.getElementById('root'));
```

```
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

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

update state

==========

Using setState() method is used to update states.

ex:

```
this.state={
        name:"Alan"
}

this.setState({name:"Kelvin"});
```

ex:1

App.js

```
import {Component} from "react";
```

```
export default class App extends Component
{
    state={
       name : "Nancy",
       rollno: 101
    }

    handleClick=()=>
    {
     this.setState({name:"Lisa",rollno:501});
    }
  render()
  {
     return(
      <>
        <h1>Name : {this.state.name}</h1>
        <h1>RollNo : {this.state.rollno}</h1>

        <button  onClick={this.handleClick}>Change state</button>
      </>
     )
  }
}
```

Q)Difference between props and state ?

props	state
Props are read-only.	States are updatable.

Props are immutable.	State is mutable.
Props allow us to pass data from one component to other components as an argument.	State holds information about the components.
Props can be accessed by the child component.	State cannot be accessed by child components because it is private.
Stateless component can have Props.	Statefull components can have state.

Phases of components in ReactJS

==========

There are four Phases of components in ReactJS.

1)Mounting

2)Updating

3)Error Handling

4)Unmounting

1)Mounting

Mounting is a process of creating an element and inserting it in a DOM tree.

2)Updating

Updating is a process of changing state or props of a component and update changes to nodes already existing in the DOM.

3)Error Handling

Error Handling used when there is a error during rendering ,in lifecycle

method or in the constructor of any child component.

4)Unmounting

Unmounting is a process of removing components from the DOM.

In general it will clear the reserved memory.

Q)Explain life cycle methods of mounting ?

Mounting phase contains four methods.

1) constructor

2) getDerivedStateFromProps

3) render()

4) coumponentDidMount()

Q)Explain life cycle methods of unmounting?

Unmounting phase contains one method.

1) componentWillUnmount()

Q)Explain life cycle methods of updating?

updating phase contains five methods.

1) getDerivedStateFromProps

2) shouldComponentUpdate()

3) render()

4) getSnapshotBeforeUpdate()

4) ComponentDidUpdate()

Diagram: react5.1

Hooks

==========

Hooks allow us to "hook" into React features such as state and lifecycle methods.

Hooks allow function components to have access to state , lifecycle methods and other React features.

Hooks allow us to use React without classes.It means you can use state and other React features without writing a class.

React provides a few built-In hooks like useState,useEffect and etc.

Hooks are new addition in React 16.8.

When use Hooks

If you write a function component and relize you need to add some state to it.

Rules of Hooks

==========

There are 3 rules for hooks:

1)Hooks can only be called inside React function components.

2)Hooks can only be called at the top level of a component.

3)Hooks cannot be conditional

Note: Hooks will not work in React class components.

Declaring State

==========

A useState() is a Hook that allows us to add React state to function components.

We call it inside a function component to add some local state to it.

A useState() returns a pair - the current state value and a function that let us update it.

React will preserve this state between re-renders.

We can call this function from an event handler or somewhere else.

ex:

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

useState("Alan");
```

or

```
const nameStateVariable=useState("Alan");
```

or

```
const [name,setName]=useState("Alan"); // it is destructure the array.
```

When we declare a state variable with useState, it returns a pair-an array with two items.So by writing square bracket we are doing array Destructuring.

App.js

```
import { useState } from "react";
function App()
{
   const [name,setName]=useState("Alan");


   const handleClick=()=>
   {
        setName("Kelvin");
   }
   return (
      <div>
         <h1>Name : {name}</h1>
         <button onClick={handleClick}>clickMe</button>
      </div>
```

```
    )
}
export default App;
```

index.js

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

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

Effect Hooks

==========

The Effect Hook let us to perform side effects in function components.

Data fetching , setting up a subscription, and manually changing the

DOM in React components are all examples of side effects.

useEffect()

==========

A useEffect is a hook for encapsulating code that has "side effects".

Uf we are familiar with React class life cycle methods.We can thing of

useEffect Hooks as componentDidMount,compnoentDidUpdate and

componentWillUnmount combined.

useEffect =componentDidMount+ componentDidUpdate +componentWillUnmount

ex:

```
import React,{useEffect} from "react";
useEffect(Function)
```

or

```
useEffect(Function ,Array)
```

The function passes to useEffect will run after the render is committed

to the screen.

Second argument to useEffect that is the array of values that the

effect depends on.(It is for condition purpose).

Note:

We can call useEffect as many times we required.

ex:

```
useEffect(()=>
{
        console.log("Hello useeffect");
});
```

ex:

```
useEffect(()=>
```

```
{
        console.log("Hello useEffect");
},[count]);
```

What does useEffect do?

By using this Hook,we can tell react that your component needs to do something after render.

React remember the function we passed and call it later after performing the DOM updates.

In this effect, we set the document title,we could also perform data fetching or call some other imperative API.

Note:

---

useEffect runs after the first render and after every update.

ex:

App.js

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

   const handleClick=()=>
   {
```

```
        setCount(count+1);
    }

    useEffect(() => {
       // Update the document title using the browser API
       document.title = count;
     });
    return (
       <div>
          <h1>You clicked  {count} Times</h1>
          <button onClick={handleClick}>clickMe</button>
       </div>
    )
}
export default App;
```

index.js

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

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

React useContext Hook (Context API)

==========

Context provides a way to pass the data through the component tree without

passing props down manually at several level.

To do this without Context, we will need to pass the state(useState) as "props" through each nested component. This is called "props drilling".

Diagram: react5.2

Project structure

---

myapp10

|

|------node-modules

|

|------public

| |

| |------favicon.ico

| |------index.html

| |------manifest.json

|

|------src

  |

  |------index.js

  |------App.js

  |------Acomponent.js

  |------Bcomponent.js

  |------Ccomponent.js

|

```
|------package.json
|------README.md
```

App.js

```
import React from 'react';
import Acomponent from "./Acomponent";
export const UseContext=React.createContext();
function App()
{

   return (
      <div>
         <UseContext.Provider value={'IHUB'}>
         <Acomponent/>
         </UseContext.Provider>

      </div>
   )
}
export default App;
```

Acomponent.js

```
import Bcomponent from "./Bcomponent";

function Acomponent()
{
   return (
      <Bcomponent/>
```

```
   )
}
export default  Acomponent;
```

Bcomponent.js

```
import Ccomponent from "./Ccomponent";

function Bcomponent()
{
   return (
     <Ccomponent/>
   )
}
export default  Bcomponent;
```

Ccomponent.js

```
import {useContext} from "./App";

function Ccomponent()
{
   return (
   <div>
    <UseContext.Consumer>
       {
          user => {
             return <div>The value is : {user} </div>
          }
```

```
        }
    </UseContext.Consumer>
   </div>
   )
}
export default  Ccomponent;
```

index.js

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

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

Custom Hooks

==========

Hooks which are created by the user based on the application requirement are called custom hooks.

ex:

```
myCustomHook()
customHook()
```

ihubHook()

myCustomCounter()

Project Structure

---

myapp11

|

|----node_modules

|

|----public

| |

|----favicon.ico

|----index.html

|----manifest.json

|

|-----src

| |

|----index.js

|----App.js

|----CustomHook.js

|

|-----package.json

|-----README.md

ex:1

CustomHook.js

---

import React from 'react'

```
import {useState} from 'react'



function CustomHook()
{

  const [count,setCount]=useState(0);

  const handleClick=()=>
  {
    setCount(count+1);
  }

    return(
       {
       count,
       handleClick
       })
}

export default CustomHook
```

App.js

```
import React from 'react'
import customHook from './CustomHook';

function App() {

  const data=customHook();
```

```
 return (
   <div>
     <h1>Count : {data.count}</h1>
     <button onClick={data.handleClick}>Increment</button>
   </div>
 )
}

export default App
```

index.js

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

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

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

React Router

==========

Routing is a process in which a user is directed to different pages based on their action or request.

ReactJS Router is mainly used for developing Single Page Web Applications.

React Router is used to define multiple routes in the application.

When a user types a specific URL into the browser, and if this URL path matches any 'route' inside the router file, the user will be redirected to that particular route.

React Router is a standard library system built on top of the React and used to create routing in the React application using React Router Package.

React contains three different packages for routing.

1)react-router:

It provides the core routing components and functions for the React Router applications.

2)react-router-native:

It is used for mobile applications.

3)react-router-dom:

It is used for web applications design.

Note:

It is not possible to install react-router directly in your application.

To use react routing, first, you need to install react-router-dom modules in your application.

We have two types of router components.

1)<BrowserRouter>:

It is used for handling the dynamic URL.

2)<HashRouter>:

It is used for handling the static request.

Project structure

---

myapp12

|

|------node-modules

|

|------public

| |

| |------favicon.ico

| |------index.html

| |------manifest.json

|

|------src

  |

|------index.js

|------App.js (Routing File)

|------Home.js

|------About.js

|------Contact.js

|------Error.js

|

|------package.json

|------README.md

step1:

------

create react "myapp12" project in VSC.

ex:

projects>create-react-app myapp12

step2:

Move to myapp12 project.

ex:

project> cd   myapp12

step3:

------

install react router dom.

ex:

project/myapp12>npm install --save react-router-dom

step4:

Restart the application .

ex:

myapp14> npm start

step5:

---

create App.js,Home.js,About.js ,Contact.js and Error.js component inside "src" folder.

App.js

```
import Home from './Home';
import Contact from './Contact';
import About from './About';
import Error from './Error'
import { BrowserRouter, Routes, Route } from "react-router-dom";

function App() {
  return (
    <div>
      <BrowserRouter>
     <Routes>
       <Route exact path="/" element={<Home />}/>
       <Route path="/about" element={<About />}/>
       <Route path="/contact" element={<Contact />}/>
       <Route path="*" element={<Error />}/>
     </Routes>
     </BrowserRouter>
```

```
    </div>
  );
}
export default App;
```

Home.js

```
function Home()
{
    return (
        <div>
          <h1>Home</h1>
        </div>
    )
}
export default Home;
```

About.js

```
function About()
{
    return (
        <div>
          <h1>About</h1>
        </div>
    )
}
export default About;
```

Contact.js

```
function Contact()
{
   return (
      <div>
        <h1>Contact</h1>
      </div>
   )
}
export default Contact;
```

Error.js

```
function Error()
{
   return(
      <div>
         <h1>OOPS! 404 Error </h1>
      </div>
   )
}
export default Error;
```

step6:

---

create index.js component to render the output inside "src" folder.

index.js

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

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

    <App/>

  </React.StrictMode>
);
```

step7:

Test the application by using below url's.

ex:

http://localhost:3000/

http://localhost:3000/home

http://localhost:3000/about

http://localhost:3000/contact

http://localhost:3000/gallery

http://localhost:3000/services

Adding Navigation using Link component

==========

A Link component is used to create links which allow to navigate on different URLs and render its content without reloading the webpage.

ex:2

App.js

```
import Home from './Home';
import Contact from './Contact';
import About from './About';
import Error from './Error'

import {Link, Routes,Route,BrowserRouter } from 'react-router-dom'
function App() {
  return (
    <div>
      <BrowserRouter>

      <nav >
        <Link style={{display:"block"}} to="/">Home</Link>
        <Link style={{display:"block"}} to="/about">About Us</Link>
        <Link style={{display:"block"}} to="/contact">Contact US</Link>
      </nav>
      <Routes>
        <Route exact path="/" element={<Home />}/>
        <Route path="/about" element={<About />}/>
        <Route path="/contact" element={<Contact />}/>
        <Route path="*" element={<Error />}/>
      </Routes>
      </BrowserRouter>
    </div>
```

```
  );
}
export default App;
```

Home.js

```
-----------
function Home()
{
   return (
      <div>
        <h1>Home</h1>
      </div>
   )
}
export default Home;
```

About.js

```
------------
function About()
{
   return (
      <div>
        <h1>About</h1>
      </div>
   )
}
export default About;
```

Contact.js

```
--------------
function Contact()
```

```
{
   return (
      <div>
        <h1>Contact</h1>
      </div>
   )
}
export default Contact;
```

Error.js

```
function Error()
{
   return(
      <div>
         <h1>OOPS! 404 Error </h1>
      </div>
   )
}
export default Error;
```

index.js

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

const root = ReactDOM.createRoot(document.getElementById('root'));
```

```
root.render(
  <React.StrictMode>

    <App/>

  </React.StrictMode>
);
```

Images/Assets in ReactJS

==========

We can set images/Asset in ReactJS using two ways.

1)Inside public Folder.

2)Inside src folder.

1)Inside public folder

If we put a file into a public folder,It will not be processed by webpack. Instead it will be copied into the build folder untouched.

To reference assets in the public folder, we need to use a special variable called PUBLIC_URL. Only files inside the public folder will be accessible by %PUBLIC_URL% prefix.

How to use image

---

1)

myapp15

|

|---public

|

|---pic.jpg

index.html

---

2)

myapp

|

|---public

|

|---image

|

|--pic.jpg

index.html

If we want to use Image in Javascript file.

App.js

---

ex:1

---

index.html

-

-

-

```
 <div id="root"></div>
<img src="%PUBLIC_URL%/team1.jpeg" alt="mypic"/>
```

-

-

-

Note:

Mostly of the time we are displaying images in Component only.

ex:

App.js

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

export default class App extends Component {
      render() {
            return (
                  <div>
<img src={process.env.PUBLIC_URL+"team1.jpeg"} alt="mypic"></img>
                  </div>
            )
      }
}
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from "./App";

//render the component to index.html
ReactDOM.render(<App />,document.getElementById("root"));
```

2)Inside src folder

---

we can import a file right in a Javascript module.This tell webpack to include that file in the bundle.

How to use

---

1)

```
myapp
|
|---src
        |
        |---pic.jpg
```

App.js

```
import pic from "./pic.jpg";
<img src={pic} alt="mypic" />
```

This ensures that when the project is built.Webpack wil correctly move the images into the build folder and provide us with correct paths.

ex:

App.js

```
import React, { Component } from 'react'
import pic from "./team1.jpeg";

export default class App extends Component {
	render() {
		return (
			<div>
				<img src={pic} alt="mypic"></img>
			</div>
		)
	}
}
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from "./App";

//render the component to index.html
ReactDOM.render(<App />,document.getElementById("root"));
```

Bootstrap in React

==========

A Single-page applications gaining popularity over the last few years, so many front-end frameworks have introduced such as Angular, Vue, Ember, etc. As a result, jQuery is not a necessary requirement for building web apps.

Currently, React is mostly used JavaScript library for building web applications, and Bootstrap become the most popular CSS framework.

Let see how to use bootstrap in react applications.

Project structure

---

myapp13

|

|----node_modules

|

|----public

|

|---favicon.ico

|---index.html

|---manifest.json

|

|------src

|

|---index.js

|

|---App.js

|

|------package.json

|

|------README.md

step1:

---

create a react project i.e myapp13.

ex:

Reactprojects> npx create-react-app myapp13

step2:

Open the VSC code editor.

ex:

Reactprojects> code .

step3:

Move/Switch to myapp13 project.

ex:

Reactprojects> cd myapp13

step4:

Install Bootstrap package.

ex:

Reactprojects/myapp13> npm install bootstrap

step5:

Run the react application.

ex:

Reactprojecs/myapp13> npm start

step6:

Create a App.js file inside "src" folder.

App.js

```
function App()
{
  return(
     <div className="container mt-5">
        <button className="btn btn-outline-primary">clickMe</button>
     </div>
  )
}
export default App;
```

step7:

Import bootstrap package inside "index.js" file.

index.js

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

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

```
    <App />
  </React.StrictMode>
);

reportWebVitals();
```

step8:

Test the application by using below request url.

ex:

http://localhost:3000

React Forms

==========

Forms are an integral part of any modern web application.

It allows the users to interact with the application as well as gather information from the users.

Forms can perform many tasks that depend on the nature of your business requirements and logic such as authentication of the user, adding user, searching, filtering, booking, ordering, etc.

A form can contain text fields, buttons, checkbox, radio button, etc.

Creating Form

React offers a stateful, reactive approach to build a form.

The component rather than the DOM usually handles the React form.

In React, the form is usually implemented by using controlled components.

Controlled component

In the controlled component, the input form element is handled by the component rather than the DOM. Here, the mutable state is kept in the state property and will be updated only with setState() method.

Controlled components have functions that govern the data passing into them on every onChange event, rather than grabbing the data only once, e.g., when you click a submit button.This data is then saved to state and updated with setState() method. This makes component have better control over the form elements and data.

Project structure

---

myapp14

|

|----node_modules

|

|----public

|

|---favicon.ico

|---index.html

|---manifest.json

|------src

|

|---index.js

|

|---App.js

|

|-------package.json

|

|-------README.md

step1:

------

create a react project i.e myapp14.

ex:

Reactprojects> npx create-react-app myapp14

step2:

Open the VSC code Editor.

ex:

Reactprojects> code .

step3:

Switch/Move to myapp14 project.

ex:

Reactprojects> cd myapp14

step4:

Install bootstrap package.

ex:

Reactprojects/myapp14> npm install bootstrap

step5:

Run the react application.

ex:

Reactprojects/myapp14> npm start

step6:

Import Bootstrap package inside "index.js" file.

index.js

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

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

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

step7:

Create App.js file inside "src" folder.

App.js

```
import {useState} from 'react';
function App()
{
  const [userRegistration,setUserRegistration]=useState({
    username:"",
    password:"",
    date:"",
    category:""
})

const handleClick=(e)=>
{
  const name=e.target.name;
  const value=e.target.value;
  //set to state
  setUserRegistration({... userRegistration,[name]:value});
}

const handleSubmit=(e)=>
{
    e.preventDefault();
    setUserRegistration({username:"",password:"",date:"",category:""});
}
```

```
return(

 <div className="container mt-4">

   <form onSubmit={handleSubmit}>
   <div className="row w-50">
   <h1 className="text-center" ><u>React Form </u></h1>
    <label htmlFor="username" className="my-3">UserName:</label>
    <input type="text" name="username" autocomplete="off"
         className="form-control"
         value={userRegistration.username}
         onChange={handleClick}/>

    <label htmlFor="password" className="my-3">Password:</label>
    <input type="password" name="password" autocomplete="off"
          className="form-control"
          value={userRegistration.password}
          onChange={handleClick}/>

    <label htmlFor="date" className="my-3">Date:</label>
    <input type="date" name="date" autocomplete="off"
          className="form-control"
          value={userRegistration.date}
          onChange={handleClick}/>

    <label htmlFor="category" className="my-3">Category</label>
    <select name="category" className="form-control"
           value={userRegistration.category}
           onChange={handleClick}>
       <option value="">none</option>
```

```
          <option value="entertainment">Entertainment</option>
          <option value="drama">Drama</option>
          <option value="action">Action</option>
        </select>

        <button className="btn btn-primary mt-4 w-100"> submit </button>
        </div>

        </form>
    </div>

  )

}
export default App;
```

step8:

Test the application by using below request url.

ex:

http://localhost:3000

Lists in ReactJs

==========

Lists are used to display data in an ordered format and mainly used to

display menus on websites. In React, Lists can be created in a similar way as we create lists in JavaScript. Let us see how we transform Lists in regular JavaScript.

The map() function is used for traversing the lists.

ex:

Project structure

---

myapp15

|

|----node_modules

|

|-----public

| |

  |---favicon.ico

  |---index.html

  |---manifest.json

|

|-----src

  |

  |---index.js

  |---App.js

|

|-----package.json

|-----README.md

step1:

---

create a react project i.e myapp15.

ex:

Reactprojects> npx create-react-app myapp15

step2:

Open the VSC code editor.

ex:

Reactprojects> code .

step3:

Move/Switch to myapp15 project.

ex:

Reactprojects> cd myapp15

step4:

Run the react application.

ex;

Reactprojects/myapp15> npm start

step5:

Create App.js file inside "src" folder.

App.js

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

export default class App extends Component {

  render() {

      var arr=[10,20,30,40];

      var newArr=arr.map((element)=>
```

```
      {
        return <li>{element}</li>
      })

   return (
    <ul>
       {newArr}
    </ul>
   )
 }
}
```

step6:

Test the application by using below request url.

ex:

http://localhost:3000

ex:2

App.js

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

export default class App extends Component {

  state={
    users:[
     {pid:101,pname:"LG",pprice:10000},
```

```
    {pid:102,pname:"LAVA",pprice:20000},
    {pid:103,pname:"MI",pprice:30000},
    {pid:104,pname:"SAMSUNG",pprice:40000}
  ]
 }

 render() {

       var newArr=this.state.users.map(user=>
       {
         return <h1>Id: {user.pid} Name: {user.pname} Price: {user.pprice}</h1>
       })

   return (
      <div>
       {newArr}
      </div>
   )
 }
}
```

ex:3

App.js

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

export default class App extends Component {

 state={
   users:[
```

```
    {pid:101,pname:"LG",pprice:10000},
    {pid:102,pname:"LAVA",pprice:20000},
    {pid:103,pname:"MI",pprice:30000},
    {pid:104,pname:"SAMSUNG",pprice:40000}
  ]
}

render() {

       var newArr=this.state.users.map(user=>
       {
         return <tr><td>{user.pid}</td> <td> {user.pname}</td> <td>{user.pprice}</td></tr>
       })

  return (
     <div>
       <table border={1} width="100%">
         <thead>
            <tr>
              <th>ID</th>
              <th>NAME</th>
              <th>PRICE</th>
            </tr>
         </thead>
         <tbody>
            {newArr}
         </tbody>
       </table>
     </div>
  )
}
```

}

Key in ReactJS

==========

A key is a special string attribute you need to include when creating

lists of elements.

Keys help react identify which items have changed are added or are removed.

ex:

App.js

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

export default class App extends Component {

 state={
   users:[
     {pid:101,pname:"LG",pprice:10000},
     {pid:102,pname:"LAVA",pprice:20000},
     {pid:103,pname:"MI",pprice:30000},
     {pid:104,pname:"SAMSUNG",pprice:40000}
   ]
 }

 render() {

       var newArr=this.state.users.map(user=>
       {
```

```
          return <tr key={user.pid}><td>{user.pid}</td> <td> {user.pname}</td>
<td>{user.pprice}</td></tr>
        })

   return (

        <table border={1} width="100%">
         <thead>
            <tr>
              <th>ID</th>
              <th>NAME</th>
              <th>PRICE</th>
            </tr>
         </thead>
         <tbody>
            {newArr}
         </tbody>
        </table>

   )
 }
}
```

Axios

=======

Axios is used to make HTTP request (GET,POST,PUT,DELETE).

Using axios we can give the request to Rest API's.

We can install axios by using below command.

ex:

reactprojects> npm install axios

or

reactprojects> yarn add axios

Project structure

---

myapp16

|

|----node_modules

|

|-----public

| |

|---favicon.ico

|---index.html

|---manifest.json

|

|-----src

|

|---index.js

|---App.js

|---FetchApi.js

|

|-----package.json

|-----README.md

step1:

---

create a react project i.e myapp16.

ex:

Reactprojects> npx create-react-app myapp16

step2:

Open the VSC code editor.

ex:

Reactprojects> code .

step3:

Move/Switch to myapp16 project.

ex:

Reactprojects> cd myapp16

step4:

Install axios in myapp16 project.

ex:

Reactprojects/myapp16> npm install axios

step5:

Run the react application.

ex;

Reactprojects/myapp15> npm start

step6:

Create App.js file inside "src" folder.

App.js

```
import FetchApi from "./FetchApi";

function App()
{

  return (
     <FetchApi/>
  )
}
export default App;
```

step7:

Arange one REST API for fetching the data.

ex:

https://jsonplaceholder.typicode.com/users

step8:

Create FetchApi.js file inside "src" folder.

FetchApi.js

```
import {useState} from 'react';
import axios from 'axios';
function FetchApi()
{

   const [data,setData]=useState([])
```

```
const handleClick=()=>
{
  axios.get("https://jsonplaceholder.typicode.com/users")
  .then(response=>
    {
      setData(response.data)
    })
    .catch(error=>
      {
        this.setData(error);
      })
}

return (
  <div>
    <center>
      <button onClick={handleClick}>Fetch API </button>
    </center>
    <table border={1} width="100%">
      <thead>
        <tr>
          <th>ID</th>
          <th>NAME</th>
          <th>USERNAME</th>
          <th>EMAIL</th>
        </tr>
      </thead>
      <tbody>
        {
          data.map(data=>
            {
```

```
            return <tr>
                    <td>{data.id}</td>
                    <td>{data.name}</td>
                    <td>{data.username}</td>
                    <td>{data.email}</td>
                   </tr>
            })
          }
      </tbody>
    </table>
   </div>
 )
}
export default FetchApi;
```

step9:

Test the application by using below request url.

ex:

http://localhost:3000